2013

Automated analysis of Learner's Research Article writing and feedback generation through Machine Learning and Natural Language Processing

Deepan Prabhu Babu
Iowa State University

Automated analysis of Learner's Research Article writing and feedback generation through Machine Learning and Natural Language Processing

by

DEEPAN PRABHU BABU

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Computer Science; Human Computer Interaction

Program of Study Committee:
Stephen Gilbert, Co-major Professor
Jin Tian, Co-major Professor
Elena Cotos

Iowa State University
Ames, Iowa
2013

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Teaching academic writing in English to native and non-native speakers is a challenging task. Quite a variety of computer-aided instruction tools have arisen in the form of Automated Writing Evaluation (AWE) systems to help students in this regard. This thesis describes my contribution towards the implementation of the Research Writing Tutor (RWT), an AWE tool that aids students with academic research writing by analyzing a learner's text at the discourse level. It offers tailored feedback after analysis based on discipline-aware corpora.

At the core of RWT lie two different computational models built using machine learning algorithms to identify the rhetorical structure of a text. RWT extends previous research on a similar AWE tool, the Intelligent Academic Discourse Evaluator (IADE) (Cotos, 2010), designed to analyze articles at the move level of discourse. As a result of the present research, RWT analyzes further at the level of discourse steps, which are the granular communicative functions that constitute a particular move. Based on features extracted from a corpus of expert-annotated research article introductions, the learning algorithm classifies each sentence of a document with a particular rhetorical move and a step. Currently, RWT analyzes the introduction section of a research article, but this work generalizes to handle the other sections of an article, including Methods, Results and Discussion/Conclusion.

This research describes RWT's unique software architecture for analyzing academic writing. This architecture consists of a database schema, a specific choice of classification features, our computational model training procedure, our approach to testing for performance evaluation, and finally the method of applying the models to a learner's writing

sample. Experiments were done on the annotated corpus data to study the relation among the features and the rhetorical structure within the documents. Finally, I report the performance measures of our 23 computational models and their capability to identify rhetorical structure on user submitted writing. The final move classifier was trained using a total of 5828 unigrams and 11630 trigrams and performed at a maximum accuracy of 72.65%. Similarly, the step classifier was trained using a total of 27689 unigrams and 27160 trigrams and performed at a maximum accuracy of 72.01%. The revised architecture presented also led to increased speed of both training (a 9x speedup) and real-time performance (a 2x speedup). These performance rates are sufficient for satisfactory usage of RWT in the classroom. The overall goal of RWT is to empower students to write better by helping them consider writing as a series of rhetorical strategies to convey a functional meaning. This research will enable RWT to be deployed broadly into a wider spectrum of classrooms.

CHAPTER 1. INTRODUCTION

1.1 Overview

The Research Writing Tutor (RWT) is an intelligent tutoring system developed at Iowa State University that assists students in learning academic writing. The RWT analyzes discourse according to discipline-specific research article genre and offers individualized feedback. With training data from an annotated corpus of 900 articles across 30 different disciplines, the tool uses natural language processing techniques and machine learning to automatically classify discourse markers. Using this expert knowledge captured through annotation, RWT analyzes a learner's writing and offers feedback towards improvement.

Creating the RWT poses a number of challenges. How to identify the rhetorical structure of a student's writing? What is an appropriate user interface for this software? What kinds of feedback can be provided for better writing? How are wrong classifications handled? Could manual rules augment the classification and are they practically viable? What is the best algorithm presently that could classify rhetorical structure practically fast? Are SVMs really the best machine learning algorithm for identifying rhetorical moves and steps? Are rhetorical features really related to the corresponding rhetorical moves and steps through a Gaussian function? What is the optimal feature vector size that could accurately identify the rhetorical functions? Could some other weighing measures do the trick with fewer feature vector size and eventually faster? With all these important questions to address, this thesis focuses on some of the machine learning challenges in RWT, namely, effective feature identification and weighing measure that could help identify the different rhetorical

functions accurately and an optimal machine learning algorithm that could be trained to use the features to predict rhetorical functions with usable performance.

1.2 Automated Writing Evaluation

The RWT is an Automated Writing Evaluation (AWE) tool. Sophisticated AWE systems analyze learner writing and offer immediate feedback regarding grammar, style, and other features of the text. The automated systems that offer feedback are considered cost effective ways to replace or enhance instructor feedback. These systems are involved in range of applications from reports on grammatical errors for ESL learners to evaluating learner essay writing holistically from content, organizational and mechanical characteristics. Considering a group of learners, automated writing evaluation offers an economic alternative to expensive hand scored assessment and feedback. Outside the classroom setting, a major drive to push evaluation to automated tools, cited by educational testing organizations has been to test content knowledge and writing competence at a larger scale. Teams of human raters are a costly investment to train and automated systems that could human raters would reduce overall cost. Successful implementations of AWE tools are already prevalent in the market used with a wide variety of applications. Intellimetric (Elliot, 2003) scores learner essays based on a list of 500 features indicating content, complexity grammar and so on. Intelligent Essay Assessor (Foltz, Laham, & Landauer, 1999) uses latent semantic analysis for deriving likelihood and relation of vocabulary used to the context. E-Rater (Attali & Burstein, 2006) uses 12 different feature variants to assess and score learner essays.

None of these tools, however, attempt to teach academic writing, as does RWT. Articulating ideas according to conventions of academic writing in English are challenging

both to native and non-native speakers. Nonnative speakers especially suffer as they tend to be less expressive, restricted to simpler style of writing, and often provide fewer claims than the appropriate amount for their research (Flowerdew, 1998). Huang (2010) documented the lack and limitation of tools that aid in discipline-specific research and writing. RWT is a discipline-specific tool built upon a corpus which is representative of particular genre through natural language processing techniques.

1.3 Perceived Issues with AWE Tools

AWE tools inherently pose limitations in implementations, such as favoring lengthiness of writing, assessing higher scores to certain types of lexico-grammatical features, lacking measures to identify illogical or incoherent writing and generating unspecific feedback during progress (Herrington & Moran, 2001; N. D. Yang, 2004). Most comments and feedback provided by AWE tools are formulaic, generic information requiring to be augmented through specific personal comments from human instructors. Formulaic responses also may encourage students to adjust their writing to the scoring criteria of the analysis engine.

AWE tools increasingly rely on surface features of responses without considering creativity or the actual content specified by the responses. They also pose vulnerability as they are easily cheated when underlying knowledge of training corpus was known. (Yongwei Yang, Buckendahl, Juszkiewicz, & Bhola, 2002) Certain tools provide unfair evaluation despite well-organized learner writing responses owing to poor mechanics (Calfee, 2000).

Finally, AWE tools are validated through their own internal measures and the accuracy of the learning model they are built on rather than the learning and through the

teaching impact they produce. Though inter-rater agreement may be high for AWE tools, their application in academic contexts must be evaluated as well. Neglecting the real-world application of the tools and their impact on learning make such research studies limited and outcome-based (Warschauer & Ware, 2006).

All these issues are related to AWE use, especially when it is not principled. It is therefore important to consider the specific learning needs in specific learning contexts when new AWE tolls are developed in order to avoid pitfalls like those outlined above.

1.4 Problem Statement

RWT, being an intelligent tutor, addresses some difficulties faced by learners of academic writing by analyzing at the discourse level considering organization of learner responses. It also offers individualized discipline-specific feedback across 30 different disciplines. Rhetorical moves are the particular communicative function performed by a text section. Rhetorical steps are finer elements of text that realize a rhetorical move. RWT extends previous research on an AWE tool, The Intelligent Academic Discourse Evaluator (IADE) (Pendar & Cotos, 2008) by analyzing academic research articles through rhetorical moves and the steps that collectively form the move.

The challenges particular to RWT include correctly identifying the writer's moves and steps, giving the writer the appropriate feedback, and conveying to the writer the rhetorical norms of his or her discipline. RWT is implemented for all sections of academic research articles. IADE was restricted to identifying the different moves from a learner's text but not the granular steps which comprise a move. IADE generated textual feedback with numerical percentages indicating the current state of a user's writing and the remaining goals

to be achieved. This format of feedback was not yet optimized for the user, and RWT aspires to improve on that communication. More importantly, IADE was implemented only for the introduction section of academic research articles, while RWT can generalize across different article sections (e.g., methods, results, and discussion/conclusion).

1.5 Research Questions

This research addresses the following questions.

1. Can a machine learning system be developed to accurately identify the rhetorical steps within an academic writer's text?
2. What combination of unigram and trigram features from the training corpus are most appropriate for optimal move and step classification?
3. How is the number of features related to the accuracy, precision and recall of the prediction task, e.g., is it a linear relationship?
4. Is it feasible to use a single classifier for identifying moves and steps across academic disciplines by using discipline-specific classification features within the classifier? If so, how accurate can this approach be?
5. Is odds ratio an appropriate feature identification and weighing measure for identifying the different rhetorical moves and steps?

1.6 Thesis Organization

This chapter is an introduction to automated writing evaluators, the perceived issues limitations and issues posed by various AWE tools. It also documents how IADE tried to solve some of the issues specifically in academic writing genre and how RWT is building on top of it through improved functionality and research. In this thesis I also document how the

synergy of overlapping fields including pedagogy, intelligent tutoring systems, genre analysis, natural language processing and machine learning address some of the functionalities for RWT.

Chapter 2 covers the literature backing the research from various fields explaining the main concepts used with the implementation of RWT. Chapter 3 documents the system architecture of the different modules within RWT's backend responsible for analyzing the data and deriving computational models capable of predicting the rhetorical organization. Chapter 4 explains the research approach and the preprocessing, training and testing procedures involved in building a classifier. Chapter 5 contains the results of experiments and metrics for evaluating the performance of various classifiers built. Chapter 6 summarizes and discusses the results along with directions to future research.

CHAPTER 2. LITERATURE REVIEW

Research Writing Tutor (RWT) derives its focus and approach from four main areas including pedagogy, intelligent tutoring systems, discourse analysis and automated text categorization. We review various theoretical concepts on which RWT's functionality is implemented.

2.1 Computer Assisted Instruction in Pedagogy

RWT aims to assist learners of research article writing by providing expert feedback on their writing skills and possible scope for improvement. Pedagogical assistance from computers has been proven to augment the regular teaching process. Individualized drill and practice to learners, tutoring of new content and feedback based dialogue with instruction are some of the successful applications of computers in a classroom scene (Suppes, 1980). Significant contributions towards effective teaching and positive attitudes of students towards instruction have been attributed to computers at college teaching. In fact, computers were quick at their application when compared with conventional teaching methods (Kulik, Kulik, & Cohen, 1980). Computer assisted language learning is a specific area of application related to pedagogy in college teaching. Each individual learns a language differently due to distinct idiosyncratic learning strategies, cognitive abilities and various affective factors (Dörnyei & Skehan, 2008). Hence, Computers have been proposed as a powerful alternative to address the individuality and instruct accordingly (Britt, 1967). Learning academic writing is a natural extrapolation to language learning and computers could assist accordingly.

2.2 Intelligent Tutoring Systems

RWT's audience consists of users attempting to learn research article writing differing in needs and requiring independent specialized instructions. One of RWT's goals is to engage learners in sustained reasoning activity and actively communicate to lead the learner to a better understanding of the subject being tutored, being made possible through intelligent tutoring systems. The intelligent tutoring system (ITS) would interact with the learner through a series of instructions and provide individualized feedback based on their actions. This is done by considering a human tutor as an educational model and applying various artificial intelligence techniques to realize it in a computer system. Thus the "instructor in the box" applies various strategies to reduce the difference between the expert in the field and a new learner in the subject.

2.2.1 Foundations

ITS lays it foundations on set of independent modules each carrying out a specific functionality. An expert module (Richardson, 1988) forms the main backbone of domain knowledge and captures the underlying intelligence behavior. Cognitive expert modules (Anderson, 1988; Richardson, 1988) aim to simulate the actual human problem solving capability in a domain. They also strongly consider psychological components essential for tutoring. A learner diagnosis module (Richardson, 1988) is used to infer the learner's understanding of the subject being tutored. The inference from this module is used to generate individualized feedback/action ranging from increasing complexity to offering crafted feedback towards improvement. Better diagnosis from this module also results on how close it references learner's misconceptions, erroneous and incorrect knowledge apart

from understanding. Pedagogical module is responsible for structuring and sequencing instructional content, messages and interventions in the ITS. The main challenge involves separating instruction from the content expertise and offering them alongside for the betterment of the learning (Anderson, 1988). The final user facing module involves the Human computer interface module. This interface is implemented for a transparent learning experience to empower learner to act independently and access the expertise it encapsulates. This module also integrates the other modules forming a single package suiting content to interface and vice versa.

2.2.2 Automated Essay Scoring using Tutors

Automated essay scoring through intelligent tutoring systems and providing appropriate feedback are important realms in essay writing. A wealth of tool implementations with strong research is prevalent in this space. Major ITS tools aiding automated essay scoring use a variety of features to analyze, assess and score essay writing.

Elliot's comprehensive specification of Intellimetric (Elliot, 2003), an automated essay scoring tool describes a model with 500 component features based on content, grammar, text complexity, sentence and word variety to intelligently assess user essays and automatically score them. Applying latent semantic dimension, the model determines correlation of candidate content to a modeled vocabulary collection. The information in terms of features gleaned from user writing is used to predict expert human score using a series intelligent mathematical model. A final score is arrived by optimizing the individual scores.

Intelligent Essay Assessor (Foltz et al., 1999) is yet another intelligent student essay scoring and assessment tool. It uses a combination of Latent semantic analysis and a reference database related to the domain to arrive at the likelihood of the vocabulary used in a written essay and its relation to the context. For e.g. Essays on the topic of computers and related contexts are weighed using a Computer science reference textbook in digital form. Relevant vocabulary is awarded a better score through this procedure. Essay valuation and scoring are done using a variety of techniques including comparing with pre-graded essays, gold standard ideal essays, comparing to portions of original text or subcomponents and comparing individual sentences against a reference textbook. An immense hardware and software requirement of the tool makes it difficult to deploy on desktop systems and hence is available for use as a web based tool only. Heavy use of statistical techniques to assess essays makes it difficult to communicate the inner structure of the scoring model eventually a threat to the scoring. Also statistical methods make assumptions of variance of prediction scores which may be different from the actual variance.

E-Rater v2.0 scoring system (Attali & Burstein, 2006) uses a combination of 12 different features to score essays. Six different areas of analysis including errors in grammar, usage , mechanics, style , discourse structure and vocabulary content are collectively used in order weigh the actual characteristics of writing. A sample of human scored essay data is used to identify and weigh in features that correspond to various human scoring criteria. These are used for fitting a model to the training data using a multiple regression procedure. By varying the sample data, appropriate models can be built for specific writing categories like grade level, topic level, thesis writing etc. The system shows high agreement with human scoring, correlation of scores between different prompts and detailed light on the scoring

process and its validity. Though functionally confirming, the system is far from covering all aspects of writing quality which could further be improved by assisting with better feature engineering measures. The model fitting procedure is also biased to identify good faith essays and performs poorly with anomalous and off-topic essay entries. Using human scored samples to build the model doesn't equip the system to identify different patterns of writing exhibited by potentially different groups of users for e.g. users of Asian ethnic background have a different pattern of writing compared to the sample essays from native English speakers.

Intelligent academic discourse evaluator IADE (Pendar & Cotos, 2008) is a genre based automated text analysis and feedback tool which targets research article introduction texts. It classifies learner's article sentences into communicative moves based on Swales framework. Similar to E-Rater, sample human annotated research article introductions are used to identify important features corresponding to communicative modes and a model is fit using support vector machines. IADE also offers a variety of informational feedback in terms of color coding and distribution statistics facilitating learner's writing process.

Yet another approach to building intelligent tutoring systems involves Effort based tutoring (Arroyo, Mehranian, & Woolf, 2010) where a student's engagement, domain knowledge, affect and meta cognition are integrated along with different dimensions of student behavior to make optimal pedagogical decisions. Specifically, student's effort at different practice items is used to distinguish student behavior. Deriving an empirical estimate of effort along with difficulty of practice item is used to model pedagogical feedback behavior by the system. The modularity of the tutoring procedure allows usage in several learning environments and domains.

The current research in RWT extends previous work done on Intelligent academic discourse evaluation Tool (IADE) (Pendar & Cotos, 2008). IADE analyzes research articles and offers constructive feedback to learner's discourse at move level. RWT's goal is to analyze research articles at move level and specifically at the granular step level which collectively form a particular move. The approach is very similar in procedure to one used by IADE but uses a series of support vector machines in order to classify learner article sentences into particular moves and steps.

2.3 Discourse Analysis

For RWT, high quality analysis of research articles is necessary to offer specialized feedback to learners. A possible methodology is to consider and dissect learner articles at the discourse level as they reveal innate organization and structure. Discourse analysis of academic articles reveals how they are organized, carried and reproduced in a particular way and as required in certain institutional practices. Knowledge at discourse level is essential to users learning research writing to compensate sentence level processing difficulties. In efforts to find basic text structures, a four part model of 'Situation', 'Problem', 'Solution' and 'Evaluation' was identified by (Hoey, 1979) which was not related to a particular discipline or specific text types, and could capture textual structure appropriately. It could be applied to wide range of disparate discourses. A formal schema is highly helpful in such cases as studied by Swales in 1990, through his research of nonnative speaker graduate students and their writing practices.

2.3.1 Rhetorical Structure Theory

A descriptive framework for analysis of discourse and text was the Rhetorical structure theory (Mann & Thompson, 1988) for linguistically describing natural text and characterizing their structure as relations among different parts of text. It also captures the transition point, hierarchy and extent of relation among parts of text.

Rhetorical structure theory (RST) accounts for textual coherence independent of lexical and grammatical forms of text. This is made possible by identifying "discourse markers" which are indicators of rhetorical relations in text. Functionally, Rhetorical relations are the effect a writer intends to achieve by having two spans of text alongside.

The asset of RST is the claim that it is a sufficient basis for analyzing vast majority of text in English language with minimal exceptions.

2.3.2 Genre Analysis

In the research article genre, the discourse structures in a section were related to communicative functions of text, resulting in analysis through rhetorical moves. A text section which performed a particular communicative function was termed a "Move". Moves being functional units, collectively come together to attain the communicative purpose of the particular genre (Douglas Biber, Connor, & Upton, 2007). Swales (J. Swales, 1981) conducted various studies on organizational patterns of research articles, to categorize discourse units within a text to their rhetorical moves.

Frequently used moves are considered conventional part of the genre whereas rare once are optional moves. Moves indeed are realized through finer elements of text which are

termed as “Steps” by Swales (J. M. Swales, 1990). Series of steps achieve the communicative purpose of a particular move to which it belongs.

2.3.3 Move Analysis of Research Articles

Swales (J. M. Swales, 1990) proposed a series of moves and steps that defined the rhetorical structure of Research article introductions. The move and step structure appropriately captured the interactions between them, apart from performing communicative functions in scientific texts. The flexibility of the structure allowed moves and steps to reoccur cyclically; hence each appearance was considered a separate occurrence. Swales conducted the move analysis on a series of 48 introduction section texts, from research articles spanning over multiple disciplines. The structure was evidently discipline independent.

Create a Research Space (CARS) by Swales is a three move model which is largely likely in most research article introductions.

Table 2-1 CARS model for research article introductions, adapted from Swales (1990, p.141)

Move 1:	**Establishing a territory**	
	Step 1	Claiming centrality and/or
	Step 2	Making topic generalizations and/or
	Step 3	Reviewing items of previous research
Move 2:	**Establishing a niche**	
	Step 1A	Counter-claiming or
	Step 1B	Indicating a gap or
	Step 1C	Question raising or
	Step 1D	Continuing a tradition
Move 3:	**Occupying the niche**	
	Step 1A	Outlining purposes or
	Step 1B	Announcing present research
	Step 2	Announcing principal findings
	Step 3	Indicating RA structure

The move analysis implies the existence of definable and predictable moves within text that makeup a particular genre. This idea could be applied back, to teach novice writers to write in particular genre, by considering each step and move as building blocks and structuring writings around them (Dudley-Evans, 1995) .

2.4 Automated Text Categorization

RWT approaches move analysis, by categorizing stretches of learner text into communicate functions of moves and steps automatically. Text categorization (Sebastiani, 2002) (aka Text classification or Topic spotting) is the automated assignment of topical categories to natural language texts or documents based on their content and relevance. Detecting patterns of similarity is thus of central importance in natural language processing tasks. Abundance of digital documents lately and a growing necessity towards their management has brought prominence towards text categorization techniques. Most implementations of text categorization involve either human engineering or statistical learning methodologies or a combination of two. RWT uses statistical means to learn the rules of underlying classification task through expert annotated samples. Following sections are basic underlying concepts related to RWT's implementation of text categorization.

2.4.1 A Formal Definition

Mathematically, the text categorization problem may be defined as task of assigning Boolean value to each possible pair of document and a category.

When $D = \{d_1, d_2, \ldots., d_n\}$, represents the domain of documents and C is the set of predefined topics/categories $C = \{c_1, c_2, \ldots., c_n\}$, where $\langle d_j, c_i \rangle \in D \times C$. A value of T

assigned to $\langle d_j, c_i \rangle$ if d_j indeed belongs to category c_i and a value of F is assigned to $\langle d_j, c_i \rangle$ otherwise.

If we are able approximate the unknown function, $: D \times C \rightarrow \{T, F\}$, we have a classifier which could automatically classify pair of $\langle d_j, c_i \rangle$ as T or F , leading to decide if document d_j belongs to c_i.

Important assumptions include,

- That categories are indeed just topical symbols and do not contain any helpful information.
- There is no knowledge from outside to help with the classification. All that is available is set of documents, its contents and labels alongside.

2.4.2 Single-label vs. Multi-label Text Categorization

Initial implementation of RWT uses single label categorization where each stretch of text is assigned a single move and a single step. When a particular categorization task forces only a single category to be assigned to each document, it is considered as a single-label categorization (aka non-overlapping categories). When the categorization task allows multiple categories to be associated to each document, it is considered as a Multi-label categorization problem (aka overlapping categories) (Sebastiani, 2002).

The single-label case is considered more important as it can be used to implement a multi-label categorization but the converse cannot be done. Thus, when we have a classifier that can classify a document with a category as true or false, it can be applied to all categories to find the multiple labels possible for a document.

Hence the above formal definition of text categorization problem can be described as consisting of $|C|$ independent categorization problems , each classifying a document in D to a particular category c_i , for i = 1,…, $|C|$. Thus, each classifier for c_i approximates, $\phi_i: D \rightarrow \{T, F\}$.

2.4.3 Document-pivoted vs. category-pivoted text categorization

RWT's perspective of text categorization is document-pivoted (DPC) wherein an input document is analyzed to be placed into possible categories. On the other hand given a particular category, all documents that could possibly belong to the category may also be analyzed; this is termed as category-pivoted categorization (CPC). Applying DPC is evident for those text categorization tasks when documents are not readily available but gradually over a period of time, e.g. email filtering. CPC is applicable when a particular category is newly added to set of existing categories in a categorization task and documents are required to be reclassified according to the new set of categories e.g. photo tagging.

2.4.4 Machine learning approach to Text Categorization

Text categorization in RWT is implemented through Machine learning (Sebastiani, 2002),which involves an inductive process resulting in a *classifier* for a particular category c, through the observation of characteristics of documents classified as category c or $\bar{c}$, by an expert in the domain. The approach of coming up with a classifier by supervising the learning using a set of samples, is an example of *supervised* learning.

Mitchell (Mitchell, 1997) defines Machine Learning as,

A Computer program is said to learn from experience E with respect to some class of tasks T and performance measures P, if its performance at tasks in T, as measured by P , improves with experience E.

ML approach is completely based on the existence of an initial corpus of manually categorized document samples $\Omega = \{d_1, d_2, \ldots, d_n\}$ categorized under $C = \{c_1, c_2, \ldots, c_n\}$. Hence, the total function $\hat{\phi} : D \times C \rightarrow \{T, F\}$ is completely defined for every pair $\langle d_j, c_i \rangle \in \Omega \times C,\ \Omega \subset D$. When $\hat{\phi}(d_j, c_i) = T$, we have a positive example of a document from category c_i. When $\phi(d_j, c_i) = F$, we have a negative example of a document from category c_i.

2.4.4.1 Training set and Test set

The initial corpus is split into two independent subsets with elements distributed randomly, of unequal size.

1. *Training set* – This is the set of documents and associated categories used for observing characteristics and inductively building (training) the classifier through supervision.

2. *Test set* – This is the set of documents for testing the efficiency of classifier which was built using the training set. Testing involves comparing the outcome category of the classifier on an input document with the actual manually assigned category. Effectiveness of the classifier is reflected by measuring how often the two categories match.

2.4.4.2 Cross Validation

Cross validation (aka rotation estimation) is usually performed, to assess how a classifier's performance would generalize on an independent data set. It is usually performed

to estimate the accuracy of an inductive process while applied in practice. k-fold cross validation (Mitchell, 1997, p. 146), involves inducing k different classifiers $\phi_1, \phi_2 \ldots, \phi_k$ from an initial corpus Ω , by splitting it into k-disjoint equal-sized subsets and iteratively considering k-1 sets as training sets and remaining set as a test set. The final performance of inductive process is obtained by averaging the performance measures of the k-different classifiers.

2.4.4.3 Text Representation

In Text categorization, compact representation of textual material is of paramount importance as it directly affects the efficiency of the inductive process of building a classifier. When text is represented concisely, processing is faster in turn speeding up the supervised training of the classifier.

As in information retrieval, a sentence is represented as a vector of weights, $d_j = \langle w_{1j}, w_{2j}, \ldots w_{|T|j} \rangle$, of dimension $|T|$, where T is known as a set of *features*. Features are individual measureable properties of observed phenomena used for learning.

A typical scheme which applies the above representation is the *bag of words* model. Here the set of distinct words in whole corpus, are indexed without concern to their order of occurrence. Thus the set of words become the features of the model. Each sentence is then represented as a bit vector of the set of all words, having a weight of '1', if the word appears in the sentence and weight of '0' otherwise; This is fed as input to the induction process for building the classifier.

Apart from using binary weights, other schemes practically use some weighing scheme, with normalized scores occurring between 0 and 1. Using the actual count of occurrence of a particular word in a sentence as weight is a well applied weighing scheme.

2.4.4.4 Feature Reduction Techniques

Feature reduction procedures are applied in RWT when training a classifier to reduce feature space while maintaining the performance of the classifier or improving it. It involves a set of transformations and combinations performed on the original feature set in order to identify features of high innate informational value (D. Biber, Conrad, & Reppen, 1998). Common feature reduction procedures used by RWT are discussed in detail.

2.4.4.4.1 Stop Words Removal

Apart from a compact sentence representation, stop words removal is another procedure that could reduce the amount of data processing involved. Before processing the whole corpus towards sentence representation, it is customary to remove all words of low informational value. Several stop lists are readily available and are usually containing common grammatical or functional words such as ‘the’, ’of’, and ‘in’. Use of such lists is common across information retrieval systems, as their removal rarely cause a significant loss of accuracy (Y. Yang, 1995).

2.4.4.4.2 Term Frequency

Frequency of occurrence of words is a useful measure to assess the relative importance of terms in documents. The terms with higher frequency are of higher importance (Rijsbergen, 1979). A natural way to reduce feature space is to consider terms having occurrence higher than certain threshold frequency. By considering words of frequency 2 or

more, the feature space would be reduced to almost half as single frequency terms normally predominate in a corpus.

2.4.4.4.2 Relative Term Frequency

Term frequency considers global frequency of terms throughout the whole corpus. Alternatively, term frequencies could be measured relatively among different categories within in the corpus. Thus, terms whose frequency is different across different categories are considered more important rather than terms with high frequency in all the categories. This relativity gives a measure of difference and affinity of a term to a single category. To consider those words that appear only in a single category is a natural extrapolation but they are almost rare offering poor performance than expectations.

Some measures of relative importance used in NLP and machine learning include Information Gain (IG), Mutual Information (MI), T-test, Odds ratio and Chi-Squared (CHI). (For a review of these and other similar measures, see (Y. Yang & Pedersen, n.d.) and (Ikonomakis, Kotsiantis, & Tampakas, 2005))

Odds ratio (Bland & Altman, 2000) in particular used by RWT for relative term frequency, is a measure of effect size, which describes the correlation between two classes or categories. It can be used effectively as a relative term frequency measure to study correlation of certain terms occurring prominently in certain specific categories. Occurrence of these terms indirectly signals association with certain categories.

$$\mathrm{OddsRatio}(f_i, c_j) = \log \frac{P\left(\frac{f_i}{c_j}\right)\left(1\text{-}P\left(\frac{f_i}{\neg c_j}\right)\right)}{P\left(\frac{f_i}{\neg c_j}\right)\left(1\text{-}P\left(\frac{f_i}{c_j}\right)\right)}$$

$P(f_i/c_j)$	Probability that observed term f_i belongs to class c_j
$P(f_i/\neg c_j)$	Probability that observed term f_i does not belong to class c_j

2.4.4.5 Term Weighing

Another approach towards reducing the feature space is to weigh features based on some informational measure and trim those which score low on the scale. This could also be combined along with other feature reduction procedures to better manage the feature space. A simple measure is considering the frequency of occurrence of a term in a corpus as a weight of the term.

Yet another popular weighing scheme is the standard $tf - idf$, *term frequency – inverse document frequency* measure (Croft, 1987). This measures the importance of a particular term or word occurring in a document of the corpus, using *term frequency* which is the number of times a term occurs in a document and *inverse document frequency* which is the log of the total number of documents in the initial corpus over the number of documents containing the term.

$tf - idf(t_i, d_j)$ =

(number of times t_i occurs in d_j)* $\log_2 \frac{|total\ number\ of\ documents\ in\ corpus|}{number\ of\ documents\ where\ t_i\ appears}$

Here t_i – a term or word from the corpus, and d_j- a sentence or document from the corpus.

The downside of using this measure (and most other measures) is it considers the importance of a term from a frequency and occurrence perspective rather than weighing syntactical role or considering into account the order of occurrence of a term.

2.4.4.6 Support Vector Machines - Learner Classifier Systems

The core system driving RWT to learn from expert annotated samples and predict for new samples, is the learner classifier system. Learner classifier systems learn to perform the best action given its input based on conditions. The core of the learning systems is an algorithm to process the input data to produce a representation of target knowledge for requested operation. Once the representation is in place, this can be applied to new test data, to obtain a general hypothesis about the data in terms of rules/concepts learnt.

Numerous algorithms exist to learn from data and have different assumptions or inductive bias according to the learning context (Mitchell, 1997). Hence some algorithms perform in certain areas and contexts better than others and choosing them accordingly affects their performance in the learning tasks. The bias of algorithms plays a key role in their applications.

Support vector machines (SVM) (Cortes & Vapnik, 1995) used by RWT, are a class of learner classifier algorithms which are based on the structural risk minimization principle from computational learning theory (Vapnik, 1999).

The main idea with a SVM is to identify a hypothesis *h* of lowest true error possible and it is guaranteed through structural risk minimization. True error is the probability that a hypothesis *h* will make an error on an unseen randomly selected data sample. Error of hypothesis *h* on a given training sample and the complexity of the actual hypothesis space containing *h* namely H, can be used to calculate an upper bound on the true error. This upper bound can be further optimized, by minimizing the bound on the true error and SVMs achieve this by controlling the dimension of hypothesis space H. Notice however, the

dimensions of H are in VC dimension. VC dimension is a measure of capacity of a learning algorithm. It is measured as the cardinality of largest set of points an algorithm can shatter.

SVMs basically learn linear functions but they can be adapted to learn complex functions using kernels. Kernel trick (Hofmann, Schölkopf, & Smola, 2008), maps observations/features to a higher dimensional inner product space where observations are linearly separated using a hyperplane.

SVMs classify by mapping input/observations/features of various categories or classes to a high dimensional feature space through the use of a non-linear mapping function chosen as a priori. In this high dimensional space, a linear surface is used to classify features according to their relevant categories. Thus optimally all features of same category are separated to the same side of the linear surface. The construction of the linear surface holds special properties to ensure generality.

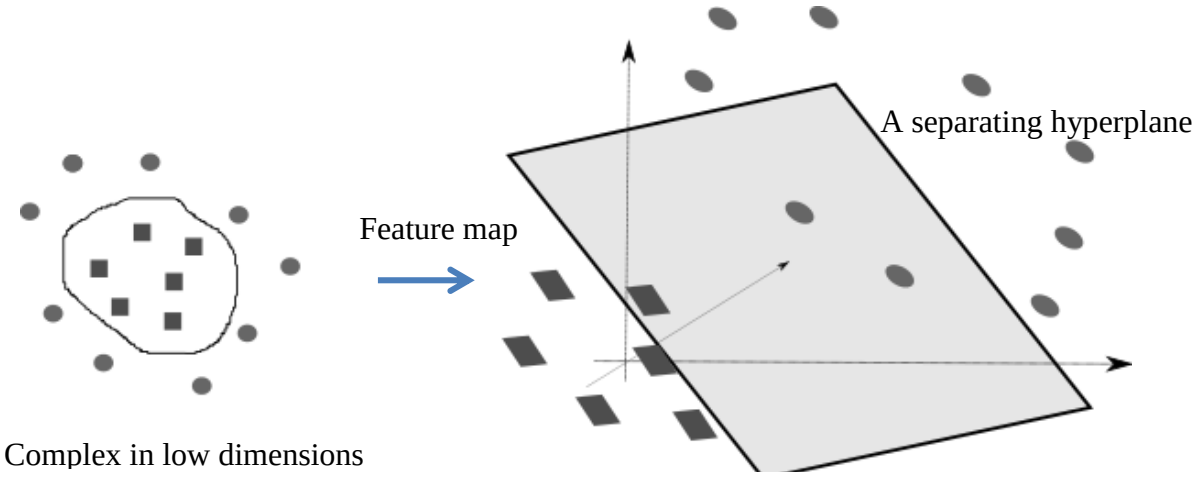

Figure 2-1 Hyperplane separates data of two

An optimal linear surface (Cortes & Vapnik, 1995; Vapnik, 1999) or hyperplane for separable categories separates features of different categories using a maximum margin.

Hence only a few features which lie close to the maximal margin are taken into consideration to actually determine the margin mathematically. These features which determine the optimal maximum margin hyperplane decision boundary are named the support vectors.

SVMs measure complexity of the hypothesis through the maximal margin and optimizing the maximal marginal width. Hence the dimensions of the feature space are irrelevant to the complexity aiding high dimensional feature vector applications.

Some theoretical concepts justifying application of SVMs to our learner's research article analysis and text categorization are discussed in detail.

2.4.4.6.1 Text categorization applications

RWT uses SVMs for identifying moves and steps for stretches of text, as the text categorization problem has high dimensional features and most features carry useful information to be discarded irrelevant (Joachims, 1998). SVMs are said to work better for text categorization applications (Joachims, 1998) as they exhibit the following properties necessary for building text classifiers. SVMs can handle high dimensional features (minimum 10,000) as their complexity depends on the composing hypothesis space and not the size of the features. Hence high dimensional feature spaces help consider more features at any point of time. Bag of words model applied to text categorization almost always yields sparse document vectors or feature vectors. Algorithms having similar inductive bias like SVMs are said to perform well for text categorization applications , empirically and theoretically (Kivinen, Warmuth, & Auer, 1997). Most text categorization problems could be solved linearly; SVMs are for finding linear boundaries of maximum marginal separation.

Thus above arguments theoretically provide evidence for the role of SVM and their application in text categorization problems.

2.4.4.6.2 Practical selection of SVM Parameters

Theoretically, SVM Regression is formulated as a minimization of the following function (Vladimir Cherkassky & Ma, 2004),

$Minimize \frac{1}{2}|\omega^2| + C\sum_{i=1}^{n}(\xi_i^* + \xi_i)$

Subject to

$y_i - f(x_i, \omega) - b \leq \varepsilon + \xi_i^*$, $f(x_i, \omega) - b - y_i \leq \varepsilon + \xi_i$, $\xi_i^*, \xi_i\,(Slack\ variables) \geq 0$

Here, C is a positive constant known as the regularization parameter and ε controls the $\varepsilon - insensitive\ loss\ function$ (Vapnik, 1999) .

$$f(\boldsymbol{x}, \omega) = \sum_{j=1}^{m} \omega_j g_j(\boldsymbol{x}) + b$$

$\boldsymbol{x}\ is\ a\ multivariate\ input$, $g_j(\boldsymbol{x})$ is a set of nonlinear transformation and 'b' is the bias term. The above minimization problem can be solved from the dual problem and its solution is given by,

$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(x_i, \boldsymbol{x}) + b$, subject to the constraints

$0 \leq \alpha_i, \alpha_i^* \leq C$ (dual variables) . $K(x_i, \boldsymbol{x})$ is the symmetric kernel function , satisfying Mercer's conditions (Vapnik, 1999).

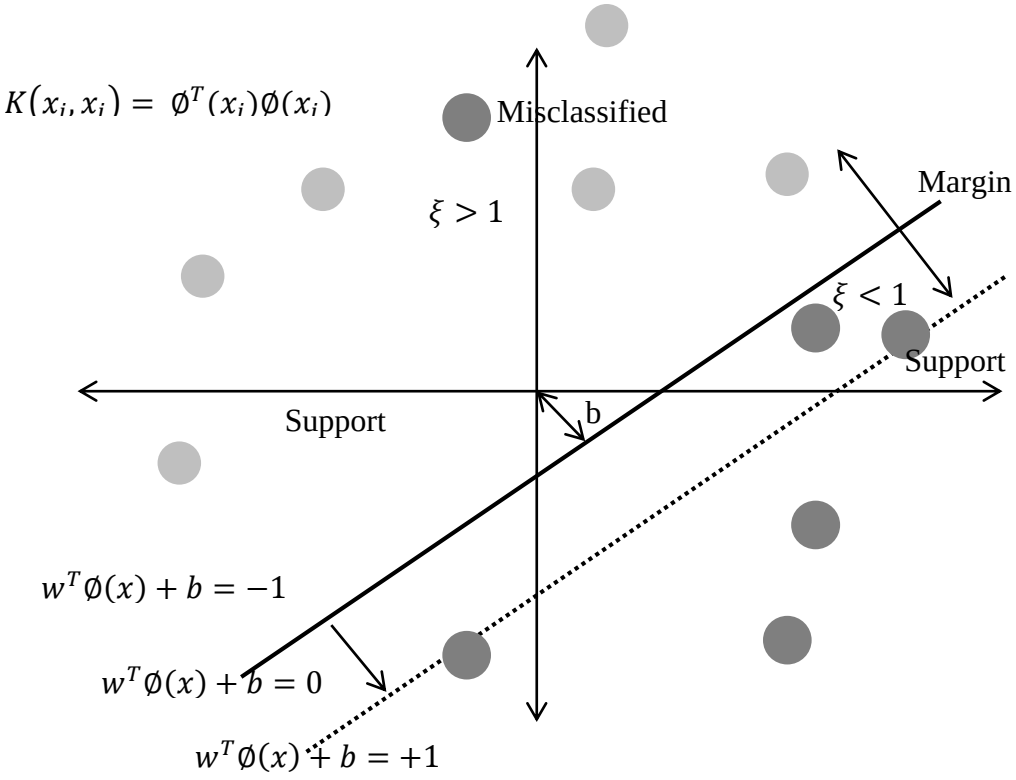

Figure 2-2 Hyperplane separates two different categories with a maximal margin separation

The estimation accuracy of the SVM thus depends on the good setting of hyper parameters C, ε and the kernel function's parameters. Parameter C controls the trade-off between degree to which deviations larger than value of ε are tolerated in the optimization and the model complexity. Parameter ε controls the width of $\varepsilon - insensitive\ zone$ to fit the training data (V. Cherkassky & Mulier, 2007).

Four basic kernel functions used frequently include the following,

- Linear kernel: $K(x_i, x_j) = {x_i}^T x_j$
- Polynomial kernel: $K(x_i, x_j) = (\gamma {x_i}^T x_j + r)^d, \gamma > 0$
- Radial basis function: $K(x_i, x_j) = \exp\left(-\gamma |x_i - x_j|^2\right), \gamma > 0$

- Sigmoid: $K(x_i, x_j) = \tanh(\gamma {x_i}^T x_j + r)$

SVM implementation in RWT uses Radial basis function (RBF) kernel as it can model a nonlinear relation between features and categorical labels. Moreover, linear kernel is a special case of RBF kernel when used with the same value of parameter C. RBF kernel also has fewer hyperparameters when a relation is modeled using it, rather than modeling with a polynomial kernel. RBF kernel is bound by constraint $0 < K(x_i, x_j) \leq 1$, while polynomial kernels are unbound and may possibly go to zero or infinity when the polynomial is of higher degree. But like all other kernels, RBF kernels suffer from limitations especially when the number of features are very large, wherein a linear kernel might be the best applicable.

Practical approaches towards hyper parameter choice of values for C and ε are summarized below (Vladimir Cherkassky & Ma, 2004),

- Prior knowledge and/or user expertise can be used to select values for C and ε (V. Cherkassky & Mulier, 2007; Schölkopf & Smola, 2002; Vapnik, 1999).
- To set values of ε proportional to noise variance was proposed by (Kwok & Tsang, 2003; Schölkopf & Smola, 2002) aligning with several other sources on SVM. Large sample sizes tend to have small ε value.
- Setting value of parameter C to range of output values (Mattera & Haykin, 1999)
- Computationally intensive cross validation can be used to select appropriate values to hyper parameters (V. Cherkassky & Mulier, 2007; Schölkopf & Smola, 2002).
- Statistical interpretation of SVM regression opens various possibilities and under this approach, value of ε can be tuned for appropriate noise density, whereas parameter C can be

estimated using cross validation(Hastie, Tibshirani, & Friedman, 2001; Schölkopf & Smola, 2002).

2.4.4.7 Evaluating Text Categorization Systems

Classifiers trained for RWT are evaluated through experiments rather than analytical calculations. Experimental evaluation of a classifier is used to measure effectiveness rather than efficiency, i.e. the right classifications made by a classifier in total are measured to realize effectiveness in a practical application. The main measures used for RWT include precision, recall and accuracy as discussed in detail below.

2.4.4.7.1 Precision and Recall

Classic Information retrieval notions of precision π and recall ρ are usually adapted to be used with text categorization systems to measure classification effectiveness.

Precision with respect to a category is defined as the probability that any random document classified to belong to the category actually belongs to it.

$\pi_i = P(\breve{\phi}(d_x, c_i) = T / \phi(d_x, c_i) = T)$,

$where\ d_x\ is\ a\ document\ , c_i\ is\ a\ category\ and\ \pi_i\ is\ precision\ of\ category\ c_i$.

Similarly, Recall with respect to a category is, probability that any random document belonging to a category is actually classified under it.

$\rho_i = P(\phi(d_x, c_i) = T / \breve{\phi}(d_x, c_i) = T)$,

$where\ d_x\ is\ a\ document\ , c_i\ is\ a\ category\ and\ \pi_i\ is\ precision\ of\ category\ c_i$.

The measures are subjective as they measure the expectation of user that system behaves appropriately when classifying an unseen document under a certain category. Contingency table organizes and makes calculating measures easy.

Table 2-2 Contingency table for Category c_i

Category c_i		Expert Judgments	
		YES	**NO**
Classifier Judgments	**YES**	TP_i	FP_i
	NO	FN_i	TN_i

Table 2-3 Global contingency table

Category set $C = \{c_1, c_2, \ldots c_{\lvert C \rvert}\}$		Expert Judgments	
		YES	**NO**
Classifier Judgments	**YES**	$TP = \sum_{i=1}^{\lvert C \rvert} TP_i$	$FP = \sum_{i=1}^{\lvert C \rvert} FP_i$
	NO	$FN = \sum_{i=1}^{\lvert C \rvert} FN_i$	$TN = \sum_{i=1}^{\lvert C \rvert} TN_i$

Here,

- FP_i $(false\ positives\ wrt\ c_i)$ is number of test documents incorrectly classified as under category c_i .
- TP_i($true\ positives\ wrt\ c_i$) is number of documents correctly classified as under category c_i.
- TN_i($true\ negatives\ wrt\ c_i$) is number of documents correctly classified as not belonging to category c_i which actually don't belong to c_i.
- FN_i($false\ negatives\ wrt\ c_i$) is number of documents incorrectly classified as not belonging to category c_i which actually belong to c_i.

The contingency table can be used to arrive at values of precision and recall of category c_i accordingly as below.

$$\widehat{\pi_i} = \frac{TP_i}{TP_i + FP_i}\ , \qquad \widehat{\rho_i} = \frac{TP_i}{TP_i + FN_i}$$

For obtaining the global precision and recall numbers, two different methods are adopted as below,

In Micro averaging, $\pi\ and\ \rho$ are calculated from summing individual decisions,

$$\widehat{\pi_\mu} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|}(TP_i + FP_i)}, \qquad \widehat{\rho_\mu} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|}(TP_i + FN_i)}$$

In Macro averaging, precision and recall are evaluated for individual categories initially and then averaged to give macro averages,

$$\widehat{\pi^M} = \frac{\sum_{i=1}^{|C|} \hat{\pi}_i}{|C|}, \qquad \widehat{\rho^M} = \frac{\sum_{i=1}^{|C|} \widehat{\rho_i}}{|C|}$$

The two quantities offer different results as their emphasis differs based on generality of categories. Their application purely depends on requirements.

2.4.4.7.2 Accuracy and Error

Measures other than precision and recall normally used among machine learning literatures include Accuracy and Error of the classifier on data. But they are insensitive to variations in number of correct decisions when compared to π and ρ (Y. Yang, 1999) due to large denominator values.

Accuracy is estimated as,

$$\hat{A} = \frac{TP + TN}{(TP + TN + FP + FN)},$$

Error is estimated as,

$$\hat{E} = \frac{FP + FN}{(TP + TN + FP + FN)},$$

CHAPTER 3. SYSTEM ARCHITECTURE

In this chapter, the design and architecture of subsystems forming the backend of RWT are discussed in detail. We consider the expert module which is responsible for analysis and interpretation, the learner diagnosis module providing inference of learner's understanding and the pedagogical module providing assistance and feedback to learner. The backend in turn has two separate independent sub systems, consisting of a training system which is used to build a classifier from data and a test system where the classifier trained previously is plugged in for use for manual input and testing.

3.1 Contributions

Research and development of RWT span over a few years and the observed accuracy of the final move and step analysis classifiers were not near the expected numbers for practical usage. Ryan (Kirk, 2011) did a great job in extracting the corpus and normalizing the data across several tables. But the training sub system and test sub system were not their best and debugging issues related to the accuracy were time consuming. Moreover, the code was ready only for unigram extraction and consumption and was not naturally extensible to accommodate trigrams from the corpus.

My initial effort as per need was to extract the trigrams, calculate odds ratios and accommodate them into model creation and testing. This was completed successfully under directions from Nick Pendar (Nick Pendar, 2011). As soon as the trigrams, were accounted for the accuracy increased from 48% to 70% and was ready for classroom usage and testing. Moreover, Ryan's code involving ensemble learners including Naive Bayes and Lexical bundles were removed as SVM classifiers showed higher accuracy independently. Trigrams

were accounted at move and step hierarchies as database tables, extracted data and Python source code for extraction, database update, model creation and model testing.

In feature extraction, previous code sections had features unprocessed or partially stubbed. This was improved to handle recurring patterns mainly URLs, domain names and HTML special characters. For RWT's model creation modules, research and experiments on multifarious classifier models that accounted for more than just binary features including SVMs that accept numerical features like odds ratios and feature frequencies were also coded in Python to test accuracy.

3.2 System Architecture of Expert Module

3.2.1 Corpus Preparation

Supervised machine learning algorithms require large amount of labeled samples for training a classifier. Classifiers are built on samples as it might be impossible or impractical to collect all different possibilities of data for a particular machine learning task. Samples reflect the desired properties and attributes we wish to learn. A corpus is a structured collection of samples of text used to represent a target language.

Corpus creation was of immense importance to RWT research and to my portion of work described. The corpus based approach and construction were mainly carried out by Dr Elena Cotos and her team of trained annotators, who manually annotated the text files using modified Swales framework (J. M. Swales, 2004; J. Swales, 1981).

A corpus of sample research articles was used as it is impossible to collect every possible variation. Research articles contain 5 major components namely, Introduction, Methods, Results, Discussion and Conclusion. Termed as “Sections” in RWT, Discussion

and Conclusion are combined as a single section, forming four major sections of research article. Current implementation focuses primarily on research article introduction section.

Research article introductions vary by structure and content across various disciplines. A stratified sample corpus of research articles from each discipline is compiled to represent a particular discipline.

Most research articles are PDF files and are not directly usable for textual analysis and processing. Hence these files are minimally preprocessed to extract the actual text content and thus converted to plain text files preserving punctuations, paragraph structures and boundaries. Any special characters, images or symbols without a text representation are discarded in this step. The main specialized corpus consists of 1,020 research articles and 1,322,089 words. The research articles span over 51 disciplines each represented through 20 texts. All articles in the corpus reported on empirical research from reputed academic journals published between 2003 and 2009.

The introduction sections from the research articles were made into separate text files to form the training data. The sub-corpus of introduction sections has 650 articles, totaling 15,460 sentences and 366,089 words.

Plain text files contain very little information to identify organizational structure of discourse. Especially, the structure of discourse varies across disciplines and doesn’t necessarily fit under a common skeleton. Hence the text files are annotated manually with different communicative functions of Moves and Steps.

Table 3-1A modified version of Swales move/step framework (J. M. Swales, 2004) used for annotation

Move	**Step**
Move 1: Establishing a territory	
	Step 1: Claiming centrality
	Step 2: Making topic generalizations
	Step 3: Reviewing previous research
Move 2: Identifying a niche	
	Step 4: Indicating a gap
	Step 5: Highlighting a problem
	Step 6: Raising general questions
	Step 7: Proposing general hypotheses
	Step 8: Presenting a justification
Move 3: Addressing the niche	
	Step 9: Introducing present research descriptively
	Step 10: Introducing present research purposefully
	Step 11: Presenting research questions
	Step 12: Presenting research hypotheses
	Step 13: Clarifying definitions
	Step 14: Summarizing methods
	Step 15: Announcing principal outcomes
	Step 16: Stating the value of the present research
	Step 17: Outlining the structure of the paper

The team used calisto workbench (Day, McHenry, Kozierok, & Riek, 2004) with a custom XML markup to annotate and tag each sentence with a step and a move they communicate. The XML markup also allows nesting several moves and steps and assigning multiple steps to multifunctional stretches of text.

3.2.2 Database design

Ryan Kirk (Kirk, 2011) made an initial effort to import annotated data into a normalized database schema. The annotated corpus contains tagged information indicating

the particular move and step a sentence belongs to. This data was loaded onto a database for faster processing, querying and management. MySQL (MySQL, 2004) was used as the database management system with a normalized schema, to hold the corpus data in an equivalent form. Importing data into the MySQL database tables was performed using custom Python scripts which parse the annotated XML files using an XML parser and import each sentence to the database along with metadata on moves and steps. XML parsers are capable of handling nested tags; hence multifunctional sentences are represented multiple times in the database with different moves and steps capturing both primary and secondary communicate functions of stretches of texts. The database is based on the following schema,

- *Sentences* – This table holds all the sentences of the corpus, along with meta data about the sentences such as discipline to which it belongs, move and step it was tagged onto, file from which it came from (to backtrack) and importantly the order of occurrence of the sentence in annotated text file (to reconstruct the actual text back from database).
- *Words* – This table contains all the unique words present in the whole corpus with a unique id as a primary key. The primary key is referred by other tables.
- *SentenceWords* – This is a foreign key association table, referring contents from *Sentences* table and *Words* table. This table is used to link sentences to its constituent words and vice versa.
- *Moves* – This table holds an index of all the moves from the CARS model using a unique identifier as primary key. This is referenced by the *Steps* table.
- *Steps* – This table holds an index of all the unique steps from the CARS model with reference to parent move they belong to. The table's primary key is referenced by *Sentences* table.

- *TrigramsGlobalfrequency* – This table holds an index of all trigram strings found throughout the corpus with frequency of occurrence globally. A custom Python script which processes each sentence from the *Sentences* table arrives at its trigram elements. These trigrams are inserted into this table with a frequency of 1, or the frequency is incremented if trigram entry already exists in the table.
- *TrigramsLocalFrequency* – This table is similar to the global frequency trigram index but holds local frequency of trigrams found in a particular step for processing. The frequency values are useful in odds ratio calculations which are used for feature reduction. Step frequency measures in turn can be used to arrive at move frequency of a particular trigram as set of steps constitute a move.
- *TrigramSentenceOccurrence* – The table holds associates T*rigramsGlobalFrequency* and *Sentences* tables. This table links trigrams to the containing sentence and vice versa.

For RWT, the advantage of moving sub-corpus into database is the ease of management and availability of SQL queries to obtain insights on data. Most of feature processing, feature reduction measures, calculations, frequency counts can be seamlessly dealt with complex nested queries instead of handwriting custom Python scripts. Also, MySQL offers various optimizations to provide high performance and concurrency which come handy when managing and analyzing corpus data of any size (Schwartz, Zaitsev, & Tkachenko, 2012) .

Various custom indexes help querying against data a lot faster and easier and have been implemented in the MySQL database using *create index* statements.

Mini Database schema diagram - pending

3.2.2.1 Database design revision

Dr Evgeny Chukharev Hudilainen (Chukharev Hudilainen, 2013) proposed the second revision to RWT's database structure aimed at optimizing the training procedure. The above schema was replaced with a single database table to hold the entire corpus of articles for different sections. The schema was denormalized and redundant but concise. Apart from changes to schema, the database engine was replaced from MySQL (MySQL, 2004) to SQLite (Hipp & KENNEDY, 2007). Once a classifier was trained from the training corpus, the training corpus can be safely disregarded during testing. This was an important design decision that chose SQLite over MySQL as MySQL was feature intensive. SQLite on the other hand stored the entire database in a plain text file, and was only used during the training procedure. The shift also ensured abstracting the entire training procedure from the test scripts exposing only the classifier model and set of features instead of the corpus itself. The revised schema also spawned new Python scripts, modifying the training subsystem entirely. The details are discussed in the upcoming section. Following is the schematic description of database fields and their data types.

Table 3-2 Revised Database Schema

Attributes	**Type**
Id	TEXT
File	TEXT
Discipline	TEXT
Section	TEXT
Move	TEXT
Step	TEXT
Is_primary	INTEGER
Primary_move	TEXT
Primary_step	TEXT
Attributes	**Type**
Content	TEXT
Sentence_id	INTEGER

3.2.3 Training sub-system

The training sub system is an important part of an expert module. Training a classifier through an inductive process after analyzing data and applying algorithms to actually learn

the characteristics pertaining to a category are the main course of actions implemented in this sub system.

In RWT, the training sub system is in turn implemented integrating independent building blocks each achieving modular tasks. The integration is brought about by a Python script which communicates with the building blocks moving data and results inside out.

Following are the building blocks which together constitute the training process for RWT.

Figure 3-1 RWT's Training Sub-System

3.2.3.1 Feature Processing

As in Chapter 2, identifying important features that capture characteristics of data we are trying to learn is necessary for learning algorithms to perform. Feature extraction and analysis is a repetitive process where each feature set is tested for performance and better performing (in terms of accuracy of the classifier) sets replace existing ones. Similarly, various feature reduction measures with experimental parametric values, may be applied each resulting in completely different set of features.

Prior research on IADE (Pendar & Cotos, 2008) before RWT noted that bigrams had a negative effect on text categorization towards discourse analysis. In RWT, unigrams and trigrams are used as the main features to build the classifier. Database tables hold unigram

and trigram strings from the whole corpus along with frequency figures measured along various hierarchy levels including whole corpus level, move level and the step level.

Before applying feature reduction procedures, a threshold frequency was considered to cut down unimportant features to further reduce the feature space. Unigrams and trigrams occurring fewer than “5” times are discarded to proceed with. This could further be increased to a higher cut off value, if text categorization task using fewer features yields similar performance. Reducing the feature space augments the performance of all subsequent processes that follow, including training file creation, actual classifier training and testing classifier for performance.

Prior implementations by Ryan kirk had numerous Python scripts to achieve feature weighing and processing. Currently in RWT, SQL statements are used to weigh (Odds ratios) features instead of custom scripts as they are easier to debug and faster to change/export. The unigrams and trigrams which exhibit high odds ratios are considered for further processing. Having the features and their frequency values in the database serves an advantage as most of the post processing and weighing can be done using simple nested SQL queries rather than custom Python scripts. Finally the features identified are exported as plain CSV files to aid training file creation and training of the classifier.

Following is an example SQL Query to identify unigrams globally occurring “5” times or more, grouped along the particular step in which they occur. The unigrams are listed in descending order weighed according to odds ratios of a unigram occurring in a particular step. Similar query can be applied to identify trigrams.

```
select t.* from
(select w.word,sent.stepid,w.globalFreq,count(*) as
LocalFreq,count(*)/(w.globalfreq-count(*)) as Odds from
sentencewords sw,
```

```
words w,
sentences sent
where sw.wordID=w.wordID
and sent.sentenceID=sw.sentenceID
and w.GLOBAlfreq >= 5s
group by w.word,sent.stepid
order by w.word ,sent.stepid ) t
order by t.odds desc
```

3.2.3.2 Training file creation

Following feature extraction, training file creation is the next step in line. In Chapter 2, compact representation of corpus and its importance are specified as it affects the inductive process of training. The training file format in turn depends on the actual algorithm implementation tool being used for training a classifier. RWT uses the bag of words model with the specification of actual class/category followed by tab/comma/space separated feature values. Depending on the classification algorithm and implementation, the individual elements in a training file can be string labels or numeric values indicating a measure/weight of a feature.

A Python script iterating over sentences in the database is used to generate feature representations for each sentence. Thus a single line of the training file, referring to a sentence in the database has category/class value as the initial element, followed by feature values. In case of RWT, the features values are binary though numerical values are supported. Feature representations can be either dense or sparse. Dense representations require all features to hold weight or value even if particular feature is absent or has no value (zero weight is used). Sparse representations on the other hand require specifying only those features with a value. All features unspecified are automatically equated to zero. Sparse representations offer small size representations and are relatively less time consuming to generate.

An example of a feature vector represented in dense and sparse formats are given below, with first cell indicating category and 4 features to follow. Category "1" is separated with a tab whereas features are separated with a space.

Dense representation – 1 1 0 2 0 0 0 0 0 0 1 0 0 0 1 0 2 0 5 0 0 0 0 0 0 0 0 0 0

Sparse representation – 1 1:1 3:2 10:1 14:1 16:2 17:5

For RWT, LIBSVM (Chang & Lin, 2011) used to implement support vector machines poses limitations on training file representation. Hence training file in RWT uses a sparse representation with binary/numerical values for features and numerical values for categories.

3.2.3.3 LIBSVM – Support Vector Machine Library

LIBSVM (Chang & Lin, 2011), a support vector machine implementation library with bindings to Python was used to train the classifier. LIBSVM supports multi-category classification. Highly configurable in application, the library's API interface allows a developer to set parameters, choose kernels, train a classifier, apply cross validations, save models trained and load models for future use. The library also allows training based on probability and returns information about the model trained including mean squared error, optimization history, accuracy and estimates of probability of classification to particular categories for each sample. A helper Python script is provided by the library for automating grid searching and parameter selection to find the best performing values by covering the parameter space extensively.

3.2.3.4 Revised training sub-system

Post revised database schema (mentioned in 3.1.2.1) adopted for RWT, resulted in streamlined training subsystem which combined the feature processing and training file

creation into a single Python script instead of two different modules. The idea was proposed by Dr Evgeny Chukharev Hudilainen (Chukharev Hudilainen, 2013) to optimize the training sub-system to contain fewer highly coupled, simple components thus speeding the entire training procedure. The script consumed the entire corpus from SQLite database and produced odds ratio weighed unigram and trigram features as text files. It also transformed the entire corpus into training files on the fly from the memory. This effort drastically reduced the time taken for feature extraction and training file creation and optimized the training sub-system appropriately. The individual steps implemented in the python script are covered in the next chapter.

3.2.4 Test sub-system

Similar to the training sub system, the functionalities to test a constructed model were based on independent modules, integrated using Python. Main inputs passed on from the training sub system include feature sets, trained model file saved to disk by LIBSVM along with chosen parametric values and set of routines to load the above data into memory as they are same.

3.2.4.1 Tester Script

A trained SVM model can be loaded on demand to analyze against a test corpus after representing them as feature vectors. In RWT, the test corpus consists of completely new data not available during training, but collected and processed similar to the training data. Performance of a model being tested is calculated by comparing model's automated classification with an expert's classification of the entire test corpus. With the feature set, an input sentence is converted to feature vector and passed as input to the model for

classification. LIBSVM predicts a particular category by minimizing the error of classification through structural risk minimization. The whole functionality was achieved in RWT through custom Python scripts.

The script is generic to load any input model and feature vectors to the memory. Once loaded, it allows a test to be performed on a given set of sample sentences. The script also produces verbose output on accuracy achieved on the test set using this model and other metrics including error and probability estimates of classification.

Above mentioned functionality for RWT is achieved through console based programs and command line activity.

3.2.4.2 User Interface Integration

User interface integration was the next step in the development of RWT. Most of RWT's user interface design, development and research were handled by separate UI team and the whole process is documented in a previous thesis by Nandhini (Cotos, Gilbert, & Link, 2012; Ramaswamy & Gilbert, 2012; Ramaswamy, 2012). Previous source code from IADE (Pendar & Cotos, 2008) facilitating communication between analysis engine and the user interface was reused for RWT as most functionality is similar and the analysis engine is agnostic by design from changes to the user interface. The source necessarily implemented a persistent daemon in Python to listen to TCP sockets for incoming messages and respond appropriately.

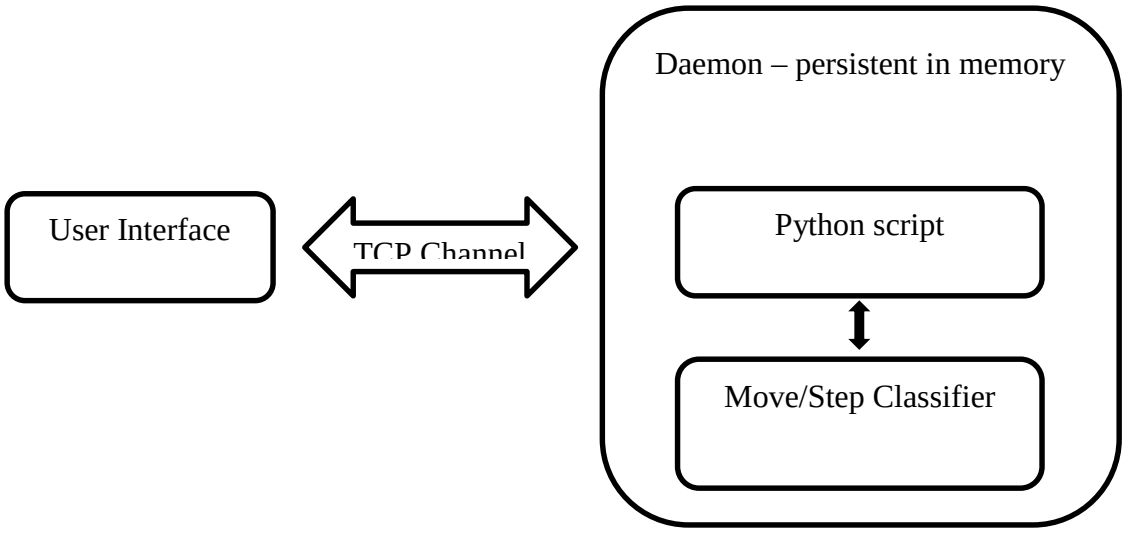

Figure 3-2 User interface connected to Daemon through Sockets

User interface elements were developed independently using PHP. The user interface allowed learners to submit their writing for analysis and get annotated results back from RWT. A daemon process encapsulating the analyzer listens to a TCP connection from the user interface and persists in memory unless stopped or killed. This daemon is the port of contact between the user interface and the tester script for passing bytes back and forth. TCP sockets abstract the complexity in handling multiple simultaneous connections and hence are best to handle multiple user interface analysis requests with minimum implementation. A request from the user interface consists of paragraphs of text to be analyzed along with metadata on the request. TCP sockets carry the request to the expert module on the other end. Once the textual data is analyzed using feature sets and the classification model, the result is passed back to the user interface as an XML. XML being a standard allows returning structured results which may be used for further processing down the line.

The following is a sample XML with nodes and attributes, listing result of analysis.

```
<? XML version="1.0" encoding="ISO-8859-1"?>
<RWTtext>
	<labelText>
		<labelText UID="deepan18" SessionID="95afdddec15fdb1b795617b7d01ff485"
AnalyzeSessionID="2012_11_29_22_42_05_746_50b8391db66ec" section="INTR" discipline="AGBE"
ID="1" text="this is a simple test." step="1" agreementSteps="100.0" feedbackSteps="You may
be providing general background about the topic of investigation here." move="1"
agreementMoves="100.0" />
	</labelText>
	<scoreMove>
		<scoreMove UID="deepan18" SessionID="95afdddec15fdb1b795617b7d01ff485"
AnalyzeSessionID="2012_11_29_22_42_05_746_50b8391db66ec" section="INTR" discipline="AGBE"
ID="1" move="1" numSentMove="2" target_numSentMove_min="1" target_numSentMove_low="1"
target_numSentMove_norm="1" target_numSentMove_hi="2" target_numSentMove_max="2"
percentMove="0.666666666667" target_percentMove_min="0.317672483623"
target_percentMove_lo="0.476508725435" target_percentMove_norm="0.635344967246"
target_percentMove_hi="0.794181209058" target_percentMove_max="1.11185369268"
feedbackMove="Too much focus on establishing the territory compared to agricultural and bio-
systems engineering papers.Needs more work."  />
	</scoreMove>
	<scoreStep>
		<scoreStep UID="deepan18" SessionID="95afdddec15fdb1b795617b7d01ff485"
AnalyzeSessionID="2012_11_29_22_42_05_746_50b8391db66ec" section="INTR" discipline="AGBE"
ID="1" step="1" move="1" numSentStep="2" target_numSentStep_min="0"
target_numSentStep_low="0" target_numSentStep_norm="0" target_numSentStep_hi="1"
target_numSentStep_max="1" percentStep="0.666666666667"
target_percentStep_min="0.120297499001" target_percentStep_lo="0.180446248502"
target_percentStep_norm="0.240594998002" target_percentStep_hi="0.300743747503"
target_percentStep_max="0.421041246504" feedbackStep="Good work on generalization.Very
similar to agricultural and bio-systems engineering papers."  />
	</scoreStep>
</RWTtext>
```

3.3 System Architecture of Learner diagnosis module

The learner diagnosis module is necessary to assess and estimate the current understanding of the learner regarding the subject to generate valuable feedback. Based on the analysis of user's text learner's model and knowledge structure are to be inferred (Richardson, 1988). Using corpus data in MySQL database it is possible to arrive at statistical measures highlighting common trends of introductory texts among various disciplines.

SQL queries are used to calculate the proportion of sentences and words present in each step, move under various disciplines of the corpus. Analytical information thus calculated, from the corpus can be considered to reflect the properties of the target language from which they are derived. Learner diagnosis can be performed using this information by comparing word counts & sentence counts of learner's text to the information we have. When the learner's text has properties similar to corpus articles from the same discipline at move and step levels, similarity of structure and hence the discourse can be inferred.

Similarity measures can also be used to measure the current performance and the actual goal performance to track gaps in learner's text. The information can be made more accessible through graphical information displays.

Whole corpus residing in the MySQL database makes calculating proportions and other frequency values easier through queries instead of custom Python scripts and iterating data manually.

A Sample SQL query to calculate total words in the corpus grouped by steps,

```
SELECT t.stepid,COUNT(t.wc) FROM
(SELECT s.sentenceid,COUNT(*) AS wc,s.stepid FROM sentences s,
sentencewords sw
WHERE s.sentenceid=sw.sentenceid
```

```
GROUP BY s.sentenceid,s.stepid) t
GROUP BY t.stepid
```

Similar queries can be used to get statistics about data at move and word levels from the corpus. In case of calculating word counts for learner's text, a custom Python script is used after processing by the expert module. Once analysis is completed, each sentence of user's text is tagged with a step and move by the classifier. This information is used to calculate frequency counts of words of learner's text at step and move levels.

3.4 System Architecture of Pedagogical Module

Pedagogical module aims to provide help towards improvement for the learner apart from other functionalities including sequencing and structuring of displayed content. The functionality of this module spans partly over the user interface and the expert module forming the backend. Learner diagnosis provides valuable insight towards feedback generation.

Custom Python code is used to generate two different kinds of feedback to direct learner and provide assistance. One type of feedback considers learner's text on the whole. Based on comparison of structure between learner's writing and the corpus data in a particular discipline, feedback is generated by the module to inform how similar learner's text complies with the corpus data. It also informs the learner, to work on his/her article to match the discipline's structure. Another type of feedback considers learner's text as set of granular communicate steps to achieve the purpose of a specific move. Based on probability estimates from the expert module, Python code generates feedback to inform if a particular functional step was clear, vague or irrelevant. Thus this feedback is related to each sentence

of the learner's text. All feedback text strings for display are fetched from the MySQL database table and displayed accordingly.

CHAPTER 4. METHODOLOGY

This chapter contains implementation details and the main procedure towards training a classifier from corpus data. In terms of individual contributions, part of the implementation including code for loading trained models for testing, interacting with the user interface as a daemon process were reused with changes from Ryan's work (Kirk, 2011). My contributions include code, rebuilding the entire training procedure from scratch including feature extraction, weighing, training file generation, and model creation. I was also responsible for integrating trigrams into the training scenario which was previously unavailable.

4.1 Introduction

Automated analysis of learner's research article introductions has four basic steps: 1) Feature selection of unigrams and trigrams, 2) Sentence representation through features, 3) Classifier training and model creation, 4) Testing and evaluation of model. Initial step involves applying feature weighing schemes and reduction techniques to identify best features from the corpus of training data. Once features are identified, the whole corpus is represented in terms of features and their corresponding weights. This is used to train a classifier, a support vector machine in our case to identify various moves and steps. The classifier hence constructed is evaluated using test data to estimate efficiency on unseen data. The process is repeated with various parameters, until the accuracy, precision and recall of the classifier modeled is best suited for practical purposes and applications.

The CARS model for discourse analysis from chapter 2 and its adaptation in chapter 3, are based on structurally splitting research article introductions into different moves and into finer steps based on their innate communicative functions. Each sentence of an article is

assigned to a move and a step. Hence, sentences can be considered as documents for training and the moves and steps they signify imply the categories for classification.

Formally each sentence in the corpus $C = \{c_1, c_2, \ldots., c_n\}$, is represented as a set of features as $c_i =< f_1, f_2, \ldots f_k >$ where each f_j measures feature j's weight in sentence i. Let M = $\{m_1, m_2, m_3\}$ denote the set of moves and S = $\{s_1, s_2, s_3, .., s_{17}\}$ denote the set of steps in the CARS model. We are required to learn the following mappings; $\{F: C \rightarrow M\}$ and $\{G: C, M \rightarrow S\}$.Though stretches of text could signify more than one step or move, the simplest case where each sentence signifies single move and single step is considered. Sentence representation based on the move feature set is initially used to classify a sentence to a particular move. Once a move is identified, sentence representation of the same sentence based on the step feature set, with move as an additional input is used to classify a sentence to a particular step. Thus classifying a sentence to a move and a step is completed through a two-step process with move passed on as an input to aid the step classification.

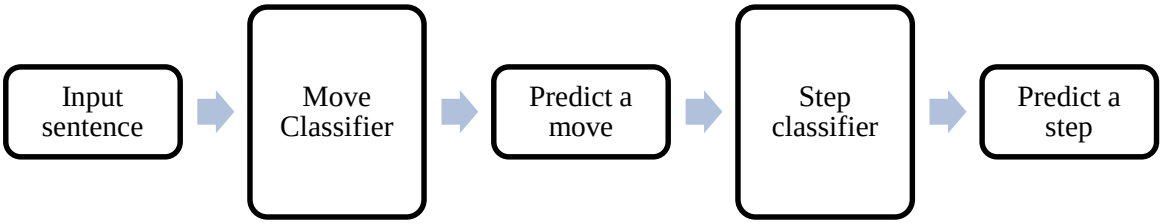

Figure 4-1 Classifying a sentence

4.2 Training a Classifier

As stated in Chapter 3, the main features used to learn attributes of various moves and steps are unigrams and trigrams (i.e. single words and set of three word sequences) from the annotated corpus. Moves are totally 3 in number whereas steps number to 17 with the

adapted CARS model, hence 2 different feature sets of unigrams and trigrams signify the moves and steps separately.

The different feature sets are,

- Umovefeatures – Unigram features for moves,
- Ustepfeatures – Unigram features for steps,
- Tmovefeatures – Trigram features for moves,
- Tstepfeatures – Trigram features for steps.

Data was preprocessed before arriving at the final feature set of unigrams and trigrams as follows,

- Each sentence is tokenized into its constituent words and stop words are removed as they have low informational value. The order of words in the sentence is unchanged.
- The individual words are stemmed using NLTK's (Bird, 2006) implementation of Porter Stemming algorithm (Porter, 2009).
- After analyzing the corpus, frequently occurring patterns are stubbed with placeholder tokens to increase consistency of resulting features. For e.g. in our corpus, all year occurrences were replaced with *__year__* and all number occurrences were replaced with *__number__*. This procedure confirms that all references to years and numbers result in the same feature. Hence the following trigram features *et al 2003, et al 1997* map to the same trigram feature *et al __year__*.
- Other patterns that result in tokens include web page references replaced with *__url__*, domain names replaced with *__domain__* and HTML special characters like &quot and &amp replaced with *__html__*.

- While replacing patterns with tokens, a particular order towards pattern search and replace is followed to reduce loss of information. For e.g. years are replaced before numbers as both patterns search for a sequence of digits and an approach vice-versa might result in no year patterns to replace.
- After replacing patterns with tokens, individual word units are stored in a unigram list whereas consequently occurring word triplets are stored in a trigram list.
- Considering frequency of the features, insignificant features with occurrence less than 5 are removed to avoid over-fitting. This also helps reducing the feature space.

After arriving at the basic set of features, feature reduction procedures are applied to further reduce the feature space and to identify the crucial features that help in the actual learning process. Odds ratio is used as a measure to reduce feature space.

As discussed in chapter 3, Odds ratio describes correlation between two classes or categories and a high value implies odds of a particular event is more probable than the other event under consideration.

Odds ratio of a term t_i (unigram or a trigram) occurring in move $m_j$is given as,

$$OR(t_i, m_j) = \frac{p(t_i|m_j).\left(1 - p(t_i|\bar{m}_j)\right)}{(1 - p(t_i|m_j)).p(t_i|\bar{m}_j)}$$

Where $p(t_i|m_j)$– Probability that term t_i occurs in move, given move $m_j$has occurred,

$p(t_i|\bar{m}_j)$– Probability that term t_i does not occur in move, given move $m_j$has occurred.

Similarly, the odds ratio of a term t_i (unigram or a trigram) occurring in step $s_j$is given as

$$OR(t_i, s_j) = \frac{p(t_i|s_j).\left(1 - p(t_i|\bar{s}_j)\right)}{(1 - p(t_i|s_j)).p(t_i|\bar{s}_j)}$$

Where $p(t_i|s_j)$– Probability that term t_i occurs in step, given step $s_j$has occurred,

$p(t_i|\bar{s}_j)$– Probability that term t_i does not occur in step, given step $s_j$has occurred.

The conditional probabilities are calculated using maximum likelihood estimates,

$$p(t_i|m_j) = \frac{count(terms\ t_i\ in\ move\ m_j)}{\sum_{k=1}^{N} count(terms\ t_k\ in\ move\ m_j)}$$

$$p(t_i|s_j) = \frac{count(terms\ t_i\ in\ step\ s_j)}{\sum_{k=1}^{N} count(terms\ t_k\ in\ step\ s_j)}$$

Here N is the total number of terms from the corpus.

For the move feature sets, the odds ratio measures are calculated for each unigram and trigram against all the 3 different moves. Thus each unigram and trigram has 3 different odds ratio measures. The final odds ratio measure for a unigram or a trigram is set to be the maximum of the 3 measures. Once we have a list of unigrams and trigrams and their corresponding maximum odds ratio measures, the list is sorted in descending order to get the final feature list. The length of feature space is the total number of individual unigrams and trigrams after sorting. The feature space could be trimmed further by considering a threshold odds ratio measure and discarding all features of lower value. A similar procedure is repeated for step feature sets, where odds ratios for each unigram and trigram are calculated against all the 17 different steps. The maximum odds ratio for each unigram and trigram is used to arrive at the final feature set, sorted in descending order. The length of the feature space could be trimmed similarly based on a threshold value.

Once the feature set of moves and steps are in place, training file generation is the next step towards building a classifier. As stated in chapter 3, a separate training file is

generated for the whole corpus using both the move feature set and the step feature set to train individual move and step classifiers.

Move classifier's training file representation uses binary values to specify presence/absence of a particular feature for a sentence, for e.g. a sentence c_i from corpus C is represented as $c_i =< mf_1: 1, mf_2: 0, mf_3: 1..mf_n: 0 >$ where $mf_1, mf_2..mf_n$ are move features. Step representation uses similar binary values to specify features, with an additional array of features representing the actual move to which sentence belongs to, for e.g. sentence c_i belonging to move 1 may be represented as $c_i =<$ $m_1: 1, m_2: 0, m_3: 0, sf_1: 1, sf_2: 0, sf_3: 1..sf_l: 0 >$. Manually annotated corpus has each sentence tagged with a particular move and step, which makes generating these representations straightforward. Sentences from the corpus are preprocessed using the same procedure used for feature selection specified above. Features from the feature set are marked with a 1, if found in the preprocessed sentence and with a 0 otherwise, to arrive at the final training file representation.

Since moves and steps have descriptive string names for representation in the corpus, the primary key in *Moves* and *Steps* database tables are used as numerical identifiers instead of string identifiers as labels in the training files. Each sentence in the corpus is transformed into a numerical label (move or step) followed by a set of index value pairs representing presence/absence of features. The ordinal position of occurrence of a feature in the ordered feature set is used as an index to indicate the particular feature followed by a colon and binary digit indicating presence of the feature.

An example of transforming a sentence to a feature representation is as below,

Sentence:

Mozilla web browser project is in active development since 1998 and has an excellent support community. – tagged as move 1.

Move Features:

Unigrams – project - 1, research -2, support -3, community -4, academic -5, proposal -6

Trigrams – development since __year__ -7, et al __year__ -8.

Generated feature representation using Move feature set:

1 1:1,2:0,3:1,4:1,5:0,6:0,7:1,8:0

LIBSVM's API calls allow loading a training file and training a SVM classifier based on the above generated format. LIBSVM is implemented in C and has bindings required to access the functionality from Python scripts. A typical Python script would load the training file into memory using *svm_read_problem* function into separate variables, one containing a list of labels (move or step) and other containing a list of index value pairs corresponding to these labels. Following this, a call to *svm_train* is used to train a classifier using the above loaded training data. The function is also used to specify type of kernel to be used for the task, values of hyper parameters, if classifier is to be trained using probability measures and if cross validation is to be applied. After invocation, the function returns the optimized model for further use. Verbose messages are also displayed during training with information on optimization of parameters and the final accuracy obtained on cross validation. The model trained and existing in memory can be saved onto a disk file and loaded later using *svm_save_model* and *svm_load_model* functions. The training procedure is complete when *svm_train* returns a model with accuracy and can be validated against test data.

After training a model, the only data required for testing and use of the classifier against user's writing are the feature sets and the trained model itself. The whole of the corpus can be disregarded once the accuracy of the classifier and the evaluation measures are suitable for practical usage and application. The model essentially holds the information in terms of support vectors whose dimensions are defined by odds ratios of the feature sets themselves.

Thus two separate classifiers are trained for classifying a sentence to a particular move and step. The move feature set representation of the training corpus is used to train a classifier for classifying sentences to a particular move, whereas the step feature set representation of the corpus is used to train a classifier for classifying sentences to a particular step. Cascading the classifiers offers better accuracy as a move predicted by a classifier is passed as an input to a step classifier aiding the prediction task.

4.2.2.1 Revised training procedure

As in the previous chapter, to optimize the entire training procedure in RWT and decreasing total time taken to process features and create training files, SQLite was adopted instead of MySQL. This also eventually replaced the existing Python scripts to a single streamlined script achieving feature processing and training file creation in a single script. The Python source code of the revised training with feature extraction, training and testing was my contribution completely.

The python script implemented the following steps,

- Entire annotated corpus in SQLite database is loaded onto memory using sqlite3 library of python, one sentence at a time.

- Unigrams and trigrams were extracted from the sentence. They were tokenized, stemmed and recurring patterns were stubbed at this stage using a python routine. Following this, their counts were accounted at particular section, move and step level hierarchies.
- Odds ratio measures were calculated for each unigram and trigram using the above counts.
- The entire unigram and trigram lists were ordered in decreasing odds ratio weight and a cut off threshold of "15000" was used to trim the feature list.
- The individual lists were dumped as plain text files, trigrams separated by tabs. This resulted in 4 different feature lists (unigram/trigram for move and step).
- The feature lists in memory were subsequently utilized to generate training files similar to the prior implementation.

The above mentioned script though runs for hours for a decent corpus containing 100,000 sentences was faster than the MySQL backed implementations.

4.2 Testing and practical application

RWT's Test scripts share the data preprocessing and sentence representation modules from the training section of a classifier. This ensures that any changes to preprocessing and transformation steps during the training process of a classifier are reflected exactly to test data and thus maintaining consistency. Most of the Python source code for this section was derived from Ryan Kirk's original work (Kirk, 2011). I was able to contribute portions of source code related to trigram integration and fix issues related to buggy feedback.

Applying the trained classifier on a test corpus and eventually on real user's introduction texts from research articles is the next step to quantify accuracy, precision and

recall for practical usage. In case of training corpus, sentences are annotated and sentence boundaries are implicitly stated by the actual user annotating them while tagging. But in case of input user articles, sentence boundaries are to be detected as users paste large chunk of research article introductions into an input textbox without explicit specification of sentence boundaries. Moreover manually stating sentence boundaries for a large body of text is user demanding task and hence would affect usability of the tool. Python's Natural language toolkit implements Punkt sentence tokenizing functionality (Garrette & Klein, 2009; Kiss & Strunk, 2006) as *sent_tokenize*, which takes a body of text as input and returns an array of sentences split at boundary for further processing. Sentence tokenizing precedes data processing steps which act on individual sentences.

Main steps involved in transforming a sentence to feature representation is similar to the procedure followed for the training corpus,

- Tokenization of individual sentences to words with order of occurrence of words unchanged.
- Removal of stop words with low informational value.
- Stemming words using porter stemmer (Porter, 2009) implementation from NLTK.
- Replacing common patterns like URLs, year references and numbers with placeholder tokens.

The resulting words represent unigrams of the sentence. The trigrams are generated using unigrams when original order of their occurrence is unchanged. The feature representation of the sentence is the next step when the unigrams and trigrams of the sentences are available. Given the set of move and step features identified during training,

feature representation marks a particular feature with a binary digit 1 if it is found in the set of unigrams and trigrams and with a binary digit 0 otherwise. Thus the input sentence is represented as binary string of features similar to the sentence representation used in training. Each user sentence results in two different feature representations based on move and step features separately.

LIBSVM's API call *svm_predict* is used to decide a move or step from the above generated binary feature representations. The function takes 3 inputs namely, a list containing binary string feature representations, a list containing user predicted classifications corresponding to the binary string representations and the model to be used towards prediction. The user predicted classifications are used to calculate accuracy of the classifier by comparing them alongside model predicted classifications. Accuracy is calculated as the percentage of results predicted by the classifier that match user predicted classifications. Inputs of the function being list of values allow multiple sentence representations to be classified using a single call to *svm_predict*. The function call returns a list of predicted labels/classes, accuracy of the prediction and probability estimates of certainty of the predictions returned.

Two different calls to *svm_predict* are required for each sentence. Initial call is used to classify a move feature set based sentence representation to a particular move. The predicted move is passed as an additional feature input to the step feature based sentence representation and a second call to the function accounts for the step classification of a particular sentence. A dedicated Python script sharing module procedures for sentence transformation from the training sources is used to implement the above functionality. The

script takes a body of text as input and returns an array of sentences and their corresponding move/step classifications.

The above procedures are kept modular and generic to be used externally by other programs and solutions. Once this infrastructure for transforming sentences and passing it through the classifier to predict a move and step are in place, the functionality can be extended to apply on test corpus and user submitted research article introductions. The moves and steps thus predicted by the classifier can be exported back for future comparison with expert predictions for studying effectiveness or can be displayed to user to reveal the discourse structure of article text effectively.

A helper Python script is used to iterate sentences from test corpus and feed them through the above Python script, to gather the resulting move and step predictions and calculate the confusion matrix depicting the prediction accuracy, precision and recall. The modularity of the test scripts also allow data from a user interface to be analyzed and results of analysis sent back to the user interface appropriately. To facilitate universal consumption of analysis results across platforms, operating systems and networks, the functionality is delivered through a web service serving the response of an analysis as an XML, through sockets. TCP/IP socket implementations are universally available across platforms and structured XML being plain text can be sent through sockets to any client requesting discourse analysis of arbitrary text. Sockets also handle multiple concurrent requests from a number of clients requesting the service, which makes serving multiple clients an innate functionality. The input to the web service consists of body of text and discipline to which it belongs.

Apart from move/step predictions, the XML results also carry learner diagnosis calculations and target percentages recommending direction of change for the user/learner to work on the text. As stated in Chapter 3, Corpus statistics consisting of number of words and sentences in each move and step level grouped by discipline are calculated using custom SQL queries and stored in the database. The database schema and normalization specified in Chapter 3, makes these calculations query only instead of writing custom Python scripts to calculate them. For a particular usage scenario consisting of user submitting his/her research article introduction, the proportion of words and sentences in each predicted move and step are compared with the corpus statistics using Python to generate appropriate feedback. Based on the proportion, various feedback messages indicating current text's discourse content present in each move or step and direction of focus indicating too much presence or a lack of particular communicative function is implied. Also the proportion values are sent in the XML aiding graphical display or manipulation of the content presence and focus direction.

CHAPTER 5. RESULTS

In this chapter, I present the results of training and performance characteristics of both move and step classifiers, along with corpus statistics. Some of them were already published in related conferences (Nick Pendar, Cotos, & Babu, 2012).

5.1 Corpus Dimensions

Data for RWT consists of manually annotated corpus containing 15460 sentences was split into separate training and test corpuses. Each step and move from the swales schema (J. Swales, 1981) were given unique numbers to represent them, steps totaling to 17 and moves totaling to 3 from the introductions section. Following is the distribution of sentences among various steps and moves in the training and test corpuses specifying volume and dimensions.

Table 5-1 Distribution of sentences across moves

Move	Total Counts	Training Counts	Testing Counts
1	9272	6039	3233
2	2535	1609	926
3	3653	2352	1301

Table 5-2 Distribution of sentences across steps

Step	Total Counts	Training Counts	Testing Counts
1	629	403	226
2	3370	2309	1061
3	5273	3327	1946
4	454	263	191
5	1127	725	402
6	72	54	18
7	276	162	114
8	606	405	201
9	995	636	359
10	348	211	137
11	101	59	42
12	166	113	53
13	51	40	11
14	448	290	158
15	517	352	165

16	400	269	131
17	627	382	245
	15460	10000	5460

5.2 Test data sets

Model construction involves feature sets of unigrams and trigrams, for both moves and steps derived from the corpus. In RWT, the move classifier was trained using 10 different feature sets and the step classifier was trained using 13 different feature sets as in the tables below. The number of unigrams and trigrams in each feature set is increasing progressively. Each move/step classifier trained with a particular feature set was tested on a test corpus to measure performance evaluation metrics of precision, accuracy and recall for comparison. The total number of features for move classifier had 5825 unigrams and 11630 trigrams. Similarly the total number of features for step classifier had 27689 unigrams and 27160 trigrams.

Following is a list of feature set specification accounting on the varying number of unigrams and trigrams used to build the classifier.

Table 5-3 Feature Set - Move classification

N-Gram Feature Set	
# Unigrams	# Trigrams
1000	0
2000	0
3000	0
0	1000
0	2000
0	3000
1000	1000
2000	2000
3000	3000
5825	11630

Table 5-4 Feature Set - Step Classification

N-Gram Feature Set	
# Unigrams	# Trigrams
1000	0
5000	0
6334	0
10000	0
26789	0
0	1000
0	5000
0	5986
0	10000
1000	1000
5000	5000
10000	10000
27689	27160

For RWT model creation, LIBSVM was used with the following parameters for training the move and step classifiers. The default standard regularized support vector classification algorithm C-SVC was used to train the classifiers. Radial basis function kernel was chosen based on its previous application in IADE. The cost parameter C was set to be 1511after multiple runs, as it offered better measures of accuracy with the training set.

5.3 Performance Metrics

In RWT, towards evaluating the performance of the move and step classifier on the test corpus, the precision, recall and accuracy of each model was taken into consideration. The following graphs and tables capture various measures for each individual feature sets

used, facilitating comparison and analyzing behavior across feature sets.

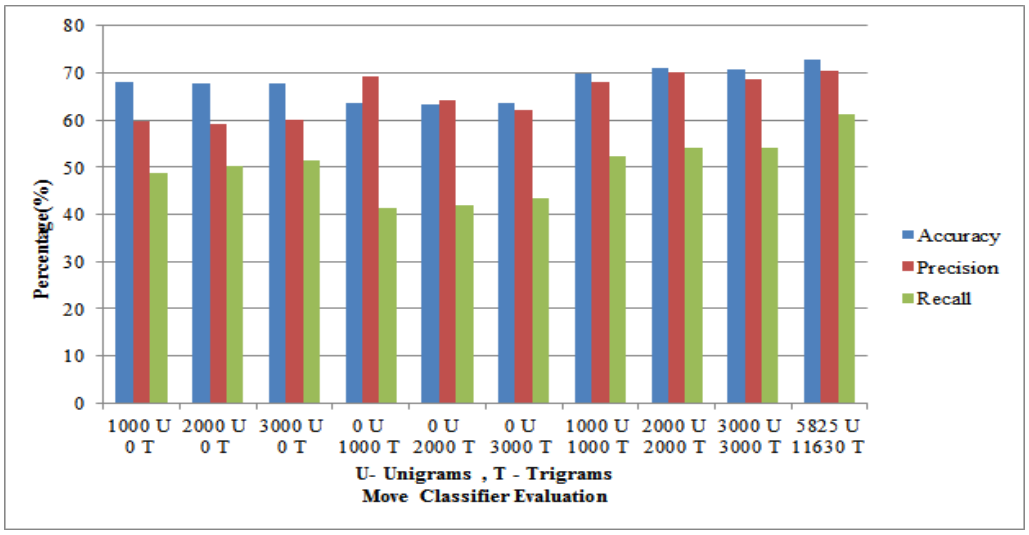

Figure 5-1 Performance of Move Classification

Table 5-5 Performance of Move Classification

Legend **U – Unigrams** **T - Trigrams**	**Accuracy**	**Macro Average (Precision)**	**Macro Average (Recall)**
1000 U 0 T	68.11355311	59.70482045	48.63274879
2000 U 0 T	67.72893773	59.02765696	50.15838347
3000 U 0 T	67.76556777	60.11068765	51.51218936
0 U 1000 T	63.44322344	69.14748429	41.31325141
0 U 2000 T	63.13186813	64.27648659	41.99449874
0 U 3000 T	63.51648352	62.21151844	43.39843442
1000 U 1000 T	69.85347985	67.93256822	52.32162123
2000 U 2000 T	71.0989011	70.19000299	54.1019552
3000 U 3000 T	70.84249084	68.57402431	54.04897081
5825 U 11630 T	72.65567766	70.28127517	61.15442846

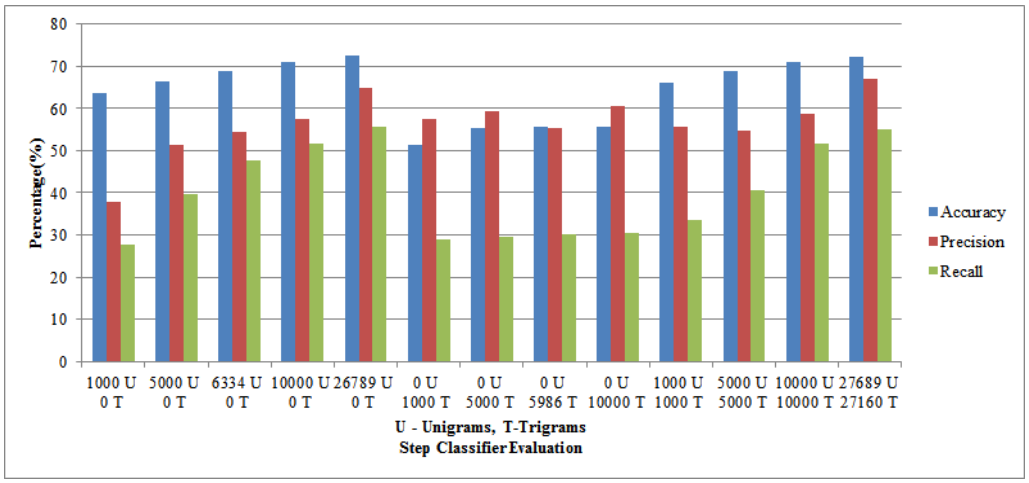

Figure 5-2 Performance of Step Classification

Table 5-6 Performance of Step Classification

Legend U – Unigrams T - Trigrams	Accuracy	Macro Averaging (Precision)	Macro Averaging (Recall)
1000 U 0 T	63.6996337	37.72693989	27.63498206
5000 U 0 T	66.52014652	51.25537231	39.64759751
6334 U 0 T	68.99267399	54.54498401	47.66632687
10000 U 0 T	71.08058608	57.41942837	51.79093818
26789 U 0 T	72.67399267	64.84375459	55.75931553
0 U 1000 T	51.22710623	57.60803916	28.9353557
0 U 5000 T	55.21978022	59.26715161	29.45616674
0 U 5986 T	55.65934066	55.21463011	30.24360779
0 U 10000 T	55.73260073	60.64500262	30.41486066
1000 U 1000 T	66.00732601	55.59311107	33.41381714
5000 U 5000 T	68.75457875	54.80574451	40.5975473
10000 U 10000 T	70.86080586	58.77756727	51.76692523
27689 U 27160 T	72.06959707	66.92326901	54.90177966

As stated in chapter 3, global precision, recall numbers for each individual moves and steps were calculated as micro averages.

Table 5-7 Micro Average Precision and Recall - Move classifier

Move #	Move name	Precision (%)	Recall (%)	Accuracy (%)
1	Establishing a territory	73.3	89	73.8
2	Identifying a niche	59.2	37.3	82.8
3	Addressing the niche	78.4	57.2	83.8
Average		**70.3**	**61.2**	**80.2**

Table 5-8 Micro Average Precision and Recall - Step classifier

Step #	Step name	Precision (%)	Recall (%)
1 (Move 1)	Claiming centrality	70.4	76.6
2 (Move 1)	Making topic generalizations	67.9	49.6
3 (Move 1)	Reviewing previous research	75.2	55.5
4 (Move 2)	Indicating a gap	51.4	55.1
5 (Move 2)	Highlighting a problem	86.7	85.2
6 (Move 2)	Raising general questions	66.3	50
7 (Move 2)	Proposing general hypotheses	44.6	51.9
8 (Move 2)	Presenting a justification	64.7	79.8
9 (Move 3)	Introducing present research descriptively	92	84.5
10 (Move 3)	Introducing present research purposefully	39.8	34.3
11 (Move 3)	Presenting research questions	50.6	61.8
12 (Move 3)	Presenting research hypotheses	68.9	66.2
13 (Move 3)	Clarifying definitions	74.2	43.4
14 (Move 3)	Summarizing methods	78.6	67.1
15 (Move 3)	Announcing principal outcomes	50	27.8
16 (Move 3)	Stating the value of the present research	100	18.2
17 (Move 3)	Outlining the structure of the paper	84.6	26.2
Average		**68.6**	**54.9**

CHAPTER 6. DISCUSSION

RWT offers genre-specific feedback to learners of academic writing, and therefore the study of both move and step classifier performance is important. The feature sets containing unigrams and trigrams for both moves and steps are split into three main categories. Feature sets contained "only unigrams" or "trigrams" or "both unigrams and trigrams," progressively increasing in number to study their correlation with performance characteristics. Based on the individual feature sets, move and step classifiers were built, and the individual models were evaluated using the performance measures and compared on their effectiveness.

6.1 Move Classifier Performance

Considering RWT's move classifier performance through various feature sets depicted in Figure 5-1 and Table 5-3 of Chapter 5, the accuracies of "unigram only" models are similar. Increasing the number of unigrams in the feature set doesn't proportionally increase the accuracy. Precision and recall increase marginally when the unigrams increase in the feature set. This implies that the initial move classifier model built on 1000 unigrams captures most of the features needed to differentiate among moves.

The "trigram only" models offer better precision numbers compared to the "unigram only" models, though the precision declines with increase of trigrams into the feature set. Recall measures are very low compared to "unigram only" models. "Trigram only" models carry precision information in identifying moves better than "unigram only" models.

By the numbers, combining unigrams and trigrams in a single feature set resulted in models of better accuracy. Highest accuracy values were result of models with feature sets

having “both unigrams and trigrams.” Precision was comparable and significantly high relative to “trigram only” models. Also, the model’s recall was best when all unigrams and trigrams were used to train the move classifier and the data suggest that recall values are proportional to total number of features used. A maximum accuracy of 72.655% for the move classifier was obtained when 5828 unigrams and 11630 trigrams were collectively used in the feature set. Macro average precision of 70.28% and macro average recall of 61.15% were the resulting performance evaluation measures of this model. The micro average precision, recall, and accuracy numbers of this move classifier are shown in Table 5-7, which shows that moves “Addressing the niche” (#3) and “Establishing a territory” (#1) are predicted with higher precision and recall than move “Identifying a niche” (#2). This can be attributed to the relatively low training and test data available move 2, while the other moves have better sample data for training (6,039 sentences for move 1; 1,609 sentences for move 2; and 2,352 sentences for move 3). This result is affirming previous research on IADE (Pendar & Cotos, 2008), where move 2 is documented to be difficult to predict owing to sparse data and possible misclassifying occurrences with move 1.

6.2 Step Classifier Performance

The step classifier performance was analyzed similarly using three different categories of feature sets, shown in Figure 5-2 and Table 5-6 of Chapter 5. The classifier trained using “unigram only” features showed good accuracy, proportionally increasing with an increase in the number of unigrams. Precision and recall, though relatively low when compared to accuracy, also proportionally increase with increase in unigrams.

“Trigram only” step classifier models depict a lower accuracy compared to all other classifier models. Precision values were not proportional and did not increase with more features. But, similar to the move classifiers, “trigram only” models carried better precision information about various steps compared to the “unigram only” models. Recall numbers were the lowest of all models tested, and increasing the number of features resulted in no change.

Finally, classifier models built using a combination of “both unigrams and trigrams” features exhibited performance similar with the “unigram only” models, though marginally resulting in better precision. Recall of the “both” models was similar to the “unigram only” models.

The step classifier using the maximum features of 27,689 unigrams and 27,160 trigrams delivered an accuracy of 72.01% with a macro average precision of 66.92% and macro recall of 54.90%. Considering micro average precision and recall of the step classifier in Table 5-8 of Chapter 5, steps belonging to move 1 and move 3 were identified more precisely than steps belonging to move 2. This result is clearly attributed to the sparse data availability of move 2 and its constituting steps. Steps which show precision below the average of 68% include: three steps from move 2 (highlighting a problem, raising general questions, proposing general hypotheses) and four steps from move 3 (introducing present research descriptively, summarizing methods, presenting research questions, announcing principal outcomes).

6.3 Step Classifier Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Total predicted	Precision
1	112	25	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	165	67.9
2	82	813	260	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1155	70.4
3	32	223	1658	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1913	86.7
4	0	0	0	106	27	0	2	6	0	0	0	0	0	0	0	0	0	141	75.2
5	0	0	0	69	321	13	41	52	0	0	0	0	0	0	0	0	0	496	64.7
6	0	0	0	3	1	5	0	1	0	0	0	0	0	0	0	0	0	10	50
7	0	0	0	1	19	0	57	9	0	0	0	0	0	0	0	0	0	86	66.3
8	0	0	0	12	34	0	14	133	0	0	0	0	0	0	0	0	0	193	68.9
9	0	0	0	0	0	0	0	0	222	26	17	11	5	57	32	50	19	439	50.57
10	0	0	0	0	0	0	0	0	10	92	4	1	0	2	5	3	0	117	78.63
11	0	0	0	0	0	0	0	0	2	0	11	0	0	0	0	0	0	13	84.61
12	0	0	0	0	0	0	0	0	4	1	0	23	0	0	1	1	1	31	74.19
13	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	100
14	0	0	0	0	0	0	0	0	50	4	4	2	2	82	22	13	5	184	44.56
15	0	0	0	0	0	0	0	0	30	3	4	13	1	12	91	16	7	177	51.41
16	0	0	0	0	0	0	0	0	32	11	2	1	1	4	11	45	6	113	39.82
17	0	0	0	0	0	0	0	0	9	0	0	2	0	1	3	3	207	225	92
Total annotated	226	1061	1946	191	402	18	114	201	359	137	42	53	11	158	165	131	245		
Recall	49.6	76.6	85.2	55.5	79.9	27.8	50	66.2	61.8	67.2	26.2	43.4	18.2	51.9	55.2	34.4	84.5		

Figure 6-1 Step Classifier - Confusion matrix

Steps predicted by the step classifier built on the total feature set of 27,689 unigrams and 27,160 trigrams was compared alongside a human expert's prediction using the test corpus (Figure 6-1) is the global contingency table or the confusion matrix). Rows represent human expert classification while columns represent machine predicted classification of steps. The diagonal of the matrix is highlighted, showing the machine predictions matching human expert predictions while other cells imply classification mismatch or misclassification to a different step. In line with observations of Anthony and Lashika (2003), a step is confused only within the realm of its move, that is, steps are confused with other steps within the same move. Step confusion occurred mainly with steps performing under stated average precision of 68%. In move 1, step 1(claiming centrality), step 2(making topic generalizations) and step 3 (reviewing previous research) are confused and vice-versa. In move 2, step 5 (highlighting a problem) was confused with step 6 (raising general questions),

step 7(proposing general hypotheses) and step 8 (presenting justification). In move 3, step 9 (introducing present research descriptively) was the most confused, misclassified as step 14 (summarizing methods), step 15 (announcing principal outcomes) and step 16 (stating the value of present research). Step 14 was also confused with step 15.

The major reason for misclassification also specified in the previous research was the scarcity of training data and uneven distribution of steps in the overall training corpus. Also, some steps were naturally heavily utilized and represented by the authors, while the others where sparingly used, resulting in an uneven distribution. Naturally, stretches of text also carry multiple rhetorical functions representing more than one step or a different move. The move and step classifiers are currently capable of predicting a single move and step and hence allow possible misclassification. Yet another important factor pointed out by Cotos (2013), was meaning ambiguity. She noticed that, despite annotators of the corpus having high interrater reliability, they can become confused over the author's rhetorical intent in the absence of lexical signals of functional meaning. Some steps are inherently difficult to be identified using natural language techniques, as they are not completely captured and encoded in functional language. This situation increases the difficulty operationalizing our learning models.

6.4 Future Work

In this work, genre analysis and machine learning have come together, relying on linguistic cues to successfully identify rhetoric functions. The move level classifier classifies new sentences with an accuracy of 72.6%, and the classifier at the step level performs at an accuracy of 72.9%. Future research concentrating on improvement of classifier accuracy will

directly enhance AWE and ICALL systems. SVMs offer probability-based training and probability estimates during prediction which could be used to identify secondary moves and steps in learner texts after experiments. Based on probability estimates on various steps and moves, the most probable element could be associated with the primary function while the next probable could be associated with the secondary function. For steps increasingly difficult to detect, a knowledge-based approach (as in Madnani, Heilman, Tetreault, & Chodorow, 2012) can be used along with SVMs. Handwritten rules as suggested by Chukharev-Khudilaynen (2013) could be used to recognize functional language and lexico-grammatical patterns identifiable in the annotated corpus. Adding this expert knowledge to the SVM classifier could be used to augment predictions. Yet another implementation to handle confusing steps involves use of Markov chains to model move/step transitions and their sequencing. One of the main faulty assumptions with the existing approach is the lack of accounting for context in each sentence considered. Context plays an important role in identifying rhetoric functions, and future experiments could also consider context for building prediction models. Application of boosting algorithms in machine learning and use of ensembles of learner classifiers are other venues for experimentation towards improvement.

6.5 Summary

This section summarizes the research work done related to the questions from section 1.5. The Research Writing Tutor (RWT) is able to analyze a student's research articles at the discourse level and identify various rhetorical moves and steps (Question 1). This achievement is a novel contribution to the AWE field, as RWT is one of the pioneer AWE

tools implementing discourse evaluation at the step level for the academic writing genre. The final move classifier was trained using a total of 5828 unigrams and 11630 trigrams and performed at a maximum accuracy of 72.65%. Similarly, the step classifier was trained using a total of 27689 unigrams and 27160 trigrams and performed at a maximum accuracy of 72.01%. These statistics address Question 2, on what combination of unigram and trigram features was most appropriate for optimal move and step classification. But more training data could always improve the above statistics especially for the steps which are underperforming in terms of good precision and recall. Odds ratio based feature weighing scheme, indeed identified the best features as implied by section 5.3. There was no direct linear relation between number of features and the performance metrics accuracy, precision and recall. But the overall performance of the classifiers consistently improved with a larger feature set covering Question 3. This also opens the venue for experiments using a larger feature set through more training data. Using discipline-specific features weighed by odds ratio, resulted in a single classifier for both move and step that performed consistently across different disciplines. This approves the feasibility requirement from Question 4 and also implies that odds ratio measures indeed identify actual discipline-specific features (Question 5). Odds ratio measures were effective in discarding repetitive features that were common among various rhetorical moves, steps and disciplines. The default standard regularized support vector classification algorithm C-SVC was used to train the classifiers. RBF kernel was chosen with the cost parameter C set to be 1511. From the performance metrics of both the move and step classifier in section 5.3, an RBF kernel is able to model the relation between features and the rhetorical moves/steps with a significant accuracy of above 70%.

REFERENCES

Anderson, J. R. (1988). The expert module. *Foundations of intelligent tutoring systems*, 21–53.

Aston, G. (2002). The learner as corpus designer. *Language and Computers*, *42*(1), 9–25.

Attali, Y., & Burstein, J. (2006). Automated essay scoring with e-rater® V. 2. *The Journal of Technology, Learning and Assessment*, *4*(3).

Biber, D., Conrad, S., & Reppen, R. (1998). *Corpus linguistics: Investigating language structure and use*. Cambridge University Press.

Biber, Douglas, Connor, U., & Upton, T. A. (2007). *Discourse on the Move: Using Corpus Analysis to Describe Discourse Structure*. John Benjamins Publishing.

Bird, S. (2006). NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions* (pp. 69–72). Association for Computational Linguistics.

Bland, J. M., & Altman, D. G. (2000). Statistics Notes: The odds ratio. *BMJ : British Medical Journal*, *320*(7247), 1468.

Britt, D. H. (1967). Learner types in computer-controlled instruction. *Paedagogica Europaea*, 70–96.

Calfee, R. (2000). To grade or not to grade. *IEEE Intelligent Systems*, *15*(5), 35–37.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*(3), 27.

Chen, C.-F. E., & Cheng, W.-Y. E. (2008). Beyond the design of automated writing evaluation: Pedagogical practices and perceived learning effectiveness in EFL writing classes. *Language Learning & Technology*, *12*(2), 94–112.

Cherkassky, V., & Mulier, F. M. (2007). *Learning from data: concepts, theory, and methods*. Wiley-IEEE Press.

Cherkassky, Vladimir, & Ma, Y. (2004). Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, *17*(1), 113–126. doi:10.1016/S0893-6080(03)00169-2

Cheville, J. (2004). Automated Scoring Technologies and the Rising Influence of Error. *The English Journal*, *93*(4), 47–52. doi:10.2307/4128980

Chukharev Hudilainen, E. (2013). Phase 2 - Revised RWT Training and Test Subsystem.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297.

Cotos, E. (2010). *Automated writing evaluation for non-native speaker English academic writing: The case of IADE and its formative feedback*. Iowa State University.

Cotos, E., Gilbert, S. B., & Link, S. (2012). Scaling Up Automated Writing Evaluation for Learning.

Croft, B. (1987). An approach to natural language for document retrieval. In *Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 26–32). ACM.

Day, D., McHenry, C., Kozierok, R., & Riek, L. (2004). *Callisto: A Configurable Annotation Workbench*.

Dudley-Evans, T. (1995). Common-core and specific approaches to the teaching of academic writing. *Academic writing in a second language: Essays on research and pedagogy*, 298–312.

Elliot, S. (2003). IntelliMetric: From here to validity. *Automated essay scoring: a cross disciplinary approach. Mahwah, NJ: Lawrence Erlbaum Associates*.

Flowerdew, L. (1998). Concordancing on an expert and learner corpus in ESP. *CÆLL Journal*, *8*(3), 3–7.

Foltz, P. W., Laham, D., & Landauer, T. K. (1999). The intelligent essay assessor: Applications to educational technology. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, *1*(2).

Garrette, D., & Klein, E. (2009). An extensible toolkit for computational semantics. In *Proceedings of the Eighth International Conference on Computational Semantics* (pp. 116–127). Association for Computational Linguistics.

Grimes, D., & Warschauer, M. (2006). Automated essay scoring in the classroom. In *Annual Meeting of the American Educational Research Association, San Francisco, CA*.

Halff, H. M. (1988). Curriculum and instruction in automated tutors. *Foundations of intelligent tutoring systems*, 79–108.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: data mining, inference, and prediction*. Springer New York:

Herrington, A., & Moran, C. (2001). What Happens When Machines Read Our Students' Writing? *College English*, *63*(4), 480–499. doi:10.2307/378891

Hipp, D. R., & KENNEDY, D. (2007). SQLite. *Available fr om: http://www. sqlite. org*.

Hoey, M. (1979). Signalling in discourse.

Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. *The annals of statistics*, 1171–1220.

Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). *A practical guide to support vector classification*.

Huang, L.-S. (2010). Seeing eye to eye? The academic writing needs of graduate and undergraduate students from students' and instructors' perspectives. *Language Teaching Research*, *14*(4), 517–539.

Hyland, K., & Hyland, F. (2006). *Feedback in Second Language Writing: Contexts and Issues*. Cambridge University Press.

Ikonomakis, M., Kotsiantis, S., & Tampakas, V. (2005). Text classification using machine learning techniques. *WSEAS transactions on computers*, *4*(8), 966–974.

Joachims, T. (1996). *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.* DTIC Document.

Kirk, R. (2011, 2012). Knowledge transfer of existing processes - database, model training and testing modules.

Kiss, T., & Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, *32*(4), 485–525.

Kivinen, J., Warmuth, M. K., & Auer, P. (1997). The Perceptron algorithm versus Winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, *97*(1), 325–343.

Kulik, J. A., Kulik, C. L. C., & Cohen, P. A. (1980). Effectiveness of computer-based college teaching: A meta-analysis of findings. *Review of educational research*, *50*(4), 525–544.

Kwok, J. T., & Tsang, I. W. (2003). Linear dependency between ε and the input noise in ε-support vector regression. *Neural Networks, IEEE Transactions on*, *14*(3), 544–553.

Lewis, D. D. (1991). Evaluating text categorization. In *Proceedings of speech and natural language workshop* (pp. 312–318).

Madnani, N., Heilman, M., Tetreault, J., & Chodorow, M. (2012). Identifying high-level organizational elements in argumentative discourse. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 20–28). Association for Computational Linguistics.

Mann, W. C., & Thompson, S. A. (1988). Rhetorical structure theory: Toward a functional theory of text organization. *Text*, *8*(3), 243–281.

Mattera, D., & Haykin, S. (1999). Support vector machines for dynamic reconstruction of a chaotic system. In *Advances in kernel methods* (pp. 211–241). MIT Press.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill Boston, MA:

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education (IJAIED)*, *10*, 98–129.

MySQL, A. B. (2004). MySQL database server. *Internet WWW page, at URL: http://www. mysql. com (last accessed/1/00)*.

Pendar, N., & Cotos, E. (2008). Automatic identification of discourse moves in scientific article introductions. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications* (pp. 62–70). Association for Computational Linguistics.

Pendar, Nick. (2011, 2013). Unigrams and Trigrams , Odds Ratios and Classifier construction.

Pendar, Nick, Cotos, E., & Babu, D. P. (2012). NLP-based analysis on rhetorical functions for AWE feedback. Presented at the Technology for Second Language Learning (TSLL), Ames,Iowa.

Porter, M. (2009). {The Porter Stemming Algorithm}.

Ramaswamy, N. (2012). *Online Tutor for Research Writing (unpublished doctoral dissertation)*. Iowa State University, Ames,Iowa.

Ramaswamy, N., & Gilbert, S. (2012). Design and User Experience Study of the Automated Research Writing Tutor. Presented at the Technology for Second Language Learning (TSLL), Ames,Iowa.

Rhetorical Structure Theory. (n.d.). Retrieved January 13, 2013, from http://www.sfu.ca/rst/01intro/intro.html

Richardson, J. J. (1988). *Intelligent Tutoring Systems*. Psychology Press.

Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann.

Schölkopf, B., & Smola, A. J. (2002). *Learning With Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press.

Schwartz, B., Zaitsev, P., & Tkachenko, V. (2012). *High Performance MySQL: Optimization, Backups, and Replication*. O'Reilly Media.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, *34*(1), 1–47.

Skehan, P. (1989). *Individual differences in second-language learning*. Cambridge Univ Press.

Sleeman, D., & Brown, J. S. (1982). *Intelligent tutoring systems*. Academic Press. Retrieved from http://books.google.com/books?id=gfBEAAAAYAAJ

Smola, A. J., Murata, N., Schölkopf, B., & Müller, K. R. (1998). Asymptotically optimal choice of ε-loss for support vector machines. In *PROCEEDINGS OF THE 8TH INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS, PERSPECTIVES IN NEURAL COMPUTING, PAGES 105--110*. Citeseer.

Soergel, D. (1985). *Organizing information: principles of data base and retrieval systems*. Morgan Kaufmann.

SQLite Home Page. (n.d.). Retrieved March 25, 2013, from http://www.sqlite.org/

Suppes, P. (1980). The teacher and computer-assisted instruction. *The computer in the school: Tutor, tool, tutee*, 231–235.

Swales, J. (1981). *Aspects of article introductions*. Language Studies Unit, University of Aston in Birmingham.

Swales, J. M. (1990). Nonnative speaker graduate engineering students and their introductions: Global coherence and local management. *Coherence in writing: Research and pedagogical perspectives*, 187–207.

Towne, D., & Munro, A. (1992). *Supporting diverse instructional strategies in a simulation-oriented training environment*. Lawrence Erlbaum, Hillsdale, NJ.

Vapnik, V. (1999). *The nature of statistical learning theory*. springer.

Warschauer, M., & Ware, P. (2006). Automated writing evaluation: Defining the classroom research agenda. *Language teaching research*, *10*(2), 157–180.

Yang, N. D. (2004). Using MY Access in EFL Writing. The proceedings of 2004 International Conference and Workshop on TEFL & Applied Linguistics.

Yang, Y. (1995). Noise reduction in a statistical approach to text categorization. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 256–263). ACM.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information retrieval*, *1*(1), 69–90.

Yang, Y., & Pedersen, J. O. (n.d.). A comparative study on feature selection in text categorization.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to people who helped me at various stages of my research work and with the thesis. Firstly, I would like to thank my major professor, Dr. Stephen Gilbert for his guidance, support and for being a wonderful mentor throughout the course. I admire his direction, way of thinking and especially his expert research sources and guidance. He gave me an opportunity to work for RWT which I enjoyed thoroughly and hope i have met some of his expectations. I sincerely thank Dr. Elena Cotos for all the guidance, feedback and opportunity she gave to work for RWT and especially the related publications. Her requests for functionality with RWT helped me understand the goal of RWT and its pedagogical applications. I thank Dr. Jin Tian for his guidance and pointers towards the latest research in state of the art machine learning techniques including the deep learning models. His machine learning class was the prior base to my understanding the requirements of RWT research. I would also like to thank Dr. Nick Pendar for his valuable consulting time in educating me to use SVMs appropriately and sharing his practical experience to handle mishap situations when training computational models. I owe a lot to Dr. Evgeny Chukharev Hudilainen and thank him sincerely for sharing his professional experience with optimizing architectures and helping me with the same. His proposals moved RWT a level up and I am happy to implement them. I intensely enjoyed working with all the great minds above and i truly value and carry important qualities from each of them. I also thank my peers Ryan Kirk, Nandhini, Andrew Vernon, Jinsook Kim, Sarah Huffman, Aysel and Vijay Kalivarapu for their help throughout the course of RWT. Finally I thank my family and friends to whom I dedicate my work and its purpose.