Software for Writing Assistance and Improvement for Advanced Learners of English

A Thesis Presented

by

Diane M. Napolitano

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

December 2008

Stony Brook University

The Graduate School

Diane M. Napolitano

We, the thesis committee for the above candidate for the Master of Science degree, hereby recommend acceptance of this thesis.

Amanda Stent - Thesis Advisor
Assistant Professor, Computer Science

Steven Skiena
Professor, Computer Science

David Warren
Professor, Computer Science

This thesis is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Thesis

Software for Writing Assistance and Improvement for Advanced Learners of English

by

Diane M. Napolitano

Master of Science

in

Computer Science

Stony Brook University

2008

Many writing assistance systems use a one-size-fits-all approach, treating every individual's problems as equivalent and failing to help the user overcome them. Many users become reliant on these software packages and their skills rarely improve. Software should teach the student, not just fix their errors; the student should be required to fix some of the errors manually so that they learn how to correct them and will, eventually, no longer need the software. However, if the student is always asked to manually correct repeated mistakes, they could become uninterested in using the program. Here, we explore techniques for making effective software to accomplish these goals, among others. We analyze the strengths and weaknesses of existing systems and discuss their relationship to ours, we describe the system we have built, TechWriter, and discuss the hurdles one encounters when building any such system, and we provide some discussion on relevant experiments we have conducted, in an effort to find data and techniques that could improve our system's performance. Our goals with the system we have built, TechWriter, are to analyze the students writing, point out their mistakes, make them correct the mistakes, let the program learn from the way the corrections are made and automatically apply the corrections in the future. In this way, TechWriter both assists the student with their writing and improves their writing ability. In addition to the automated error detection, TechWriter provides the student with many NLP tools that they can use to gain a better understanding of their discourse. Our target audience is advanced learners of English, both native and non-native.

Table of Contents

List of Figures

Acknowledgements

I would sincerely like to thank my advisor, Amanda Stent, for all of her guidance and support over the course of both this project and my degree program. I would also like to thank all of my friends and family for their continued support and motivation, despite the fact that their efforts mainly consisted of asking me, "You're still working on that?" In particular, I appreciate the immeasurable patience from my boyfriend, Chris, and my friend Justin for his help with regular expressions and for entertaining my ideas, even though they were well outside his areas of expertise. I am grateful to both him and Mike Hart for proofreading this and supplying me with valuable feedback.

Introduction

In ancient societies, literacy was often a privilege granted on the basis of money, birthright, or social standing; in our modern world, it is a bare necessity. Poor writing skills can often serve as a disadvantage to a student who is otherwise quite bright and hard-working, and the development of these skills is something that many students overlook, particularly in a discipline such as Computer Science where emphasis is placed on mathematical ability and abstract reasoning. Writing is a key component of communication, and the ability to effectively communicate is at the top of a list of skills that employers look for in prospective hires, according to both the National Association of Colleges and Employers [1], and also the Association for Computing Machinery [2]. Writing is an important skill which needs to be developed, no matter what sort of career path a student chooses to take. A student wishing to enter academia, or simply pass their college courses, can find themselves at a disadvantage with inadequate writing skills; a problem which may seem insurmountable, especially if their writing is required to be in a language different from their native one.

Today, it is not only important to be literate, but also to be well-versed in the English language. English is regarded as the language *du jour* around the world, and is used, almost exclusively, in both business and academia. Approximately 750 million people use English as a second language—as opposed to 375 million people who speak it natively—and as much as 74 percent of all writing done in English is done by non-native speakers [9]. In 2001, as many as 53 percent of all students in graduate engineering programs in the United States were temporary visa holders, [11] and for the academic year spanning 2003-2004, a sizable majority of Computer Science graduate students in the United States—52.8 percent at the Ph.D level and 50.6 percent at the Master's—were nonresident aliens [25]. Science education in the US, especially on the graduate level, is incredibly popular with international students, as it rewards them with a strong degree and a solid understanding of English. These students are entering the US with varying degrees of writing and speaking ability in English, as they are required to pass the Test of English as a Foreign Language (TOEFL) prior to being admitted to their graduate programs; however, no matter how advanced their grasp of the language is, even some non-native speakers can tell

the difference between writing done by a native or non-native speaker [16].

Our work here focuses on these two types of students: native English speakers whose writing skills are somewhat poor, and non-native English speakers with some significant understanding of the English language. For the sake of brevity, we refer to the former group as the "L1"—students who are in the process of gaining a better grasp of their first language—and the latter group as the "L2"—students who are learning, or working on improving their knowledge of, a second language. These two groups contain students who are fairly *advanced* in their English writing skills, and will be treated as such in the type of assistance and feedback they receive on their work.

These advanced English learners need occasional assistance with their writing, and their style and grammar could benefit from some improvement. The International Corpus of Learner English (ICLE) [12] consists almost entirely of this type of writing; it contains essays written by students in the L2 whose English writing is not exactly wrong, but when read by a native speaker with a strong command of the language, not quite right. The authors of these essays, however, are highly advanced in their learning. For example, consider the following sentence from the Russian subcorpus[1]:

(1) The matter is that sometimes it's difficult to make them answer in proper time.

While this sentence is not altogether wrong, it is somewhat awkward, and some of the terms and expressions are ones which may be proper in Russian, but not in English; for instance, in English we would say "the problem" or "the issue", instead of "the matter". If we are to assume that this sentence comes from an essay that was thoroughly proof-read by the author, we can see that this type of error is difficult for a non-native writer of English to detect. Lacking a second proof-reader, the student could perhaps benefit from software that could not only point out their mistakes, but also offer suggestions on how to fix them.

Many software packages already exist that attempt to *assist* the student in this way, but often do little to *teach* them the correct way to write, failing to help them

[1]By "Russian subcorpus", we mean a subcorpus from ICLE where the authors were all native Russian speakers, writing in English.

improve on their personal skills. It is easy for the student to become dependent on this type of software, and to pay little attention to the way they are fixing their mistakes—they may just simply choose a suggested correction from the list without thinking, in an effort to silence the program. The suggestion they have chosen may not have been the best one in terms of the overall flow of the sentence or paragraph, or may not capture the mood or idea they were trying to convey. A larger context window is required to make accurate suggestions to the student and should contain a list of past corrections of the same type made throughout the entire document. In other words, the software should become *personalized*, to some extent, to the writing style and errors made by the student. A general, "one-size-fits-all" treatment of the student's deficiencies can fail to help them understand their specific mistakes, and can only offer them broad suggestions on how to fix them.

In order to accomplish this, we have developed TechWriter, which adopts the following approach:

- Detect errors in the student's writing, and provide a precise explanation of every error.
- Offer suggestions for correcting the error, and allow the student to select one or input their own.
- Have the student repeat this process a small number of times—this forces them to learn from their mistakes, thereby *improving* their writing ability.
- Have the software learn from the way the student is correcting his or her mistakes, and then automatically apply those corrections for the student—this keeps the student from becoming frustrated by the monotony inherent in fixing repeated mistakes, and provides them with some level of *assistance* with their writing.

TechWriter compares the student's sentences and paragraphs against a corpus of American English writing and outputs patterns in their writing which are not found in the corpus. An example of this is the sentence (1) from the Russian L2 student. In order to detect errors and offer suggestions for them, TechWriter will take this

sentence and tag every word with its part-of-speech (POS), and separately, stem every word in the sentence, producing the following output:

(2) The/DT matter/NN is/VBZ that/WP sometimes/RB it's/RBR difficult/JJ to/TO make/VB them/PRP answer/VB in/IN proper/JJ time./.

(3) the matter is that sometim it s difficult to make them answer in proper time

These two sentences are then compared to data from the corpus, and if a portion of the sentence is not found in the corpus, it is reported back to the user as an error. In particular, for (3), TechWriter gives a long list of errors, each of which has a number of suggestions for the student. One such error, "the matter is", receives the following suggestions, among others: "the problem is", "the result is", "the situation is", "the answer is", "the economy is", "the idea is", and so on. Many of these are reasonable suggestions for this sentence, but the ideal one that best fits, given the context, would be "the problem is", which is then replaced in the original sentence, giving the following result:

(4) The problem is that sometimes it's difficult to make them answer in proper time.

TechWriter will be explained in full in chapter 3, where we will discuss its current operation, as well as ideas that never quite came to be incorporated into the system and also future directions. In chapter 2, we will discuss some work that is similar in theme and functionality to TechWriter, and also discuss some issues that arise when developing such a system. In chapter 4 we will detail experiments we have performed that are either directly related to TechWriter, or found to be less valuable. We will present the results of these experiments within chapter 4 and offer our final thoughts and conclusions in chapter 5.

Chapter I

Related Work

Many different tools already exist that can help the student with their writing: grammar checking software, spell checking software and the Internet, to broadly name a few. Setting aside the last two for now, many students happen to think that their writing problems can be solved with grammar-checking software; but as we saw with our example in the introduction, some sentences can simply be ill-formed according to the rules of the language, yet still grammatically-correct. Gregor Thurmair precisely defines the difference between grammar and style: "Grammar checking tries to find ill-formedness which by definition is considered to be a mistake and MUST be corrected; style checking has to do with well-formed but somehow marked texts" [24]. Grammar and spelling are certainly vital, but they do not make up the whole picture, and both receive moderate attention in our work.

Although Thurmair developed a grammar checker for German, he discusses some key issues that appear in the development of any such tool. In order to detect stylistic errors, a parser must be able to successfully parse any given sentence, but to accomplish that, "information must be used which could have been violated". In other words, the parser must essentially know what the correct sentence should look like and perform the parse on that sentence; otherwise it will not perform a proper parse. To borrow another example from the ICLE, this sentence, written by a native German speaker,

(5) And the strong feelings you get may not only be positive.

is POS tagged by our software as

(6) And/CC the/DT strong/JJ feelings/, you/PRP get/VBP may/MD not/RB only/RB be/VB positive./VBN

As can be seen, the word "feelings" is mis-tagged, and should be tagged as it would in a "proper" sentence such as

(7) She/PRP developed/VBD strong/JJ feelings/NNS towards/IN him./NN

Parsers and taggers are vital tools which can provide information that can be extremely helpful to detect errors, but in order to be useful, the tools need to tag the input correctly. In order to do this successfully, they almost need to know the correct form of the input in order to produce the desired output—they must know the output before producing the output [24]. Parsers and taggers rely on the output that they have already produced in order to properly parse or tag the current word or phrase, within some window of context. For a word Y, surrounded by words—or a sequence of words—X and Z, Y will not be properly tagged if the tagger cannot determine its tag by looking at the tags for X and Z; if, perhaps, they are mis-tagged or the tagger cannot find a matching sequence of tags in its model from which to infer the tag for Y, based on X and Z.

This ties in to another problem, in which a sentence may be error-free, but still unable to be properly tagged or parsed. Consider the sentence

(8) The man saw a boy with a telescope.[1]

This sentence is completely correct, but incredibly ambiguous; did the man use a telescope to see the boy, or did the boy have a telescope when the man saw him? If this sentence did indeed contain an error, our software would not be able to provide much useful information to directly attack that mistake; as Thurmair states, a parser, operating on any such sentence, "has to find 'the best path' and interpret it" [24]. Our software's dependency parser decided to choose the man seeing the boy as the

[1]This example is from Church, K. and R. Patil. "Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table". *Computational Linguistics*, 1982.

nominal subject of the sentence and the boy being seen by the man as the direct object[2], but it is entirely possible that the writer of that sentence could have meant the opposite. This is more of an issue when the student writes a sentence that is both ambiguous and incorrect; the software could potentially have them fix it in such a way that could destroy the original intention of the sentence, unbeknownst to the student who operates under the assumption that the software is never wrong.

There is a large body of work relating to solving these and other problems, both as key components or completely independently of larger, comprehensive systems such as ours. We will first discuss these systems and highlight ways in which they differ from ours. We will then talk about some related technologies that are pertinent to our task, and how we have either incorporated them, discarded them, or could potentially use them to improve our system.

I.1 Comprehensive, Complete Systems

In 1992, Genthial and Courtin proposed a detailed architecture for a system they called "Computer Aided Writing", or CAW [10]. The system is motivated by their perception that, "In their life-cycle from creation to publishing, all texts nowadays take an electronic form", so logically, every aspect of the development and maintenance of texts should be computerized, not just their initial creation and layout. CAW would include several modules that would each handle typographic errors, spelling errors, lexical errors, syntactic agreement, and also standard word processor features such as layout and printing. Each of the modules would be capable of sharing data with one another, although some would require input from other modules and one, the Lexicon, only provides information to the other modules. The information provided in the Lexicon is "all of the knowledge of the system": essentially a dictionary of words and their morphological, syntactic and semantic roles. This data would be used in almost all of the modules, particularly in the modules which perform error

[2][det(man-2, A-1), nsubj(saw-3, man-2), det(boy-5, a-4), dobj(saw-3, boy-5), det(telescope-8, a-7), prep_with(saw-3, telescope-8)]

detection and correction, which they propose to be done both manually and automatically. When an error is made, their system would suggest one or more corrections to it, based on the data in its lexicon; but in most cases, they feel "the good correction can be chosen automatically", implying that their system would utilize error correction which is almost entirely automatic, unlike our system. For selecting the best possible correction, Genthial and Courtin propose the following criteria for their software to use [10]:

- Favor corrections where the lowest number of errors are corrected within a "group"; for example, "little cat are funny pets" would be corrected to "little cats are funny pets" as opposed to "a little cat is a funny pet".
- Make corrections that do not destroy the original phonetic meaning of the sentence; i.e., favor "the ski slides" over "the skis slide" for fixing the phrase "the skis slides".
- Give priority to the "head" of the phrase, which would correct "those who is" to "those who are". Here, it is safe to assume that the head of such a phrase is in line with the idea the student wished to express, and it is the words written to support it that are in error, since they are not as integral as the idea itself (as with our example in chapter 1).
- It is unlikely that a writer added a letter to a word, or an additional word to a sentence, and more probable that they will omit one, so favor corrections that add necessary letters or words rather than those that remove a word already in the sentence.

It is not evident from the paper whether or not Genthial and Courtin ever released a version of their system for public use, but one similar, modularized system that was actually quite popular was the Writer's Workbench [17]. The Writer's Workbench was a series of UNIX command line programs that hinged on the `style` and `diction` commands[3] and provided proofreading, comments on style and a reference manual

[3]These programs have been since ported to Linux and are available from the Free Software Foundation.

for the English language. The proofreading program, `proofr`, invoked five separate programs to perform spelling correction, check punctuation, determine if a word was repeated consecutively, display potentially "poor" phrases to the student (with a related program that offered improvements to these), and check for split infinitives. There were separate commands to give stylistic feedback, based on an analysis of parts of speech, and also some interesting programs that told the user how abstract or sexist (according to guidelines supplied by Bell Labs) their document was. Overall, however, the software was limited, as it did not provide the student with the means of correcting their work online; it was up to them to take the feedback from the Writer's Workbench, interpret it, and manually apply it to their document.

The Writer's Workbench did, however, contain a lot of useful, relevant work, and its output was extremely helpful to many writers. Most successful systems have done well to combine these tools with an architecture along the lines of that proposed by Genthial and Courtin, an example of which could be the Educational Testing Service's *Criterion* service [5]. This system contains an application called *Critique* which detects "numerous errors under the broad headings of grammar, usage, and mechanics" using bigrams of adjacent words and POS tags created from a corpus of 30 million words of newspaper text. It relies on bigram frequencies to determine whether an error exists, and uses pointwise mutual information to determine whether an observed bigram occurs more or less often than it is expected to, with log-likelihood serving in cases where the data is sparse. When compared to the other systems, our error detection operates most similar to *Critique*'s in that we also rely on the frequencies of n-grams in our analysis; where *Critique* differs in this regard is in their use of "filters" for "low probability, but nonetheless grammatical, sequences". At the moment, our system simply treats everything not found in the corpus as an error and offers suggestions.

In *Criterion*, the errors found in an essay are reported to the student, but, similar to the Writer's Workbench, there is no evidence that the system gives suggestions to the student on how they could fix the error. Providing suggestions to the student is one of the most important aspects of PENS, a system specifically designed for Chinese learners of English [16]. PENS employs a parallel corpus of both Chinese and English

which allows the student to see what they just wrote in English in Chinese, and vice-versa; ideally, this should help them find their mistakes in the foreign language if they can see them in their native language. PENS provides suggestions to the student by querying its corpus of sentences with both the original student's sentence and expanded queries based on this. The results are ranked and the top two are presented to the user. This is in contrast to the way our system currently operates, which is to offer suggestions at the n-gram level, but is nonetheless very similar in basic functionality and purpose.

Other systems choose to follow the "web as a corpus" model, rather than use their own corpus. Moré et al. developed a grammar and style checker that relies on data in WordNet and results from Yahoo and AltaVista [19], but fails to consider the possibility that, simply because something exists on the Internet, it is not automatically correct (or, for that matter, automatically incorrect). Having a constant corpus from which to draw from seems to be a much more effective and practical approach.

To summarize, Genthial and Courtin outlined an interesting system for writing assistance and improvement, which was similar to its predecessor, the Writer's Workbench, but more unified and complete. These ideas were expanded upon with *Criterion*, and new directions in the same vein were explored with PENS. Each of these systems relies on individual components, smaller parts combined to create the whole; it is these technologies we will focus on next.

I.2 Components of Writing Systems

In this section we will focus on software that has been developed to perform one specific task, potentially within a larger system similar to ours. Some of this work is in the vein of fixing the problems presented by Thurmair in regards to parsing error-laiden text. For instance, Eric Atwell developed CLAWS, the "Constituent Likelihood Automatic Word-tagging System", which attempts to detect grammatical errors without performing any parsing [3]. CLAWS uses a first-order Markov model to first assign tags, then for each set of ambiguously-tagged words, a Constituent Likelihood Grammar is used to assign a "likelihood" to each tag within that set. The

product of the likelihoods for that set is compared against a frequency table to see if the value indicates a possible error in the set. Ideally, the tags assigned to the words will be correct for each of them, and with this procedure, errors should be detected during the process, as opposed to afterwards, where tagged sentences are examined for errors. If CLAWS works as described, it is a potential solution to the problems discussed by Thurmair.

Another attempt at solving the parsing problem was made with ICICLE, a system targeted specifically for L2 learners of English [23]. With ICICLE, authors Schneider and McCoy attempt to augment a standard parser with "mal-rules", which are designed to catch common "ungrammaticalities", such as missing words and noun-determiner disagreements. A grammar is used to parse sentences and is catered specifically to the problems made by students of the native language ICICLE was designed to work with (American Sign Language), and as a result of this, when combined with the mal-rules, the parser actually works quite well. It properly tags 63% of the evaluation sentences for determiner and agreement errors, which indicates that a parser designed for and trained on text written in the L2 by students with a different L1, as opposed to a general purpose English parser, is a very viable option.

A different way to look at the same problem is to use a parser, but to only focus on one type of error. Work done by Chodorow et al. focuses on determining preposition errors, which account for about 29% of all errors committed in L2 English writing [7]. Their approach uses a Maximum Entropy classifier to determine whether or not the correct preposition has been selected (for example, "The book was *on* the table" as opposed to "The book was *at* the table") which was trained on "events". Each event consists of the preposition as it is used by data within the training corpus and a feature-value pair with its associated context, extracted from the corpus after having been POS tagged and chunked into noun phrases. The system works quite well, and while it does perform some parsing, it is done on text assumed to be correct for the purposes of this system, decreasing the possibility for errors made by the system as a result of parsing.

However, work done by Chodorow and Leacock attempts to find errors without doing any parsing. They developed ALEK, a system which is able to "automatically

identify *inappropriate usage* of specific vocabulary words in essays by looking at the local contextual clues around a target word" [6]. Here, error detection is simply an extension of word-sense disambiguation, where the intended sense of a polysemous word is determined by examining example sentences that cover all of the possible senses for that word, extracting "contextual clues" from these sentences, and using this data to build a model for each sense. While our system does not exactly include word-sense disambiguation, it does rely on example sentences from our corpus to determine the correct term or phrase the student perhaps meant to use.

While all of these systems target writing which is highly error-prone, only the work done by Chodorow et al. makes use of an "error corpus", a collection of documents that are not guaranteed to be perfectly correct, and with good reason. Error corpora can be useful in detecting errors made by a student: if the student's writing exists in the corpus, it is safe to assume it is an error.[4] Nagata et al. use a "feedback corpus", a collection of corrected essays written by L2 students, to account for differences in the correct English used in corpora such as newspaper texts and in the English used by these students [21]. An unfortunate result of this is that the feedback corpus was found to have little or no impact on the performance of the program, due to its size; the size of their corpus of general text was significantly larger than their feedback corpus, and overpowered it.

Perhaps Nagata et al.'s method would have been improved if they added students' writing, as it was corrected, to the corpus "on-the-fly". General corpora tend to consist of newspaper articles and other such texts whose writing is both grammatically and stylistically correct, thus useful for an application such as ours where writing improvement is a goal. An error corpus can also be useful to support these general corpora, since not all newspaper text can be guaranteed to be error-free.

[4]This can be supported further by an "error-free corpus"; if the phrase exists in the error corpus and not in this one, it must almost definitely an error.

Chapter II

System Description

Our solution to the problems posed thus far is known as TechWriter, and is centered on improving technical and formal, academic writing in advanced L1 and L2 students. The technologies and knowledge available to TechWriter specifically cater to this and to our overall concept of software that both assists with and improves the student's writing. We shall demonstrate this in the subsequent sections of this chapter; we will begin with a broad overview of TechWriter's functionalities, discuss it's error detection and suggestion system in-depth, then consider some future directions for the system.

II.1 Features and Functionalities in TechWriter

Our system is written entirely in Java as a plugin for the open source text editor, jEdit [14]. jEdit is a fully-functional, feature-rich programmer's editor similar to Emacs; we chose it, therefore, because we felt students in technical programs would be keen on using it for purposes other than TechWriter, thus they would not have to download much additional software. Another advantage to using jEdit is its inherent multiple windowing system, known as "views". In jEdit, the student can run one instance of TechWriter with multiple essays—or multiple copies of the same essay—at a time, each in their own view. When using TechWriter with a selected view, jEdit will only affect the text in that view, known as the "buffer". This allows the student to develop

```
<sentence number="0">
    <before>My brother is only five feet tall.</before>
    <after>My brother enjoys playing basketball, but he is only five
        feet tall.</after>
</sentence>
<sentence number="1">
    <before>If he were a foot taller, he would be a great basketball
        player.</before>
    <after>If he were a foot taller, he would be a great basketball
        player.</after>
</sentence>

<paragraph number="0">
    <before>My brother is only five feet tall.  If he were a foot
        taller, he would be a great basketball player.</before>
    <after>My brother enjoys playing basketball, but he is only five
        feet tall.  If he were a foot taller, he would be a great
        basketball player.</after>
</paragraph>
```

Figure II.1: XML for the before and after snapshots of the sentences and paragraphs of this (very short) document.

their own sort of versioning system for their work, or perhaps to compare two different essays side-by-side, without utilizing all of their computer's resources.

This sort of manual versioning system is not entirely necessary, however, because TechWriter supplies the student with its own. It has the ability to store a one-level history of the student's document in an XML-based format in two files, one for sentences and the other for paragraphs. Each sentence and paragraph is numbered for reference and labelled as either the previous version ("before") or the current version ("after", as in, after the changes were made to the previous version). An example of this is given in figure II.1. This sort of versioning system can help the student follow their progress; by seeing a prior iteration of an isolated sentence or paragraph, they can try to understand how these small portions relate to the whole document. They can also use this edit history to examine the mistakes they previously made and gain a better perspective on their errors. Similarly, TechWriter can use these histories for

automatic error correction. It can examine the way a sentence or paragraph *used* to look and, assuming it has since been corrected, can model its behavior from it. It is also possible for the student to tell TechWriter to "revert" back to a previous iteration of a sentence or paragraph, without affecting the entire document.

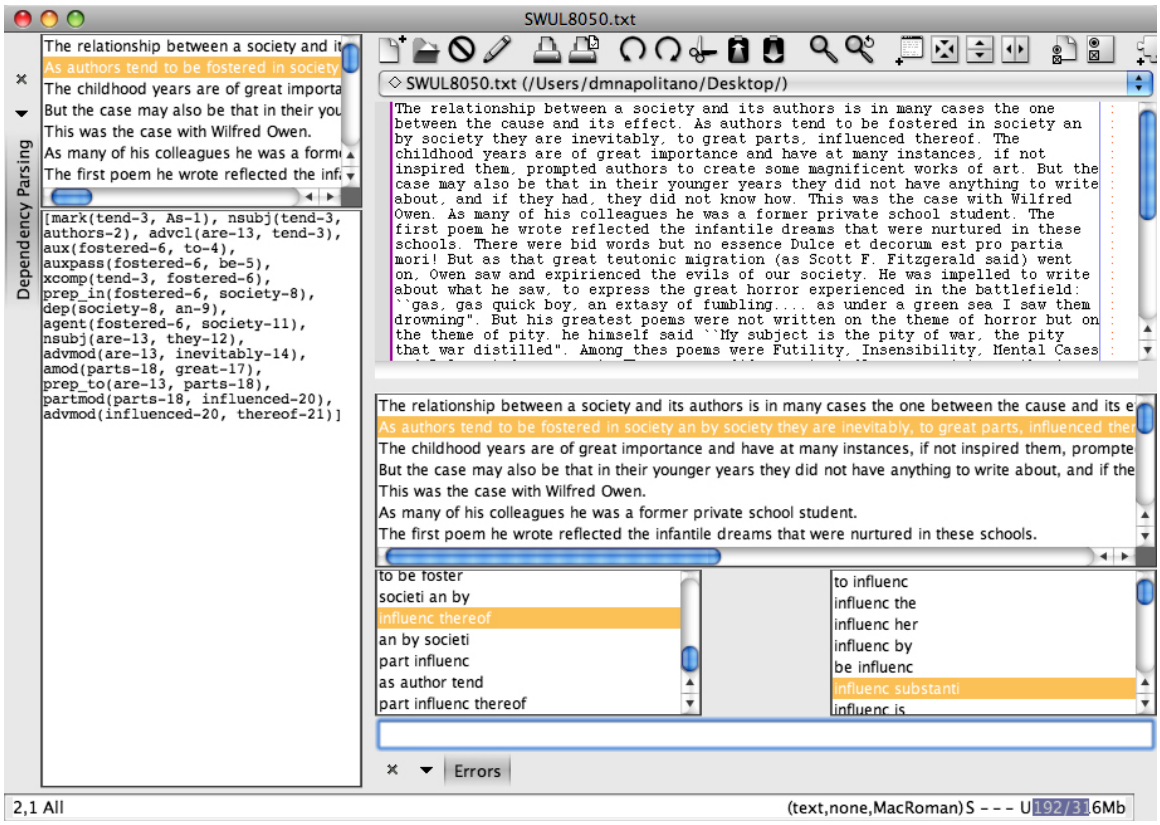

Figure II.2: A screenshot of the current version of TechWriter, showing the system's dependency parsing (left) and error correction (bottom) dialogs. TechWriter is working with the same essay provided in appendix A, which is loaded into jEdit and visible at the top, right-hand corner. The other two dialogs can be detached from this or attached to any side of this main editing window.

Our versioning system, and the capabilities it can afford our system, is augmented by many Natural Language Processing tools. The buffer from the current view in jEdit is imported into each of these tools each as separate instances for each view. The output is either presented as instructional material to the student—who can then

make changes to their document in the view and update this output accordingly—or it is used "behind the scenes" as input data to the system's automatic error detection, suggestion and correction technologies. All of our Natural Language Processing is done with OpenNLP [20] with the exception of dependency parsing, which is provided by the Stanford Parser [18]. In TechWriter, OpenNLP performs sentence detection, POS tagging, named entity recognition, sentence chunking and Treebank parsing, all with Maximum Entropy models built from the Wall Street Journal corpus. The Stanford Parser relies on rules to determine dependencies (see the introduction to chapter 2 for an example of a dependency parse), but before it does this, it uses a very similar parser to perform the initial parsing of sentences. Spell checking is also provided with the Jazzy spell checking API.[1] Figure II.2 demonstrates TechWriter's dependency parsing and error detection and correction features in dialogs that are "docked" or attached to their view.[2]

In the next section, we will further describe the error detection and correction suggestion components we have previously mentioned here. Before we can do that, however, we will discuss the data that TechWriter uses in order to provide these services to the student. Our system draws from corpora of both published, native American English and advanced learner English to make suggestions and detect errors. We use portions of the "plos", "oup" and "slate" subcorpora, containing technical articles, non-technical journal articles, and non-fiction writing, respectively, from the open part[3] of the American National Corpus (ANC) [13]. We used 252 texts from plos, 45 from oup (which is the entirety of these two subcorpora) and 129 from slate (out of 4531), for a total of 413 texts, which we then tagged for parts of speech and created bigrams, trigrams and four-grams of words and POS tags. We counted the frequency of each n-gram in the corpus as well as the relative frequency, which is defined as the total number of occurrences of this n-gram divided by the total number of n-grams that begin with the first word in this n-gram. Our data typically

[1] http://jazzy.sourceforge.net

[2] jEdit allows GUI windows to be designed as "dockables", which give them this ability to be attached to any side of a current view. "Docking" a window is entirely optional.

[3] The "Open American National Corpus" contains portions of the second release of the corpus, and is available for free from http://americannationalcorpus.org/OANC/index.html.

ngram, raw_freq, relative_freq
NNP NNP NNP, 7033, 0.185
NNP NNP VBZ, 910, 0.024
NNP VBZ IN, 257, 0.0070
VBZ IN DT, 636, 0.049
IN DT NNP, 2931, 0.036

Figure II.3: The first five lines, plus heading, of oanc_pos_Trigram.csv.

has the format given in figure II.3. These features are compared to those generated by TechWriter from the student's document, as we mentioned before. We will now elaborate on this in the following section.

II.2 Error Detection and Correction Suggestion

We will now discuss, in more detail, how the error detection and correction suggestion features of TechWriter work; a high-level description of these can be found in the Introduction. In order to explain how these two aspects work, we will use an essay from the ICLE, the entirety of which can be found in appendix A. The author of this essay was a native Swedish speaker.

Error detection currently operates at the sentence level. When a student asks to see the errors in their document, TechWriter first finds all of the sentences in the document, then separately tags them for parts-of-speech and stems them [22]. Then, for each sentence, it creates bigrams and trigrams[4] of both the sentence's stemmed words and POS tags. Each n-gram is compared against the proper data set from the ANC (as in, either bigrams or trigrams of words or POS tags) and every n-gram that exists in the student's document that was not found in the ANC is returned to the student. A list of these n-grams exists for each sentence where an error was found, so that the student may see their errors *per sentence*, in isolation.

For our sample essay, TechWriter has found an error in each sentence in the essay, and this is the list of error sentences it returns. Now, the student may select a sentence

[4]We will continually reference data of four-grams of stemmed words and POS tags in other sections. We do indeed have this data, but it has not yet been incorporated into TechWriter.

from this list and see a list of errors within it, in the form of bigrams and trigrams of stemmed words and POS tags from that sentence. Rather than focus on the entire essay, for the sake of brevity we will choose a small subset of sentences to demonstrate how detection and suggestion works.

When we select the following sentence from the list of error sentences:

(9) As authors tend to be fosted in society an by society they are inevitably, to great parts, influenced thereof.

we receive the following list of errors:

in societi an	foster in	thei ar inevit
JJ|NN As/IN JJ	to great part	JJ|VBN This/DT VBD
VBG|NN NN PRP	JJ|VBN This/DT	societi an
PRP .	JJ|NN As/IN	be foster in
inevit to great	an by	author tend to
author tend	foster in societi	be foster
great part influenc	by societi thei	to be foster
ar inevit to	societi an by	influenc thereof
an by societi	part influenc	as author tend
part influenc thereof		

When we select one of these errors, we will be presented with a list of possible corrections, pulled straight from the ANC in the same format as the error (i.e., for an error that is a trigram of stemmed words, the list of corrections will also contain trigrams of stemmed words). This list is generated in one of the following ways:

- If the selected error is a bigram *error*, with two unigrams *error.a* and *error.b* (where *error.a* and *error.b* are the first and second unigrams of the bigram, respectively), TechWriter will compare it to a bigram X from the ANC and add X to a list of suggestions if either $X.a$ equals *error.a*, $X.a$ equals *error.b*, $X.b$ equals *error.a* or $X.b$ equals *error.b*.

- If the selected error is a trigram *error*, TechWriter will compare it to a trigram Y from the ANC and add Y to a list of suggestions if *error* and Y have any

two unigrams in common. Trigrams that only have one unigram in common are not considered here because, if an error exists in any pair of two words in the trigram, that bigram itself will appear in the list of errors, and the student can view those suggestions separately.

Now let us choose an error from the list of errors in (9) and examine the suggested corrections. We first select a word bigram, "influenc thereof",[5] since it is an obvious mistake. For this bigram, TechWriter provides an extensive list of suggestions, so we have attempted to narrow down this list by setting a threshold. As we mentioned earlier, we have calculated the raw and relative frequencies of each n-gram in the corpus, so we can set a threshold such that no n-grams with either a raw or relative frequency below it will appear in the list of suggestions. We have tried setting thresholds on both the raw and relative frequencies, separately, to see which one gives better results. To attempt to determine a good threshold, let us select a particular bigram from our initial list of suggestions to use as a baseline. We would like to use "influenc substanti" (as in, "influenced substantially") in place of "influenc thereof", which has a raw frequency of 2 and a relative frequency of 0.012. Since we have decided that it is the "ideal" correction for our error, we wish for it to always show up in the list of suggestions for this error. We will also borrow advice from Genthial and Courtin [10] and favor the head of the bigram; so we will add a bit of weight to bigrams whose first unigram matches the error's first unigram, which is "influenc" in this case.

Our first attempt is with the relative frequencies, but because relative frequencies are floating-point values, it is difficult to use them in a boolean comparison (i.e. "Is this relative frequency greater than the threshold?"). Also, many relative frequencies of n-grams are the same, or almost equal to one another, while raw frequencies have a much wider range; this makes it difficult to find a good threshold on the relative frequencies. We then switched to a threshold on the raw frequency values, and used the raw frequency of "influenc substanti" as a baseline. This decreased the number of suggestions to a considerably smaller amount, but still somewhat too long; however, we run the risk of making the list too short to be useful for other sentences. Consider

[5]Recall that this bigram, as well as others that we will discuss in the future, were built from stemmed words.

the sentence:

(10) Now, the case is not always this.

We can see that this sentence should be written as, "Now, this is not always the case", and the trigram "now thi is" does exist in our data, but is not returned as a suggestion because it has a raw frequency of 1. In short, setting thresholds, as we have done, is a good way to shorten the list of results, but sometimes the list is made too narrow. Perhaps in the future we could move some of these additional matches to a "More Suggestions" dialog, so as to not limit the suggestions the student receives, but also to not overwhelm them with too many suggestions.

TechWriter is well on its way to detecting errors and providing useful suggestions for corrections to mistakes made by students. Aside from what we have already mentioned, we would like to provide useful feedback to the student and explain to them the error they have made, so that perhaps they could understand why their writing is wrong. We could certainly use POS tags for this task. We would also like to provide suggestions in many different word forms, if they are available for a word—we would like to show the student the different tenses, pluralization, and other varieties of a word and try to suggest the best fit for their context. The student should also be presented with what TechWriter perceives to be, generally, the best fit—there is no real reason a user would pick "now this is" to fix a mistake in (10), but, the reader will agree, it is the best choice given the context. We somehow need to tell TechWriter to favor certain suggestions, and relay this preference back to the student. In the next section, we will propose an idea as to how we believe TechWriter could do this, should the mistake be one that is repeated by the student.

II.3 Future Directions

We are interested in presenting to the student the best correction to an error and also providing some level of automatic error correction. As we mentioned in the beginning of this section, TechWriter stores a one-level history of the student's document, of both sentences and paragraphs. Here, TechWriter can use this to get a broad picture

of an error and its correction. For example, if the student often writes things similar to (10), perhaps writing "Now, the situation is not always this" in the same or a different document, TechWriter can look at (10) and the modifications made to it, and apply those modifications to this new sentence. These sentences could be matched by counting the number of words or POS tags they have in common.[6] Once the errors have been found and TechWriter has some history of those errors and the corrections made to them, we would like TechWriter to step in, at some point, and automatically apply these corrections.

Aside from performing this sort of rule-based pattern matching using edit histories, we also think it would be useful to keep track of which n-grams have been used to correct errors in n-grams, and to bias TechWriter towards offering these suggestions more often. Say, for example, the student wrote a sentence similar to (9), only with one error:

(11) Authors tend to be influenced thereof by the society that fosters them.

For the bigram "influenced thereof", the student can choose to correct it with the bigram "influenced substantially", and we can add an additional weight to this bigram. Later on, should the user make a similar mistake, say

(12) Art is often influenced moreover by technology.

TechWriter will place "influenced substantially" at the top of the list of suggestions and encourage the student to use it. Each time the bigram is used, its weight will increase, since this bigram represents the most common way that this student has tried to express a similar thought, but previously could not figure out how to phrase it correctly.

However, the student may feel that they have phrased their thought correctly, and cannot understand why they are in error. They would probably like some explanation as to why their writing contains an error, what the error means and why a particular suggestion for it is correct, as opposed to what they wrote themselves. Unfortunately,

[6] As far as the POS tags are concerned, this is probably more interesting and useful when the sentences have little or no words in common, but have things such as phrases of similar tenses.

generating feedback based on a particular, unique error can be difficult if we want it to be useful. We could simply just tell the student that they have poor style or that the way they have written something is not the way one would write it properly in English, as is the case with (9). Word choice is often a matter of style and a question of what is proper as far as the overall tone of one's writing, and TechWriter emphasizes a more formal style appropriate for academic or professional writing. Since this is the style the documents in our corpus employ, the student may find it beneficial to see an example sentence which demonstrates proper usage of one or more of the words in their error.

We can also help the student understand their error by not only presenting them with an example sentence written by someone else, but by correctly restructuring their sentence. Previous considerations for TechWriter included rule templates and used transformation-based error-driven learning to make corrections to the student's writing, should their error match a given rule [4]. Instead of using these rule templates to correct the student's writing, we can use these rule templates to build new sentences out of their erroneous one. The student can have the option of using the reconstructed sentence—or one reconstructed sentence out of several—in their writing, or they can just use the suggestion to understand how their sentence ought to be constructed. This is similar to what we already do in TechWriter, where we allow the student to choose a suggestion for an error n-gram and replace their error with it in their sentence. However, what we are proposing here is a restructuring of the *entire* sentence, not just a portion of it.

A technique that would aid both sentence restructuring and also our currently implemented technology is something we are calling "reverse stemming", which is simply building every possible word form out of a stemmed word. Recalling our example from section 3.1, the bigram "influenc substanti" in its stemmed form may not provide much assistance to the student; they may not know whether "influenced substantial", "influence substantially" or "influenced substantially" is the best fit for their sentence. It would be beneficial to the student to see all of these forms and to either pick the optimal one or have TechWriter do so for them, as part of TechWriter's automatic error correction.

Certain suggestions can be further biased, and errors can be explained by supplying the student with some information on writers of their same background. As we will describe in the next chapter, we have discovered that students of particular native languages make mistakes in the L2 that students of other native languages do not make, and there are some mistakes that are made quite frequently, which indicate some common problems among these students. This leads us to believe that there are many errors which are particular to students of a particular background and we can offer customized assistance to them based on this information. We can bias certain suggestions based not so much on the frequency of a related error made by the student, but also based on the frequency of these errors made by others with the same native language, with the rationale that, when students of a certain native language write a particular error-prone phrase, they most often mean to write a particular error-free one. We can demonstrate this to the student, and perhaps this will help them to understand their mistakes.

Chapter III

Errors: Unique to the User or Their Background?

Statistics show that Chinese L2 students have enough knowledge of the English language to be able to differentiate between English written by a native speaker and that written by another native speaker of Chinese [16]. This implies that there are some subtle differences in the way native Chinese speakers write, as opposed to native speakers of English, that are remarkably common among these L2 writers. We hypothesize that this is true for learners from any background; a native German speaker can probably determine when the author of an English text is German, and similarly for a Russian native speaker, a Japanese native speaker, and so on. We would like to both determine whether or not this is universally true, and why this is so: what mistakes does a non-native English student make that perhaps reveals their native language?

Chorodow et al. has found that, in the writing of 53 intermediate to advanced L2 students, about 29% of their grammar errors were with prepositions [7]. However, learners in the L1—that is, students who are native speakers of English—tend to not have any problem with prepositions, and instead, we imagine their errors will be more stylistic [9]. We can use this information to our benefit: if a student's writing contains numerous preposition errors, we can safely say that their native language is not English. We are then presented with the task of precisely determining what their

native language is.

From the ICLE, we took the Spanish (260 essays), Russian (279 essays) and German (450 essays) subcorpora and divided them each into training and testing sets of 90% for training, 10% for testing. We combined all of the testing essays into one file for each subcorpus and ran that file through OpenNLP's POS tagger. This gave us two files for each subcorpus, one with just the essays, and one with the essays after they were tagged for POS. We then created unigrams, bigrams, trigrams, four- and five-grams for each of these files, counting the raw and relative frequencies and storing them in the same format discussed in chapter 3. With this data, we ultimately performed two tasks: we attempted to determine the native language of the author of each of the testing examples with the given training data, and we also attempted to see which errors were common and unique to writers of a particular native language.

Our work in this chapter falls under the larger heading of "corpus similarity", which is the task of determining how similar two corpora are to one another, and why. Much work in this regard has been done by Kilgarriff, who, together with Rose has cited the use of statistical measures as a method of determining how similar two corpora, or even two individual documents, are to one another [15]. The best measure they found is the chi-squared test, in which they take the n most common words between the two corpora being compared, calculate the number of occurrences of each word in both corpora under the assumption that both corpora are random samples from the same population, and apply the chi-squared measure to this data.[1] This provides a metric of *how* similar two corpora are to one another, but one may ask *what* it is that makes these two corpora so similar. Kilgarriff and Rose point out that the answer to this question is complex and "multi-dimensional", since corpora themselves are multi-dimensional, and on some levels, will be both similar and dissimilar to one another [15]. We can thus look towards the work of Kenneth Church, who has found that within a document, certain uncommon words such as proper nouns are unlike lightening, which will never strike the same place twice. These words will, in fact, appear in the document at least once more, with a higher probability than a word that is generally used more often in English [8]. For our work, we rely on both proper

[1] We will do something very similar using Pearson's r coefficient in section 4.1.

nouns and phrases which contain nouns and express an idea; as we shall see, we have found that certain phrases, which are obviously linked to ideas, appear quite frequently in the writing of certain groups of non-native speakers. It can be said, therefore, that the way in which these ideas are expressed are not like lightening and are linked to the writing style of the author's native language. We have found that these words and phrases are the best criteria for determining the native language of a writer.

III.1 Determining the Native Language of a Writer

We combined all the training essays in a given subcorpus into one file and generated five files for each of these, separately containing unigrams, bigrams, trigrams, four- and five-grams, and their raw and relative frequencies. Then, for each testing essay, we generated the same n-grams, along with the frequencies, and compared this data to the training data for each native language and, for each testing essay–n-gram–native language pair, we created files with data along the lines of the following:

ngram, raw_freq_exp, relative_freq_exp, raw_freq_obs, relative_freq_obs
of the, 1739, 0.252, 1, 0.1
is not, 323, 0.072, 1, 0.2
base on, 54, 0.519, 1, 1.0
to the, 602, 0.106, 1, 0.1
of time, 48, 0.0070, 1, 0.1
it is, 638, 0.269, 1, 1.0

Figure III.1: A portion of RUMO1019_spanish_Bigram.csv, which contains bigrams that appear both in the essay in the file RUMO1019.txt and in the writing of someone whose native language is Spanish. The native language of the author of the essay is Russian.

The first line of the file, the header, indicates that each of the comma-separated values on every line are actually columns (which makes each line a row). In the header, the suffix _exp indicates that this is the value of the raw or relative frequency that we expect to see, or rather, the frequency of the n-gram in the training data (in

this case the file containing bigrams from the Spanish training data). The suffix _obs is what we observe, or specifically, the raw or relative frequency of the n-gram in the testing data (here, the essay in the file RUMO1019.txt).

All of the n-grams for one testing essay/training data pair were combined into one file, which gives the testing essay RUMO1019.txt four files: RUMO1019_english, RUMO1019_german, RUMO1019_russian and RUMO1019_spanish, all in the same format described above. The English training data comes from a portion of the Wall Street Journal corpus which contained 11,676 words. As we briefly mentioned above, each of these files contain n-grams that appear both in the testing essay and in the respective training data.

Then, for each of these four files that we had for each testing essay, we used their expected and observed relative frequency data and calculated Pearson's r coefficient. This data was stored in a spreadsheet in the following format:

File Name	Native Language	English (WSJ)	German	Russian	Spanish
GEAU1001	German	0.802915	0.7196509	0.7642184	0.743472
GEAU1002	German	0.8185328	0.6904784	0.6934538	0.7553417
GEAU1003	German	0.747711	0.6528048	0.6883687	0.6524887
GEAU1004	German	0.9312493	0.7083856	0.7469997	0.755657
GEAU1005	German	0.7688159	0.6900201	0.6867212	0.7299155
GEAU1006	German	0.8516055	0.6663766	0.6459643	0.6751704

Figure III.2: Our data for the first six testing essays whose author's native language is German. Here, the data used is n-grams of words, and the last four columns hold the values of the Pearson's r coefficient that we calculated with the four different sets of training data. The second column represents our prior knowledge; i.e., that the native language of the writer of the corresponding essay is German (which we know from the ICLE).

Now, to recap, we are trying to determine which of these native languages is that of the author of a given test essay, based solely on either the words or the POS tags used within that essay. We are using Pearson's r coefficient as a measurement of this, and our insight is that the larger the value of this coefficient, the larger the difference between that essay and essays written by speakers of that native language; in other words, if an essay x of native language X has a high r coefficient when compared

against training data of native language Y, the native language of the author of x is not Y. Thus, we took the minimum of the four values for each row in the spreadsheet (in the columns labeled "English (WSJ)", "German", "Russian" and "Spanish") and said that the predicted native language of the author was whichever r coefficient corresponded to that minimum. Below is the predicted native language for the six test essays given in the table in III.2, omitting the calculated r coefficients.

File Name	Native Language	Most Similar To
GEAU1001	German	German
GEAU1002	German	German
GEAU1003	German	Spanish
GEAU1004	German	German
GEAU1005	German	Russian
GEAU1006	German	Russian

Figure III.3: The predicted native language of six essays whose author's native language is German, based on n-grams of words.

We completed this test for all 99 test essays (45 from the ICLE native German subcorpus, 28 from the Russian subcorpus, and 26 from Spanish), for both n-grams of words and POS tags, separately. Since English was never predicted to be the native language, we decided to omit English from the tests with n-grams of POS tags; considering this, along with our results (presented in figure III.4), we have concluded that these tests would be better served with more training data for each language, including English. We were best able to predict German as a native language, and our worst predictions were with Spanish, between which there is a difference of 171 training essays. Also, German was the most frequently predicted native language across all of the predictions, accounting for 48.5% of them in the tests with word n-grams, and 39.4% of the predictions in the tests with n-grams of POS tags.[2]

[2]This result isn't as dramatic as the former one, however; the percentages of how many essays were predicted to come from an author of each native language is as follows: German with word n-grams, 48.5%, German with POS n-grams, 39.4%; Russian with word n-grams, 33.3%, Russian with POS n-grams, 34.3%; Spanish with word n-grams, 18.2%, Spanish with POS n-grams, 26.3%.

Native Language	N-Grams Used	Predicted Correctly	Number of Training Essays
German	Words	35/45 (77.8%)	405
	POS tags	17/45 (37.8%)	
Russian	Words	19/28 (67.9%)	251
	POS tags	9/28 (32.1%)	
Spanish	Words	14/26 (53.9%)	234
	POS tags	8/26 (30.8%)	

Overall Accuracy for Word N-Grams: 68/99 (69.9%)
Overall Accuracy for N-Grams of POS tags: 34/99 (34.3%)

Figure III.4: Results from our native language prediction tests using Pearson's r coefficient.

After looking at these results, we wondered if the amount of native German data we had was overshadowing the rest of the data, so we removed random essays from that training set so that we had the same number of essays as the training set with the smallest number of essays—in this case 234—and equivalently, we tested this with the smallest number of test essays, which was 26. We discovered that while our hypothesis was somewhat true, our overall accuracy and our ability to predict German as a native language suffered dramatically. In this test, German was no longer the most frequently predicted language, but was now Russian, which was predicted (either correctly or incorrectly) 50% of the time (40/80 essays); we imagine this is because of the 17 additional training essays in the Russian data set.

Native Language	Predicted Correctly	Number of Training Essays
German	12/26 (26.7%)	234
Russian	20/28 (71.4%)	251
Spanish	16/26 (61.5%)	234

Overall Accuracy: 48/80 (48.5%)

Figure III.5: Results from our native language prediction tests using Pearson's r coefficient and a smaller training set of essays whose author's native language was German. We only performed these tests with n-grams of words.

From the results of these tests we arrive at two conclusions: more training data improves accuracy, and it is the words of the essay that serve to distinguish its author's native language from that of another author's—the combinations of parts-of-speech

used do little to support this. All words in English have a part-of-speech associated with them, which means that there is a small number of categories into which an almost infinite number of words are placed; thus there are fewer unique combinations of POS tags than there are of words. One may now wonder exactly what words, or combinations of words, are particular to a non-native speaker of a certain language that helps to categorize it as coming from a native speaker of that language; this will be the focus of the next section.

III.2 What is Characteristic of Non-Native English?

Knowing which words and combinations of words are unique to writers of a particular background can help us improve our accuracy at predicting the student's native language, and also in pointing out their errors. For example, if writers from one language often write the same phrase incorrectly, we can show this to the student and also show them how these writers ought to be phrasing it.

For these tests, we used the exact same training and testing sets that we used in the experiments mentioned in section 4.1, with the addition of the entire portion of the ANC that we used in TechWriter. This gives us four datasets to test against: American English writing and German, Russian and Spanish L2 writing. We used the same n-gram data we generated earlier, of bigrams, trigrams and four-grams of both words and POS tags, separately. With this data, we wanted to determine two things:

1. What are the most frequently used word and part-of-speech combinations in the L2 writing of native German, Russian and Spanish students?

2. What errors are made by these students that do not appear in the writing of L2 students of any of the other backgrounds; i.e., what errors would a native German speaker make that a native Russian speaker would not? Also, what errors are made by writers in the L2 that are particular to their native background?

Before we can answer these questions, we should briefly return to our discussion discussion about the problems with POS taggers. Most POS tagging software tends to rely on a window of context on either side of the word it is currently trying to tag, and we have reason to believe our tagger also works in this way. As a result, our tagger will often wrongly tag the current word if it cannot determine what the word's tag should be based on its model or based on the tags surrounding it, especially if those tags themselves are wrong.[3] This results in curious tag combinations such as the four-gram : `` DT JJ found in Spanish L2 data as part of the following fragment:

(13) At/IN the/DT end/NN we/PRP can/MD see/VB how/WRB Mosca/NNP cheats/: Volpone./'' The/DT characters/JJ play/NN with/IN double/JJ faces./JJ|NN

When isolated away from the rest of the essay and put into TechWriter, the first sentence in (13) is tagged slightly differently, yet still wrong; "cheats" is tagged as a plural noun (NNS) when it should be a third-person singular verb (VBZ). The most likely cause of this it that the tagger cannot determine what "Mosca" is; it can see that it is a noun, but it cannot tell that it is a noun that represents something capable of performing an action, as we can infer from the context. The unfortunate effect of this is that the results of our tests with POS tags are somewhat skewed. Many incorrectly-tagged sentences or phrases do contain an error, and an error in their POS tags would properly indicate this; however, this is not automatically true for all sentences.

Keeping that in mind, we will now present the most frequent n-grams in our training data from each of the three subcorpora we used in figures III.6 and III.7. One will notice that, for POS tags, the results are almost identical. We thought perhaps that the frequency of how often these n-grams appeared would help to differentiate the writing of speakers of one native language from another, but the raw frequencies and the combinations of tags are so similar to one another that the impact of these n-grams cannot be too significant. Thus we have highlighted results that we feel may help to identify an author's native language. While these results are almost identical,

[3]Recall our discussion of this and related problems in chapter 2, as presented by Thurmair [24].

the order in which the n-grams appear in each list (for the most common bigrams, trigrams and four-grams) differs slightly for each language. There is also a large gap between the third and the fourth most frequently found bigram in the Spanish data, and the bigram TO VB shows up far less in Spanish than in the other two languages. Despite the similarities in the results, we feel that this data, along with the *unique* n-grams in each training set may have influenced the results we got in the tests performed in section 4.1.

Bigrams					
German		Russian		Spanish	
NN IN	11725	NN IN	12895	NN IN	11724
DT NN	11433	DT NN	12227	DT NN	11646
IN DT	9844	IN DT	9917	IN DT	10160
JJ NN	8307	JJ NN	9160	JJ NN	**7270**
DT JJ	5729	DT JJ	5686	DT JJ	5456
Trigrams					
German		Russian		Spanish	
IN DT NN	5112	DT NN IN	5496	DT NN IN	5445
DT NN IN	4730	IN DT NN	5172	IN DT NN	5394
NN IN DT	4131	NN IN DT	4412	NN IN DT	4690
DT JJ NN	3632	DT JJ NN	3687	DT JJ NN	3555
JJ NN IN	3091	JJ NN IN	3315	JJ NN IN	2919
Four-Grams					
German		Russian		Spanish	
NN IN DT NN	2092	**IN DT NN IN**	2263	NN IN DT NN	2489
IN DT NN IN	1982	**NN IN DT NN**	2197	IN DT NN IN	2399
DT NN IN DT	1576	DT NN IN DT	1766	DT NN IN DT	2086
DT JJ NN IN	1493	DT JJ NN IN	1566	**IN DT JJ NN**	1552
IN DT JJ NN	1441	IN DT JJ NN	1517	**DT JJ NN IN**	1552

Figure III.6: The most frequently occurring combinations (n-grams) of POS tags in our testing data.

However, as we also discussed in section 4.1, we believe that the words used in a student's writing will be a better indicator of their native language than POS tags. Regardless of how many unique n-grams of POS tags our data may have, the n-grams

still consist of the same POS tags, and there will be considerable overlap; with n-grams of words, this is significantly less likely to occur. We can see this from figure III.7, which lists the most frequent word n-grams from our training data, regardless of whether or not they contain an error.

Bigrams					
German		Russian		Spanish	
of the	1204	of the	1432	of the	1739
in the	1000	in the	1020	in the	1057
it is	720	it is	991	it is	638
to be	615	to the	589	to the	602
to the	509	to be	562	is the	509
on the	475	on the	438	to be	424
Trigrams					
German		Russian		Spanish	
in order to	122	a lot of	206	in order to	216
seem to be	104	on of the	120	a lot of	142
a lot of	102	dream and imagin	111	on of the	124
the fact that	99	there is no	101	of the plai	105
it is a	81	i think that	97	on the other	99
on of the	80	point of view	95	the other hand	99
Four-Grams					
German		Russian		Spanish	
on the other hand	56	i would like to	71	on the other hand	90
at the same time	40	at the same time	62	the end of the	49
in front of the	31	the root of all	50	at the same time	48
all over the world	26	root of all evil	48	on of the most	48
i would like to	25	on the other hand	43	at the end of	45
on the on hand	21	ar a lot of	42	the import of be	39

Figure III.7: The most frequently occurring combinations (n-grams) of stemmed words in our testing data.

As we can see, there is almost no variation in the bigrams across these three datasets aside from frequency counts, and we also see some of the same trigrams and four-grams in each set. Even from this small subset of data we can see distinctions in the writing of students from these three backgrounds; despite the fact that all

the students were given similar essay topics,[4] there is considerable difference in word choice and ideas among them, which seem to be common to these writers as a whole. For example, Russian students seem to be quite interested in abstract, philosophical concepts such as dreams and "the root of all evil".

One thing we cannot tell from this list, however, is whether or not Russian students are the *only* group writing about "the root of all evil". Therefore, our next test was to determine what word n-grams appear in the writing of each of these groups, but not in any of the others; i.e., what phrases appear in the writing of Russian L2 students but not in that of German or Spanish L2 students. We feel that these differences are key in identifying the background of a student, given a sample of their writing.

Our method for determining this was quite simple: we loaded the n-grams for a set of training data for language X and the n-grams for a test essay y from language Y into memory and for each n-gram that existed in both X and y, we removed it from X, and if it appeared only in y, we added it to X; this gave us a list of n-grams that were uncommon among X and y (or, to follow the wording used in figure III.8, a list of n-grams found in L2 writing from students of the language corresponding to y that were not found in L2 writing by students from the language corresponding to y). The lists we generated for each y were then combined into a list for Y, with their relative and raw frequencies updated across the entire set.

Notice there is a clear difference in both the ideas that are expressed within the writing from students from these backgrounds and also in the way they are expressed. It is more likely that a Spanish writer in the L2 will say "because it" than "because of", a phrase which is highly probable for a German writer. This is valuable data which can assist us in determining the native language of a student.

Now that we can identify the native language of the student, it would be extremely helpful to go one step farther and identify the errors that are particular to the native language of the student. Similar to our work in chapter 3, we operate under the assumption that any n-gram that does not appear in the ANC must be an error. We also used an identical method to the one we used to find the uncommon n-grams

[4]Typically, ICLE data is compiled by teachers who assign essays on one or several topics specified by the Centre for English Corpus Linguistics at the Université catholique de Louvain, which compiles the corpus [12].

Bigram	Found in L2 from...	...but not	Raw Frequency
prison system	Russian	German	22
becaus it	Spanish	German	15
becaus of	German	Russian	12
becaus it	Spanish	Russian	15
becaus of	German	Spanish	12
profession soldier	Russian	Spanish	10
Trigram	**Found in L2 from...**	**...but not**	**Raw Frequency**
the prison system	Russian	German	11
to the caus	Spanish	German	6
a speed limit	German	Russian	9
to the caus	Spanish	Russian	6
a speed limit	German	Spanish	9
of profession soldier	Russian	Spanish	5
Four-Gram	**Found in L2 from...**	**...but not**	**Raw Frequency**
entir of profession soldier	Russian	German	4
to the caus of	Spanish	German	6
car ought to be	German	Russian	8
to the caus of	Spanish	Russian	6
car ought to be	German	Spanish	8
entir of profession soldier	Russian	Spanish	4

Figure III.8: The most common bigrams, trigrams and four-grams found in the writing of German, Russian and Spanish students that are exclusive to students from those native languages.

among the training sets, except rather than use training data for every language, we only used data from our subset of the ANC in each comparison. In figure III.9, we present n-grams that do not appear in the ANC but appear in our training data with the highest raw frequency for each set.

The most interesting observation about these results is that none of the n-grams seem to contain an error. We imagine that this will be the most common situation since our software is geared towards, and the ICLE contains samples from, advanced learners of English. However, while they may appear correct when isolated, these n-grams may represent poor phrases within their entire sentence, particularly from a stylistic perspective. A search through our German L2 training data with the German

Bigram	Language	Raw Frequency
by car	German	15
prison system	Russian	22
becaus it	Spanish	15
Trigram	**Language**	**Raw Frequency**
to be ban	German	9
the prison system	Russian	11
to the caus	Spanish	6
Four-Gram	**Language**	**Raw Frequency**
car ought to be	German	8
serv in the armi	Russian	4
to the caus of	Spanish	6

Figure III.9: The bigrams, trigrams and four-grams that appear most often in our L2 test data that do not appear in our English data (ANC).

bigram "by car" finds the following sentence where the phrase is used in an awkward way:

(14) Another cousin of mine was innocently killed in a car accident, just because two young fellows had played cats and dogs by car.

At least in American English, the correct way to phrase that last portion of the sentence would be "two young fellows were playing Cats and Dogs with their cars."

Knowing about errors that are common across students from the same native language helps us to correct their errors. In a future experiment, we would like to determine what these students are most often trying to express when they make these errors so that we can offer this to students as a heavily-weighted correction to the particular error. However, our data shows n-grams that are not found in our English corpus and are *common* mistakes among students from a specific background; we cannot guarantee that all errors made by a student will fall under this category. We now wonder how many errors are idiosyncratic, or particular to the student, rather than to their background. Across bigrams, trigrams and four-grams, the average number of times each unique n-gram appears in any of our training sets is 2; thus the chance that a phrase is simply particular to the student who wrote it is extremely

high.

When we ran our tests earlier, comparing n-grams in training set X to test essays from language Y, we also compared test essays x from language X against the training data for X and similarly combined the results. This gave us a list of all n-grams from these essays that do not appear in the L2 training data for the essay's respective native language. We then wondered which of these n-grams were errors or were not found in the ANC. We went through this list of idiosyncratic n-grams and kept the ones that did not appear in the ANC; the numerical results of this are presented in figure III.10.

Bigrams			
Language	Total	Total Errors	Percent of Total
German	4435	4293	96.8%
Russian	1657	1551	93.6%
Spanish	2794	2602	93.1%
Trigrams			
Language	Total	Total Errors	Percent of Total
German	11470	11404	99.4%
Russian	5110	5081	99.4%
Spanish	7943	7852	98.9%
Four-Grams			
Language	Total	Total Errors	Percent of Total
German	14672	14660	99.9%
Russian	7027	7022	99.9%
Spanish	10777	10758	99.8%

Figure III.10: The number of unique n-grams that were found in the testing essays for their respective training data, along with the number of unique n-grams from this set that were not found in the ANC.

It is interesting to note that almost all of these so-called idiosyncratic phrases do not appear within the ANC, and would be considered errors by TechWriter, which may or may not be accurate.

While error detection and correction will work exactly the same way for these idiosyncratic errors as it will for errors that are common among students from the same background, knowing the student's background can help to put their errors in

perspective, making them feel more comfortable about their mistakes and helping them to understand why they have made them. Similarly, this information can help improve our software if we are able to determine a common, most probable correction to these common mistakes; we can then automatically apply these corrections or push them to the top of the list of suggestions for the student.

Chapter IV

Results and Conclusions

The English language happens to be quite complex, and the ability to effectively communicate with it is becoming increasingly important, from a global standpoint. Writing in English can be a serious problem, even for an advanced student of the language, regardless of how long they have been working with the language. With TechWriter, we have attempted to address this with a *personalized* approach that both *assists* the student with their writing and *improves* on their skills at the same time. The system aims to be personalized by tailoring itself to the student, tailoring its interactions with the student as it determines more information about them, such as their native language. TechWriter assists the student by removing some of the more redundant work from essay writing, such as correcting the same type of error repeatedly; however, the system will only automatically correct these errors after the student has made an initial correction themselves, demonstrating to TechWriter how to fix these mistakes. In this way, the system improves the student's writing skills by forcing them to think about their mistakes; the student must choose the best way to fix an error, using the tools provided by TechWriter.

To help the student learn, TechWriter provides a wide range of data on their writing provided by various NLP tools, as well as a large list of suggestions for ways they can correct their errors—which TechWriter points out to them—straight from actual English writing. In the future, the system will be able to reconstruct the student's error-laiden sentence and offer suggestions for corrections based on corrections they

themselves have made in the past and also based on data from other writers of their background.

In chapter 4, we have adequately demonstrated that it is entirely possible to determine the native language of an L2 writer, based almost entirely on the words within their writing. As we have shown from our experiments, more data would be helpful to improve our accuracy, but we are limited here by what is provided in the ICLE. Even in TechWriter, many phrases are marked as errors when perhaps they should not be, based on the fact that the phrase does not appear in our English corpus. While more data would also help in this case, it is important to recall that TechWriter is geared towards improving formal, academic, professional writing, and much of the writing samples we have looked at here (from the ICLE) are primarily casual, and should not be.

A system such as ours is capable of incorporating many diverse technologies to accomplish its goals, and as a result, can be pulled in many different directions. We have discussed many of these tools and systems in chapter 2, most of which have similar goals to TechWriter; we have also mentioned some of the issues one can encounter when developing such tools, primarily in regards to parsing and tagging L2 writing. We believe that we have demonstrated possible solutions to this problem and how we can avoid them in TechWriter, when necessary.

There is plenty of room for TechWriter to grow and perhaps, in the future, the software can be incorporated into a curriculum at the high school or college level, particularly for students in disciplines which are math- or science-intensive.

Bibliography

[1] Allen, C. "Class of 2008 Steps into Good Job Market". *National Association of Colleges and Employers*, 2008. Retrieved May 10, 2008, from `http://www.jobweb.com/Jobs/market.aspx?id=1219`.

[2] Aspray W., F. Mayadas, and M. Vardi, editors. "Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force". *Association for Computing Machinery*, 2006. Retrieved May 6, 2008, from `http://www.acm.org/globalizationreport`.

[3] Atwell, E. S. "How to Detect Grammatical Errors in a Text Without Parsing It". *Proceedings of EACL 87*, Copenhagen, 1987.

[4] Brill, E. "Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging". *Computational Linguistics*, 1995.

[5] Burnstein, J., M. Chodorow and C. Leacock. "Automated Essay Evaluation: The *Criterion* Online Writing Service". *AI Magazine*, 2004.

[6] Chodorow, M. and C. Leacock. "An Unsupervised Method for Detecting Grammatical Errors". *Proceedings of NAACL 00*, Seattle, 2000.

[7] Chodorow, M., J. R. Tetreault and N. Han. "Detection of Grammatical Errors Involving Prepositions". *Proceedings of the 4th ACL-SIGSEM Workshop on Prepositions*, Prague, 2007.

[8] Church, K. W. "Empirical Estimates of Adaptation: The Chance of Two Noriegas is Closer to $p/2$ than p^2". *Proceedings of the 18th Conference on Computational Linguistics*, Saarbrücken, 2000.

[9] Gamon, M., J. Gao, C. Brockett, A. Klementiev, W. B. Dolan, D. Belenko and L. Vanderwende. "Using Contextual Speller Techniques and Language Modeling for ESL Error Correction". *Proceedings of IJCNLP*, Hyderabad, 2008.

[10] Genthial, D. and J. Courtin. "From Detection/Correction to Computer Aided Writing". *Proceedings of COLING-92*, Nantes, 1992.

[11] Gilmore, M. and M. Hall. "Intl. Grad Student Rates Climb Back Up [Electronic version]". *The Cavalier Daily*, 2007. Retrieved April 15, 2008, from `http://www.cavalierdaily.com/CVArticle.asp?ID=29242&pid=1544`

[12] Granger, S. "The International Corpus of Learner English: A New Resource for Foreign Language Learning and Teaching and Second Language Acquisition Research". *TESOL Quarterly*, 2003.[1]

[13] Ide, N. and C. Macleod. "The American National Corpus: A Standardized Resource of American English". *Proceedings of Corpus Linguistics*, Lancaster UK, 2001.

[14] "jEdit - Programmer's Text Editor - Overview". *jEdit*, 2008. Retrieved May 12, 2008 from `http://www.jedit.org/`.

[15] Kilgarriff, A. and T. Rose. "Measures for Corpus Similarity and Homogeneity". *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing*, Granada, 1998.

[16] Liu, T., M. Zhao, J. Gao, E. Xun and C. Huang. "PENS: A Machine-aided English Writing System for Chinese Users". *Proceedings of the 38th Annual Meeting on Association For Computational Linguistics*, Hong Kong, 2000.

[17] Macdonald, N. H., L. T. Frase, P. S. Gingrich and S. A. Keenan. "The Writer's Workbench: Computer Aids for Text Analysis". *IEEE Transactions on Communications*, 1982.

[18] de Marneffe, MC., B. MacCartney and C. D. Manning. "Generating Typed Dependency Parses from Phrase Structure Parses". *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, Genoa, 2006.

[1]For further, up-to-date information on the ICLE, one can consult the website at http://cecl.fltr.ucl.ac.be/Cecl-Projects/Icle/icle.htm.

[19] Moré, J., S. Climent and A. Oliver. "A Grammar and Style Checker Based on Internet Searches". *Proceedings of the LREC2004*, Lisbon, 2004.

[20] Morton, T. S. "The OpenNLP Homepage". *OpenNLP*, 2006. Retrieved May 12, 2008 from `http://opennlp.sourceforge.net/`.

[21] Nagata, R., A. Kawai, K. Morihiro and N. Isu. "A Feedback-Augmented Method for Detecting Errors in the Writing of Learners of English". *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, 2006.

[22] Porter, M. F. "An Algorithm for Suffix Stripping". *Program: Electronic Library and Information Systems*, 1980.

[23] Schneider, D. and K. F. McCoy. "Recognizing Syntactic Errors in the Writing of Second Language Learners". *Proceedings of ACL 1998*, Montreal, 1998.

[24] Thurmair, G. "Parsing for Grammar and Style Checking". *Proceedings of COLING-90*, Helsinki, 1990.

[25] Zweben, S. "2003-2004 Taulbee Survey: Record Ph.D. Production on the Horizon; Undergraduate Enrollments Continue in Decline". *Computing Research Association*, 2005. Retrieved May 6, 2008, from `http://www.cra.org/CRN/articles/may05/taulbee.html`.

Appendix A

Sample Essay from the ICLE: SWUL8050.txt

<ICLE-SW-LND-0050.8>

The relationship between a society and its authors is in many cases the one between the cause and its effect. As authors tend to be fostered in society an by society they are inevitably, to great parts, influenced thereof. The childhood years are of great importance and have at many instances, if not inspired them, prompted authors to create some magnificent works of art. But the case may also be that in their younger years they did not have anything to write about, and if they had, they did not know how. This was the case with Wilfred Owen. As many of his colleagues he was a former private school student. The first poem he wrote reflected the infantile dreams that were nurtured in these schools. There were bid words but no essence Dulce et decorum est pro partia mori! But as that great teutonic migration (as Scott F. Fitzgerald said) went on, Owen saw and expirienced the evils of our society. He was impelled to write about what he saw, to express the great horror experienced in the battlefield: "gas, gas quick boy, an extasy of fumbling.... as under a green sea I saw them drowning". But his greatest poems were not written on the theme of horror but on the theme of pity. he himself said "My subject is the pity of war, the pity that war distilled". Among thes poems were Futility, Insensibility, Mental Cases and Dulce et decorum est. These poems did more to influence society on the issue of war than all the newspapers did during the First World War. In this way we see how society influenced (if not killed him in the case of Owen) the poet and how he in his turn influenced society by his ideas, expressed in the form of poetry.

Now, the case is not always this. In what way can we say that Oscar Wilde was influenced by society when he wrote the Happy Prince for his two sons? This sad fairy-tale cannot be but an offspring of his mind, a creation by his fantasy, a tale told to amuse children with no social implications. One may of course argue that since he wrote this tale for his both sons, and since he incorporated a theme concerned with moral issues he might have intended this tale as mean to educate his sons in Christian

values. However the case is, we can conclude that society more often than not, the authors, by their works, influence society, albeit unitentionally.