UNIVERSITY OF CALIFORNIA, SAN DIEGO

Whole Sentence Spelling and Grammar Correction Using a Noisy Channel Model

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Yonghahk Albert Park

Committee in charge:

Professor Roger Levy, Chair
Professor Garrison W. Cottrell, Co-Chair
Professor Charles Elkan
Professor Lawrence K. Saul
Professor Nuno Vasconcelos

2013

The Dissertation of Yonghahk Albert Park is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California, San Diego

2013

DEDICATION

To my dear parents and loving wife

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my Ph.D. advisor, Professor Roger Levy, who has been a fantastic advisor throughout this journey. He has been the most supportive of my work, providing me with excellent guidance and support, and encouragement in times of need. In addition to all this, he has also been very patient and understanding, more than anyone I have known. Roger has made my journey an enjoyable one, for which I have the utmost gratitude.

I would also like to thank Randy West, who has been a great office mate and contributor to this work. He has provided an extra set of eyes on this work, as well as for my code, and has helped me implement more than I could have done on my own. Working with Randy was not only enjoyable, but also came with an extra boost of enthusiasm toward the project, constantly helping it move forward.

I would also like to thank the UCSD Computational Psycholinguistics Lab members for their ongoing help and support. Though the internet seems to encompass so much human knowledge, the CPL lab members have often been the best access points, providing me with higher quality information and insights which can only be obtained by means through time and experience.

Thanks are also due to Markus Dreyer and Jason Eisner for supplying unpublished code for using the expectation semiring with OpenFST, which helped move this project forward in a timely manner. Also, I would like to thank Natalie Katz for reading through and marking up thousands of manual corrections, and the San Diego Supercomputer Center for setting us up to use their system, and supplying us with resources for running our experiments.

Finally, I wish to thank my parents and my wife for their continuous support and understanding. They have always been there when I have needed them most, always offering support and advice.

Chapters 1, 2, 3, 4, 5, 6, and 7 are, in part, a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

VITA

1994-1997	Seoul Science High School, Seoul, Korea National University, Seoul, Korea
1997-2005	B. S. in Computer Engineering *magna cum laude*, Seoul
2000-2004	Research Engineer, Penta Security Systems, Seoul, Korea
2005-2013	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

Fei Sha, Y. Albert Park, and Lawrence K. Saul, "Multiplicative updates for L1-regularized linear and logistic regression", *Proceedings of the Seventh Symposium on Intelligent Data Analysis (IDA)*, pp. 13-24, 2007

Y. Albert Park, Roger Levy, "Minimal-length linearizations for mildly context-sensitive dependency trees", *Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT)*, pp. 335-343, 2009

Randy West, Y. Albert Park and Roger Levy, "Bilingual Random Walk Models for Automated Grammar Correction of ESL Author-Produced Text", *Proceedings of the ACL Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, 2011

Y. Albert Park, Roger Levy, "Automated Whole Sentence Grammar Correction Using a Noisy Channel Model", *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 934-944, 2011

ABSTRACT OF THE DISSERTATION

Whole Sentence Spelling and Grammar Correction Using a Noisy Channel Model

by

Yonghahk Albert Park

Doctor of Philosophy in Computer Science

University of California, San Diego, 2013

Professor Roger Levy, Chair
Professor Garrison W. Cottrell, Co-Chair

Automated grammar correction techniques have seen improvement over the years, but there is still much room for increased performance. Current correction techniques mainly focus on identifying and correcting a specific type of error, such as verb form misuse or preposition misuse, which restricts the corrections to a limited scope. We introduce a novel technique, based on a noisy channel model, which can utilize the whole sentence context to determine proper corrections. We show how to use the EM algorithm to learn the parameters of the noise model, using only a data set of erroneous sentences, given the proper language model. This frees us from the burden of acquiring a large corpora of corrected sentences. We

also present a cheap and efficient way to provide automated evaluation results for grammar corrections by using BLEU and METEOR, in contrast to the commonly used manual evaluation.

Chapter 1

Introduction

1.1 The problem of editing written text

Humans live in a noisy environment. Our senses constantly receive input, but the input is not always exactly what we wish to obtain. This is because the input is often distorted by some type of noise. For example, when we are trying to have a conversation with another person, we not only hear their voice, but also hear the other ambient sounds in the environment, such as the sound of a car honking in the distance, or the hum of a copier churning out documents. Another source of distortion could be the speaker himself, who may misuse words or make grammatical mistakes. Despite living in such a noisy environment, humans seem to be very good at figuring out the underlying source information of the noisy input. In many cases this is done quite effortlessly, such as filtering out ambient noise when talking to a friend, or reading highway signs which may be partly obscured by other traffic.

Another example of this type of process is the editing of written text. The process of editing written text is performed by humans on a daily basis. In the case of minor spelling errors, humans often do not even notice the existence of the misspellings when reading text, and have no trouble figuring out the underlying intent of the writer. When asked to edit a sentence with various errors, humans can often read the text and correct the errors without much effort. For example, given the sentence "I would like go home", native English speakers can almost

automatically fix the sentence to "I would like to go home". Unfortunately, current computer systems are still far from human capabilities and performance when it comes to the task of correcting erroneous text input. In fact, just identifying erroneous sentences in itself is currently a difficult task for computer systems.

This work investigates the problem of automated grammar and spelling correction of human generated written sentences. The focus is on fixing a given sentence in its entirety. Although this idea could be applied to any human-produced utterance, in practice we focus on correcting the grammatical errors in non-native written English. Since humans are currently much more capable than computers at this task, a little insight into how humans actually figure out the correct editing may be quite informative. Unfortunately, our current knowledge of the process used by humans to derive a valid correction of an erroneous sentence is very limited. In this work, we create a model of language production which can be used to find a sentence writer's original intent. We approach the problem by modeling various types of human errors using a noisy channel (Shannon, 1948). In our overall model, correct sentences are produced by a predefined probabilistic generative language model, and then lesioned (or distorted) by the noise model. The language model parameters are learned from a monolingual corpus of "correct" English, and we learn the noise model parameters from raw errorful sentences using an expectation-maximization (EM) approach (Dempster et al., 1977; Wu, 1983). By making use of EM, we eliminate the need for costly annotation of sentences during training of the noise model. Our model allows us to deduce the original intended sentence by looking for the highest probability parses over the entire sentence, which leads to automated whole sentence spelling and grammar correction based on contextual information.

1.2 Background

Much of the previous work in the domain of automated grammar correction has focused on identifying grammatical errors. Chodorow and Leacock (2000) took an unsupervised approach to identifying grammatical errors by looking for

contextual cues in a ±2 word window around a target word. To identify errors, they searched for cues which did not appear in the correct usage of words. Eeg-olofsson and Knutsson (2003) used rule-based methods to approach the problem of discovering preposition and determiner errors of L2 writers. In recent years, various machine learning techniques have been applied to the problem. Many proposed techniques use classifier-based methods using local context features, including words and part of speech tags. Izumi et al. (2003) and Izumi et al. (2004b) use a maximum entropy classifier for detecting 13 types of grammatical and lexical errors on a corpus of spoken English. Han et al. (2006) train a maximum entropy classifier to select between 4 articles. Maximum entropy models have also been proposed for use on detecting preposition errors in ESL writing by Tetreault and Chodorow (2008) and De Felice and Pulman (2008). Gamon et al. (2008) train a decision tree classifier which is used in conjunction with a 5-gram language model to detect and correct preposition and determiner errors in a corpus of Chinese ESL student essays.

While most of these methods have been proposed primarily to *identify* grammatical errors, some of the classifier-based methods could actually be used to determine suggestions for corrections. This can be done by using the scores or probabilities from the classifiers for other possible words. To do this, for each word which is determined to have a possibility of being incorrect, we need a set of words which could be plausible replacements. This set is often referred to as the *confusion set*. The highest scoring word in the confusion set may be used to replace words which have been identified as errors. Rozovskaya and Roth (2011) use this method to evaluate 4 different classifier algorithms for correcting preposition errors and article errors, and compare their performance on an annotated ESL corpus. While this is a plausible approach for grammar correction, there is one fundamental difference between this approach and the way humans edit. The output scores of classifiers do not take into account that the observed erroneous word was actually observed. In other words, the observed word is treated no differently from any other word in the confusion set. This changes the task of editing into a fill-in-the-blank selection task. In contrast, editing takes the writer's erroneous

word with more consideration, which often encompasses information necessary to correctly deduce the writer's intent. For example, let us examine the two sentences 'We plan to go on vacation about July' and 'We plan to go on vacation beyond July'. Taking the full sentence into account, including the erroneous word, we can deduce that the former sentence should probably be corrected to 'We plan to go on vacation around July', whereas the latter should probably be corrected to 'We plan to go on vacation after July'. While the two sentences are different, using the classifier method to find the correct word would change both sentences into the same problem, looking for the best scoring word to fill in the blank in the sentence 'We plan to go on vacation (blank) July'. Thus we can see that making use of the actual erroneous word gives us insight into what the intent of the actual writer is, and thus gives us valuable information as to what the correction should be.

Generation-based approaches to grammar correction have also been taken, such as Lee and Seneff (2006), where sentences are paraphrased into an over-generated word lattice, and then parsed to select the best rephrasing. As with the previously mentioned approaches, these approaches often have the disadvantage of ignoring the writer's selected word when used for error correction instead of just error detection.

Other work which relates to automated grammar correction has been done in the field of machine translation. Machine translation systems often generate output which is grammatically incorrect, and automated post-editing systems have been created to address this problem. For instance, when translating Japanese to English, the output sentence often needs to be edited to include the correct articles, since the Japanese language does not contain articles. For example, if the translation output of a Japanese to English MT system was 'Girl went home after brief walk in park.', this would need to be edited to 'The girl went home after a brief walk in the park.' in the post-editing stage. Early works, such as Murata and Nagao (1993), Bond et al. (1995), Bond and Ikehara (1996) and Heine (1998) use heuristic rules in combination with the syntactic and semantic information from the Japanese source to determine the noun phrase properties, which can then be used to determine the correct article for the English translation. Knight

and Chander (1994) also address the problem of selecting the correct article for MT systems, but instead of using hand-crafted rules, they focus on automatically generating the rules by training a decision-tree builder using lexical features, part of speech information, and other subcategory information. These types of systems could also be used to facilitate grammar correction.

While grammar correction can be used on the output of MT systems, note that the task of grammar correction itself can also be thought of as a machine translation task, where we are trying to 'translate' a sentence from an 'incorrect grammar' language to a 'correct grammar' language. Under this idea, the use of statistical machine translation techniques to correct grammatical errors has also been explored. Brockett et al. (2006) use phrasal SMT techniques to identify and correct mass noun errors of ESL students. Désilets and Hermet (2009) use a round-trip translation from L2 to L1 and back to L2 to correct errors using an SMT system, focusing on errors which link back to the writer's native language.

Despite the underlying commonality between the tasks of machine translation and grammar correction, there is a practical difference in that the field of grammar correction suffers from a lack of good quality parallel corpora. While machine translation has taken advantage of the plethora of translated documents and books, from which various corpora have been built, the field of grammar correction does not have this luxury. Annotated corpora of grammatical errors do exist, such as the NICT Japanese Learner of English corpus and the Chinese Learner English Corpus (Shichun and Huizhong, 2003), but the lack of definitive corpora often makes obtaining data for use in training models a task within itself, and often limits the approaches which can be taken. As of late, there has been a push to create corpora for the use of grammar correction, which has resulted in new annotated corpora such as NUCLE (Dahlmeier and Ng, 2011), or the corpora used in Rozovskaya and Roth (2010). While this is a very heartening stride forward, these corpora are still limited to one language, and to the population of learners from which the annotated corpus was collected. We would need many more annotated corpora, in various languages, to create automated grammar correction systems which actually work for each language if the systems rely on supervised

learning. Also, even within the same language, there may be various subgroups of people we wish to address. For example, the types of mistakes commonly made by Korean ESL students may be very different from the types of mistakes commonly made by French ESL students. So, while the introduction of new corpora is a great stride for the field, it seems we still are very lacking in terms of annotated data. In terms of creating annotated data, Tetreault et al. (2010) shows promise of using Amazon Mechanical Turk for annotating grammatical errors using untrained raters Turkers, reducing the time and cost needed for annotation.

Using classification or rule-based systems for grammatical error detection has thus proven to be successful to some extent, but many of these approaches are not yet sufficient for real-world automated grammar correction for various reasons. First, as we have already mentioned, classification systems and generation-based systems do not make full use of the given data when trying to make a selection. This limits the system's ability to make well-informed edits which match the writer's original intent. Second, many of the systems start with the assumption that there is only one type of error. However, ESL students often make several combined mistakes in one sentence. These combined mistakes can throw off error detection/correction schemes which assume that the rest of the sentence is correct. For example, if a student erroneously writes 'much poeple' instead of 'many people', a system trying to correct 'many/much' errors may skip correction of 'much' to 'many' because it does not have any reference to the misspelled word 'poeple'. Also, having knowledge that 'many/much' errors can occur may help the correction of 'poeple' to 'people', as the word 'many' could be used as a contextual cue for the correction of the spelling. Thus there are advantages in looking at the sentence as a whole, and creating models which allow several types of errors to occur within the same sentence.

In the domain of automated spelling correction, early spell checkers searched for words which were not included in a machine-readable dictionary. This approach limits spell checkers to finding words which have been misspelled into non-words, skipping words misspelled into another existing word. For example, if the word 'trying' was misspelled as 'tying', a spell checker which looks for misspelled words

by matching each word to a dictionary entry will not find this error as it is classified as an existing word. To address this problem, various context-sensitive spelling correction techniques have been proposed. Golding and Roth (1999) use a variant of the weight-update algorithm Winnow (Littlestone, 1988) and a variant of weighted-majority voting to create a high performance context-sensitive spelling correction algorithm. Previous to their approach, several other techniques were also proposed. Mays et al. (1991) use trigrams to find misspellings by using the n-gram context. Bayesian classifiers (Gale et al., 1993), decision lists (Yarowsky, 1994), latent semantic analysis (Jones and Martin, 1997), and other techniques have also been used (Golding and Schabes, 1996; Mangu and Brill, 1997; Powers, 1997). Golding (1995) uses a hybrid model of decision lists and Bayesian classifiers to select the proper word from a confusion set.

In 2000, Brill and Moore (2000) propose using a noisy channel to model the context-sensitive spelling correction problem. Toutanova and Moore (2002) extend the noisy channel model by using word pronunciation information, and build an explicit error model for word pronunciations, increasing performance. More recent work to date on automated spelling correction has been done in context of using web search query data as a source of training data and also as the target of correction (Li et al., 2006; Cucerzan and Brill, 2004; Ahmad and Kondrak, 2005; Kwon et al., 2009; Whitelaw et al., 2009; Sun et al., 2010; Duan and Hsu, 2011).

It is interesting to note that many of the approaches taken above have taken the problem of grammar or spelling correction and split it into two separate parts, identifying an error and then correcting the error. Another approach which is sometimes taken, is to view the problem as a single step process: given a certain observed output, figure out what the originally intended (error-free) output might be. These two approaches are conceptually different, and the resulting attempts also are very different. In this dissertation, we take the second approach, and try to solve the problem as a whole, instead of separating it into an identification and then a correction stage.

1.3 Limitations of this work

Our work is currently based on correcting sentences by only making use of the sentence we wish to correct, without the preceding or following sentences. While this may work in some cases, in many cases the context of the sentence may be needed to determine the actual intent of the writer. For example, when asked to correct the sentence "She drive home.", people may either respond with "She drove home" or "She drives home" or some other plausible correction. However, when also given the previous sentence as context, such as "How does she get home? She drive home.", the correction should be "She drives home" instead of "She drove home", whereas if given the sentences "How did she get home? She drive home." the correction would be "She drove home". This type of context can restrict the possible corrections, and in some cases confine corrections to an exact correction. In actual writing, we often find that the overall context surrounding a sentence will affect the decision of how to correct a sentence. While this may be more preferable when correcting essays or various types of writing, in this work we have restricted the input to just the sentence to correct, so we cannot take the larger context into account in making our decisions. Thus our aim is to correct any given sentence so that it may in agreement with any of the possible sentence corrections produced by humans. Future work may try to address this problem by increasing the scope of context, but the current work described in this dissertation is limited in this matter.

1.4 Dissertation outline

Chapter 2 discusses the basic concepts of our framework and goes into the details of the implementation. The chapter explains our overall generative model of erroneous language production, as well its two component models, the language model and the noise model. We also show how to use unsupervised learning to learn the parameters of the noise model, and explain how to efficiently find the expectation values needed for doing EM.

Chapter 3 explains our need of a data set, and details the specifics of how

we created a data set for our experiments. By collecting various TOEFL writing samples, we created a data set but also needed some manual corrections for evaluation, which is explained in more detail in Chapter 4. We also explain how we collected manual corrections for use in evaluation of our model performance, with much less effort compared to using a professionally trained annotator.

Chapter 4 presents a new method for evaluating the output of automated grammar and spelling correction systems. The lack of a unified process in evaluating performance on automated grammar correction is an obstacle for the current field. We propose a methodology borrowed from the field of machine translation, and run a simple experiment to validate the use of this evaluation technique.

Chapters 5 and 6 contain the details of our various noise models, as well as performance measurements for each model, and analyses of what effects each model has on performance. Some of the models have better performance, while some of the models actually make performance worse.

Finally, Chapter 7 summarizes our contributions and outlines possible future work.

Chapter 1, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

Chapter 2

Framework

In this chapter, we present our model of human language generation, which supports the combination of various types of error processes into one combined model. The model makes use of the whole observed sentence when used to infer the original intended sentence. We also explain how to train the model, by showing how to make use of existing optimized algorithms for weighted finite-state transducers. And finally, we explain how to use the model to find corrections for erroneous observed sentences.

2.1 Base Model

Our base model is a noisy channel model which consists of two main components, a language model and a noise model. The language model is a probabilistic model which places a probability distribution over "error-free" sentences[1]. The probabilistic noise model then takes this sentence and stochastically introduces various types of errors, such as spelling mistakes, article choice errors, wordform choice errors, etc., with probabilities determined by its parameters (see Figure 2.1 for example). Using this model, we can find the posterior probability $p(S_{org}|S_{obs})$ using Bayes rule where S_{org} is the original sentence created by our base language

[1]In reality, the language model will most likely produce sentences with errors as seen by humans, but from the modeling perspective, we assume that the language model is a perfect representation of the language for our task.

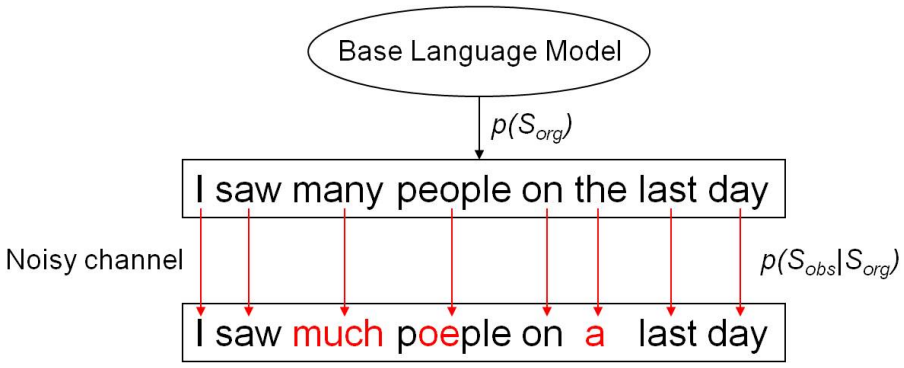


Figure 2.1: Example of noisy channel model

model, and S_{obs} is the observed erroneous sentence.

$$p(S_{org}|S_{obs}) = \frac{p(S_{obs}|S_{org})p(S_{org})}{p(S_{obs})}$$

For the language model, we can use various known probabilistic models which already have defined methods for learning the parameters, such as n-gram models or PCFGs. For the noise model, we need some way to learn the parameter values for the mistakes that a group of specified writers (such as Korean ESL students) make. We address this issue in Section 2.2.

Using this model, we can find the highest posterior probability error-free sentence for an observed output sentence by tracing all possible paths from the language model through the noise model and ending in the observed sentence as output. Of course, to do this, we need to be able to search the set of all possible paths which result in our observed sentence. This is dependent on the type of model used for the language and noise model. We show how to do this in Section 2.2 using our model implementations.

2.2 Implementation

Our model was implemented using a bigram model for the language model. Various noise models which introduce spelling errors, article choice errors, preposition choice errors, etc. were created. The language and noise models are all implemented using weighted finite-state transducers (wFSTs). For operations on

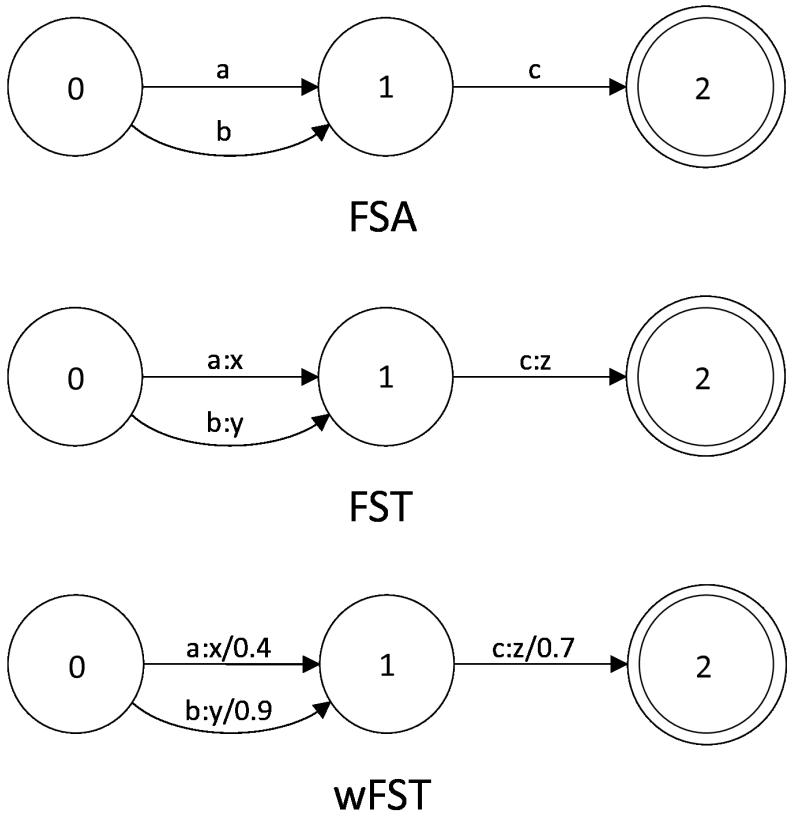


Figure 2.2: Example of an FSA, an FST, and a wFST. The FSA accepts strings 'ac' and 'bc'. The FST accepts the same strings, and outputs 'xz' and 'yz', respectively. The wFST accepts and outputs the same strings as the FST, and also has weights for travelling over each arc.

the wFSTs, we use OpenFST Allauzen et al. (2007), along with expectation semiring code supplied by Markus Dreyer for Dreyer et al. (2008). Before going into more detail about the models, we first introduce weighted finite-state transducers and semirings.

2.2.1 Weighted finite-state transducers and semirings

Motivated in part by applications in speech recognition, machine translation, and other areas of natural language processing, much research has been done on the weighted finite-state transducers (Mohri, 1997, 2005). This research has led to the development of essential algorithms on wFSTs such as composition, de-

terminization, and minimization (Allauzen and Mohri, 2003; Mohri et al., 1996; Pereira and Riley, 1997). In this section, we examine wFSTs and the composition algorithm.

A finite-state transducer (FST) is a finite-state automaton (FSA) which has output values in addition to the input values on the state transitions. A weighted finite-state transducer is an FST which also has a weight for going over each arc. An example of the differences is shown in Figure 2.2. The weights on the arcs of a wFST can be used to calculate the total weight for going over various paths in the wFST. The actual calculations on the weights of a wFST can be abstracted out to the general notion of a semiring (Kuich and Salomaa, 1986). A semiring is an algebraic structure similar to a ring, but which does not require an additive inverse. Thus a semiring S can be defined as a set having two binary operations $\oplus$, $\otimes$ and their identity elements $\bar{0}$, $\bar{1}$, respectively, such that $\oplus$ is associative and commutative, $\otimes$ is associative, $\otimes$ distributes over $\oplus$, and $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$. The abstraction of weights to a semiring allows for definitions of operations over a broad class of weight sets, and the development of algorithms for these operations. Various types of semirings can be used to calculate different types of values. For example, the weight of a path in a wFST is defined to be the product ($\otimes$) of all the weights on the path, and the weight of a set of paths is the sum ($\oplus$) of the path weights. If we have an algorithm which can calculate the weight of a set of paths over any given semiring, the we can calculate the total probability of going over any of the paths using this algorithm by doing the following. We set the weights of each arc to be the probability of going over the arc based on the current state, and then use the semiring ($\mathbb{R}$,+,$\cdot$,0,1), often called the *probability semiring*, to calculate the weight of the set of paths. The results would be the total probability of the paths. Another way to do this would be to use the *log semiring* ($\{\mathbb{R}, \infty\}$,$-\log[e^{-x} + e^{-y}]$,+,∞,0) by setting the weights to the $-\log$ values of the probabilities. The log semiring basically moves the real semiring into log space, and is often preferable to the probability semiring for avoiding underflow caused by floating point arithmetic. The result of running our algorithm on the log semiring will be the $-\log$ value of the total probability

of the paths. Instead of calculating the total probability, perhaps we wish to find the probability of the highest probability path. To do this, we can use the *tropical semiring* ($\{\mathbb{R}, \infty\}$,min,+,∞,0), and again set the arc weights to be the $-\log$ probabilities and run our algorithm for finding the total weight of the paths. The resulting weight will be the $-\log$ probability of the highest probability path, because the $\otimes$ operation is +, effectively adding the $-\log$ probabilities together to find the $-\log$ probability of each path, and then the $\oplus$ operation, defined as min, will select the smallest value.

Now that we have defined a semiring, we can define a weighted finite-state transducer. Formally, a wFST is defined as $T = (\Sigma, \Gamma, Q, E, i, F, \lambda, \rho)$ over the semiring $\mathbb{S}$, such that Σ is a finite set called the input alphabet, Γ is a finite set called the output alphabet, Q is a finite set of states, ε is the empty string, $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times \mathbb{S} \times Q$ is a finite set of transitions, $i \in Q$ is the initial state, $F \subseteq Q$ the set of final states, λ is the initial weight, and ρ is the final weight function. We use $E[q]$ to denote the set of transitions which have the initial state $q \in Q$.

Let us denote, for transition $e \in E$, the previous state as $p[e]$, the next state as $n[e]$, the input label as $i[e]$, the output label as $o[e]$, and the weight as $w[e]$. Then we can define a path $\pi = e_1 e_2 \cdots e_k$ as an element in E^* such that $n[e_i] = p[e_{i+1}]$ for all $i = 1, 2, \cdots, k-1$. We can expand the definition of p and n so that for path π, $p[\pi] = p[e_1]$ and $n[\pi] = n[e_k]$. Let $P(q, q')$ denote the set of paths that start at q (i.e. $p[e_1] = q$) and end at q' (i.e. $n[e_k] = q'$). Let $P(q, x, y, q')$ denote all paths from q to q' with input string $x \in \Sigma^*$ and output string $y \in \Gamma^*$. We can expand the definition of P to $P(q, x, y, R) = \cup_{q' \in R} P(q, x, y, q')$ where $R \subseteq Q$. We define $w[\pi]$, the weight of a path π, as follows.

$$w[\pi] = w[e_1] \otimes w[e_2] \otimes \cdots \otimes w[e_k] \tag{2.1}$$

Using this definition of a path, we define the weight associated by weighted finite-state transducer T to an input, output string pair (x, y) as

$$[T](x,y) = \bigoplus_{\pi \in P(i,x,y,F)} \lambda \otimes w[\pi] \otimes \rho(n[\pi]) \tag{2.2}$$

and $[T](x,y) = \bar{0}$ when $P(i,x,y,F) = \emptyset$.

In this work, we make use of the *composition* operation (Mohri, 2009; Pereira and Riley, 1997). The composition operation, denoted as $X \circ Y$, is defined as follows. Let $X = (\Sigma, \Gamma, Q_X, E_X, i_X, F_X, \lambda_X, \rho_X)$ and $Y = (\Gamma, \Delta, Q_Y, E_Y, i_Y, F_Y, \lambda_Y, \rho_Y)$ be weighted transducers defined over a commutative semiring $\mathbb{S}$, where the output alphabet of X is equivalent to the input alphabet of Y. Then $Z = X \circ Y$ results in a wFST $Z = (\Sigma, \Delta, Q, E, i, F, \lambda, \rho)$ such that for any pair of input, output strings (x, y) such that $x \in \Sigma^*, y \in \Delta^*$, we have

$$[Z](x,y) = \bigoplus_{z} [X](x,z) \otimes [Y](z,y). \tag{2.3}$$

2.2.2 Computing the composition of wFSTs

To make use of composition, we need to be able to compute the composition efficiently. In the case where both transducers are free of transitions with ε labels, we can use the algorithm from Mohri (2005), slightly modified to fit our definition of a wFST, as shown in Figure 2.3. This algorithm creates states from the space $Q_X \times Q_Y$, which refer to a pair of states in X and Y. Starting with state (i_X, i_Y), the algorithm traverses all possible paths by keeping a queue of newly found state pairs, and finding all possible new state pairs from the current pair. The possible new state pairs are found by going through the transitions starting from the current pair of states and checking whether any output from the transition of X matches the input of a transition of Y. For each matching pair of transitions, a new state pair is created using the next states, and added to the queue if the state pair has not already been seen. A transition is created and added to the output transducer as seen in line 18.

When ε labels are used, more factors must be considered. The ε label in a weighted transducer, when placed as the input symbol of a transition, indicates

```
1: procedure COMPOSE(X, Y)
2:     Q ← (i_X, i_Y)
3:     S ← (i_X, i_Y)
4:     i ← (i_X, i_Y)
5:     λ ← λ_X ⊗ λ_Y
6:     while S ≠ ∅ do
7:         (q_1, q_2) ← HEAD(S)
8:         DEQUEUE(S)
9:         if (q_1, q_2) ∈ F_X × F_Y then
10:            F ← F ∪ {(q_1, q_2)}
11:            ρ(q_1, q_2) ← ρ_X(q_1) ⊗ ρ_Y(q_2)
12:        end if
13:        for each (e_1, e_2) ∈ E[q_1] × E[q_2] such that o[e_1] = i[e_2] do
14:            if (n[e_1], n[e_2]) ∉ Q then
15:                Q ← Q ∪ {(n[e_1], n[e_2])}
16:                ENQUEUE(S, (n[e_1], n[e_2]))
17:            end if
18:            E ← E ∪ {((q_1, q_2), i[e_1], o[e_2], w[e_1] ⊗ w[e_2], (n[e_1], n[e_2]))}
19:        end for
20:    end while
21:    return (Σ, Δ, Q, E, i, F, λ, ρ)
22: end procedure
```

Figure 2.3: Algorithm based on Mohri (2005) for composing two weighted finite-state transducers, assuming no ε transitions

that no input is consumed when traversing the transition. When placed as the output symbol of a transition, it indicates that no symbol is output when traversing the transition. The ε label can be used to denote various errors such as the deletion of an intended word, the insertion of an unintended word, or the replacement of n words with a different number of words. Usage of the ε label introduces some complexity to the previous composition algorithm. Efficient composition of weighted transducers which contain ε labels is discussed in Mohri et al. (1996). In the paper, the authors present a way to transform the weighted finite-state transducers being composed so that the first transducer has no ε output transitions, and the second transducer has no ε input transitions. This will let us use the compose algorithm above. We use an example to show how the transformation is done. Suppose we are trying to compose the transducers X and Y as shown in Figure 2.4. We can change X and Y so that all ε input transitions in Y are changed to temporary input symbol ε_2, and all ε output transitions in X are changed to temporary output symbol ε_1. An ε input transition is then added to each state in X such that the input is ε and the output is ε_2, and an ε output transition is added to each state in Y, such that the output is ε and the input is ε_1, creating our transformed transducers X' and Y' as shown in Figure 2.5. Since we no longer have any ε output labels in X', and ε input labels in Y', we can now use our previous compose algorithm. Composing X' and Y' results in the transducer shown in Figure 2.6. Unfortunately, the resulting transducer has multiple paths which lead to the same input/output transitions. Each path corresponds to a different selection order of the epsilon transitions. For the weights to be correct, we need only one path, or the weight of each path will be added multiple times. To avoid these multiple paths, we can add a filter which removes the redundant paths. This filter can be represented as a transducer G, shown in Figure 2.7. Using this filter, we now calculate

$$X' \circ G \circ Y'. \tag{2.4}$$

which results in the transducer shown in Figure 2.8. Since the filter only allows one of the paths, the composed weight is identical to the composed weight of the

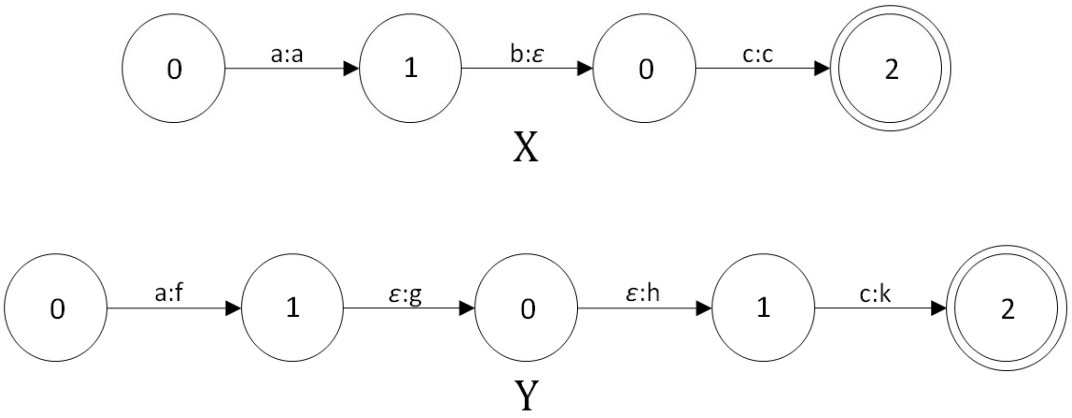


Figure 2.4: Example of finite-state transducers X and Y with ε input and ε output transitions

original transducers, and thus is equivalent to $X \circ Y$. A more detailed explanation of the algorithm is given in Mohri (2009).

The composition operation can be used to combine our language model and noise models, and also confine the output of a wFST to a specified string, which we will explain in Section 2.2.6.

2.2.3 Base language model

The base language model is a bigram model implemented by using a weighted finite-state transducer (wFST). An example is shown in Figure 2.9. The bigram model is implemented as a wFST using the following method. Each state in the wFST represents a bigram context, i.e. the previous word, with the exception of the end state. The arcs of the wFST correspond to having the next word as output, given the previous word. The weights of each arc are the bigram probability of the output word given the previous word specified by the from state. The input word is set to be the same as the output word. The to state of the arc is the context state of the output word. Thus, given a set of n words in the vocabulary, the language model wFST has one start state, from which n arcs extended to each of their own context states. From each of these nodes, n arcs extend to each of the n context states. Transition to the end state gives the probability of the sentence ending, given the last word. In Figure 2.9, these transitions have been denoted as

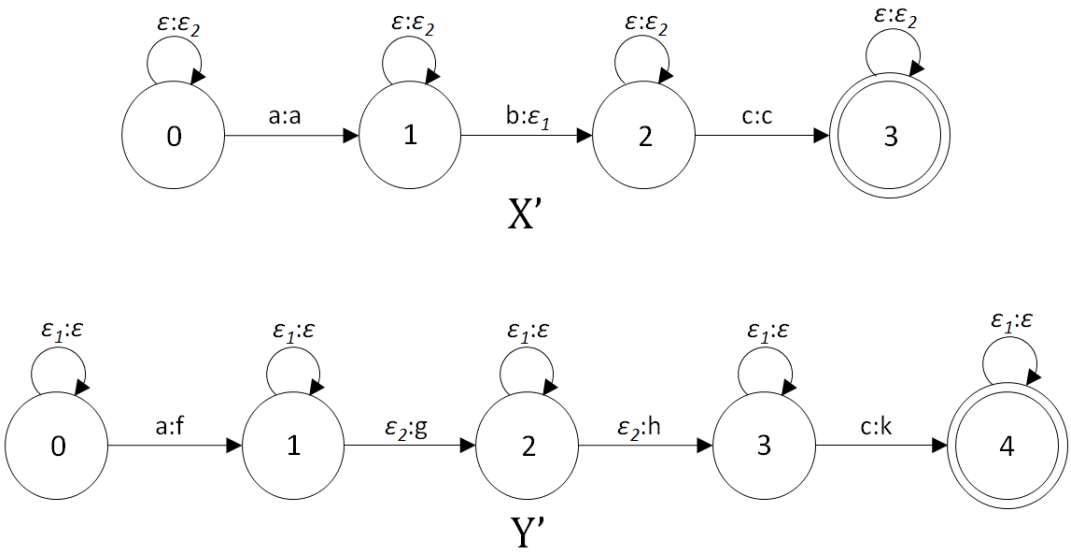


Figure 2.5: Finite-state transducers X and Y from Figure 2.4 transformed to eliminate ε output transitions on X and ε input transitions on Y

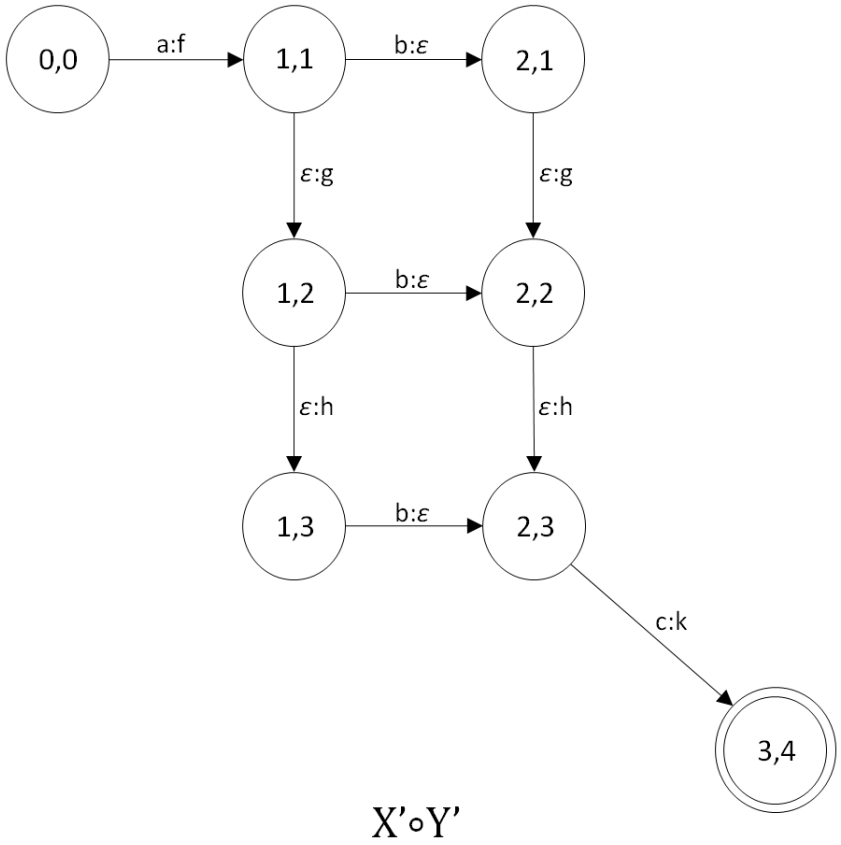


Figure 2.6: Result of $X' \circ Y'$ from Figure 2.5.

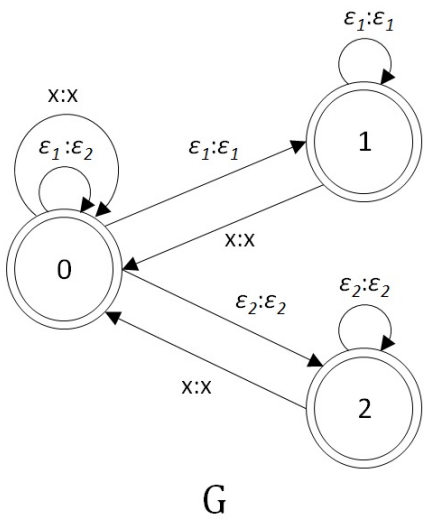


Figure 2.7: Filter for restraining multiple paths due to *epsilon* transition orderings. The 'x' corresponds to any word in the alphabet.

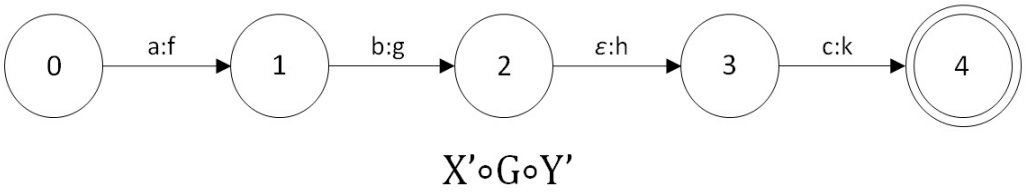


Figure 2.8: Result of $X' \circ G \circ Y'$ using the filter G in Figure 2.7 to restrain multiple paths occurring from *epsilon* transitions

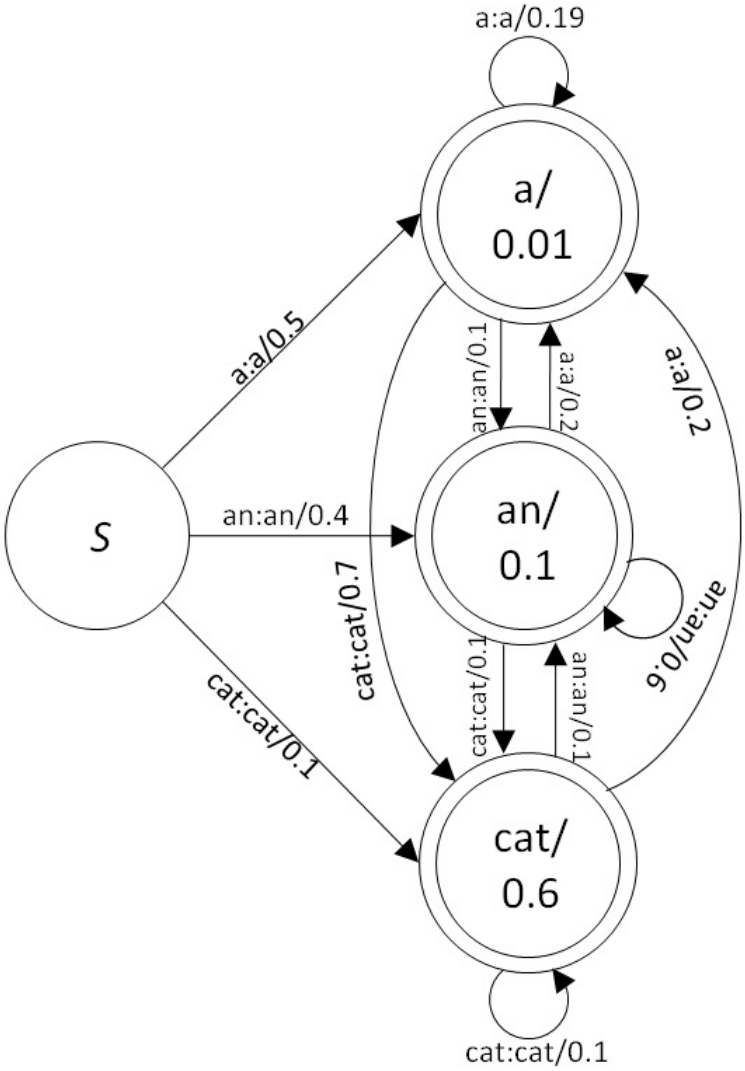


Figure 2.9: Example of a bigram language model containing the 3 words *a*, *an* and *cat*.

epsilon transitions, as we are not taking the sentence delimiter, such as a period or question mark, into account. Thus the number of states in the language model is $n+2$ and the number of arcs is n^2+2n. Creating language models with 10^5 words (the BNC has over 6×10^5) means we would need to create over 10^{10} arcs, which would quickly fill up our memory. To decrease our memory usage, only bigrams whose words are found in the observed sentences, or are determined to be possible candidates for the correct words of the original sentence (due to the noise models) are used. While we use a bigram model here for simplicity, any probabilistic language model having a tractable composition with wFSTs could be used.

The bigram model parameters are learned from the British National Corpus, which was modified to use American English spellings. Due to the sparsity problem, there are many bigrams which have a count of zero. Thus, if we are using maximum likelihood to calculate the bigram model parameters, and $c(x)$ denotes the count of all appearances of x, so that our bigram model parameters are

$$p_{\text{ML}}(x) = \frac{c(x)}{c(e)}$$

there will likely be many bigrams which appear in our observed data but have $p_{\text{ML}}(x) = 0$ because they did not appear in our training data (BNC). To address this problem of data sparsity, a common approach is to use smoothing. Smoothing is used to give probability mass to events which have not been observed. Various types of smoothing algorithms have been proposed for n-gram models, such as additive smoothing (Lidstone, 1920), Good-Turing estimation (Good, 1953), Jelinek-Mercer smoothing (Jelinek and Mercer, 1980), Katz smoothing (Katz, 1987), Witten-Bell smoothing (Bell et al., 1990; Witten and Bell, 1991), absolute discounting (Ney et al., 1994), Kneser-Ney smoothing (Kneser and Ney, 1995), modified Kneser-Ney smoothing (Chen and Goodman, 1998), Church-Gale smoothing (Church and Gale, 1991), Bayesian smoothing (Nádas, 1984; MacKay and Peto, 1995), etc. Most of these approaches can be classified as either using *backoff* or *interpolation*. Backoff smoothing models use the higher-order distribution when the count is nonzero, and use the lower-order distribution only when the count of the n-gram is zero, whereas interpolation models take the linear in-

terpolation of higher-order and lower-order n-gram models, without regard to the distribution of the current n-gram. In our language model, smoothing is done using modified Kneser-Ney, as proposed in Chen and Goodman (1998). Chen and Goodman show that modified Kneser-Ney significantly outperforms regular Kneser-Ney smoothing, as well as several other tested methods. Modified Kneser-Ney takes the original Kneser-Ney smoothing technique and makes two changes. First, interpolation is used instead of backoff. This is a simple switch, and can be done by interpolating the lower-order distribution with all words, instead of just those with zero counts. The second modification splits the single discount parameter D into three discount parameters, D_1, D_2 and D_{3+}. The discount parameter D is set to D_1 for n-grams with a count of one, D_2 for n-grams with a count of two, and D_{3+} for n-grams which have a count of three and above. This change is made due to empirical evidence that the ideal average discount for n-grams with one count or two counts is significantly different from the ideal average discount for n-grams of three and above. With these changes, using the notation used in Chen and Goodman (1998), the modified Kneser-Ney smoothed probabilities can be calculated as

$$p_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i}))}{\sum_{w_i} c(w_{i-n+1}^{i})} + \gamma(w_{i-n+1}^{i-1}) p_{\text{KN}}(w_i|w_{i-n+2}^{i-1})$$

where $D(c)$ is defined as

$$D(c) = \begin{cases} 0 & \text{, if } c = 0 \\ D_1 & \text{, if } c = 1 \\ D_2 & \text{, if } c = 2 \\ D_{3+} & \text{, if } c \geq 3 \end{cases}$$

and

$$\gamma(w_{i-n+1}^{i-1}) = \frac{D_1 N_1(w_{i-n+1}^{i-1}\bullet) + D_2 N_2(w_{i-n+1}^{i-1}\bullet) + D_{3+} N_{3+}(w_{i-n+1}^{i-1}\bullet)}{\sum_{w_i} c(w_{i-n+1}^{i})}$$

given the definitions of $N_1(w_{i+n-1}^{i}\bullet)$, $N_2(w_{i+n-1}^{i}\bullet)$ and $N_{3+}(w_{i+n-1}^{i}\bullet)$ as

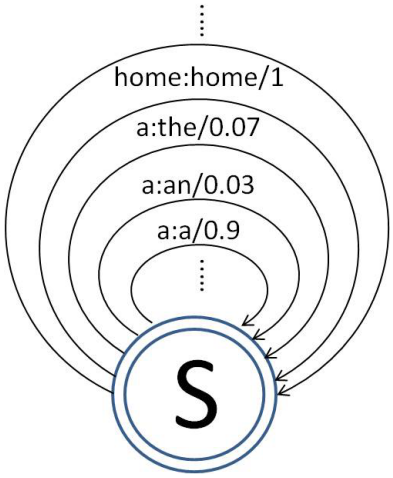


Figure 2.10: Partial example of noise model for article choice error

$$\begin{aligned} N_1(w_{i-n+1}^{i-1}\bullet) &= \left|\left\{w_i : c(w_{i-n+1}^{i-1}w_i) = 1\right\}\right| \\ N_2(w_{i-n+1}^{i-1}\bullet) &= \left|\left\{w_i : c(w_{i-n+1}^{i-1}w_i) = 2\right\}\right| \\ N_{3+}(w_{i-n+1}^{i-1}\bullet) &= \left|\left\{w_i : c(w_{i-n+1}^{i-1}w_i) \geq 3\right\}\right|. \end{aligned}$$

The default parameters D_1, D_2 and D_{3+} are calculated by the following estimates

$$\begin{aligned} Y &= \frac{n_1}{n_1 + 2n_2} \\ D_1 &= 1 - 2Y\frac{n_2}{n_1} \\ D_2 &= 2 - 3Y\frac{n_3}{n_2} \\ D_{3+} &= 3 - 4Y\frac{n_4}{n_3}. \end{aligned}$$

where n_1, n_2 and n_3 are the total number of n-grams which have one, two and three count respectively. To process unknown words, or words we observe in our data set sentences but are not found in the British National Corpus, one randomly chosen word which appeared once in the corpus was changed to the UNK token. When a word not found in the BNC vocabulary is observed in our training, development, or evaluation set, or created from our noise models, it is mapped to the UNK token just before being composed with the language model.

2.2.4 Noise models

For our noise model, we create a weighted finite-state transducer (wFST) which accepts error-free input, and outputs erroneous sentences with a specified probability. To model various types of human errors, we created several different noise models and selectively composed them together, creating a layered noise model. The noise models we implement are spelling errors, article choice errors, preposition choice errors, insertion errors, and deletion errors. We explain these models in more detail later in Chapters 5 and 6.

The design of the more basic noise model wFSTs starts with an initial state, which is also the final state of the wFST. For each word found in the language model, an arc going from the initial state to itself is created, with the input and output values set as each possible word in the vocabulary. These arcs model the case of no error being made. In addition to these arcs, new arcs representing human errors are also inserted. For example, in the article choice error model, an arc is added for each possible (*input, output*) article pair, such as *an:a* for making the mistake of writing an *a* where an *an* should be used. The weights of the arcs are the probabilities of introducing errors (or not, in the case where the input and output values are identical), given the input word from the language model. For example, Figure 2.10 shows a noise model in which *a* will be written correctly with a probability of 0.9, and will be erroneously changed to *an* or *the* with probabilities 0.03 and 0.07, respectively. For this model to work correctly, the parameter values for each noise model is required. How the parameter values are found is explained in the following sections.

2.2.5 Expectation-maximization algorithm

To learn the parameters, a process we explain in the next section, we make use of the expectation-maximization (EM) algorithm (Dempster et al., 1977). The EM algorithm is an iterative method for solving maximum likelihood estimation problems, where the model depends on unobserved latent variables. Each iteration is a two step process: the expectation step (E step), which computes the expected value of the latent parameters, fixing the log-likelihood function and allowing for

the maximization step (M step), which computes the new estimated parameters which maximize the expected log-likelihood calculated in the E step. The new estimated parameters are then used for the subsequent E step, starting a new iteration.

The main idea behind the EM algorithm is to separate the parameters from the latent variables, and use a guess of the parameters and the observed data to make the best possible estimate of the latent variables, and then use those results to find a better estimate of the parameters, iterating this process until the parameter values converge to a local maxima.

A more rigorous definition of the EM algorithm is as follows. Let $\mathbf{X}$ be a set of observed data, $\mathbf{Z}$ the set of unobserved latent data, $\boldsymbol{\theta}$ a vector of unknown parameters, and let $L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z}) = p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})$ denote the likelihood function. Then our objective is to maximize

$$L(\boldsymbol{\theta}; \mathbf{X}) = p(\mathbf{X}|\boldsymbol{\theta}) = \textstyle\sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}),$$

the marginal likelihood of the observed data. Optimizing this quantity is often intractable. Instead, the EM algorithm uses the following two steps iteratively until the values converge. Before the initial step, we arbitrarily select a value for $\boldsymbol{\theta}^{(0)}$.

The E step Calculate the expected value of the log likelihood function as a function of $\boldsymbol{\theta}$. This is done with respect to the conditional distribution of $\mathbf{Z}$, given $\mathbf{X}$ and our estimated value of $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{(t)}$.

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = E_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)}} \left[\log L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z})\right]$$

The M step Find the parameters that maximize the expected value of the likelihood function

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$$

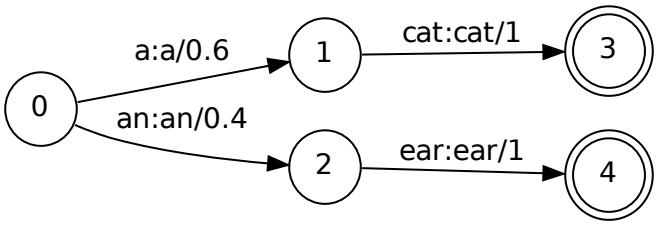
The EM algorithm can be used when it is easy to calculate the distribution of possible values of $\mathbf{Z}$ given some value of the parameter $\boldsymbol{\theta}$, as is done in the E step, and it is possible to find an estimate of $\boldsymbol{\theta}$ when we know the values of $\mathbf{Z}$. It can be proven that the EM algorithm will monotonically converge to the local maximum of the likelihood function, as shown in Wu (1983).

2.2.6 Learning noise model parameters

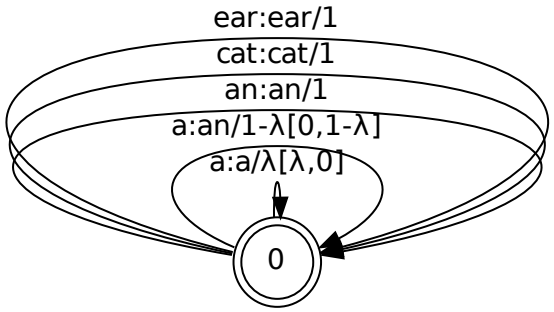
To achieve maximum performance, we wish to learn the parameters of the noise models. If we had a large set of erroneous sentences, along with a hand-annotated list of the specific errors and their corrections, it would be possible to do some form of supervised learning to find the parameters. We looked at the NICT Japanese Learner of English (JLE) corpus, which is a corpus of transcripts of 1,300 Japanese learners' English oral proficiency interview. This corpus has been annotated using an error tagset (Izumi et al., 2004a). However, because the JLE corpus is a set of transcribed sentences, it is in a different domain from our task. The Chinese Learner English Corpus (CLEC) contains erroneous sentences which have been annotated, but we found that the CLEC corpus had too many manual errors, such as typos in both tags and corrections, as well as many incorrect annotations, making it very difficult to automate the processing. Many of the corrections themselves were also incorrect. We were not able to find of a set of annotated errors which fit our task[2]. Instead, we collected a large data set of possibly erroneous sentences from Korean ESL students (Chapter 3). Since these sentences are not annotated, we use an unsupervised learning method to learn our parameters. By using an unsupervised method, we can use our model in other domains and languages, and are not restricted to languages and subsets of people for whom we happen to have an annotated corpus.

To learn the parameters of the noise models, we assume that the collected sentences are random output of our model, and train our model using the

[2]Work focusing on creating such annotated sets is being pursued as of late, resulting in a number of corpora including NUCLE which was introduced in Dahlmeier and Ng (2011) and another data set used in Rozovskaya and Roth (2010).



Language model



Noise model

Training data
A cat.
An cat.
A cat.

Figure 2.11: A simple example language model (top), noise model (middle), and a set of sentences (bottom) for training the parameters. The language model has been reduced from a bigram language model for simplicity, but is sufficient to illustrate the use of the EM algorithm for training the noise model parameters.

Our noise model has one parameter λ which we wish to optimize. Using parameter λ, we can calculate the probability of making the error of writing an instead of a $[p(a : an) = 1 - \lambda]$, or not making an error $[p(a : a) = \lambda]$. The vector of the V-expectation semiring weight is in brackets. The first value of the vector denotes the expected count of no error being made on writing a, and the second that of writing an instead of a.

expectation-maximization algorithm, as described in the previous section. Efficient computation of the expected values of the latent variables in our wFSTs given an estimate of the parameters is done by making use of the V-expectation semiring (Eisner, 2002). The V-expectation semiring is a semiring in which the weight is defined as $\mathbb{R}_{\geq 0} \times V$, where $\mathbb{R}$ can be used to keep track of the probability, and V is a vector which can be used to denote expected arc traversal counts or feature counts. The binary operations are defined to be:

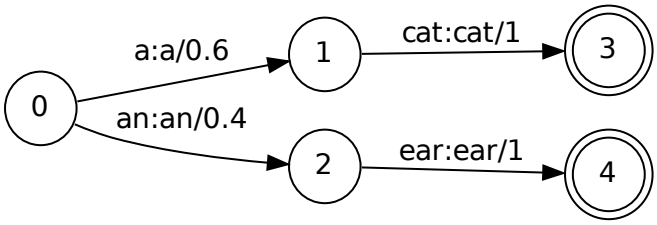
$$(p_1, v_1) \otimes (p_2, v_2) \stackrel{def}{=\!=} (p_1 p_2, p_1 v_2 + v_1 p_2) \tag{2.5}$$

$$(p_1, v_1) \oplus (p_2, v_2) \stackrel{def}{=\!=} (p_1 + p_2, v_1 + v_2) \tag{2.6}$$

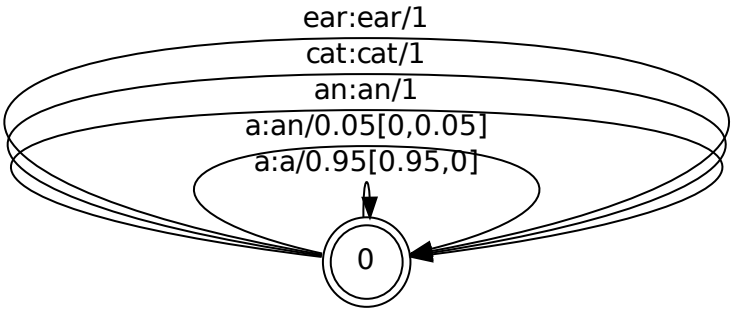
The weight for each of the arcs in the noise models is set so that the real value is the probability p calculated from the current parameters, and the vector which denotes the choice of the arc is set to pe_i where e_i is a standard basis vector, indexed on the type of error or non-error being made. Doing this will set up the wFST so that the real component of the weight of a path is equal to the probability of that path, while the vector component will keep track of the expected counts of the times an arc has been chosen. An example noise model set up to use the V-expectation semiring is shown in the second wFST of Figure 2.11. In this example, we have one parameter value λ we wish to optimize, which is equal to the probability of the article *a* being written correctly, opposed to *a* being erroneously written as *an* with a probability of $1 - \lambda$. By using the expectation semiring, we can keep track of the probability of each path going over an erroneous arc or non-erroneous arc. [3]

Once we have set up our language and noise model, the first step of EM is to initialize our parameters with some random values. After initializing the parameters, we can create a generative language-noise model by composing the language model wFST with the noise model wFSTs. Using our example from Figure 2.11, and initializing our parameter λ so that $\lambda^{(1)} = 0.95$, we show the result of composition of the language model and the initialized noise model in Figure 2.12.

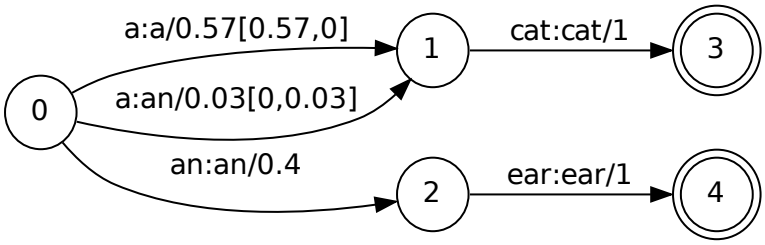
[3] I would like to express my gratitude to Markus Dreyer and Jason Eisner for supplying unpublished code for using the expectation semiring with OpenFST.



Language model

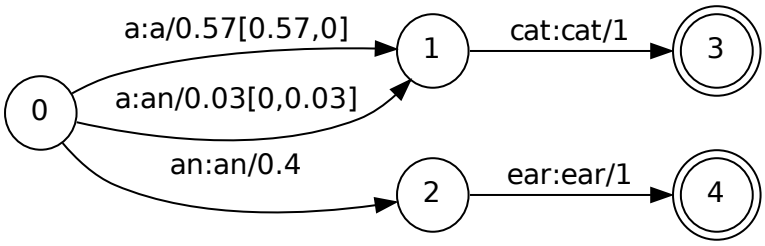


Noise model

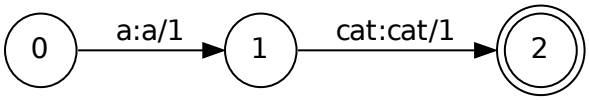


Language-noise model

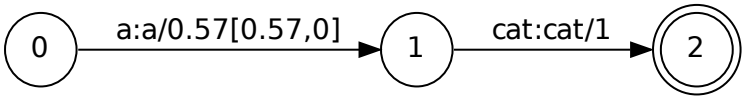
Figure 2.12: Example of language model (top) and noise model (middle) wFST composition, using the V-expectation semiring. The noise model parameter has been initialized to $\lambda^{(1)} = 0.95$, and the result of the composition is shown in the bottom wFST.



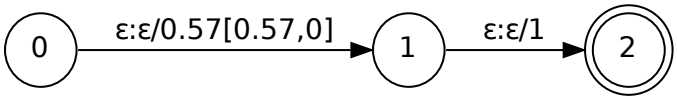
Language-noise model (1)



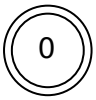
Training sentence wFST (2)



Composition of wFSTs (1) and (2)



Input and output removal



0.57[0.57,0]

ε removal

Figure 2.13: Example of calculating expectation counts. The top wFST shows our language-noise wFST. The second wFST is created using the observed sentence. These two wFSTs are composed, resulting in the middle wFST which only has the observed sentence as output. For training, we change all the input and output values to ε, and then use the epsilon-removal operation to find the total weight of the wFST. The real value of the weight is the total probability of the model generating the sentence, and we can divide each of the vector index values by the total probability to find the expected counts for this sentence.

After our model is set up for the E step using the initial parameters, we must compute the expected counts of the noise model arc traversals for use in calculating our new parameters. To do this, we need to find all possible paths resulting in the observed sentence as output, for each observed sentence. Once we have found all paths, for each possible path, we need to calculate its probability given the output sentence. This will give us the expected counts of going over each erroneous and error-free arc. We can find a wFST which only contains paths which output a given observed sentence by composing the language-noise wFST with the observed sentence wFST. The observed sentence wFST is created in the following manner. Given an observed sentence, an initial state is created. For each word in the sentence, in the order appearing in the sentence, a new state is added, and an arc is created going from the previously added state to the newly added state. The new arc takes the observed word as input and also uses it as output. The weight/probability for each arc is set to 1. For example, if our observed sentence is "a cat", then we create the second wFST shown in Figure 2.13. Composing the sentence wFST with the language-noise wFST has the effect of restricting the new wFST to only allow original sentences which can output the observed sentence from the language-noise wFST. Using the first wFST in Figure 2.13 as our language-noise model, we compose the first two wFSTs to get the third wFST. We now have a new wFST where all valid paths are the paths which can produce the observed sentence. To find the total weight of all paths, we first change all input and output symbols into the empty string. Since all arcs in this wFST are epsilon arcs, we can use the epsilon-removal operation (Mohri, 2002), which will reduce the wFST to one state with no arcs. This operation combines the total weight of all paths into the final weight of the sole state, giving us the total expectation value for that sentence. In our toy example, we can see that the resulting weight is $0.57[0.57, 0]$. 0.57 is the total probability of our observed sentence being output by our language-noise model. We divide all the expectation semiring vector values by the total probability, i.e. the sentence marginal probability, to get the expected counts of going over the *a:a* arc and the *a:an* arc. By doing this for each sentence, and adding the expectation values for each sentence, we can easily compute the

expectation step, from which we then can find the maximizing parameters and update our parameters accordingly. Using our example, the resulting weights from our three training data sentences in Figure 2.11 will be $0.57[0.57, 0]$, $0.03[0.0.03]$ and $0.57[0.57, 0]$. Dividing by the total probability for each sentence gives us expected counts [1,0], [0,1] and [1,0]. In this case, for our M step we can sum the expected counts vectors to get [2,1], which means that the expected count for (a:a) is 2 and (a:an) is 1. We use this to maximize the expected value of the log likelihood function by computing our new estimated value of λ,

$$\lambda^{(2)} = \frac{2}{2+1} = \frac{2}{3}.$$

We are now finished with the first iteration of EM, and can use $\lambda^{(2)}$ to do our next iteration. Iterating is done until the parameter value converges, or the difference between each iteration is lower than some threshold. Because our toy example is very simple, all of our expected count vectors happened to be identity vectors, and at first glance it may seem that there is no reason to use the V-expectation semiring. However, in our application, this is not the case, since smoothing of the bigram counts will permit all possible word combinations, and various types of noise models may create different paths to the output sentence. Fortunately, even when our language model and noise models become more complicated, the V-expectation semiring will calculate the expectation counts we need.

2.2.7 Finding the maximum posterior probability correction

Once the parameters are learned for our language model and noise model, we can use the model to find the maximum posterior probability error-free sentence for any observed sentence. This is done by creating the language model and noise model with the learned parameters, but this time we set the weights of the noise model to just the probabilities, and use the log semiring instead of the V-expectation semiring. Since we only need to find the highest probability sentence, we do not need to keep track of the expected values. We keep the input

strings of the language model arcs the same word as the output, instead of using an empty string. We create a sentence wFST using the observed sentence we wish to correct, the same way the observed sentence wFST for training was created. Once again, the language model, the noise model, and the wFST created using the observed sentence are composed together in order. As with training, we create the noise models and language models by backtracking from each observed sentence, and compose the noise models with the observed sentence first, after which we compose with the language model. We are now left with a wFST in which each path denotes a transition from a sentence produced by the language model to the observed sentence, and the weight is the probability of the sentence production times the probability of the given changes. Thus, to find the maximum posterior probability correction, all we need to do is find the shortest path (in minus-log probabilities) of the new wFST, and the input to that path will be our corrected sentence.

2.2.8 Performance considerations

Since we are doing exact inference, in practice, the language model and noise model are very large. Trying to put them into limited memory space is not an easy task, even with the 142GB memory on the systems we were working on. In order to fit our models into memory, as well as speed things up, we backtracked from the observed sentences and added only the necessary arcs of the noise and language models which would be used in composition. This is done by finding all possibly appearing original sentence words based on the words seen in the observed sentence, and then only adding arcs which use those words into our language and noise models. For example, if the sentence "I like the cat" was observed, and we were using just the article choice error model, the only possible words that could be in the original sentence would be the set of words {I, like, the, a, an, cat}. Since our language model and noise model will be composed with the observed sentence wFST, it is not necessary to include any arcs which have words which are not in the set. Thus we can leave those arcs out, which reduces the amount of memory necessary for the language and noise model. To be more specific, the output

vocabulary of any noise model or language model can be restricted to the input vocabulary of the following model. By starting from the observed sentence, we can minimize the size of each of the noise and language models by only adding arcs which have words in the input vocabulary of the following wFST. For maximum performance, we used small batches of observed sentences and created the noise model and language models for each batch, and used these smaller models to calculate the expectation values.

Another optimization we made concerned the order of the composition. Because of the way the compose algorithm in OpenFST is structured, and also because of the properties of wFSTs in general, the order of composition has large effects on the memory usage and running time. We found that it was much faster to compose the observed sentence wFST to the noise model first, and then compose the output wFST with the language model. This is because the size of the intermediary wFST would be reduced, which in turn would speed up the composition process. Also, the sorting of the arcs done before composition also has a large affect on the performance. Making sure that the FST with the greater out-degree is sorted can greatly decrease the run time.[4]

Chapter 2, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

[4]OpenFST has since been updated, so that if both FSTs to be composed are sorted, it will choose the optimal transducer to match against (per state pair), so it should be advisable to sort both.

Chapter 3

Datasets

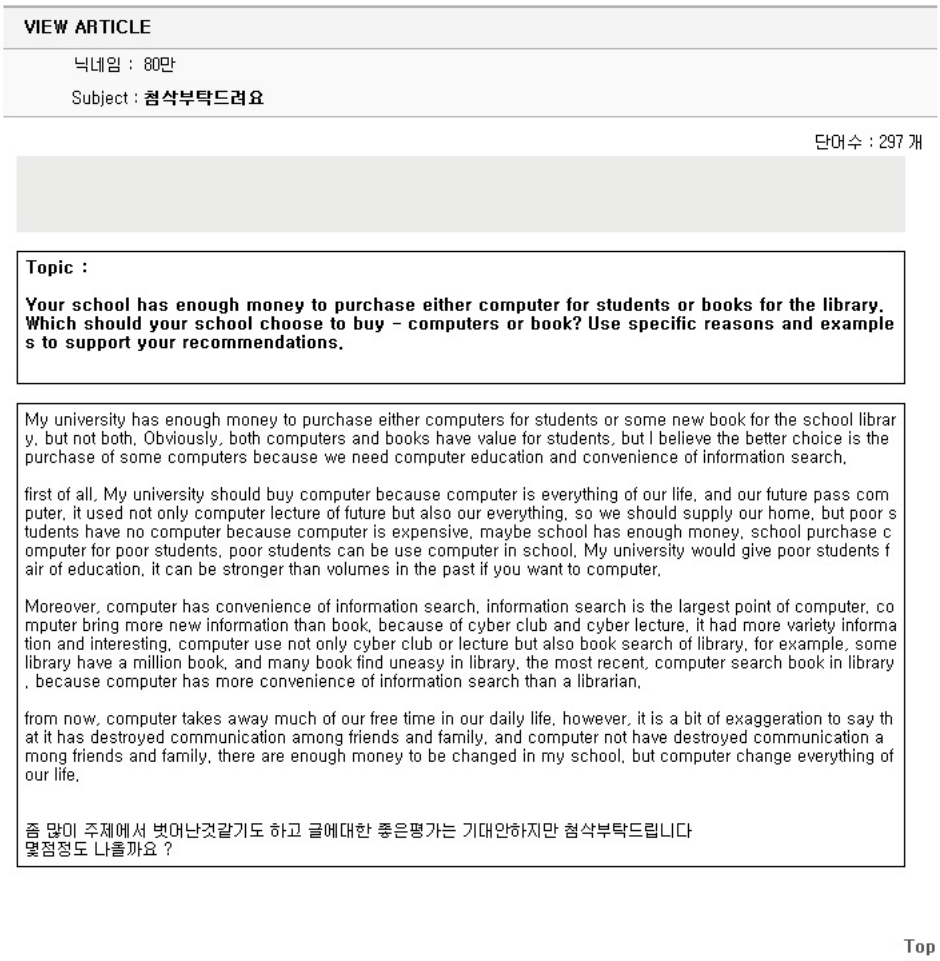
This chapter explains the methodology used to create our data sets for training and evaluation of our models.

3.1 Collection of ESL sentences

As described in Chapter 2, we use EM to learn the parameters of the noise models. This requires observed data, in the form of actual produced noisy sentences. We collected 478,350 sentences written by Korean ESL students, using the following methodology.

3.1.1 Data acquisition

To train and evaluate the noise models, 25,000 practice essays written by Korean ESL students preparing for the TOEFL writing exam were collected. The essays were collected from a free internet web board whose main purpose is for users to post their TOEFL writing samples and ask for feedback. The users of the board are Korean ESL students preparing for the TOEFL writing exam. The posts mainly fall into one of three categories: writing samples, commentative posts, and edited writing samples. A couple of writing samples are shown in Figures 3.1 and 3.2. As can be seen, the format of the writing samples differs from post to post, complicating automated extraction of the sentences to some degree. (The board,



VIEW ARTICLE

닉네임 : 80만

Subject : 첨삭부탁드려요

단어수 : 297 개

Topic :

Your school has enough money to purchase either computer for students or books for the library. Which should your school choose to buy – computers or book? Use specific reasons and example s to support your recommendations.

My university has enough money to purchase either computers for students or some new book for the school librar y, but not both, Obviously, both computers and books have value for students, but I believe the better choice is the purchase of some computers because we need computer education and convenience of information search,

first of all, My university should buy computer because computer is everything of our life, and our future pass com puter, it used not only computer lecture of future but also our everything, so we should supply our home, but poor s tudents have no computer because computer is expensive, maybe school has enough money, school purchase c omputer for poor students, poor students can be use computer in school, My university would give poor students f air of education, it can be stronger than volumes in the past if you want to computer,

Moreover, computer has convenience of information search, information search is the largest point of computer, co mputer bring more new information than book, because of cyber club and cyber lecture, it had more variety informa tion and interesting, computer use not only cyber club or lecture but also book search of library, for example, some library have a million book, and many book find uneasy in library, the most recent, computer search book in library , because computer has more convenience of information search than a librarian,

from now, computer takes away much of our free time in our daily life, however, it is a bit of exaggeration to say th at it has destroyed communication among friends and family, and computer not have destroyed communication a mong friends and family, there are enough money to be changed in my school, but computer change everything of our life,

좀 많이 주제에서 벗어난것같기도 하고 글에대한 좋은평가는 기대안하지만 첨삭부탁드립니다
몇점정도 나올까요 ?

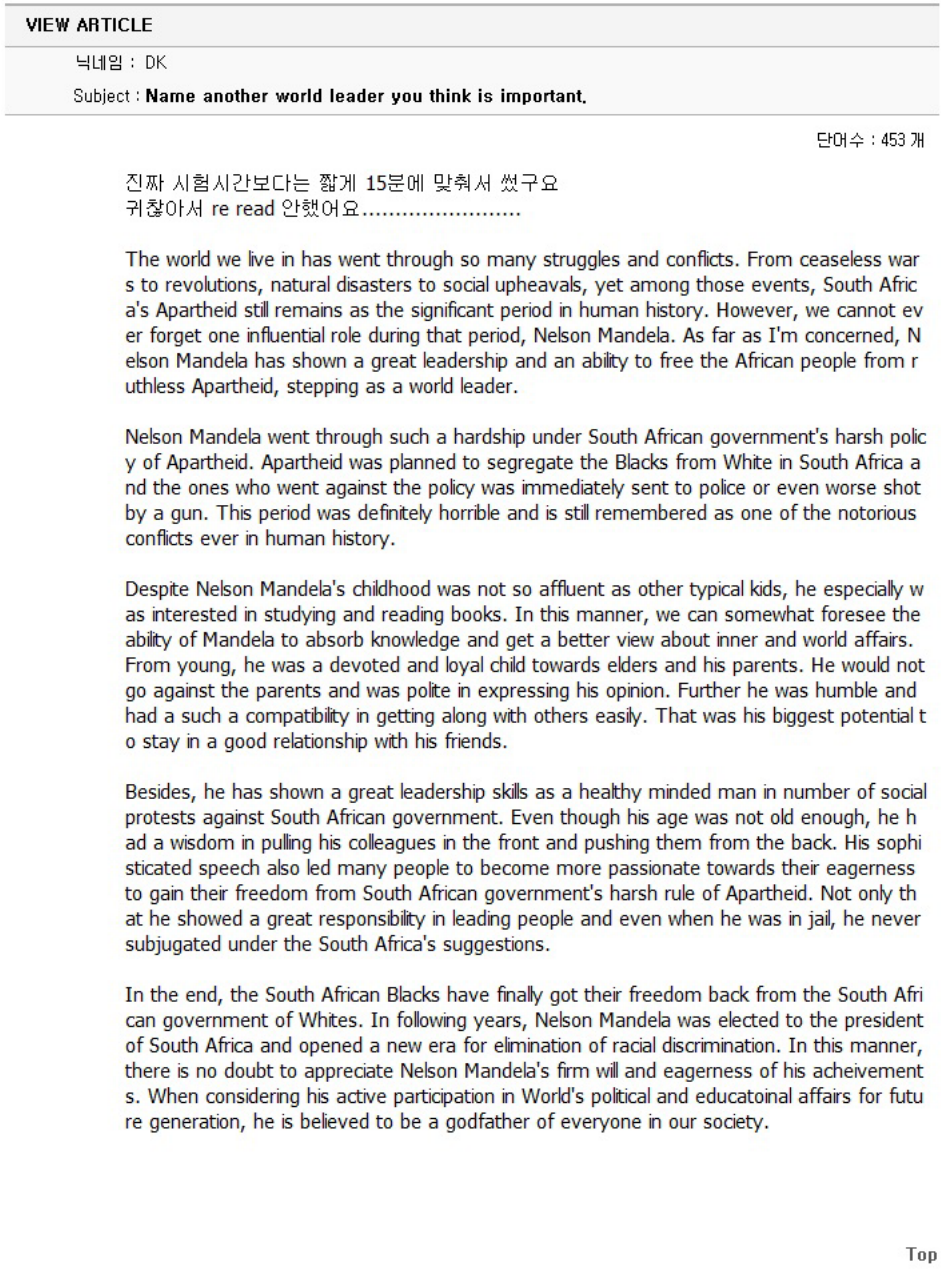
Top

Figure 3.1: TOEFL writing sample

at the time of writing, is located at http://www.gohackers.com/html/?id=essay. gohackers.com is a community website for Korean students who are studying for the TOEFL or GRE exam, and is widely used by students wishing to go to graduate school in the US.)

3.1.2 Sentence extraction

The extraction of sentences from the posts was done through an automated process. First, all the posts on the web board were downloaded. The downloaded HTML files were then processed as follows. For each HTML file, the section with the actual posting content was extracted by deleting the top and bottom portions



VIEW ARTICLE

닉네임 : DK

Subject : **Name another world leader you think is important.**

단어수 : 453 개

진짜 시험시간보다는 짧게 15분에 맞춰서 썼구요
귀찮아서 re read 안했어요.....

The world we live in has went through so many struggles and conflicts. From ceaseless war s to revolutions, natural disasters to social upheavals, yet among those events, South Afric a's Apartheid still remains as the significant period in human history. However, we cannot ev er forget one influential role during that period, Nelson Mandela. As far as I'm concerned, N elson Mandela has shown a great leadership and an ability to free the African people from r uthless Apartheid, stepping as a world leader.

Nelson Mandela went through such a hardship under South African government's harsh polic y of Apartheid. Apartheid was planned to segregate the Blacks from White in South Africa a nd the ones who went against the policy was immediately sent to police or even worse shot by a gun. This period was definitely horrible and is still remembered as one of the notorious conflicts ever in human history.

Despite Nelson Mandela's childhood was not so affluent as other typical kids, he especially w as interested in studying and reading books. In this manner, we can somewhat foresee the ability of Mandela to absorb knowledge and get a better view about inner and world affairs. From young, he was a devoted and loyal child towards elders and his parents. He would not go against the parents and was polite in expressing his opinion. Further he was humble and had a such a compatibility in getting along with others easily. That was his biggest potential t o stay in a good relationship with his friends.

Besides, he has shown a great leadership skills as a healthy minded man in number of social protests against South African government. Even though his age was not old enough, he h ad a wisdom in pulling his colleagues in the front and pushing them from the back. His sophi sticated speech also led many people to become more passionate towards their eagerness to gain their freedom from South African government's harsh rule of Apartheid. Not only th at he showed a great responsibility in leading people and even when he was in jail, he never subjugated under the South Africa's suggestions.

In the end, the South African Blacks have finally got their freedom back from the South Afri can government of Whites. In following years, Nelson Mandela was elected to the president of South Africa and opened a new era for elimination of racial discrimination. In this manner, there is no doubt to appreciate Nelson Mandela's firm will and eagerness of his acheivement s. When considering his active participation in World's political and educatoinal affairs for futu re generation, he is believed to be a godfather of everyone in our society.

Top

Figure 3.2: Another TOEFL writing sample

with unnecessary HTML content. Next, all instances of "" were deleted[1], and all instances of "ï" and "é" were replaced with the characters 'i' and 'e', respectively. The files were then run through a parsing process which extracted the title, number, user id, etc. of the post. Following the parsing process, the files were run through html2text to extract the content into text format. Once this processing was finished, all posts that had a "[re" or "re.." in the title were discarded. This was done to remove repeating text appearing in edit posts and commentative posts.

In most but not all cases, when the post was a writing sample, it loosely followed the following format: a topic question at the beginning, followed by the writing sample we are interested in extracting, and then maybe a few words from the writer asking for help at either the beginning or end of the post. Since there was no obvious marker to automatically separate these items, we used the following procedure to extract what we deemed were valid sentences from the main content of each post.

First, each line was preprocessed by looking for instances of various characters which were deemed to be English characters, but in a different encoding format than the usual ASCII coding. These anomalies were likely introduced by copying and pasting from various types of editors into web browsers before uploading content. The found instances were replaced with their matching ASCII characters. In addition, all tab characters were changed to blank spaces. After removing blank characters from the beginning and end of each line, all lines which did not begin with an ASCII letter were deemed inappropriate and were removed. After verifying the first character, the following characters were subjected to a test of being a valid character. The set of valid characters was comprised of all ASCII letters, a blank space, all ASCII digits, and the characters listed in Table 3.1. The test was applied to each character starting from the first character to the last, in order. During this process, if a Korean character was found, the line was removed, and a counter keeping track of the lines with a Korean character was increased.

[1]Only 5 HTML files were found to contain "", the HTML code for a soft hyphen, and manual inspection of these cases deemed them to be irrelevant characters.

Table 3.1: Allowed characters for line data

Allowed characters
’ , ; : . ! ( ) ” - / & % $ + ? * ℃

If an invalid character was found and the character was a ‘→’ or ‘[’ character or the line contained the string “->” or “=>”, the whole post was assumed to be an edited writing sample and discarded. Otherwise, only the line was removed. If the number of lines found including Korean characters was greater than 15, the text was also discarded, under the assumption that the post was not a TOEFL writing sample, but more likely a commentative post. If the line was made up of only valid characters, it was then subjected to a test checking whether it was a TOEFL topic question or the actual writing response. This test simply checked whether the line contained one of the following strings : “Compare”, “Describe”, “Do”, “Use”, “Explain”, “Give”, “. use”, or “? use”. If the line contained any one of the strings, it was deemed to be a TOEFL topic question and was removed.

Once all the valid lines of a post were found, the lines were then concatenated together. The concatenated text was then checked to see if it is longer than 500 characters, under the assumption that a TOEFL writing sample would be at least that length. This was done to remove posts which are not writing samples, but are questions or comments posted about a few sentences possibly from another previous post. The concatenated text was then split into sentences by looking for the punctuation marks *.*, *!*, and *?*. Sentences with a length of 1 or 2 characters were removed. This process is very rudimentary, and is especially affected by the multiple uses of ‘.’. For example, instances of “a.m.” and “p.m.” will create sentence fragments, as having those words in a sentence will cause the sentence to be cut into 3 fragments instead of one full sentence. Also, in some cases, students misuse a ‘.’ in place of a ‘,’, creating sentence fragments instead of full sentences. This is one area where our data set could use more improvement. Manual evaluation of 1521 randomly extracted sentences returned 17 sentence fragments.

3.2 Training, development and evaluation set

For training and evaluation purposes, the data set was split into a training set, a development set, and an evaluation set. The training set was used to learn the parameters for the noise models. The development set was used to evaluate the model performance during development. The final evaluations were done on the evaluation set. For the development set, we randomly selected 504 and 1016 sentences for the development set and evaluation set, respectively. The remaining 476,829 sentences were used as the training set.

3.2.1 Obtaining reference sentences for evaluation

As we will explain in Chapter 4, we use BLEU and METEOR metrics to evaluate the performance of our models. These two metrics require corrected versions of erroneous sentences, or *reference* sentences. Since the development set and evaluation set are used to evaluate performance, it is necessary to obtain manual corrections as reference sentences for the sentences in the two sets. To do this, the development set and evaluation set sentences were put on Amazon Mechanical Turk as a correction task. The workers on Amazon Mechanical Turk were asked to manually correct the sentences in the two sets. Workers had a choice of selecting 'Impossible to understand', 'Correct sentence', or 'Incorrect sentence'. When 'Incorrect sentence' was chosen, they were asked to correct the sentence so that no spelling errors, grammatical errors, or punctuation errors were present. Each sentence was given to 8 workers, giving us a set of 8 or fewer reference sentences for each incorrect sentence. We asked workers not to completely rewrite the sentences, but to maintain the original structure as much as possible. Each hit was comprised of 6 sentences, and the reward for each hit was 10 cents, which resulted in a total payment of $223.52. Only workers based in the US were permitted to perform the task, as previous tests of this task showed that the corrections of users outside of the US were highly unreliable.

Despite being based in the US, we still found that many of the workers' corrections had errors. To remove the erroneous corrections, and ensure the quality

Figure 3.3: Partial template of Amazon Mechanical Turk task for collecting reference sentences for evaluation

of our reference sentences, a native English research assistant went over each of the manual corrections and marked them as being either correct or incorrect. The original sentence was also marked to be correct or incorrect by our assistant.

Finally, to create our set of reference sentences, we used the following process. For each sentence, we looked at each worker's selected choice. If the choice was 'Correct sentence', then we referred to our assistant's marking for the original sentence, and if the original sentence was deemed correct, it was added to the reference set (if it had not already been added). If the choice was 'Impossible to understand', then we ignored that worker's input for the sentence, and no reference sentence was added for that worker. If the worker chose 'Incorrect sentence', then we referred to our research assistant's labeling for the worker's correction, and added the correction only if it had been labeled as being correct by our assistant. Otherwise, we discarded the correction. Thus, some sentences, for example a correct sentence, having been labeled as such by both the research assistant and all 8 workers would only have the original sentence as a reference sentence. If one of the workers had labeled that sentence as incorrect, and changed the sentence to a different sentence which was still grammatically correct, then our research assistant would have marked the 'correction' as correct, and we would have had two reference sentences. If an incorrect sentence had been corrected by all 8 workers, but only 3 of the 'corrections' were deemed correct by our research assistant, there would be 3 reference sentences for the incorrect sentence. For sentences which were deemed incorrect and had no correct 'corrections', the reference set size was 0 and these sentences were dropped from the evaluation set. Removing the incorrect 'corrections' effectively decreased the BLUE and METEOR scores of the observed ESL sentences. An observation of the sentences showed that this was mostly due to the fact that when an incorrect 'correction' only partially fixed a sentence, and left some part of the sentence unfixed in a reference sentence, this would result in boosting the score of the observed sentence. For many sentences, not all workers agreed on whether the sentence was a 'Correct sentence' or an 'Incorrect sentence' or was 'Impossible to understand', but despite this our methodology selects all possible correct reference sentences that we have obtained.

We introduce our method for evaluation of automated full sentence grammar/spelling correction in the following chapter.

Chapter 3, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Evaluation Method

In the previous chapters, we discussed how to create a system for finding automated grammar corrections using a probabilistic model. While finding good corrections is the main part of our task, we still need a way to evaluate the performance of our system.

In the current literature, grammar correction tasks are often manually evaluated for each output correction, or evaluated by taking a set of proper sentences, artificially introducing some error, and seeing how well the algorithm fixes the error. Manual evaluation of automatic corrections may be the best method for getting a more detailed evaluation, but to do manual evaluation for every test output requires a large amount of human resources, in terms of both time and effort. In the case where artificial lesioning is introduced, the lesions may not always reflect the actual distribution of errors found in human data, and it is also difficult to replicate the actual tendency of humans to make a variety of different mistakes in a single sentence. Thus, this method of evaluation, which may be suitable for evaluating the correction performance of specific grammatical errors, would not be fit for evaluating our model's overall performance.

For evaluation of the given task, we have incorporated evaluation techniques based on current evaluation techniques used in machine translation, i.e. BLEU and METEOR. Machine translation addresses the problem of changing a sentence in one language to a sentence of another. The task of correcting erroneous sentences can also be thought of as translating a sentence from a given language A, to

Candidate	A boy a boy a boy a a
Reference 1	A boy was playing on the playground with some girl
Reference 2	A boy and a girl were playing on the playground
Reference 3	A couple of kids were playing on the playground

Figure 4.1: Example for BLEU metric calculation

another language B, where A is an 'incorrect grammar' language, and B is the 'correct grammar' language. Under this context, we can apply machine translation evaluation techniques to evaluate the performance of our system. We now explain the BLEU and METEOR metrics.

4.1 BLEU and METEOR

BLEU (Papineni et al., 2002) and METEOR (Lavie and Agarwal, 2007) are two state-of-the-art evaluation metrics currently being used in the field of machine translation. Both of these metrics compare a machine translated output sentence[1] to reference translations, and try to evaluate the similarity of the translation output to the reference. The reference translations are a set of high quality translations, often obtained from various existing translations of the same document, or by asking several qualified translators to translate a given document. Both BLEU and METEOR return a score between 0 and 1, where 1 is the best possible score and a higher score translation is deemed to be better than a lower score translation.

4.1.1 BLEU

The BLEU (Bilingual Evaluation Understudy) metric is based on a modified form of precision. The unmodified version of n-gram precision is defined in machine translation to be

$$P = \frac{m}{w_t}, \tag{4.1}$$

[1]The unit of output used is actually a segment, but usually a segment will be mapped to one sentence

where m is the number of n-grams in the translation output that can be found in any of the reference sentences, and w_t is the number of n-grams in the translation output. Using precision in this form may have undesirable consequences when a word is repeated several times. For example, let us consider the case shown in Figure 4.1, where we are trying to calculate the score of the candidate sentence given three reference sentences. Our output translation, or candidate sentence, is the string "a boy a boy a boy a a". Since both of the words 'a' and 'boy' are found in at least one of the reference sentences, the unigram precision has a score of $\frac{8}{8} = 1$. To address this problem, BLEU uses a modified form of n-gram precision by constraining repeated n-grams to only be counted up to the max number of times it appears in any single reference sentence. In our example, each of the three reference sentences had 1, 2, and 1 instances of 'a' and 1, 1, and 0 instances of 'boy', respectively. Then the modified unigram precision used in BLEU would be

$$\frac{\min(5, \max(1, 2, 1)) + \min(3, \max(1, 0, 0))}{8} = \frac{2+1}{8} = \frac{3}{8} \tag{4.2}$$

instead of 1. The modified n-gram precision for each value of n, up to a selected N-gram count, is calculated separately and then combined together by taking a weighted geometric mean of the values, where the weights sum to 1. In our example above, the modified unigram and bigram precision would be $\frac{3}{8}$ and $\frac{1}{7}$, respectively. Experiments have shown that when $N = 4$, the BLEU scores correlate most highly with human evaluation scores.

The use of modified n-gram precision has its benefits, but it still is confined to the characteristics of precision. Longer sentences may be penalized by the modified n-gram precision, but only using modified precision allows for very short translations which omit large amounts of the translation to achieve high scores. Usually, in these type of cases, recall is used concurrently with precision to get a better evaluation of the performance. In the case of machine translation, however, there are often many different ways a sentence can be translated, making the use of recall a much more complicated task.

To address this problem, BLEU adds in a brevity penalty factor. The brevity penalty BP is calculated over the whole output, not just each individ-

Candidate	A boy was on the playground playing with a girl
Reference 1	A boy was playing on the playground with some girl
Reference 2	A boy and a girl were playing on the playground
Reference 3	A couple of kids were playing on the playground

Figure 4.2: Example for METEOR metric calculation

ual sentence. For each output sentence, the sentence with the closest length is matched, and the sum of the lengths for the matching reference sentences are summed together to calculate the effective reference corpus length r. The length c of the candidate translation is used with r to calculate the brevity penalty BP, which is defined as

$$\mathrm{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r. \end{cases} \tag{4.3}$$

Finally, the BLEU score is calculated by multiplying the brevity penalty to the previously calculated weighted geometric mean as follows:

$$\mathrm{BLEU}_{\mathrm{score}} = \mathrm{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right). \tag{4.4}$$

The brevity penalty for our example would be $e^{1-9/8}$, since our candidate sentence is 8 words long, and the shortest reference sentence is 9, and our whole output is just one sentence. We can calculate the final BLEU score for our example using $N = 2$ as

$$e^{1-9/8} \cdot \exp\left(\tfrac{1}{2}\log\tfrac{3}{8} + \tfrac{1}{2}\log\tfrac{1}{7}\right) = 0.2043$$

We introduce the METEOR metric in the next section.

4.1.2 METEOR

The METEOR (Metric for Evaluating Translation with Explicit Ordering) metric was designed to improve evaluation performance over BLEU at the sentence

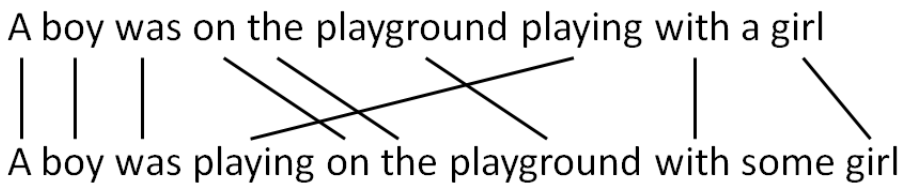


Figure 4.3: Example of METEOR alignment

(segment) level. METEOR, unlike BLEU, uses only one reference sentence to calculate the final METEOR score. The algorithm is run on all reference sentences, and the reference sentence with the best score is used. METEOR is based on using word alignments between the output translation and a reference sentence used for evaluation. Alignments are created by mapping words in the output sentence to the words in the reference sentence. A mapping is a pairing of one word in the output sentence to a word in a reference sentence, and an alignment is a set of mappings, as seen in the example in Figure 4.3. Each word may only be mapped to one word in the other sentence. The alignment is produced incrementally using three different modules: the 'exact' module, the 'porter stem' module, or the 'WN synonymy' module. The 'exact' module allows words to be mapped together if they are exactly the same, while the 'porter stem' module maps words if they are the same after being stemmed using the Porter stemmer (Porter, 1980). The 'WN synonymy' module maps words if they both belong to the same 'sysnet' in WordNet (Fellbaum, 1998). The 'porter stem' and 'WN synonymy' modules help score translations which may not be the exact same word as that in the reference sentence, but have captured the meaning of the foreign text to some degree.

Using all possible mappings from the 'exact' module, alignments are created, and the alignment which maximizes the number of mappings is selected. If there are several possible alignments with the same maximum size, the alignment which has the least number of unigram mapping crossings is selected. The alignment is then augmented incrementally by adding new mappings using the 'porter stem' module, and then again using the 'WN synonymy' module. While the default ordering of METEOR is 'exact', 'porter stem', and then 'WN synonymy', we can also change the ordering or use only a subset of the available modules.

Once the full alignment has been found, the unigram precision P is calculated as:

$$P = \frac{m}{w_t} \tag{4.5}$$

where m is the number of unigrams which were mapped in the given alignment, and w_t is the number of unigrams in the output sentence. The unigram recall R is calculated as:

$$R = \frac{m}{w_r} \tag{4.6}$$

where m is the same as above, and w_r is the number of unigrams in the reference sentence. If we use the alignment in Figure 4.3, the precision and recall values would both be $\frac{9}{10}$. The precision and recall values are combined together into F_{mean} by using a weighted harmonic mean:

$$F_{mean} = \frac{PR}{\alpha P + (1-\alpha)R} \tag{4.7}$$

where α is a factor to balance out the importance of precision to recall. In our example, we can calculate the F_{mean} value using an α value of 0.9, which gives us:

$$F_{mean} = \frac{0.9(0.9)}{0.9(0.9) + (0.1)0.9} = 0.9.$$

So far, the precision, recall, and F_{mean} values have only been based on the unigram mappings, and do not have anything to do with the ordering. To add ordering into consideration, METEOR calculates the fragmentation degree of the ordering. This is done by dividing the mappings in the alignment into *chunks*. A chunk is a set of mappings which map consecutive words from the output sentence to consecutive words in the reference sentence, in identical word order. An example is shown in Figure 4.4. The alignment is divided into the fewest possible number of chunks c, and the fragmentation value frag is then calculated as:

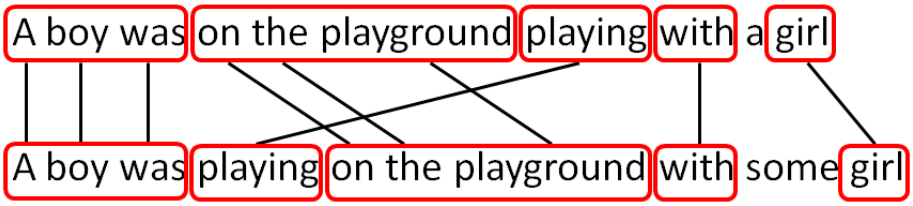


Figure 4.4: Example of METEOR chunks

$$\text{frag} = \frac{c}{m} \tag{4.8}$$

where m is the number of mappings in the alignment. In our example, the frag value is $\frac{5}{9}$. The penalty MP is then calculated as:

$$\text{MP} = \gamma \cdot \text{frag}^{\beta} \tag{4.9}$$

where γ is a factor which determines the maximum penalty ($0 \leq \gamma \leq 1$) and β determines the functional relation between fragmentation and the penalty. The penalty is combined with F_{mean} to produce the final METEOR score as follows:

$$\text{METEOR}_{\text{score}} = (1 - \text{MP}) \cdot F_{mean} \tag{4.10}$$

The values of α, β and γ can be adjusted, and Lavie and Agarwal (2007) reports various experiments for optimizing these values for various languages. Using $\alpha = 0.9$, $\beta = 3$, and $\gamma = 0.5$ for purposes of this example, we can finally calculate the METEOR score of our example as follows:

$$\text{METEOR}_{\text{score}} = (1 - 0.5 \cdot \tfrac{5}{9}^3)0.9 = 0.8228$$

In the next section, we talk about using BLEU and METEOR for grammar correction evaluation.

4.2 Using BLEU and METEOR for grammar correction evaluation

For evaluation of the given task, we have incorporated evaluation techniques based on BLEU and METEOR. In order to use BLEU and METEOR, we need to have reference translations on which to score our output. As we explained in Chapter 3, we have collected a set of reference sentences for evaluation. Again, we point out that collecting reference sentences is not a difficult task, especially by making use of cheap manual labor through systems like Amazon Mechanical Turk. Another problem with annotating an evaluation set is that the same error may be corrected in several different ways. It is not an easy task to list all possible corrections. Also, the fashion in which one error in a sentence is corrected may affect the way another error in the sentence must be corrected. For example, consider the sentence 'I like book that has pictures.' to correct. This sentence may be corrected to 'I like a book that has pictures', but it also may be corrected to 'I like books that have pictures'. Thus, how we correct the first part of the sentence influences whether or not the second part of the sentence must be corrected. If both corrections are in our reference set, we will be fine. But while it may be possible, it would be much more difficult to assess the performance of the correction when we only have annotated data on the errors and their corrections. Having a large number of reference sentences would help mitigate this type of problem to some degree with regard to the evaluation of such sentences. It is also notable that, in comparison with machine translation evaluation, collecting corrections for the sentences is a much easier task than finding various correct translations, since the task of editing is much easier than translation and there usually exists a much larger set of qualified people. Instead of having to find people who are fluent in two languages, we only need to find people who are qualified in a given language to obtain proper reference sentences.

Using our manually corrected reference sentences, we evaluate our model's correction performance using METEOR and BLEU. Since METEOR and BLEU are fully automated once we have our reference translations (manual corrections),

Table 4.1: BLEU and METEOR scores for ESL sentences vs manual corrections on 686 sentences, averaged by equally weighting each reference sentence

	METEOR	BLEU
Original ESL sentences	0.8468	0.7644
Manual corrections	0.9702	0.9561

Table 4.2: BLEU and METEOR scores for ESL sentences vs manual corrections on 686 sentences, averaged by equally weighting each observed sentence

	METEOR	Better(M)	BLEU	Better(B)
Original ESL sentences	0.8399	18	0.7513	44
Manual corrections	0.9647	576	0.9475	570

we can run evaluation on our tests without the need for any further manual input. While these two evaluation methods were created for machine translation, they also have the potential of being used in the field of grammar correction evaluation. While these parallels exists, we note that the output of machine translation is often not at the level of fluency as the sentences needing grammar correction. Thus, one difference between machine translation and our task is that finding the right lemma is in itself something to be rewarded in MT, but is not sufficient for our task. Because of this, evaluation of grammar correction should be more strict. Using the 'stemmer' or 'synonymy' modules would actually allow for some of the exact errors that we are trying to fix. Thus, for METEOR, we use the 'exact' module for evaluation of our output sentences. For BLEU, we used a max of 4-grams, and set the weights for the geometric mean to be uniform ($w_i = \frac{1}{4}$). For METEOR, the parameters were set so that (α,β,γ)=(0.8,0.83,0.28), based on the parameter performance evaluation of English in Lavie and Agarwal (2007).

To validate our evaluation method, we ran a simple test by calculating the METEOR and BLEU scores for the observed sentences, and compared them with the scores for the manually corrected sentences, obtained using the methodology in Section 3.2.1, to test for an expected increase. The scores for each correction were evaluated using the set of corrected sentences minus the correction sentence being

Table 4.3: Number of sentences which have a higher average BLEU, METEOR score on 686 sentences. The improvements are significant by the sign test at $p < 0.00001$.

	METEOR	BLEU
Original ESL sentences	18	44
Manual corrections	576	570
Tied	92	72

evaluated. For example, suppose we have the observed sentence o, and correction sentences c_1, c_2, c_3, c_4 and c_5 from Mechanical Turk. We run METEOR and BLEU on both o and c_1 using c_2, c_3, c_4 and c_5 as the reference set, as shown on the top part of Figure 4.5. We repeat the process for o and c_2, using c_1, c_3, c_4 and c_5 as the reference set, shown in the middle segment of Figure 4.5, and so on, until we have run METEOR and BLEU on all 5 correction sentences. Using our evaluation set of 1017 manually labeled sentences, we ran this test. Note that for the test, we must have at least 2 reference sentences, or corrections which claim the original sentence to be incorrect. Of the 1016 sentences, 686 had at least two reference sentences. We calculated the averages for the BLEU and METEOR values by weighting the result for each reference sentence with the same weight. The average METEOR score for the ESL sentences was 0.8468, whereas the corrected sentences had an average score of 0.9702. For BLEU, the average scores were 0.7644 and 0.9561, respectively, as shown in Table 4.1. We also calculated the averages for the BLEU and METEOR values by equally weighting the average score values for each observed sentence. The average METEOR score in this case for the ESL sentences were 0.8399, whereas the corrected sentences had an average score of 0.9647. For BLEU, the average scores were 0.7513 and 0.9475, respectively, as shown in Table 4.2. We can see that in both cases, the manual corrections score much higher than the original ESL sentences, for both metrics. To make sure this was not just an artifact of a few high scores, we also counted the number of sentences for which the BLEU and METEOR scores had a higher average for the original sentence compared to the reference sentences, and vice versa, as shown in Table 4.3. Both

metrics give us a much higher rate of agreement between the manual references than with the ESL sentence. The improvements are significant by the sign test at $p < 0.00001$. Thus, we have confirmed that the corrected sentences score higher than the ESL sentence.

We now present each of our noise models and their performance in the following chapter.

Chapter 4, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

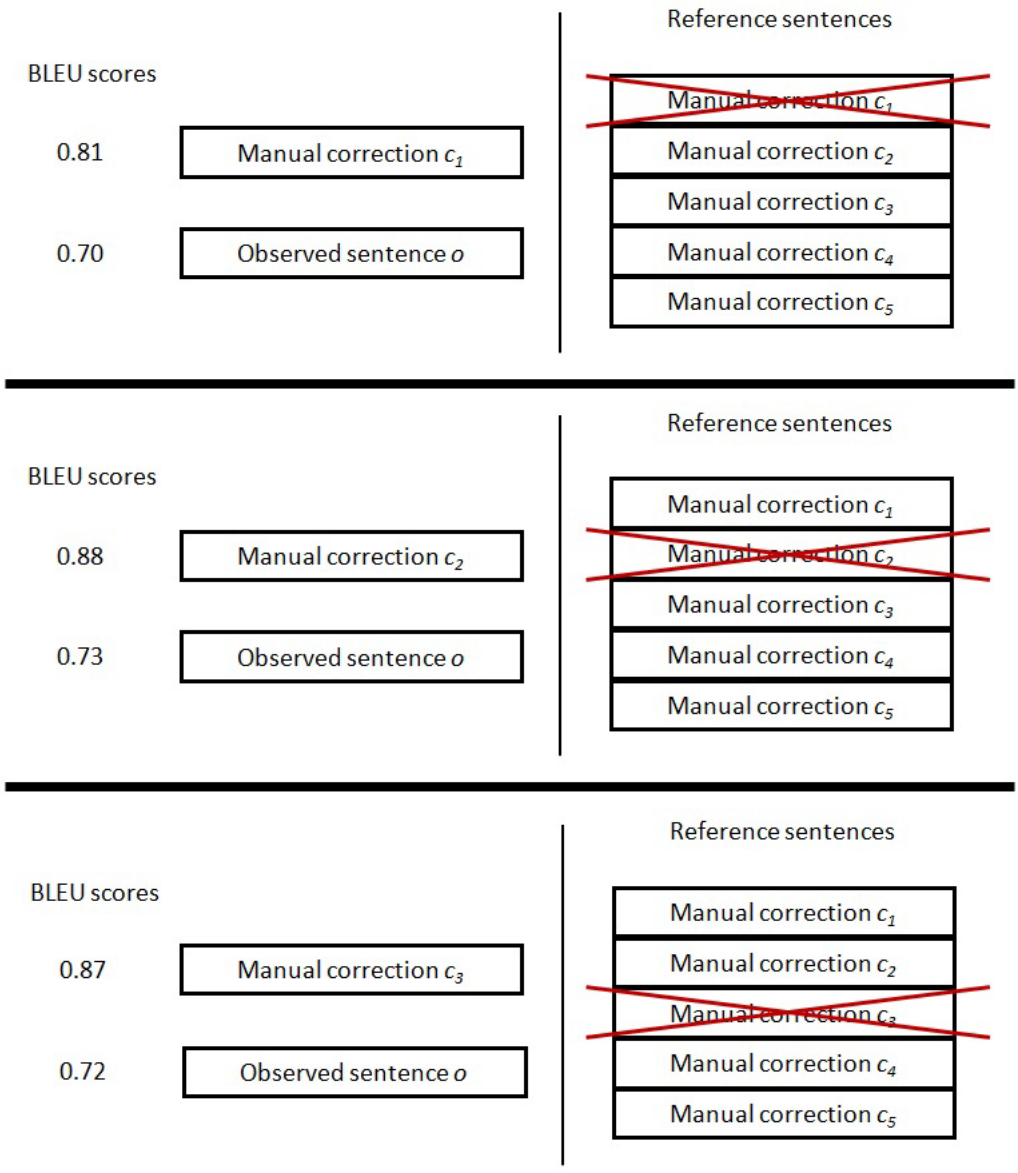


Figure 4.5: Evaulation validation method example for the first three reference sentences and the observed sentence. The top figure shows the BLEU scores being calculated on the observed sentence o and manual correction c_1, using c_2, c_3, c_4 and c_5 as reference sentences. The middle figure shows calculations of the BLEU score on the observed sentence and manual correction c_2, using c_1, c_3, c_4 and c_5 as reference sentences, and so on.

Chapter 5

Noise models

In this chapter, we present the various noise models used to emulate the mistakes ESL students make during English sentence production. Each noise model represents a type of common mistake made by ESL students. All noise models take the form of a wFST. As such, these noise models can be cascaded into a combined noise model simply by composing them together. To do this, we simply need to define an ordering of the noise models. This is significant in that we can combine any selection of noise models without much difficulty. Because making changes to the noise models is not necessary, this process is almost effortless, which allows us to integrate various noise models at will.

In our explanation of the models below, we make use of the expression '$w_1 \rightarrow w_2$'. This expression is used to signify that the input word w_1 is changed by the specified noise model to the output word w_2. We use the term *backtracking* to denote the creation of a model by setting the output values of the model to the input vocabulary of the succeeding noise model or observed sentence, as explained in Section 2.2.6. We use the terms *preceding model* and *following model* to denote the models on the composition chain which come right before and right after the current model. The language model will always be the first model in the chain, and the observed sentence wFST will always be the last model of the chain. For example, if we use a language model L, two noise models N_1 and N_2, and an observed sentence wFST S, and decide to order the models in the order $L \rightarrow N_1 \rightarrow N_2 \rightarrow S$, then the preceding model of N_1 is the language model L, and the

following model is N_2. We use the term *language vocabulary* to denote all words in our language model. The term *input vocabulary set* denotes the set of words or strings which are possible inputs of the current wFST, and *output vocabulary set* denotes the set of possible output words or strings of the current wFST. All parameters for our noise models were initialized so that the probability of making an error was 0.01. When multiple error types were possible at a given choice point in the transduction, the error probability was equally divided between each type.

5.1 Spelling errors

5.1.1 Model

The spelling error noise model accounts for spelling errors made by writers. For spelling errors, we allowed all spelling errors which were a Damerau-Levenshtein distance of 1 (Damerau, 1964; Levenshtein, 1966) and preserved the first letter. The DL is a string metric, or the "distance" between two strings, and is equal to the minimal number of operations needed to get from one string to the other. The possible operations are insertion, deletion, substitution and transposition (of adjacent characters). While allowing a DL distance of 2 or higher may likely have better performance, the model was constrained to a distance of 1 due to memory constraints. The first letter was also preserved for the same reason. We specified one parameter λ_n for each possible word length n. Each parameter is the total probability of making a spelling error for a given word length. For each word length we distributed the probability of each possible spelling error equally among all possible error results. For example, the probability of 'car→cae' has the same as the probability as 'car→can', which also has the same probability as 'car→carg' and 'pet→prt'. For word length n, and a word with no consecutively repeating characters, we have $n-1$ deletion errors, $25(n-1)$ substitution errors, $n-2$ transposition errors, and $25n+1$ insertion errors, and the probability for each possible error is $\frac{\lambda_n}{n-1+25(n-1)+n-2+25n+1}$. In the case of words with consecutive repeating letters, the repeating letters decrease the number of possible erroneous outcomes for deletion and transposition, and the denominator is adjusted accordingly. The

maximum word length for spelling errors is set to 22, giving us 22 parameters.

5.1.2 wFST construction

The construction of the wFST for the spelling error model is quite simple. The wFST is a one state wFST, which is both the start state and the end state. Using backtracking, we find every string in the input vocabulary set of the following model. For each word in the set, we create an arc for every possible word in the language vocabulary which may map to that string, by either being the same (as in having no spelling error) or having a DL distance of 1. The weight of each arc is determined by length of the input word and the corresponding parameter, and is calculated as specified above. For example, if the string 'cav' was found, arcs such as 'can→cav', 'car→cav', 'cave→cav', etc. would be added. The probability for the first two would be identical, since they both have the same length, and the probability of 'cave→cav' would be dependent on the parameter for four letter words.

5.1.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error noise model for 15 iterations. The resulting values for each parameter are shown in Table 5.1. Using these parameters, we found the maximum posterior probability corrections for the sentences in our evaluation set, as explained in Section 2.2.7. This resulted in sentences which achieved higher BLEU and METEOR scores in comparison to the original sentences, as shown in Table 5.2. We can see that the evaluation scores from METEOR and BLEU both increase when using the spelling error noise model to correct the sentences. To make sure the higher score of our corrections were not due to some random score increases, we checked the number of sentences which had score improvements, as well as the number of sentences which had a decrease in score. For BLEU, we can see that 48 out of the 1016 sentences had an increase in score, while only 3 had a decrease in score. The number of sentences with increasing/decreasing scores is similar for METEOR.

Table 5.1: Parameter values for spelling error noise model

Word length	Parameter value
1	0.00009289
2	0.0001053
3	0.0007390
4	0.0008533
5	0.002957
6	0.006895
7	0.01133
8	0.01115
9	0.01616
10	0.01285
11	0.01672
12	0.01817
13	0.01650
14	0.01551
15	0.01172
16	0.00000001077
17	0.001044
18	0
19	0
20	0
21	0
22	0

Table 5.2: Average evaluation scores for spelling error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling	0.721565	48	3	0.824764	45	4

We also compared the results of using the spelling error noise model to the output results of using the GNU Aspell 0.60.6 spelling checker, using our development data set. The development set was used to restrict any visual access, i.e. by the researcher, to the evaluation set. Since we are using METEOR and BLEU for our evaluation metric, we needed to obtain a set of corrected sentences from Aspell. Aspell lists the suggested spelling corrections of misspelled words in a ranked order, so we replaced each misspelled word found by Aspell with the word with the highest rank (lowest score) for the Aspell corrections. One difference between Aspell and our model is that Aspell only corrects words which do not appear in the dictionary, while our method looks at all words, even those found in the dictionary. Thus our model can correct words which look correct by themselves, but seem to be incorrect due to the bigram context. Another difference is that Aspell has the capability to split words, whereas our model does not allow the insertion of spaces.

The results of the experiment are shown in Table 5.3. We can see that our model has better performance, due to better word selection, despite the advantage that Aspell has by using phonological information to find the correct word, and the disadvantage that our model is restricted to spellings which are within a Damerau-Levenstein distance of 1. Another noticeable difference is that Aspell makes many incorrect 'corrections'. The higher score for our model is due to the fact that it is context-sensitive, and can use other information in addition to the misspelled word. For example, the sentence 'However , I couldn't concentrate on studing for my subjects because i always thought for him .' was edited in Aspell by changing 'studing' to 'studding', while our model correctly selected 'studying'. The sentence 'So I can reach the theater in ten minuets by foot' was not edited by Aspell, but our model changed 'minuets' to 'minutes'. Another difference that can be seen by looking through the results is that Aspell changes every word not found in the dictionary, while our algorithm allows words it has not seen by treating them as unknown tokens. Since we are using smoothing, these tokens are left in place if there is no other high probability bigram to take its place. This helps leave the proper nouns and words not found in the vocabulary intact.

Table 5.3: Aspell vs Spelling noise model

	BLEU	↑	−	↓	METEOR	↑	↓
ESL Baseline	0.679113				0.798580		
Spelling	0.687684	24	2	1	0.803293	25	1
Aspell	0.683646	26	14	22	0.801122	25	24

Looking into the actual results, we found the following statistics. 63 of the 504 sentences were edited by either Aspell or our model. We found that 26 of these sentences were edited by both Aspell and our spelling noise model, and 36 were edited only by Aspell, and 1 sentence was edited only by our model. Based on the BLEU scores of the edited sentences, 14 of the 26 sentences which were edited by both systems had the same scores. 12 of these sentences had improved scores, and 2 had the same score with respect to the original ESL sentence. Most of these sentences had the same edits. In only 3 of the 24 sentences did Aspell have better scores, compared to 9 sentences for our spelling noise model. Out of the 36 sentences edited only by Aspell, 10 sentences had improved scores, 9 had the same score, and 17 had a decreased scores. This shows that Aspell was not doing well in selecting the correct edit, compared to our system, which leaves the word alone if it cannot find a substitute with high enough probability.

5.2 Article choice errors

5.2.1 Model

The article choice error noise model simulates incorrect selection of articles. In this model we learn $n(n-1)$ parameters, one for each article pair. Since there are only 3 articles (*a, an, the*), we only have 6 parameters for this model (a→an, a→the, an→a, an→the, the→a, the→an).

Table 5.4: Calculated probabilities for article choice error noise model

Word length	Parameter value
the→the	0.95920
the→a	0.03970
the→an	0.00110
a→a	0.95790
a→the	0.04149
a→an	0.00061
an→an	0.69647
an→the	0.18577
an→a	0.11776

Table 5.5: Average evaluation scores for spelling and article error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Article	0.722032	53	4	0.825081	50	5

5.2.2 wFST construction

The construction of the wFST for the article choice error model is even simpler than the spelling error model. Again, the wFST is a one state wFST, which is both the start state and the end state. Using backtracking, we find every string in the input vocabulary set of the following model. For each word in the set, we create an arc for every word in the set, and set the probability weight to 1, with the exception of the 3 articles a, an and the. For each of these words, we create 3 arcs for each of the possible transitions, and assign the corresponding weights according to their parameters.

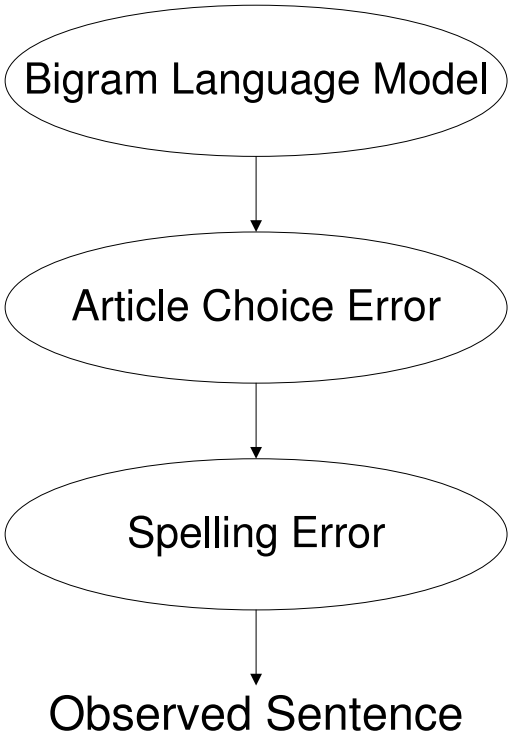


Figure 5.1: Full model used for article choice error test. The generation of a sentence, according to the model, starts with the language model generating a sentence in the language. This sentence is subjected to the article choice error model where article choice errors may be introduced, and the output is then subjected to the spelling error noise model. The output coming from the spelling error noise model is our observed sentence.

5.2.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error and article choice error noise model for 15 iterations. The order of the noise models was such that article choice error model was directly after the language model, and followed by the spelling error model, as shown in Figure 5.1. If spelling errors are introduced, other noise models further down the chain may not be able to introduce noise due to not being able to find the misspelled word. In reality, a grammatically erroneous word may also be misspelled, and thus the spelling error noise model is always put at the end of the chain when more than one noise model is used. The resulting values for each parameter are shown in Table 5.4. Using these parameters, we find the maximum posterior probability corrections for the sentences in our evaluation set, as explained in Section 2.2.7. This results in sentences which achieve higher BLEU and METEOR scores in comparison to the spelling model sentences, as shown in Table 5.5. We can see that the evaluation scores from METEOR and BLEU both increase when using the spelling error noise model to correct the sentences. Again, we checked the number of sentences which had score improvements, as well as the number of sentences which had a decrease in score. For BLEU, we can see that 53 out of the 1016 sentences had an increase in score, while only 4 had a decrease in score. The number of sentences with increasing/decreasing scores is similar for METEOR. Overall, we can see that our model is correcting some errors. Looking through a test run on the sentences in our development set, we found the following corrections. For the sentence input 'I thanked for the advertisement which makes me know a existence about various channel speakers.', our model correctly edits 'a' to 'the', improving the quality of the sentence. Other corrections we found include editing 'a important decision' to 'an important decision', and 'a entire stranger' to 'an entire stranger'. Counting through our development set sentences, we found that all 4 incorrect instances of using 'a' instead of 'an' were successfully corrected. A case of interest which was not corrected was the sentence 'If we living without nature then we must *make a oxgen* without a break.'. This is a case where two consecutive words have errors, an article insertion error followed by a spelling error. In this case, not only is article

Table 5.6: 12 most commonly misused prepositions by ESL writers used in preposition choice noise model

Prepositions list
of, in, for, to, by, with, at, on, from, as, about, since

insertion an error, but also the choice of the erroneous article can also be viewed as an error. In this case, we might expect our model to edit 'a' to 'an'. While this type of case could possibly be fixed by our system, in this particular case the bigram model did not give enough probability to 'make an oxygen without', causing our system to leave the phrase as it was. As for 'an', there was only one incorrect usage of an, in the case it was compounded with another error not covered by our system (two 'an's in a row), and thus was not corrected.

5.3 Preposition choice errors

5.3.1 Model

The preposition choice error noise model simulates incorrect selection of prepositions. We take the 12 most commonly misused prepositions by ESL writers (Gamon et al., 2009) and specify one parameter for each preposition pair, as we do in the article choice error noise model, giving us a total of $12 \times 11 = 132$ parameters. The prepositions are listed in Table 5.6.

5.3.2 wFST construction

The construction of the wFST for the preposition choice error model is identical with the article choice error model, except instead of the article arcs, we do the same process for the preposition arcs.

Table 5.7: Calculated probabilities for preposition choice error noise model

from\to	-	of	in	for	to	by	with	at	on	from	as	about	since
of	0.9415	-	0.0137	0.0073	0.0066	0.0038	0.0055	0.0013	0.0017	0.0074	0.0021	0.0088	0.0004
in	0.8844	0.0259	-	0.0169	0.0164	0.0083	0.0121	0.0057	0.0053	0.0099	0.0058	0.0081	0.0013
for	0.8195	0.0442	0.0324	-	0.0349	0.0112	0.0187	0.0069	0.0047	0.0089	0.0061	0.0101	0.0024
to	0.9806	0.0032	0.0035	0.0035	-	0.0017	0.0023	0.0005	0.0006	0.0013	0.0018	0.0009	0.0001
by	0.8043	0.0292	0.0384	0.0266	0.0267	-	0.0289	0.0032	0.0053	0.0168	0.0090	0.0096	0.0021
with	0.8382	0.0262	0.0425	0.0188	0.0251	0.0091	-	0.0057	0.0044	0.0088	0.0078	0.0117	0.0017
at	0.4997	0.0891	0.2288	0.0413	0.0382	0.0111	0.0318	-	0.0128	0.0188	0.0101	0.0173	0.0010
on	0.5566	0.0596	0.1246	0.0456	0.0658	0.0156	0.0491	0.0096	-	0.0326	0.0125	0.0259	0.0023
from	0.6889	0.0617	0.0730	0.0331	0.0516	0.0226	0.0268	0.0078	0.0067	-	0.0071	0.0189	0.0017
as	0.8543	0.0131	0.0319	0.0199	0.0124	0.0060	0.0114	0.0036	0.0145	0.0074	-	0.0079	0.0177
about	0.8367	0.0229	0.0263	0.0133	0.0194	0.0189	0.0336	0.0030	0.0061	0.0155	0.0027	-	0.0016
since	0.6622	0.0245	0.0849	0.0340	0.0094	0.0119	0.0100	0.0087	0.0565	0.0269	0.0661	0.0049	-

5.3.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error and preposition choice error noise model for 15 iterations. The order of the noise models was such that preposition choice error model was directly after the language model, and followed by the spelling error model. The resulting values for each parameter are shown in Table 5.7. Using these parameters, we found the maximum posterior probability corrections for the sentences in our evaluation set, as explained in Section 2.2.7. This resulted in sentences which achieved lower BLEU and METEOR scores in comparison to the spelling model sentences, as shown in Table 5.9, signaling decreased performance. We can see that the evaluation scores from METEOR and BLEU both decrease compared to using the spelling error noise model alone to correct the sentences. Again, we checked the number of sentences which had score improvements, as well as the number of sentences which had a decrease in score. For BLEU, we can see that 52 out of the 1016 sentences had an increase in score, which is an increase of 4 sentences over using just the spelling noise model. However, 7 additional sentences were changed for the worse when adding in the preposition noise model. The number of sentences with increasing/decreasing scores is similar for METEOR. Overall, we can see that our model is correcting some errors, but introducing more errors than it is correcting. A test run on the sentences on our development set returned the corrections in Table 5.8. We can see that in the first five examples, the noise model is making incorrect changes. Looking only at the bigram context, the changes look probable, but using the sentence as a whole, we can see that the changes are incorrect. Thus the poor performance of the preposition model is due to the lack of sufficient context information for choosing the correct preposition. The last three preposition changes improve the sentences, showing that this model does have limited success in correcting preposition errors. Judging from the need of context for prepositions, a better language model may significantly increase the performance of this error model.

Chapter 5, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceed-

Table 5.8: Original and fixed sentence pairs with preposition changes using the spelling and preposition error noise model on the development set. The original observed sentences are denoted S_n, and the corresponding fixed version is denoted $\overline{S}_n$, where n is the sentence number

S_1	Distrust about desire between two have been growing in their ...
$\overline{S}_1$	Distrust *of* desire between two have been growing in their ...
S_2	My seat was so remotely located from the stage that...
$\overline{S}_2$	My seat was so remotely located *on* the stage that...
S_3	... and there were many examples about effect on secondhand smoke.
$\overline{S}_3$	...and there were many examples *in* effect on secondhand smoke.
S_4	Some people say that listening to the advice of the family ...
$\overline{S}_4$	Some people say that listening to the advice *on* the family ...
S_5	... rely on parents more and more by accomplishing something alone.
$\overline{S}_5$	... rely on parents more and more *for* accomplishing something alone.
S_6	At that time, there were so many sources to handle in internet that I ...
$\overline{S}_6$	At that time, there were so many sources to handle *on* internet that I ...
S_7	... he only concentrate in math.
$\overline{S}_7$	... he only concentrate *on* math.
S_8	... makes me know a existence about various channel speakers.
$\overline{S}_8$	... makes me know a existence *of* various channel speakers.

Table 5.9: Average evaluation scores for spelling and preposition error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Preposition	0.721451	52	10	0.824450	49	12

ings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

Chapter 6

More noise models

In this chapter, we present some of the more complicated noise models. We use the same terminology as defined in the previous chapter.

6.1 Wordform choice errors

6.1.1 Model

The wordform choice error noise model simulates choosing the incorrect wordform of a word. For example, choosing the incorrect tense of a verb (e.g. *went*→*go*), or the incorrect number marking on a noun or verb (e.g. *are*→*is*) would be a part of this model. This error model has one parameter for every number of possible inflections, up to a maximum of 12 inflections. This results in 11 parameters, since there is no need for a parameter for words with only one inflection, i.e. itself. For example, the word 'swim' has the inflections swim, swims, swam, swimming, swum. Since it has 4 inflections other than itself, in our wordform choice noise model 'swim' would be associated with the 4^{th} parameter. Each parameter is the total probability of choosing the wrong inflection of a word, and the probability is spread evenly between each possible inflection. For example, if the 4^{th} parameter had a value of 0.04, then the probability of swim→swim would be $1 - 0.04 = 0.96$, and the probability of swim→swims, swim→swam, swim→swimming, swim→swum would each be $0.04/4 = 0.01$. We used CELEX

Table 6.1: Parameter values for wordform error noise model

Number of forms	Parameter value
1	0.0
2	0.020143
3	0.00360
4	0.01334
5	0.00970
6	0.00308
7	0.00027
8	0.00450
9	0.00033
10	1.0
11	0.00112
12	0.00049

(Baayen et al., 1995) to find all the possible wordforms of each observed word.

6.1.2 wFST construction

The construction of the wFST for the wordform choice error model is similar to the spelling error model. The wFST is a one state wFST, which is both the start state and the end state. Using backtracking, we find every word in the input vocabulary set of the following model. For each word in the set, we create an arc for every possible word in the language vocabulary which may map to that string by being some form of inflection of that word. The weight of each arc is determined by the number of inflections of the input word of the arc, and is calculated as specified above.

6.1.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error and wordform choice error noise model for 15 iterations. The order of the noise models was such that the wordform choice error model was directly after the language model, and followed by the spelling error model. The resulting values for each parameter are shown in Table 6.1. We can see that as the number of

Table 6.2: Average evaluation scores for spelling and wordform error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Wordform	0.722004	69	15	0.824976	67	16

wordforms increases, there is a trend of having a smaller probability of using the incorrect wordform. One oddity which may catch your eye is the fact that the parameter value for words with 10 inflections is 1. This is due to the fact that there is only one word which actually has 10 inflections. Because this word never appeared in any of our training sentences, but some of its inflections did appear, the parameter value resulted in a value of 1. We note that placing a prior on the noise model parameters and using maximum a posteriori (MAP) estimation would remedy this phenomenon, and would be one way of expanding on this work. Using these parameters, we found the maximum posterior probability corrections for the sentences in our evaluation set, as explained in Section 2.2.7. This resulted in sentences which achieved improved BLEU and METEOR scores in comparison to the spelling model sentences, as shown in Table 6.2, signaling better performance. Again, we checked the number of sentences which had score improvements, as well as the number of sentences which had a decrease in score. For BLEU, we can see that 69 out of the 1016 sentences had an increase in score, which is an increase of 21 sentences over using just the spelling noise model. The number of sentences with decreased BLEU increased by 12 sentence in comparison to just using the spelling noise model. The number of sentences with increasing/decreasing scores is similar for METEOR. Overall, we can see that our model is correcting more errors than it is making, and our overall BLEU and METEOR scores are better than using only the spelling model. To get a feel for the actual changes, we ran a test run on the sentences on our development set. The results are shown in Table 6.3.

Table 6.3: Original and fixed sentence pairs with wordform changes using the spelling and wordform error noise model on the development set. The original observed sentences are denoted S_n, and the corresponding fixed version is denoted $\overline{S}_n$, where n is the sentence number

S_1	As I mention above, I agree the statement that we ...
$\overline{S}_1$	As I *mentioned* above, I agree the statement that we ...
S_2	In Korea student spend much money in bookstores, game facilities , ...
$\overline{S}_2$	In Korea *students* spend much money in bookstores, game facilities , ...
S_3	... daughters who want to get marry with foreigners.
$\overline{S}_3$	... daughters who want to get *married* with foreigners.
S_4	Maybe there is people who oppose to live in the big city ...
$\overline{S}_4$	Maybe there is people who *opposed* to live in the big city ...
S_5	... due to the feelings including satisfactions from win, prides from goal.
$\overline{S}_5$	... due to the feelings including satisfactions from win, *pride* from goal.
S_6	First of all, the more time spend with, the more influence give and take.
$\overline{S}_6$	First of all, the more time *spent* with, the more influence give and take.
S_7	... shows an idea that a person expresses by putting himself into ...
$\overline{S}_7$	... shows an idea that a person *expressed* by putting himself ...
S_8	... can evaluate students and the student can also appraise teachers.
$\overline{S}_8$	... can evaluate students and the student can also *appraising* teachers.
S_9	If their contents are more popular, they can entice more advertisements.
$\overline{S}_9$	If their contents are more popular, they *could* entice more advertisements.

The results in Table 6.3 are ordered from best score increase to worst score decrease, by ratio of the BLEU scores. We can see that the corrections to S_1, S_2, and S_3 are correct. For S_4, we can see that there are many mistakes in the original sentence. A proper correction might be 'Maybe there are people who are opposed to living in the big city . . . '. Our model has failed to correct 'is' to 'are', and 'live' to 'living', but it has identified and changed 'oppose' to 'opposed', increasing the BLEU and METEOR scores. If we look at the partial phrase 'to live in', we can see that this phrase, without the preceding context, seems to be a legit phrase. The probability of this phrase using a bigram model is not something that our model would calculate to be very low in comparison with 'to living in'. Thus our model leaves 'to live in' unchanged. Using the bigram model as our language model does not give us enough information to fix this error. This is also seems true for sentence S_8, in which the change being made is ignorant of the context falling outside of the bigram window which should restrict it. For sentence S_6, the actual meaning of the sentence itself is not quite clear. For S_7 and S_9, it is not quite clear whether the changes are for the better, but they both seem to be acceptable. The BLEU and METEOR scores, however, decreased, due to the fact that the task used to collect evaluation sentences specified to use as much of the original sentence as possible. Through the examples, we can see how using the bigram model restricts the extent of our corrections, and also how valid changes to words which do not need changing decreases our METEOR and BLEU scores. We posit that using a better language model will likely increase the model's performance.

6.2 Word insertion errors

6.2.1 Model

The word insertion error noise model simulates the addition of extraneous words to the original sentence. For this error model, one consideration which we need to take into account is the bigram language model's bias toward shorter sentences. Because the bigram model gives higher probabilities to sentences with fewer words, we find that allowing all words to be possible extraneous words results

in the model correcting sentences by removing far too many words from each sentence. Thus, instead of allowing all words to be inserted, we create a list of words which we shall call the extraneous word set, by combining the prepositions and articles found in the article choice and preposition choice errors. We assume that each of the words on the list has a probability of being inserted erroneously. There is a parameter for each word, which is the probability of that word being inserted. Thus we have a total of 15 parameters for this noise model. Only one word can be inserted between existing words.

6.2.2 wFST construction

The construction of the wFST for the insertion error model is not complicated. The wFST is a two state wFST, the start state and the insertion state. Both states are end states. Using backtracking, we find every string in the input vocabulary set of the following model. For each word in the set, we create an arc from the start state to itself, and set the probability weight to be $1 - (\textit{total insertion probabilities})$. In addition to this arc, we also create an arc from the insertion state to the start state, with a probability weight of 1. These arcs guarantee that a max of one word is inserted between words. Finally, we add an arc for each word in our extraneous word set between the start state and the insertion state. The input value of the arc is set to the empty string, and the output value is set to the current word. The probability weight is set to be the corresponding parameter.

6.2.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error and word insertion error noise model for 15 iterations. The order of the noise models was such that the word insertion error model was directly after the language model, and followed by the spelling error model. The resulting values for each parameter are shown in Table 6.4. The word insertion error model was restricted to articles and 12 prepositions, and thus did not make many changes,

Table 6.4: Parameter values for word insertion error noise model

Word	Parameter value
the	0.000826
a	0.000524
an	0.000016
of	0.000130
in	0.000342
for	0.000092
to	0.000262
by	0.000067
with	0.000121
at	0.000072
on	0.000122
from	0.000092
as	0.000130
about	0.000186
since	0.000046

Table 6.5: Average evaluation scores for spelling and word insertion error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Insertion	0.722218	53	3	0.824928	49	5

Table 6.6: A couple of examples of original and fixed sentence pairs with extraneous word removal using the spelling and insertion error noise model on the development set. The original observed sentences are denoted S_n, and the corresponding fixed version is denoted $\overline{S}_n$, where n is the sentence number

S_1	: Thus, in order to have good grades, people attend the classes mandatorily.
$\overline{S}_1$	: Thus, in order to have good grades, people attend classes mandatorily.
S_2	: ...that people attend a big event and a exciting place.
$\overline{S}_2$	: ...that people attend a big event and exciting place.

but increased the BLEU and METEOR scores when it did, as seen in Table 6.5. A couple of examples from our development set can be found in Table 6.6. As the sample set is too small, it is hard to make an analysis of the performance of this model using the limited examples. The lack of abundance of corrections is likely due to the very tight restriction on the type of words this model includes. One thing to note is that since we are using a bigram language model, the language model itself is biased toward shorter sentences. Since we only included words which were needed when they were used, we did not run into problems with this bias. When we tried including a large set of commonly used words, we found that a very large number of words from the observed sentences were being erased because of the bigram model's probabilistic preference for shorter sentences. If a different language model, one which is not biased toward shorter sentences, is used, it may be possible to include a much larger set of words, which may increase the performance of this noise model by increasing the number of corrections it can make.

6.3 Word deletion errors

6.3.1 Model

The word deletion error noise model models the opposite phenomenon of the word insertion noise model, i.e. the error of omitting a word. The word deletion

noise model has just one parameter, which corresponds to the probability of a word being deleted. Because of memory constraints, we put two restrictions on our word deletion error noise model. The first restriction is that if any word is deleted, the following word is not eligible for deletion. In other words, consecutive words may not be deleted simultaneously. The second restriction is that, based on the previous word, only the top 1,000 following words based on the frequency of the bigram counts from the BNC can be deleted. For example, given the sentence 'The boy went home .', and assuming that 'boy' was not deleted, 'went' would have a probability of being deleted only if it was one of the top 1,000 words that follow the word 'boy' in terms of counts, per the BNC. These two restrictions keep the size of our noise model tractable, which we will explain in more detail below. Each of the words which were deemed eligible candidates for deletion had the same probability of being deleted, equal to the lone parameter of this model.

Let us consider the case of a deletion error noise model without the above restrictions. If there is no restriction on consecutive word deletions, the number of paths which can lead to any observed sentence would be infinite. For example, the sentence 'I can go .' may be derived from the sentence 'I can not go .' or it could be derived form 'I can not not go .' or from 'I can not not not go .' and so on. This would make our training problem much more time consuming, with little perceived benefit. Instead, we restricted deletions of consecutive words. This means that given an observed sentence with n words, the sentence can be derived from a sentence with at most $2 \times (n + 1)$ words, making the number of possible original sentences finite. If we add no other restrictions, for each of the $n+1$ places in which a deleted word may have existed we would have to take into account every possible word in the language vocabulary. This brings about two effects: the input to the word deletion noise model will be every possible word in the vocabulary, and the number of arcs from each observed word context state will be the size of the vocabulary. The first effect makes noise models higher up on the chain much larger, as their output vocabulary must match the input vocabulary of the lower level noise model, and this trickles up to the language model, which must then contain all words in the vocabulary, making the language model much larger than

desirable. The second effect makes our noise model quite large, especially when combined with other models such as the spelling noise model which is already adding a lot of new words into higher level noise models. Thus, to stave off this huge increase of noise model vocabulary size, we put a restriction on the words which may be deleted given the previous context. The restriction, as stated above, only allows the top 1,000 words that follow the previous context word to be deleted. This reduces the number of arcs from each observed word state to 1,000, which is much less than the size of the language vocabulary, and eliminates the need for the language model to include every possible word.

6.3.2 wFST construction

To construct the word deletion error model, we first use backtracking to find every string in the input vocabulary set of the following model. Using this set, we create a start state, an epsilon state, and a state for every word in the input vocabulary set. All word states and the epsilon state are end states. To insert the arcs for which no deletion has occurred, we do the following. For each state s, including the start state and the epsilon state, we add a new arc for each word w in the input vocabulary set. Given state s and word w, we create an arc from that s to the word state of w, setting the input and output values to w. If the current word is not one of the top 1,000 following words given the word context of s, we set the probability weight to 1. If the word is one of the top 1,000 then we set the probability weight to $1 - \textit{parameter value}$. Note that it does not really matter how many words we have decided to use, since our parameter refers to the probability of any deletable word being deleted. If we only have the top 100 words or would like to do the same for the top 2,000 words following one of the words, the process would be just the same, and we would set all deletion contestants to have a probability weight of $1 - \textit{parameter value}$. The epsilon state has no word context, and thus all arcs from the epsilon state have a probability weight of 1. Finally, we add the arcs representing word deletion errors by doing the last step. For each state s, excluding the epsilon state, we add arcs from the current state to the epsilon state for each of the top 1,000 words following the current state word.

Table 6.7: Parameter value for word deletion model

Parameter value
0.000003251

Table 6.8: Average evaluation scores for spelling and word deletion error noise model run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Deletion	0.721565	48	3	0.824764	45	4

The input word of the arc is the current word of the 1,000 words, and the output word is the empty string. The weight of each of the arcs is set to the parameter value, which is equal to the probability of a word being deleted. For example, for the word state of *go*, if *back* was one of the top 1,000 words following *go*, and the parameter was 0.03, then we would add an arc from word state *go* to the epsilon state with the input word as *back*, the output word as ε, and the weight to be 0.03.

6.3.3 Results

Using a set of 10,000 sentences from the training set, we trained our spelling error and deletion error noise model for 15 iterations. The order of the noise models was such that word deletion error model was directly after the language model, and followed by the spelling error model. The resulting value for the parameter is shown in Table 6.7. As mentioned in the previous section, one thing to note is that since we are using a bigram model for the language model, the model itself is biased toward shorter sentences. This means that the model generally gives higher probability to shorter sentences, and thus will favor not having deletions than to having them, due to the language model. Thus, for our model to detect a deletion error of some word w_d between words w_n and w_{n+1}, given the probability

of deletion p_d, we find that $p(w_d|w_n) \times p(w_{n+1}|w_d) \times p_d$ must be greater than $p(w_{n+1}|w_n) \times (1 - p_d)$. Since the probability value found in Table 6.7 is also very small, this means that $p(w_d|w_n) \times p(w_{n+1}|w_d)$ must be much greater than $p(w_{n+1}|w_n)$ to be true, which is not something which occurs often. In Table 6.8, we can see that our deletion model does nothing to change the sentences, presumably because of the bias of our language model toward shorter sentences.

6.4 Combined models

6.4.1 Model

In addition to defining various new noise models, one of the advantageous characteristics of our system is that we can compose various noise models together into one, as we have already done with our various noise models and the spelling error noise model. We are not restricted to just 2 noise models, and may compose as many noise models as we wish, creating noise models which model various types of error simultaneously (or to be more exact, in a specified order on the given sentence, allowing multiple types of errors). We are only constrained by the computational feasibility of the composition of the models. To see the capabilities of composing various noise models together, we composed a set of four noise models together and evaluated the performance. We chose the models from the previously described noise models, allowing all noise models which had an increase in evaluation scores over the ESL baseline, and in cases where the spelling error noise model was composed, we used the spelling error noise model as the baseline. The chosen noise models were the spelling error noise model, the article choice error noise model, the wordform choice error noise model and the word insertion noise error model.

6.4.2 wFST construction

To construct the wFST for the combined model, we simply construct each of the separate noise models as previously described, and compose the noise models

	BLEU	↑	↓	METEOR	↑	↓
ESL Baseline	0.715634			0.821000		
Spelling only	0.721565	48	3	0.824764	45	4
Spelling & Article	0.722032	53	4	0.825081	50	5
Spelling & Preposition	0.721451	52	10	0.824450	49	12
Spelling & Wordform	0.722004	69	15	0.824976	67	16
Spelling & Insertion	0.722218	53	3	0.824928	49	5
Spelling & Deletion	0.721565	48	3	0.824764	45	4
Combined models	0.723113	79	18	0.825508	75	18

Table 6.9: Average evaluation scores for various noise models run on 1016 sentences, along with counts of sentences with increased (↑) and decreased (↓) scores.

together, resulting in our combined noise model. Before we can compose the noise models, we need to decide on an ordering. For our combined model, we combined the models in the order 'word insertion → wordform error → article choice error → spelling error model'.

6.4.3 Results

Using a set of 10,000 sentences from the training set, we trained our combined noise model for 30 iterations. We have already specified the ordering in the previous section.

The results of the combined model are shown in Table 6.9, along with the scores of the previous models. We can see that the combined noise model shows the best performance thus far, having the highest BLEU and METEOR scores. The combined noise model makes the corrections for various also has the largest number of sentences with increased BLEU and METEOR scores.

Chapter 6, in part, is a reprint of the material as it appears in Automated Whole Sentence Grammar Correction Using a Noisy Channel Model in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL-HLT 2011), Y. Albert Park and Roger Levy. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Conclusion

We have introduced a novel way of finding grammar and spelling corrections, which uses the EM algorithm to train the parameters of our noisy channel approach. One of the benefits of our approach is that it does not require a parallel set of erroneous sentences and their corrections. This allows us to train on a set of unedited texts, which are much more readily available. Depending on the domain one wishes to train on, this may often be quite feasible as we have shown by our collection of Korean ESL students' writing samples. Our data set could easily have been for ESL students of any native language, or could also be a data set of other groups such as young native English speakers, or the whole set of English speakers for grammar correction. Data sets from testing agencies of foreign language skill such as ETS, or essays collected by various educational language centers such as those in universities created to help foreign students may be used to train our model, if made available. Using these data sets, we can train our noisy channel model, as we have shown using a bigram language model, and a wFST for our noise model. We have also shown how to use weighted finite-state transducers and the expectation semiring, as well as wFST algorithms implemented in OpenFST to train the model using EM. Another advantage of our approach is that our model is not confined to a specific error type. As long as an error model can be built using a wFST, multiple error models may be added on. All that is needed is an extra composition step for each model. We have shown that composing various error models increases the performance of our model. Since our approach finds

the highest probability path over all possible sentences which are found by backtracking from the observed sentence, multiple errors can be fixed in one sentence. Even better, errors which are compounded on each other and cannot be solved by looking at just one type of error can be fixed by our system.

For evaluation, we have introduced a novel way of evaluating grammar corrections, using MT evaluation methods, that we have not seen in other grammar correction literature. The introduction of new corpora for the field of grammar correction during the last couple of years is very promising. These type of corpora are valuable because they can be used to find precision and recall values for grammar correction systems, and can be used to pinpoint the performance on specific types of errors. Unfortunately, there are only a few languages for which such corpora are currently available, and the corpora are very limited in the scope of the producers. Further research in languages for which such corpora have not been developed could make use of our evaluation technique.

We wish to note that the machine learning community has also been pondering over the problem of evaluating machine translation output, and this is an ongoing area of research. Various types of metrics have been proposed, such as BLEU, METEOR, NIST (Martin and Pryzbocki, 2006), TER (Snover et al., 2006), ROUGE (Lin and Och, 2004), etc. Research has also been done on developing metrics for evaluating the various proposed metrics agreement with human judgment, called meta-metrics, to see which is a better metric for evaluating machine translation output (Callison-Burch et al., 2007; Zhang and Vogel, 2010; Amigó et al., 2006). While BLEU currently seems to be one of the most popular metrics used in the machine translation field, research is continuously being done in search of better methods of evaluation, and we believe better metrics may appear in the future. The main concept we wish to emphasize is that of the parallels between the task of machine translation and grammar correction. Due to these parallels, future advances in evaluation techniques by the machine learning community could also be put to use for evaluating grammar correction performance, in the way we have done.

The experiment results have shown that our model does succeed in correct-

ing some of the erroneous sentences, but it is far from perfect. Further examination of the produced corrections shows the restrictions of using a bigram language model. We have seen that the bigram model does not carry enough contextual information to correct various grammatical errors that are dependent on words that are outside of the bigram context window. Using a language model which captures these types of dependencies may allow our system to identify more corrections, and make better quality corrections with less mistakes. For example, the sentence 'I prefer being indoors to going outside' (s_1) was incorrectly changed to 'I prefer being indoors to *go* outside' (s_2). Unfortunately, the bigram model gives a much higher probability to 'to go outside' than to 'to going outside', since it does not take the structure of the sentence into consideration. Using a syntactic language model which is aware of the sentence structure, and also of phrasal structures such as 'prefer NP to NP' so that it assigns a much higher probability to s_1 over s_2 would eliminate the change of the original sentence to the erroneous 'I prefer being indoors to go outside.'. For example, if the language model generates s_1 with probability 0.05, whereas s_2 is generated with a probability of 0.0008, and the probability of the noise model changing *go* to *going* is 0.01, whereas the probability of *going* not being changed is 0.9, then we can see that $p(s_1 : s_1) = 0.045$, which is much greater than $p(s_2 : s_1) = 0.000008$. Thus, our system would not have made the erroneous correction. Having a better language model will increase the accuracy of our estimated probabilities for each sentence, which should increase performance of the overall system. Some may argue that for speech recognition systems, which often use a noisy channel approach, n-gram models currently seem to work better than syntactic models, and this may also be true for grammar correction. But there is a large difference in the underlying quality of the data for speech recognition and grammar correction. In speech recognition, figuring out what words a sound wave maps into is in itself a task, where as in grammar correction, we already start out with a somewhat understandable sentence, and already have a list of words. Thus we are starting from a drastically different starting point, in which using syntactic information is much more feasible, as we have seen our example.

Another area in which it may be possible to increase our system's perfor-

mance is the structure of the noise models. Our current noise models are very simple, and have a minimal amount of parameters. Adding more parameters and increasing the complexity of the noise models may also lead to enhancements of our model. For example, adding various parameters for each type of spelling error may capture a much better representation of the mistakes a writer will make. Increasing model complexity does not come without challenges. Computational constraints must be taken into account, as wFSTs can become very large (as has already happened in our models, which required over 100GB to run). Our models have already run into memory size problems that have required optimization, and the use of machines with large memory capacity.

Our current approach lays the framework for using a noisy channel approach to the task of automated grammar correction. We have shown that this approach is feasible and tractable, and produces results which improve the writing of Korean ESL students.

Bibliography

Ahmad, F. and Kondrak, G. (2005). Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 955–962, Stroudsburg, PA, USA. Association for Computational Linguistics.

Allauzen, C. and Mohri, M. (2003). Efficient algorithms for testing the twins property. In *Journal of Automata, Languages and Combinatorics*, volume 8(2), pages 117–144.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer. `http://www.openfst.org`.

Amigó, E., Giménez, J., Gonzalo, J., and Màrquez, L. (2006). Mt evaluation: human-like vs. human acceptable. In *Proceedings of the COLING/ACL on Main conference poster sessions*, COLING-ACL '06, pages 17–24, Stroudsburg, PA, USA. Association for Computational Linguistics.

Baayen, H. R., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database. Release 2 (CD-ROM).* Linguistic Data Consortium, University of Pennsylvania, Philadelphia, Pennsylvania.

Bell, T. C., Cleary, J. G., and Witten, I. H. (1990). *Text Compression.* Prentice Hall, Englewood Cliffs, NJ.

Bond, F. and Ikehara, S. (1996). When and how to disambiguate? - countability in machine translation.

Bond, F., Ogura, K., and Kawaoka, T. (1995). Noun phrase reference in japanese-to-english machine translation. In *In Sixth International Conference on Theoretical and Methodological Issues in Machine Translation: TMI-95*, pages 1–14.

Brill, E. and Moore, R. C. (2000). An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association*

for Computational Linguistics, ACL '00, pages 286–293, Stroudsburg, PA, USA. Association for Computational Linguistics.

Brockett, C., Dolan, W. B., and Gamon, M. (2006). Correcting ESL errors using phrasal SMT techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 249–256, Morristown, NJ, USA. Association for Computational Linguistics.

Callison-Burch, C., Fordyce, C., Koehn, P., Monz, C., and Schroeder, J. (2007). (meta-) evaluation of machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 136–158, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chen, S. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Technical report, Computer Science Group, Harvard University.

Chodorow, M. and Leacock, C. (2000). An unsupervised method for detecting grammatical errors. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 140–147, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced good-turing and deleted estimation methods for estimating probabilities of english bigrams. *Computer Speech and Language*, 5:19–54.

Cucerzan, S. and Brill, E. (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*, pages 293–300.

Dahlmeier, D. and Ng, H. T. (2011). Grammatical error correction with alternating structure optimization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 915–923, Portland, Oregon, USA. Association for Computational Linguistics.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7:171–176.

De Felice, R. and Pulman, S. G. (2008). A classifier-based approach to preposition and determiner error correction in L2 English. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 169–176, Morristown, NJ, USA. Association for Computational Linguistics.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):pp. 1–38.

Désilets, A. and Hermet, M. (2009). Using automatic roundtrip translation to repair general errors in second language writing. In *Proceedings of the twelfth Machine Translation Summit*, MT Summit XII, pages 198–206.

Dreyer, M., Smith, J., and Eisner, J. (2008). Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 1080–1089, Honolulu, Hawaii. Association for Computational Linguistics.

Duan, H. and Hsu, B.-J. P. (2011). Online spelling correction for query completion. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 117–126, New York, NY, USA. ACM.

Eeg-olofsson, J. and Knutsson, O. (2003). Automatic grammar checking for second language learners - the use of prepositions. In *In Nodalida*.

Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–8, Philadelphia.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, MA ; London.

Gale, W. A., Church, K. W., and Yarowsky, D. (1993). A method for disambiguating word senses in a large corpus. In *Computers and the Humanitzes*, volume 26, pages 415–439.

Gamon, M., Gao, J., Brockett, C., and Klementiev, R. (2008). Using contextual speller techniques and language modeling for esl error correction. In *In Proceedings of IJCNLP 2008*.

Gamon, M., Leacock, C., Brockett, C., Dolan, W. B., Gao, J., Belenko, D., and Klementiev, A. (2009). Using statistical techniques and web search to correct ESL errors. In *Calico Journal*, Vol 26, No. 3, pages 491–511, Menlo Park, CA, USA. CALICO Journal.

Golding, A. R. (1995). A bayesian hybrid method for context-sensitive spelling correction. In *In Proceedings of the Third Workshop on Very Large Corpora*, pages 39–53.

Golding, A. R. and Roth, D. (1999). A winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3):107–130.

Golding, A. R. and Schabes, Y. (1996). Combining trigram-based and feature-based methods for context-sensitive spelling correction. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 71–78, Stroudsburg, PA, USA. Association for Computational Linguistics.

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3 and 4):237–264.

Han, N.-R., Chodorow, M., and Leacock, C. (2006). Detecting errors in english article usage by non-native speakers. *Nat. Lang. Eng.*, 12(2):115–129.

Heine, J. E. (1998). Definiteness predictions for japanese noun phrases. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, pages 519–525, Stroudsburg, PA, USA. Association for Computational Linguistics.

Izumi, E., Uchimoto, K., and Isahara, H. (2004a). The NICT JLE corpus exploiting the language learnersspeech database for research and education. In *International Journal of the Computer, the Internet and Management*, volume 12(2), pages 119–125.

Izumi, E., Uchimoto, K., and Isahara, H. (2004b). Sst speech corpus of Japanese learners' English and automatic detection of learners' errors. *ICAME (International Computer Archive of Modern and Medieval English) Journal*, 28:31–48.

Izumi, E., Uchimoto, K., Saiga, T., Supnithi, T., and Isahara, H. (2003). Automatic error detection in the Japanese learners' English spoken data. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 2*, ACL '03, pages 145–148, Morristown, NJ, USA. Association for Computational Linguistics.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of markov source parameters from sparse data. In *In Proceedings of the Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam, The Netherlands: North-Holland.

Jones, M. P. and Martin, J. H. (1997). Contextual spelling correction using latent semantic analysis. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, pages 166–173, Stroudsburg, PA, USA. Association for Computational Linguistics.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Singal processing*, volume ASSP-35(3), pages 400–401.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, pages 181–184.

Knight, K. and Chander, I. (1994). Automated postediting of documents. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, AAAI '94, pages 779–784, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Kuich, W. and Salomaa, A. (1986). *Semirings, automata, languages*. EATCS monographs on theoretical computer science. Springer-Verlag.

Kwon, Y. H., Lee, M. H., and Kim, S.-R. (2009). Effective spelling correction in web queries and run-time db construction. In *Proceedings of the 2009 International Conference on Hybrid Information Technology*, ICHIT '09, pages 581–586, New York, NY, USA. ACM.

Lavie, A. and Agarwal, A. (2007). Meteor: an automatic metric for MT evaluation with high levels of correlation with human judgments. In *StatMT '07: Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231, Morristown, NJ, USA. Association for Computational Linguistics.

Lee, J. and Seneff, S. (2006). Automatic grammar correction for second-language learners. In *Proceedings of Interspeech*.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710.

Li, M., Zhang, Y., Zhu, M., and Zhou, M. (2006). Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 1025–1032, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lidstone, G. J. (1920). Note on the general case of the bayes-laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192.

Lin, C.-Y. and Och, F. J. (2004). Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318.

MacKay, D. J. C. and Peto, L. (1995). A hierarchical Dirichlet language model. *Natural Language Engineering*, 1:1–19.

Mangu, L. and Brill, E. (1997). Automatic rule acquisition for spelling correction. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 187–194, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Martin, A. and Pryzbocki, M. (2006). Nist 2003 language recognition evaluation.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Inf. Process. Manage.*, 27(5):517–522.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Mohri, M. (2002). Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers. In *International Journal of Foundations of Computer Science 13*, pages 129–143.

Mohri, M. (2005). Statistical natural language processing. In Lothaire, M., editor, *Applied Combinatorics on Words*. Cambridge University Press.

Mohri, M. (2009). Weighted automata algorithms. In Droste, M., Kuich, W., and Vogler, H., editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 213–254. Springer.

Mohri, M., Pereira, F., and Riley, M. (1996). Weighted automata in text and speech processing. In *Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended Finite state models of language*, pages 46–50, Budapest, Hungary. John Wiley and Sons.

Murata, M. and Nagao, M. (1993). Determination of referential property and number of nouns in japanese sentences for machine translation into english. In *In Proceedings of the 5th TMI*, pages 218–225.

Nádas, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32:859–861.

Ney, H., Essen, U., and Kneser, R. (1994). On structuring probabilistic dependences in stochastic language modelling. *Computer, Speech and Language*, 8:1–38.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

Pereira, F. C. N. and Riley, M. D. (1997). Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press.

Porter, M. F. (1980). An Algorithm for Suffix Stripping. *Program*, 14(3):130–137.

Powers, D. M. W. (1997). Learning and application of differential grammars. In *In Proc. Meeting of the ACL Special Interest Group in Natural Language Learning*, pages 88–96.

Rozovskaya, A. and Roth, D. (2010). Annotating ESL errors: Challenges and rewards. In *Proceedings of the NAACL Workshop on Innovative Use of NLP for Building Educational Applications*.

Rozovskaya, A. and Roth, D. (2011). Algorithm selection and model adaptation for ESL correction tasks. In *Proc. of the Annual Meeting of the Association of Computational Linguistics (ACL)*, Portland, Oregon. Association for Computational Linguistics.

Shannon, C. (1948). A mathematical theory of communications. *Bell Systems Technical Journal*, 27(4):623–656.

Shichun, G. and Huizhong, Y. (2003). Chinese Learner English Corpus. Shanghai Foreign Language Education Press.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *In Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

Sun, X., Gao, J., Micol, D., and Quirk, C. (2010). Learning phrase-based spelling error models from clickthrough data. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 266–274, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tetreault, J. R. and Chodorow, M. (2008). The ups and downs of preposition error detection in ESL writing. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, COLING '08, pages 865–872, Morristown, NJ, USA. Association for Computational Linguistics.

Tetreault, J. R., Filatova, E., and Chodorow, M. (2010). Rethinking grammatical error annotation and evaluation with the amazon mechanical turk. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications*, IUNLPBEA '10, pages 45–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Toutanova, K. and Moore, R. C. (2002). Pronunciation modeling for improved spelling correction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 144–151, Stroudsburg, PA, USA. Association for Computational Linguistics.

Whitelaw, C., Hutchinson, B., Chung, G. Y., and Ellis, G. (2009). Using the web for language independent spellchecking and autocorrection. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 890–899, Stroudsburg, PA, USA. Association for Computational Linguistics.

Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1094.

Wu, C.-F. J. (1983). On the convergence properties of the EM algorithm. *Ann. Statist.*, 11(1):95–103.

Yarowsky, D. (1994). Decision lists for lexical ambiguity resolution: application to accent restoration in spanish and french. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, ACL '94, pages 88–95, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, Y. and Vogel, S. (2010). Significance tests of automatic machine translation evaluation metrics. *Machine Translation*, 24:51–65.