AUTOMATED ESSAY ANALYSIS

by

Isaac Persing

APPROVED BY SUPERVISORY COMMITTEE:

Vincent Ng, Chair

Nicholas Ruozzi

Latifur Khan

Vibhav Gogate

AUTOMATED ESSAY ANALYSIS

by

ISAAC PERSING, AS, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2017

ACKNOWLEDGMENTS

First I would like to thank my advisor Professor Vincent Ng. It is difficult to capture the full scope of the guidance and support he has given me throughout the last decade. Whenever I asked him for help on some matter, whether it was directly related to my research with him, or just some oddball request like the time I asked him to write a letter on my behalf for an obscure insurance matter, he was always quick to respond. Thanks to his extensive knowledge of the NLP field, his dedication, and his creativity, many of the ideas contained in this document could best be described as my interpretations of his original inspirations.

I would also like to thank Professors Latifur Khan, Vibhav Gogate, and Nicholas Ruozzi for their guidance and for taking time out of their busy schedules to serve on my committee. Next I would like to thank Professor Ng's lab for their help on a variety of matters throughout the years. Members of Professor Ng's lab who I would particularly like to thank are Alan Davis who made many early contributions to the essay analysis project including co-authoring chapter 3 of this dissertation and Kazi Saidul Hasan who answered a lot of questions for me early on. I would like to thank the UT Dallas community in general for fostering an environment that helped me to complete this work.

I am grateful to my parents Edward Garza and Teri Persing for always being there for me throughout my time at UT Dallas and in my life in general, and to my extended family who in turn were there for my parents when I could not be.

Finally I would like to thank the donors who funded the Excellence in Education Doctoral Fellowship and the Jonsson Distinguished Research Fellowship, without whose support I could not have completed this research. I am also grateful to the National Science Foundation, whose grants IIS-0812261, IIS-1147644, IIS-1219142, and IIS-1528037 supported much of this work.

March 2017

AUTOMATED ESSAY ANALYSIS

Isaac Persing, PhD
The University of Texas at Dallas, 2017

Supervising Professor: Vincent Ng, Chair

Automated essay analysis is one of the most important educational applications of natural language processing. The practical value of automated essay analysis systems is that they can perform essay analysis tasks faster, more cheaply, and more consistently than a human can, and can be made available for student use at any time. For example, a human teacher may spend hours grading a class's essay assignments, and due to issues like fatigue may not grade the last essay with as much care as the first. An automated essay analysis system, by contrast, can grade a stack of essays in seconds, and can be guaranteed to grade the last essay as carefully as the first.

Unfortunately, automated essay analysis is complicated by the fact that essay analysis tasks often require a deep understanding of essay text, so designing accurate automated essay analysis systems is not a trivial task. For example, we can't judge how well an essay is organized from just the words it contains. Instead, we must often develop task-specific features in order to help a computer make sense of it.

This dissertation focuses on advancing the state-of-the-art in automated essay analysis. Specifically, we define and present new computational approaches to seven different essay analysis tasks, namely 1) scoring how well an essay is organized, 2) scoring the clarity of its thesis, 3) detecting which errors it makes that hinder its thesis's clarity, 4) scoring how well it adheres to the prompt it was written in response to, 5) scoring its argument's quality, 6)

detecting the stance its author takes on a given topic, and 7) detecting the structure of the argument it makes. For each of these tasks, our approach significantly outperforms competing approaches in an evaluation on student essays annotated for the task. To stimulate future work on automated essay analysis, we make the annotations we produced for these tasks publicly available.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Automated essay analysis is one of the most important educational applications of Natural Language Processing (NLP). Given an essay, an automated essay analysis system's goal is to provide annotations on the essay with respect to some specific task without human intervention. Some tasks an automated essay processor might perform include scoring essays along any of several dimensions of quality (e.g., applying a score between 0 and 6 to describe how coherent the essay is or how well it adheres to its prompt), identifying various types of errors that occur in it (e.g., finding spelling or grammatical errors in its text or logical errors in its argumentation structure), labeling its sentences according to what function they perform in the overall essay (e.g., identifying thesis sentences), or more abstract tasks like constructing a tree from the essay's text to describe how its argument is structured.

1.1 Why Are These Tasks Important?

While it is interesting that these tasks can be performed by a computer without human intervention, of what practical use is an automated essay processing system? Some major reasons for preferring an automated essay processing system over a human annotator include time, cost, consistency, and availability.

The amount of time a human annotator requires to perform one of these essay analysis tasks is heavily dependent on the nature of the task. For example, she may be able to assign the essay a holistic score relatively quickly, as assigning such a score only requires the annotator to read the essay and then to make one scoring decision based on the essay's quality. By contrast, it would take the annotator much longer to perform some task that involves making decisions about each sentence or clause in the essay (as in some argument analysis tasks). Regardless of the complexity of the task, however, the annotator has to, at

a minimum, read an essay in order to provide any annotation on it, and this may take many minutes depending on the length of the essay. An automated essay processing system, by contrast, can usually generate annotations on an essay in a fraction of a second.

Designing an automated essay analysis for performing some task, however, can take years and requires a lot of specialist knowledge. As such skilled labor is expensive, it seems at first glance that cost would be a major reason to prefer human annotation over automated essay analysis. However, once such a system has been created, it can be applied to an unlimited number of essays essentially for free. Human essay annotation is comparatively expensive on a per-essay basis. Thus an organization like the Educational Testing Service which evaluates many thousands of essays written for the GRE and TOEFL tests may find automated essay analysis to be an appealing alternative to human annotation.

Humans can introduce inconsistencies into the annotation process. For example, it is often impractical to arrange for the same human to process all the essays that need to be processed. If some annotators are more lenient than others (and thus prefer to give higher scores on scoring tasks), it is conceivable that a student's likelihood of passing an essay examination depends at least partly on luck, where a luckier student is one whose essay is evaluated by a lenient annotator, or an annotator who is a good mood when she evaluates the essay.

Finally, since computers do not need to sleep, an automated essay analysis system can be made available to evaluate and provide feedback on a student's essay at any time. A human teacher, by contrast, may see her students only a few times a week. And even when her class does meet, she may not have time to give each student much individual attention.

1.2 What Makes These Tasks Challenging?

As we will discuss in Chapter 2, a considerable amount of work has been done on a wide variety of automated essay analysis tasks. However, many of the tasks we address in this

dissertation are new to the NLP community. Since they have not been addressed before, there is not an existing framework to help us understand how to think about the problems.

This would not be a problem, however, if it were the case that these tasks could successfully be addressed by posing them to a standard machine learning algorithm using common NLP features. Unfortunately, this is not the case. A deeper understanding of an essay's text is required in order to perform these tasks well. We must therefore form new linguistic insights into each task and then develop ways to incorporate these new insights into automated essay analysis systems in order achieve good performances on these tasks. In the next section we discuss how we accomplish this by introducing new linguistically-motivated features and making adjustments to our models for solving the problems.

1.3 Our Goals and Contributions

Our goal in this dissertation is to design state-of-the-art approaches to a variety of automated essay analysis tasks. For each task, we demonstrate our approach's quality by showing that it significantly outperforms a reasonable baseline approach for the task.

We categorize the tasks we address into ones where the goal is to score essays along some dimension of quality, and ones where the goal is to form an understanding of essay argumentative structure.

1.3.1 Scoring Tasks

The first task we address is automated organization scoring of student essays. The goal in this task is to assign each essay a score on a scale of 1-4 to indicate how well the essay is organized. We describe our approach to this task in Chapter 3.

Our main insight into this task is that it is possible to abstract an essay's organization as a sequence of sentence or paragraph labels. For example, a five paragraph essay might start with an introductory (I) paragraph, followed by three body (B) paragraphs which

offer support for the essay's thesis, and finally a conclusion (C) paragraph which (re)states the essay's thesis and summarizes its main points. For organization scoring, the order in which these paragraphs appears (in this example, IBBBC) is more important than, say, any of the words the essay uses. For this reason, in our approach we develop methods for automatically applying such labels to all of an essay's paragraphs and sentences. We then incorporate various flat and structured features into our approach based on an essay's sequence of sentence or paragraph labels.

In Chapter 4, we develop an approach to the task of automatically scoring the clarity of an essay's thesis on a scale of 1-4.

For this task, we notice that normal essays have a fairly clear thesis, and thus should get a high thesis clarity score. When an essay receives a lower thesis clarity score, it is usually because the essay has committed some error that makes its thesis less clear. For this reason, we constructed a list of five common errors that cause essays to receive low thesis clarity scores. We then developed linguistically-motivated features for detecting each of these errors. Our approach, then, incorporates all of these error detection features into a standard regressor.

To make our system more useful, we additionally train classifiers for detecting the presence of each error, as we believe this could be used to give students more informative feedback about their thesis clarity scores.

All of the essays in the corpora we use for developing our approaches to automated essay analysis tasks are written in response to an essay prompt, which asks the student to express and defend their opinion on some specific topic. In Chapter 5, we develop an approach for the task of scoring how well an essay adheres to this prompt on a scale of 1-4.

For this task, we notice that an essay that adheres to its prompt consistently stays on topic and is free of irrelevant digressions. For this reason, our approach employs a variety of features that measure an essay's similarity to its prompt through the use of the same words, semantically similar words, or topically similar words.

The last essay scoring task we address is argument quality scoring. In Chapter 6, we develop an approach for the task of scoring an essay's argument quality on a scale of 1-4.

For this task, we notice that an essay with a high argument quality score should present a sound argument and should be able to persuade readers of its thesis. To get at soundness, we develop a variety of features that capture whether the argument is correctly structured. For example, some of our features attempt to capture whether the essay has a thesis, or whether it contains enough evidence to support its claims. We get at persuasiveness by developing features that try to understand the semantic content of a student's essay. For example, students who advocate for unpopular views in their essays will have a hard time persuading readers, so we use linguistic features to detect whether a student is expressing an unpopular view, then employ this classification as a feature for argument quality scoring.

1.3.2 Argument Analysis Tasks

The first major non-scoring automated essay analysis task we undertake is stance classification. Recall that all of our essays are written in response to a prompt which expresses a position on some topic. Given an essay and the prompt to which it responds, the goal of a stance classifier is to determine whether the essay agrees or disagrees with the prompt and the intensity of that (dis)agreement. Our approach to this task is described in Chapter 7.

Our main insight into this task is that essays typically explicitly state their position with respect to the prompt's stance in only one or two sentences in the entire essay. The rest of the essay does not provide much additional useful information for stance classification. To exploit this observation, we develop heuristics for detecting where these explicit statements of the author's stance occur. We then develop linguistically-motivated features based on our attempt to extract stance information out of these statements.

The final task we address is argumentation mining in student essays, which involves two challenging subtasks: identifying the components of the essay's argument and identifying

the relations that occur between them. In Chapters 8, 9, and 10 we propose three different approaches to these tasks.

Our main insight into this task is that the relatedness of its subtasks means that we can benefit from solving both problems jointly rather than sequentially (i.e., performing both tasks at once rather than first identifying arguments then identifying relationships between them). In Chapter 8 we perform joint inference on the tasks in order to build an essay's argument tree via integer linear programming (ILP). This gives us an elegant way of enforcing essay-level constraints (e.g., an essay must include one thesis statement), while building trees that maximize the subtasks' expected average f-score.

However, our ILP approach can be thought of as making a set of unrelated local decisions which it then stitches together into a consistent argument tree. It cannot exploit tree-level information (like depth of the proposed argument tree or how many claims it contains) to determine if a proposed argument tree is good. To address this concern, in Chapter 9, we pose argumentation mining as a structure prediction problem, wherein we use a structured perceptron with tree-level features to identify which of a set of trees is most likely to be the essay's true argument tree.

Both of these approaches, however, are supervised, meaning that they require a lot of annotated data in order to work. In Chapter 10, we develop new linguistic insights into the task that allow us to address it in an unsupervised manner. More specifically, we first use our insights to heuristically label some training data. We then augment this data with additional labeled (but unannotated) instances obtained via self-training. We finally train a learner on this data.

1.4 Roadmap

The remainder of this dissertation is organized as follows. In Chapter 2, we discuss previous work on automated essay analysis. Each of the next five chapters describes a particular

essay analysis task, presents our approach to solving the task, and analyzes our approach's performance on the task compared to a reasonable baseline approach. More specifically, the tasks covered in these chapters are organization scoring (Chapter 3), thesis clarity scoring (Chapter 4), prompt adherence scoring (Chapter 5), argument quality scoring (Chapter 6), and stance classification (Chapter 7). The next three chapters perform the same kinds of analyses on the essay argumentation mining task, with each chapter focusing on a different approach to the task. These approaches include integer linear programming (Chapter 8), structured perceptrons (Chapter 9), and unsupervised argumentation mining (Chapter 10). Finally, we conclude in Chapter 11.

CHAPTER 2

RELATED WORK

In this chapter we will discuss the existing work on automated essay analysis.

2.1 Automated Essay Scoring

In this section we will discuss several of the dimensions along which essays can be automatically scored.

2.1.1 Holistic Essay Scoring

By far the most common automated essay scoring task is holistic essay scoring. That is, given an essay, the goal of a holistic automated essay scorer is to evaluate its overall quality. This type of evaluation abstracts away questions about an essay's quality along specific dimensions (e.g., how well is it organized, or how strong is its argument in the case of argumentative essays, how technically correct its grammar is), and instead tries to represent the essay's quality along these dimensions in one score.

While we will find that some of the works described in later sections also develop systems for predicting an essay's holistic score, works in later sections' ultimate goals will be to achieve some narrower task. For example, one such question may address whether it is possible to demonstrate the importance of a specific dimension to holistic essay score by integrating features targeted at that dimension into a state of the art holistic essay score? In this section, however, we will focus solely on works aimed at developing a holistic automated essay scoring system.

In Burstein et al. (1998), the authors develop the holistic essay scoring system e-rater. Given an essay, e-rater's goal is to assign the essay a score in the range of 0-6 describing the essay's overall quality, with a score of 0 being associated with the poorest essays and a score

of 6 being associated with the best essays. To perform this task, the authors train a linear regression model from score-annotated training data using three categories of features.

In order to generate their syntactic features, they first syntactically parse all sentences in their corpus. From these syntactic parses, they extract features such as the frequencies of complement clauses, subordinate clauses, relative clauses, and occurrences of subjunctive modal auxiliary verbs (e.g., "would", "could", "should").

To generate rhetorical structure analysis features, they first run each essay through a program that automatically segments it into its component arguments based on syntactic and paragraph-based distribution of cue words (e.g., "first", "second") and labels its argument units (e.g., which subsegments of an argument mark its beginning, and which subsegments develop the argument). They extract features from these annotations such as the total number of pronouns in subsegments marked as beginning an argument and total number of words in subsegments that develop an argument.

Finally, they generate two topic analysis features, both of which are just predictions of an essay's score based on the vocabulary it uses. To build the first of these features, they first construct seven vocabulary vectors from the training set, one for each 0-6 score. Each element in a vector indicates the frequency with which a particular word occurs in essays having the associated score. They create another of these frequency vectors from each test essay, and assign the essay a score by computing its cosine similarity with each of the seven training set vocabulary vectors. The test essay's predicted score, then, is the score associated with the training vector that was found to be most similar. The rationale for this is that high scoring essays should use precise vocabulary to discuss a topic, while low scoring essays will use more general vocabulary. So essays with similar scores should have similar vocabulary vectors.

The second topic analysis feature is just another essay score prediction generated using broadly the same method. Particularly, they still create 7 vocabulary vectors from the

training essay, one for each score. Each element in the vector still represents one word in the vocabulary, but the the value of the element is calculated from a formula involving the word's frequency in essays having the same score, its frequency in all training set essays, and the frequency of the most frequent word in essays with the associated score. For each test essay, one of these vectors is generated for each argument (as automatically delineated in the rhetorical structure analysis description). Each argument, then, is scored by taking its vector's cosine similarity with the seven training vectors and selecting the score associated with the most similar vector. Finally, they take an adjusted mean of an essay's argument scores as the second topic analysis feature.

For evaluation the authors use a corpus of about 9570 essays written by college students for the GMAT and the TWE tests in response to 15 different prompts. Each of the essays is assigned a holistic score by two different human annotators. For each prompt, the authors train their regressor on 270 essays and evaluate its performance on the remaining essays responding to the same prompt. One of their major findings are that e-rater's performance is only slightly worse than a human annotator's, as it disagreed with human annotations only slightly more often than the humans disagreed with each others' annotations. They also notice that this performance is consistent regardless of whether or not the essay is written by a native English speaker. Further analysis of e-rater's performance with regards to essays written by non-native English speakers can be found in Burstein and Chodorow (1999).

One difficulty automated essay scoring systems encounter is that they attempt to assign an essay a score based on the entirety of its text. However, it may be the case that only a subset of essay's text actually represents salient concepts, and features derived from the rest of it may merely confuse the learner. To address this problem, Burstein and Marcu (2000) experiment with summarizing essays before attempting to score them.

In particular, they apply two methods of summarization to both training essays and test essays. The positional summarizer summarizes text based on its position in the essay.

Particularly, in order to produce a k% summarization of an essay, it extracts only the first k% of words appearing in the essay to represent it. The discourse-based summarizer is considerably more sophisticated. It first automatically discourse parses an essay into a binary rhetorical structure tree, where leaf node is associated with a clause in the text, each internal node is associated with a contiguous span of text that subsumes its children, and each edge describes a relationship between text spans (e.g., one text span may provide evidence backing up a claim made in another, or two text spans may contain contrasting information). In the process of constructing a tree, each node is also assigned a status of nucleus or satellite, where nucleus nodes represent text that is more important to an argument, and satellite nodes represent text that is subsidiary to a nucleus's text. Each leaf is assigned an importance value depending on its proximity to the tree's root, its status, and the structure of the tree. To construct a k% summarization, then, the summarizer chooses the most important clauses until k% of words in the essay have been included.

For evaluation, the authors use 15,400 holistically scored essays written by college students for the GMAT test. For each prompt represented in this data, they run many experiments using 270 essays for training and 500 essays for testing. The experiments involve 20%, 40%, 60% and 100% (full text) positional and discourse-based summarizations of both the training and test essays, and they compare each set of predictions derived using summarization to the predictions made using the full texts. Since this paper's focus is on summarization, they just use the second topic analysis feature we described from e-rater which, recall, is a score prediction in its own right and was found to be the most predictive feature in the previous work.

The authors find that 40% and 60% discourse-based summaries always perform significantly better than full texts, regardless of whether only the test essays are summarized, or both the training and test essays are summarized. All 20% and positional summaries, however, always perform worse than full texts.

As we come to the end of this section, we notice that, while we have discussed several works and their approaches on automated holistic essay scoring, we have not addressed why holistic essay scoring is useful. This is important because other finer-grained automated essay analysis tasks are much more straightforwardly useful. For example, an automated essay analysis system designed for detecting thesis statements can inform a student if her essay has no thesis statement, and this would help the student improve her writing because she would know specifically what needs to be changed to correct her essay. A holistic essay scorer cannot provide the student such feedback because a holistic score does not make clear to the student what aspect of her writing a poor score can be attributed to.

While Powers et al. (2000) doesn't describe any new methods for holistic automated essay scoring, it does provide a rationale for automated holistic scoring. More specifically, when a student is asked to write an essay for evaluation, it is not the quality of the essay itself we are interested in. Instead, we do expect this essay's quality to be indicative of the quality of work the student is able to produce. Potential graduate students are required to write essays for the Graduate Record Examination (GRE), for example, because the graduate schools into which these students want to be admitted believe that a student who writes a high quality essay for this test can do well in graduate school, while a student who writes a poor essay may not be ready for graduate work.

The authors of this work attempt to determine whether this is a reasonable assumption by calculating the correlation between human and automatically assigned scores for essays and other student quality metrics unrelated to the essays. For example, they measure whether students who reported having significant writing accomplishments in the past (e.g., getting works published) are also more likely to get high scores on the essays they write for the GRE. They also measure whether students who do well on the GRE essays have high GPAs in writing courses. The authors reason that if GRE essay scores are correlated strongly enough with other measures of achievement, then it is reasonable to use GRE essay scores as a proxy for expected student performance.

In an evaluation of 1700 students who completed the GRE, the authors find that holistic GRE essay score is indeed moderately correlated with some of these other measures of student success, with the degree of correlation varying depending on whether human-annotated or automatically assigned holistic essay scores are used and which student success metric is being compared. In general, human-assigned GRE essay scores are more strongly correlated with student success than are scores automatically assigned by e-rater, the automated essay scoring system used by the Educational Testing Service (ETS).

However, ETS requires that all GRE essays be evaluated by two human annotators in order to ensure consistency of scoring, so the authors also measure the correlation between the various metrics of student success and the average of the two human-assigned scores. The authors find that this considerably improves the correlation between human-annotated score and student success, justifying ETS's decision to require two human annotators per essay. However, the authors find that if they replace one of the two human-annotated scores with a holistic score automatically assigned by e-rater before taking the average, the correlation between all performance metrics and and average score drops only slightly compared to when two human-assigned scores are used. These findings suggest that ETS's essay scoring costs can be almost halved by simply replacing one human annotator on each essay with e-rater with very little drop in performance, thus justifying the use of automated essay scoring systems like e-rater.

2.1.2 Prompt Adherence

Prompt adherence is the dimension of essay quality dealing with whether an essay appears to be on topic/how on-topic the essay is. That is, the type of essay we are typically concerned with in automated essay scoring (the type that is written by a student for some assignment or test) is written in response to a particular prompt. The essay writer's goal is to write an essay on the topic described by this prompt. If the essay for some reason does not accomplish

this, we say that the essay does not adhere to its prompt. This problem is typically treated as a binary classification problem, where an essay is considered either on or off topic. However, it is also possible to rate essays on a numerical scale based on how well their authors adhere to the prompt or to determine which sentences in an essay address the prompt, and which are off topic.

Higgins et al. (2006) is one of the works that treats prompt adherence scoring as a binary classification task. The authors identify five common categories of off-topic essays, empty essays (the writer didn't write anything), unexpected topic (the writer wrote an essay on a different topic than the one the prompt describes), banging-on-the-keyboard (the essay consists of gibberish as might result from banging on a keyboard), copied prompt (the entirety of the essay is a possibly slightly modified copy of the prompt), and irrelevant musings (the writer knowingly wrote something unrelated to the prompt). This work is concerned only with identifying instances of unexpected topic, copied prompt, and irrelevant musings, the latter two of which are collectively called "bad-faith".

The authors describe several methods for detecting these types off-topic essays. One method they use is to represent each essay and each prompt as a vector using content vector analysis. If the cosine similarity between a given essay and all other essays having the same prompt is beneath some threshold, it is classified as off topic. Similarly, if the essay's cosine similarity with its prompt is beneath some threshold, the essay is classified as off topic.

Another method involves assigning a weight to each word in the training set such that the word's weight is high if it occurs frequently in essays responding to the essay under consideration's prompt and infrequent in essays for other prompts. The word's weight should be low otherwise. They then average the weights of all words in the essay under consideration. If this average is beneath some threshold, the essay is considered off topic.

The authors also present some off-topic category specific methods for identifying off topic essays. Their method for identifying unexpected topic essays again represents the essay under

consideration and all prompts in the training set and the essay under consideration's prompt as vectors using content vector analysis. Then, the cosine similarity between the essay and all these prompts is calculated and the prompts are ranked according to how similar they are to the essay. If the correct prompt is not one of the top few prompts in this list, the essay is considered off-topic.

Finally, the authors presented a method for identifying essays written in bad-faith. They represent each essay as a vector that includes features like the count of words in the essay that aren't in the prompt and frequency of direct address markers (e.g., "hello"). They then train an SVM to recognize bad-faith essays using these features.

Since the tasks are slightly different and the available essays are not all annotated exactly the same way, the methods are not all evaluated on the same dataset. But taken together, all the datasets used for evaluation include 11138 different essays written on no fewer than 36 prompts by students aged from 6th grade to college. The authors found via cross-validation experiments that, while the first two methods are better at identifying whether an essay is off-topic, the latter two methods might be preferred because they require no training data written on the same topic as test essays. This would allow teachers give students new essay prompts while still allowing them to evaluate their students' essays automatically.

Louis and Higgins (2010) also views prompt adherence scoring as a binary classification, where an essay is either on topic or off topic. As a baseline against which to compare their systems for this task, they use Higgins et al. (2006)'s system for identifying unexpected topic essays described earlier. That is, they represent a test essay and several prompts (including the prompt the essay was written in response to) as vectors. They then take the cosine similarities between the essay and each prompt, and if the most similar prompt is the prompt the essay was written in response to, the essay is considered on topic. Otherwise it is considered off topic.

A problem with this approach is that essay writers might not use exactly the same words that appear in the prompt when discussing the essay's topic. So for example, given a

prompt about "friends", an author might instead discuss "buddies". This use of a synonym for "friend" would result in a lower cosine similarity between the essay and its prompt than is desired since the essay is conceptually on topic. Motivated by problems like this, the authors develop several methods for adding prompt-related words to prompts before encoding them as vectors. So when a modified prompt's cosine similarity with a conceptually similar essay is calculated, the similarity score should still be high even if the essay does not use the same vocabulary as the prompt.

The methods the authors use to expand the essay prompts are 1) to add inflected forms of each word in the prompt to the prompt (e.g., for the prompt word "friend", they would add words like "friendly" and "friendliness"), 2) to add synonyms of each prompt word to the prompt, 3) to add words that appear in similar contexts as the prompt words to the prompt (e.g., for the prompt word "friendly", they may add "polite" or "cheerful"), 4) to add words associated with each prompt word to the prompt, where "associated" words are found via psychological word free association experiments conducted by Nelson et al. (2004). The authors additionally experiment with automatically correcting the spelling of essays before calculating cosine similarities in order to ensure that spelling errors do not cause their system to believe an essay is off topic.

In an evaluation on four different essay corpora written by college-age students including both native English speakers and learners of English as a second language, the authors find that their methods in some cases greatly reduce the number of essays wrongly classified as off topic compared to the baseline system.

2.1.3 Coherence

Coherence is the essay dimension that describes how well an essay's meaning can be constructed. That is, a coherent essay is one whose meaning can easily be understood by a reader, while an essay whose meaning cannot be understood is considered incoherent.

While coherence contributes to an essay's overall quality, it is only one of the factors that contributes to it. To illustrate this point, note that persuasive essays with insufficient evidence to back up their points and essays that do not adhere to their prompts may receive low holistic scores while remaining highly coherent. Indeed, Burstein et al. (2013) measure the correlation between annotations of essay holistic scores and coherence scores on a corpus of 1,555 student essays. The measured Pearson correlation coefficients are between 0.46 and 0.58, suggesting that there is a medium to strong correlation between essay coherence and overall quality, but that overall quality still captures aspects of essay writing unrelated to coherence.

Burstein et al. (2013) describe a scheme for annotating essays with respect to the coherence dimension on a three point scale. A score of 1 on this scale means that an essay is incoherent so that no meaning can be constructed from it. A score of 2 means that an essay is mostly coherent so its meaning can still be constructed, but one or two points in the text remain confusing to the reader. Finally, a score of 3 indicates a highly coherent essay whose meaning can easily be constructed. However, because fine-grained analysis of essay scoring tasks is often difficult, there was not sufficient inter-annotator agreement on this task. So after annotation, the authors treated automated essay coherence scoring as a binary classification task, where essays annotated with a score of 1 are considered incoherent, and essays with a score of 2 or 3 are considered coherent.

The authors also construct a system for automatically predicting whether a given essay is coherent or not. Their decision tree classifier employs grammatical error features from e-rater (Attali and Burstein, 2006), entity-grid transition probabilities (Barzilay and Lapata, 2008), rhetorical structure theory features (Mann and Thompson, 1988), latent semantic analysis features, and entity type/token ratios recovered from the aforementioned entity-grid. They employ cross validation to evaluate their system on the aforementioned 1,555 essay corpus, which includes summarization, expository, and persuasive essays written by native

and non-native English speakers ranging in age from sixth grade to the college graduate level. Experimental results show that their system outperforms baselines which employ only the features derived from e-rater and which employ the e-rater features and grammar error features.

In Burstein et al. (2010) the authors describe another system for binary coherence classification, where each essay is classified as having either low coherence or high coherence. The primary contribution of this work is its proposed feature set, which consists of features developed for other tasks and settings. More specifically, the authors train a C5.0 decision tree classifier using features from e-rater, grammar, usage, and mechanics error features, token type ratio features, and entity transition features described in (Barzilay and Lapata, 2008).

The authors evaluate their system's performance using n-fold cross-validation on 800 essays submitted for various tests (e.g., TOEFL, GRE). The essays are written by native and non-native speakers of english ranging in age from middle school to adult. By comparing performances of systems trained using different subsets of the aforementioned features, the authors find different subsets of the features are most useful for different types of essays (e.g., the feature subset most useful for predicting coherence on the GRE essays is different than the subset most useful for predicting coherence on TOEFL essays). But regardless of which essay subset is used for evaluation, the best system always involves some version of entity transition ratios.

In order to measure the degree to which topic shifts impact coherence, Miltsakaki and Kukich (2004) use centering theory to generate features on which a coherence prediction system could be based. Broadly, the idea behind this is that each of an essay's paragraphs should be topically uniform. If an essay's intra paragraph topics shift in a particular way, this is a sign that the essay is incoherent.

More specifically, the authors represent each essay as a sequence of paragraphs, each of which in turn is viewed as a sequence of utterances, where utterances are defined as a main

clause paired with all its associated dependent clauses. Each utterance in turn mentions a number of entities, each of which is assigned a salience rank according to what role it plays in the utterance. For example, the subject of a main clause is ranked higher than the main clause's object, which in turn is ranked higher than any of the subjects in the utterance's dependent clauses. If we define Cb(i) as the highest ranked entity from utterance i-1 that also appears in utterance i, then the authors define a *r*ough-shift as follows: if Cb(i) $\neq$ Cb(i-1), and the highest ranked entity in utterance i $\neq$ Cb(i), this constitutes a rough-shift.

To construct an automatic essay scorer for the coherence dimension, the authors argue that it would be useful to incorporate a count of rough shifts that occur within each essay as a feature. While they do not build such an essay scorer, they infer the rough-shift count's usefulness performing some statistical analyses on a set of 100 essays written for the Graduate Management Admission Test. For each essay, they know a human assigned annotation of the essay's holistic score as well as a prediction assigned by the e-rater essay scoring system. They find there is a correlation between the count of rough-shifts and an essay's human-annotated holistic score, and furthermore, they use the technique of jackknifing to determine that e-rater would make better holistic score predictions if it incorporated rough-shift counts. The authors rely on holistic essay scores for their analysis rather than coherence scores only because coherence scores were not available to them for this corpus.

A lexical chain in a text is a sequence of (usually non-contiguous) related words and phrases. Given an essay, Somasundaran et al. (2014) constructed lexical chains consisting of noun, noun-noun, and adjective-noun phrases in the following way. If two words/phrases anywhere in the essay are exactly the same (exact string match), they are considered to have an extra-strong relationship, and must belong to the same chain. If two words/phrases are synonyms for each other, they are considered to have a strong relationship, and must belong to the same chain only if they occur within six sentences of each other. Finally, if two words/phrases are fairly semantically similar but are not synonyms, they are considered

to have a medium-strong relationship, and must only belong to the same chain if they occur within three sentences of each other. The result of this process is a clustering of the words and phrases in an essay, where at one extreme, some clusters may contain only a single word or phrase which was not related strongly enough to any nearby words or phrases to be added to a chain. At the other extreme, some clusters may contain many words and phrases and run the entire length of the essay.

The authors believe that these chains may be useful in predicting several aspects of an essay's coherence. Particularly, coherent essays usually focus on one main theme, so a coherent essay is likely to have a chain extending over the essay's entire length whose words and phrases are related to the essay's topic. Coherent essays are also likely to elaborate on some points to aid the reader's understanding of them, resulting in chains containing many words and phrases used to explain the author's idea in more depth or in different ways. Frequent repetition of the same word can harm the discourse quality, so a coherent essay should generate lexical chains containing a variety of similar words that do not have an extra-strong relationship. Finally, a coherent essay must be well-organized, with discourse connectives that describe things such as when the expression of one idea ends and another begins. So the authors believe, for example, that new discourse chains should begin immediately after discourse connectives like "secondly" or "finally" which indicate that a new idea is about to be expressed.

For these reasons, the authors extract a set of 176 features from an essay's lexical chains to exploit the observations above. They then train a gradient boosting regressor to predict essay coherence on a four point scale (with 1 indicating poor coherence and 4 indicating excellent coherence). On a 10-fold cross-validation evaluation of 876 expository, persuasive, descriptive, narrative, contrastive summary, and subject matter essays written by students ranging from elementary through graduate school-aged, the authors find that the addition of their lexical chain features to a state of the art coherence prediction system significantly improves its performance.

Rather than concerning themselves with essay coherence as a whole, Higgins et al. (2004) decompose essay coherence into four separate dimensions, each of which is annotated at the sentence level. They are 1) how related sentences are to the essay's prompt (DIM_P), 2) how related sentences are to the thesis (DIM_T), 3) how related sentences are to other nearby sentences (DIM_S), and how flawed sentences are with respect to some technical errors (DIM_E).

The authors manually partition essays into discourse segments, each of which is assigned a discourse label such as "background" or "thesis" or "support" and consists of one or more sentences. They then assign coherence scores for each of the four dimensions at the sentence level as follows. Sentences that appear in certain discourse element types (e.g., background, main idea) and which are related to the prompt receive high DIM_P scores. Sentences that appear in other certain discourse element types such as background material and conclusion are assigned a high DIM_T score if they are related to the thesis. Sentences are assigned a high DIM_S score if they are related to at least one other sentence appearing in the same discourse segment. Finally, sentences receive a high DIM_E score if they do not contain too many grammatical, word usage, or mechanics errors. These are (nearly) binary classification tasks, so any sentence not receiving a high score for one of the dimensions receives a low score for the dimension. Exceptions occur only when a sentence does not appear in a discourse element of the appropriate type in order for the dimension to be relevant (e.g., sentences in thesis segments do not get assigned DIM_T scores).

The authors use two different methods to predict sentence scores for these dimensions. DIM_P and DIM_T are predicted using support vector machines. The feature representation of each sentence varies depending on which dimension is being learned, though most features involve the use of random indexing (RI) (Sahlgren, 2005), which is technique for representing words in a low dimensional space such that semantically similar words appear close together in the space. For example, one of these features measures the cosine similarity between

a sentence's RI vector and the essay's prompt's RI vector, where a sentence's vector is calculated by simply summing the vectors of its component words. The approach the authors use for predicting DIM_E is a bit more heuristic in style, as they simply feed the essay into the Criterion online writing service (Burstein et al., 2004) and assign sentences a high score if Criterion reported few of the technical errors mentioned above. No system was constructed for predicting DIM_S. They evaluated the performances of these systems using 10-fold cross-validation on 989 annotated persuasive or expository essays written by 12th graders and college freshmen.

The most important aspect of this work, perhaps, is that it clearly illustrates that many of the dimensions of essay quality we deal with in automated essay scoring are not, in fact, orthogonal. For example, DIM_P shows that the authors think of a sentence's relatedness to its essay's prompt an aspect of coherence. However, the description of this dimension sounds very much like prompt adherence, which we described in Section 2.1.2.

Higgins and Burstein (2007) follow up on this work as it applies to DIM_P. More specifically, recall that Higgins et al. (2004) predicted DIM_P on a sentence by extracting several features from it based on RI, then feeding those features into a naive bayes classifier. Higgins and Burstein (2007) use only the sentence's RI cosine similarity to the essay's prompt in order to predict whether the sentence should get a high or low DIM_P score. If the similarity is above a given threshold, they classify the sentence as having a high DIM_P score, and classify it as having a low DIM_P score otherwise.

The contribution of this work, however, is not that it constructs the best DIM_P predictor. Instead, its main purpose is to explain how overcome a particular theoretical difficulty that arises from the straightforward application of RI similarity to this problem described earlier. Particularly, the authors explain that it is inappropriate to obtain a vector representation for a sentence by simply summing the RI vectors of its words because this tends to cause longer sentences to have higher cosine similarities. They instead calculate the mean vector of all

words (with each word's contribution weighted by its occurrence frequency), then subtract this mean vector from the vector for each word. A sentence's vector, then, is the sum of the mean-subtracted vectors of its words. This technique, they note, has the added benefit of making stopword removal unnecessary since stop words, being frequent, tend to be near the mean vector.

On an analysis of 2330 sentences from 292 essays from the previously mentioned essay corpus, the authors found that representing sentences in this way yielded better accuracy (70.1%) for DIM_P scoring than representing sentences as merely the sum of their words' RI vectors (67.1%). Additionally, they found that by combining this similarity measure between a sentence and its essay's prompt with their LSA similarity yielded yet more improvement, bringing prediction accuracy up to 72.1%.

2.2 Argumentation Mining

Argumentation mining in student essays deals with the task of automatically discovering the argumentative structure of persuasive or argumentative essays. For example, some works such as Faulkner (2014) predict which side of an issue an argumentative essay is arguing for, while other works like Falakmasir et al. (2014) and Burstein and Marcu (2003b) attempt to identify important components that make up an argument such as an essay's thesis or its conclusion, and still others like Stab and Gurevych (2014b) additionally identify the claims and premises within an essay that make up its argument and predict how these components are related to each other (e.g., which premise supports which claim).

Burstein and Marcu (2003b) describes two methods for identifying which sentences in an essay best capture its thesis and conclusion. Though the authors do not describe what they do as argumentation mining, these are important concepts in argumentation mining because an essay's thesis sentence explains its stance with respect to the essay prompt (the position it is arguing for), and the conclusion summarizes the argument the writer has made.

In order to perform this task, the authors first annotate a collection of student essays to indicate which sentences describe the essays' theses, and which sentences describe the essays' conclusions. They treat this as a classification task, where each sentence is either a thesis, a conclusion, or neither, and they allow multiple sentences in a single essay to take any one of these labels (since, for example, a thesis can be expressed in multiple sentences).

The authors train a C5.0 classifier for identifying thesis and conclusion sentences using four types of features. The first feature type indicates a sentence's position within its paragraph and the essay in several ways. For example, one of these features indicates whether the sentence occurs in the last paragraph. The second feature type is based on rhetorical structure theory. The authors automatically discourse parse (Marcu, 2000) the essays in order to extract discourse relationships from which they construct these rhetorical structure theory features. For example, one feature indicates whether the sentence forms the nucleus of a relationship (i.e., whether it is the most essential of the sentences it is related to for understanding the writer's intention). The third feature type indicates the presence of expressions that link discourse spans and signal semantic relationships. These cue phrase features include expressions like "second", which indicates that the sentence describes the second point in a list. Finally, the authors construct features that capture language commonly used in essays as observed in Burstein et al. (2001) Burstein and Marcu (2003a). Some examples of this language are words like "should" and "might". Some of these essay words like "opinion" and "feel" are likely to appear in thesis sentences, while other essay words like "in conclusion" are associated with conclusions.

In an evaluation on 1200 persuasive and informative essays written by 12th graders and college freshmen in response to six different prompts, the authors find that their approach outperforms a heuristic baseline which classifies sentences as theses or conclusions based only on their position in the essay and the length of the essay. This observation holds true regardless of which of two settings is used for evaluation. In the first setting, essays from all

prompts appear in the training and the test set. In the second setting, the essays from one prompt make up the entirety of the test set, and so there are no essays written in response to that prompt in the training set. Further, the authors find that, despite the second setting being the more difficult of the two (because the training set less closely resembles the test set), their system in the second setting still performs competitively compared to its performance in the first setting. This is important because the second setting simulates the real life situation where a teacher has asked students to respond to some prompt for which there is no thesis or conclusion annotated data available.

Falakmasir et al. (2014) follows up on the previous work, also predicting which sentences in an essay are its theses or conclusions. Like the aforementioned work, the authors ask annotators to label each sentence in an essay as a thesis, a conclusion, or other, allowing multiple sentences in each essay to take any of these labels. Unlike the previous work, however, annotators are also asked to rate each thesis or conclusion sentence on a three point scale, with a score of 1 indicating a vague or incomplete thesis or conclusion, a score of 2 indicating a simple but acceptable thesis or conclusion, and a score of 3 indicating a sophisticated thesis or conclusion.

The authors use this labeled data to simultaneously perform 3 prediction tasks. Particularly, they predict whether each sentence is a thesis, whether each sentence is a conclusion, and whether each essay contains at least one thesis or conclusion. The first two tasks are treated as one trinary prediction task, where one of the three labels "thesis" "conclusion", or "other" must be applied to each sentence. For the latter prediction task, they simply aggregate the sentence-level thesis and conclusion predictions into an essay level prediction (i.e., if they predict that one of the sentences in an essay is a thesis, they also predict that the essay contains a thesis or conclusion). For the purpose of training and prediction, only sentences receiving a score of 2 or 3 in the three point scale are considered true conclusion or thesis statements, with all unannotated sentences and sentences receiving a score of 1 being considered part of the "other" class.

The authors train three types of classifiers, support vector machines, naive bayes, and decision trees to detect thesis and conclusion statements in student essays using 19 features, with many of the features being inspired by the previously discussed work. Particularly, they use positional features indicating paragraph number, sentence number within the paragraph, and and whether the paragraph is the first, last, or a body paragraph. They also use features such the counts of prepositional and gerund phrases, counts of adjectives and adverbs, counts of phrases like "because" and "due to", the number of words overlapping with the prompt, and a rhetorical structure theory feature indicating the sentence's importance within the essay as described in Marcu (1999).

The authors evaluate their systems on a corpus consisting of 432 essays written by high school students on cultural literacy and world literature topics. While the performance of each of the learning algorithms varies depending on which task is evaluated on and how the data is split up for testing (i.e., does the test set contain essays written in response to prompts that also appear in the training set or not?), the authors find that each of the learners always outperforms a purely positional heuristic baseline that predicts all sentences in the first paragraph are theses and all sentences in the last paragraph are conclusions.

In the discussion of the previous two works, we mentioned that it is questionable whether they should be included in a section on argumentation mining. We argued that their inclusion was appropriate because theses and conclusions are important parts of argumentative essay. After all, an argumentative essay's thesis expresses its most important claim, and its conclusion summarizes its argument. However, this does not make a very rich ontology for argumentation, as there are many other important components of an argument that are not captured by these two labels.

Addressing this problem, Ong et al. (2014) describe a new ontology for labeling the different components of an argumentative essay. Particularly, they think of an argumentative essay as being structured like a tree, with some sentences representing nodes in the tree with

labels like current study, hypothesis, claim, and citation, and other sentences representing edges in the tree, indicating which nodes in the tree support or oppose each other. So for example, the claim sentence "low traffic conditions increases traffic violations and risk while driving" is supported by the citation sentence "Otmanl et al. (2005) found that young drivers faced a significant decrease in alertness while in low traffic conditions", and the sentence that indicates the support relation is "when one is less alert then he/she willl be likely commit more traffic violations."

In addition to constructing an ontology for argumentative essays, the authors constructed heuristic rules for automatically applying these sentence labels to an essay's sentences. So for example, a sentence beginning with a comparison discourse connective like "however" or "but" gets tagged as opposes, while a sentence containing any substrings like "suggest", "evidence", or "shows" gets tagged as a claim. The authors use eight rules like these to tag all sentences in an essay with an appropriate label from their argumentation ontology. After applying these labels, the authors then heuristically score the essay based on the sentence labels using six rules. Some example rules are that an essay having at least one current study sentence gets one point added to its score, as does an essay containing at least one hypothesis. An essay's final predicted score is the sum of all points it is awarded through these rules.

The authors evaluate their system's performance on a corpus of 52 argumentative essays written by university undergraduate students. Despite this work's concern with its argumentative essay ontology, these essays are not annotated with the sentence labels from the ontology. Rather, each essay is given a holistic score on a scale of 1 to 5, with 1 being lowest and 5 being highest. When annotating essays with this score, annotators were asked to take into account grammar usage, flow, organization, and the logic behind the essay's argument. The authors find that the essay scores predicted by their heuristic method are highly rank-correlated with the annotated essay scores, suggesting that their ontology provides useful

insights into what makes an essay good, though the small dataset size makes it difficult to draw any firm conclusions.

Like Ong et al. (2014), Stab and Gurevych (2014b) also understand arguments as being structured like a tree. However, the ontology they develop in Stab and Gurevych (2014a) is somewhat different. In particular, they describe an argument as a set of related argument components. Each argument component in an essay is associated with a clause-level segment of text and an argument component type. The three argument component types include major claims, which express the author's stance with respect to the topic (this is a close analogue to the theses in previously discussed works), claims, which are controversial statements whose truth should not be accepted without additional support, and premises, which underpin the validity of claims.

Unlike in Ong et al. (2014), the relationships between these argument components are not explicitly represented by segments of text, but they are still permitted to have support or attack relations. Aside from the absence of explicit text representing them, these support and attack relations are direct analogues to the support and oppose relationships in that work.

To give some idea of the argumentative structure of a typical persuasive essay under this ontology, an essay is required to have exactly one major claim (thesis). Each of the essay's paragraphs tends to focus on one claim, which usually supports or attacks the major claim. And finally, each paragraph tends to contain several premises which support or attack the paragraph's claim. With the exception of the one major claim requirement, these are just tendencies. For example, an essay may contain two claims, or it may contain 0 claims, and premises may also support or attack major claims or other premises.

The authors use the 90 persuasive essays collected from the Essay Forum website (`http://www.essayforum.com`) and annotated in Stab and Gurevych (2014a) to perform two tasks, argument component identification and relation identification.

In the argument component identification task, the authors are given an essay along with a set consisting of all the essay's argument component boundaries and sentences that contain no argument components. The goal in this task is to label each of these text segments with one of four labels, major claim, claim, premise, or none (which is the appropriate label for the sentences containing no argument components).

The authors train four types of learners, including support vector machines, naive bayes, C4.5 decision trees, and random forests, to perform this four class classification task using five categories of features. Structural features include information such as the lengths of the argument component and its containing sentence, paragraph position, and counts of punctuations in the argument component. Lexical features include information about the n-grams in the argument component and the presence of modal verbs (e.g., "should" or "could"). Syntactic features include information about the tense of the argument component's main verb and production rules extracted from automatically generated parse trees. Indicator features include discourse markers that appear at the beginning of some argument components (e.g., "therefore", "thus", or "consequently") and words referring to the author such as "I" or "myself". Finally, contextual features contain information about the sentences surrounding the argument component like their lengths and their counts of clauses.

In a 10-fold cross validation evaluation on the 90 annotated essays, the authors find that all of the classifiers outperform a heuristic baseline which just predicts the most frequent class for all argument components. Furthermore, they find that the support vector machine learner performs best out of all their learning systems. This finding somewhat weak since the setting is unrealistic (the given argument component boundaries are only available on annotated essays) and the baseline is simple, though the ontology is intuitively appealing as a description of argument structure, and there is always the possibility that later works will address these problems.

The authors treat the relation identification task as a binary classification task between pairs of argument components. Particularly, they train the same four types of learning

algorithms to predict whether a pair of argument components contains a support relation or not (attack relations, being much less frequent in the corpus, are ignored). The features they use are mostly the same as the features they used for the argument component identification task, though modified to take into account the fact that pairs of argument components are being classified rather than single argument components. Further, they add one new type of feature, which includes information about the type of the argument components as predicted by an argument component identification system.

In another 10-fold cross validation evaluation on the same 90 essays, the authors again find that their learning-based systems all outperform a heuristic relationship identification system. In this case, the heuristic baseline classifies all argument component pairs as non-support. They also once again find that the support vector machine classifier performs best out of all the learners, though the same criticisms apply to these results as applied to the argument component identification results.

To end with a more general note, though 90 essays seems like a small corpus compared to the sizes of most of the essay corpora we have discussed thus far, it is important to keep in mind that the size of training corpus needed is heavily dependent on the task to be performed. For example, holistic essay scoring tasks can be expected to require a large number of score-annotated essays since each essay can be used to produce only one training instance. For more fine-grained tasks like Ong et al. (2014)'s sentence labeling task, each annotated essay yields many training examples since an annotated essay contains many labeled sentences.

Song et al. (2014) understand arguments as conforming to different, pre-defined argumentation schemes (i.e., reasoning patterns) (Walton, 1996), which can be useful for predicting essay quality. As an example of one of these argumentation schemes, consider arguments about policy proposals. An argument that adheres to the policy argumentation scheme will likely address several pre-defined categories of critical questions about the policy. For example, in the problem category, the essay may discuss whether the problem the policy addresses

is really a problem or whether the problem is well defined. In the goal category, it may discuss whether the goal the policy aims to achieve is desirable. Or in the plan implementation category, it may discuss whether it is practically possible to carry out the proposed policy.

To address the question of whether information about an essay's argumentation scheme can be useful for understanding essay quality, the authors first select 600 holistically scored (on a scale of 0 to 6) essays written in response to two prompts, and determine which of the pre-defined argumentation schemes the essays responding to each prompt should fit into (e.g., all of the essays responding to a prompt about a policy question should fit the policy argumentation scheme). They then annotate each sentence of the essays according to which pre-defined category of critical question it addresses.

They then use multiple regression analysis to predict the holistic essay scores under three settings. In the first (baseline) setting, they use only features from a state of the art holistic essay scoring system (Burstein et al., 2013). In the second setting, they combine the baseline features with features extracted directly from the argumentation scheme annotations. Particularly, they construct a feature for each category of critical question encoding the frequency of sentences belonging to that category in the essay. In the third setting, they combine the baseline features with one additional feature that encodes the frequency of sentences that respond to any critical question. The rationale for this is that essays containing a lot of sentences that do not respond to any of the critical questions are probably dwelling excessively on irrelevant matters, and should therefore get lower scores.

In an evaluation on 600 of their essays, using 520 essays for training and 80 for testing, they find that the third system performs best. However, they cannot draw very strong conclusions from this result because this additional feature cannot be automatically generated, as it requires that all training and test essays include sentence level argumentation scheme annotations. The conclusion they do draw from the result, however, is that information about an essay's argumentation scheme can potentially be useful for essay scoring.

To address this weakness, the authors also train L2-penalized logistic regression models to determine whether or not individual sentences address any critical questions. They train these models using standard NLP features such as word n-grams from each sentence and its surroundings, sentence length, sentence position, presence or absence of POS tags, and some similarity measures between the prompt and the sentence. The resulting classifier yields predictions that are fairly well-correlated with the annotations ($.217 \leq Cohen's Kappa \leq .498$, depending on which prompt(s) were trained/tested on), though they leave much room for improvement. Ideally, if the quality of this classifier can be improved enough, it can be used to automatically label sentences with enough confidence that the predictions can be used to automatically generate the holistic essay scoring feature from the third setting.

Faulkner (2014) deal with the problem of predicting an argumentative essay's stance. That is, given an essay and its prompt (which in their case is an expression of support for some proposition), the goal is to determine whether the essay's writer agrees or disagrees with the prompt.

While this task itself is unlike the previous tasks discussed in this subsection, the author also breaks with previously discussed works on argumentative essays in his understanding of how an argument is expressed. Previous works have focused on identifying particular pieces of an argument like introductions or conclusions or premises or on building tree structures to represent the argument. Faulkner, however, mostly approaches the task by constructing features out of many segments of text within the essay that appear to express a stance, thus ignoring the concept that perhaps only one or two sentences express the essay's stance (the major claim or thesis), while the remaining sentences build an argument around these sentence(s).

Faulkner begins by collecting 1319 argumentative essays written by college-aged non-native English speakers. He annotates each of these essays with one of three stance labels, for, which indicates that the writer agrees with the prompt, against, which indicates that the

writer disagrees with the prompt, or neither, which indicates that the writer neither agrees nor disagrees with the prompt.

He then attempts to identify stancetaking text within the essay in the following way. First he collects a lexicon of words that indicate a writer is taking a stance on something from Hyland (2005) and the Multi-Perspective Question Answering (MPQA) corpus ((Wiebe et al., 2005)). For each stancetaking word in the essay, he identifies the nearest opinion-bearing word from the MPQA subjectivity lexicon. The idea here is that writers express their stances by taking a stance on a proposition, where an opinion-bearing word can serve as a proxy for the proposition. As an example, "I can only say that this statement is completely true" expresses the writer's agreement with the prompt, and it contains a stancetaking word "can" and an opinion-bearing word "true".

Because there are too many ways to express this sentiment, one feature that encodes the presence of this string in an essay would not be useful since it is only ever likely to occur once. For this reason, Faulkner generalizes the expression in two ways. First, he dependency parses the sentence, keeping only the dependency tree nodes on the path between the stancetaking word and the opinion-bearing word. Then he replaces the nodes between the two words with their part of speech tags and prepends the word "not" in case of negation. So from the example above, he would extract the feature "can-V-true" ("say" is the only word appearing between "can" and "true" in the dependency path, and it is a verb) as a feature.

Faulkner also extracts as features any content words appearing in close proximity to stance words if they are sufficiently similar (above some threshold) to one of the content words from the prompt, as measured by Wikipedia link-based measure ((Witten and Milne, 2008)). We would expect these features to be effective at predicting stance because the words are closely related to the prompt, and different prompts provoke different amounts of disagreement. For instance, the prompt "feminists have done more harm to the cause of women than good" elicits much more disagreement than normal in the ICLE corpus. So, if

essays written in response to this prompt appeared in both the training and test sets, the feature "feminist" would be strongly correlated with the against class. This is unfortunate since intuitively the word "feminist" tells us nothing about the writer's stance given the prompt.

Faulkner trains a multinomial naive bayes classifier and a support vector machine on these features to predict whether an essay's stance is for or against the prompt. In an evaluation on the 1235 essays that are both dependency-parsable and annotated with one of these two labels, the author finds that his systems outperforms a bag of words-based baseline and a more sophisticated baseline based on the work of Somasundaran and Wiebe (2010) for stance prediction in on-line ideological debates. Furthermore, he finds that the SVM outperforms the naive bayes classifier.

2.3 Other Essay Analysis

There exists research on a number of other essay analysis tasks that do not involve applying scores to entire essays or their components and do not involve argument mining. In this section we will present a sample of these tasks.

In Burstein and Wolska (2003), the authors are interested in identifying errors in one aspect of essay style. Particularly, they attempt to identify words whose repetition is distracting enough to interfere with the essay's overall quality.

As the algorithm they present is supervised, the authors first ask two experienced essay graders to identify which words in a set of essays are too repetitive. The authors then lemmatize the essay and represent each word it contains as a 7 dimensional feature vector. Particularly, the features encode 1) the absolute count of occurrences of the word, 2) the fraction of words in the essay having this lemmatized form, 3) the average frequency of the word per paragraph, 4) the frequency with which the word occurs in the paragraph where it occurs most frequently, 5) the length of the word in characters, 6) whether the word is a

pronoun, and 7) the distance in tokens between the word and its previous occurrence. They train a C5.0 learning algorithm to predict word repetitiveness using these features.

In a 10-fold cross-validation evaluation on 296 essays written by students ranging in age from 6th grade to college freshmen, the authors find that this feature set yields a learner that can identify excessive repetition with 97% f-score. This system's prediction f-score is higher than the agreement on this task between the two annotators.

2.4 Automated Essay Analysis Systems

Many of the works discussed thus far describe components of commercially-available automated essay scoring systems such as e-rater (Burstein et al., 2013) or The Intelligent Essay Assessor (Landauer et al., 2003). In total, we know of nine of these systems, each of which automatically provides a different set of essay annotations such as scores along different dimensions or different types of sentence labels. For an overview of these systems, see Shermis (2014) and Dikli (2006).

CHAPTER 3

ORGANIZATION SCORING[1]

3.1 Introduction

Automated essay scoring, the task of employing computer technology to evaluate and score written text, is one of the most important educational applications of natural language processing (NLP) (see Shermis and Burstein (2003) and Shermis et al. (2010) for an overview of the state of the art in this task). Recent years have seen a surge of interest in this and other educational applications in the NLP community, as evidenced by the panel discussion on "Emerging Application Areas in Computational Linguistics" at NAACL 2009, as well as increased participation in the series of workshops on "Innovative Use of NLP for Building Educational Applications". Besides its potential commercial value, automated essay scoring brings about a number of relatively less-studied but arguably rather challenging discourse-level problems that involve the computational modeling of different facets of text structure, such as content, coherence, and organization.

A major weakness of many existing essay scoring engines such as IntelliMetric (Elliot, 2003) and Intelligent Essay Assessor (Landauer et al., 2003) is that they adopt a holistic scoring scheme, which summarizes the quality of an essay with a single score and thus provides very limited feedback to the writer. In particular, it is not clear which dimension of an essay (e.g., coherence, relevance) a score should be attributed to. Recent work addresses this problem by scoring a particular dimension of essay quality such as coherence (Miltsakaki and Kukich, 2004), technical errors, and relevance to prompt (Higgins et al., 2004). Automated systems that provide instructional feedback along multiple dimensions of essay quality such as *Criterion* (Burstein et al., 2004) have also begun to emerge.

[1]This chapter was previously published in Persing et al. (2010).

Nevertheless, there is an essay scoring dimension for which few computational models have been developed — *organization*. Organization refers to the structure of an essay. A high score on organization means that writers introduce a topic, state their position on that topic, support their position, and conclude, often by restating their position (Silva, 1993). A well-organized essay is structured in a way that logically develops an argument. Note that organization is a different facet of text structure than coherence, which is concerned with the transition of ideas at both the global (e.g., paragraph) and local (e.g., sentence) levels. While organization is an important dimension of essay quality, state-of-the-art essay scoring software such as e-rater V.2 (Attali and Burstein, 2006) employs rather simple heuristic-based methods for computing the score of an essay along this particular dimension.

Our goal in this chapter is to develop a computational model for the organization of student essays. While many models of text coherence have been developed in recent years (e.g., Barzilay and Lee (2004), Barzilay and Lapata (2005), Soricut and Marcu (2006), Elsner et al. (2007)), the same is not true for text organization. One reason is the availability of training and test data for coherence modeling. Coherence models are typically evaluated on the sentence ordering task, and hence training and test data can be generated simply by scrambling the order of the sentences in a text. On the other hand, it is not particularly easy to find poorly organized texts for training and evaluating organization models. We believe that student essays are an ideal source of well- and poorly-organized texts. We evaluate our organization model on a data set of 1003 essays annotated with organization scores.

In sum, our contributions in this chapter are two-fold. First, we address a less-studied discourse-level task — predicting the organization score of an essay — by developing a computational model of organization, thus establishing a baseline against which future work on this task can be compared. Second, we annotate a subset of our student essay corpus with organization scores and make this data set publicly available. Since progress in organization modeling is hindered in part by the lack of a publicly annotated corpus, we believe that our data set will be a valuable resource to the NLP community.

3.2 Corpus Information

We use as our corpus the 4.5 million word International Corpus of Learner English (ICLE) (Granger et al., 2009), which consists of more than 6000 essays written by university undergraduates from 16 countries and 16 native languages who are learners of English as a Foreign Language. 91% of the ICLE texts are argumentative. The essays we used vary greatly in length, containing an average of 31.1 sentences in 7.5 paragraphs, averaging 4.1 sentences per paragraph. About one quarter of the essays had five or fewer paragraphs, and another quarter contained nine or more paragraphs. Similarly, about one quarter of essays contained 24 or fewer sentences and the longest quarter contained 36 or more sentences

We selected a subset consisting of 1003 essays from the ICLE to annotate and use for training and testing of our model of essay organization. While *narrative* writing asks students to compose descriptive stories, *argumentative* (also known as *persuasive*) writing requires students to state their opinion on a topic and to validate that opinion with convincing arguments. For this reason, we selected only argumentative essays rather than narrative pieces, because they contain the discourse structures and kind of organization we are interested in modeling.

To ensure representation across native languages of the authors, we selected mostly essays written in response to topics which are well-represented in multiple languages. This avoids many issues that may arise when certain vocabulary is used in response to a particular topic for which essays written by authors from only a few languages are available. Table 3.1 shows some of the twelve topics selected for annotation. Fifteen native languages are represented in the set of essays selected for annotation.

3.3 Corpus Annotation

To develop our essay organization model, human annotators scored 1003 essays using guidelines in an essay annotation rubric. Annotators evaluated the organization of each essay

Table 3.1. Some example writing topics.

Topic
Some people say that in our modern world, dominated by science, technology and industrialisation, there is no longer a place for dreaming and imagination. What is your opinion?
Most university degrees are theoretical and do not prepare students for the real world. They are therefore of very little value.
The prison system is outdated. No civilized society should punish its criminals: it should rehabilitate them.
In his novel *Animal Farm*, George Orwell wrote "All men are equal but some are more equal than others." How true is this today?
In the words of the old song: "Money is the root of all evil."
All armies should consist entirely of professional soldiers: there is no value in a system of military service.
Feminists have done more harm to the cause of women than good.
Television is the opium of the masses in modern society. Discuss.
Crime does not pay.

using a numerical score from 1 to 4 at half-point increments. This contrasts with previous work on essay scoring, where the corpus is annotated with a binary decision (i.e., *good* or *bad*) for a given scoring dimension (e.g., Higgins et al. (2004)). Hence, our annotation scheme not only provides a finer-grained distinction of organization quality (which can be important in practice), but also makes the prediction task more challenging.

The meaning of each integer score was described and discussed in detail. Table 3.2 shows the description of each score for the organization dimension.

Table 3.2. Description of each organization score.

Score	Description of Essay Organization
4	essay is **well structured** and is organized in a way that logically develops an argument
3	essay is **fairly well structured** but could somewhat benefit from reorganization
2	essay is **poorly structured** and would greatly benefit from reorganization
1	essay is **completely unstructured** and requires major reorganization

Our annotators were selected from over 30 applicants who were familiarized with the scoring rubric and given sample essays to score. The six who were most consistent with the expected scores were given additional essays to annotate. To ensure consistency in scoring, we randomly selected a large subset of our corpus (846 essays) to have graded by two different annotators. Analysis of these doubly annotated essays reveals that, though annotators only exactly agree on the organization score of an essay 29% of the time, the scores they apply are within 0.5 points in 71% of essays and within 1.0 point in 93% of essays. Additionally, if we treat one annotator's scores as a gold standard and the other annotator's scores as predictions, the predicted scores have a mean error of 0.54 and a mean squared error of 0.50. Table 3.3 shows the number of essays that received each of the seven scores for organization.

Table 3.3. Distribution of organization scores.

score	1.0	1.5	2.0	2.5	3.0	3.5	4.0
essays	24	14	35	146	416	289	79

3.4 Function Labeling

As mentioned before, a high score on organization means that writers introduce a topic, support their position, and conclude. If one or more of these elements are missing or if they appear out of order (e.g., the conclusion appears before the introduction), the resulting essay will typically be considered poorly organized. Hence, knowing the *discourse function label* of each paragraph in an essay would be helpful for predicting its organization score.

Two questions naturally arise. First, how can we obtain the discourse function label of each paragraph? One way is to automatically acquire such labels from a corpus of student essays where each paragraph is annotated with its discourse function label. To our knowledge, however, there is no publicly available corpus that is annotated with such information. As a result, we will resort to labeling a paragraph with its function label heuristically.

Second, which paragraph function labels would be most useful for scoring the organization of an essay? Based on our linguistic intuition, we identify four potentially useful paragraph function labels: Introduction, Body, Rebuttal, and Conclusion. Table 3.4 gives the descriptions of these labels.

Table 3.4. Description of paragraph function labels.

Label	Name	Paragraph Type
I	Introduction	introduces essay topic and states author's position and main ideas
B	Body	provides reasons, evidence, and examples to support main ideas
C	Conclusion	summarizes and concludes arguments made in body paragraphs
R	Rebuttal	considers counter-arguments to thesis or main ideas

Setting aside for the moment the problem of exactly how to predict an essay's organization score given its paragraph sequence, the problem of obtaining paragraph labels to use for this task still remains. As mentioned above, we adopt a heuristic approach to paragraph function labeling. The question, then, is: what kind of knowledge sources should our heuristics be based on? We have identified two types of knowledge sources that are potentially useful for paragraph labeling. The first of these are positional, dealing with where in the essay a paragraph appears. So for example, the first paragraph in an essay is likely to be an Introduction, while the last is likely to be a Conclusion. A paragraph in any other position, on the other hand, is more likely to be a Body or Rebuttal paragraph.

A second potentially useful knowledge source involves the types of sentences appearing in a paragraph. This idea presupposes that, like paragraphs, sentences too can have discourse function labels indicating the logical role they play in an argument. The sentence label schema we propose, which is described in Table 3.5, is based on work in discourse structure by Burstein et al. (2003), but features additional sentence labels.

To illustrate why these sentence function labels may be useful for paragraph labeling, consider a paragraph containing a Thesis sentence. The presence of a Thesis sentence is a

Table 3.5. Description of sentence function labels.

Label	Name	Sentence Function
P	Prompt	restates the prompt given to the author and contains no new material or opinions
T	Transition	shifts the focus to new topics but contains no meaningful information
H	Thesis	states the author's position on the topic for which he/she is arguing
M	Main Idea	asserts reasons and foundational arguments that support the thesis
E	Elaboration	further explains reasons and ideas but contains no evidence or examples
S	Support	provides evidence and examples to support the claims made in other statements
C	Conclusion	summarizes and concludes the entire argument or one of the main ideas
R	Rebuttal	considers counter-arguments that contrast with the thesis or main ideas
O	Solution	puts to rest the questions and problems brought up by counter-arguments
U	Suggestion	proposes solutions to the problems brought up by the argument

strong indicator that the paragraph containing it is either an Introduction or Conclusion. Similarly, a paragraph containing Rebuttal or Solution sentences is more likely to be a Body or Rebuttal paragraph.

Hence, to obtain a paragraph's function label, we need to first label its sentences. However, we are faced with the same problem: how can we obtain the sentence function labels? One way is to learn them from a corpus where each sentence is manually annotated with its sentence function label, which is the approach adopted by Burstein et al. (2003). However, this annotated corpus is not publicly available. In fact, to our knowledge, there is no publicly-available corpus that is annotated with sentence function labels. Consequently, we adopt a heuristic approach to sentence function labeling.

Overall, we created a knowledge-lean set of heuristic rules labeling paragraphs and sentences. Because many of the paragraph labeling heuristics depend on the availability of sentence labels, we will describe the sentence labeling heuristics first. For each sentence function label x, we identify several features whose presence increases our confidence that a given sentence is an example of x. So for example, the presence of any of the words "agree",

"think", or "opinion" increases our confidence that the sentence they occur in is a Thesis. If the sentence instead contains words such as "however", "but", or "argue", these increase our confidence that the sentence is a Rebuttal. The features we examine for sentence labeling are not limited to words, however. Each content word the sentence shares with the essay prompt gives us evidence that the sentence is a restatement of the prompt. Having searched a sentence for all these clues, we finally assign the sentence the function label having the most support among the clues found.

The heuristic rules for paragraph labeling are similar in nature, though they depend heavily on the labels of a paragraph's component sentences. If a paragraph contains Thesis, Prompt, or Background sentences, the paragraph is likely to be an Introduction. However, if a paragraph contains Main Idea, Support, or Conclusion sentences, it is likely to be a Body paragraph. Finally, as mentioned previously, some positional information is used in labeling paragraphs. For example, a paragraph that is the first paragraph in an essay is likely to be an Introduction, but a paragraph that is neither the first nor the last is likely to be either a Rebuttal or Body paragraph. After searching a paragraph for all these features, we gather the pieces of evidence in support of each paragraph label and assign the paragraph the label having the most support.[2]

3.5 Heuristic-Based Organization Scoring

Having applied labels to each paragraph in an essay, how can we use these labels to predict the essay's score? Recall that the importance of each paragraph label stems not from the label itself, but from the sequence of labels it appears in. Motivated by this observation, we exploit a technique that is commonly used in bioinformatics — *sequence alignment*. While sequence alignment has also been used in text and paraphrase generation (e.g., Barzilay and

[2]See our website at `http://www.hlt.utdallas.edu/~alan/ICLE/` for the complete list of heuristics.

Lee (2002) and Barzilay and Lee (2003)), it has not been extensively applied to other areas of language processing, including essay scoring. In this section, we will present two heuristic approaches to organization scoring, one based on aligning *paragraph sequences* and the other on aligning *sentence sequences*.

3.5.1 Aligning Paragraph Sequences

As mentioned above, our first approach to heuristic organization scoring involves aligning paragraph sequences. Specifically, this approach operates in two steps. Given an essay e in the test set, we (1) find the k essays in the training set that are most similar to e via paragraph sequence alignment, and then (2) predict the organization score of e by aggregating the scores of its k nearest neighbors obtained in the first step. Below we describe these two steps in detail.

First, to obtain the k nearest neighbors of e, we employ the Needleman-Wunsch alignment algorithm (Needleman and Wunsch, 1970), which computes a similarity score for any pair of essays by finding an optimal alignment between their paragraph sequences. To illustrate why we believe sequence alignment can help us determine which essays are most similar, consider two example essays. One essay, which we will call IBBBC, begins with an Introductory paragraph, follows it with three Body paragraphs, and finally ends with a Concluding paragraph. Another essay CRRRI begins with a paragraph stating its Conclusion, follows it with three Rebuttal paragraphs, and ends with a paragraph Introducing the essay's topic. We can tell by a casual glance at the sequences that any reasonable similarity function should tell us that they are not very similar. The Needleman-Wunsch alignment algorithm has this effect since the score of the alignment it produces would be hurt by the facts that (1) there is not much overlap in the sets of paragraph labels each contains, and (2)

the paragraph labels they do share (I and C) do not occur in the same order. The resulting alignment would therefore contain many mismatches or indels.[3]

If we now consider a third essay whose paragraph sequence could be represented as IBRBC, a good similarity function should tell us that IBBBC and IBRBC are very similar. The Needleman-Wunsch alignment score between the two paragraph sequences has this property, as the alignment algorithm would discover that the two sequences are identical except for the third paragraph label, which could be mismatched for a small penalty. We would therefore conclude that the IBBBC and IBRBC essays should receive similar organization scores.

To fully specify how to find the k nearest neighbors of an essay, we need to define a similarity function between paragraph labels. In sequence alignment, similarity function $S(i, j)$ tells us how likely it is that symbol i (in our case, a paragraph label) will be substituted with another symbol j. While we expect that in an alignment between high-scoring essays, an Introduction paragraph is most likely to be aligned with another Introduction paragraph, how much worse should the alignment score be if an Introduction paragraph needs to be mismatched with a Rebuttal paragraph or replaced with an indel? We solve this problem by heuristically defining the similarity function as follows: $S(i, j) = 1$ when $i = j$, $S(i, j) = -1$ when $i \neq j$, and also $S(i, -) = S(-, i) = -1$, where '$-$' is an indel. In other words, the similarity function encourages the alignment between two identical function labels and discourages the alignment between two different function labels, regardless of the type of function labels.

After obtaining the k nearest neighbors of e, the next step is to predict the organization score of e by aggregating the scores of its k nearest neighbors into one number. (Note that we know the organization score of each nearest neighbor, since they are all taken from the

[3]In pairwise sequence alignment, a mismatch occurs when one symbol has to be substituted for another to make two sequences match. An indel indicates that in order to transform one sequence to match another, we must either **in**sert a symbol into one sequence or **del**ete a symbol from the other sequence.

training set.) One natural way to do this would be to take the mean, median, or mode of its k nearest neighboring essays from the training set. Hence, our first heuristic method H_p for scoring organization has three variants.

3.5.2 Aligning Sentence Sequences

An essay's paragraph sequence captures information about its organization at a high level, but ignores much of its lower level structure. Since we have also heuristically labeled sentences, it now makes sense to examine the sequences of sentence function labels within an essay's paragraphs. The intuition is that at least some portion of an essay's organization score can be attributed to the organization of the sentence sequences of its component paragraphs.

To address this concern, we propose a second heuristic approach to organization scoring. Given a test essay e, we first find for each *paragraph* in e the k *paragraphs* in the training set that are most similar to it. Specifically, each paragraph is represented by its sequence of *sentence* function labels. Given this paragraph representation, we can find the k nearest neighbors of a paragraph by applying the Needleman-Wunsch algorithm described in the previous subsection to align *sentence* sequences, using the same similarity function we defined above.

Next, we score each paragraph p_i by aggregating the scores of its k nearest neighbors obtained in the first step, assuming the score of a nearest neighbor paragraph is the same as the organization score of the training set essay containing it. As before, we can employ the mean, median, or mode to aggregate the scores of the nearest neighbors of p_i.

Finally, we predict the organization score of e by aggregating the scores of its paragraphs obtained in the second step. Again, we can employ mean, median, or mode to aggregate the scores. Since we have three ways of aggregating the scores of a paragraph's nearest neighbors and three ways of aggregating the resulting paragraph scores, this second method H_s for scoring organization has nine variants.

3.6 Learning-Based Organization Scoring

In the previous section, we proposed two heuristic approaches to organization scoring, one based on aligning paragraph label sequences and the other based on aligning sentence label sequences. In the process of constructing these two systems, however, we created a lot of information about the essays which might also be useful for organization scoring, but which the heuristic systems are unable to exploit. To remedy the problem, we introduce three learning-based systems which abstract the additional information we produced in three different ways. In each system, we use the SVM^{light} (Joachims, 1999) implementation of regression support vector machines (SVMs) (Cortes and Vapnik, 1995) to train a regressor because SVMs have been frequently and successfully applied to a variety of NLP problems.

3.6.1 Linear Kernel

Owing to the different ways we presented of combining the scores of an essay's nearest neighbors, the paragraph label sequence alignment approach has three variants, and its sentence label sequence alignment counterpart has nine. Unfortunately, these heuristic approaches suffer from two major weaknesses. First, it is not intuitively clear which of these 12 ways for predicting an essay's organization score is clearly better than the others. Second, it is not clear that the k nearest neighbors of an essay will always be similar to it with respect to organization score. While we do expect the alignment scores between good essays with reasonable paragraph sequences to be high, poorly organized essays by their nature have more random paragraph sequences. Hence, we have no intuition about the k nearest neighbors of a poor essay, as it may have as high an alignment score with another poorly organized essay as with a good essay.

Our solution to these problems is to use the organization scores obtained by the 12 heuristic variants as features in a linear kernel SVM learner. We believe that using the estimates given by all the 12 variants of the two heuristic approaches rather than only one

of them addresses the first weakness mentioned above. The second weakness, on the other hand, is addressed by treating the organization score predictions obtained by the nearest neighbor methods as features for an SVM learner rather than as estimates of an essay's organization score.

The approach we have just described, however, does not exploit the full power of linear kernel SVMs. One strength of linear kernels is that they make it easy to incorporate a wide variety of different types of features. In an attempt to further enhance the prediction capability of the SVM regressor, we will provide it with not only the 12 features derived from the heuristic-based approaches, but also with two additional types of features.

First, to give our learner more direct access to the information we used to heuristically predict essay scores, we can extract *paragraph label subsequences*[4] from each essay and use them as features. To illustrate the intuition behind these features, consider two paragraph subsequences: Introduction–Body and Rebuttal–Introduction. It is fairly typical to see the first subsequence, I–B, at the beginning of a good essay, so its occurrence should give us a small amount of evidence that the essay it occurs in is well-organized. The presence of the second subsequence, R–I, however, should indicate that its essay's organization is poor because, in general, a good essay should not give a Rebuttal before an Introduction. Because we can envision subsequences of various lengths being useful, we create a binary presence or absence feature in the linear kernel for each paragraph subsequence of length 1, 2, 3, 4, or 5 appearing in the training set.

Second, we employ *sentence label subsequences* as features in the linear kernel. Recall that when describing our alignment-based nearest neighbor organization score prediction methods, we noted that an essay's organization score may be partially attributable to how well the sentences within its paragraphs are organized. For example, if one of an essay's paragraphs contains the sentence label subsequence Main Idea–Elaboration–Support–Conclusion

[4]Note that a subsequence is not necessarily contiguous.

this gives us some evidence that the essay is overall well-organized since one of its component paragraphs contains this reasonably-organized subsequence. An essay with a paragraph containing the subsequence Conclusion–Support–Thesis–Rebuttal, however, is likely to be poorly organized because this is a poorly-organized subsequence. Since sentence label subsequences of differing lengths may be useful for score prediction, we create a binary presence or absence feature for each sentence label subsequence of length 1, 2, 3, 4, or 5 in the training set.

While the number of nearest neighbor features is manageable, the presence of a large number of features can sometimes confuse a learner. For that reason, we do feature selection on the two types of subsequence features, selecting only 100 features for each type that has the highest information gain (see Yang and Pedersen (1997) for details). We call the system resulting from the use of these three types of features Rl_{nps} because it uses **R**egression with **l**inear kernel to predict essay scores, and it uses **n**earest neighbor, **p**aragraph subsequence, and **s**entence subsequence features.

3.6.2 String Kernel

In a traditional learning setting, the feature set employed by an off-the-shelf learning algorithm typically consists of *flat* features (i.e., features whose values are discrete- or real-valued, as the ones described in the Linear Kernel subsection). Advanced machine learning algorithms such as SVMs, on the other hand, have enabled the use of *structured* features (i.e., features whose values are structures such as parse trees and sequences), owing to their ability to employ *kernels* to efficiently compute the similarity between two potentially complex structures.

Perhaps the most obvious advantage of employing structured features is *simplicity*. To understand this advantage, consider learning in a traditional setting. Recall that we can only employ flat features in this setting, as we did with the linear kernel. Hence, if we

want to use information from a parse tree as features, we will need to design heuristics to extract the desired parse-based features from parse trees. For certain tasks, designing a good set of heuristics can be time-consuming and sometimes difficult. On the other hand, SVMs enable a parse tree to be employed directly as a structured feature, obviating the need to design heuristics to extract information from potentially complex structures. However, structured features have only been applied to a handful of NLP tasks such as semantic role labeling (Moschitti, 2004), syntactic parsing and named entity identification (Collins and Duffy, 2002), relation extraction (Bunescu and Mooney, 2005), and coreference resolution (Versley et al., 2008). Our goal here is to explore this rarely-exploited capability of SVMs for the task of essay scoring.

While the vast majority of previous NLP work on using structured features have involved tree kernels, we employ a kernel that is rarely investigated in NLP: *string kernels* (Lodhi et al., 2002). Informally, a string kernel aims to efficiently compute the similarity between two strings (or sequences) of symbols based on the similarity of their subsequences. We apply string kernels to essay scoring as follows: we represent each essay using its paragraph function label sequence, and employ a string kernel to compute the similarity between two essays based on this representation. Typically, a string kernel takes as input two parameters: K (which specifies the length of the subsequences in the two strings to compare) and λ (which is a value between 0 and 1 that specifies whether matches between non-contiguous subsequences in the two strings should be considered as important as matches between contiguous subsequences). In our experiments, we select values for these parameters in a somewhat arbitrary manner. In particular, since λ ranges between 0 and 1, we simply set it to 0.5. For K, since in the flat features we considered all paragraph label sequences of lengths from 1 to 5, we again take the middle value, setting it to 3. We call the system using this kernel *Rs* because it uses a **R**egression SVM with a **s**tring kernel to predict essay scores.

3.6.3 Alignment Kernel

In general, the purpose of a kernel function is to measure the similarity between two examples. The string kernel we described in the previous subsection is just one way of measuring the similarity of two essays given their paragraph sequences. While this may be the most obvious way to use paragraph sequence information from a machine learning perspective, our earlier use of the Needleman-Wunsch algorithm suggests a more direct way of extracting structured information from paragraph sequences.

More specifically, recall that the Needleman-Wunsch algorithm finds an optimal alignment between two paragraph sequences, where an optimal alignment is defined as an alignment having the highest possible alignment score. The optimal alignment score can be viewed as another similarity measure between two essays. As such, with some slight modifications, the alignment score between two paragraph sequences can be used as the kernel value for an Alignment Kernel.[5] We call the system using this kernel *Ra* because it uses a **R**egression SVM with an **a**lignment kernel to predict essay scores.

3.6.4 Combining Kernels

Recall that the flat features are computed using a linear kernel, while the two types of structured features are computed using string and alignment kernels. If we want our learner to make use of more than one of these types of features, we need to employ a *composite* kernel to combine them. Specifically, we define and employ the following composite kernel:

$$K_c(F_1, F_2) = \frac{1}{n} \sum_{i=1}^{n} K_i(F_1, F_2),$$

[5]In particular, we note that for theoretical reasons, a kernel function must always return a non-negative value. The alignment score function does not have this property, so we increase all alignment scores until their theoretical minimum value is 0.

where F_1 and F_2 are the full set of features (containing both flat and structured features) that represent the two essays under consideration, K_i is the ith kernel we are combining, and n is the number of kernels we are combining. To ensure that each kernel under consideration contributes equally to the composite kernel, each kernel value $K_i(F_1, F_2)$ is normalized so that its value falls between 0 and 1.

3.7 Evaluation

In this section we will discuss how we evaluate our organization scoring system.

3.7.1 Evaluation Metrics

We designed three evaluation metrics to measure the error of our organization scoring system. The simplest metric, $S1$, is perhaps the most intuitive. It measures the frequency at which a system predicts the wrong score out of the seven possible scores. Hence, a system that predicts the right score only 25% of the time would receive an $S1$ score of 0.75.

The $S2$ metric is slightly less intuitive than $S1$, but no less reasonable. It measures the average distance between the system's score and the actual score. This metric reflects the idea that a system that estimates scores close to the annotator-assigned scores should be preferred over a system whose estimations are further off, even if both systems estimate the correct score at the same frequency.

Finally, the $S3$ evaluation metric measures the average square of the distance between a system's organization score estimations and the annotator-assigned scores. The intuition behind this system is that not only should we prefer a system whose estimations are close to the annotator scores, but we should also prefer one whose estimations are not too frequently very far away from the annotator scores. These three scores are given by:

$$\frac{1}{N}\sum_{A_i \neq E_i} 1, \quad \frac{1}{N}\sum_{i=1}^{N} |A_i - E_i|, \quad \frac{1}{N}\sum_{i=1}^{N} (A_i - E_i)^2,$$

Table 3.6. Organization results.

	System	$S1$	$S2$	$S3$
1	Avg	.585	.412	.348
2	H_p	.548	.339	.198
3	H_s	.575	.397	.329
4	Rl_{nps}	.520	.331	.186
5	Rs	.577	.369	.222
6	Ra	.686	.519	.429
7	Rls_{nps}	.534	.332	.187
8	Rla_{nps}	.541	.332	.178
9	Rsa	.517	.325	.177
10	$Rlsa_{nps}$	.517	.323	.175

where A_i and E_i are the annotator assigned and system estimated scores respectively for essay i, and N is the number of essays. Since many of the systems we have described assign test essays real-valued organization scores, to obtain E_i for system $S1$ we round the outputs of each system to the nearest of the seven scores the human annotators were permitted to assign (1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0).

To test our system, we performed 5-fold cross validation on our 1003 essay set, micro-averaging our results into three scores corresponding to the three scoring metrics described above.

3.7.2 Results and Discussion

The average baseline. As mentioned before, there is no standard baseline for organization modeling against which we can compare our systems. To start with, we employ a simple "average" baseline. Avg computes the average organization score of essays in the training set and assigns this score to each test set essay. Results of this baseline are shown in row 1 of Table 3.6. Though simple, this baseline is by no means easy-to-beat, since 41% of the essays have a score of 3, and 96% of the essays have a score that is within one point of 3.

Heuristic baselines. Recall that we have 12 versions of the two heuristic approaches to organization prediction. We show the the best results achieved by the three versions based on aligning paragraph label sequences in row 2 (H_p) and the best results achieved by the nine versions based on aligning sentence label sequences in row 3 (H_s) of Table 3.6. It is clear from the results that the H_p systems yielded the best baseline predictions under all three scoring metrics, performing significantly better than both the Avg and H_s systems ($p < 0.01$) with respect to the $S2$ and $S3$ metrics, but its $S1$ performance is less significant with respect to Avg ($p < 0.1$) and is indistinguishable at even the $p < 0.1$ level from H_s.[6] In general, however, it appears to be the case that systems based on aligning paragraph label sequences achieve better results than systems that attempt to align sentence label sequences.

Learning-based approaches. Rows 4–6 of Table 3.6 show the results we obtained using each of the three single-kernel systems. When compared to the best baseline, these results suggest that H_p is a pretty good heuristic approach to organization scoring. In fact, only one of these three learning-based systems (Rl_{nps}) performs better than H_p under the three scoring metrics, and in each case, the difference between the two is not significant even at $p < 0.1$. This suggests that, even though Rl_{nps} performs slightly better than H_p, the only major benefit we have obtained by using the linear kernel is that it has made it unnecessary for us to choose between the 12 proposed heuristic systems.

Considering that the second best one-kernel system, Rs, does not have access to any of the nearest neighbor features, which have already proven useful, its performance seems reasonably good in that its performance is at least better than the Avg system. This suggests that, even though Rs does not perform exceptionally, it is extracting some useful information for organization scoring from the heuristically assigned paragraph label sequences. The best one-kernel system, Rl_{nps}, however, is significantly better than Rs with respect to all

[6] All significance tests are two-tailed paired t-tests.

three scoring metrics, with $p < 0.1$ for $S1$ and $p < 0.05$ for $S2$ and $S3$. By contrast, it initially appears that the alignment kernel is not extracting any useful information from these paragraph sequences at all, since its $S1$, $S2$, and $S3$ scores are all much worse than all of the baseline systems. The second best one-kernel system Rs performs significantly better than Ra at $p < 0.01$ for all three scoring metrics.

Next, we explore the impact of composite kernels, which allow our learners to make use of multiple types of flat and structured features. Specifically, the results shown in rows 7–9 are obtained by combining two kernels at a time. These experiments reveal the surprising result that the two worst performing single-kernel systems, Rs and Ra, when combined into Rsa, yield the best two-kernel system results, which are significant with respect to the best one-kernel system results under $S3$ at $p < 0.1$. This result suggests that these two different methods of extracting information from paragraph sequences provide us with different kinds of evidence useful for organization scoring, although neither method by itself was exceptionally useful. Though Rsa does not have any access to nearest neighbor information, it still performs significantly better than H_p at $p < 0.05$ under $S1$ and $S3$.

While we have already pointed out that Rsa is the best composite two-kernel system, it is not clear which of Rls_{nps} and Rla_{nps} is second-best. Neither system consistently performs better than the other under all three scoring metrics, and the differences between them are not significant even at $p < 0.1$. It is clear only that Rsa is better than both, as its scores are statistically significantly better at $p < 0.01$ with respect to Rls_{nps} and Rla_{nps} under at least one of the three scoring metrics in each case.

Finally, in the last row of Table 3.6, we combine all three kernels into one SVM learner. The most important lesson we learn from this experiment is that each of the three kernels provides the learner with a different kind of useful information, so that a composite kernel using all three sources of information performs better than any system using fewer kernels. Although the improvements over the best two-kernel system (Rsa) and one-kernel system

(Rl_{nps}) are small, they are still statistically significant at $p < 0.1$ under one of the scoring metrics, $S3$. When we compare this combined system to the best baseline (H_p), we discover the improvements derived from the three-kernel system are significant improvements over it at $p < 0.05$ and $p < 0.01$ with respect to $S1$ and $S3$ respectively.

Feature analysis. To better understand which of the three flat features (nearest neighbors, paragraph label sequences, or sentence label sequences) contributes the most to the linear kernel portion of the systems' performances, we analyze the three feature types on Rl_{nps} using the backward elimination feature selection algorithm. First, we remove each of the three feature groups independently from the Rl_{nps}'s feature set and determine which of the three removals yields the best performance according to each scoring metric. Next, among the remaining two feature groups, we repeat the same step, removing each of the two groups independently from the feature set to determine which of the two removals yields the best performance.

The trend is consistent among all three scoring metrics: the first feature type to remove is paragraph sequences (meaning that they are the least important) and the last to remove is the nearest neighbor features. Nevertheless, performance always drops when a feature type is removed, indicating that all three feature types contribute positively to overall performance. The fact that flat paragraph sequence features proved to be least useful highlights the importance of the structured methods we presented for using paragraph sequence information.

3.8 Summary

We have investigated the relatively less-studied problem of modeling the organization in student essays. The contributions of our work include the novel application of two techniques from bioinformatics and machine learning — sequence alignment and string kernels, as well as the introduction of alignment kernels — to essay scoring. We showed that each technique

makes a significant contribution to a scoring system, and we hope that this work will increase awareness of these powerful techniques among NLP researchers. Finally, to stimulate work on this problem, we make our corpus of annotated essays available to other researchers.

CHAPTER 4

THESIS CLARITY SCORING[1]

4.1 Introduction

Automated essay scoring, the task of employing computer technology to evaluate and score written text, is one of the most important educational applications of natural language processing (NLP) (see Shermis and Burstein (2003) and Shermis et al. (2010) for an overview of the state of the art in this task). A major weakness of many existing scoring engines such as the Intelligent Essay Assessor^{TM}(Landauer et al., 2003) is that they adopt a holistic scoring scheme, which summarizes the quality of an essay with a single score and thus provides very limited feedback to the writer. In particular, it is not clear which dimension of an essay (e.g., style, coherence, relevance) a score should be attributed to. Recent work addresses this problem by scoring a particular dimension of essay quality such as coherence (Miltsakaki and Kukich, 2004), technical errors, Relevance to Prompt (Higgins et al., 2004), and organization (Persing et al., 2010). Essay grading software that provides feedback along multiple dimensions of essay quality such as E-*rater*/Criterion (Attali and Burstein, 2006) has also begun to emerge.

Nevertheless, there is an essay scoring dimension for which few computational models have been developed — *thesis clarity*. Thesis clarity refers to how clearly an author explains the *thesis* of her essay, i.e., the position she argues for with respect to the topic on which the essay is written.[2] An essay with a high thesis clarity score presents its thesis in a way that is easy for the reader to understand, preferably but not necessarily directly, as in essays with

[1]This chapter was previously published in Persing and Ng (2013).

[2]An essay's thesis is the overall message of the *entire* essay. This concept is unbound from the the concept of thesis sentences, as even an essay that never explicitly states its thesis in any of its sentences may still have an overall message that can be inferred from the arguments it makes.

explicit thesis sentences. It additionally contains no errors such as excessive misspellings that make it more difficult for the reader to understand the writer's purpose.

Our goals in this chapter are two-fold. First, we aim to develop a computational model for scoring the thesis clarity of student essays. Because there are many reasons why an essay may receive a low thesis clarity score, our second goal is to build a system for determining why an essay receives its score. We believe the feedback provided by this system will be more informative to a student than would a thesis clarity score alone, as it will help her understand which aspects of her writing need to be improved in order to better convey her thesis. To this end, we identify five common errors that impact thesis clarity, and our system's purpose is to determine which of these errors occur in a given essay. We evaluate our thesis clarity scoring model and error identification system on a data set of 830 essays annotated with both thesis clarity scores and errors.

In sum, our contributions in this chapter are three-fold. First, we develop a scoring model and error identification system for the thesis clarity dimension on student essays. Second, we use features explicitly designed for each of the identified error types in order to train our scoring model, in contrast to many existing systems for other scoring dimensions, which use more general features developed without the concept of error classes. Third, we make our data set consisting of thesis clarity annotations of 830 essays publicly available in order to stimulate further research on this task. Since progress in thesis clarity modeling is hindered in part by the lack of a publicly annotated corpus, we believe that our data set will be a valuable resource to the NLP community.

4.2 Corpus Information

We use as our corpus the 4.5 million word International Corpus of Learner English (ICLE) (Granger et al., 2009) described in Chapter 3.2 We select a subset consisting of 830 argumentative essays from the ICLE to annotate and use for training and testing of our models

Table 4.1. Description of each thesis clarity score.

Score	Description of Thesis Clarity
4	essay presents a **very clear thesis** and requires little or no clarification
3	essay presents a **moderately clear thesis** but could benefit from some clarification
2	essay presents an **unclear thesis** and would greatly benefit from further clarification
1	essay presents **no thesis of any kind** and it is difficult to see what the thesis could be

of essay thesis clarity. Table 3.1 shows several of the thirteen topics selected for annotation. Fifteen native languages are represented in the set of essays selected for annotation.

4.3 Corpus Annotation

For each of the 830 argumentative essays, we ask two native English speakers to (1) score it along the thesis clarity dimension and (2) determine the subset of the five pre-defined errors that detracts from the clarity of its thesis.

Scoring. Annotators evaluate the clarity of each essay's thesis using a numerical score from 1 to 4 at half-point increments (see Table 4.1 for a description of each score). This contrasts with previous work on essay scoring, where the corpus is annotated with a binary decision (i.e., *good* or *bad*) for a given scoring dimension (e.g., Higgins et al. (2004)). Hence, our annotation scheme not only provides a finer-grained distinction of thesis clarity (which can be important in practice), but also makes the prediction task more challenging.

To ensure consistency in annotation, we randomly select 100 essays to have graded by both annotators. Analysis of these essays reveals that, though annotators only exactly agree on the thesis clarity score of an essay 36% of the time, the scores they apply are within 0.5 points in 62% of essays and within 1.0 point in 85% of essays. Table 4.2 shows the number of essays that receive each of the seven scores for thesis clarity.

Table 4.2. Distribution of thesis clarity scores.

score	1.0	1.5	2.0	2.5	3.0	3.5	4.0
essays	4	9	52	78	168	202	317

Error identification. To identify what kinds of errors make an essay's thesis unclear, we ask one of our annotators to write 1–4 sentence critiques of thesis clarity on 527 essays, and obtain our list of five common error classes by categorizing the things he found to criticize. We present our annotators with descriptions of these five error classes (see Table 4.3), and ask them to assign zero or more of the error types to each essay.

It is important to note that we ask our annotators to mark an essay with one of these errors only when the error makes the thesis less clear. So for example, an essay whose thesis is irrelevant to the prompt but is explicitly and otherwise clearly stated would not be marked as having a Relevance to Prompt error. If the irrelevant thesis is stated in such a way that its inapplicability to the prompt causes the reader to be confused about what the essay's purpose is, however, then the essay would be assigned a Relevance to Prompt error.

Table 4.3. Description of thesis clarity errors.

Id	**Error**	**Description**
CP	**Confusing Phrasing**	The thesis is phrased oddly, making it hard to understand the writer's point.
IPR	**Incomplete Prompt Response**	The thesis seems to leave some part of a multi-part prompt unaddressed.
R	**Relevance to Prompt**	The apparent thesis's weak relation to the prompt causes confusion.
MD	**Missing Details**	The thesis leaves out important detail needed to understand the writer's point.
WP	**Writer Position**	The thesis describes a position on the topic without making it clear that this is the position the writer supports.

To measure inter-annotator agreement on error identification, we ask both annotators to identify the errors in the same 100 essays that were doubly-annotated with thesis clarity

scores. We then compute Cohen's Kappa (Carletta, 1996) on each error from the two sets of annotations, obtaining an average Kappa value of 0.75, which indicates fair agreement. Table 4.4 shows the number of essays assigned to each of the five thesis clarity errors. As we can see, Confusing Phrasing, Incomplete Prompt Response, and Relevance to Prompt are the major error types.

Table 4.4. Distribution of thesis clarity errors.

error	CP	IPR	R	MD	WP
essays	152	123	142	47	39

Relationship between clarity scores and error classes. To determine the relationship between thesis clarity scores and the five error classes, we train a linear SVM regressor using the SVM^{light} software package (Joachims, 1999) with the five error types as independent variables and the reduction in thesis clarity score due to errors as the dependent variable. More specifically, each training example consists of a target, which we set to the essay's thesis clarity score minus 4.0, and six binary features, each of the first five representing the presence or absence of one of the five errors in the essay, and the sixth being a bias feature which we always set to 1. Representing the reduction in an essay's thesis clarity score with its thesis clarity score minus 4.0 allows us to more easily interpret the error and bias weights of the trained system, as under this setup, each error's weight should be a negative number reflecting how many points an essay loses due to the presence of that error. The bias feature allows for the possibility that an essay may lose points from its thesis clarity score for problems not accounted for in our five error classes. By setting this bias feature to 1, we tell our learner that an essay's default score may be less than 4.0 because these other problems may lower the average score of otherwise perfect essays.

After training, we examined the weight parameters of the learned regressor and found that they were all negative: -0.6 for CP, -0.5998 for IPR, -0.8992 for R, -0.6 for MD, -0.8

for WP, and -0.1 for the bias. These results are consistent with our intuition that each of the enumerated error classes has a negative impact on thesis clarity score. In particular, each has a demonstrable negative impact, costing essays an average of more than 0.59 points when it occurs. Moreover, this set of errors accounts for a large majority of all errors impacting thesis clarity because unenumerated errors cost essays an average of only one-tenth of one point on the four-point thesis clarity scale.

4.4 Error Classification

In this section, we describe in detail our system for identifying thesis clarity errors.

4.4.1 Model Training and Application

We recast the problem of identifying which thesis clarity errors apply to an essay as a multi-label classification problem, wherein each essay may be assigned zero or more of the five pre-defined error types. To solve this problem, we train five binary classifiers, one for each error type, using a one-versus-all scheme. So in the binary classification problem for identifying error e_i, we create one training instance from each essay in the training set, labeling the instance as positive if the essay has e_i as one of its labels, and negative otherwise. Each instance is represented by seven types of features, including two types of baseline features (Section 4.2) and five types of features we introduce for error identification (Section 4.3).

After creating training instances for error e_i, we train a binary classifier, b_i, for identifying which test essays contain error e_i. We use SVM^{light} for classifier training with the regularization parameter, C, set to c_i. To improve classifier performance, we perform feature selection. While we employ seven types of features (see Sections 4.2 and 4.3), only the word n-gram features are subject to feature selection.[3] Specifically, we employ the top n_i n-gram

[3]We do not apply feature selection to the remaining feature types since each of them includes only a small number of overall features that are expected to be useful.

features as selected according to information gain computed over the training data (see Yang and Pedersen (1997) for details). Finally, since each classifier assigns a real value to each test essay presented to it indicating its confidence that the essay should be assigned error e_i, we employ a classification threshold t_i to decide how high this real value must be in order for our system to conclude that an essay contains error e_i.

Using held-out validation data, we jointly tune the three parameters in the previous paragraph, c_i, n_i, and t_i, to optimize the F-measure achieved by b_i for error e_i.[4] However, an exact solution to this optimization problem is computationally expensive. Consequently, we find a local maximum by employing the simulated annealing algorithm (Kirkpatrick et al., 1983), altering one parameter at a time to optimize F-measure by holding the remaining parameters fixed.

After training the classifiers, we use them to classify the test set essays. The test instances are created in the same way as the training instances.

4.4.2 Baseline Features

Our **B**aseline system for error classification employs two types of features. First, since labeling essays with thesis clarity errors can be viewed as a text categorization task, we employ lemmatized word unigram, bigram, and trigram features that occur in the essay that have not been removed by the feature selection parameter n_i. Because the essays vary greatly in length, we normalize each essay's set of word features to unit length.

[4]For parameter tuning, we employ the following values. c_i may be assigned any of the values 10^2, 10^3, 10^4, 10^5, or 10^6. n_i may be assigned any of the values 3000, 4000, 5000, or ALL, where ALL means all features are used. For t_i, we split the range of classification values b_i returns for the test set into tenths. t_i may take the values 0.0, 0.1, 0.2, ..., 1.0, and X, where 0.0 classifies all instances as negative, 0.1 classifies only instances b_i assigned values in the top tenth of the range as positive, and so on, and X is the default threshold, labeling essays as positive instances of e_i only if b_i returns for them a value greater than 0. It was necessary to assign t_i in this way because the range of values classifiers return varies greatly depending on which error type we are classifying and which other parameters we use. This method gives us reasonably fine-grained thresholds without having to try an unreasonably large number of values for t_i.

The second type of baseline features is based on random indexing (Kanerva et al., 2000). Random indexing is "an efficient, scalable and incremental alternative" (Sahlgren, 2005) to Latent Semantic Indexing (Deerwester et al., 1990; Landauer and Dumais, 1997) which allows us to automatically generate a semantic similarity measure between any two words. We train our random indexing model on over 30 million words of the English Gigaword corpus (Parker et al., 2009) using the S-Space package (Jurgens and Stevens, 2010). We expect that features based on random indexing may be particularly useful for the Incomplete Prompt Response and Relevance to Prompt errors because they may help us find text related to the prompt even if some of its components have been rephrased (e.g., an essay may talk about "jail" rather than "prison", which is mentioned in one of the prompts). For each essay, we therefore generate four random indexing features, one encoding the entire essay's similarity to the prompt, another encoding the essay's highest individual sentence's similarity to the prompt, a third encoding the highest entire essay similarity to one of the prompt sentences, and finally one encoding the highest individual sentence similarity to an individual prompt sentence. Since random indexing does not provide a straightforward way to measure similarity between groups of words such as sentences or essays, we use Higgins and Burstein's (2007) method to generate these features.

4.4.3 Novel Features

Next, we introduce five types of novel features.

Spelling. One problem we note when examining the information gain top-ranked features for the Confusing Phrasing error is that there are very few common confusing phrases that can contribute to this error. Errors of this type tend to be unique, and hence are not very useful for error classification (because we are not likely to see the same error in the training and test sets). We notice, however, that there are a few misspelled words at the top of

the list. This makes sense because a thesis sentence containing excessive misspellings may be less clear to the reader. Even the most common spelling errors, however, tend to be rare. Furthermore, we ask our annotators to only annotate an error if it makes the thesis less clear. The mere presence of an awkward phrase or misspelling is not enough to justify the Confusing Phrasing label. Hence, we introduce a **m**isspelling feature whose value is the number of spelling errors in an essay's most-misspelled sentence.[5]

Keywords. Improving the prediction of majority classes can greatly enhance our system's overall performance. Hence, since we have introduced the misspelling feature to enhance our system's performance on one of the more frequently occurring errors (Confusing Phrasing), it makes sense to introduce another type of feature to improve performance on the other two most frequent errors, Incomplete Prompt Response and Relevance to Prompt. For this reason, we introduce **k**eyword features. To use this feature, we first examine each of the 13 essay prompts, splitting it into its component pieces. For our purposes, a component of a prompt is a prompt substring such that, if an essay does not address it, it may be assigned the Incomplete Prompt Response label. Then, for each component, we manually select the most important (primary) and second most important (secondary) words that it would be good for a writer to use to address the component. To give an example, the lemmatized version of the third component of the second essay prompt in Table 3.1 is "it should rehabilitate they" rather than "it should rehabilitate them". For this component we selected "rehabilitate" as a primary keyword and "society" as a secondary keyword. To compute one of our keyword features, we compute the random indexing similarity between the essay and each group of primary keywords taken from components of the essay's prompt and assign the feature the lowest of these values. If this feature has a low value, that suggests that the essay may have

[5]We employ SCOWL (`http://wordlist.sourceforge.net/`) as our dictionary, assuming that a word that does not appear in the dictionary is misspelled.

an Incomplete Prompt Response error because the essay probably did not respond to the part of the prompt from which this value came. To compute another of the keyword features, we count the numbers of combined primary and secondary keywords the essay contains from each component of its prompt, and divide each number by the total number of primary and secondary features for that component. If the greatest of these fractions has a low value, that indicates the essay's thesis might not be very Relevant to the Prompt.[6]

Aggregated word n-gram features. Other ways we could measure our system's performance (such as macro F-score) would consider our system's performance on the less frequent errors no less important than its performance on the most frequent errors. For this reason, it now makes sense for us to introduce a feature tailored to help our system do better at identifying the least-frequent error types, Missing Details and Writer Position, each of which occurs in fewer than 50 essays. To help with identification of these error classes, we introduce aggregated **w**ord n-gram features. While we mention in the previous section one of the reasons regular word n-gram features can be expected to help with these error classes, one of the problems with regular word n-gram features is that it is fairly infrequent for the exact same useful phrase to occur too frequently. Additionally, since there are numerous word n-grams, some infrequent ones may just by chance only occur in positive training set instances, causing the learner to think they indicate the positive class when they do not. To address these problems, for each of the five error classes e_i, we construct two Aggregated word features $Aw+_i$ and $Aw-_i$. For each essay, $Aw+_i$ counts the number of word n-grams we believe indicate that an essay is a positive example of e_i, and $Aw-_i$ counts the number of word n-grams we believe indicate an essay is not an example of e_i. $Aw+$ n-grams for the Missing Details error tend to include phrases like "there is something" or "this statement",

[6]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for the complete list of keyword features.

while $Aw-$ n-grams are often words taken directly from an essay's prompt. N-grams used for Writer Position's $Aw+$ tend to suggest the writer is distancing herself from whatever statement is being made such as "every person", but n-grams for this error's $Aw-$ feature are difficult to find. Since $Aw+_i$ and $Aw-_i$ are so error specific, they are only included in an essay's feature representation when it is presented to learner b_i. So while aggregated word n-grams introduce ten new features, each learner b_i only sees two of these ($Aw+_i$ and $Aw-_i$).

We construct the lists of word n-grams that are aggregated for use as the $Aw+$ and $Aw-$ feature values in the following way. For each error class e_i, we sort the list of all features occurring at least ten times in the training set by information gain. A human annotator then manually inspects the top thousand features in each of the five lists and sorts each list's features into three categories. The first category for e_i's list consists of features that indicate an essay may be a positive instance. Each word n-gram from this list that occurs in an essay increases the essay's $Aw+_i$ value by one. Similarly, any word n-gram sorted into the second category, which consists of features the annotator thinks indicate a negative instance of e_i, increases the essay's $Aw-$ value by one. The third category just contains all the features the annotator did not believe were useful enough to either class, and we make no further use of those features. For most error types, only about 12% of the top 1000 features get sorted into one of the first two categories.

POS n-grams. We might further improve our system's performance on the Missing Details error type by introducing a feature that aggregates **p**art-of-speech (POS) tag n-grams in the same way that the Aw features aggregate word n-gram features. For this reason, we include POS tag 1, 2, 3, and 4-grams in the set of features we sort in the previous paragraph. For each error e_i, we select POS tag n-grams from the top thousand features of the information gain sorted list to count toward the $Ap+_i$ and $Ap-_i$ aggregation features. We believe this kind of feature may help improve performance on Missing Details because the list of features

aggregated to generate the $Ap+_i$ feature's value includes POS n-gram features like CC " NN " (scare quotes). This feature type may also help with Confusing Phrasing because the list of POS tag n-grams our annotator generated for its $Ap+_i$ contains useful features like DT NNS VBZ VBN (e.g., "these signals has been"), which captures noun-verb disagreement.

Semantic roles. Our last aggregated feature is generated using FrameNet-style semantic role labels obtained using SEMAFOR (Das et al., 2010). For each sentence in our data set, SEMAFOR identifies each semantic frame occurring in the sentence as well as each frame element that participates in it. For example, a semantic frame may describe an event that occurs in a sentence, and the event's frame elements may be the people or objects that participate in the event. For a more concrete example, consider the sentence "They said they do not believe that the prison system is outdated". This sentence contains a Statement frame because a statement is made in it. One of the frame elements participating in the frame is the Speaker "they". From this frame, we would extract a feature pairing the frame together with its frame element to get the feature "Statement-Speaker-they". This feature indicates that the essay it occurs in might be a positive instance of the Writer Position error since it tells us the writer is attributing some statement being made to someone else. Hence, this feature along with several others like "Awareness-Cognizer-we all" are useful when constructing the lists of frame features for Writer Position's aggregated **f**rame features $Af+_i$ and $Af-_i$. Like every other aggregated feature, $Af+_i$ and $Af-_i$ are generated for every error e_i.

4.5 Score Prediction

Because essays containing thesis clarity errors tend to have lower thesis clarity scores than essays with fewer errors, we believe that thesis clarity scores can be predicted for essays by utilizing the same features we use for identifying thesis clarity errors. Because our score

prediction system uses the same feature types we use for thesis error identification, each essay's vector space representation remains unchanged. Only its label changes to one of the values in Table 4.1 in order to reflect its thesis clarity score. To make use of the fact that some pairs of scores are more similar than others (e.g., an essay with a score of 3.5 is more similar to an essay with a score of 4.0 than it is to one with a score of 1.0), we cast thesis clarity score prediction as a regression rather than classification task.

Treating thesis clarity score prediction as a regression problem removes our need for a classification threshold parameter like the one we use in the error identification problem, but if we use SVM^{light}'s regression option, it does not remove the need for tuning a regularization parameter, C, or a feature selection parameter, n.[7] We jointly tune these two parameters to optimize performance on held-out validation data by performing an exhaustive search in the parameter space.[8]

After we select the features, construct the essay instances, train a regressor on training set essays, and tune parameters on validation set essays, we can use the regressor to obtain thesis clarity scores on test set essays.

4.6 Evaluation

In this section, we evaluate our systems for error identification and scoring. All the results we report are obtained via five-fold cross-validation experiments. In each experiment, we

[7]Before tuning the feature selection parameter, we have to sort the list of n-gram features occurring the training set. To enable the use of information gain as the sorting criterion, we treat each distinct score as its own class.

[8]The absence of the classification threshold parameter and the fact that we do not need to train multiple learners, one for each score, make it feasible for us to do two things. First, we explore a wider range of values for the two parameters: we allow C to take any value from 10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6, or 10^7, and we allow n to take any value from 1000, 2000, 3000, 4000, 5000, or ALL. Second, we exhaustively explore the space defined by these parameters in order to obtain an exact solution to the parameter optimization problem.

use 3/5 of our labeled essays for model training, another 1/5 for parameter tuning, and the final 1/5 for testing.

4.6.1 Error Identification

Evaluation metrics. To evaluate our thesis clarity error type identification system, we compute precision, recall, micro F-score, and macro F-score, which are calculated as follows. Let tp_i be the number of test essays correctly labeled as positive by error e_i's binary classifier b_i; p_i be the total number of test essays labeled as positive by b_i; and g_i be the total number of test essays that belong to e_i according to the gold standard. Then, the precision (P_i), recall (R_i), and F-measure (F_i) for each classifier b_i and the macro F-score ($\hat{\mathrm{F}}$) of the combined system for one test fold are calculated by

$$\mathrm{P}_i = \frac{tp_i}{p_i}, \mathrm{R}_i = \frac{tp_i}{g_i}, \ \mathrm{F}_i = \frac{2\mathrm{P}_i\mathrm{R}_i}{\mathrm{P}_i + \mathrm{R}_i}, \hat{\mathrm{F}} = \frac{\sum_i \mathrm{F}_i}{5}.$$

However, the macro F-measure calculation can be seen as giving too much weight to the less frequent errors. To avoid this problem, we also calculate for each system the micro precision, recall, and F-measure (P, R, and F), where

$$\mathrm{P} = \frac{\sum_i tp_i}{\sum_i p_i}, \mathrm{R} = \frac{\sum_i tp_i}{\sum_i g_i}, \mathrm{F} = \frac{2\mathrm{PR}}{\mathrm{P} + \mathrm{R}}.$$

Since we perform five-fold cross-validation, each value we report for each of these measures is an average over its values for the five folds.[9]

Results and discussion. Results on error identification, expressed in terms of precision, recall, micro F-score, and macro F-score are shown in the first four columns of Table 4.5. Our **B**aseline system, which only uses word n-gram and random indexing features, seems to

[9]This averaging explains why the formula for F does not exactly hold in the Table 4.5 results.

Table 4.5. Five-fold cross-validation results for thesis clarity error identification and scoring.

		Error Identification				Scoring		
	System	P	R	F	$\hat{\text{F}}$	$S1$	$S2$	$S3$
1	**B**	24.8	44.7	31.1	24.0	.658	.517	.403
2	**Bm**	24.2	44.2	31.2	25.3	.654	.515	.402
3	**Bmk**	29.2	44.2	34.9	26.7	.663	.490	**.369**
4	**Bmkw**	28.5	49.6	35.5	31.4	**.651**	.484	.374
5	**Bmkwp**	**34.2**	49.6	40.4	34.6	.671	**.483**	.377
6	**Bmkwpf**	33.6	**54.4**	**41.4**	**37.3**	.672	.486	.382

perform uniformly poorly across both micro and macro F-scores (F and $\hat{\text{F}}$; see row 1). The per-class results show that, since micro F-score places more weight on the correct identification of the most frequent errors, the system's micro F-score (31.1%) is fairly close to the average of the scores obtained on the three most frequent error classes, CP, IPR, and R, and remains unaffected by very low F-scores on the two remaining infrequent classes.[10]

When we add the **m**isspelling feature to the baseline, resulting in the system called **Bm** (row 2), the micro F-score sees a very small, insignificant improvement.[11] What is pleasantly surprising, however, is that, even though the misspelling features were developed for the Confusing Phrasing error type, they actually have more of a positive impact on Missing Details and Writer Position, bumping their individual error F-scores up by about 5 and 3 percent respectively. This suggests that spelling difficulties may be correlated with these other essay-writing difficulties, despite their apparent unrelatedness. This effect is strong enough to generate the small, though insignificant, gain in macro F-score shown in the table.

When we add **k**eyword features to the system, micro F-score increases significantly by 3.7 points (row 3). The micro per-class results reveal that, as intended, keyword features

[10]Since parameters for optimizing micro F-score and macro F-score are selected independently, the per-class F-scores associated with micro F-score are different than those used for calculating macro F-score. Hence, when we discuss per-class changes influencing micro F-score, we refer to the former set, and otherwise we refer to the latter set.

[11]All significance tests are paired t-tests, with $p < 0.05$.

improve Incomplete Prompt Response and Relevance to Prompt's F-scores reveals that they do by 6.4 and 9.2 percentage points respectively. The macro F-scores reveal this too, though the macro F-score gains are 3.2 points and 11.5 points respectively. The macro F-score of the overall system would likely have improved more than shown in the table if the addition of keyword features did not simultaneously reduce Missing Details's score by several points.

While we hoped that adding aggregated **w**ord n-gram features to the system (row 4) would be able to improve performance on Confusing Phrasing due to the presence of phrases such as "in university be" in the error's $Aw+_i$ list, there turned out to be few such common phrases in the data set, so performance on this class remains mostly unchanged. This feature type does, however, result in major improvements to micro and macro performance on Missing Details and Writer Position, the other two classes this feature was designed to help. Indeed, the micro F-score versions of Missing Details and Writer Position improve by 15.3 and 10.8 percentage points respectively. Since these are minority classes, however, the large improvements result in only a small, insignificant improvement in the overall system's micro F-score. The macro F-score results for these classes, however, improve by 6.5% and 17.6% respectively, giving us a nearly 5-point, statistically significant bump in macro F-score after we add this feature.

Confusing Phrasing has up to now stubbornly resisted any improvement, even when we added features explicitly designed to help our system do better on this error type. When we add aggregated **p**art of speech n-gram features on top of the previous system, that changes dramatically. Adding these features makes both our system's F-scores on Confusing Phrasing shoot up almost 8%, resulting in a significant, nearly 4.9% improvement in overall micro F-score and a more modest but insignificant 3.2% improvement in macro F-score (row 5). The micro F-score improvement can also be partly attributed to a four point improvement in Incomplete Prompt Response's micro F-score. The 13.7% macro F-score improvement of the Missing Details error plays a larger role in the overall system's macro F-score improvement than Confusing Phrasing's improvement, however.

The improvement we see in micro F-score when we add aggregated **f**rame features (row 6) can be attributed almost solely to improvements in classification of the minority classes. This is surprising because, as we mentioned before, minority classes tend to have a much smaller impact on overall micro F-score. Furthermore, the overall micro F-score improvement occurs despite declines in the performances on two of the majority class errors. Missing Details and Writer Position's micro F-score performances increase by 19.1% and 13.4%. The latter is surprising only because of the magnitude of its improvement, as this feature type was explicitly intended to improve its performance. We did not expect this aggregated feature type to be especially useful for Missing Details error identification because very few of these types of features occur in its $Af+_i$ list, and there are none in its $Af-_i$ list. The few that are in the former list, however, occur fairly often and look like fairly good indicators of this error (both the examples "Event-Event-it" and "Categorization-Item-that" occur in the positive list, and both do seem vague, indicating more details are to be desired).

Overall, this system improves our baseline's macro F-score performance significantly by 13.3% and its micro F-score performance significantly by 10.3%. As we progressed, adding each new feature type to the baseline system, there was no definite and consistent pattern to how the precisions and recalls changed in order to produce the universal increases in the F-scores that we observed for each new system. Both just tended to jerkily progress upward as new feature types were added. This confirms our intuition about these features – namely that they do not all uniformly improve our performance in the same way. Some aim to improve precision by telling us when essays are less likely to be positive instances of an error class, such as any of the $Aw-_i$, $Ap-_i$, or $Af-_i$ features, and others aim to tell us when an essay is more likely to be a positive instance of an error.

4.6.2 Scoring

Scoring metrics. We design three evaluation metrics to measure the error of our thesis clarity scoring system. The $S1$ metric measures the frequency at which a system predicts

the wrong score out of the seven possible scores. Hence, a system that predicts the right score only 25% of the time would receive an $S1$ score of 0.75.

The $S2$ metric measures the average distance between the system's score and the actual score. This metric reflects the idea that a system that estimates scores close to the annotator-assigned scores should be preferred over a system whose estimations are further off, even if both systems estimate the correct score at the same frequency.

Finally, the $S3$ metric measures the average square of the distance between a system's thesis clarity score estimations and the annotator-assigned scores. The intuition behind this system is that not only should we prefer a system whose estimations are close to the annotator scores, but we should also prefer one whose estimations are not too frequently very far away from the annotator scores. These three scores are given by:

$$\frac{1}{N}\sum_{A_j \neq E'_j} 1, \quad \frac{1}{N}\sum_{i=1}^{N} |A_j - E_j|, \quad \frac{1}{N}\sum_{i=1}^{N} (A_j - E_j)^2$$

where A_j, E_j, and E'_j are the annotator assigned, system estimated, and rounded system estimated scores[12] respectively for essay j, and N is the number of essays.

Results and discussion. Results on scoring are shown in the last three columns of Table 4.5. We see that the thesis clarity score predicting variation of the **B**aseline system, which employs as features only word n-grams and random indexing features, predicts the wrong score 65.8% of the time. Its predicted score is on average 0.517 points off of the actual score, and the average squared distance between the predicted and actual scores is 0.403.

We observed earlier that a high number of misspellings may be positively correlated with one or more unrelated errors. Adding the **m**isspelling feature to the scoring systems,

[12]Since our regressor assigns each essay a real value rather than an actual valid thesis clarity score, it would be difficult to obtain a reasonable $S1$ score without rounding the system estimated score to one of the possible values. For that reason, we round the estimated score to the nearest of the seven scores the human annotators were permitted to assign (1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0) only when calculating $S1$.

Table 4.6. Distribution of predicted thesis clarity scores.

	$S1$ **(Bmkw)**			$S2$ **(Bmkwp)**			$S3$ **(Bmk)**		
Gold	.25	.50	.75	.25	.50	.75	.25	.50	.75
1.0	3.5	3.5	3.5	3.0	3.2	3.5	3.1	3.2	3.3
1.5	2.5	3.0	3.0	2.8	3.1	3.2	2.6	3.0	3.2
2.0	3.0	3.0	3.5	3.0	3.2	3.5	3.0	3.1	3.4
2.5	3.0	3.5	3.5	3.0	3.3	3.6	3.0	3.3	3.5
3.0	3.0	3.5	3.5	3.1	3.4	3.5	3.1	3.3	3.5
3.5	3.5	3.5	4.0	3.2	3.4	3.6	3.2	3.4	3.5
4.0	3.5	3.5	4.0	3.4	3.6	3.8	3.4	3.5	3.7

however, only yields minor, insignificant improvements to their performances under the three scoring metrics.

While adding **k**eyword features on top of this system does not improve the frequency with which the right score is predicted, it both tends to move the predictions closer to the actual thesis clarity score value (as evidenced by the significant improvement in $S2$) and ensures that predicted scores will not too often stray too far from the actual value (as evidenced by the significant improvement in $S3$). Overall, the scoring model employing the **Bmk** feature set performs significantly better than the **B**aseline scoring model with respect to two out of three scoring metrics.

The only remaining feature type whose addition yields a significant performance improvement is the aggregated **w**ord feature type, which improves system **Bmk**'s $S2$ score significantly while having an insignificant impact on the other S measures.

Neither of the remaining aggregative features yields any significant improvements in performance. This is a surprising finding since, up until we introduced aggregated **p**art-of-speech tag n-gram features into our regressor, each additional feature that helped with error classification made at least a small but positive contribution to at least two out of the three S scores. These aggregative features, which proved to be very powerful when assigning error labels, are not as useful for thesis clarity scoring.

To more closely examine the behavior of the best scoring systems, in Table 4.6 we chart the distributions of scores they predict for each gold standard score. As an example of how to read this table, consider the number 2.8 appearing in row 1.5 in the .25 column of the $S2$ (**Bmkwp**) region. This means that 25% of the time, when system **Bmkwp** (which obtains the best $S2$ score) is presented with a test essay having a gold standard score of 1.5, it predicts that the essay has a score less than or equal to 2.8 for the $S2$ measure.

From this table, we see that each of the best systems has a strong bias toward predicting more frequent scores as there are no numbers less than 3.0 in the 50% columns, and about 82.8% of all essays have gold standard scores of 3.0 or above. Nevertheless, no system relies entirely on bias, as evidenced by the fact that each column in the table has a tendency for its scores to ascend as the gold standard score increases, implying that the systems have some success at predicting lower scores for essays with lower gold standard scores.

Finally, we note that the difference in error weighting between the $S2$ and $S3$ scoring metrics appears to be having its desired effect, as there is a strong tendency for each entry in the $S3$ subtable to be less than or equal to its corresponding entry in the $S2$ subtable due to the greater penalty the $S3$ metric imposes for predictions that are very far away from the gold standard scores.

4.7 Summary

We examined the problem of modeling thesis clarity errors and scoring in student essays. In addition to developing these models, we proposed novel features for use in our thesis clarity error model and used these features, each of which was explicitly designed for one or more of the error types, to train our scoring model. We make our thesis clarity annotations publicly available in order to stimulate further research on this task.

CHAPTER 5

PROMPT ADHERENCE SCORING[1]

5.1 Introduction

Automated essay scoring, the task of employing computer technology to evaluate and score written text, is one of the most important educational applications of natural language processing (NLP) (see Shermis and Burstein (2003) and Shermis et al. (2010) for an overview of the state of the art in this task). A major weakness of many existing scoring engines such as the Intelligent Essay Assessor$^{\text{TM}}$(Landauer et al., 2003) is that they adopt a holistic scoring scheme, which summarizes the quality of an essay with a single score and thus provides very limited feedback to the writer. In particular, it is not clear which dimension of an essay (e.g., style, coherence, relevance) a score should be attributed to. Recent work addresses this problem by scoring a particular dimension of essay quality such as coherence (Miltsakaki and Kukich, 2004), technical errors, organization (Persing et al., 2010), and thesis clarity (Persing and Ng, 2013). Essay grading software that provides feedback along multiple dimensions of essay quality such as E-*rater*/Criterion (Attali and Burstein, 2006) has also begun to emerge.

Our goal in this chapter is to develop a computational model for scoring an essay along an under-investigated dimension — *prompt adherence*. Prompt adherence refers to how related an essay's content is to the prompt for which it was written. An essay with a high prompt adherence score consistently remains on the topic introduced by the prompt and is free of irrelevant digressions.

To our knowledge, little work has been done on scoring the prompt adherence of student essays since Higgins et al. (2004). Nevertheless, there are major differences between Higgins et al.'s work and our work with respect to both the way the task is formulated and the approach. Regarding task formulation, while Higgins et al. focus on classifying each *sentence*

[1]This chapter was previously published in Persing and Ng (2014).

as having either *good* or *bad* adherence to the prompt, we focus on assigning a prompt adherence score to the entire *essay*, allowing the score to range from one to four points at half-point increments. As far as the approach is concerned, Higgins et al. adopt a *knowledge-lean* approach to the task, where almost all of the features they employ are computed based on a word-based semantic similarity measure known as *Random Indexing* (Kanerva et al., 2000). On the other hand, we employ a large variety of features, including lexical and knowledge-based features that encode how well the concepts in an essay match those in the prompt, LDA-based features that provide semantic generalizations of lexical features, and "error type" features that encode different types of errors the writer made that are related to prompt adherence.

In sum, our contributions in this chapter are two-fold. First, we develop a scoring model for the prompt adherence dimension on student essays using a feature-rich approach. Second, in order to stimulate further research on this task, we make our data set consisting of prompt adherence annotations of 830 essays publicly available. Since progress in prompt adherence modeling is hindered in part by the lack of a publicly annotated corpus, we believe that our data set will be a valuable resource to the NLP community.

5.2 Corpus Information

We use as our corpus the 4.5 million word International Corpus of Learner English (ICLE) (Granger et al., 2009) described in Chapter 3.2. We select a subset consisting of 830 argumentative essays from the ICLE to annotate for training and testing of our essay prompt adherence scoring system. Table 3.1 shows several of the 13 topics selected for annotation. Fifteen native languages are represented in the set of annotated essays.

Table 5.1. Description of each prompt adherence score.

Score	Description of Prompt Adherence
4	essay fully addresses the prompt and **consistently stays on topic**
3	essay mostly addresses the prompt or **occasionally wanders off topic**
2	essay does not fully address the prompt or **consistently wanders off topic**
1	essay does not address the prompt at all or is **completely off topic**

5.3 Corpus Annotation

We ask human annotators to score each of the 830 argumentative essays along the prompt adherence dimension. Our annotators were selected from over 30 applicants who were familiarized with the scoring rubric and given sample essays to score. The six who were most consistent with the expected scores were given additional essays to annotate. Annotators evaluated how well each essay adheres to its prompt using a numerical score from one to four at half-point increments (see Table 5.1 for a description of each score). This contrasts with previous work on prompt adherence essay scoring, where the corpus is annotated with a binary decision (i.e., *good* or *bad*) (e.g., Higgins et al. (2004; 2006), Louis and Higgins (2010)). Hence, our annotation scheme not only provides a finer-grained distinction of prompt adherence (which can be important in practice), but also makes the prediction task more challenging.

To ensure consistency in annotation, we randomly select 707 essays to have graded by multiple annotators. Analysis reveals that the Pearson's correlation coefficient computed over these doubly annotated essays is 0.243. Though annotators exactly agree on the prompt adherence score of an essay only 38% of the time, the scores they apply fall within 0.5 points in 66% of essays and within 1.0 point in 89% of essays. For the sake of our experiments, whenever annotators disagree on an essay's prompt adherence score, we assign the essay the average of all annotations rounded to the nearest half point. Table 5.2 shows the number of essays that receive each of the seven scores for prompt adherence.

Table 5.2. Distribution of prompt adherence scores.

score	1.0	1.5	2.0	2.5	3.0	3.5	4.0
essays	0	0	8	44	105	230	443

5.4 Score Prediction

In this section, we describe in detail our system for predicting essays' prompt adherence scores.

5.4.1 Model Training and Application

We cast the problem of predicting an essay's prompt adherence score as 13 regression problems, one for each prompt. Each essay is represented as an instance whose label is the essay's true score (one of the values shown in Table 5.2) with up to seven types of features including baseline (Section 4.2) and six other feature types proposed by us (Section 4.3). Our regressors may assign an essay any score in the range of $1.0-4.0$.

Using regression captures the fact that some pairs of scores are more similar than others (e.g., an essay with a prompt adherence score of 3.5 is more similar to an essay with a score of 4.0 than it is to one with a score of 1.0). A classification system, by contrast, may sometimes believe that the scores 1.0 and 4.0 are most likely for a particular essay, even though these scores are at opposite ends of the score range.

Using a different regressor for each prompt captures the fact that it may be easier for an essay to adhere to some prompts than to others, and common problems students have writing essays for one prompt may not apply to essays written in response to another prompt. For example, in essays written in response to the prompt "Marx once said that religion was the opium of the masses. If he was alive at the end of the 20th century, he would replace religion with television," students sometimes write essays about all the evils of television, forgetting that their essay is only supposed to be about whether it is "the opium of the

masses". Students are less likely to make an analogous mistake when writing for the prompt "Crime does not pay."

After creating training instances for prompt p_i, we train a linear regressor, r_i, with regularization parameter c_i for scoring test essays written in response to p_i using the linear SVM regressor implemented in the LIBSVM software package (Chang and Lin, 2001). All SVM-specific learning parameters are set to their default values except c_i, which we tune to maximize performance on held-out validation data.

After training the classifiers, we use them to classify the test set essays. The test instances are created in the same way as the training instances.

5.4.2 Baseline Features

Our baseline system for score prediction employs various features based on Random Indexing.

1. Random Indexing Random Indexing (RI) is "an efficient, scalable and incremental alternative" (Sahlgren, 2005) to Latent Semantic Indexing (Deerwester et al., 1990; Landauer and Dumais, 1997) which allows us to automatically generate a semantic similarity measure between any two words. We train our RI model on over 30 million words of the English Gigaword corpus (Parker et al., 2009) using the S-Space package (Jurgens and Stevens, 2010). We expect that features based on RI will be useful for prompt adherence scoring because they may help us find text related to the prompt even if some of its concepts have have been rephrased (e.g., an essay may talk about "jail" rather than "prison", which is mentioned in one of the prompts), and because they have already proven useful for the related task of determining which sentences in an essay are related to the prompt (Higgins et al., 2004).

For each essay, we therefore attempt to adapt the RI features used by Higgins et al. (2004) to our problem of prompt adherence scoring. We do this by generating one feature encoding

the entire essay's similarity to the prompt, another encoding the essay's highest individual sentence's similarity to the prompt, a third encoding the highest entire essay similarity to one of the prompt sentences, another encoding the highest individual sentence similarity to an individual prompt sentence, and finally one encoding the entire essay's similarity to a manually rewritten version of the prompt that excludes extraneous material (such as "In his novel Animal Farm, George Orwell wrote," which is introductory material from the fourth prompt in Table 3.1). Our RI feature set necessarily excludes those features from Higgins et al. that are not easily translatable to our problem since we are concerned with an entire essay's adherence to its prompt rather than with each of its sentences' relatedness to the prompt. Since RI does not provide a straightforward way to measure similarity between groups of words such as sentences or essays, we use Higgins and Burstein's (2007) method to generate these features.

5.4.3 Novel Features

Next, we introduce six types of novel features.

2. N-grams As our first novel feature, we use the 10,000 most important lemmatized unigram, bigram, and trigram features that occur in the essay. N-grams can be useful for prompt adherence scoring because they can capture useful words and phrases related to a prompt. For example, words and phrases like "university degree", "student", and "real world" are relevant to the second prompt in Table 3.1, so it is more likely that an essay adheres to the prompt if they appear in the essay.

We determine the "most important" n-gram features using information gain computed over the training data (Yang and Pedersen, 1997). Since the essays vary greatly in length, we normalize each essay's set of n-gram features to unit length.

3. Thesis Clarity Keywords Our next set of features consists of the keyword features we introduced in our previous work on essay thesis clarity scoring (Persing and Ng, 2013). Below we give an overview of these keyword features and motivate why they are potentially useful for prompt adherence scoring.

The keyword features were formed by first examining the 13 essay prompts, splitting each into its component pieces. As an example of what is meant by a "component piece", consider the first prompt in Table 1. The components of this prompt would be "Most university degrees are theoretical", "Most university degrees do not prepare students for the real world", and "Most university degrees are of very little value."

Then the most important (primary) and second most important (secondary) words were selected from each prompt component, where a word was considered "important" if it would be a good word for a student to use when stating her thesis about the prompt. So since the lemmatized version of the third component of the third prompt in Table 3.1 is "it should rehabilitate they", "rehabilitate" was selected as a primary keyword and "society" as a secondary keyword.

Features are then computed based on these keywords. For instance, one thesis clarity keyword feature is computed as follows. The RI similarity measure is first taken between the essay and each group of the prompt's primary keywords. The feature then gets assigned the lowest of these values. If this feature has a low value, that suggests that the student ignored the prompt component from which the value came when writing the essay.

To compute another of the thesis clarity keyword features, the numbers of combined primary and secondary keywords the essay contains from each component of its prompt are counted. These numbers are then divided by the total count of primary and secondary features in their respective components. The greatest of the fractions generated in this way

is encoded as a feature because if it has a low value, that indicates the essay's thesis may not be very relevant to the prompt.[2]

4. Prompt Adherence Keywords The thesis clarity keyword features described above were intended for the task of determining how clear an essay's thesis is, but since our goal is instead to determine how well an essay adheres to its prompt, it makes sense to adapt keyword features to our task rather than to adopt keyword features exactly as they have been used before. For this reason, we construct a new list of keywords for each prompt component, though since prompt adherence is more concerned with what the student says about the topics than it is with whether or not what she says about them is stated clearly, our keyword lists look a little different than the ones discussed above. For an example, we earlier alluded to the problem of students merely discussing all the evils of television for the prompt "Marx once said that religion was the opium of the masses. If he was alive at the end of the 20th century, he would replace religion with television." Since the question suggests that students discuss whether television is analogous to religion in this way, our set of prompt adherence keywords for this prompt contains the word "religion" while the previously discussed keyword sets do not. This is because a thesis like "Television is bad" can be stated very clearly without making any reference to religion at all, and so an essay with a thesis like this can potentially have a very high thesis clarity score. It should not, however, have a very high prompt adherence score, as the prompt asked the student to discuss whether television is like religion in a particular way, so religion should be at least briefly addressed for an essay to be awarded a high prompt adherence score.

Additionally, our prompt adherence keyword sets do not adopt the notions of primary and secondary groups of keywords for each prompt component, instead collecting all the

[2]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for the complete list of thesis clarity keyword features.

keywords for a component into one set because "secondary" keywords tend to be things that are important when we are concerned with what a student is saying about the topic rather than just how clearly she said it.

We form two types of features from prompt adherence keywords. While both types of features measure how much each prompt component was discussed in an essay, they differ in how they encode the information. To obtain feature values of the first type, we take the RI similarities between the whole essay and each set of prompt adherence keywords from the prompt's components. This results in one to three features, as some prompts have one component while others have up to three.

We obtain feature values of the second type as follows. For each component, we count the number of prompt adherence keywords the essay contains. We divide this number by the number of prompt adherence keywords we identified from the component. This results in one to three features since a prompt has one to three components.

5. LDA Topics A problem with the features we have introduced up to this point is that they have trouble identifying topics that are not mentioned in the prompt, but are nevertheless related to the prompt. These topics should not diminish the essay's prompt adherence score because they are at least related to prompt concepts. For example, consider the prompt "All armies should consist entirely of professional soldiers: there is no value in a system of military service." An essay containing words like "peace", "patriotism", or "training" are probably not digressions from the prompt, and therefore should not be penalized for discussing these topics. But the various measures of keyword similarities described above will at best not notice that anything related to the prompt is being discussed, and at worst, this might have effects like lowering some of the RI similarity scores, thereby probably lowering the prompt adherence score the regressor assigns to the essay. While n-gram features do not have exactly the same problem, they would still only notice that these example words are

related to the prompt if multiple essays use the same words to discuss these concepts. For this reason, we introduce Latent Dirichlet Allocation (LDA) (Blei et al., 2003) features.

In order to construct our LDA features, we first collect all essays written in response to each prompt into its own set. Note that this feature type exploits unlabeled data: it includes all essays in the ICLE responding to our prompts, not just those in our smaller annotated 830 essay dataset. We then use the MALLET (McCallum, 2002) implementation of LDA to build a topic model of 1,000 topics around each of these sets of essays. This results in what we can think of as a soft clustering of words into 1,000 sets for each prompt, where each set of words represents one of the topics LDA identified being discussed in the essays for that prompt. So for example, the five most important words in the most frequently discussed topic for the military prompt we mentioned above are "man", "military", "service", "pay", and "war".

We also use the MALLET-generated topic model to tell us how much of each essay is spent discussing each of the 1,000 topics. The model might tell us, for example, that a particular essay written on the military prompt spends 35% of the time discussing the "man", "military", "service", "pay", and "war" topic and 65% of the time discussing a topic whose most important words are "fully", "count", "ordinary", "czech", and "day". Since the latter topic is discussed so much in the essay and does not appear to have much to do with the military prompt, this essay should probably get a bad prompt adherence score. We construct 1,000 features from this topic model, one for each topic. Each feature's value is obtained by using the topic model to tell us how much of the essay was spent discussing the feature's corresponding topic. From these features, our regressor should be able to learn which topics are important to a good prompt adherent essay.

6. Manually Annotated LDA Topics A weakness of the LDA topics feature type is that it may result in a regressor that has trouble distinguishing between an infrequent topic

that is adherent to the prompt and one that just represents an irrelevant digression. This is because an infrequent topic may not appear in the training set often enough for the regressor to make this judgment. We introduce the manually annotated LDA topics feature type to address this problem.

In order to construct manually annotated LDA topic features, we first build 13 topic models, one for each prompt, just as described in the section on LDA topic features. Rather than requesting models of 1,000 topics, however, we request models of only 100 topics[3]. We then go through all 13 lists of 100 topics as represented by their top ten words, manually annotating each topic with a number from 0 to 5 representing how likely it is that the topic is adherent to the prompt. A topic labeled 5 is very likely to be related to the prompt, where a topic labeled 0 appears totally unrelated.

Using these annotations alongside the topic distribution for each essay that the topic models provide us, we construct ten features. The first five features encode the sum of the contributions to an essay of topics annotated with a number ≥ 1, the sum of the contributions to an essay of topics annotated with a number ≥ 2, and so on up to 5.

The next five features are similar to the last, with one feature taking on the sum of the contributions to an essay of topics annotated with the number 0, another feature taking on the sum of the contributions to an essay of topics annotated with the number 1, and so on up to 4. We do not include a feature for topics annotated with the number 5 because it would always have the same value as the feature for topics ≥ 5.

Features like these should give the regressor a better idea how much of an essay is composed of prompt-related arguments and discussion and how much of it is irrelevant to the prompt, even if some of the topics occurring in it are too infrequent to judge just from training data.

[3]We use 100 topics for each prompt in the manually annotated version of LDA features rather than the 1,000 topics we use in the regular version of LDA features because 1,300 topics are not too costly to annotate, but manually annotating 13,000 topics would take too much time.

7. Predicted Thesis Clarity Errors In our previous work on essay thesis clarity scoring (Persing and Ng, 2013), we identified five classes of errors that detract from the clarity of an essay's thesis:

Confusing Phrasing. The thesis is phrased oddly, making it hard to understand the writer's point.

Incomplete Prompt Response. The thesis leaves some part of a multi-part prompt unaddressed.

Relevance to Prompt. The apparent thesis's weak relation to the prompt causes confusion.

Missing Details. The thesis leaves out an important detail needed to understand the writer's point.

Writer Position. The thesis describes a position on the topic without making it clear that this is the position the writer supports.

We hypothesize that these errors, though originally intended for thesis clarity scoring, could be useful for prompt adherence scoring as well. For instance, an essay that has a Relevance to Prompt error or an Incomplete Prompt Response error should intuitively receive a low prompt adherence score. For this reason, we introduce features based on these errors to our feature set for prompt adherence scoring[4].

While each of the essays in our data set was previously annotated with these thesis clarity errors, in a realistic setting a prompt adherence scoring system will not have access to these manual error labels. As a result, we first need to predict which of these errors is present in each essay. To do this, we train five maximum entropy classifiers for each prompt, one for each of the five thesis clarity errors, using MALLET's (McCallum, 2002) implementation of maximum entropy classification. Instances are presented to classifier for prompt p for error e in the following way. If a training essay is written in response to p, it will be used

[4]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for the complete list of error annotations.

to generate a training instance whose label is 1 if e was annotated for it or 0 otherwise. Since error prediction and prompt adherence scoring are related problems, the features we associate with this instance are features 1−6 which we have described earlier in this section. The classifier is then used to generate probabilities telling us how likely it is that each test essay has error e.

Then, when training our regressor for prompt adherence scoring, we add the following features to our instances. We add a binary feature indicating the presence or absence of each error. Or in the case of test essays, the feature takes on a real value from 0 to 1 indicating how likely the classifier thought it was that the essay had each of the errors. This results in five additional features, one for each error.

5.5 Evaluation

In this section, we evaluate our system for prompt adherence scoring. All the results we report are obtained via five-fold cross-validation experiments. In each experiment, we use $\frac{3}{5}$ of our labeled essays for model training, another $\frac{1}{5}$ for parameter tuning, and the final $\frac{1}{5}$ for testing.

5.5.1 Experimental Setup

Scoring Metrics

We employ four evaluation metrics. As we will see below, $S1$, $S2$, and $S3$ are *error* metrics, so lower scores imply better performance. In contrast, PC is a *correlation* metric, so higher correlation implies better performance.

The simplest metric, $S1$, measures the frequency at which a system predicts the wrong score out of the seven possible scores. Hence, a system that predicts the right score only 25% of the time would receive an $S1$ score of 0.75.

The $S2$ metric measures the average distance between a system's score and the actual score. This metric reflects the idea that a system that predicts scores close to the annotator-assigned scores should be preferred over a system whose predictions are further off, even if both systems estimate the correct score at the same frequency.

The $S3$ metric measures the average square of the distance between a system's score predictions and the annotator-assigned scores. The intuition behind this system is that not only should we prefer a system whose predictions are close to the annotator scores, but we should also prefer one whose predictions are not too frequently very far away from the annotator scores. These three scores are given by:

$$\frac{1}{N}\sum_{A_j \neq E'_j} 1, \quad \frac{1}{N}\sum_{i=1}^{N} |A_j - E_j|, \quad \frac{1}{N}\sum_{i=1}^{N} (A_j - E_j)^2$$

where A_j, E_j, and E'_j are the annotator assigned, system predicted, and rounded system predicted scores[5] respectively for essay j, and N is the number of essays.

The last metric, PC, computes Pearson's correlation coefficient between a system's predicted scores and the annotator-assigned scores. PC ranges from -1 to 1. A positive (negative) PC implies that the two sets of predictions are positively (negatively) correlated.

Parameter Tuning

As mentioned earlier, for each prompt p_i, we train a linear regressor r_i using LIBSVM with regularization parameter c_i. To optimize our system's performance on the three error measures described previously, we use held-out validation data to independently tune each

[5]Since our regressor assigns each essay a real value rather than an actual valid score, it would be difficult to obtain a reasonable $S1$ score without rounding the system estimated score to one of the possible values. For that reason, we round the estimated score to the nearest of the seven scores the human annotators were permitted to assign (1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0) only when calculating $S1$. For other scoring metrics, we only round the predictions to 1.0 or 4.0 if they fall outside the $1.0-4.0$ range.

Table 5.3. Five-fold cross-validation results for prompt adherence scoring.

System	$S1$	$S2$	$S3$	PC
Baseline	.517	.368	.234	.233
Our System	.488	.348	.197	.360

of the c_i values[6]. Note that each of the c_i values can be tuned independently because a c_i value that is optimal for predicting scores for p_i essays with respect to any of the error performance measures is necessarily also the optimal c_i when measuring that error on essays from all prompts. However, this is not case with Pearson's correlation coefficient, as the PC value for essays from all 13 prompts cannot be simplified as a weighted sum of the PC values obtained on each individual prompt. In order to obtain an optimal result as measured by PC, we jointly tune the c_i parameters to optimize the PC value achieved by our system on the same held-out validation data. However, an exact solution to this optimization problem is computationally expensive, as there are too many (7^{13}) possible combinations of c values to exhaustively search. Consequently, we find a local maximum by employing the simulated annealing algorithm (Kirkpatrick et al., 1983), altering one c_i value at a time to optimize PC while holding the remaining parameters fixed.

5.5.2 Results and Discussion

Five-fold cross-validation results on prompt adherence score prediction are shown in Table 5.3. On the first line, this table shows that our baseline system, which recall uses only various RI features, predicts the wrong score 51.7% of the time. Its predictions are off by an average of .368 points, and the average squared distance between its predicted score and the actual score is .234. In addition, its predicted scores and the actual scores have a Pearson correlation coefficient of 0.233.

[6]For parameter tuning, we employ the following values. c_i may be assigned any of the values 10^0 10^1, 10^2, 10^3, 10^4, 10^5, or 10^6.

The results from our system, which uses all seven feature types described in Section 4, are shown in row 2 of the table. Our system obtains $S1$, $S2$, $S3$, and PC scores of .488, .348, .197, and .360 respectively, yielding a significant improvement over the baseline with respect to $S2$, $S3$, and PC with $p < 0.05$, $p < 0.01$, and $p < 0.06$ respectively[7]. While our system yields improvements by all four measures, its improvement over the baseline $S1$ score is not significant. These results mean that the greatest improvements our system makes are that it ensures that our score predictions are not too often very far away from an essay's actual score, as making such predictions would tend to drive up $S3$, yielding a relative error reduction in $S3$ of 15.8%, and it also ensures a better correlation between predicted and actual scores, thus yielding the 16.6% improvement in PC.[8] It also gives more modest improvements in how frequently exactly the right score is predicted ($S1$) and is better at predicting scores closer to the actual scores ($S2$).

5.5.3 Feature Ablation

To gain insight into how much impact each of the feature types has on our system, we perform feature ablation experiments in which we remove the feature types from our system one-by-one.

Results of the ablation experiments when performed using the four scoring metrics are shown in Table 5.4. The top line of each subtable shows what our system's score would be if we removed just one of the feature types from our system. So to see how our system performs by the $S1$ metric if we remove only predicted thesis clarity error features, we would look at the first row of results of Table 5.4(a) under the column headed by the number 7 since predicted thesis clarity errors are the seventh feature type introduced in Section 4.

[7]All significance tests are paired t-tests.

[8]These numbers are calculated $\frac{B-O}{B-P}$ where B is the baseline system's score, O is our system's score, and P is a perfect score. Perfect scores for error measures and PC are 0 and 1 respectively.

The number here tells us that our system's $S1$ score without this feature type is .502. Since Table 5.3 shows that when our system includes this feature type (along with all the other feature types), it obtains an $S1$ score of .488, this feature type's removal costs our system .014 $S1$ points, and thus its inclusion has a beneficial effect on the $S1$ score.

From row 1 of Table 5.4(a), we can see that removing feature 4 yields a system with the best $S1$ score in the presence of the other feature types in this row. For this reason, we permanently remove feature 4 from the system before we generate the results on line 2. Thus, we can see what happens when we remove both feature 4 and feature 5 by looking at the second entry in row 2. And since removing feature 6 harms performance least in the presence of row 2's other feature types, we permanently remove both 4 and 6 from our feature set when we generate the third row of results. We iteratively remove the feature type that yields a system with the best performance in this way until we get to the last line, where only one feature type is used to generate each result.

Since the feature type whose removal yields the best system is always the rightmost entry in a line, the order of column headings indicates the relative importance of the feature types, with the leftmost feature types being most important to performance and the rightmost feature types being least important in the presence of the other feature types. This being the case, it is interesting to note that while the relative importance of different feature types does not remain exactly the same if we measure performance in different ways, we can see that some feature types tend to be more important than others in a majority of the four scoring metrics. Features 2 (n-grams), 3 (thesis clarity keywords), and 6 (manually annotated LDA topics) tend to be the most important feature types, as they tend to be the last feature types removed in the ablation subtables. Features 1 (RI) and 5 (LDA topics) are of middling importance, with neither ever being removed first or last, and each tending to have a moderate effect on performance. Finally, while features 4 (prompt adherence keywords) and 7 (predicted thesis clarity errors) may by themselves provide useful information to our

Table 5.4. In each subtable, the first row shows how our system would perform if each feature type was removed. We remove the least important feature type, and show in the next row how the adjusted system would perform without each remaining type. For brevity, a feature type is referred to by its feature number: (1) RI; (2) n-grams; (3) thesis clarity keywords; (4) prompt adherence keywords; (5) LDA topics; (6) manually annotated LDA topics; and (7) predicted thesis clarity errors.

(a) Results using the $S1$ metric

3	5	1	7	2	6	4
.527	.502	.512	.502	.511	.500	.488
.527	.502	.512	.501	.513	.500	
.525	.508	.505	.505	.504		
.513	.527	.520	.513			
.523	.520	.506				
.541	.527					

(b) Results using the $S2$ metric

2	6	3	1	4	5	7
.356	.350	.348	.350	.349	.348	.348
.351	.349	.348	.348	.348	.347	
.351	.349	.348	.348	.347		
.350	.349	.348	.348			
.358	.351	.349				
.362	.352					

(c) Results using the $S3$ metric

2	6	1	5	4	7	3
.221	.201	.197	.197	.197	.197	.196
.215	.201	.197	.196	.196	.196	
.212	.203	.199	.197	.196		
.212	.203	.199	.197			
.212	.203	.199				
.223	.204					

(d) Results using the PC metric

6	3	2	1	7	5	4
.326	.332	.303	.344	.348	.348	.361
.326	.332	.304	.343	.348	.348	
.324	.337	.292	.345	.352		
.322	.337	.297	.346			
.316	.321	.323				
.218	.325					

system, in the presence of the other feature types they tend to be the least important to performance as they are often the first feature types removed.

While there is a tendency for some feature types to always be important (or unimportant) regardless of which scoring metric is used to measure performance, the relative importance of different feature types does not always remain consistent if we measure performance in different ways. For example, while we identified feature 3 (thesis clarity keywords) as one of the most important feature types generally due to its tendency to have a large beneficial impact on performance, when we are measuring performance using $S3$, it is the least useful feature type. Furthermore, its removal increases the $S3$ score by a small amount, meaning that its inclusion actually makes our system perform worse with respect to $S3$. Though

feature 3 is an extreme example, all feature types fluctuate in importance, as we see when we compare their orders of removal among the four ablation subtables. Hence, it is important to know how performance is measured when building a system for scoring prompt adherence.

Feature 3 is not the only feature type whose removal sometimes has a beneficial impact on performance. As we can see in Table 5.4(b), the removal of features 4, 5, and 7 improves our system's $S2$ score by .001 points. The same effect occurs in Table 5.4(c) when we remove features 4, 7, and 3. These examples illustrate that under some scoring metrics, the inclusion of some feature types is actively harmful to performance. Fortunately, this effect does not occur in any other cases than the two listed above, as most feature types usually have a beneficial or at least neutral impact on our system's performance.

For those feature types whose effect on performance is neutral in the first lines of ablation results (feature 4 in $S1$, features 3, 5, and 7 in $S2$, and features 1, 4, 5, and 7 in $S3$), it is important to note that their neutrality does not mean that they are unimportant. It merely means that they do not improve performance in the presence of other feature types. We can see this is the case by noting that they are not all the least important feature types in their respective subtables as indicated by column order. For example, by the time feature 1 gets permanently removed in Table 5.4(c), its removal harms performance by .002 $S3$ points.

5.5.4 Analysis of Predicted Scores

To more closely examine the behavior of our system, in Table 5.5 we chart the distributions of scores it predicts for essays having each gold standard score. As an example of how to read this table, consider the number 3.06 appearing in row 2.0 in the .25 column of the $S3$ region. This means that 25% of the time, when our system with parameters tuned for optimizing $S3$ is presented with a test essay having a gold standard score of 2.0, it predicts that the essay has a score less than or equal to 3.06.

From this table, we see that our system has a strong bias toward predicting more frequent scores as there are no numbers less than 3.0 in the table, and about 93.7% of all essays have

Table 5.5. Distribution of predicted prompt adherence scores.

	$S1$			$S2$			$S3$			PC		
Gold	.25	.50	.75	.25	.50	.75	.25	.50	.75	.25	.50	.75
2.0	3.35	3.56	3.79	3.40	3.52	3.73	3.06	3.37	3.64	3.06	3.37	3.64
2.5	3.43	3.63	3.80	3.25	3.52	3.79	3.24	3.45	3.67	3.24	3.46	3.73
3.0	3.64	3.78	3.85	3.56	3.70	3.90	3.52	3.65	3.74	3.52	3.66	3.79
3.5	3.73	3.81	3.88	3.63	3.78	3.90	3.59	3.70	3.81	3.60	3.74	3.85
4.0	3.76	3.84	3.88	3.70	3.83	3.90	3.63	3.75	3.84	3.66	3.78	3.88

gold standard scores of 3.0 or above. Nevertheless, our system does not rely entirely on bias, as evidenced by the fact that each column in the table has a tendency for its scores to ascend as the gold standard score increases, implying that our system has some success at predicting lower scores for essays with lower gold standard prompt adherence scores.

Another interesting point to note about this table is that the difference in error weighting between the $S2$ and $S3$ scoring metrics appears to be having its desired effect, as every entry in the $S3$ subtable is less than its corresponding entry in the $S2$ subtable due to the greater penalty the $S3$ metric imposes for predictions that are very far away from the gold standard scores.

5.6 Summary

We proposed a feature-rich approach to the under-investigated problem of predicting essay-level prompt adherence scores on student essays. In an evaluation on 830 argumentative essays selected from the ICLE corpus, our system significantly outperformed a Random Indexing based baseline by several evaluation metrics. To stimulate further research on this task, we make all our annotations, including our prompt adherence scores, the LDA topic annotations, and the error annotations publicly available.

CHAPTER 6

ARGUMENT QUALITY SCORING[1]

6.1 Introduction

Automated essay scoring, the task of employing computer technology to evaluate and score written text, is one of the most important educational applications of natural language processing (NLP) (see Shermis and Burstein (2003) and Shermis et al. (2010) for an overview of the state of the art in this task). A major weakness of many existing scoring engines such as the Intelligent Essay Assessor^{TM}(Landauer et al., 2003) is that they adopt a holistic scoring scheme, which summarizes the quality of an essay with a single score and thus provides very limited feedback to the writer. In particular, it is not clear which dimension of an essay (e.g., style, coherence, relevance) a score should be attributed to. Recent work addresses this problem by scoring a particular dimension of essay quality such as coherence (Miltsakaki and Kukich, 2004), technical errors, relevance to prompt (Higgins et al., 2004; Persing and Ng, 2014), organization (Persing et al., 2010), and thesis clarity (Persing and Ng, 2013). Essay grading software that provides feedback along multiple dimensions of essay quality such as E-*rater*/Criterion (Attali and Burstein, 2006) has also begun to emerge.

Our goal in this chapter is to develop a computational model for scoring the essay dimension of *argument quality*, which is arguably the most important aspect of argumentative essays. Argument quality refers to the quality of the argument an essay makes for its thesis. An essay with a high argument quality score presents a sound argument for its thesis and would convince most readers. While there has been work on *designing* argument schemes (e.g., Burstein et al. (2003), Song et al. (2014), Stab and Gurevych (2014a)) for *annotating* arguments manually (e.g., Song et al. (2014), Stab and Gurevych (2014b)) and automatically (e.g., Falakmasir et al. (2014), Song et al. (2014)) in student essays, little work has been

[1]This chapter was previously published in Persing and Ng (2015).

done on scoring the argument quality of student essays. It is worth mentioning that some work has investigated the use of automatically determined argument labels for heuristic (Ong et al., 2014) and learning-based (Song et al., 2014) essay scoring, but their focus is *holistic* essay scoring, not argument quality essay scoring.

In sum, our contributions in this chapter are two-fold. First, we develop a scoring model for the argument quality dimension on student essays using a *feature-rich* approach. Second, in order to stimulate further research on this task, we make our data set consisting of argument quality annotations of 1000 essays publicly available. Since progress in argument quality modeling is hindered in part by the lack of a publicly annotated corpus, we believe that our data set will be a valuable resource to the NLP community.

6.2 Corpus Information

We use as our corpus the 4.5 million word International Corpus of Learner English (ICLE) (Granger et al., 2009) described in Chapter 3.2. 91% of the ICLE texts are written in response to prompts that trigger argumentative essays. We select 10 such prompts, and from the subset of argumentative essays written in response to them, we select 1000 essays to annotate for training and testing of our essay argument quality scoring system. Table 3.1 shows several of the 10 topics selected for annotation. Fifteen native languages are represented in the set of annotated essays.

6.3 Corpus Annotation

We ask human annotators to score each of the 1000 argumentative essays along the argument quality dimension. Our annotators were selected from over 30 applicants who were familiarized with the scoring rubric and given sample essays to score. The six who were most consistent with the expected scores were given additional essays to annotate. Annotators evaluated the argument quality of each essay using a numerical score from one to four at

Table 6.1. Description of each argument quality score.

Score	Description of Argument Strength
4	essay makes a **strong argument** for its thesis and would convince most readers
3	essay makes a **decent argument** for its thesis and could convince some readers
2	essay makes a **weak argument** for its thesis or sometimes even **argues against it**
1	essay **does not make an argument** or it is often **unclear what the argument is**

half-point increments (see Table 6.1 for a description of each score).[2] This contrasts with previous work on essay scoring, where the corpus is annotated with a binary decision (i.e., *good* or *bad*) for a given scoring dimension (e.g., Higgins et al. (2004)). Hence, our annotation scheme not only provides a finer-grained distinction of argument quality (which can be important in practice), but also makes the prediction task more challenging.

To ensure consistency in annotation, we randomly select 846 essays to have graded by multiple annotators. Though annotators exactly agree on the argument quality score of an essay only 26% of the time, the scores they apply fall within 0.5 points in 67% of essays and within 1.0 point in 89% of essays. For the sake of our experiments, whenever the two annotators disagree on an essay's argument quality score, we assign the essay the average the two scores rounded down to the nearest half point. Table 6.2 shows the number of essays that receive each of the seven scores for argument quality.

Table 6.2. Distribution of argument quality scores.

score	1.0	1.5	2.0	2.5	3.0	3.5	4.0
essays	2	21	116	342	372	132	15

[2]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for the complete list of argument quality annotations.

6.4 Score Prediction

We cast the task of predicting an essay's argument quality score as a regression problem. Using regression captures the fact that some pairs of scores are more similar than others (e.g., an essay with an argument quality score of 2.5 is more similar to an essay with a score of 3.0 than it is to one with a score of 1.0). A classification system, by contrast, may sometimes believe that the scores 1.0 and 4.0 are most likely for a particular essay, even though these scores are at opposite ends of the score range. In the rest of this section, we describe how we train and apply our regressor.

Training the regressor. Each essay in the training set is represented as an instance whose label is the essay's gold score (one of the values shown in Table 6.2), with a set of baseline features (Section 5) and up to seven other feature types we propose (Section 6). After creating training instances, we train a linear regressor with regularization parameter c for scoring test essays using the linear SVM regressor implemented in the LIBSVM software package (Chang and Lin, 2001). All SVM-specific learning parameters are set to their default values except c, which we tune to maximize performance on held-out validation data.[3]

Applying the regressor. After training the regressor, we use it to score the test set essays. Test instances are created in the same way as the training instances. The regressor may assign an essay any score in the range of 1.0−4.0.

6.5 Baseline Systems

In this section, we describe two baseline systems for predicting essays' argument quality scores.

[3]For parameter tuning, we employ the following c values: 10^0 10^1, 10^2, 10^3, 10^4, 10^5, or 10^6.

6.5.1 Baseline 1: Most Frequent Baseline

Since there is no existing system specifically for scoring argument quality, we begin by designing a simple baseline. When examining the score distribution shown in Table 6.2, we notice that, while there exist at least a few essays having each of the seven possible scores, the essays are most densely clustered around scores 2.5 and 3.0. A system that always predicts one of these two scores will very frequently be right. For this reason, we develop a *most frequent* baseline. Given a training set, Baseline 1 counts the number of essays assigned to each of the seven scores. From these counts, it determines which score is most frequent and assigns this most frequent score to each test essay.

6.5.2 Baseline 2: Learning-based Ong et al.

Our second baseline is a learning-based version of Ong et al.'s (2014) system. Recall from the introduction that Ong et al. presented a rule-based approach to predict the holistic score of an argumentative essay. Their approach was composed of two steps. First, they constructed eight heuristic rules for automatically labeling each of the sentences in their corpus with exactly one of the following argument labels: Opposes, Supports, Citation, Claim, Hypothesis, Current Study, or None. After that, they employed these sentence labels to construct five heuristic rules to holistically score a student essay.

We create Baseline 2 as follows, employing the methods described in Section 4 for training, parameter tuning, and testing. We employ Ong et al.'s method to tag each sentence of our essays with an argument label, but modify their method to accommodate differences between their and our corpus. In particular, our more informal corpus does not contain Current Study or Citation sentences, so we removed portions of rules that attempt to identify these labels (e.g., portions of rules that search for a four-digit number, as would appear as the year in a citation). Our resulting rule set is shown in Table 6.3. If more than one of these rules applies to a sentence, we tag it with the label from the earliest rule that applies.

Table 6.3. Sentence labeling rules.

#	Rule
1	Sentences that begin with a comparison discourse connective or contain any string prefixes from "conflict" or "oppose" are tagged OPPOSES.
2	Sentences that begin with a contingency connective are tagged SUPPORTS.
3	Sentences containing any string prefixes from "suggest", "evidence", "shows", "Essentially", or "indicate" are tagged CLAIM.
4	Sentences in the first, second, or last paragraph that contain string prefixes from "hypothes", or "predict", but do not contain string prefixes from "conflict" or "oppose" are tagged HYPOTHESIS.
5	Sentences containing the word "should" that contain no contingency connectives or string prefixes from "conflict" or "oppose" are also tagged HYPOTHESIS.
6	If the previous sentence was tagged hypothesis and this sentence begins with an expansion connective, it is also tagged HYPOTHESIS.
7	Do not apply a label to this sentence.

After labeling all the sentences in our corpus, we then convert three of their five heuristic scoring rules into features for training a regressor.[4] The resulting three features describe (1) whether an essay contains at least one sentence labeled HYPOTHESIS, (2) whether it contains at least one sentence labeled OPPOSES, and (3) the sum of CLAIM sentences and SUPPORTS sentences divided by the number of paragraphs in the essay. If the value of the last feature exceeds 1, we instead assign it a value of 1. These features make sense because, for example, we would expect essays containing lots of SUPPORTS sentences to offer stronger arguments.

6.6 Our Approach

Our approach augments the feature set available to Baseline 2 with seven types of novel features.

[4]We do not apply the remaining two of their heuristic scoring rules because they deal solely with current studies and citations.

1. POS N-grams (POS) Word n-grams, though commonly used as features for training text classifiers, are typically not used in automated essay grading. The reason is that any list of word n-gram features automatically compiled from a given set of training essays would be contaminated with prompt-specific n-grams that may make the resulting regressor generalize less well to essays written for new prompts.

To generalize our feature set in a way that does not risk introducing prompt-dependent features, we introduce POS n-gram features. Specifically, we construct one feature from each sequence of $1-5$ part-of-speech tags appearing in our corpus. In order to obtain one of these features' values for a particular essay, we automatically label each essay with POS tags using the Stanford CoreNLP system (Manning et al., 2014), then count the number of times the POS tag sequence occurs in the essay. An example of a useful feature of this type is "CC NN ,", as it is able to capture when a student writes either "for instance," or "for example,". We normalize each essay's set of POS n-gram features to unit length.

2. Semantic Frames (SFR) While POS n-grams provide syntactic generalizations of word n-grams, FrameNet-style semantic role labels provide semantic generalizations. For each essay in our data set, we employ SEMAFOR (Das et al., 2010) to identify each semantic frame occurring in the essay as well as each frame element that participates in it. For example, a semantic frame may describe an event that occurs in a sentence, and the event's frame elements may be the people or objects that participate in the event. For a more concrete example, consider the sentence "I said that I do not believe that it is a good idea". This sentence contains a Statement frame because a statement is made in it. One of the frame elements participating in the frame is the Speaker "I". From this frame, we would extract a feature pairing the frame together with its frame element to get the feature "Statement-Speaker-I". We would expect this feature to be useful for argument quality scoring because we noticed that essays that focus excessively on the writer's personal opinions and experiences tended to receive lower argument quality scores.

As with POS n-grams, we normalize each essay's set of Semantic Frame features to unit length.

3. Transitional Phrases (TRP) We hypothesize that a more cohesive essay, being easier for a reader to follow, is more persuasive, and thus makes a stronger argument. For this reason, it would be worthwhile to introduce features that measure how cohesive an essay is. Consequently, we create features based on the 149 transitional phrases compiled by Study Guides and Strategies[5]. Study Guides and Strategies collected these transitions into lists of phrases that are useful for different tasks (e.g., a list of transitional phrases for restating points such as "in essence" or "in short"). There are 14 such lists, which we use to generalize transitional features. Particularly, we construct a feature for each of the 14 phrase type lists. For each essay, we assign the feature a value indicating the average number of transitions from the list that occur in the essay per sentence. Despite being phrase-based, transitional phrases features are designed to capture only *prompt-independent* information, which as previously mentioned is important in essay grading.

4. Coreference (COR) As mentioned in our discussion of transitional phrases, a strong argument must be cohesive so that the reader can understand what is being argued. While the transitional phrases already capture one aspect of this, they cannot capture when transitions are made via repeated mentions of the same entities in different sentences. We therefore introduce a set of 19 coreference features that capture information such as the fraction of an essay's sentences that mention entities introduced in the prompt, and the average number of total mentions per sentence.[6] Calculating these feature values, of course, requires that the

[5] `http://www.studygs.net/wrtstr6.htm`

[6] See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for a complete list of coreference features.

text be annotated with coreference information. We automatically coreference-annotate the essays using the Stanford CoreNLP system.

5. Prompt Agreement (PRA) An essay's prompt is always either a single statement, or can be split up into multiple statements with which a writer may Agree Strongly, Agree Somewhat, be Neutral, Disagree Somewhat, Disagree Strongly, Not Address, or explicitly have No Opinion on. We believe information regarding which of these categories a writer's opinion falls into has some bearing on the quality of her argument because, for example, a writer who explicitly mentions having no opinion has probably not made a persuasive argument.

For this reason, we annotate a subset of 830 of our ICLE essays with these agreement labels. We then train a multiclass maximum entropy classifier using MALLET (McCallum, 2002) for identifying which one of these seven categories an author's opinion falls into. The feature set we use for this task includes POS n-gram and semantic frame features as described earlier in this section, lemmatized word 1-3 grams, the keyword and prompt adherence keyword features we described in Persing and Ng (2013) and Persing and Ng (2014), respectively, and a feature indicating which statement in the prompt we are attempting to classify the author's agreement level with respect to.

Our classifier's training set in this case is the subset of prompt agreement annotated essays that fall within the training set of our 1000 essay argument quality annotated data. We then apply the trained classifier to our entire 1000 essay set in order to obtain predictions from which we can then construct features for argument quality scoring. For each prediction, we construct a feature indicating which of the seven classes the classifier believes is most likely, as well as seven additional features indicating the probability the classifier associates with each of the seven classes.

We produce additional related annotations on this 830 essay set in cases when the annotated opinion was neither Agree Strongly nor Disagree Strongly, as the reason

the annotator chose one of the remaining five classes may sometimes offer insight into the writer's argument. The classes of reasons we annotate include cases when the writer: (1) offered Conflicting Opinions, (2) Explicitly Stated an agreement level, (3) gave only a Partial Response to the prompt, (4) argued a Subtler Point not capturable by extreme opinions, (5) did not make it clear that the Writer's Position matched the one she argued, (6) only Briefly Discussed the topic, (7) Confusingly Phrased her argument, or (8) wrote something whose Relevance to the topic was not clear. We believe that knowing which reason(s) apply to an argument may be useful for argument quality scoring because, for example, the Conflicting Opinions class indicates that the author wrote a confused argument, which probably deserves a lower argument quality score.

We train eight binary maximum entropy classifiers, one for each of these reasons, using the same training data and feature set we use for agreement level prediction. We then use the trained classifiers to make predictions for these eight reasons on all 1000 essays. Finally, we generate features for our argument quality regressor from these predictions by constructing two features from each of the eight reasons. The first binary feature is turned on whenever the maximum entropy classifier believes that the reason applies (i.e., when it assigns the reason a probability of over 0.5). The second feature's value is the probability the classifier assigns for this reason.

6. Argument Component Predictions (ACP) Many of our features thus far do not result from an attempt to build a deep understanding of the structure of the arguments within our essays. To introduce such an understanding into our system, we follow Stab and Gurevych (2014a), who collected and annotated a corpus of 90 persuasive essays (not from the ICLE corpus) with the understanding that the arguments contained therein consist of three types of argument components. In one essay, these argument components typically include a Major Claim, several lesser Claims which usually support or attack the major claim, and Premises which usually underpin the validity of a claim or major claim.

Stab and Gurevych (2014b) trained a system to identify these three types of argument components within their corpus given the components' boundaries. Since our corpus does not contain annotated argument components, we modify their approach in order to simultaneously identify argument components and their boundaries.

We begin by implementing a maximum entropy version of their system using MALLET for performing the argument component identification task. We feed our system the same structural and lexical features they described. We then augment the system in the following ways.

First, since our corpus is not annotated with argument component boundaries, we construct a set of low precision, high recall heuristics for identifying the locations in each sentence where an argument component's boundaries might occur. The majority of these rules depend primarily on a syntactic parse tree we automatically generated for the sentence using the Stanford CoreNLP system. Since a large majority of annotated argument components are substrings of a simple declarative clause (an S node in the parse tree), we begin by identifying each S node in the sentence's tree.

Given one of these clauses, we collect a list of left and right boundaries where an argument component may begin or end. The rules we used to find these boundaries are summarized in Table 6.4.

Given an S node, we use our rules to construct up to $l \times r$ argument component candidate instances to feed into our system by combining each left boundary with each right boundary that occurs after it, where l is the number of potential left boundaries our rules found, and r is the number of right boundaries they found.

The second way we augment the system is by adding a boundary rule feature type. Whenever we generate an argument component candidate instance, we augment its normal feature set with two binary features indicating which heuristic rule was used to find the candidate's left boundary, and which rule was used to find its right boundary. If two rules

Table 6.4. Rules for extracting candidate argument component boundary locations.

(a) Potential left boundary locations

#	Rule
1	Exactly where the S node begins.
2	After an initial explicit connective, or if the connective is immediately followed by a comma, after the comma.
3	After nth comma that is an immediate child of the S node.
4	After nth comma.

(b) Potential right boundary locations

#	Rule
5	Exactly where the S node ends, or if S ends in a punctuation, immediately before the punctuation.
6	If the S node ends in a (possibly nested) SBAR node, immediately before the nth shallowest SBAR.[6]
7	If the S node ends in a (possibly nested) PP node, immediately before the nth shallowest PP.

can be used to find the same left or right boundary position, the first rule listed in the table is the one used to create the boundary rule feature. This is why, for example, the table contains multiple rules that can find boundaries at comma locations. We would expect some types of commas (e.g., ones following an explicit connective) to be more significant than others.

A last point that requires additional explanation is that several of the rules contain the word "nth". This means that, for example, if a sentence contains multiple commas, we will generate multiple left boundary positions for it using rule 4, and the left boundary rule feature associated with each position will be different (e.g., there is a unique feature for the first comma, and for the the second comma, etc.).

The last augmentation we make to the system is that we apply a NONE label to all argument component candidates whose boundaries do not exactly match those of a gold standard argument component. While Stab and Gurevych also did this, their list of such argument component candidates consisted solely of sentences containing no argument components at

[6]The S node may end in an SBAR node which itself has an SBAR node as its last child, and so on. In this case, the S node could be said to end with any of these "nested" SBARs, so we use the position before each (nth) one as a right boundary.

all. We could not do this, however, since our corpus is not annotated with argument components and we therefore do not know which sentences these would be.

We train our system on all the instances we generated from the 90 essay corpus and apply it to label all the instances we generated in the same way from our 1000 essay ICLE corpus. As a result, we end up with a set of automatically generated argument component annotations on our 1000 essay corpus. We use these annotations to generate five additional features for our argument quality scoring SVM regressor. These features' values are the number of major claims in the essay, the number of claims in the essay, the number of premises in the essay, the fraction of paragraphs that contain either a claim or a major claim, and the fraction of paragraphs that contain at least one argument component of any kind.

7. Argument Errors (ARE) We manually identified three common problems essays might have that tend to result in weaker arguments, and thus lower argument quality scores. We heuristically construct three features, one for each of these problems, to indicate to the learner when we believe an essay has one of these problems.

It is difficult to make a reasonably strong argument in an essay that is too short. For this reason, we construct a feature that encodes whether the essay has 15 or fewer sentences, as only about 7% of our essays are this short.

In the Stab and Gurevych corpus, only about 5% of paragraphs have no claims or major claims in them. We believe that an essay that contains too many of these claim or major claim-less paragraphs may have an argument that is badly structured, as it is typical for a paragraph to contain one or two (major) claim(s). For this reason, we construct a feature that encodes whether more than half of the essay's paragraphs contain no claims or major claims, as indicated by the previously generated automatic annotations.

Similarly, only 5% of the Stab and Gurevych essays contain no argument components at all. We believe that an essay that contains too many of these component-less paragraphs

is likely to have taken too much space discussing issues that are not relevant to the main argument of the essay. For this reason, we construct a feature that encodes whether more than one of the essay's paragraphs contain no components, as indicated by the previously generated automatic annotations.

6.7 Evaluation

In this section, we evaluate our system for argument quality scoring. All the results we report are obtained via five-fold cross-validation experiments. In each experiment, we use 60% of our labeled essays for model training, another 20% for parameter tuning and feature selection, and the final 20% for testing. These correspond to the training set, held-out validation data, and test set mentioned in Section 6.4.

6.7.1 Scoring Metrics

We employ four evaluation metrics. As we will see below, $S1$, $S2$, and $S3$ are *error* metrics, so lower scores on them imply better performance. In contrast, PC is a *correlation* metric, so higher correlation implies better performance.

The simplest metric, $S1$, measures the frequency at which a system predicts the wrong score out of the seven possible scores. Hence, a system that predicts the right score only 25% of the time would receive an $S1$ score of 0.75.

The $S2$ metric measures the average distance between a system's predicted score and the actual score. This metric reflects the idea that a system that predicts scores close to the annotator-assigned scores should be preferred over a system whose predictions are further off, even if both systems estimate the correct score at the same frequency.

The $S3$ metric measures the average square of the distance between a system's score predictions and the annotator-assigned scores. The intuition behind this metric is that not only should we prefer a system whose predictions are close to the annotator scores, but

Table 6.5. Five-fold cross-validation results for argument quality scoring.

System	$S1$	$S2$	$S3$	PC
Baseline 1	.668	.428	.321	.000
Baseline 2	.652	.418	.267	.061
Our System	.618	.392	.244	.212

we should also prefer one whose predictions are not too frequently very far away from the annotated scores. The three error metric scores are given by:

$$\frac{1}{N}\sum_{A_j \neq E'_j} 1, \quad \frac{1}{N}\sum_{j=1}^{N} |A_j - E_j|, \quad \frac{1}{N}\sum_{j=1}^{N} (A_j - E_j)^2$$

where A_j, E_j, and E'_j are the annotator assigned, system predicted, and rounded system predicted scores[7] respectively for essay j, and N is the number of essays.

The last metric, PC, computes Pearson's correlation coefficient between a system's predicted scores and the annotator-assigned scores. PC ranges from -1 to 1. A positive (negative) PC implies that the two sets of predictions are positively (negatively) correlated.

6.7.2 Results and Discussion

Five-fold cross-validation results on argument quality score prediction are shown in Table 6.5. The first two rows show our baseline systems' performances. The best baseline system (Baseline 2), which recall is a learning-based version of Ong et al.'s (2014) system, predicts the wrong score 65.2% of the time. Its predictions are off by an average of .418 points, the average squared error of its predictions is .267, and its average Pearson correlation coefficient with the gold argument quality score across the five folds is .061.

Results of our system are shown on the third row of Table 6.5. Rather than using all of the available features (i.e., Baseline 2's features and the novel features described in Section 6),

[7]We round all predictions to 1.0 or 4.0 if they fall outside the 1.0−4.0 range and round $S1$ predictions to the nearest half point.

our system uses only the feature subset selected by the backward elimination feature selection algorithm (Blum and Langley, 1997) that achieves the best performance on the validation data (see Section 6.7.3 for details). As we can see, our system predicts the wrong score only 61.8% of the time, predicts scores that are off by an average of .392 points, the average squared error of its predictions is .244, and its average Pearson correlation coefficient with the gold scores is .212. These numbers correspond to relative error reductions[8] of 5.2%, 6.2%, 8.6%, and 16.1% over Baseline 2 for S1, S2, S3, and PC, respectively, the last three of which are significant improvements.[9] The magnitudes of these improvements suggest that, while our system yields improvements over the best baseline by all four measures, its greatest contribution is that its predicted scores are best-correlated with the gold standard argument quality scores.

6.7.3 Feature Ablation

To gain insight into how much impact each of the feature types has on our system, we perform feature ablation experiments in which we remove the feature types from our system one-by-one.

We show the results of the ablation experiments on the held-out validation data as measured by the four scoring metrics in Table 6.6. The top line of each subtable shows what a system that uses all available features's score would be if we removed just one of the feature types. So to see how our system performs by the PC metric if we remove only prompt agreement (PRA) features, we would look at the first row of results of Table 6.6(d) under the column headed by PRA. The number here tells us that the resulting system's PC score is .303. Since our system that uses all feature types obtains $S1$, $S2$, $S3$, and PC scores of

[8]These numbers are calculated $\frac{B-O}{B-P}$ where B is the baseline system's score, O is our system's score, and P is a perfect score. Perfect scores for error measures and PC are 0 and 1 respectively.

[9]All significance tests are paired t-tests with $p < 0.05$.

Table 6.6. Argument quality feature ablation results. In each subtable, the first row shows how our system would perform on the validation set essays if each feature type was removed. We then remove the least important feature type, and show in the next row how the adjusted system would perform without each remaining type.

(a) Results using the $S1$ metric

SFR	ACP	TRP	PRA	POS	COR	ARE	BAS
.534	.594	.530	.524	.522	.532	.529	**.521**
.530	.554	.526	.529	.526	.528	.525	
.534	.555	.525	.531	.528	.522		
.543	.558	.536	.530	.527			
.565	.561	.536	.529				
.563	.547	.539					
.592	.550						

(b) Results using the $S2$ metric

POS	PRA	ACP	TRP	BAS	SFR	COR	ARE
.370	.369	.375	.367	.367	.366	.366	.365
.369	.369	.375	.366	.366	.365	.365	
.370	.371	.372	.367	.366	**.365**		
.374	.374	.376	.368	.366			
.377	.375	.374	.368				
.381	.377	.376					
.385	.382						

(c) Results using the $S3$ metric

POS	PRA	ACP	TRP	BAS	COR	ARE	SFR
.221	.220	.225	.219	.218	.217	.217	.211
.220	.219	.221	.214	.212	.211	.211	
.218	.218	.220	.212	.211	**.209**		
.221	.216	.218	.212	.210			
.224	.217	.218	.212				
.228	.220	.219					
.229	.225						

(d) Results using the PC metric

POS	ACP	PRA	TRP	BAS	ARE	COR	SFR
.302	.270	.303	.326	.324	.347	.347	.356
.316	.300	.327	.344	.361	.366	.371	
.346	.331	.341	.356	.367	**.378**		
.325	.331	.345	.362	.375			
.297	.331	.339	.360				
.280	.320	.321					
.281	.281						

.521, .366, .218, and .341 on the validation data respectively, the removal of PRA features costs the complete system .038 PC points, and thus we can infer that the inclusion of PRA features has a beneficial effect.

From row 1 of Table 6.6(a), we can see that removing the Baseline 2 feature set (BAS) yields a system with the best $S1$ score in the presence of the remaining feature types in this row. For this reason, we permanently remove the BAS features from the system before we generate the results on line 2. We iteratively remove the feature type that yields a system with the best performance in this way until we get to the last line, where only one feature type is used to generate each result.

Since the feature type whose removal yields the best system is always the rightmost entry in a line, the order of column headings indicates the relative importance of the feature types, with the leftmost feature types being most important to performance and the rightmost

Table 6.7. Distribution of predicted argument quality scores.

	$S1$			$S2$			$S3$			PC		
Gold	.25	.50	.75	.25	.50	.75	.25	.50	.75	.25	.50	.75
1.0	2.90	2.90	2.90	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74	2.74
1.5	2.69	2.78	2.89	2.36	2.67	2.78	2.52	2.63	2.71	2.52	2.63	2.81
2.0	2.61	2.72	2.85	2.54	2.69	2.79	2.60	2.69	2.78	2.60	2.70	2.80
2.5	2.64	2.71	2.85	2.65	2.75	2.86	2.66	2.75	2.85	2.69	2.79	2.89
3.0	2.73	2.84	2.92	2.71	2.81	2.91	2.70	2.80	2.90	2.72	2.83	2.90
3.5	2.74	2.85	2.97	2.78	2.89	3.02	2.79	2.90	3.00	2.81	2.90	2.98
4.0	2.75	2.87	3.10	2.76	2.85	3.09	2.76	2.83	3.08	2.81	2.86	3.19

feature types being least important in the presence of the other feature types. The score corresponding to the best system is boldfaced for emphasis, indicating that all feature types appearing to its left are included in the best system.[10]

It is interesting to note that while the relative importance of different feature types does not remain exactly the same if we measure performance in different ways, we can see that some feature types tend to be more important than others in a majority of the four scoring metrics.

From these tables, it is clear that POS n-grams (POS), prompt agreement features (PRA), and argument component predictions (ACP) are the most generally important feature types in roughly that order. They all appear in the leftmost three positions under the tables for metrics $S2$, $S3$, and PC, the three metrics by which our system significantly outperforms Baseline 2. Furthermore, removing any of them tends to have a larger negative impact on our system than removing any of the other feature types.

Transitional phrase features (TRP) and Baseline 2 features (BAS), by contrast, are of more middling importance. While both appear in the best feature sets for the aforementioned

[10]The reason the performances shown in these tables appear so much better than those shown previously is that in these tables we tune parameters and display results on the validation set in order to make it clearer why we chose to remove each feature type. In Table 6.5, by contrast, we tune parameters on the validation set, but display results using those parameters on the test set.

metrics (i.e., they appear to the left of the boldfaced entry in the corresponding ablation tables), the impact of their removal is relatively less than that of POS, PRA, or ACP features.

Finally, while the remaining three feature types might at first glance seem unimportant to argument quality scoring, it is useful to note that they all appear in the best performing feature set as measured by at least one of the four scoring metrics. Indeed, semantic frame features (SFR) appear to be the most important feature type as measured by the $S1$ metric, despite being one of the least useful feature types as measured by the other performance metrics. From this we learn that when designing an argument quality scoring system, it is important to understand what the ultimate goal is, as the choice of performance metric can have a large impact on what type of system will seem ideal.

6.7.4 Analysis of Predicted Scores

To more closely examine the behavior of our system, in Table 6.7 we chart the distributions of scores it predicts for essays having each gold standard score. As an example of how to read this table, consider the number 2.60 appearing in row 2.0 in the .25 column of the $S3$ region. This means that 25% of the time, when our system with parameters tuned for optimizing $S3$ (including the $S3$ feature set as selected in Table 6.6(c)) is presented with a test essay having a gold standard score of 2.0, it predicts that the essay has a score less than or equal to 2.60.

From this table, we see that our system has a bias toward predicting more frequent scores as the smallest entry in the table is 2.36 and the largest entry is 3.19, and as we saw in Table 6.2, 71.4% of essays have gold scores in this range. Nevertheless, our system does not rely entirely on bias, as evidenced by the fact that each column in the table has a tendency for its scores to ascend as the gold standard score increases, implying that our system has some success at predicting lower scores for essays with lower gold standard argument quality scores and higher scores for essays with higher gold standard argument quality scores. The major

exception to this rule is line 1.0, but this is to be expected since there are only two essays having this gold score, so the sample from which the numbers on this line are calculated is very small.

6.8 Summary

We proposed a feature-rich approach to the new problem of predicting argument quality scores on student essays. In an evaluation on 1000 argumentative essays selected from the ICLE corpus, our system significantly outperformed a baseline system that relies solely on features built from heuristically labeled sentence argument function labels by up to 16.1%. To stimulate further research on this task, we make all of our annotations publicly available.

CHAPTER 7

STANCE CLASSIFICATION[1]

7.1 Introduction

State-of-the-art automated essay scoring engines such as E-*rater* (Attali and Burstein, 2006) do not grade essay content, focusing instead on providing diagnostic trait feedback on categories such as grammar, usage, mechanics, style and organization. Hence, persuasiveness and other content-dependent dimensions of argumentative essay quality are largely ignored in existing automated essay scoring research. While full-fledged content-based essay scoring is still beyond the reach of state-of-the-art essay scoring engines, recent work has enabled us to move one step closer to this ambitious goal by analyzing essay content, attempting to determine the argumentative structure of student essays (Stab and Gurevych, 2014b) and the persuasiveness of the arguments made in these essays (Persing and Ng, 2015).

Stance classification is an important first step in determining how persuasive an argumentative student essay is because persuasiveness depends on how well the author argues *w.r.t. the stance she takes* using the supporting evidence she provides. For instance, if her stance is *Agree Somewhat*, a persuasive argument would involve explaining what reservations she has about the given proposition. As another example, an argumentative essay in which the author takes a *neutral* stance or the author presents evidence that does not support the stance she claims to take should receive a low persuasiveness score.

Given the important role played by stance classification in determining an essay's persuasiveness, our goal in this chapter is to examine stance classification in argumentative student essays. While there is a large body of work on stance classification,[2] stance classification in argumentative essays is largely under-investigated and is different from previous

[1]This chapter was previously published in Persing and Ng (2016b).

[2]Previous approaches to stance classification have focused on three discussion/debate settings, namely congressional floor debates (Thomas et al., 2006; Bansal et al., 2008; Balahur et al., 2009; Yessenalina et al.,

work in several respects. First, in automated essay grading, the majority of the essays to be assessed are written by students who are learners of English. Hence our stance classification task could be complicated by the authors' lack of fluency in English. Second, essays are longer and more formally written than the text typically used in previous stance classification research (e.g., debate posts). In particular, a student essay writer typically expresses her stance on the essay's topic in a thesis sentence/clause, while a debate post's author may never even explicitly express her stance. Although the explicit expression of stance in essays seems to make our task easier, identifying stancetaking text in the midst of non-stancetaking sentences in a potentially long essay, as we will see, is by no means a trivial task.

To our knowledge, the essay stance classification task has only been attempted by Faulkner (2014). However, the version of the task we address is different from his. First, Faulkner only performed two-class stance classification: while his corpus contains essays labeled with *For* (Agree), *Against* (Disagree), and *Neither*, he simplified the task by leaving out the arguably most difficult-to-identify stance, *Neither*. In contrast, we perform *fine-grained* stance classification, where we allow essay stance to take one of six values: *Agree Strongly*, *Agree Somewhat*, *Neutral*, *Disagree Somewhat*, *Disagree Strongly*, and *Never Addressed*, given the practical need to perform fine-grained stance classification in student essays, as discussed above. Second, given that many essay prompts are composed of multiple simpler propositions (e.g., the prompt "Most university degrees are theoretical and do not prepare students for the real world" has two parts, "Most university degrees are theoretical" and "Most university degrees do not prepare students for the real world."), we manually split such prompts into *prompt part*s and determine the stance of the author w.r.t. each part, whereas Faulkner assigned an overall stance to a given prompt regardless of whether it is composed of multiple

2010; Burfoot et al., 2011), company-internal discussions (Agrawal et al., 2003; Murakami and Raymond, 2010), and online social, political, and ideological debates (Wang and Rosé, 2010; Biran and Rambow, 2011; Walker et al., 2012; Abu-Jbara et al., 2013; Hasan and Ng, 2013; Boltužić and Šnajder, 2014; Sobhani et al., 2015; Sridhar et al., 2015).

propositions. The distinction is important because an analysis of our annotations described in Section 7.2 shows that essay authors take different stances w.r.t. different prompt parts in 49% of essays, and in 39% of essays, authors even take stances with different polarities w.r.t. different prompt parts.

In sum, our contributions in this chapter are two-fold. First, we propose a computational model for essay stance classification that outperforms four baselines, including our re-implementation of Faulkner's approach. Second, in order to stimulate further research on this task, we make our annotations publicly available. Since progress on this task is hindered in part by the lack of a publicly annotated corpus, we believe that our data set will be a valuable resource for the NLP community.

7.2 Corpus

We use as our corpus the 4.5 million word International Corpus of Learner English (ICLE) (Granger et al., 2009) described in Section 3.2. 91% of the ICLE texts are written in response to prompts that trigger argumentative essays, and thus are expected to take a stance on some issue. We select 11 such prompts, and from the subset of argumentative essays written in response to them, we select 826 essays to annotate for training and testing our stance classification system.[3] Table 7.1 shows two of the 11 topics selected for annotation.

We pair each of the 826 essays with each of the prompt parts to which it responds, resulting in 1,593 *instance*s.[4] We then familiarize two human annotators, both of whom are native speakers of English, with the stance definitions in Table 7.2 and ask them to assign each instance the stance label they believe the essay's author would have chosen if

[3]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for the complete list of essay stance annotations.

[4]We do not segment the essays' texts according to which prompt part is being responded to. Each (entire) essay is viewed as a response to all of its associated prompt parts.

Table 7.1. Some example essay prompts with their associated parts.

Prompt	Prompt Parts
Most university degrees are theoretical and do not prepare students for the real world. They are therefore of very little value.	1) Most university degrees are theoretical. 2) Most university degrees do not prepare students for the real world. 3) Most university degrees are of very little value.
The prison system is outdated. No civilized society should punish its criminals: it should rehabilitate them.	1) The prison system is outdated. 2) No civilized society should punish its criminals. 3) Civilized societies should rehabilitate criminals.

Table 7.2. Stance label counts and definitions.

Stance	Definition
Agree Strongly (885)	The author seems to agree with and care about the claim.
Agree Somewhat (148)	The author generally agrees with the claim, but might be hesitant to choose "Agree Strongly".
Neutral (28)	The author agrees with the claim as much as s/he disagrees with it.
Disagree Somewhat (91)	The author generally disagrees with the claim, but might be hesitant to choose "Disagree Strongly".
Disagree Strongly (416)	The author seems to disagree with and care about the claim.
Never Addressed (25)	A stance cannot be inferred because the proposition was never addressed.

asked how strongly she agrees with the prompt part. We additionally furnish the annotators with descriptions of situations that might cause an author to select the more ambiguous classes. For example, an author might choose Agree Somewhat if she appears to mostly agree with the prompt part, but qualifies her opinion in a way that is not captured by the prompt part's bluntness (e.g., an author who claims the prison system in a lot of countries is outdated would Agree Somewhat with the first part of Table 7.1's second prompt). Or she may choose Disagree Somewhat if she appears to disagree with the prompt part, but mentions the disagreement only in passing because she does not care much about the topic.

To ensure consistency in annotation, we randomly select 100 essays (187 instances) for annotation by both annotators. Their labels agree in 84.5% of the instances, yielding a Cohen's (1960) Kappa of 0.76. Each case of disagreement is resolved through discussion between the annotators.

7.3 Baseline Stance Classification Systems

In this section, we describe four baseline systems.

7.3.1 Agree Strongly Baseline

Given the imbalanced stance distribution shown in Table 7.2, we create a simple but by no means weak baseline, which predicts that every instance has most frequent class label (Agree Strongly), regardless of the prompt part or the essay's contents.

7.3.2 N-Gram Baseline

Previous work on stance classification, which assumes that stance-annotated training data is available for every topic for which stance classification is performed, has shown that the N-Gram baseline is a strong baseline. Not only is this assumption unrealistic in practice, but it has led to undesirable consequences. For instance, the proposition "feminists have done more harm to the cause of women than good" elicits much more disagreement than normal. So, if instances from this proposition appeared in both the training and test sets, the unigram feature "feminist" would be strongly correlated with the disagreement classes even though intuitively it tells us nothing about stance. This partly explains why the N-Gram baseline was strong in previous work (Somasundaran and Wiebe, 2010). In light of this problem, we perform leave-one-out cross validation where we partition the instances by *prompt*, leaving the instances created for one prompt out in each test set.

To understand how strong n-grams are when evaluated in our leave-one-prompt-out cross-validation setting, we employ them as features in our second baseline. Specifically, we train a multiclass classifier on our data set using a feature set composed solely of unigram, bigram, and trigram features, each of which indicates the number of times the corresponding n-gram is present in the associated essay.

7.3.3 Duplicated Faulkner Baseline

While it is true that no system exists for solving our exact problem, the system proposed by Faulkner (2014) comes fairly close. Hence, as our third baseline, we train a multiclass classifier on our data set for fine-grained essay stance classification using the two types of features proposed by Faulkner, as described below.

Part-of-speech (POS) generalized dependency subtrees. Faulkner first constructs a lexicon of stance words in the style of Somasundaran and Wiebe (2010). The lexicon consists of (1) the set of stemmed first unigrams appearing in all stance-annotated text spans in the Multi-Perspective Question Answering (MPQA) corpus (Wiebe et al., 2005), and (2) the set of boosters (clearly, decidedly), hedges (claim, estimate), and engagement markers (demonstrate, evaluate) from the appendix of Hyland (2005). He then manually removes from this list any words that appear not to be stancetaking, resulting in a 453 word lexicon.

Stance words target propositions, which Faulkner notes, usually contain some opinion-bearing language that can serve as a proxy for the targeted proposition. In order to find the locations in an essay where a stance is being taken, he first finds each stance word in the essay. Then he finds the shortest path from the stance word to an opinion word in the sentence's dependency tree, using the MPQA subjectivity lexicon of opinion words (Wiebe et al., 2005). If this nearest opinion word appears in the stance word's immediate or embedded clause,

he creates a binary feature by concatenating all the words in the dependency path, POS generalizing all words other than the stance and opinion word, and finally prepending "not" if the stance word is adjacent to a negator in the dependency tree. Thus given the sentence "I **can** only say that this statement is completely *true*." he would add the feature *can-V-true*, which suggests agreement with the prompt.

Prompt topic words. Recall that for the previous feature type, a feature was generated whenever an opinion word occurred in a stance word's immediate or embedded clause. Each content word in this clause is used as a binary feature if its similarity with one of the prompt's content words meets an empirically determined threshold.

7.3.4 N-Gram+Duplicated Faulkner Baseline

To build a stronger baseline, we employ as our fourth baseline a classifier trained on both n-gram features and duplicated Faulkner's features.

7.4 Our Approach

Our approach to stance classification is a learning-based approach where we train a multiclass classifier using four types of features: n-gram features (Section 3.2), duplicated Faulkner's features (Section 3.3), and two novel types of features, stancetaking path-based features (Section 4.1) and knowledge-based features (Section 4.2).

7.4.1 Stancetaking Path-Based Features

Recall that, in order to identify his POS generalized dependency subtrees, Faulkner relies on two lexica, a lexicon of stancetaking words and a lexicon of opinion-bearing words. He then extracts a feature any time words from the two lexica are syntactically close enough. A major problem with this approach is that the lexica are so broad that nearly 80% of sentences in our

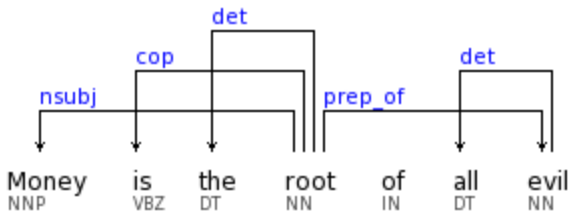

Figure 7.1. Automatic dependency parse of a prompt part.

corpus contain text that can be identified as stancetaking using this method. Intuitively, an essay may state its stance w.r.t. a prompt part in a thesis or conclusion sentence, but most of essay's text will be at most tangentially related to any particular prompt part. For this reason, we propose to identifying stancetaking text to target *only text that appears directly related to the prompt part*. Below we first show how we identify and stance-labeling relevant stancetaking dependency paths, and then describe the features we derive from these paths.

Identifying relevant stancetaking paths

As noted above, we first identify stancetaking text that appears directly related to the prompt part.

To begin, we note that the prompt parts themselves must express a stance on a topic if they can be agreed or disagreed with. By examining the dependency parses[5] of the prompt parts, we can recognize elements of how stancetaking text is structured. From the prompt part shown in Figure 7.1, for example, we notice that the important words that express a stance in the sentence are "money", "root", and "evil". By analyzing the dependency structure in this and other prompt parts, we discovered that stancetaking text often consists of (1) a **subject** word, which is the child in an nsubj or nsubjpass relation, (2) a **governor** word which is the subject's parent, and (3) an **object**, which is a content word from which

[5]Dependency parsing, POS tagging, and lemmatization are performed automatically using the Stanford CoreNLP system (Manning et al., 2014)

there is a (not always direct) dependency path from the governor. We therefore abstract a stance in an essay as a dependency path from a subject to an object that passes through the governor. Thus, the stancetaking dependency path we identify from the prompt part shown in Figure 7.1 could be represented as money-root-evil.

The obvious problem with identifying stancetaking text in this way is that nearly all sentences contain this kind of stancetaking structure, and just as with Faulkner's dependency paths, there is little reason to believe that any particular path is relevant to an instance's prompt part. Does this mean that nearly all sentences are stancetaking? We would argue that they can be, as even sentences that appear on their face to be mere statements of fact with no apparent value judgment can be viewed as taking a stance on the factuality of the statement, and people often disagree about the factuality of statements. For this reason, after we have identified a stancetaking path, we must determine whether the stance being expressed is relevant to the prompt part before extracting features from it.

For this reason, we ignore all stancetaking paths that do not meet the following three relevance conditions. First, the lemma of the path's governor must match the lemma of a governor in the prompt part. Second, the lemma of the path's object must match the lemma of some content word[6] in the prompt part. Finally, the containing sentence must not contain a question mark or a quotation, as such sentences are usually rhetorical in nature. We do not require that the subject word match the prompt part's subject word because this substantially reduces coverage for various reasons. For one, of the three words (subject, governor, object), the subject is the word most likely to be replaced with some other word like a pronoun, and possibly because the essays were written by non-native English speakers, automatic coreference resolution cannot reliably identify these cases. We also do not fully trust that the subject identified by the dependency parser will reliably match the subject

[6]For our purpose, a content word (1) is a noun, pronoun, verb, adjective, or adverb, (2) is not a stopword, and (3) is at the root, is a child in a dobj or pobj relation, or is the child in a conj relation whose parent is the child in a dobj or pobj relation in the dependency tree.

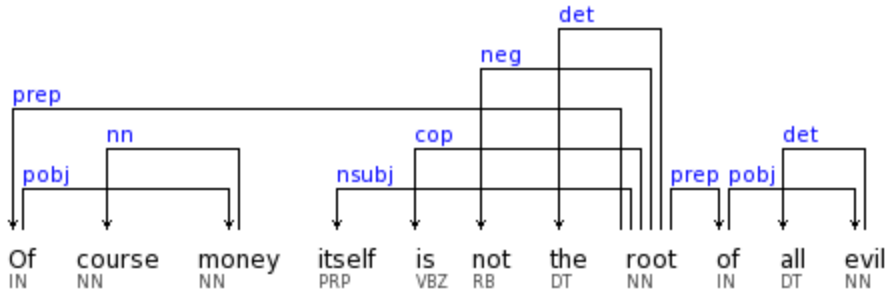

Figure 7.2. Automatic dependency parse of an essay sentence.

we are looking for. Given these constraints, we can automatically identify the "itself-root-of-evil" dependency path in Figure 7.2 as a relevant stancetaking path.

Stance-labeling the paths

Next, we determine whether a stancetaking path identified in the previous step appears to agree or disagree with the prompt part.

To begin, we count the number of negations occurring in the prompt part. Any word like "no", "not", or "none" counts as a negation unless it begins a non-negation phrase like "no doubt" or "not only".[7] Thus, the count of negations in the prompt part in Figure 7.1 is 0.

After that, we count the number of times the identified stancetaking path is negated. Because these paths occur in student essays and are therefore often not as simply-stated as the prompt parts, this is a little bit more complicated than just counting the containing sentence's negations since the sentence may contain a lot of additional material. To do this, we construct a list of all the dependency nodes in the stancetaking path as well as all of their dependency tree children. We then remove from this list any node that, in the sentence, occurs after the last node in the stancetaking path. The total negation count we are looking for is the number of nodes in this list that correspond to negation words (unless the negation

[7]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for our list of manually constructed negation words and non-negation phrases.

word begins a negation phrase). Thus, because the word "not" is the child of "root" in the path "itself-root-of-evil" we identified in Figure 7.2, we consider this path to have been negated one time.

Finally, we sum the prompt part negations and the stancetaking path negations. If this sum is even, we believe that the relevant stancetaking path agrees with the prompt part in the instance. If it is odd, however (as in the case of the prompt part and stancetaking text in the dependency tree figures), we believe that it disagrees with the prompt part. To illustrate why we are concerned with whether this sum is even, consider the following examples. If both the prompt part and the stancetaking text are negated, both disagree with the opposite of the prompt part's stance. Thus, they agree with each other, and their negation sum is even (2). If the stancetaking path was negated twice, however, the sum would be odd (3) due to the stance path's double negations canceling each other out, and the stancetaking path would disagree with the prompt part.

Deriving path-based features

We extract four features from the relevant stancetaking dependency paths identified and stance-labeled so far, as described below.

The first feature encodes the count of relevant stancetaking paths that appear to agree with the prompt part. The second feature encodes the count of relevant stancetaking paths that appear to disagree with the prompt part. While we expect these first two features to be correlated with the agreement and disagreement classes, respectively, they may not be sufficient to distinguish between agreeing and disagreeing instances. It is possible, for example, that both features may be greater than zero in a single instance if we have identified one stancetaking path that appears to agree with the prompt part and another stancetaking path that appears to disagree with the prompt part. It is not clear whether this situation is indicative of only the Neutral class, or perhaps it indicates partial (Somewhat) (Dis)Agreement, or

maybe our method of detecting disagreement is not reliable enough, and it therefore makes sense, when we get these conflicting signals, to ignore them entirely and just assign the instance to the most frequent (Agree Strongly) class. For that matter, if neither feature is greater than zero, does this mean that the instance Never Addressed the prompt part, or does it instead mean that our method for identifying stancetaking paths doesn't have high enough recall to work on all instances? We let our learner sort these problems out by adding two more binary features to our instances, one which indicates that both of the first two features are zero, and one that indicates whether both are greater than zero.

7.4.2 Knowledge-Based Features

Our second feature type is composed of five linguistically informed binary features that correspond to five of the six classes in our fine-grained stance classification task. Intuitively, if an instance has one of these features turned on, it should be assigned to the feature's corresponding class.

1. Neutral. Stancetaking text indicating neutrality tends to be phrased somewhat differently than stancetaking text indicating any other class. In particular, neutral text often makes claims that are about the prompt part's subject, but which are tangential to the proposition expressed in the prompt part. For this reason, we search the essay for words that match the prompt part's subject lemmatically.

After identifying a sentence that is about the prompt part's subject in this way, we check whether the sentence begins with any neutral indicating phrase.[8] If we find a sentence that both begins with a neutral phrase and is about the prompt part's subject, we turn the Neutral feature on. Thus, sentences like the following can be captured: "*In all probability* university

[8]We construct a list of neutral phrases for introducing another person's ideas from a writing skills website (`http://www.myenglishteacher.eu/question/other-ways-to-say-according-to/`).

students wonder whether or not they spend their time uselessly in studying through four or five years in order to take their *degree*."

2. (Dis)Agree Somewhat. In order to set the values of the features associated with the Somewhat classes, we first identify relevant stancetaking paths as described above. We then trim the list of paths by removing any path whose governor or subject does not have a hedge word as an adverb modifier child in the dependency tree.[9] Thus, we are able to determine that the essay containing the sentence "There is *nearly* no place left for dream and imagination" is likely to belong to one of the Somewhat classes w.r.t. the prompt part "There is no longer a place for dreaming and imagination."

The question now is how to determine which (if any) of the Somewhat classes it should belong to. We analyze all the paths from the list for negation in much the same way we described above, but with one major difference. We hypothesize that when taking a Somewhat stance, students are more likely to explicitly state that the stance being taken is their opinion rather than stating the stance bluntly without attribution. For example, one Disagree Somewhat essay includes the sentence, "*I* **never** *believed* these people were honest if saying that money is just the root of all evil." In order to determine that this sentence contains an indication of the Disagree Somewhat class, we need to account for the negation that occurs at the beginning, far away from the stancetaking path (money-root-of-evil). To do this, we semantically parse the sentence using SEMAFOR (Das et al., 2010). Each of the *semantic frames* detected by SEMAFOR describes an event that occurs in a sentence, and the event's *frame elements* may be the people or other entities that participate in the event. One of the semantic frames detected in this example sentence describes a Believer (I) and the content of his or her belief (all the text after "believed"). Because the sentence includes

[9]See our website at `http://www.hlt.utdallas.edu/~persingq/ICLE/` for our manually constructed list of hedge words.

a semantic frame that (1) contains a first person (I, we) Cognizer, Speaker, Perceiver, or Believer element, (2) contains an element that covers all the text in the dependency path (a Content frame element, in this case), and (3) the word that triggers the frame ("believed") has a negator child in the dependency tree, we add one to this relevant stancetaking path's negation count. This makes this hedged stancetaking path's negation count odd, so we believe that this sentence likely disagrees with its instance's prompt part somewhat. If we find a hedged stancetaking path with an odd negation count, we turn on the Disagree Somewhat feature. Similarly, if we find a hedged stancetaking path with an even negation count, we turn on the Agree Somewhat feature.

3. (Dis)Agree Strongly. When we believe there is strong evidence that an instance should belong to one of the Strongly classes, we turn on the corresponding (Dis)Agree Strongly feature. In particular, if we find a relevant stancetaking path that appears to agree with the prompt part (as described in Section 7.4.1), but do not find any such path that appears to disagree with it, we turn on the Agree Strongly feature. Similarly, if we find a relevant stancetaking path that appears to disagree with the prompt part, but do not find a relevant stancetaking path that appears to agree with it, we turn on the Disagree Strongly feature.

7.5 Evaluation

In this section we discuss how we evaluate our approach to stance classification.

7.5.1 Experimental Setup

Data partition. All our results are obtained via leave-one-prompt-out cross-validation experiments. So, in each fold experiment, we partition the instances from our 11 prompts into a training set (10 prompts) and a test set (1 prompt).

Evaluation metrics. We employ two metrics to evaluate our systems: (1) micro F-score, which treats each instance as having equal weight; and (2) macro F-score, which treats each class as having equal weight.[10] To gain insights into how different systems perform on different classes, we additionally report per-class F-scores.

Training. We train the baselines and our approach using two learning algorithms, MALLET's (McCallum, 2002) implementation of maximum entropy (MaxEnt) classification and our own implementation of the one nearest neighbor (1NN) algorithm using the cosine similarity metric. Note that these two learners have their own strengths and weaknesses: in comparison to 1NN, MaxEnt is better at exploiting high-dimensional features but less robust to skewed class distributions. For the baseline systems, we select the learner by performing cross validation on the training folds to maximize the average of micro and macro F-scores in each fold experiment.

When training our approach, we perform exhaustive feature selection to determine which subset of the four sets of features (i.e., n-gram, duplicated Faulkner, path-based, and knowledge-based features) should be used. Specifically, we select the feature groups and learner jointly by performing cross validation on the training folds, choosing the combination yielding the highest average of micro and macro F-scores in each fold experiment. To prevent any feature type from dominating the others, to each feature we apply a weight of one divided by the number of features having its type.

Testing. In case of a tie when applying 1NN, the tie is broken by selecting the class appearing higher in Table 7.2.

[10]Since stance classification is a multiclass, single-label task, micro F-score, precision, recall, and accuracy are all equivalent.

Table 7.3. Cross-validation results for fine-grained essay stance classification, including per-class F-scores for Agree Strongly (A+), Agree Somewhat (A−), Neutral (Neu), Disagree Somewhat (D−), Disagree Strongly (D+), and Never Addressed (Nev).

	System	Micro-F	Macro-F	A+	A−	Neu	D−	D+	Nev
1	Always Agree Strongly	55.6	11.9	71.4	.0	.0	.0	.0	.0
2	N-Gram	55.4	12.0	71.3	.0	.0	.0	.5	.0
3	Duplicated Faulkner	50.8	15.6	66.8	4.0	.0	.0	22.9	.0
4	N-Gram + Duplicated Faulkner	53.4	15.4	69.1	2.5	.0	.0	20.6	.0
5	Our approach	**60.6**	**20.1**	73.6	.0	.0	2.1	44.8	.0

7.5.2 Results and Discussion

Results on fine-grained essay stance classification are shown in Table 7.3. The first four rows show our baselines' performances. Among the four baselines, Always Agree Strongly performs best w.r.t. micro F-score, obtaining a score of 55.6%, whereas Duplicated Faulkner performs best w.r.t. macro F-score, obtaining a score of 15.6%. Despite its poor performance, Duplicated Faulkner is a state-of-the-art approach on this task. Its poor performance can be attributed to three major factors. First, it was intended to identify only Agree and Disagree instances (note that Faulkner simply removed neutral instances from his experimental setup), which should not prevent them from performing well w.r.t. micro F-score. Second, it is far too permissive, generating features from a large majority of sentences while relevant sentences are far rarer. Third, while it does succeed at predicting Disagree Strongly far more frequently than either of the other baselines that excludes the Faulkner feature set, the problem's class skewness means that a learner is much more likely to be punished for predicting minority classes, which are more difficult to predict with high precision.

The fact that it makes an attempt to solve the problem rather than relying on class skewness for good performance makes Duplicated Faulkner a more interesting baseline than either N-Gram or Always Agree Strongly, even though both technically outperform it w.r.t. micro F-score. Similarly, the statistically significant improvements in micro and macro F-

score our approach achieves over the best baselines are more impressive when taking the skewness problem into consideration.

The results of our approach, which has access to all four feature groups, are shown in row 5 of the table. It obtains micro and macro F-scores of 60.6% and 20.1%, which correspond to statistically significant relative error reductions over the best baselines of 11.3% and 5.3%, respectively.[11]

Recall that we turned on one of our knowledge-based features only when we believed there was strong evidence that an instance belonged to its associated class. To get an idea of how useful these features are, we calculate the precision, recall, and F-score that would be obtained for each class if we treated our knowledge-based features as heuristic classifiers. The respective precisions, recalls, and F-scores we obtained are: 0.66/0.28/0.40 (A+), 0.50/0.02/0.04 (A−), 0.00/0.00/0.00 (Neu), 0.50/0.01/0.02 (D−), and 0.63/0.31/0.42 (D+). Since the rule predictions are encoded as features for the learner, they may not necessarily be used by the learner even if the underlying rules are precise. For instance, despite the rule's high precision on the Agree Somewhat class, the learner did not make use of its predictions due to its low coverage.

7.5.3 Additional Experiments

Since all the systems we examined fared poorly on identifying Somewhat classes, one may wonder how these systems would perform if we considered a simplified version of the task where we merged each Somewhat class with the corresponding Strongly class. In particular, since Faulkner's approach was originally not designed to distinguish between Strongly and Somewhat classes, it may seem fairer to compare our approach against Duplicated Faulkner on the four-class essay stance classification task, where stance can take one of four values:

[11] All significance tests are approximate randomization tests with $p < 0.01$. Boldfaced results are significant w.r.t. micro F-score for the Always Agree Strongly baseline, and macro F-score w.r.t. the Duplicated Faulkner baseline.

Agree (created by merging Agree Strongly and Agree Somewhat), Disagree (created by merging Disagree Strongly and Disagree Somewhat), Neutral, and Never Addressed.

In the results for the different systems on this four-class stance classification task, shown in Table 7.4, we see that the same patterns we noticed in the six-class version of the task persist. The approaches' relative order w.r.t. micro and macro F-score remains the same, though they are adjusted upwards due to the problem's increased simplicity. Our approach's performance on Agree increases (compared to Agree Strongly) because Agree is a bigger class, making predictions of the class safer. Our approach's performance decreases on Disagree (compared to Disagree Strongly) since it is not good at predicting Disagree Somewhat instances which are part of the class.

Table 7.4. Cross-validation results for four-class essay stance classification for Agree (A), Neutral (Neu), Disagree (D), and Never Addressed (Nev).

	System	Micro-F	Macro-F	A	Neu	D	Nev
1	Always Agree Strongly	64.8	19.7	78.7	.0	.0	.0
2	N-Gram	64.3	19.7	78.2	.0	.8	.0
3	Duplicated Faulkner	62.3	25.1	75.1	.0	25.2	.0
4	N-Gram + Duplicated Faulkner	62.6	23.7	75.8	.0	19.0	.0
5	Our approach	**67.6**	**29.1**	78.5	.0	38.0	.0

7.5.4 Error Analysis

To gain additional insights into our approach, we analyze its six major sources of error below.

Stances not presented in a straightforward manner. As an example, consider "To my opinion this technological progress triggers off the imagination in a certain way." To identify this sentence as strongly disagreeing with the proposition "there is no longer a place for dreaming and imagination", we need to understand (1) the world knowledge that technological progress is occurring, (2) that "triggers off the imagination in a certain way" means

that the technological progress coincides with imagination occurring, (3) that if imagination is occurring, there must be "a place for dreaming and imagination", and (4) that the prompt part is negated. In general, in order to construct reliable features to increase our coverage of essays that express their stance like this, we would need additional world knowledge and a deeper understanding of the text.

Rhetorical statements occasionally misidentified as stancetaking. For example, our method for identifying stancetaking paths misidentifies "I am going to discuss the topic that television is the opium of the masses in modern society" as stancetaking. To handle this, we need to incorporate more sophisticated methods for detecting rhetorical statements than those we are using (e.g., ignoring sentences ending in question marks).

Negation expressed without negation words. Our techniques for capturing negation are unable to detect when negation is expressed without the use of simple negation words. For example, "In this sense money is the root of life" should strongly disagree with "money is the root of all evil". The author replaced "life" with "evil", and detecting that this constitutes something like negation would require semantic knowledge about words that are somehow opposite of each other.

Insufficient feature/heuristic coverage of the Disagree Strongly class. Our stancetaking path-based features that we identified as intuitively having a connection to the Disagree Strongly class together cover only 51% of Disagree Strongly instances, meaning that it is in principle impossible for our system to identify the remaining 49%. However, our decision to incorporate only features that are expected to have fairly high precision for some class was intentional, as the lesson we learned from the Faulkner-based system is that it is difficult to learn a good classifier for stance classification using a large number of weakly or non-predictive features. To solve this problem, we would therefore need to exploit other aspects of strongly disagreeing essays that act as reliable predictors of the class.

Rarity of minority class instances. It is perhaps not surprising that our learning-based approach performs poorly on the minority classes. Even though the knowledge-based features were designed in part to improve the prediction of minority classes, our results suggest that the resulting features were not effectively exploited by the learners. To address this problem, one could employ a hybrid rule-based and learning-based approach where we use our machine-learned classifier to classify an instance only if it cannot be classified by any of these rules.

Lack of obvious similarity between instances of the same class. For example, if the most straightforward stancetaking sentence in an Agree Somewhat instance reads something like this, "In conclusion, I will not go to such extremes as to declare nihilistically that university does not prepare me for the real world in the least", (given the prompt part "Most university degrees do not prepare us for real life"), and we somehow managed to identify the instance's class as Agree Somewhat, what would the instance have in common with other Agree Somewhat instances? Given the numerous ways of expressing a stance, we believe a deeper understanding of essay text is required in order automatically detect how instances like this are similar to instances of the same class, and such similarities are required for learning in general.

7.6 Summary

We examined the new task of fine-grained essay stance classification, in which we determine stance for each prompt part and allow stance to take one of six values. We addressed this task by proposing two novel types of features, stancetaking path-based features and knowledge-based features. In an evaluation on 826 argumentative essays, our learning-based approach, which combines our novel features with n-gram features and Faulkner's features, significantly outperformed four baselines, including our re-implementation of Faulkner's system. Compared to the best baselines, our approach yielded relative error reductions of 11.3%

and 5.3%, in micro and macro F-score, respectively. Nevertheless, accurately predicting the Somewhat, Neutral, and Never Addressed stances remains a challenging task. To stimulate further research on this task, we make all of our stance annotations publicly available.

CHAPTER 8

ARGUMENTATION MINING WITH INTEGER LINEAR PROGRAMMING[1]

8.1 Introduction

There has been a surge of interest in argumentation mining in recent years. Argumentation mining typically involves addressing two subtasks: (1) *argument component identification* (ACI), which consists of identifying the locations and types of the components that make up the arguments (i.e., Major Claims, Claims, and Premises), and (2) *relation identification* (RI), which involves identifying the type of relation that holds between two argument components (i.e., Support, Attack, None). As a first step towards mining arguments in persuasive essays, Stab and Gurevych (S&G) annotated a corpus of 90 student essays with argument components and their relations (Stab and Gurevych, 2014a). To illustrate, consider the following excerpt from one essay:

> From this point of view, I firmly believe that (1) we should attach more importance to cooperation during primary education. First of all, (2) through cooperation, children can learn about interpersonal skills which are significant in the future life of all students. (3) What we acquired from team work is not only how to achieve the same goal with others but more importantly, how to get along with others.

In this example, premise (3) supports claim (2), which in turn supports major claim (1).

Using their annotated corpus, S&G presented initial results on *simplified* versions of the ACI and RI tasks (Stab and Gurevych, 2014b). Specifically, they applied their learned ACI classifier to classify only *gold* argument components (i.e., text spans corresponding to a Major Claim, Claim, or Premise in the gold standard) or sentences that contain no gold argument components (as *non-argumentative*). Similarly, they applied their learned RI classifier to classify only the relation between two *gold* argument components. In other words,

[1]This chapter was previously published in Persing and Ng (2016a).

they simplified both tasks by avoiding the challenging task of identifying the locations of argument components. Consequently, their approach cannot be applied in a realistic setting where the input is an *unannotated* essay.

Motivated by this weakness, we examine in this chapter argument mining in persuasive student essays in a considerably more challenging setting than that of S&G: the *end-to-end* setting. In other words, we perform argument mining on raw, unannotated essays. Our work makes three contributions. First, we present the first results on end-to-end argument mining in student essays using a *pipeline* approach, where the ACI task is performed prior to the RI task. Second, to avoid the error propagation problem inherent in the pipeline approach, we perform *joint inference* over the outputs of the ACI and RI classifiers in an Integer Linear Programming (ILP) framework (Roth and Yih, 2004), where we design constraints to enforce global consistency. Finally, we argue that the typical objective function used extensively in ILP programs for NLP tasks is not ideal for tasks whose primary evaluation metric is F-score, and subsequently propose a novel objective function that enables F-score to be maximized directly in an ILP framework. We believe that the impact of our work goes beyond argument mining, as our F-score optimizing objective function is general enough to be applied to any ILP-based joint inference tasks.

8.2 Related Work

Recall that identifying argumentative discourse structures consists of (1) identifying the locations and types of the argument components, and (2) identifying how they are related to each other. Below we divide related works into five broad categories based on which of these subtasks they addressed.

Argument location identification. Works in this category aimed to classify whether a sentence contains an argument or not (Florou et al., 2013; Moens et al., 2007; Song et al., 2014; Swanson et al., 2015).

Argument component typing. Works in this category focused on determining the *type* of an argument. The vast majority of previous works perform argument component typing at the *sentence* level. For instance, Rooney et al. (2012) classified sentences into premises, conclusions, premise-conclusions, and non-argumentative components; Teufel (1999) classified each sentence into one of seven rhetorical classes (e.g., claim, result, purpose); Burstein et al. (2003), Ong et al. (2014), and Falakmasir et al. (2014) assigned argumentative labels (e.g., claim, thesis, conclusion) to an essay's sentences; Levy et al. (2014) detected sentences that support or attack an article's topic; Lippi and Torroni (2015; 2016) detected sentences containing claims; and Rinott et al. (2015) detected sentences containing evidence for a given claim. or where in the sentence they are.

Argument location identification and typing. Some works focused on the more difficult task of *clause-level* argument component typing (Park and Cardie, 2014; Goudas et al., 2015; Sardianos et al., 2015), training a Conditional Random Field to jointly identify and type argument components.

Relation identification. Nguyen and Litman (2016) showed how context can be exploited to identify relations between argument components.

Argument component typing and relation identification. Given the difficulty of clause-level argument component location identification, recent argument mining works that attempted argument component typing and relation identification are not end-to-end. Specifically, they simplified the task by assuming as input *gold* argument components (Stab and Gurevych, 2014b; Peldszus and Stede, 2015).

End-to-end argument mining. Besides Persing and Ng (2016a), Palau and Moens (2009) addressed all the argument mining subtasks. They employed a hand-crafted context-free grammar (CFG) to generate (i.e., extract and type) argument components at the clause

level and identify the relations between them. A CFG approach is less appealing in the essay domain because (1) constructing a CFG is a time- and labor-intensive task, (2) which would be more difficult in the less-rigidly structured essay domain, which contains fewer rhetorical markers indicating component types (e.g., words like "reject", or "dismiss" which indicate a legal document's conclusion); and (3) about 20% of essay arguments' structures are non-projective (i.e., when mapped to the ordered text, their argument trees have edges that cross), and thus cannot be captured by CFGs.

8.3 Corpus

Our corpus consists of 90 persuasive essays collected and annotated by S&G. Some relevant statistics are shown in Table 8.1. Each essay is an average of 4.6 paragraphs (18.6 sentences) in length and is written in response to a topic such as "should high school make music lessons compulsory?" or "competition or co-operation-which is better?".

The corpus annotations describe the essays' argument structure, including the locations and types of the components that make up the arguments, and the types of relations that hold between them. The three annotated argument component types include: **Major Claims**, which express the author's stance with respect to the essay's topic, **Claims**, which are controversial statements that should not be accepted by readers without additional support, and **Premises**, which are reasons authors give to persuade readers about the truth of another argument component statement. The two relation types include: **Support**, which indicates that one argument component supports another, and **Attack**, which indicates that one argument component attacks another.

8.4 Pipeline-Based Argument Mining

Next, we describe our end-to-end *pipeline* argument mining system, which will serve as our baseline. In this system, ACI is performed prior to RI.

Table 8.1. Argument mining corpus statistics (v1).

Essays: 90	Paragraphs: 417	Sentences: 1,673
Major Claims: 90	Claims: 429	Premises: 1,033
Support Relations: 1,312		Attack Relations: 161

8.4.1 Argument Component Identification

We employ a two-step approach to the ACI task, where we first heuristically extract *argument component candidates* (ACCs) from an essay, and then classify each ACC as either a premise, claim, major claim, or non-argumentative, as described below.

Extracting ACCs

We extract ACCs by constructing a set of low precision, high recall heuristics for identifying the locations in each sentence where an argument component's boundaries might occur. The majority of these rules depend primarily on a syntactic parse tree we automatically generated for all sentences in the corpus using the Stanford CoreNLP (Manning et al., 2014) system. Since argument components are a clause-level annotation and therefore a large majority of annotated argument components are substrings of a simple declarative clause (an S node in the parse tree), we begin by identifying each S node in a sentence's tree.

Given an S clause, we collect a list of left and right boundaries where an argument component may begin or end. The rules we used to find these boundaries are summarized in Table 8.2. We then construct ACCs by combining each left boundary with each right boundary that occurs after it. As a result, we are able to find exact (boundaries exactly match) and approximate (over half of tokens shared) matches for 92.1% and 98.4% respectively of all ACs.

[1] An additional point that requires explanation is that the last two right boundary rules mention "possibly nested" nodes. In boundary rule 7, for example, this means that the S node might end in a PP node, which itself has a PP node as its last child, and so on. We generate a separate right boundary immediately before each of these PP nodes.

Table 8.2. Rules for extracting ACC boundary locations.

(a) Potential left boundary locations

#	Rule
1	Exactly where the S node begins.
2	After an initial explicit connective, or if the connective is immediately followed by a comma, after the comma.
3	After nth comma that is an immediate child of the S node.
4	After nth comma.

(b) Potential right boundary locations

#	Rule
5	Exactly where the S node ends, or if S ends in a punctuation, immediately before the punctuation.
6	If the S node ends in a (possibly nested) SBAR node, immediately before the nth shallowest SBAR.[1]
7	If the S node ends in a (possibly nested) PP node, immediately before the nth shallowest PP.

Training the ACI Classifier

We train a classifier for ACI using MALLET's (McCallum, 2002) implementation of maximum entropy classification. We create a training instance from each ACC extracted above. If the ACC's left and right endpoints exactly match an annotated argument component's, the corresponding training instance's class label is the same as that of the component. Otherwise, its class label is "non-argumentative". Each training instance is represented using S&G's structural, lexical, syntactic, indicator, and contextual features for solving the same problem. Briefly, the structural features describe an ACC and its covering sentence's length, punctuations, and location in the essay. Lexical features describe the $1-3$ grams of the ACC and its covering sentence. Syntactic features are extracted from the ACC's covering sentence's parse tree and include things such as production rules. Indicator features describe any explicit connectives that immediately precede the ACC. Contextual features describe

the contents of the sentences preceding and following the ACC primarily in ways similar to how the structural features describe the covering sentence.

8.4.2 Relation Identification

We consider RI between pairs of argument components to be a five class classification problem. Given a pair of ACCs A_1 and A_2 where A_1 occurs before A_2 in the essay, either they are unrelated, A_1 supports A_2, A_2 supports A_1, A_1 attacks A_2, or A_2 attacks A_1. Below we describe how we train and apply our classifier for RI.

We learn our RI classifier using MALLET's implementation of maximum entropy classification. Each training instance, which we call a training *relation candidate* (RC), consists of a pair of ACCs and one of the above five labels. By default, the instance's label is "no relation" unless each ACC has the exact boundaries of a gold standard argument component and one of the remaining four relations holds between the two gold argument components.

We create training RCs as follows. We construct RCs corresponding to true relations out of all pairs of argument components in a training essay having a gold relation. As the number of potential RCs far exceeds the number of gold relations in an essay, we undersample the "no relation" class in the following way. Given a pair of argument components A and B between which there is a gold relation, we define A_p to be the closest previous ACC in the essay as generated in Section 8.4.1 such that A_p's text doesn't overlap with A. We also define A_s as the closest succeeding ACC after A such that A_s and A do not overlap. We define B_p and B_s similarly with respect to B. From these ACCs, we generate the the four instances (A_p, B), (A_s, B), (A, B_p), and (A, B_s), all of which have the "no relation" label, as long as the pairs' text sequences do not overlap. We believe the resulting "no relation" training instances are informative since each one is "close" to a gold relation. We represent each instance using S&G's structural, lexical, syntactic, and indicator features for solving the same problem. Briefly, RC structural features describe many of the same things about each

ACC as did the ACC structural features, though they also describe the difference between the ACCs (e.g., the difference in punctuation counts). Lexical features consist primarily of the unigrams appearing in each ACC and word pairs, where each word from one ACC is paired with each word from the other. Syntactic and indicator features encode the same information about each ACC as the ACC syntactic and indicator features did.

We now apply the classifier to test essay RCs, which are created as follows. Given that this is a pipelined argument mining system, in order to ensure that the RI system's output is consistent with that of the ACI system, we generate test RCs from all possible pairs of ACCs that the ACI system predicted are real components (i.e., it labeled them something other than "non-argumentative").

8.5 Joint Inference for Argument Mining

In this section we discuss our joint inference approach to argument mining.

8.5.1 Motivation

There are two major problems with the pipeline approach described in the previous section. First, many essay-level within-task constraints are not enforced. For instance, the ACI task has the constraint that each essay has exactly one major claim, and the RI task has the constraint that each claim has no more than one parent. This problem arises because our ACI and RI classifiers, like those of S&G, classify each ACI and RI test instance independently of other test instances. We propose to enforce such essay-level within-task constraints in an ILP framework, employing ILP to perform joint inference over the outputs of our ACI and RI classifiers so that the resulting classifications satisfy these constraints.[2]

[2]Note that while we partition *documents* into folds in our cross-validation experiments, S&G partition *instances* into folds. Hence, S&G's evaluation setting prevents them from enforcing essay-level constraints in addition to being unrealistic in practice.

The second problem with the pipeline approach is that errors made early on in the pipeline propagate. For instance, assume that a Support relation exists between two argument components in a test essay. If the pipeline system fails to (heuristically) extract one or both of these argument components, or if it successfully extracts them but misclassifies one or both of them as non-argumentative, then the pipeline system will not be able to identify the relationship between them because no test RCs will be created from them. The above problem arises because the RI classifier assumes the *most probable* output of the ACI classifier for each ACC as input. Hence, one possible solution to this problem is to make use of the *n-best* outputs of the ACI classifier for each argument component type, as this increases the robustness of the pipeline to errors made by the ACI classifier.

We obtain the n-best ACI outputs and employ them as follows. Recall that the ACI system uses a maximum entropy classifier, and therefore its output for each ACC is a list of probabilities indicating how likely it is that the ACC belongs to each of the four classes (premise, claim, major claim, or non-argumentative). This means that it is possible to rank all the ACCs in a text by these probabilities. We use this idea to identify (1) the 3 most likely premise ACCs from each sentence, (2) the 5 most likely claim ACCs from each paragraph, and (3) the 5 most likely major claim ACCs from each essay.[3] Given these most likely ACC lists, we combine pairs of ACCs into test RCs for the RI classifier in the following way. As long as the ACCs do not overlap, we pair (1) each likely premise ACC with every other likely ACC of any type occurring in the same paragraph, and (2) each likely claim ACC with each likely major claim ACC. We then present these test RCs to the RI classifier normally, making no other changes to how the pipeline system works.

Employing n-best ACI outputs, however, introduces another problem: the output of the RI classifier may no longer be *consistent* with that of the ACI classifier because the RI

[3]We select more premise than claim ACCs (3 per sentence vs 5 per paragraph) because the corpus contains over twice as many premises as claims. We select major claim ACCs only from the first and last paragraph because major claims never occur in middle paragraphs.

classifier may posit a relationship between two ACCs that the ACI classifier labeled non-argumentative. To enforce this cross-task consistency constraint, we also propose to employ ILP. The rest of this section details our ILP-based joint inference approach for argument mining, which addresses both of the aforementioned problems with the pipeline approach.

8.5.2 Basic ILP Approach

We perform joint inference over the outputs of the ACI and RI classifiers by designing and enforcing within-task and cross-task constraints in the ILP framework. Specifically, we create one ILP program for each test essay, as described below.

Let Xn_i, Xp_i, Xc_i, and Xm_i be binary indicator variables representing whether the ILP solver believes ACC i has type none, premise, claim, and major claim, respectively. Let Cn_i, Cp_i, Cc_i, and Cm_i be the probabilities that ACC i has type none, premise, claim, and major claim, respectively, as dictated by the ACI maximum entropy classifier's output.[4] Let a be the count of ACCs.

Let $Yn_{i,j}$, $Ys_{i,j}$, $Ya_{i,j}$, $Yrs_{i,j}$, and $Yra_{i,j}$ be binary indicator variables representing whether the ILP solver believes ACCs i and j have no relation, i is supported by j, i is attacked by j, j is supported by i, and j is attacked by i, respectively, where (i, j) appears in the set of RCs B that we presented to the RI system as modified in Section 8.5.1. We assume all other ACC pairs have no relation. Let $Dn_{i,j}$, $Ds_{i,j}$, $Da_{i,j}$, $Drs_{i,j}$, and $Dra_{i,j}$ be the probabilities that component candidates i and j have no relation, i is supported by j, i is attacked by j, j is supported by i, and j is attacked by i, respectively, as dictated by the modified RI classifier described in Section 8.5.1.[4]

Given these definitions and probabilities, our ILP program's default goal is to find an assignment of these variables X and Y in order to maximize $P(X) + P(Y)$, where:

[4]We additionally reserve .001 probability mass to distribute evenly among Cn_i, Cp_i, Cc_i, and Cm_i (or $Dn_{i,j}$, $Ds_{i,j}$, $Da_{i,j}$, $Drs_{i,j}$, and $Dra_{i,j}$) to prevent math errors involving taking the log of 0 which might otherwise occur in the formulas below.

$$P(X) = \frac{1}{a}\sum_{i=1}^{a} \log(Cn_i Xn_i + Cp_i Xp_i + Cc_i Xc_i + Cm_i Xm_i) \tag{8.1}$$

$$P(Y) = \frac{1}{|B|}\sum_{(i,j)\in B} \log(Dn_{i,j} Yn_{i,j} + Ds_{i,j} Ys_{i,j} + Da_{i,j} Ya_{i,j} + Drs_{i,j} Yrs_{i,j} + Dra_{i,j} Yra_{i,j}) \tag{8.2}$$

subject to the *integrity* constraints that: (8.3) an ACC is either not an argument component or it has exactly one of the real argument component types, (8.4) a pair of component candidates (i, j) must have exactly one of the five relation types, and (8.5) if there is a relation between ACCs i and j, i and j must each be real components.[5]

$$Xn_i + Xp_i + Xc_i + Xm_i = 1 \tag{8.3}$$

$$Yn_{i,j} + Ys_{i,j} + Ya_{i,j} + Yrs_{i,j} + Yra_{i,j} = 1 \tag{8.4}$$

$$(Xp_i + Xc_i + Xm_i) + (Xp_j + Xc_j + Xm_j) - 2(Ys_{i,j} + Ya_{i,j} + Yrs_{i,j} + Yra_{i,j}) \geq 0 \tag{8.5}$$

In the objective function, a and $|B|$ serve to balance the contribution of the two tasks, preventing one from dominating the other.

8.5.3 Enforcing Consistency Constraints

So far we have described integrity constraints, but recall that our goal is to enforce consistency by imposing within-task and cross-task constraints, which force the ILP solutions to

[5]Note that because of previous integrity constraints, the first term in constraint 8.5 is 1 only if we predict that i is a real component, the second term is 1 only if we predict that j is a real component, and the third term is -2 only if we predict that there is a relationship between them. Otherwise, each term is 0. Thus term 3 prevents us from predicting a relationship between i and j unless the first and second terms cancel it out through predicting that i and j are real components.

more closely resemble real essay argument structures. Our consistency constraints fall into four categories.

Our constraints on *major claims* are that: (8.6) there is exactly one major claim in each essay, (8.7) major claims always occur in the first or last paragraph, and (8.8) major claims have no parents.

$$\sum_{i=1}^{a} Xm_i = 1 \tag{8.6}$$

$$Xm_i = 0 \mid i \notin first\, or\, last\, paragraph \tag{8.7}$$

$$\begin{gathered} Ys_{i,j} + Xm_j \leq 1\ ,\ Ya_{i,j} + Xm_j \leq 1 \\ Yrs_{i,j} + Xm_i \leq 1\ ,\ Yra_{i,j} + Xm_i \leq 1 \end{gathered} \tag{8.8}$$

Our constraints on *premises* are that: (8.9) a premise has at least one parent, and (8.10) a premise is related only to components in the same paragraph.

$$\begin{aligned} &\sum_{\{i|(i,j)\in B\}} (Ys_{i,j} + Ya_{i,j}) \\ +\ &\sum_{\{k|(j,k)\in B\}} (Yrs_{j,k} + Yra_{j,k}) - Xp_j \geq 0 \end{aligned} \tag{8.9}$$

$$\begin{gathered} for\ i < j,\ i \notin Par(j) : Xp_j - Yn_{i,j} \leq 0 \\ for\ k > j,\ k \notin Par(j) : Xp_j - Yn_{j,k} \leq 0 \end{gathered} \tag{8.10}$$

where $i < j$ and $j < k$ mean ACC i appears before j, and j appears before k, and $Par(j)$ is the set of ACCs in j's covering paragraph.

Our constraints on *claims* state that: (8.11) a claim has no more than one parent[6], and (8.12) if a claim has a parent, that parent must be a major claim.

[6]The coefficient of Xc_j is equal to the number of terms in the two summations on the left hand side of the equation. The intention is that component j's claim status should have equal weight with the relations it might potentially participate in, so it is possible for j to participate as a child in a relation with all other available components unless it is a claim, in which case it can participate in at most one relation as a child.

$$\begin{aligned}
&\Big(\Big|\{i|(i,j) \in B\}\Big| + \Big|\{k|(j,k) \in B\}\Big|\Big) Xc_j \\
&\qquad + \sum_{\{i|(i,j)\in B\}} (Ys_{i,j} + Ya_{i,j}) \\
&\qquad + \sum_{\{k|(j,k)\in B\}} (Yrs_{j,k} + Yra_{j,k}) \\
&\leq \Big|\{i|(i,j) \in B\}\Big| + \Big|\{k|(j,k) \in B\}\Big| + 1
\end{aligned} \tag{8.11}$$

$$\begin{aligned}
for\ j < k,\ Xm_k - Xc_j - Yrs_{j,k} - Yra_{j,k} \geq -1 \\
for\ j > i,\ Xm_i - Xc_j - Ys_{i,j} - Ya_{i,j} \geq -1
\end{aligned} \tag{8.12}$$

The last category, which comprises constraints that do not fit well into any other category, are: (8.13) the boundaries of actual components never overlap, (8.14) each paragraph must have at least one claim or major claim, and (8.15) each sentence may have at most two argument components.[7]

$$for\ i\ overlaps\ j\ ,\ Xn_i + Xn_j \geq 1 \tag{8.13}$$

$$\forall\ paragraphs\ P\ :\ \sum_{i \in P} Xc_i + Xm_i \geq 1 \tag{8.14}$$

$$\forall\ sentences\ S\ :\ \sum_{i \in S} Xp_i + Xc_i + Xm_i \leq 2 \tag{8.15}$$

We solve each ILP program using Gurobi.[8]

[7]Unlike the other constraints, the last two constraints are only mostly true: 5% of paragraphs have no claims or major claims, and 1.2% of sentences have 3 or more components.

[8]`http://www.gurobi.com`

8.5.4 F-score Maximizing Objective Function

The objective function we employ in the previous subsection attempts to maximize the average probability of correct assignment of variables over the ACI and RI problems. This kind of objective function, which aims to maximize classification accuracy, was originally introduced by Roth and Yih (2004) in their seminal ILP paper, and has since then been extensively applied to NLP tasks. However, it is arguably not an ideal objective function for our task, where F-score rather than classification accuracy is used as the evaluation metric.

In this section, we introduce a novel method for constructing an ILP objective function that directly maximizes the average F-score over the two problems. Recall that F-score can be simplified to:

$$F = \frac{2TP}{2TP + FP + FN} \tag{8.16}$$

where TP, FP, and FN are the counts of true positives, false positives, and false negatives respectively. Unfortunately, we cannot use this equation for F-score in an ILP objective function for two reasons. First, this equation involves division, which cannot be handled using ILP since ILP can only handle linear combinations of variables. Second, TP, FP, and FN need to be computed using gold annotations, which we don't have in a test document. We propose to instead maximize F by maximizing the following:

$$G = \alpha 2TP_e - (1 - \alpha)(FP_e + FN_e) \tag{8.17}$$

where TP_e, FP_e, and FN_e, are *estimated* values for TP, FP, and FN respectively, and α attempts to balance the importance of maximizing the numerator vs minimizing the denominator.[9] We ignore the $2TP$ term in the denominator because minimizing it would directly reduce the numerator.

[9]We tune α on the development set, allowing it to take any value from 0.7, 0.8, or 0.9, as this range tended to perform well in early experiments.

To maximize average F-score, we can therefore attempt to maximize the function $\frac{G_c+G_r}{2}$, where G_c and G_r are the values of G in equation 8.17 as calculated using the estimated values from the ACI and RI problem respectively.

The question that still remains is, how can we estimate values for TP, FP, and FN mentioned in Equation 8.17? Our key idea is inspired by the E-step of the Expectation-Maximization algorithm (Dempster et al., 1977): while we cannot compute the actual TP, FP, and FN due to the lack of gold annotations, we can compute their *expected* values using the probabilities returned by the ACI and RI classifiers. Using the notation introduced in Section 5.2, the expected TP, FP, and FN values for the ACI task can be computed as follows:

$$TP_e = \sum_{i,g} Cg_i Xg_i \tag{8.18}$$

$$FP_e = \sum_{i,g} (1 - Cg_i) Xg_i \tag{8.19}$$

$$FN_e = \sum_{i,g} \left(Xg_i \sum_{h \neq g} Ch_i \right) + \sum_i Xn_i (1 - Cn_i) \tag{8.20}$$

where g and h can be any argumentative class from the ACI problem (i.e., premise (p), claim (c), or major claim (m)). The formulas we use to calculate TP_e, FP_e, and FN_e for the RI problem are identical except C is replaced with D, X is replaced with Y, and g and h can be any class from the RI problem other than no-relation.

8.6 Evaluation

In this section we discuss how we evaluate our joint inference approach to argument mining.

8.6.1 Experimental Setup

Corpus. As mentioned before, we use as our corpus the 90 essays annotated with argumentative discourse structures by S&G. All of our experiments are conducted via five-fold

cross-validation on this corpus. In each fold experiment, we reserve 60% of the essays for training, 20% for development (selecting features and tuning α), and 20% for testing.

Evaluation metrics. To calculate F-score on each task using Equation 8.16, we need to explain what constitutes a true positive, false positive, or false negative on each task. Given that j is a true argument component and i is an ACC, the formulas for the ACI task are:

$$TP = \Big|\{j \mid \exists i \; |gl(j) = pl(i) \wedge i \doteq j\}\Big| \tag{8.21}$$

$$FP = \Big|\{i| \; pl(i) \neq n \wedge \nexists j \mid gl(j) = pl(i) \wedge i \doteq j\}\Big| \tag{8.22}$$

$$FN = \Big|\{j \mid \nexists i \; |gl(j) = pl(i) \wedge i \doteq j\}\Big| \tag{8.23}$$

where $gl(j)$ is the gold standard label of j, $pl(i)$ is the predicted label of i, n is the non-argumentative class, and $i \doteq j$ means i is a *match* for j. i and j are considered an *exact match* if they have exactly the same boundaries, whereas they are considered an *approximate match* if they share over half their tokens.

We perform most of our analysis on approximate match results rather than exact match results as it can be difficult even for human annotators to identify exactly the same boundaries for an argument component.[10] We use the same formulas for calculating these numbers for the RI problem except that j and i represent a true relation and an RC respectively, two relations approximately (exactly) match if both their source and target ACCs approximately (exactly) match, and n is the no-relation class.

[10]Approximate match has been used in evaluating opinion mining systems (e.g., Choi et al. (2006), Yang and Cardie (2013)), where researchers have also reported difficulties in having human annotators identify exactly the same boundaries for an opinion expression and its sources and targets. They have adopted an even more relaxed notion of approximate match: they consider two text spans an approximate match if they share at least one overlapping token.

8.6.2 Results and Discussion

Approximate and exact match results of the pipeline approach (BASE) and the joint approach (OUR) are shown in Table 8.3. As we can see, using approximate matching, OUR system achieves highly significant[11] improvements over the pipelined baseline system by a variety of measures.[12] The most important of these improvements is shown in the last column, where our system outperforms the baseline by 13.9% absolute F-score (a relative error reduction of 18.5%). This is the most important result because it most directly measures our performance in pursuit of our ultimate goal, to maximize the average F-score over both the ACI and RI problems. The highly significant improvements in other measures, particularly the improvements of 13.2% and 14.6% in ACI and RI F-score respectively, follow as a consequence of this maximization. Using exact matching, the differences in scores between OUR system and BASE's are smaller and highly significant with respect to a smaller number of measures. In particular, under Exact matching OUR system's performances on the RI-P and RI-R metrics are significant ($p < 0.02$), while under Approx matching, they are highly significant ($p < 0.002$).

8.6.3 Ablation Results

To analyze the performance gains yielded by each improvement to our system, we show ablation results in Table 8.4. Each row of the table shows the results of one ablation experiment on the test set. That is, we obtain them by removing exactly one feature set or improvement type from our system.

[11] Boldfaced results in Table 8.3 are highly significant ($p < 0.002$, paired t-test) compared to the baseline.

[12] All the results in Tables 8.3 and 8.4 are averaged across five folds, so it is not true that $F_{avg} = \frac{2P_{avg}R_{avg}}{P_{avg}+R_{avg}}$. Our F-score averaging method is preferable to calculating F-scores using the above formula because the formula can be exploited to give artificially inflated F-scores by alternating between high precision, low recall, and low precision, high recall labelings on different folds.

Table 8.3. Five-fold cross-validation average percentages for argument component identification (ACI) and relation identification (RI) for OUR system and the pipeline-based BASELine system. Column abbreviations are Major Claim F-score (MC-F), Claim F-score (C-F), Premise F-score (P-F), Precision (P), Recall (R), F-score (F), Support F-score (S-F), and Attack F-score (A-F).

		ACI						RI					Avg
	System	MC-F	C-F	P-F	P	R	F	S-F	A-F	P	R	F	F
Approx	BASE	11.1	26.9	51.9	64.0	33.6	44.0	6.1	0.8	5.7	6.2	5.8	24.9
	OUR	22.2	**42.6**	**66.0**	56.6	**57.9**	**57.2**	**21.3**	1.1	**16.8**	**28.0**	**20.4**	**38.8**
Exact	BASE	7.4	24.2	43.2	50.4	29.6	37.3	4.4	0.8	4.1	4.7	4.3	20.8
	OUR	16.9	**37.4**	**53.4**	47.5	**46.7**	**47.1**	**13.6**	0.0	12.7	15.4	**12.9**	**30.0**

The **B**aseline feature sets we remove include those for the A**C**I task (C_b) from Section 8.4.1 and those for the **R**I task (R_b) from Section 8.4.2.[13] The **I**LP improvement sets we remove are the **D**efault ILP (I_d) system[14] from Section 8.5.2, the **M**ajor claim (I_m), **P**remise (I_p), **C**laim (I_c), and **O**ther (I_o) constraints from Section 8.5.3 Equations 8.6−8.8, 8.9−8.10, 8.11−8.12, and 8.13−8.15 respectively, and the **F**-score maximizing objective function from Section 8.5.4.

Broadly, we see from the last column that all of our improvement sets are beneficial (usually significantly[15]) to the system, as performance drops with their removal. Notice also that whenever removing an ILP improvement set harms average F-score, it also simultaneously harms ACI and RI F-scores, usually significantly. This holds true even when the improve-

[13]When we remove a baseline feature set, we represent each instance to the corresponding classifier using no features. As a result, the classifier's predictions are based solely on the frequency of the classes seen during training.

[14]Note that removing the default ILP system (I_d) necessitates simultaneously removing all other ILP-related improvements. Thus, a system without it is equivalent to BASE, but with RCs generated as described in Section 8.5.1.

[15]Boldfaced results are significantly lower than ALL, the system with all improvements left intact, with $p < 0.05$.

Table 8.4. How OUR system performs on one development set as measured by percent Precision, Recall, and F-score if each improvement or feature set is removed.

Mod	ACI			RI			Avg
	P	R	F	P	R	F	F
ALL	54.9	58.8	56.7	16.7	26.4	20.4	38.6
C_b	**44.0**	**38.5**	**40.9**	14.2	**14.8**	**14.4**	**27.7**
R_b	57.9	58.7	58.2	16.6	24.0	18.2	38.2
I_d	64.0	**33.6**	**44.0**	**6.6**	**21.8**	**10.1**	**27.0**
I_m	**44.6**	**46.7**	**45.6**	**14.7**	27.8	19.2	**32.4**
I_p	51.5	58.4	54.7	**13.2**	26.1	**17.3**	36.0
I_c	**49.0**	**51.5**	**50.2**	**14.2**	27.9	18.8	**34.5**
I_o	**40.5**	65.9	**50.1**	**7.0**	29.2	**11.2**	**30.7**
I_f	61.1	**40.4**	**48.5**	22.3	**15.7**	**18.2**	**33.4**

ment set deals primarily only with one task (e.g., I_o for the ACI task), suggesting that our system is benefiting from joint inference over both tasks.

8.6.4 Error Analysis and Future Work

Table 8.3 shows that OUR system has more trouble with the RI task than the ACI task. A closer inspection of OUR system's RI predictions reveals that its low precision is mostly due to predicted relationships wherein one of the participating ACCs is not a true argument component. Since false positives in the ACI task have an outsized impact on RI precision, it may be worthwhile to investigate ILP objective functions that more harshly penalize false positive ACCs.

The RI task's poor recall has two primary causes. The first is false negatives in the ACI task. It is impossible for an RI system to correctly identify a relationship between two ACs if the ACI system fails to identify either one of them as an AC. We believe ACI recall, and by extension, RI recall, can be improved by exploiting the following observations. First, we noticed that many argument components OUR system fails to identify, regardless of their type, contain words that are semantically similar to words in the essay's topic (e.g.,

if the topic mentions "school", argument components might mention "students"). Hence, one way to improve ACI recall, and by extension, RI recall, would be to create ACI features using a semantic similarity measure such as the Wikipedia Link-based similarity measure (Witten and Milne, 2008). Second, major claims are involved in 32% of all relationships, but OUR system did an especially poor job at identifying them due to their scarcity. Since we noticed that major claims tend to include strong stancetaking language (e.g., words like "should", "must", and "believe"), it may be possible to improve major claim identification by constructing an arguing language lexicon as in Somasundaran and Wiebe (2010), then encoding the presence of any of these arguing words as ACC features.

The second major cause of OUR system's poor RI recall is its failure to identify relationships between two correctly extracted ACs. We noticed many of the missed relationships involve ACs that mention some of the same entities. Thus, a coreference resolver could help us build features that describe whether two ACCs are talking about the same entities.

8.7 Summary

We presented the *first* results on *end-to-end* argument mining in persuasive student essays using a pipeline approach, improved this baseline approach by designing and employing global consistency constraints to perform joint inference over the outputs of the tasks in an ILP framework and proposed a novel objective function that enables F-score to be maximized directly by an ILP solver. In an evaluation on Stab and Gurevych's corpus of 90 essays, our approach yields an 18.5% relative error reduction in F-score over the pipeline system.

CHAPTER 9

ARGUMENTATION MINING WITH STRUCTURED PERCEPTRONS

9.1 Introduction

As we discussed in the last chapter, argumentation mining typically involves addressing two subtasks: (1) *argument component identification* (ACI), which consists of identifying the locations and types of the components that make up the arguments (i.e., major claims, claims, and premises), and (2) *relation identification* (RI), which involves identifying the type of relation that holds between two argument components (i.e., support, attack, none). As an example, consider the following text segment taken from a corpus of student essays that Stab and Gurevych (S&G) annotated with argument components and their relations (Stab and Gurevych, 2016):

> In my view point, (1) I would agree to idea of taking a break before starting higher education. (2) Students who take break, gain lot of benefits by traveling around the places or by working. (3) They tend to contribute more due to their real world experiences which makes them more mature.

In this example, premise (3) supports claim (2), and (1) is a major claim.

Despite the large amount of recent work on computational argumentation, there is an area of argument mining where progress is fairly stagnant: the development of end-to-end argument mining systems. With the rarest exceptions, existing argument mining systems are *not* end-to-end. For example, using their annotated corpus, S&G trained an ACI classifier and applied it to classify only *gold* argument components (i.e., text spans corresponding to a major claim, claim, or premise in the gold standard) or sentences that contain no gold argument components (as *non-argumentative*). Similarly, they applied their learned RI classifier to classify only the relation between two *gold* argument components. In other words, they simplified both tasks by avoiding the challenging task of identifying the locations of argument components. Consequently, their approach cannot be applied in a realistic setting

where the input is an *unannotated* essay. Recent related work has focused on using joint inference (via Integer Linear Programming (ILP), for instance) to improve the ACI and RI classifications (Peldszus and Stede, 2015; Stab and Gurevych, 2016; Wei et al., 2017), but since they continue to make the same assumptions that S&G made, these systems remain non-end-to-end.

In this chapter, we propose a new approach to the challenging but under-investigated task of *end-to-end* argument mining in student essays by viewing it as a *structure learning* task. Specifically, we train a structured perceptron to directly induce the argument tree associated with each paragraph. Since a structured perceptron's training instance consist of a paragraph paired with the correct (i.e., hand-annotated) argument tree rather than individual or pairs of text segments that we need to make some decision about, it can encode information about the global structure of an argument tree. So for example, in structured perceptrons we can encode information about tree depth so we can learn how likely 3 is to have children based on how far it is from the tree's root (2) rather than just from information about 3 and any potential children. We can also learn that the tree where 2 is the parent of only one argument component (3) should be preferred to a tree where 2 is the parent of many children, as claims usually have only a small number of children.

We believe that structured perceptrons are conceptually a better approach to end-to-end argument mining than joint inference mechanisms such as ILP. One weakness of these joint inference mechanisms is that they can be viewed as stitching a lot of purely local decisions (i.e., decisions about whether a text segment like 1, 2, or 3 is an ACC and its type, and decisions about whether two ACCs are related and their relationship type) together to construct an argument tree satisfying some constraints. While joint inference systems could theoretically encode a non-local constraint against claims having a large number of children, or could painstakingly encode some preference for claims having small numbers of children in the objective function, structured perceptrons can automatically learn how many children are appropriate for a claim and how much this branching factor matters.

We compare our approach to the recent work on end-to-end argument mining in student essays by Persing and Ng (2016a) (P&N), who employed integer linear programming to coordinate the decisions made by an ACI classifier and those made by an RI classifier on heuristically extracted ACCs. In an evaluation on a corpus of 402 essays annotated by S&G, our structured perceptron significantly outperforms Persing and Ng's ILP approach by 4.7%.

9.2 Related Work

As the task we address in this chapter is nearly identical to the task addressed in the previous chapter, related work is discussed in Section 8.2.

9.3 Corpus

While the task we address in this chapter is nearly identical to that of the previous chapter, the corpus we use in this chapter is a newer version of the argumentation mining corpus introduced in Stab and Gurevych (2014a). More specifically, our corpus consists of 402 persuasive essays collected and annotated in Stab and Gurevych (2016). Some relevant statistics are shown in Table 9.1. Each essay is an average of 4.6 paragraphs (16.8 sentences) in length and is written in response to a topic such as "should high school make music lessons compulsory?" or "competition or co-operation-which is better?".

The corpus annotations describe the essays' argument structure, including the locations and types of the components that make up the arguments, and the types of relations that hold between them. The three annotated argument component types include: **major claims**, which express the author's stance with respect to the essay's topic, **claims**, which are controversial statements that should not be accepted by readers without additional support, and **premises**, which are reasons authors give to persuade readers about the truth of another argument component statement. The two relation types include: **support**, which

Table 9.1. Argument mining corpus statistics (v2).

Essays: 402	Paragraphs: 1,833	Sentences: 6,741
Major claims: 751	Claims: 1,506	Premises: 3,838
Support relations: 3,613	Attack relations: 219	

indicates that one argument component supports another, and **attack**, which indicates that one argument component attacks another.

9.4 Baselines

In this section, we describe our baseline approaches to the task of end-to-end argument mining.

9.4.1 Pipeline (PIPE)

Our first baseline system is a reimplementation of the pipeline-based baseline used by P&N for the argument mining task, adapted to account for changes in the annotation guidelines for version 2 of S&G's corpus. It is called a pipeline system because it solves the argument mining subtasks of ACI and RI sequentially. That is, it first identifies the argument components and their types from the essay, and then it identifies the relationships between them.

Argument Component Identification

PIPE's approach to the ACI subtask, in turn, also consists of two steps. In the first step, PIPE identifies a set of *argument component candidates* (ACCs), sequences of text that could potentially be argument components, from a paragraph. To do this, it first parses each of the paragraph's sentences using the Stanford CoreNLP toolkit (Manning et al., 2014). Then it applies a set of low precision, high recall heuristic rules developed by P&N based on the sentence's syntactic parse tree to extract the ACCs. Since argument components are

annotated at the clause level, an example of these rules states that an ACC can be extracted from the text of any simple declarative clause (an S node in the syntactic parse) beginning after the first comma and ending immediately before any terminal punctuation. The rules have sufficiently high recall that they are able to identify an ACC with the same exact boundaries as 92% of all argument components and an ACC with approximately the same boundaries as 98% of all argument components.

Given this set of extracted ACCs, PIPE trains a classifier for ACI using MALLET's (McCallum, 2002) implementation of maximum entropy classification to label the ACCs with argument component labels. Each ACC serves as an instance whose class label is the same as that of the argument component sharing its exact boundaries (major claim, claim, or premise). If there is no AC sharing the ACC's exact boundaries, it is labeled "non-argumentative".

Each instance is represented using S&G's structural, lexical, syntactic, indicator, and contextual features for solving (a simplified version of) the same problem. Briefly, the structural features describe an ACC and its covering sentence's length, punctuations, and location in the essay. Lexical features describe the $1-3$ grams of the ACC and its covering sentence. Syntactic features are extracted from the ACC's covering sentence's parse tree and include things such as production rules. Indicator features describe any explicit connectives that immediately precede the ACC. Contextual features describe the contents of the sentences preceding and following the ACC primarily in ways similar to how the structural features describe the covering sentence.

Relation Identification

After the classifier in Section 9.4.1 predicts which ACCs in a paragraph represent real argument components as well as their labels, PIPE is ready to determine how the ACCs are

related to each other. It considers RI between pairs of ACCs a five class classification problem. Given a pair of ACCs A_1 and A_2 where A_1 occurs before A_2 in the essay, either they are unrelated, A_1 supports A_2, A_2 supports A_1, A_1 attacks A_2, or A_2 attacks A_1.

To solve this problem, PIPE also learns an RI maximum entropy classifier. Each training instance, called a *relation candidate* (RC), consists of a pair of ACCs and one of the above five labels. By default, the instance's label is "no relation" unless each ACC has the exact boundaries of a gold standard argument component and one of the remaining four relations holds between the two gold argument components.

To train the classifier, an RC corresponding to each of the gold relations in all paragraphs is created as an instance of one of the four argumentative labels mentioned above. From this gold relation, PIPE also creates four additional "no relation" RCs by replacing each of the original relationship's participating ACs with nearby ACCs.

PIPE represents each RC using S&G's feature set for (a simpler version of) the same task. This feature set consists of structural, lexical, syntactic, and indicator features very similar to those it used for ACI, though adapted to deal with relationships between pairs of ACCs. For example, the ACI lexical features consist mainly of word unigrams in an ACC. The RI version pairs unigrams from the two participating ACCs together.

Since information about gold ACs is not available in the test set, test RCs must be created differently. PIPE generates test RCs from all possible pairs of ACCs that the ACI system predicted are real components within the same paragraph (i.e., it labeled them something other than "non-argumentative").

9.4.2 Integer Linear Programming (ILP)

Two problems with the PIPE approach outlined above are that errors made by the ACI subsystem propagate to the RI task, thus reducing RI performance, and there are constraints on the annotations in S&G's corpus that aren't enforced by PIPE.

To solve both of these problems, P&N propose solving the ACI and RI tasks jointly using integer linear programming (Roth and Yih, 2004), a technique for solving constrained optimization problems. To understand how their ILP system does this, first recall that an instance labeled by a maximum entropy classifier is assigned a probability distribution over all the instance's possible labels. The PIPE system just assumes that an instance's predicted label is the label the learner assigns the highest probability for that instance. The ILP system instead uses integer linear programming to label all the ACCs and RCs in an essay in order to choose the best labels for all of them. So for example, if a paragraph contains an RC that is very likely to be a support relation (according to the RI classifier), but one of its participating ACCs' probability of being argumentative according to the ACI classifier is just barely too low for the PIPE system to label it as such, the ILP system can choose to label the ACC with one of the argumentative ACI labels in order to allow it to participate in the relationship.

In order to benefit from this kind of reasoning, the ILP system cannot use the same set of ACCs and RCs used by the PIPE system, as the PIPE system excludes from consideration any RCs that can be generated from ACCs that are most probably non-argumentative. To account for this, the ILP system selects (1) the 3 most likely premise ACCs from each sentence, (2) the 5 most likely claim ACCs from each paragraph, and (3) the 5 most likely major claim ACCs from each essay occurring in the first or last paragraph.[1] It then constructs RCs from all pairings of these ACCs occurring in the same paragraph as long as at least one was chosen because it might be a premise and neither one was chosen because it might be a major claim, since the corpus adheres to the constraint that all premises are in a relationship and all major claims are not. The test RCs generated in this way are presented to the RI

[1]The ILP system selects more premise than claim ACCs (3 per sentence vs 5 per paragraph) because the corpus contains over twice as many premises as claims. The corpus constrains major claims so that they may only appear in an essay's first or last paragraph.

maximum entropy classifier normally, making no other changes to how the pipeline system works.

This introduces the problem that the RI classifier may posit a relationship between incompatible ACCs (e.g., one might be labeled non-argumentative). We mentioned above that integer linear programming is a technique for solving constrained optimization problems, so it is possible to construct one that not only assigns the most probable labels to all ACCs and RCs in a paragraph, but also ensures that the ACCs and RCs are consistent. In particular, the constraints encoded into the integer linear program ensure that (1) each ACC is assigned exactly one type, (2) each RC is assigned exactly one type, (3) if there is a relationship between two ACCs, the ACCs must both be assigned argumentative types (i.e., they can't be labeled non-argumentative), (4) major claims can only occur in the first or last paragraphs, (5) major claims have no parents, (6) a premise must have a parent, (7) argumentatively labeled ACCs cannot overlap in the text, (8) a paragraph contains at least one claim or major claim, and (9) a sentence must not have more than two argumentatively labeled ACCs. The optimization problem of finding the best labeling of instances subject to these constraints is solved using Gurobi[2].

We claimed above that integer linear programming can be used to find the best assignment of labels to our ACCs and RIs, but did not define how to measure an assignment's quality. Because performance on end-to-end argumentation mining is evaluated as the average of ACI and RI F-scores, the ILP system's objective function is constructed to maximize this number. Recall that the general F-score formula can be simplified to:

$$F = \frac{2TP}{2TP + FP + FN} \tag{9.1}$$

where TP, FP, and FN are counts of true positive, false positive, and false negative predictions respectively. An integer linear program cannot directly maximize this formula for any task

[2]http://www.gurobi.com

because division cannot be encoded into an objective function and TP, FP, and FN can only be calculated if the system has access to gold standard test data. For this reason, ILP instead encodes $\frac{G_{ACI}+G_{RI}}{2}$ as the objective function its integer linear program maximizes, where G_{ACI} and G_{RI} are the values of G below as calculated on the ACI and RI tasks respectively:

$$G = \alpha 2TP_e - (1-\alpha)(FP_e + FN_e) \tag{9.2}$$

where TP_e, FP_e, and FN_e, are *expected* values for TP, FP, and FN respectively, and α attempts to balance the importance of maximizing the numerator vs minimizing the denominator in the F-score formula. These expected values for a complete labeling of all ACCs and RCs are calculated based on the probabilities assigned to the labeled instances as returned by the ACI and RI classifiers.

9.5 Approach

A problem with the baseline approaches described above is that they can be viewed as making a collection of purely local decisions. They cannot exploit argument tree level information (e.g., how many claims should an argument tree have) except through the use of hard constraints, which must be painstakingly encoded manually.

To address this shortcoming, we need a learning algorithm that can select the best *Candidate Argument Tree* (CAT) from each paragraph. A CAT for a paragraph consists of 0 or more labeled ACCs[3] in the paragraph's text and 0 or more labeled RCs describing how they are related to each other argumentatively.[4] The best CAT, then, is the one whose structure

[3]In previous sections, a labeled ACC or RC could have a non-argumentative or not related label, respectively. For the remainder of this document, a labeled ACC or RC is assumed to have an argumentative or related label.

[4]Technically, a CAT (or GAT) may be a forest, as there is no constraint on this corpus ensuring that all ACs are connected by relationships.

most closely matches the paragraph's *Gold Argument Tree* (GAT), which consists of exactly the set of ACs and relationships between them that have been annotated.

Since there are many ways to select and label text segments and relationships between them, a paragraph can contain many possible CATs. How can we select the best CAT given a paragraph? To perform this task, we employ a variation of the structured perceptron (Collins, 2002) algorithm. Conceptually, a structured perceptron is a learner which accepts as input a set of structured instances (such as our CATs) and outputs the one that is best. More concretely, a structured perceptron is just a normal perceptron whose input is (the feature representation of) one structured instance (like a CAT) and whose output is a real value. A structured perceptron's output, then, is the instance that yields the greatest real value when fed into the normal perceptron out of the set of all instances fed to it.

A structured perceptron's learning algorithm, too, is just like that of a normal perceptron. More specifically, our structured perceptron is sequentially fed sets of CATs from each paragraph in the training data. If the CAT it outputs for a paragraph does not match the paragraph's GAT, the perceptron's edge weights get updated according to:

$$w_{i,j} = w_{i-1,j} + \beta(g_j - c_j) \tag{9.3}$$

where $w_{i-1,j}$ is the old weight associated with feature j, $w_{i,j}$ is its new value, β is a learning rate (which we set to 0.1), and g_c and c_j are GAT and CAT's values respectively for feature j.

To initialize our structured perceptron's weights, we first assign each edge a random weight. Then, for each paragraph in the training set, we generate the set of CATs that are *adjacent* to the paragraph's GAT. A CAT is adjacent to a GAT (or another CAT) if it conforms to all the constraints in Section 9.4.2 and is structurally identical (contains the same labeled ACCs and labeled RCs) to the GAT except that it 1) contains one more labeled ACC, 2) has one labeled ACC removed, 3) has one labeled ACC relabeled (e.g.,

from claim to premise), 4) has one labeled RC relabeled (e.g., from support to attack), or 5) one of its labeled ACCs has a different parent.[5] Then, for each generated CAT, we apply the weight update rule above as if the structured perceptron had chosen the CAT. We initialize weights in this way because perceptrons with randomly-initialized weights can learn slowly. By initially training it on CATs that are structurally similar to the GAT, we give our perceptron an initial idea about which features are important.

Our structured perceptron is trained differently than a standard structured perceptron as well. Since there are so many possible CATs for each paragraph, it would be impractical to generate and compare all of them with a structured perceptron. For that reason, we employ the following search procedure to find the best CAT.

We maintain a priority queue of the best CATs, where a CAT's priority is the numerical value it yields when fed to the perceptron. We initialize the priority queue with a CAT extracted from the pipeline system's predictions. More specifically, we assign each ACC its pipeline-predicted label. We then force the CAT to conform to the corpus's constraints by removing overlapping ACCs with argumentative labels. If the pipeline system predicted the presence of any premises in a paragraph, but predicted no claims, we relabel the ACC with the highest probability of being a claim as a claim. This allows us to add support RCs for each premise to a claim in the CAT.

We then iterate the following procedure. For each of the CATs in the queue, we generate all its adjacent CATs and add them to the queue if they have never been added to it on a previous iteration. If at the end of an iteration, the size of the queue has not grown, we say that the structured perceptron chooses the CAT at the head of the queue. Otherwise, we truncate the queue to length n^6 and start the next iteration.

[5]See the supplemental file for details on how we apply all these adjacency rules to generate CATs that don't violate any constraints.

[6]We use $n = 10$

To better understand this algorithm, consider the case where $n = 1$. This would be a straightforward greedy search where, upon each iteration, we move to the best adjacent CAT until we reach a local maximum where the CAT we have selected obtains a higher score from the perceptron than any of its neighbors. Setting n to larger values makes it less likely that we will get stuck at a local maximum since it means we keep track of several high quality CATs at once. Intuitively, this algorithm should allow us to walk towards a CAT that is very similar to the GAT because a CAT's rank in the queue represents how much it resembles a generic gold argument tree. We employ this same search algorithm during testing.

A shortcoming of the structured perceptron algorithm is that by default it attempts to tune its weights to minimize training error. However, as mentioned in Section 9.4.2, this is not the metric by which we will evaluate its performance. Recall that the ILP system used a custom objective function in order to maximize average F-score between the ACI and RI tasks. We believe it is possible to do something similar with structured perceptrons in the following way.

The output of a normal perceptron is given by $w \cdot f$, where w is the perceptron's weight vector and f is an instance's feature representation. By augmenting this dot product with a loss function, we can encourage our structured perceptron to select CATs that will help it make instructive mistakes during training. Our structured perceptron's output function, then, is given by:

$$o_i = w \cdot f_i + c \;\; \alpha \cdot \{FP_{ACIi}, FN_{ACIi}, FP_{RIi}, FN_{RIi}\} \tag{9.4}$$

where o_i is the perceptron's output for CAT i, f_i is CAT i's feature vector, and c is a cost factor associated with the loss function, and FP_{ACIi} is the number of false positive labeled ACCs CAT i contains when compared the paragraph's gold standard argument tree GAT i. α is a vector determining how much impact we want a CAT's false positives and false negatives on the ACI and RI tasks to have on the perceptron's training output. By selecting

the c and α parameters appropriately, we can ensure that these mistakes help it improve its average F-score.[7] Note that this loss term is only applied to the structured perceptron's output during training. During testing, we use the perceptron's standard output function.

We iterate the training procedure over all the paragraphs in the training set 100 times, as the algorithm's performance plateaus at around 100 iterations. After each iteration over the training set, we replace the perceptron with a perceptron whose weights are the average of the weights of all the perceptrons learned at the completion of all previous iterations. We implement this behavior in our structured perceptron as well, as it slows the process of overfitting.

Thus far, we have discussed how the structured perceptron learning algorithm works, but we have not discussed how we incorporate information about CAT and GAT structures as features for the perceptron. We employ two types of features in our structured perceptron which help us capture structural information about CATs and GATs.

The first feature type captures information about the labeled ACCs that comprise a CAT. For each labeled ACC in a CAT, we generate six features. The first three are the sequence of one, two, and three lemmatized words immediately preceding the ACC's text paired with the ACC's label (major claim, claim, premise) in this CAT. The second set of three features are the same, but encode lemmatized words immediately succeeding the ACC's text rather than preceding it.

To illustrate why these features would be useful, consider the example text from the introduction. "believe that-MajorClaim" seems like a feature that would appear in a GAT, as "believe that" is a strong indicator that the text following it is a major claim. These features simultaneously convey information to the learner about the numbers of labeled ACCs of each type in a CAT, which is useful since, for example, most GATs contain a few premises.

[7]We select $c = 100$ and $\alpha = \{1, 20, 1, 20\}$ (normalized to unit length) to encourage increased precision.

The second feature type captures information about the labeled RCs that comprise a CAT. Given a labeled RC, a feature of this type is comprised of the following triple of values: (any one feature of the previous type from the RC's child, any one feature of the previous type from the RC's parent, the RC's type (support or attack)).

To understand why these features are useful, consider the following sentence: "Given that (1) the Internet is used as an international and public space, (2) the government has no right over the information which may be presented via the Internet." A CAT with the right RC would generate from this sentence's paragraph the feature ('give that', ',' , 'supports'), which is a feature that is likely to be generated by a gold relationship.

9.6 Evaluation

In this section we discuss how we evaluate our structured perceptron based approach to argument mining.

9.6.1 Experimental Setup

Corpus. As mentioned before, we use as our corpus the 402 essays annotated with argumentative discourse structures by S&G. All of our experiments are conducted via five-fold cross-validation on this corpus. In each fold experiment, we reserve 60% of the essays for training, 20% for development (tuning α), and 20% for testing.

Evaluation metrics. To calculate F-score, on each task using Equation 9.1, we need to explain what constitutes a true positive, false positive, or false negative on each task. Given that j is a true argument component and i is an ACC, the formulas for the ACI task are:

$$TP = \left|\{j \mid \exists i \; |gl(j) = pl(i) \wedge i \doteq j\}\right| \tag{9.5}$$

$$FP = \left|\{i| \; pl(i) \neq n \wedge \nexists j \mid gl(j) = pl(i) \wedge i \doteq j\}\right| \tag{9.6}$$

$$FN = \left|\{j \mid \nexists i \; |gl(j) = pl(i) \wedge i \doteq j\}\right| \tag{9.7}$$

where $gl(j)$ is the gold standard label of j, $pl(i)$ is the predicted label of i, n is the non-argumentative class, and $i \doteq j$ means i is a *match* for j. i and j are considered an *exact match* if they have exactly the same boundaries and an *approximate match* if they share over half their tokens.

We perform most of our analysis on approximate match results rather than exact match results as it can be difficult even for human annotators to identify exactly the same boundaries for an argument component.[8] We use the same formulas for calculating these numbers for the RI problem except that j and i represent a true relation and an RC respectively, two relations approximately (exactly) match if both their source and target ACCs approximately (exactly) match, and n is the no-relation class.

9.6.2 Results and Discussion

Approximate and exact match results of the pipeline approach (PIPE), the joint inference approach (ILP), and our structured perceptron approach (SP) are shown in Table 9.2. As we can see, using approximate matching, our SP system achieves significant[9] improvements over the baseline systems by a variety of measures. The most important of these improvements is shown in the penultimate column, where our SP system outperforms the best baseline by 6.0% absolute F-score (a relative error reduction of 9.6%). The results in this column measure system performances on the relation identification task, which is the more difficult of

[8]Approximate match has been used in evaluating opinion mining systems (e.g., Choi et al. (2006), Yang and Cardie (2013)), where researchers have also reported difficulties in having human annotators identify exactly the same boundaries for an opinion expression and its sources and targets. They have adopted an even more relaxed notion of approximate match: they consider two text spans an approximate match if they share at least one overlapping token.

[9]Boldfaced results in Table 9.2 are significant ($p < 0.02$, paired t-test) compared to the best baseline system, which is determined on a per-performance metric basis.

Table 9.2. Five-fold cross-validation F-score percentages for argument component identification (ACI) and relation identification (RI). Column abbreviations are major claim F-score (MC-F), claim F-score (C-F), premise F-score (P-F), Precision (P), Recall (R), F-score (F), support F-score (S-F), and attack F-score (A-F).

		ACI						RI					Avg
	System	MC-F	C-F	P-F	P	R	F	S-F	A-F	P	R	F	F
Approx	PIPE	44.0	36.1	69.8	75.2	49.3	59.6	20.1	00.0	22.3	17.4	19.5	39.5
	ILP	50.7	42.6	76.7	61.0	66.9	63.8	32.5	01.4	23.9	47.5	31.8	47.8
	SP	49.2	44.2	74.9	63.6	65.3	64.4	**38.9**	02.9	**37.0**	38.7	**37.8**	**51.1**
Exact	PIPE	39.6	33.9	61.7	64.0	45.8	53.4	17.2	00.0	18.6	15.2	16.7	35.1
	ILP	41.7	34.9	63.5	50.6	55.1	52.7	23.0	00.0	16.9	33.4	22.4	37.6
	SP	36.4	**39.2**	61.0	51.9	53.3	52.6	**29.4**	02.0	**27.9**	29.2	**28.6**	**40.6**

the two tasks since, in order to correctly identify a relationship, a system must first correctly identify the two argument components participating in it.

Our SP system also outperforms the baselines with respect to the ACI task. As a result, we see in the last column that on the combined task, whose performance metric is the average of the F-scores for the ACI and RI tasks, our system outperforms the best the baseline by 3.3% absolute F-score (a relative error reduction of 6.7%). So our system significantly improves performance on the RI task without harming performance on the ACI task.

Using exact matching, the relationships we described between our SP system's performance and those of the best baseline are mostly similar. Our system significantly outperforms the best baseline with respect to nearly all the same metrics and by similar amounts.

9.6.3 Error Analysis and Future Work

It is obvious from Table 9.2 that our system has considerably more difficulty with the RI task than with the ACI task, regardless of whether evaluation occurs in an exact or approximate match setting. While our approach is considerably better at this task than the baseline systems, it nevertheless has much room for improvement.

A closer inspection of our SP system's RI predictions reveals that its low precision is mostly due to predicted relationships wherein one of the participating labeled ACCs is not a true argument component. Its low RI recall, similarly, is mostly due to cases where a candidate tree does not correctly identify one of the participating argument components as a true argument component. Recalling that our structured perceptron's CAT search algorithm begins with a CAT proposed by one of our baseline systems, and that perceptrons tend not to be as robust to large feature sets as the maximum entropy learner based baselines, the best way to improve the structured perceptron's performance may be to introduce additional features into the maximum entropy classifier targeted at improving local ACI.

9.7 Summary

We developed a structured perceptron approach to the under-investigated task of *end-to-end* argument mining in persuasive student essays. While most previously proposed approaches to the task predict argument tree structure by making a collection of local decisions about the argument's structure (i.e., whether a given text span is part of the argument structure and how it is related to other text spans in the argument), our structured perceptron approach judges the quality of a proposed argument tree holistically. In an evaluation on version 2 of Stab and Gurevych's corpus of 402 argument annotated essays, our approach yields a 9.6% relative error reduction on the subtask of identifying relationships between text segments compared to two state of the art baseline systems that have previously been employed for the end-to-end argument mining task.

CHAPTER 10

UNSUPERVISED ARGUMENTATION MINING

10.1 Introduction

As we discussed in the last two chapters, argumentation mining typically involves addressing two subtasks: (1) *argument component identification* (ACI), which consists of identifying the locations and types of the components that make up the arguments (i.e., major claims, claims, and premises), and (2) *relation identification* (RI), which involves identifying the type of relation that holds between two argument components (i.e., support, attack, none). As an example, consider the following text segment taken from a corpus of student essays that Stab and Gurevych (S&G) annotated with argument components and their relations (Stab and Gurevych, 2016):

> In my view point, (1) I would agree to idea of taking a break before starting higher education. (2) Students who take break, gain lot of benefits by traveling around the places or by working. (3) They tend to contribute more due to their real world experiences which makes them more mature.

In this example, premise (3) supports claim (2), and (1) is a major claim.

Despite the large amount of recent work on computational argumentation, there is an area of argument mining where progress is fairly stagnant: the development of end-to-end argument mining systems. With the rarest exceptions, existing argument mining systems are *not* end-to-end. For example, using their annotated corpus, S&G trained an ACI classifier and applied it to classify only *gold* argument components (i.e., text spans corresponding to a major claim, claim, or premise in the gold standard) or sentences that contain no gold argument components (as *non-argumentative*). Similarly, they applied their learned RI classifier to classify only the relation between two *gold* argument components. In other words, they simplified both tasks by avoiding the challenging task of identifying the locations of argument components. Consequently, their approach cannot be applied in a realistic setting

where the input is an *unannotated* essay. Recent related work has focused on using joint inference (via Integer Linear Programming (ILP), for instance) to improve the ACI and RI classifications (Peldszus and Stede, 2015; Stab and Gurevych, 2016; Wei et al., 2017), but since they continue to make the same assumptions that S&G made, these systems remain non-end-to-end.

In the previous two chapters, we presented two end-to-end approaches for addressing the argumentation mining task. While they are improvements on previous approaches in that they do not make S&G's simplifying assumptions, they nevertheless both share one major shortcoming. Being supervised approaches, they both require a lot of argument annotated training data in order to perform the end-to-end argumentation mining task.

What's worse, argument-labeled essays are more expensive to produce than, say, essays annotated with any of the quality scores described in Chapters 3, 4, 5, and 6. The reason for this is that annotating an essay for one of these scoring tasks requires an annotator to, after one careful reading, assign the essay one score. Argumentation mining annotation, by contrast, requires annotators to identify an average of 15.2 argument components and 9.5 relationships between them per essay.

To address this problem, in this chapter we propose a new unsupervised approach to the challenging but under-investigated task of *end-to-end* argument mining in student essays. More specifically, we develop heuristics for automatically identifying argument components and their relationships within an essay. We then employ self-training to leverage these heuristically-labeled components and relationships to train an argument mining system whose performance is near state-of-the-art despite requiring no labeled data.

We compare our approach to the recent work on end-to-end argument mining in student essays described in the previous two chapters, which, recall, used integer linear programming and structured perceptrons to argumentatively label essays. In an evaluation on a corpus of 402 essays annotated by S&G, our approach performs competitively with both systems.

10.2 Related Work

As the task we address in this chapter is nearly identical to the task addressed in the previous two chapters, related work is discussed in Section 8.2.

10.3 Corpus

We similarly use the same corpus as that described in the previous chapter, so discussion of our argumentation mining corpus can be found in Section 9.3.

10.4 Baselines

Our first two baselines for argumentation mining on student essays are described in Chapter 9.4.

More specifically, our first baseline is a pipeline system (PIPE) wherein we solve the argument mining subtasks of ACI and RI sequentially. This is an adaptation of the pipeline system described in Persing and Ng (2016a) that allows it to handle the newer S&G corpus appropriately. The adapted pipeline system is described in Chapter 9.4.1.

A problem with the pipeline system is that the output of its ACI subsystem is not 100% accurate, but the RI system depends on the ACI system's output in order to determine how argument component candidates are related. Thus errors can propagate from the ACI system to the RI system, which can harm the RI system's output. Our second baseline addresses this problem by solving both the ACI and RI subtasks jointly using integer linear programming (ILP), as originally described in Persing and Ng (2016a). The version of this system we use has also been adapted for use on version 2 of S&G's corpus as described in Chapter 9.4.2.

A problem with the baseline approaches described above is that they can be viewed as making a collection of purely local decisions. They cannot exploit argument tree level information (e.g., how many claims should an argument tree have) except through the use of hard

constraints, which must be painstakingly encoded manually. To address this shortcoming, we use as our third baseline a structured perceptron approach (SP) which can select the best *Candidate Argument Tree* (CAT) from each paragraph. Whereas the instances in the previous two approaches are Argument Component Candidates (ACCs), segments of text that could possibly be major claims, claims, or premises, and Relationship Candidates (RCs), pairs of ACCs that might have a support or attack relationship, the structured perceptron approach treats as its instances entire CATs. A CAT for a paragraph consists of 0 or more labeled ACCs in the paragraph's text and 0 or more labeled RCs describing how they are related to each other argumentatively. The best CAT, then, is the one whose structure most closely matches the paragraph's *Gold Argument Tree* (GAT), which consists of exactly the set of ACs and relationships between them that have been annotated. Thus a CAT's feature representation in the structured perceptron can include tree level features. This system is described in Chapter 9.5.

10.5 Unsupervised Approach

A problem with all the baseline approaches is that they are purely supervised. That is, they require a lot of argument-annotated essay data in order to perform the argumentation mining task. In this section, we will describe an unsupervised approach to argumentation mining which avoids this problem by using no annotated training data.

Broadly, our unsupervised approach to argumentation mining involves first identifying a large set of ACCs that could potentially be argument components from an essay, as described in Chapter 8.4.1. Our approach heuristically labels a subset of these ACCs with argument component labels, and then iteratively builds a larger training set using the following method. It iteratively 1) trains a maximum entropy classifier on the heuristically labeled data, 2) uses the classifier to make probabilistic predictions about the labels of ACCs in an unlabeled essay set, and 3) adds the ACCs whose labels we are most confident of (according to the

probabilistic predictions) to the heuristically labeled set. After several iterations of this process, it heuristically stitches the maximum entropy classifier's output into an argument tree.

10.5.1 Heuristic Labeling

The approaches we proposed for the argumentation mining task in the previous two chapters focus on exploiting properties of the problem's structure rather than introducing insights into what argument components look like or what argumentative relationships look like. In order to heuristically label ACCs and RCs, however, we will need to introduce such insights into our approach. The heuristics we introduce for automatically labeling ACCs rely on three factors, 1) the number of the paragraph the ACC appears in (i.e., whether it is the first, last, or a middle paragraph), 2) the location of the sentence the ACC appears in within its paragraph (i.e., whether it is the first, last, or a middle sentence), and 3) the context n-grams surrounding the ACC.

Heuristically Labeling Major Claims

The reason our heuristics depend on an ACC's paragraph's location within an essay is that the typical argumentative structure of, for example, the first paragraph, looks very different than the typical argumentative structure of a body paragraph (which occurs between the first and last paragraph). Indeed, there is a hard constraint on this corpus that major claims can only appear in the first and last paragraphs, which makes sense because essays are often introduced with with a thesis statement (a major claim), and often conclude with one. So for this reason, we search only the first and last paragraphs for ACCs to heuristically label as major claims.

To find major claims in the first paragraph, we first observe that each type of argument component is often introduced using some discourse marker like those shown in Table 10.1. In

Table 10.1. Context n-grams used to heuristically label argument components of each type.

	Major Claim	Claim	Premise
Preceding AC	consequently contend that feel that i agree i believe i feel i think in conclusion , in short , maintain that my opinion my view so suppose that therefore think that thus ,	consequently contend that feel that finally , first , first of all , firstly , i agree i believe i feel i think in conclusion , in short , last , lastly , maintain that my opinion my view second , secondly , so suppose that therefore think that third , thirdly , thus ,	for instance , further , furthermore , in addition , in fact , indeed , moreover , since to illustrate ,
Succeeding AC	because that is ,	because that is ,	, so , therefore

particular, major claims often follow the phrases in the first row of the major claim column. For this reason, we identify the last occurrence of one of these connectives within an essay's first paragraph (because introductory paragraphs are more likely to conclude with a major claim than they are to begin with one). We then find the ACC that appears most closely

after the phrase which ends at the end of a simple, declarative clause (as determined by a syntactic parse of the sentence), heuristically labeling this ACC as a major claim.

If no major claim can be identified in this way, we then search the essay for the last occurrence of one of the n-grams in major claim's succeeding row in Table 10.1. The last ACC occurring before this n-gram that exactly covers a simple, declarative clause is heuristically labeled as a major claim, as n-grams in this cell frequently indicate that whatever preceded them is about to be further explained or reasoned about, which in turn indicates the importance of the preceding text to the argument.

If no major claim ACC can be identified in either of these ways, we simply do not heuristically label any major claims from this paragraph.

Essays' last paragraphs tend to be structurally very similar to their first paragraphs, containing major claims and little else. For this reason, we follow the same procedure outlined above for identifying major claims in an essay's last paragraph. However, since major claims are sometimes expressed without any nearby discourse markers like those shown in Table 10.1, it is important for us to identify a few major claims without them in order to our eventual learned system generalize about major claim structure.

Since it is a little bit more common for last paragraphs to include major claims than it is for first paragraphs to include them, and since a paragraph's last sentence is more likely to include a major claim than its first, we employ the following additional rule for identifying major claims in the last paragraph. If an essay's last paragraph is very short, including two or fewer sentences, and we are not able to heuristically label any other major claims from this last paragraph using any of the methods described above, we heuristically label the longest simple, declarative clause in the last sentence as a major claim.

Heuristically Labeling Claims

While claims can occur in any paragraph, we only attempt to heuristically label claims occurring in body (middle) paragraphs. To understand the reason why we do this, it is

helpful to compare the first two columns of Table 10.1. Notice that the columns' contents are identical except that the discourse markers preceding claims include ordinal markers (e.g., first, second, last). This is because, just as major claims express the main purpose of an essay, claims express the main purpose of a paragraph. Thus, they tend to be introduced using similar phrases. This makes it difficult to distinguish between claims and major claims occurring in an essay's first or last paragraphs.

Because of this similarity between claims and major claims, the way we heuristically identify claims in a body paragraph is identical to the way we identify major claims in the first paragraph except for the following two differences. First, we use discourse markers from the claim column of Table 10.1 rather than ones from the major claim column. Second, we exclude from consideration any ACCs occurring outside of the paragraph's first or last sentences. We do this because, while claims can occur in any sentence, body paragraphs tend to either begin or conclude by stating the claim the paragraph is about. The remaining sentences tend to be filled with premises that support the claim.

Heuristically Labeling Premises

Since premises are rare in first or last paragraphs, as with claims, we also only heuristically label premises occurring in body paragraphs. We also exclude from premise labeling consideration any ACCs occurring in a paragraph's last sentence because, even if we can not heuristically identify a claim in a last sentence, there is too strong a chance that the reason for this is because our claim labeling heuristics do not have a high enough recall and simply failed to recognize an existing claim's presence.

With those exceptions, our premise labeling heuristics are very similar to our claim and major claim labeling heuristics. We first find all occurrences in an essay of preceding premise discourse markers from Table 10.1. For each of these occurrences, we find the closest following ACC that ends at the end of a simple declarative clause and heuristically label it as a premise.

We then find all occurrences in an essay of succeeding premise discourse markers from the table. For each of these occurrences, we heuristically label as a premise the nearest preceding ACC having the same boundaries as a simple declarative clause. The selected preceding or succeeding ACC can neither overlap with a previously heuristically labeled ACC, nor can it cannot occur in a different sentence than the discourse marker that triggered its labeling.

Unlike our heuristics for labeling claims and major claims, our heuristics for labeling premises are not limited to labeling one premise per paragraph, as can be seen in the description in the previous paragraph. The reason for this is that it is most common for a paragraph to contain 1 or fewer claims or major claims, but body paragraphs tend to contain very few non argumentative sentences. Those body paragraph sentences that do not contain a claim usually contain a premise.

However, despite being more permissive than the claim and major claim heuristics, this premise labeling method still fails to capture that most sentences contain a premise. For this reason, we search all body paragraph sentences except the last sentence for occurrences of any of the phrases from Table 10.1. If we do not find any of these phrases, we take this as a strong indication that the sentence itself is a premise, since the presence of one of these phrases might indicate that we just failed to identify an argument component that is actually present for reasons of precision. We therefore heuristically label the longest ACC occurring in such a sentence as a premise.

An exception to the last premise labeling heuristic is that, if the sentence begins with some introductory phrase measuring three tokens or less followed by a comma, we instead label the longest ACC occurring after this phrase as the premise. We do this because there are too many phrases that can be used to introduce a premise for us to enumerate, but as you can see from the premise column of the table, when they do exist, they are usually short and end with a comma.

10.5.2 Self-Training

A problem with the labels we applied in the previous subsection is that they are intended to be high precision but low recall. Ideally, an ACI system will strike an appropriate balance between precision and recall in order to maximize its ACI f-score. For this reason, our ACI system needs to be more general than the hand-constructed heuristics described above. So rather than treating our heuristic labeling as an ACI system's predicted labels, we instead employ the heuristically labeled data as our initial training data in an unsupervised variation of *s*elf-training.

More specifically, we first extract all the ACCs from an unannotated training set. We label all instances that can be labeled heuristically with the appropriate ACI class label (major claim, claim, or premise), and label the remaining ACCs as probably *n*on-argumentative. Given this set of extracted ACCs, we train a four class classifier for ACI using MALLET's (McCallum, 2002) implementation of maximum entropy classification.

Each instance is represented using S&G's structural, lexical, syntactic, indicator, and contextual features for solving (a simplified version of) the same problem. Briefly, the structural features describe an ACC and its covering sentence's length, punctuations, and location in the essay. Lexical features describe the $1-3$ grams of the ACC and its covering sentence. Syntactic features are extracted from the ACC's covering sentence's parse tree and include things such as production rules. Indicator features describe any explicit connectives that immediately precede the ACC. Contextual features describe the contents of the sentences preceding and following the ACC primarily in ways similar to how the structural features describe the covering sentence.

The result of this process is a maximum entropy classifier that can assign a probability distribution to any ACC describing how likely it is to be a major claim, claim, premise, or non-argumentative. A problem with this classifier is that, recall, it was trained on data wherein the non-argumentative training instances may have been labeled as such simply

because we could not heuristically determine with enough confidence that they should have an argumentative label.

For this reason, we now apply the classifier to the non-argumentatively labeled training ACCs. We select the 10 of these instances that the classifier is most confident are argumentative if it assigns at least an 80% probability to one of the argumentative classes. We then label these ACCs as with the labels the classifier indicates are appropriate and add them to the heuristically labeled dataset before training a new maximum entropy classifier on this labeled data. We repeat this process until there are no more training ACCs labeled with the non-argumentative label that can be relabeled in this way (i.e., the classifier assigns none of them an 80% probability of belonging to one of the argumentative classes).

10.5.3 Building the Argument Tree

Recall that in Chapter 9.5 we heuristically constructed a candidate argument tree (CAT) from the predictions made by an ACI maximum entropy classifier just like the one described above. The difference between the classifiers lies only in the method by which the training data was obtained (hand annotated or heuristically labeled). We apply the same technique to construct a CAT for each paragraph from our self-trained classifier predictions with some changes meant to more strongly enforce our observations about the most likely structure of an argument tree. These changes are needed because we have less confidence in the labels of our heuristically labeled training data (particularly the non-argumentatively labeled ACCs) than we would in hand-annotated data.

Because the only ACs appearing in most first or last paragraphs are major claims, and because it is uncommon for more than one major claim to appear in one of these paragraphs, we heuristically label the ACCs in such paragraphs as follows. First, we identify the ACC that is most likely to be a major claim, as indicated by the ACI maximum entropy classifier. If its probability of being a major claim is $> .5$, we label it as a major claim. Otherwise it is labeled as non-argumentative. All other ACCs in the paragraph are labeled non-argumentative.

As we discussed earlier, body paragraphs are structured differently. A typical body paragraph contains only one claim and many premises that support it. Major claims are not permitted in body paragraphs as per the annotation guidelines. For this reason, to construct our body paragraph CAT, we first identify the ACC in the body paragraph with the highest probability of being a claim, as determined by the ACI maximum entropy classifier. We label this ACC as a claim. We then rank all remaining ACCs by their probability of being a premise, from most probable to least probable. Progressing through the ranked list sequentially, we label each ACC as a premise if it meets two conditions. First, it must not overlap with any previously-labeled ACC. Second, its probability of being a premise according to the ACI classifier must be greater than both its probability of being a claim and its probability of being a major claim. For each ACC we label as a premise, we add a support relationship between the premise and the ACC we labeled as the paragraph's claim.

10.6 Evaluation

In this section we discuss our evaluation of our unsupervised approach to argument mining.

10.6.1 Experimental Setup

We use the same corpus and evaluation metrics as described in Chapter 9.6

10.6.2 Results and Discussion

Approximate and exact match results of the pipeline approach (PIPE), the joint inference approach (ILP), the structured perceptron approach (SP), and our unsupervised approach (U) are shown in Table 10.2. As we can see, using approximate matching, our unsupervised system achieves results that are statistically indistinguishable[1] from those of the best baseline

[1]Boldfaced results are not significantly lower ($p > .02$, paired t-test) than the highest-scoring baseline.

Table 10.2. Five-fold cross-validation F-score percentages for argument component identification (ACI) and relation identification (RI). Column abbreviations are major claim F-score (MC-F), claim F-score (C-F), premise F-score (P-F), Precision (P), Recall (R), F-score (F), support F-score (S-F), and attack F-score (A-F).

		ACI						RI					Avg
	System	MC-F	C-F	P-F	P	R	F	S-F	A-F	P	R	F	F
Approx	PIPE	44.0	36.1	69.8	75.2	49.3	59.6	20.1	00.0	22.3	17.4	19.5	39.5
	ILP	50.7	42.6	76.7	61.0	66.9	63.8	32.5	01.4	23.9	47.5	31.8	47.8
	SP	49.2	44.2	74.9	63.6	65.3	64.4	38.9	02.9	37.0	38.7	37.8	51.1
	U	24.7	**42.2**	**77.2**	**73.5**	57.5	**64.5**	**37.8**	**00.0**	**38.1**	35.4	**36.7**	**50.6**
Exact	PIPE	39.6	33.9	61.7	64.0	45.8	53.4	17.2	00.0	18.6	15.2	16.7	35.1
	ILP	41.7	34.9	63.5	50.6	55.1	52.7	23.0	00.0	16.9	33.4	22.4	37.6
	SP	36.4	39.2	61.0	51.9	53.3	52.6	29.4	02.0	27.9	29.2	28.6	40.6
	U	18.9	**36.4**	59.6	57.7	45.2	50.7	**25.9**	**00.0**	**26.1**	24.3	**25.2**	**37.9**

system by a variety of measures. The most important of these results is shown in the last column, where our unsupervised approach, despite using no labeled training data, achieves performance that is only 0.5% lower than that of the structured perceptron baseline. This column measures the average f-score between the two argument mining subtasks (ACI and RI).

Our unsupervised system also achieves results that are statistically indistinguishable from the best baseline on each of the subtasks, scoring only 0.9% less in f-score than the SP system on the RI task and 0.1% more in f-score than the SP system on the ACI task. The other measures on which our unsupervised system's approximate performance is indistinguishable from the best baseline are claim f-score, premise f-score, ACI precision, support f-score, attack f-score, and ACI precision.

Due to the difficulty of heuristically detecting an AC's exact boundaries, our unsupervised system's exact matching performance is statistically indistinguishable from fewer baseline performances than was its approximate performance. By the most important measure, however, our unsupervised system's performance is still indistinguishable from that of the SP baseline system, scoring only 2.7% less than it in absolute average f-score.

10.6.3 Error Analysis and Future Work

It is obvious from Table 10.2 that our system has considerably more difficulty with the RI task than with the ACI task, regardless of whether evaluation occurs in an exact or approximate match setting. Indeed, this is also true of all the baseline systems. In Chapters 8 and 9, we noted that this is mainly because the RI task is inherently more difficult than the ACI task because, in order to correctly identify a relationship, an argument mining system must also correctly identify the two participating argument components.

While this observation is correct, what it misses is that there do not currently exist high quality features for RI. As we can see from Table 8.4, removing all of the features designed by S&G specifically for the RI task from an argument mining system results in only a tiny drop in performance. The reason for this is that, unlike ACs, argumentative relations are often not triggered by discourse markers in the text. This is largely because argumentative relationships can occur between ACs in non-adjacent sentences. Thus a deeper semantic understanding of ACs is necessary in order to develop features or heuristics for detecting argumentative relationships between them.

While our unsupervised approach achieves state of the art performance on the approximate ACI task and near state of the art performance on the approximate RI task, it performs less well on the exact match ACI and RI tasks. This is largely due to the fact that it is difficult to develop heuristics for finding the exact boundaries of an AC. There are simply too many ways for an author to introduce and trigger the termination of an argument component. Indeed, this is why we needed to develop the complicated boundary detection rules illustrated in Table 8.4.1. This is a subtask on which some annotated training data may be useful, so a semi-supervised approach that focuses on annotating essays having a wide variety of sentence structures may help improve our exact matching performances most.

10.7 Summary

We developed a wholly unsupervised approach to the under-investigated task of *end-to-end* argument mining in persuasive student essays. While all existing approaches employ a lot of annotated training data for the task, our approach achieves near state-of the art performance without using any human annotations. Particularly, in an evaluation on version 2 of Stab and Gurevych's corpus of 402 argument annotated essays, our unsupervised approach yields a performance that is only 0.5% lower in absolute f-score than a state of the art supervised argument mining system.

CHAPTER 11

CONCLUSION

In this dissertation, we have presented new, state-of-the-art approaches to a variety of automated essay analysis tasks. In particular, the tasks we addressed are 1) scoring an essay according to the quality of its organization (Chapter 3), 2) scoring an essay according to how clear its thesis is (Chapter 4), 3) classifying an essay according to which errors it makes that can hinder its thesis's clarity (Chapter 4), 4) scoring an essay according to how well it adheres to its prompt (Chapter 5), 5) scoring the quality of the argument an essay presents for its thesis (Chapter 6), 6) classifying an essay according to what stance it takes with respect to its prompt (Chapter 7), and 7) detecting the argumentative structure of an essay (Chapters 8, 9, and 10).

For each of these seven tasks, we first defined the new problem. In the cases of the scoring tasks, this involved designing new grading rubrics. In the case of thesis clarity error detection, it involved devising and defining the five error classes. For stance classification and argumentation mining, we defined our tasks as more challenging versions of previously-addressed tasks.

In each chapter after related work, we presented our approach to one of these tasks alongside one or more reasonable competing approaches. Each of our approaches for each task significantly outperformed the competing approaches in an evaluation on student essays annotated for the task.

So what was novel about our approaches that made them outperform the baseline systems? Each of our approaches is based on incorporating task-specific linguistic insights into a system for performing the task. The methods we used to exploit our task-specific linguistic insights fall into two broad categories.

Most of our insights were incorporated into our approaches as linguistically-motivated heuristics or features on which we train a machine learner. In all, we presented new heuristics

or features for all seven of the tasks, so new linguistically-motivated heuristics or features appear in every chapter except for Chapter 8, which focuses exclusively on exploiting the structure of the argumentation mining problem.

The other broad category of methods for incorporating our insights into the tasks involved the modification of appropriate machine learning algorithms for solving the problems. Our contributions in this category included the modifications we made to the argumentation mining integer linear program and structured perceptron in Chapters 8 and 9, respectively.

To stimulate future research on automated essay analysis, we make our annotations on the scoring, thesis clarity error classification, and stance classification publicly available.

REFERENCES

Abu-Jbara, A., B. King, M. Diab, and D. Radev (2013, August). Identifying opinion subgroups in arabic online discussions. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Sofia, Bulgaria, pp. 829–835. Association for Computational Linguistics.

Agrawal, R., S. Rajagopalan, R. Srikant, and Y. Xu (2003). Mining newsgroups using networks arising from social behavior. In *Proceedings of the 12th international conference on World Wide Web*, pp. 529–535. ACM.

Attali, Y. and J. Burstein (2006). Automated essay scoring with e-rater® v. 2. *The Journal of Technology, Learning and Assessment 4*(3).

Balahur, A., Z. Kozareva, and A. Montoyo (2009). Determining the polarity and source of opinions expressed in political debates. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 468–480. Springer.

Bansal, M., C. Cardie, and L. Lee (2008, August). The power of negative thinking: Exploiting label disagreement in the min-cut classification framework. In *Coling 2008: Companion volume: Posters*, Manchester, UK, pp. 15–18. Coling 2008 Organizing Committee.

Barzilay, R. and M. Lapata (2005). Modeling local coherence: An entity-based approach. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics* (ACL-05), pp. 141–148.

Barzilay, R. and M. Lapata (2008). Modeling local coherence: An entity-based approach. *Computational Linguistics 34*(1), 1–34.

Barzilay, R. and L. Lee (2002). Bootstrapping lexical choice via multiple-sequence alignment. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pp. 164–171.

Barzilay, R. and L. Lee (2003). Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 16–23.

Barzilay, R. and L. Lee (2004). Catching the drift: Probabilistic content models, with applications to generation and summarization. In *HLT-NAACL 2004: Main Proceedings*, pp. 113–120.

Biran, O. and O. Rambow (2011). Identifying justifications in written dialogs. In *Proceedings of the 2011 IEEE Fifth International Conference on Semantic Computing*, ICSC '11, Washington, DC, USA, pp. 162–168. IEEE Computer Society.

Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research 3*, 993–1022.

Blum, A. and P. Langley (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence 97*(1–2), 245–271.

Boltužić, F. and J. Šnajder (2014, June). Back up your stance: Recognizing arguments in online discussions. In *Proceedings of the First Workshop on Argumentation Mining*, Baltimore, Maryland, pp. 49–58. Association for Computational Linguistics.

Bunescu, R. and R. Mooney (2005). A shortest path dependency kernel for relation extraction. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*, pp. 724–731.

Burfoot, C., S. Bird, and T. Baldwin (2011, June). Collective classification of congressional floor-debate transcripts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, pp. 1506–1515. Association for Computational Linguistics.

Burstein, J. and M. Chodorow (1999). Automated essay scoring for nonnative english speakers. In *Proceedings of a Symposium on Computer Mediated Language Assessment and Evaluation in Natural Language Processing*, pp. 68–75. Association for Computational Linguistics.

Burstein, J., M. Chodorow, and C. Leacock (2004). Automated essay evaluation: The criterion online writing service. *Ai Magazine 25*(3), 27.

Burstein, J., K. Kukich, S. Wolff, C. Lu, M. Chodorow, L. Braden-Harder, and M. D. Harris (1998). Automated scoring using a hybrid feature identification technique. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pp. 206–210. Association for Computational Linguistics.

Burstein, J. and D. Marcu (2000). Towards using text summarization for essay-based feedback. In *La 7e Conference Annuelle sur Le Traitement Automatique des Langues Naturelles TALN*.

Burstein, J. and D. Marcu (2003a). Automated evaluation of discourse structure in student essays. *Automated essay scoring: A cross-disciplinary perspective*, 209–229.

Burstein, J. and D. Marcu (2003b). A machine learning approach for identification thesis and conclusion statements in student essays. *Computers and the Humanities 37*(4), 455–467.

Burstein, J., D. Marcu, S. Andreyev, and M. Chodorow (2001). Towards automatic classification of discourse elements in essays. In *Proceedings of the 39th annual Meeting on Association for Computational Linguistics*, pp. 98–105. Association for Computational Linguistics.

Burstein, J., D. Marcu, and K. Knight (2003). Finding the write stuff: Automatic identification of discourse structure in student essays. *IEEE Intelligent Systems 18*(1), 32–39.

Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model coherence in student essays. In *Human language technologies: The 2010 annual conference of the North American chapter of the Association for Computational Linguistics*, pp. 681–684. Association for Computational Linguistics.

Burstein, J., J. Tetreault, M. Chodorow, H. Zinsmeister, and B. Webber (2013). Holistic annotation of discourse coherence quality in noisy essay writing. *Dialogue and Discourse 4*(2), 34–52.

Burstein, J., J. Tetreault, and N. Madnani (2013). The e-rater automated essay scoring system. *Handbook of automated essay evaluation: Current applications and new directions*, 55–67.

Burstein, J. and M. Wolska (2003). Toward evaluation of writing style: finding overly repetitive word use in student essays. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pp. 35–42. Association for Computational Linguistics.

Carletta, J. (1996). Assessing agreement on classification tasks: The Kappa statistic. *Computational Linguistics 22*(2), 249–254.

Chang, C.-C. and C.-J. Lin (2001). *LIBSVM: A library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Choi, Y., E. Breck, and C. Cardie (2006, July). Joint extraction of entities and relations for opinion recognition. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, pp. 431–439. Association for Computational Linguistics.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement 20*(1), 37–46.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 1–8. Association for Computational Linguistics.

Collins, M. and N. Duffy (2002). New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 263–270.

Cortes, C. and V. Vapnik (1995). Support-vector networks. *Machine Learning 20*(3), 273–297.

Das, D., N. Schneider, D. Chen, and N. A. Smith (2010). Probabilistic frame-semantic parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, Stroudsburg, PA, USA, pp. 948–956. Association for Computational Linguistics.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman (1990). Indexing by latent semantic analysis. *Journal of American Society of Information Science 41*(6), 391–407.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, 1–38.

Dikli, S. (2006). An overview of automated scoring of essays. *The Journal of Technology, Learning and Assessment 5*(1).

Elliot, S. (2003). Intellimetric: From here to validity. *Automated essay scoring: A cross-disciplinary perspective*, 71–86.

Elsner, M., J. Austerweil, and E. Charniak (2007). A unified local and global model for discourse coherence. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pp. 436–443.

Falakmasir, M. H., K. D. Ashley, C. D. Schunn, and D. J. Litman (2014). Identifying thesis and conclusion statements in student essays to scaffold peer review. In *International Conference on Intelligent Tutoring Systems*, pp. 254–259. Springer.

Faulkner, A. (2014). Automated classification of stance in student essays: An approach using stance target information and the wikipedia link-based measure. *Science 376*(12), 86.

Florou, E., S. Konstantopoulos, A. Koukourikos, and P. Karampiperis (2013, August). Argument extraction for supporting public policy formulation. In *Proceedings of the 7th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, Sofia, Bulgaria, pp. 49–54. Association for Computational Linguistics.

Goudas, T., C. Louizos, G. Petasis, and V. Karkaletsis (2015). Argument extraction from news, blogs, and the social web. *International Journal on Artificial Intelligence Tools 24*(5).

Granger, S., E. Dagneaux, F. Meunier, and M. Paquot (2009). *International Corpus of Learner English (Version 2)*. Presses universitaires de Louvain.

Hasan, K. S. and V. Ng (2013, October). Stance classification of ideological debates: Data, models, features, and constraints. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, Nagoya, Japan, pp. 1348–1356. Asian Federation of Natural Language Processing.

Higgins, D. and J. Burstein (2007). Sentence similarity measures for essay coherence. In *Proceedings of the 7th International Workshop on Computational Semantics*, pp. 1–12.

Higgins, D., J. Burstein, and Y. Attali (2006). Identifying off-topic student essays without topic-specific training data. *Natural Language Engineering 12*(02), 145–159.

Higgins, D., J. Burstein, D. Marcu, and C. Gentile (2004). Evaluating multiple aspects of coherence in student essays. In *Human Language Technologies: The 2004 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 185–192.

Hyland, K. (2005). *Metadiscourse: Exploring interaction in writing*. Number London. Continuum.

Joachims, T. (1999). Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*, Chapter 11, pp. 169–184. Cambridge, MA: MIT Press.

Jurgens, D. and K. Stevens (2010). The S-Space package: An open source package for word space models. In *Proceedings of the ACL 2010 System Demonstrations*, pp. 30–35.

Kanerva, P., J. Kristoferson, and A. Holst (2000). Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, pp. 103–106.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science 220*(4598), 671–680.

Landauer, T. K. and S. T. Dumais (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 211–240.

Landauer, T. K., D. Laham, and P. W. Foltz (2003). Automated scoring and annotation of essays with the intelligent essay assessor. *Automated essay scoring: A cross-disciplinary perspective*, 87–112.

Levy, R., Y. Bilu, D. Hershcovich, E. Aharoni, and N. Slonim (2014). Context dependent claim detection. In *Proceedings of the 25th International Conference on Computational Linguistics*, pp. 1489–1500.

Lippi, M. and P. Torroni (2015). Context-independent claim detection for argument mining. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pp. 185–191.

Lippi, M. and P. Torroni (2016). Argument mining from speech: Detecting claims in political debates. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*.

Lodhi, H., C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. J. C. H. Watkins (2002). Text classification using string kernels. *Journal of Machine Learning Research 2*, 419–444.

Louis, A. and D. Higgins (2010). Off-topic essay detection using short prompt texts. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 92–95. Association for Computational Linguistics.

Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse 8*(3), 243–281.

Manning, C. D., M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60.

Marcu, D. (1999). Discourse trees are good indicators of importance in text. *Advances in automatic text summarization*, 123–136.

Marcu, D. (2000). *The theory and practice of discourse parsing and summarization*. MIT press.

McCallum, A. K. (2002). MALLET: A Machine Learning for Language Toolkit. `http://mallet.cs.umass.edu`.

Miltsakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay scoring systems. *Natural Language Engineering 10*(1), 25–55.

Moens, M.-F., E. Boiy, R. Palau, and C. Reed (2007). Automatic detection of arguments in legal texts. In *Proceedings of the 11th International Conference on Artificial Intelligence and Law*, ICAIL '07, New York, NY, USA, pp. 225–230. ACM.

Moschitti, A. (2004). A study on convolution kernels for shallow statistic parsing. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*, pp. 335–342.

Murakami, A. and R. Raymond (2010, August). Support or oppose? classifying positions in online debates from reply activities and opinion expressions. In *Coling 2010: Posters*, Beijing, China, pp. 869–875. Coling 2010 Organizing Committee.

Needleman, S. B. and C. D. Wunsch (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology 48*(3), 443–453.

Nelson, D. L., C. L. McEvoy, and T. A. Schreiber (2004). The university of south florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers 36*(3), 402–407.

Nguyen, H. and D. Litman (2016, August). Context-aware argumentative relation mining. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, pp. 1127–1137. Association for Computational Linguistics.

Ong, N., D. Litman, and A. Brusilovsky (2014, June). Ontology-based argument mining and automatic essay scoring. In *Proceedings of the First Workshop on Argumentation Mining*, Baltimore, Maryland, pp. 24–28. Association for Computational Linguistics.

Palau, R. M. and M.-F. Moens (2009). Argumentation mining: The detection, classification and structure of arguments in text. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law*, ICAIL '09, New York, NY, USA, pp. 98–107. ACM.

Park, J. and C. Cardie (2014, June). Identifying appropriate support for propositions in online user comments. In *Proceedings of the First Workshop on Argumentation Mining*, Baltimore, Maryland, pp. 29–38. Association for Computational Linguistics.

Parker, R., D. Graf, J. Kong, K. Chen, and K. Maeda (2009). *English Gigaword Fourth Edition*. Linguistic Data Consortium, Philadelphia.

Peldszus, A. and M. Stede (2015). Joint prediction in mst-style discourse parsing for argumentation mining. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 938–948.

Persing, I., A. Davis, and V. Ng (2010). Modeling organization in student essays. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 229–239.

Persing, I. and V. Ng (2013). Modeling thesis clarity in student essays. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 260–269.

Persing, I. and V. Ng (2014). Modeling prompt adherence in student essays. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1534–1543.

Persing, I. and V. Ng (2015, July). Modeling argument strength in student essays. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China, pp. 543–552. Association for Computational Linguistics.

Persing, I. and V. Ng (2016a, June). End-to-end argumentation mining in student essays. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, pp. 1384–1394. Association for Computational Linguistics.

Persing, I. and V. Ng (2016b). Modeling stance in student essays. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2174–2184.

Powers, D. E., J. C. Burstein, M. Chodorow, M. E. Fowles, and K. Kukich (2000). Comparing the validity of automated and human essay scoring. *ETS Research Report Series 2000*(2), i–23.

Rinott, R., L. Dankin, C. Alzate Perez, M. M. Khapra, E. Aharoni, and N. Slonim (2015). Show me your evidence - an automatic method for context dependent evidence detection. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 440–450.

Rooney, N., H. Wang, and F. Browne (2012). Applying kernel methods to argumentation mining.

Roth, D. and W.-t. Yih (2004). A linear programming formulation for global inference in natural language tasks. In *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning* (CoNLL-2004), pp. 1–8. Association for Computational Linguistics.

Sahlgren, M. (2005). An introduction to random indexing. In *Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering*.

Sardianos, C., I. M. Katakis, G. Petasis, and V. Karkaletsis (2015). Argument extraction from news. In *Proceedings of the 2nd Workshop on Argumentation Mining*, pp. 56–66.

Shermis, M. D. (2014). State-of-the-art automated essay scoring: Competition, results, and future directions from a united states demonstration. *Assessing Writing 20*, 53–76.

Shermis, M. D., J. Burstein, D. Higgins, and K. Zechner (2010). Automated essay scoring: Writing assessment and instruction. *International encyclopedia of education 4*, 20–26.

Shermis, M. D. and J. C. Burstein (2003). *Automated essay scoring: A cross-disciplinary perspective*. Routledge.

Silva, T. (1993). Toward an understanding of the distinct nature of l2 writing: The esl research and its implications. *Tesol Quarterly 27*(4), 657–677.

Sobhani, P., D. Inkpen, and S. Matwin (2015, June). From argumentation mining to stance classification. In *Proceedings of the 2nd Workshop on Argumentation Mining*, Denver, CO, pp. 67–77. Association for Computational Linguistics.

Somasundaran, S., J. Burstein, and M. Chodorow (2014). Lexical chaining for measuring discourse coherence quality in test-taker essays. In *COLING*, pp. 950–961.

Somasundaran, S. and J. Wiebe (2010). Recognizing stances in ideological on-line debates. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, CAAGET '10, Stroudsburg, PA, USA, pp. 116–124. Association for Computational Linguistics.

Song, Y., M. Heilman, B. Beigman Klebanov, and P. Deane (2014, June). Applying argumentation schemes for essay scoring. pp. 69–78.

Soricut, R. and D. Marcu (2006). Discourse generation using utility-trained coherence models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pp. 803–810.

Sridhar, D., J. Foulds, B. Huang, L. Getoor, and M. Walker (2015, July). Joint models of disagreement and stance in online debate. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China, pp. 116–125. Association for Computational Linguistics.

Stab, C. and I. Gurevych (2014a, August). Annotating argument components and relations in persuasive essays. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, Dublin, Ireland, pp. 1501–1510. Dublin City University and Association for Computational Linguistics.

Stab, C. and I. Gurevych (2014b, October). Identifying argumentative discourse structures in persuasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 46–56. Association for Computational Linguistics.

Stab, C. and I. Gurevych (2016). Parsing argumentation structures in persuasive essays. Technical report, arXiv preprint arXiv:1604.07370.

Swanson, R., B. Ecker, and M. Walker (2015, September). Argument mining: Extracting arguments from online dialogue. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, Prague, Czech Republic, pp. 217–226. Association for Computational Linguistics.

Teufel, S. (1999). *Argumentative Zoning: Information Extraction from Scientific Text*. Ph. D. thesis, University of Edinburgh,.

Thomas, M., B. Pang, and L. Lee (2006, July). Get out the vote: Determining support or opposition from congressional floor-debate transcripts. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, pp. 327–335. Association for Computational Linguistics.

Versley, Y., A. Moschitti, M. Poesio, and X. Yang (2008). Coreference systems based on kernels methods. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pp. 961–968.

Walker, M., P. Anand, R. Abbott, and R. Grant (2012). Stance classification using dialogic properties of persuasion. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 592–596.

Walton, D. (1996). *Argumentation Schemes for Presumptive Reasoning*. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Wang, Y.-C. and C. P. Rosé (2010, June). Making conversational structure explicit: Identification of initiation-response pairs within online discussions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Los Angeles, California, pp. 673–676. Association for Computational Linguistics.

Wei, Z., C. Li, and Y. Liu (2017). A joint framework for argumentative text analysis incorporating domain knowledge. Technical report, arXiv preprint arXiv:1701.05343.

Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emotions in language. *Language resources and evaluation 39*(2-3), 165–210.

Witten, I. and D. Milne (2008). An effective, low-cost measure of semantic relatedness obtained from wikipedia links. In *Proceeding of AAAI Workshop on Wikipedia and Artificial Intelligence: an Evolving Synergy, AAAI Press, Chicago, USA*, pp. 25–30.

Yang, B. and C. Cardie (2013, August). Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Sofia, Bulgaria, pp. 1640–1649. Association for Computational Linguistics.

Yang, Y. and J. O. Pedersen (1997). A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pp. 412–420.

Yessenalina, A., Y. Yue, and C. Cardie (2010, October). Multi-level structured models for document-level sentiment classification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, pp. 1046–1056. Association for Computational Linguistics.

BIOGRAPHICAL SKETCH

Isaac Persing obtained an MS in Computer Science in 2011 from The University of Texas at Dallas, a BS in Mathematics and Computer Science in 2008 from The University of Arizona South, and an AS in Computer Science from Cochise Community college in 2006. Isaac's areas of interest are natural language processing and data mining.

CURRICULUM VITAE

Isaac Persing
persingq@hlt.utdallas.edu

Interests

Natural Language Processing
Data Mining

Education

MS in Computer Science
The University of Texas at Dallas
August 2011
Intelligent Systems Track
GPA: 3.94

BS in Mathematics/Computer Science
The University of Arizona South
May 2008
GPA: 4.0

AS in Computer Science
Cochise College
May 2006
GPA: 4.0

Experience

Research Assistant and "Hourly Worker"
The University of Texas at Dallas
August 2008-May 2011, December 2011-May 2016, August 2016-Present
Worked on natural language processing tasks in the aviation safety, software engineering, social polling, and essay grading domains using machine learning

Teaching Assistant
The University of Texas at Dallas
August 2016-December 2016
Advised students, graded papers, and occasional taught classes for the Introduction to Machine Learning course

Graduate Intern
Oak Ridge National Laboratory
June 2016-August 2016
Built unstructured text analysis module for cyber security system using machine learning (https://is.gd/B4hCQJ)

Grants, Scholarships, and Other Awards

Excellence in Education Doctoral Fellowship
The University of Texas at Dallas
2015-2017

Jonsson Distinguished Research Fellowship
The University of Texas at Dallas
2008-2011

Outstanding Senior of the Year
The University of Arizona South
2008

Hispanic College Fund Google Scholarship
2007
$5000

National SMART Grant
2007
$4000

University of Arizona Grant
2007
$1790

Northrop Grumman Scholarship
2006,2007
$2200

Employee of the Year & Employee of the Month x 2
Windemere Hotel and Conference Center
2003,2004

Other Activities

Reviewer for Innovative Use of NLP for Building Educational Applications NAACL Workshop
2016

Reviewer for Computational Approaches to Causality in Language EACL Workshop
2014

Title V Summer Bridge Program mentor at UA South
2007

Computer Languages

Experienced with: Java

Familiar with: C, C++, Python, Unix shell scripting

Publications

Isaac Persing and Vincent Ng. 2016. Modeling Stance in Student Essays. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).

Isaac Persing and Vincent Ng. 2016. End-to-End Argumentation Mining in Student Essays. Proceedings of Human Language Technologies: The 2016 Conference of the North American Chapter of the Association for Computational Linguistics.

LiGuo Huang, Vincent Ng, Isaac Persing, Mingrui Chen, Zeheng Li, Ruili Geng, and Jeff Tian. 2015. AutoODC: Automated Generation of Orthoganal Defect Classifications. Automated Software Engineering, 22(1), 3-46.

Isaac Persing and Vincent Ng. 2015. Modeling Argument Strength in Student Essays. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers).

Isaac Persing and Vincent Ng. 2014. Vote Prediction on Comments in Social Polls. Proceedings of the 2014 Empirical Methods in Natural Language Processing.

Isaac Persing and Vincent Ng. 2014. Modeling Prompt Adherence in Student Essays. Main Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics.

LiGuo Huang, Vincent Ng, Isaac Persing, Mingrui Chen, Zeheng Li, Ruili Geng, and Jeff Tian. 2014. AutoODC: Automated Generation of Orthogonal Defect Classifications. Automated Software Engineering: Special Issue on Realizing Synergies in AI and SE.

Isaac Persing and Vincent Ng. 2013. Modeling Thesis Clarity in Student Essays. Main Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics.

LiGuo Huang, Vincent Ng, Isaac Persing, Ruili Geng, Xu Bai, Jeff Tian. 2011. AutoODC: Automated Generation of Orthogonal Defect Classifications. Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering.

Isaac Persing, Alan Davis, Vincent Ng. 2010. Modeling Organization in Student Essays. Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing.

Isaac Persing and Vincent Ng. 2010. Improving Cause Detection Systems with Active Learning. Proceedings of the 2010 Conference on Intelligent Data Understanding.

Isaac Persing and Vincent Ng. 2009. Semi-supervised Cause Identification from Aviation Safety Reports. Main Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing.