An Unsupervised Approach to Detecting and Correcting Errors in Text

by

Md. Aminul Islam

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Ph.D. degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

uOttawa

Abstract

In practice, most approaches for text error detection and correction are based on a conventional domain-dependent background dictionary that represents a fixed and static collection of correct words of a given language and, as a result, satisfactory correction can only be achieved if the dictionary covers most tokens of the underlying correct text. Again, most approaches for text correction are for only one or at best a very few types of errors.

The purpose of this thesis is to propose an unsupervised approach to detecting and correcting text errors, that can compete with supervised approaches and answer the following questions:

- Can an unsupervised approach efficiently detect and correct a text containing multiple errors of both syntactic and semantic nature?
- What is the magnitude of error coverage, in terms of the number of errors that can be corrected?

We conclude that (1) it is possible that an unsupervised approach can efficiently detect and correct a text containing multiple errors of both syntactic and semantic nature. Error types include: real-word spelling errors, typographical errors, lexical choice errors, unwanted words, missing words, prepositional errors, article errors, punctuation errors, and many of the grammatical errors (e.g., errors in agreement and verb formation). (2) The magnitude of error coverage, in terms of the number of errors that can be corrected, is almost double of the number of correct words of the text. Although this is not the upper limit, this is what is practically feasible.

We use engineering approaches to answer the first question and theoretical approaches to answer and support the second question. We show that finding inherent properties of a correct text using a corpus in the form of an n-gram data set is more appropriate

and practical than using other approaches to detecting and correcting errors. Instead of using rule-based approaches and dictionaries, we argue that a corpus can effectively be used to infer the properties of these types of errors, and to detect and correct these errors. We test the robustness of the proposed approach separately for some individual error types, and then for all types of errors. The approach is language-independent, it can be applied to other languages, as long as n-grams are available.

The results of this thesis thus suggest that unsupervised approaches, which are often dismissed in favor of supervised ones in the context of many Natural Language Processing (NLP) related tasks, may present an interesting array of NLP-related problem solving strengths.

Dedication

- Rubia Islam

  A natural philosopher, an epitome of patience.

- Md. Rafiqul Islam

  A natural scientist, born with unnatural human values.

- Shanti Pada Biswas

  I did not find any researcher better than him. He was a school teacher in a very remote area in Bangladesh.

- S.M. Khairul Alam

  He is still in my heart.

- Bushra Afrin

  She did a lot of sacrifice. A perfect mother.

Acknowledgements

It has been my very great privilege to know and work with my advisor Dr. Diana Inkpen in various capacities over the past five years. I would like to thank Dr. Inkpen for her great guidance and timely support. I am greatly thankful that she would take me as a student, teach me the research methodology, guide me in choosing interesting and influential research topics, and encourage me. I feel very lucky to have had Dr. Inkpen as an advisor, and look forward to having more opportunities to learn from and work with her in the future.

I would like to thank Dr. Vlado Keselj, Dr. Jean-Pierre Corriveau, Dr. Iluju Kiringa, and Dr. Nathalie Japkowicz for being on my thesis committee and for all their incisive, insightful and immensely helpful comments.

I am grateful to Emi Izumi and Kiyotaka Uchimoto from National Institute of Information and Communications Technology (NICT) for providing the NICT JLE corpus. Many thanks to Alistair Kennedy and Colette Joubarne for their help in evaluating our text correction approach. I would like to express my gratitude to all the researchers who have worked in Natural Language Processing at the School of Information Technology and Engineering. Support, by funding my Ph.D., comes from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

I got some motivating teachers throughout my study life. I am grateful to Anwar Hossain, my early life teacher who was more than a friend, Dr. Mozammel Huq Azad Khan, a motivating teacher, Dr. Rafiqul Islam, an industrious teacher, Golam Rasul Chaman, who first introduced me the world of Natural Language Processing. He is a perfect example of how to teach in class.

I am thankful to my friend Helal Uddin.

I am greatly indebted to my family members: my parents Rubia Islam and Nurul Islam, my sisters Shahinur Islam, Sharmin Akhter, and Shamanta Islam, my brother-in-

laws Shamsul Alam, Amanur Rahman, and G.M. Minar, my nephews and nieces Samia Sanjabin, Shaker Islam, Mehrin Fatema, Arafa, Safa, and Shajnin Fatema for their love and support.

Last but not the least, I am greatly indebted to my wife Bushra Afrin and my daughter Yushra Amin for their unconditional love and for believing in me throughout. I could not have done this without you.

Contents

List of Tables

List of Figures

List of Symbols

Chapter 3

Chapter 4

T	An input text having p $(2 \leq p \leq 9)$ words	67
$\boxed{w_i}$	A set of m prepositions, $\{s_1, s_2, \cdots, s_j, \cdots, s_m\}$, for gap position i	67

Chapter 5

v_1	Normalized Longest Common Subsequence ($NLCS$)	81
v_2	Normalized Maximal Consecutive LCS starting at the first character ($NMCLCS_1$)	81
v_3	$NMCLCS$ starting at any character n ($NMCLCS_n$)	82
v_4	$NMCLCS$ ending at the last character ($NMCLCS_z$)	82
α_1	Weight factor for v_1	82
α_2	Weight factor for v_2	82
α_3	Weight factor for v_3	82
α_4	Weight factor for v_4	82
$S_1(s_1, s_2)$	String similarity between string s_1 and s_2	82
n	Number of replacements of a word w_i which are $w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}$	84
f_{ij}	The frequency of a n-gram where any candidate word w_{ij} is a member of the n-gram	84
$F(w_{ij})$	Normalized frequency value of a candidate word w_{ij}	84
T	An input text having m words, $w_1, w_2, \ldots, w_i, \ldots, w_m$	85
R_1	Set of n replacements of w_i which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$	86
F_1	Set of n frequencies $\{f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in}\}$	86
β	Weight factor for string similarity function	86
$\bar{j}$	Index of the word having the maximum weight value	91

Chapter 6

$S_2(T_1, T_2)$	Common-word similarity value between text T_1 and T_2	103
m	Number of tokens in input text T or T_1	103
n	Number of tokens in text T_2	103
δ	Number of tokens in text T_1 that exactly matches with text T_2	103
$S_3(T_1, T_2)$	Common-word order similarity value between text T_1 and T_2	105
$S_4(T_1, T_2)$	Non-common word similarity value between text T_1 and T_2	107
w_{ij}	jth token of the candidate text T_i	108
m_i	Number of tokens a candidate text T_i has	108
F_i	Set of frequencies of all the 5-grams of a candidate text T_i	109
f_{ij}	The frequency of the jth 5-gram of a candidate text T_i	109
$S_5(T_i)$	Normalized frequency value of a candidate text T_i	109
$\tilde{n}$	Number of candidate texts for input text T	108
N	A set of $\tilde{n}$ numbers	108
$\bar{n}$	Number of candidate 5-grams generated from each 5-gram rule that is used for further processing	110
I_j	jth word inserted	111
R_i	The token in between i-1 and i+1 is replaced	111
n_{tp}	Number of steps needed for an input text T	113
$S(T_i)$	Correctness value of a candidate text T_i	121
β_1	Weight factor for $S_2(T_i, T)$	121
β_2	Weight factor for $S_3(T_i, T)$	121
β_3	Weight factor for $S_4(T_i, T)$	121
β_4	Weight factor for $S_5(T_i)$	121

Glossary

accuracy Accuracy is the proportion of correct results (both true positives and true negatives).

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{false positives} + \text{false negatives} + \text{true negatives}}$$

correction precision (P) The fraction of amended errors which are correct is known as correction precision.

$$P = \frac{\text{number of errors correctly amended}}{\text{total number of errors amended}} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

correction recall (R) The fraction of errors correctly amended is known as correction recall.

$$R = \frac{\text{number of errors correctly amended}}{\text{total number of errors}} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

detection precision (P) The fraction of detected errors by the system which are cor-

rect is known as detection precision[1].

$$P = \frac{\text{number of detected errors which are correct}}{\text{total number of errors detected by the system}} = \frac{\text{true positives}}{\text{true positives + false positives}}$$

detection recall (R) The fraction of errors correctly detected is known as detection recall.

$$R = \frac{\text{number of detected errors which are correct}}{\text{total number of errors}} = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

F-measure The harmonic mean of *precision* and *recall* is known as F-measure (or F_1).

$$F_1 = \frac{2PR}{P + R}$$

false negative (FN) If a system says that an input text has no error (i.e., negative from the detection point of view) and if this is false, this suggestion is known as *false negative*.

false negative decreasing function The function that helps to decrease the number of *false negatives* is called *false negative decreasing* function.

false positive (FP) If a system says that a returned suggestion is correct (i.e., positive) and if this is not true (i.e., false), this suggestion is known as *false positive*. In short, a returned suggestion which is wrong is known as *false positive*.

false positive decreasing function The function that helps to decrease the number of *false positives* is called *false positive decreasing* function.

[1]The definitions of correction precision and detection precision seem to be the same, based on true positives and false positives; but the values of the true positives and the false positives for correction precision and detection precision might be different for the same input. For example, if a method can correctly detect x errors and correct y errors, and if $x > y$, for the same input then the true positives for detection and correction would be x and y, respectively.

Google web 1T n-gram The Google Web 1T n-gram data set, contributed by Google Inc., contains English word n-grams (from unigrams to 5-grams) and their observed frequency counts calculated over 1 trillion words from web page text collected by Google in January 2006.

lexical choice See **near-synonyms**.

near-synonyms Words that have the same meaning, but differ in lexical nuances. A sentence may be perfectly grammatical and yet violates the habitual usage of words which can result in awkward collocations. For example, in the sentence, "*She had a cup of powerful tea.*", the word '*strong*' is more habitual than the word '*powerful*'.

no suggestion It means that the method does not find any better suggestion than the observed word.

real-word Words which are in dictionary or thesaurus.

real-word spelling errors Real-word spelling errors are words in a text that occur when a user mistakenly types a correctly spelled word when another was intended.

true negative (TN) TN = total number of words in the correct text − (true positives + false positives + false negatives)

true positive (TP) If a system says that a returned suggestion is correct (i.e., positive) and if this is true, the suggestion is known as *true positive*. In short, a returned suggestion which is correct is known as *true positive*.

Chapter 1

Introduction

1.1 Motivation

To the best of our knowledge, there is no fully-unsupervised approach that corrects a text containing multiple errors of both syntactic and semantic nature. Syntactic errors refer to all kinds of grammatical errors. For example, in the sentence, "*Our method correct real-word spelling errors.*", there is an error of syntactic nature in subject-verb agreement, whereas, in the sentence, "*Our method corrects real-world spelling errors.*", '*real-world*' should be replaced by '*real-word*' in order to convey the proper intended meaning of the sentence, based on the context. The latter is an example of a semantic error. Again, most approaches to text correction are for only one or at best for a few types of errors, which motivates us to propose an approach that can correct a text containing multiple types of errors. Also, our approach is able to correct multiple errors in the same sentence (almost double number of errors as the number of correct word), while other approaches work only for a small number of errors in the same sentence.

Machine learning and data mining has resulted in probably thousands of supervised algorithms and models with an extreme imbalance between the number of published

algorithms versus those really repeatable and executable in the real world (Cao, 2010). For example, surveys of supervised data mining algorithms for business applications in various domains have shown that findings cannot make themselves executable in the real world (Cao, 2008). This also motivates us toward fully-unsupervised approaches that can easily work in the real world.

This thesis seeks to advance the state-of-the-art in text correction including both syntactic, as well as semantic errors, using an unsupervised statistical method.

1.2 Scope of Research Topic

We will start the scope of this thesis with a very simple example of input text, "*he wss very good tea teacher*" which has six tokens[1]. By preserving the intended or semantic meaning of the input text as much as possible, it is obvious that one of the best candidates for the corrected version of the input text would be "*he was a very good teacher*". To have this corrected version of the input text, we need to **replace** '*wss*' by '*was*', to **insert** '*a*' in between '*wss*' and '*very*', and to **delete** the word '*tea*'. For any text correction, these are the three operations: **replace**, **insert**, and **delete** that need to be performed.

We need to define the notions of *sentence*, *clause*, *phrase*, and *text* to have the clear purview of a *text*. From these definitions, we try to find relations among these terms. These definitions will answer the question of why we focus only on *text* as a unit not on a *sentence* or on other levels.

According to Merriam-Webster Online Dictionary[2], a *phrase* is:

> "a word or group of words forming a syntactic constituent with a single grammatical function. For example, "*the house at the end of the street*"

[1] We use the term '*token*' to denote alphanumeric character(s) that constitute either a correct or an incorrect word.

[2] `http://www.merriam-webster.com/dictionary/`

(example 1) is a phrase. It acts like a noun. It contains the phrase "*at the end of the street*" (example 2), a prepositional phrase which acts like an adjective. Example 1 could be replaced by "*white*", to make the phrase "*the white house*" . Examples 1 and 2 contain the phrase "*the end of the street*" (example 3) which acts like a noun."

According to Merriam-Webster Online Dictionary, a *clause* is:

> "a group of words containing a subject and predicate and functioning as a member of a complex or compound sentence." For example, "*Because I had to move*" is a dependent clause, whereas, "*I went to the store with my friends*" is an independent clause."

According to Merriam-Webster Online Dictionary, a *sentence* is:

> "a word, clause, or phrase or a group of clauses or phrases forming a syntactic unit which expresses an assertion, a question, a command, a wish, an exclamation, or the performance of an action, that in writing usually begins with a capital letter and concludes with appropriate end punctuation, and that in speaking is distinguished by characteristic patterns of stress, pitch, and pauses."

Some of the many definitions of *text* from Merriam-Webster Online Dictionary are:

- matter chiefly in the form of words or symbols that is treated as data for processing by computerized equipment.
- the original words and form of a written or printed work.
- the main body of printed or written matter on a page.

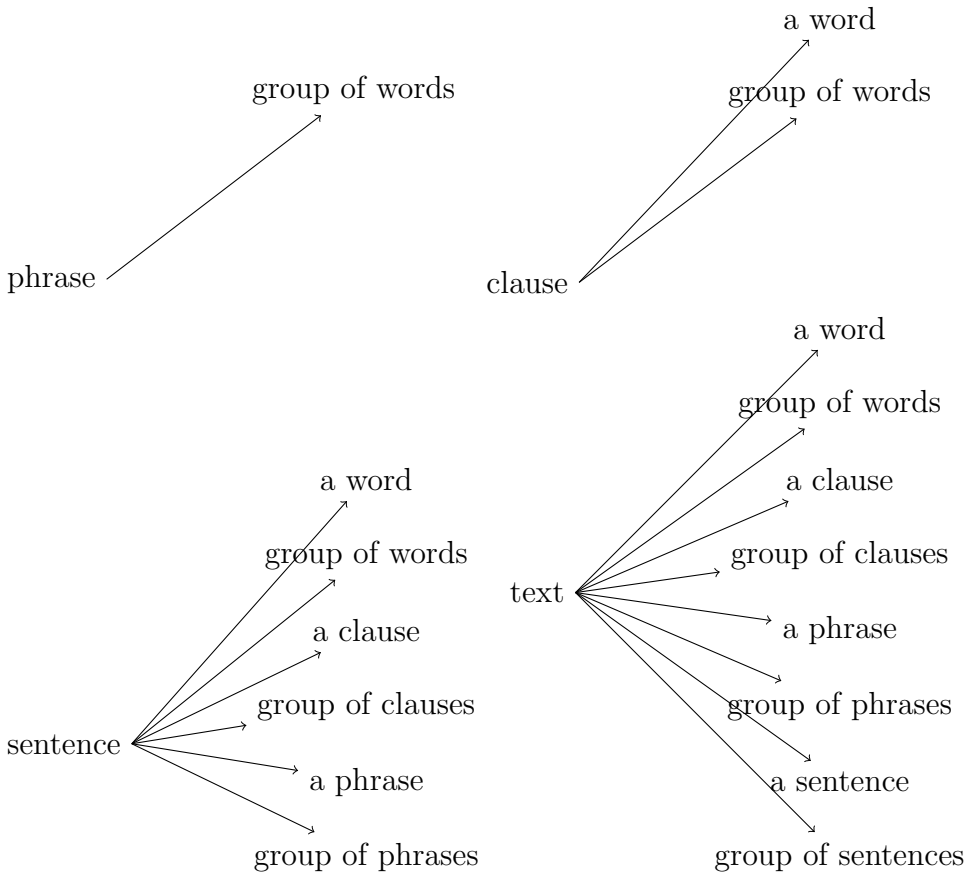

Figure 1.1: Constitutive unit(s) of phrase, clause, sentence, and text.

Figure 1.1 demonstrates at a glance the constitutive unit(s) of phrase, clause, sentence, and text. Thus, in short, we can define a *text* as a superset of a sentence or a group of sentences, a *sentence* as a superset of a clause or a group of clauses, a *clause* as a superset of a word or a group of words, a *phrase* as a superset of group of words. Figure 1.2 demonstrates this definition graphically in Euler diagram. Table 1.1 shows some examples of texts.

Although the focus of this thesis is on English, our methods do not draw on any specificity of the language. Our methods could be used with a minute change or almost changing nothing to many other languages having enough available n-grams. In October, 2009, Google released Web 1T 5-gram, 10 European Languages Version 1 (Brants

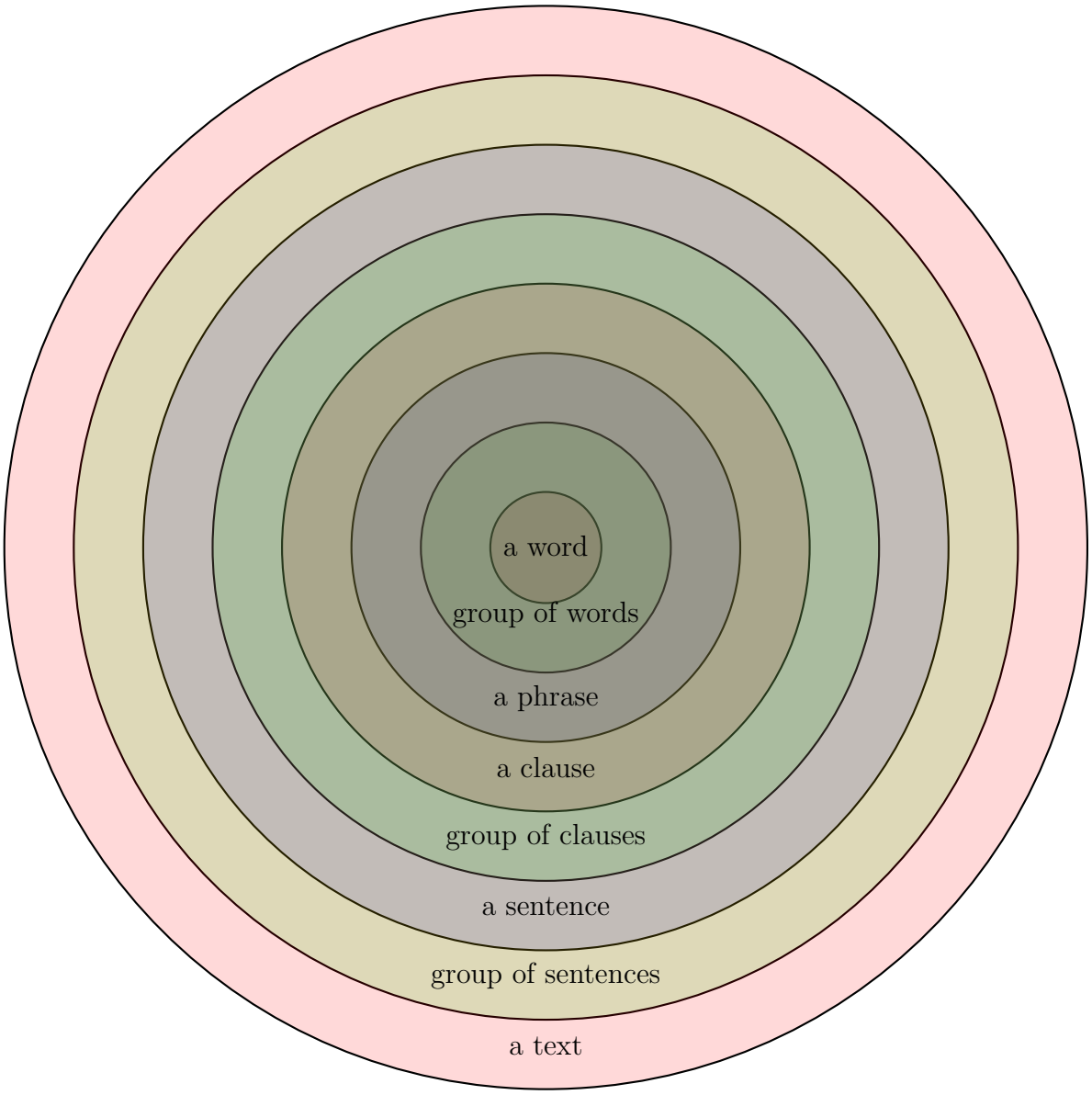

Figure 1.2: Euler diagram showing a *text* is a superset of a sentence or a group of sentences and so on.

and Franz, 2009) consisting of word n-grams and their observed frequency counts for ten European languages: Czech, Dutch, French, German, Italian, Polish, Portuguese, Romanian, Spanish and Swedish. Thus, it is possible to use the proposed methods to these languages.

We use the following three strict assumptions and one weak assumption for the input

Table 1.1: Examples of Texts

Examples of Text	Reason
stop	a word (command)
the white house	a phrase (noun)
the end of the street	a phrase (noun)
at the end of the street	a phrase (prepositional)
the white house at the end of the street	group of phrases
because I had no headlight	a clause (dependent)
I was asked to stop	a clause (independent)
I was asked to stop near the white house at the end of the street because I had no headlight.	a sentence

text that needs to be corrected:

Strict Assumptions

1. The first token is a word[3].

2. There should be at least three words in an input text.

3. There might be at most two consecutive errors in between two words. We also assume that there might be at most two consecutive errors after the last word. The only exception of this assumption is that among the first three words there might be at most two errors in total, which in turn might be consecutive; but after that, in between two words, there might be at most two consecutive errors.

Weak Assumption

- We try to preserve the intended semantic meaning of the input text as much as possible.

The error types that are targeted in this thesis are: real-word spelling errors, typographical errors, lexical choice errors, unwanted words, missing words, prepositional

[3]Whenever we use only the term '*word*' without an adjective (e.g., correct or incorrect) we imply a correct word.

errors, article errors, punctuation errors, and many of the grammatical errors (e.g., errors in agreement and verb formation).

1.3 Resource Used

The Google Web 1T n-gram data set is the only resource that is used for all the approaches discussed in this thesis. The Google Web 1T n-gram data set (Brants and Franz, 2006) contains English word n-grams (from unigrams to 5-grams) and their observed frequency counts calculated over 1 trillion words. The text was tokenized following the Penn Treebank tokenisation, except that hyphenated words, dates, email addresses and URLs are kept as single tokens. The sentence boundaries are marked with two special tokens <S> and </S>. Words that occurred fewer than 200 times were replaced with the special token <UNK>. Table 1.2 shows the data sizes of the Web 1T corpus. The n-grams themselves must appear at least 40 times to be included in the Web 1T

Table 1.2: Google Web 1T Data Sizes

Number of	Number	Size on disk (in KB)
Tokens	1,024,908,267,229	N/A
Sentences	95,119,665,584	N/A
Unigrams	13,588,391	185,569
Bigrams	314,843,401	5,213,440
Trigrams	977,069,902	19,978,540
4-grams	1,313,818,354	32,040,884
5-grams	1,176,470,663	33,678,504

corpus[4]. It is expected that this data will be useful for statistical language modeling, e.g., for machine translation or speech recognition, as well as for other uses.

Table 1.3 shows some examples of the n-gram data contained in the Web 1T corpus.

[4]Details of the Web 1T are in www.ldc.upenn.edu/Catalog/docs/LDC2006T13/readme.txt

Table 1.3: Examples of Google n-gram Data

n=	n-grams	Frequencies
3	he was a	3,683,417
	hehe was a	52
	he was an	563,471
	he was am	121
	he was awesome	7,520
	he was awsome	548
4	he was a vegetarian	1,357
	he was a vendor	74
	he was a versatile	251
	he was a veritable	454
	he was a very	65,325
	he was a veteran	2,979
5	he was a very generous	276
	he was a very gentle	200
	he was a very genuine	58
	he was a very gifted	177
	he was a very good	7,447
	he was a very great	1,122

1.4 Contributions

In this thesis, we make the following contributions.

- First, we present a generalized unsupervised approach to tackle *replace*, the most crucial text correction operation among the three. The proposed approach, without incorporating string similarity, uses the Google Web 1T n-gram data set in 'a back off fashion', which means that if our proposed method fails to find a candidate solution using 5-grams then we move from 5-grams to 4 grams, and so on. The approach uses an extra phase, if required, to increase the performance. We test the approach to two different *replace*-related problems: near-synonym choice, and preposition error correction, that do not require string similarity. Our evaluation experiments showed that the proposed two-phase approach outperforms the state-of-the-art approaches for those two tasks.

- Second, we present another unsupervised approach that exploits the *replace* operation, incorporating string similarity. The proposed approach also uses an extra phase, if required, to increase the performance. The approach has been tested on a *replace*-related problem: real-word spelling errors that require string similarity.

- Third, we present four generalized similarity functions: a modified *string similarity function* that determines the similarity between two strings; *common-word similarity function* that determines the similarity between two texts based on the common words that the two texts share; *common word-order similarity function* that determines the similarity between two texts based on the order of the common words that the two texts share; *non-common word similarity function* that determines the similarity between two texts based on the non-common words in two texts. We found that all the four similarity functions can successfully be applied for semantic text similarity (Islam and Inkpen, 2008) and text correction. These functions can also be useful in solving some other Natural Language Processing (NLP) related problems. Some examples are: normalizing short messages (e.g., tweets and SMS messages), which have limited length and may contain misspellings, abbreviations, or slang, as well as detecting and correcting speech recognition errors.

- Finally, we present a fully-unsupervised approach that corrects a text containing multiple errors of both syntactic and semantic nature. This is the approach that uses the *insert* and *delete* operations along with the *replace* operation. An input text having $3m-2$ tokens can have at most $2m-2$ errors to be handled while having at least m correct words, assuming $m \geq 3$ (second assumption on page 6 of Chapter 1). For example, an input text having seven tokens can have at most four erroneous tokens. Detailed theoretical analysis will be presented for several parameters. For example, the limit of the number of candidate texts for an input text, the number of steps needed, etc. We generate a new evaluation data set containing multiple errors

in short texts because we found a lack of publicly available data sets to evaluate any generic text correction approach. We also compared our unsupervised approach with a supervised approach and got better results.

1.5 Outline

The different issues that we consider throughout the thesis are part of the same family. For example, this is very clear in going from Chapters 3 to 4: prepositions could be thought of as near-synonyms and therefore replacing a preposition is almost like changing any other type of word. The transition is not as smooth for Chapter 5, because a wrongly spelled word or a real-word spelling error can not be construed as one that needs a near-synonym to replace it.

In Chapters 3, 4, and 5, only the *replace* operation is used in three different applications, contrary to Chapter 6, where all the three operations (*replace*, *insert*, and *delete*) are used to detect and correct errors in a text. Figure 1.3 shows a generic flow diagram

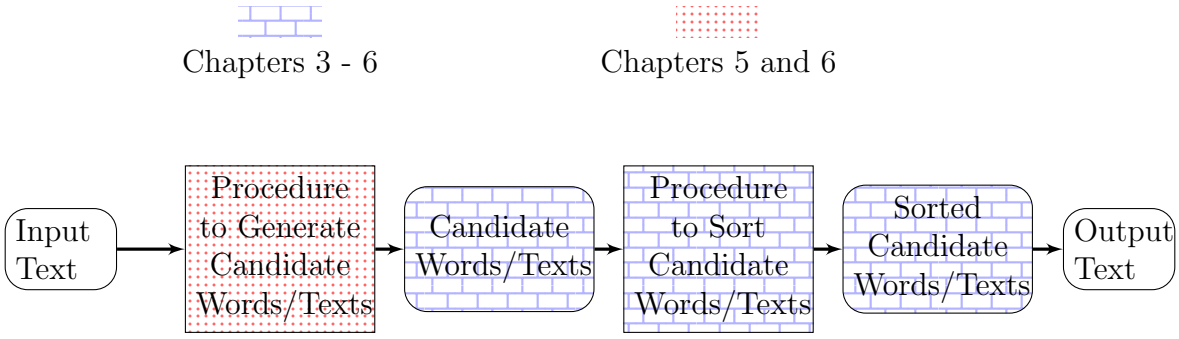

Figure 1.3: A generic flow diagram of Chapters 3, 4, 5, and 6.

of Chapters 3, 4, 5, and 6.

In Chapters 3 and 4, where we use only the *replace* operation, our task is to find the best choice (or word) of a near-synonym or a preposition, from a given list of candidate choices or words. In Chapter 5, we use the *replace* operation to find the best suggestion

(or word) of a real-word spelling correction, where we are to generate a list of candidate suggestions. In Chapters 3, 4, and 5, we use only the normalized frequency value function in the process of choosing the best candidate from a list of candidate words. Thus, the basic difference between Chapters 3, 4 and Chapter 5 is the use of the modified string similarity functions in the process of generating a list of candidate words for Chapter 5. In Chapter 6, we also use the modified string similarity functions. When we use *replace* operation in Chapter 6, our task is three-fold. First, to determine a list of replaceable candidates (if we are in need to replace a token for correction purpose) which is not given in this case and, second, to generate a list of candidate texts by applying all/any of the remaining two text correction operations (if we are in need to insert words or delete tokens for correction purpose) and finally, to find the best text from the list of candidate texts.

That is, in Chapters 3, 4, and 5, the best matching text is generated by applying only the *replace* operation, whereas, in Chapter 6, all the candidate texts are generated by applying any of the three text correction operations. In Chapter 6, we use the normalized frequency value of each candidate text, whereas in Chapters 3, 4, and 5, we use the normalized frequency value of each candidate word (i.e., text level versus word level). This is why the normalized frequency value that is discussed in Chapter 6 differs from those used in Chapters 3, 4, and 5. In Chapter 6, we also use some other similarity functions, to sort the candidate texts. The rest of this thesis is organized as follows[5]:

Chapter 2 provides a brief review of previous research related to the correction of grammatical errors, as well as research related to the correction of text, with more related work being described in the chapters for specific tasks. This chapter also presents a brief overview of the related work that uses the Google Web 1T data set.

Chapter 3 proposes an unsupervised statistical method for automatic choice of near-

[5]For the readers who have sufficient background, the thesis is organized in such a way that each chapter can be read in isolation.

synonyms using the Google Web 1T n-gram data set and compares with the state-of-the-art. As mentioned before, here we do not use any procedure to generate candidate words, rather we use normalized frequency value function to sort the provided candidate words. The method has two phases and it uses the Google Web 1T data set in a back off fashion that increases the performance.

Chapter 4 uses the same unsupervised statistical method discussed in Chapter 3 for automatic correction of preposition errors using the Google Web 1T n-gram data set and compares to the state-of-the-art.

Chapter 5 presents an unsupervised method for correcting real-word spelling errors using the Google Web 1T n-gram data set. We use a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm to generate candidate words, and a normalized frequency value function to sort candidate words. Our method is focused mainly on how to improve the correction recall (the fraction of errors corrected) while keeping the correction precision (the fraction of suggestions that are correct) as high as possible.

Chapter 6 proposes a more general unsupervised statistical method for automatic text correction using the Google Web 1T n-gram data set. The proposed text correction approach first generates a set of probable candidates and then sorts those candidates based on some functions that help to decrease the rates of *false positives* and *false negative*. Here, we use normalized frequency value function, along with some other functions unlike in Chapters 3, 4, and 5, where we use only normalized frequency value function to sort candidate texts. The question of how we sort the candidates or how we set the balance between the original input text and each candidate text — one of the key issues of our proposed approach — is also in focus in this chapter. We prepare a data set to compare our approach to two human judges.

Chapter 7 makes concluding remarks and sketches directions for future work.

Appendix A uses the same unsupervised statistical method for automatic text correction described in Chapter 6 with a different set of assumptions, which are less strict than the assumptions used in Chapter 6. Here, we solve a simple case of text correction, with a smaller number of errors that can be corrected.

Appendix B describes how the Google Web 1T 5-gram data set can efficiently be used and managed as a context window. We present an efficient way of accessing all the 5-grams containing the context words for a specific word of interest from the stored files. We measure the maximum access and processing efficiency achievable for any word of interest. We also compare results (access time and memory requirements) on the task of accessing all the 5-grams containing the context words for a list of words, on both the processed and the original organization of the data set. To efficiently store the Google data set, it is preprocessed by removing some special characters, numbers, etc., to reduce the storage size. This preprocessed data set is not used in the text correction approach as the preprocessing removes some important features that need to be kept. In spite of this, the efficient way of storing and using mechanism of the Google data set has been added in appendix because it can be useful for some other NLP-related problems. It can be used for text correction if we reindex the n-grams by skipping the preprocessing steps.

1.6 Related Publications

Parts of the thesis work have been published in some journals and conference proceedings. Below is an incomplete list:
Related publications of Chapter 3 include:

- Islam, A. and Inkpen, D. (2010a). Near-synonym choice using a 5-gram language model. *Research in Computing Science*, 46:41–52. Special issue on Natural Language Processing and its Applications.

- Islam, A. and Inkpen, D. (2010b). Unsupervised near-synonym choice using the Google Web 1T n-gram data set. *IEEE Trans. Knowl. Data Eng.* Submitted (Manuscript ID: TKDE-2010-04-0241).

Related publications of Chapter 4 include:

- Islam, A. and Inkpen, D. (2010c). An unsupervised approach to preposition error correction. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'10)*, Beijing, China.

Related publications of Chapter 5 include:

- Islam, A. and Inkpen, D. (2009b). Real-word spelling correction using Google Web 1T n-gram data set. In Cheung, D. W.-L., Song, I.-Y., Chu, W. W., Hu, X., and Lin, J. J., editors, *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009*, pages 1689–1692, Hong Kong. ACM.

- Islam, A. and Inkpen, D. (2009a). Real-word spelling correction using Google Web 1T 3-grams. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009*, pages 1241–1249, Singapore. Association for Computational Linguistics.

- Islam, A. and Inkpen, D. (2009d). Real-word spelling correction using Google Web 1T n-gram with backoff. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'09)*, Dalian, China.

Related publications of Chapter 6 include:

- Islam, A. and Inkpen, D. (2008). Semantic text similarity using corpus-based word similarity and string similarity. *ACM Trans. Knowl. Discov. Data*, 2(2):1–25.

- Islam, A., Inkpen, D., and Kiringa, I. (2008). Applications of corpus-based semantic similarity and word segmentation to database schema matching. *The VLDB Journal*, 17(5):1293–1320.

- Islam, A. and Inkpen, D. (2009c). Semantic similarity of short texts. In Nicolov, N., Angelova, G., and Mitkov, R., editors, *Recent Advances in Natural Language Processing V*, Current Issues in Linguistic Theory 309, pages 227–236. John Benjamins Publishing Company. Selected papers from Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP) 2007.

- Inkpen, D. and Islam, A. (2010). Unsupervised Approaches to Text Correction using Google n-grams for English and Romanian. In Tufiş, D. and Forăscu, C., editors, *Multilinguality and Interoperability in Language Processing with Emphasis on Romanian*, pages 270–285, Bucharest, Romania. Romanian Academy Publishing House.

Related publications of Appendix A include:

- Islam, A. and Inkpen, D. (2011a). Correcting different types of errors in texts. In *24th Canadian Conference on Artificial Intelligence, Canadian AI 2011*, pages 192–203, LNAI 6657, St. John's, Canada.

Related publications of Appendix B include:

- Islam, A. and Inkpen, D. (2009d). Managing the Google Web 1T 5-gram data set. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'09)*, Dalian, China.

- Islam, A. and Inkpen, D. (2008a). Efficient storing and accessing of Google's Web 1T 5-gram data set. In *Proceedings of the International Conference on Electronics, Computer and Communication (ICECC 2008)*, Rajshahi, Bangladesh.

- Islam, A. and Inkpen, D. (2009e). Using and managing the Google Web 1T 5-gram data set as a context window. *Information Processing & Management*. submitted (Submission number: IPM-S-09-00377).

Chapter 2

Related Work

In this chapter, we review previous research related to the correction of grammatical errors as well as research related to the correction of text. The work that is closely related to ours is that of Lee (2009)'s supervised method, where a machine learning framework using Support Vector Machine (SVM) distinguishes *non-native* ('ill-formed') English sentences from *native* ('well-formed') ones. Lee (2009) describes a generation-based approach on a restricted domain to grammar correction on a sentence-level, where a word lattice of candidate corrections is generated from an ill-formed input, and a traditional n-gram language model is used to produce a small set of N-best candidates, which are then reranked by parsing, using a stochastic context-free grammar. In addition, to improve precision, he utilizes the Google Web 1T 5-gram corpus as filters, allowing a correction only when its n-gram count in the corpus is greater than that of the original. This filtering step reduced false positives from 46.4% to less than 1%. The error types that are targeted in his work are: articles, prepositions, noun number (i.e., singular or plural), auxiliary verbs and verb forms. Our unsupervised approach, which uses only the Google Web 1T data set as resource, covers all of the above errors in addition to real-word spelling errors, typographical errors, unwanted words, missing words, lexical

choice errors, and so on.

We summarize previous research in three main paradigms: scoring errors, using parsers, and machine translation. Our approach falls into the first category. We also review the work based on the Google Web 1T data set.

2.1 Scoring Errors

In some approaches, instead of actually correcting the error, which is the focus of this thesis, the focus is on simply detecting sentences that contain errors, or computing a score that reflects the quality of the text.

2.1.1 Text Error Detectors

There is a debate on the issue of whether automatic error detection is at all feasible. Granger et al. (2007) present an in-depth discussion on the topic. Most of the early work centered on detecting *real-word* spelling errors, spelling errors or grammatical errors resulting in other legitimate words of the language. Atwell (1987) described an automatic word-tagging system that can be modified to detect grammatical errors, essentially by flagging unlikely word-class transitions in the input text on the basis of infrequent POS tag sequences.

This task requires a binary decision: whether a text or sentence is good enough or not; was originally designed for machine translation (MT). Chodorow and Leacock (2000) present an unsupervised method for detecting grammatical errors (e.g., *a desks*) by inferring negative evidence from edited textual corpora. In order to detect this type of problem, the method uses a 30-million word general corpus of English from the San Jose Mercury News and, for each target word, a set of 10,000 example sentences from North American newspaper text. The system was developed and tested using essay-

length responses to prompts on the Test of English as a Foreign Language (TOEFL). Wang and Seneff (2004) describe a translation framework aimed at achieving high-quality speech translation within restricted conversational domains where the parse failure would strongly suggest that the translation is flawed. Seneff (1992) parses the low-quality output providing a seamless interface between syntactic and semantic analysis, and also producing a highly constraining probabilistic language model to improve recognition performance while using a semantic parser to ultimately detect such low-quality output. Tomokiyo and Jones (2001) show that naive Bayes classification can be used to identify non-native utterances of English by characterizing part-of-speech sequences that play a role in detecting non-native speech. Their method relies on text, not on acoustic features, and can be used when the acoustic source is not available. Eeg-olofsson and Knutsson (2003) discuss the construction and implementation of a set of matching rules for a grammar checking system, written with the purpose of detecting second language learners writing errors related to the use of prepositions.

Gamon et al. (2005) investigate the possibility of evaluating machine translation (MT) quality and fluency at the sentence level in the absence of reference translations. They measure the correlation between automatically-generated scores and human judgments and show that they can substantially improve on the correlation between language model perplexity scores and human judgment by combining these perplexity scores with class probabilities from a machine-learned classifier. Corston-Oliver et al. (2001) present a machine learning approach to evaluating the well-formedness of the output of a machine translation system, using classifiers that learn to distinguish human reference translations from machine translations. Sjöbergh (2005) describes a method for grammar checking by comparing the output of a chunker on new texts to the output on known correct text and rare chunk sequences that occur in the new texts are reported as suspected errors. The method requires unannotated text for training. Though the method is evaluated on

Swedish texts from a few different genres, Sjöbergh (2005) claims that the method can be used without modifications on any language, as long as a chunker is available for that language.

Lee (2009) uses a machine learning framework using SVM-Light, an implementation of Support Vector Machine (SVM), for the classification task to identify non-native sentences where he uses five features extracted from each sentence: the first two are real numbers; the rest are indicator functions of the presence of lexical and/or syntactic properties.

2.1.2 Text Error Correction

Descriptions of some currently available resources specifically for the purpose of correcting written texts can be found in (Thomas, 2004).

An initial approach to automatic acquisition for context-based spelling correction was a statistical language-modeling approach using word and part of speech n-grams (Atwell and Elliot, 1987; Gale and Church, 1990; Mays et al., 1991; Church and Gale, 1991), in which a low-probability word sequence is used to detect real-word errors and high-probability ones are used to rank correction candidates. At that time, all these n-gram language models were based on tri-grams, bi-grams and uni-grams as the higher n-grams were computationally expensive to generate until Google released the Web 1T data set. As a result, they did not capture long-range dependencies. A different approach is to treat context-based spelling correction as a problem of ambiguity resolution. The ambiguity among words is modeled by confusion sets. Rather than attempting to detect and correct all errors, this approach attempts to choose between commonly confused words (such as there vs. their). The first efforts within this framework for spelling correction include Bayesian classifiers and decision lists (Gale et al., 1994; Yarowsky, 1994; Golding, 1995).

Golding (1995) used a Bayesian hybrid method for context-sensitive spelling correc-

tion by using the presence of particular words within some distance of the ambiguous target word and the pattern of words and part-of-speech tags around the target word, and by using these as features to train Bayesian classifiers to select the correct target word. In (Golding, 1995), decision lists are first used to choose the proper word from a confusion set. It was determined that better results can be obtained when evidence is combined in a more powerful way, by taking into consideration not just the strongest piece of evidence, but all the available evidence. Golding (1995) ran the same experiments with Bayesian classifiers, and achieved a small improvement over decision lists. In (Golding and Schabes, 1996), an extension is made to the Bayesian classifier, where a part of speech tagger is first used to tag the text. When all words in a confusion set differ in part of speech, the tagger is used to determine the most likely tag, and thereby the most likely word, in a context. When words do not differ in part of speech, the original Bayesian classifier is used. This resulted in an improvement over the original Bayesian classifier. Golding and Roth (1999) combine the Winnow algorithm, a multiplicative weight-updating algorithm which achieves a good accuracy by being able to handle a large number of features, with weighted-majority voting, using nearby and adjacent words as features. Mangu and Brill (1997) describe an approach to automatically learn linguistic knowledge for spelling correction. A major feature of this approach is the fact that the acquired knowledge is captured in a small set of easily understood rules, as opposed to a large set of opaque features and weights. Potentially confusable words are grouped into sets, from which the appropriate one is selected on the basis of surrounding lexical items and POS tags.

Domeij et al. (1999) describe the construction of a surface-oriented system that can detect and suggest corrections for a number of grammatical errors in Swedish texts using carefully constructed error detection rules. The error detection rules are optimized using statistics of part-of-speech bigrams and words in such a way that each rule needs to

be checked as seldom as possible. The system combines probabilistic and rule-based methods to achieve high efficiency and robustness.

2.1.3 Text Correction as Classification

In some text correction approaches, the prediction is typically framed as a classification task for a specific linguistic class, e.g., prepositions, near-synonym choices, or a set of predefined classes. The classifier is trained on a large amount of well-written English text, where features are extracted from the context of the target word. A system is designed in (Felice, 2008; Felice and Pulman, 2008) to automatically acquire models of occurrence for English prepositions and determiners to allow for the detection and correction of errors in their usage, especially in the writing of non-native speakers of the language. The contexts of these parts of speech are represented by a sophisticated feature set, incorporating a variety of semantic and syntactic elements. Felice (2008) concludes that the construction of contextual models is a viable approach to the task of preposition and determiner selection, despite outstanding issues pertaining to the domain of non-native writing. Related work on preposition error correction and near-synonym choice are discussed in more detail in Section 4.2 and 3.2, respectively.

2.1.4 Text Scoring

Research in text scoring has mostly been done at the document level. Systems for automatic assessment of student essays such as *e-rater* (Attali and Burstein, 2006; Burstein et al., 2004) usually provide scores on the basis of grammatical, stylistic, and coherence issues within essays written in English by both L1 and L2 speakers. Ishioka and Kameda (2006) develop an automated Japanese essay scoring system that needs expert writings rather than expert raters to build the evaluation model. Some of the features that are examined in their system are: syntactic variety, or the use of various structures in the ar-

rangement of phases, clauses, and sentences, characteristics associated with the orderly presentation of ideas, such as rhetorical features and linguistic cues, and vocabulary related to the topic, such as relevant information and precise or specialized vocabulary.

2.2 Using Parsers

In this paradigm, a full syntactic analysis of the sentence is done using a parser to detect errors and propose corrections. We further categorize this paradigm into two different groups: those that constrain rules of the grammar, and those that use error-production rules.

2.2.1 Rules Relaxation

Most of the approaches of this category use some grammar rules that are progressively relaxed until the sentence can be parsed. Fouvry (2003) present a definition of unification of weighted feature structures designed to deal with constraint relaxation such that inconsistent values do not lead to failure, but are penalized and these penalties are based on the signature and the shape of the feature structures, and thus realize an elegant and general approach to relaxation. Vogel and Cooper (1995) introduce a parser that prefers maximally consistent interpretations, and accommodates inconsistencies by discovering the minimal set of constraints that need to be relaxed for a parse to go through, without employing backtracking or post processing.

2.2.2 Error-production Rules

In this category, error rules are constructed to model errors in input sentences and each of these rules corresponds to a specific category of grammatical errors. Foster and Vogel (2004) present a robust parsing approach based on the concept of an error grammar,

a grammar of ungrammatical sentences, derived from a conventional grammar on the basis of an analysis of a corpus of observed ill-formed sentences. The robust parsing algorithm, which combines a rule from the error grammar with rules from the normal grammar to arrive at a parse for an ungrammatical sentence, is applied after a conventional bottom-up parsing algorithm has failed. Wagner et al. (2007) propose two approaches: a deep linguistic processing approach using a parser and an English grammar and a POS n-gram-based shallow processing approach to the automatic detection of common grammatical errors and they show that combining the deep and shallow approaches gives an additional improvement on all but one error type. Andersen (2007) investigates how evidence from corpora can efficiently be used, e.g., through parsing, to find ungrammatical constructions, specifically limited to those that can be identified as such in absence of extra-sentential context.

Michaud et al. (2000) describe a prototype implementation of a system that will take a piece of text written by a deaf student, analyze that text using an English grammar augmented with error-production rules, or mal-rules that allow sentences containing errors to be parsed with the grammar, and enable the system to flag errors when they occur, and engage that student in a tutorial dialogue, enabling the student to generate appropriate corrections to the text. Michaud and McCoy (2001) improve the above-mentioned system by modeling the user's L2 proficiency based on past interactions. For example, in cases where multiple corrections are possible for an ill-formed sentence, the model guides to select the best correction, and tailors the feedback message according to the perceived L2 proficiency of the user. Dan et al. (2004) present a tutorial system for language learners, using a computational grammar augmented with mal-rules for analysis, error diagnosis, and semantics-centered generation of corrected forms. They evaluate the effectiveness of the aligned generation strategy by randomly selecting a sample of 221 items from the Japanese Learners of English corpus (Izumi et al., 2005; Izumia et al.,

2004), representing the error types they attempt to handle in their prototype and balance to reflect the proportion of those errors over the whole corpus, including both single-error and multiple-error sentences. Park et al. (1997) present a prototype grammar checker for English as a Second Language (ESL) students, utilizing Combinatory Categorial Grammar (CCG) where instead of attempting to handle all possible grammatical errors, the grammar checker identifies certain specific types of grammatical mistakes that appear more regularly than others in the domain of application.

2.2.3 Robust Parsing

In some approaches, the goal is not to correct grammatical errors, but rather to ensure that the syntactic analysis is robust. Lee (2009) proposes a method to improve robustness in the correction of verb forms. Lee (2009) views the confusions among verb forms as symptoms of one of the two main underlying categories of errors: semantic and syntactic. To give an example of semantic error (according to Lee (2009)), in the sentence, "*She lives here since January.*", the verb "*live*" is used in the simple present tense, rather than the perfect progressive. Depending on the context, either "*has been living*" or "*had been living*" may be the correction. Without any temporal information, correction of tense would be even more challenging. The sentence, "*She have been living here since January.*", where the verb is not correctly inflected in number and person with respect to the subject, is an example of syntactic error. Lee (2009) builds on the basic approach of template-matching on parse trees in two ways. To improve recall, he used the observed tree patterns for a set of verb form usages and introduced verb form errors into the AQUAINT corpus of English news text, then re-parsed the corpus, and compiled the changes in the "disturbed" trees into a catalog. To improve precision, he utilized the Web 1T 5-gram corpus as filters.

2.3 Machine Translation

One of the most recent paradigms of grammar checking is to map the task as a translation from *bad English* into *good English*. This task then essentially allows one to leverage and use existing methods in Statistical Machine Translation (SMT). The basic requirement of these methods is to have large parallel corpora of the source (i.e., incorrect texts) and the target (i.e., corrected version of those incorrect texts) languages.

Brockett et al. (2006) present the use of a phrasal SMT techniques to identify and correct writing errors made by ESL (English as a Second Language) learners where the identification of an error is triggered by any difference between the input texts and the model's target texts stored in the system. Using mass noun errors found in the Chinese Learner Error Corpus (CLEC) to guide creation of an engineered training set, they show that application of the SMT paradigm can capture errors not well addressed by widely-used proofing tools designed for native speakers.

2.4 Work Based on the Google Web 1T

The idea of using the Google Web 1T n-gram data set as a resource in different natural language processing applications has been exploited by many researchers. Nulty and Costello (2009) deduce the semantic relation that holds between two nouns in a noun-noun compound phrase such as "flu virus" or "morning exercise" using lexical patterns in the Google Web 1T corpus. Klein and Nelson (2009) investigate the relationship between *term count* (*TC*) and *document frequency* (*DF*) values of terms occurring in the Web as Corpus (WaC) and also the similarity between *TC* values obtained from the WaC and the Google n-gram dataset and they mention that a strong correlation between the two would give them confidence in using the Google n-grams to estimate accurate inverse document frequency (IDF) values in order to generate well-performing

lexical signatures based on the TF-IDF scheme. Murphy and Curran (2007) explore the strengths and limitations of Mutual Exclusion Bootstrapping (MEB) by applying it to two novel lexical-semantic extraction tasks: extracting bigram named entities and WordNet lexical file classes (Fellbaum, 1998) from the Google Web 1T 5-grams.

Curran et al. (2007) use the Google Web 1T 5-grams to demonstrate the superiority of MEB (Mutual Exclusion Bootstrapping) to standard bootstrapping in extracting named entities. In their paper (Curran et al., 2007) they say:

> ... using 2% of the data by type and 3.6% of the data by token. However, the number of terms and contexts by type is still extremely large. The data set is 666MB on disk which all needs to be loaded into memory at once.

Krishnakumaran and Zhu (2007) use the Web 1T 2-grams to automatically classify sentences into metaphoric or normal usages. Carlson and Fette (2007) study the problem of correcting spelling mistakes in text using memory-based learning techniques and use the Web 1T n-gram occurrences as training data. Their approach uses the context in which an error appears to select the most likely candidate from words which might have been intended in its place. Carlson and Fette (2007) discuss the limitations of this data set; for example:

> This dataset is 87GiB on disk, without any database indexes. This may limit uses, as the data is far too large to distribute with client applications. However, the dataset is still useable in an online scenario where clients are contacting centralized servers, as the query load is light and the queries themselves can be executed quickly. To enable fast queries for our application, we loaded the data set into a MySQL database and indexed each table, which brought the total size of the database up to roughly 250GiB.

White et al. (2007) experiment with filtering the Web 1T data using the vocabulary from the CCGbank (Hockenmaier, 2003). They mention the difficulties of using this data set:

> We are also investigating language models using the 5-gram counts that Google has made available, generated from 1 trillion tokens of text from web pages. The SRILM toolkit now has count-based language models that can read this data, but they require inordinate amounts of RAM.

Hassan et al. (2007) use the Web 1T to model the semantic fit of a candidate substitute within the given context using a language model to combine knowledge sources for automatic lexical substitution. Trnka and McCoy (2007) measure the out of vocabulary (OOV) words in reference to the vocabulary from the Web 1T 5-gram corpus in the task of word prediction from corpus studies, showing the percentage of OOV words for seven other corpora they use. Trnka and McCoy (2007) plan to expand this study to include a very large, general-purpose language model, such as the Web 1T 5-gram language model as they believe the heterogeneous nature of the web should cause this model to be a suitable general-purpose model to serve as the base language modeling component of an interpolated model.

McCarthy et al. (2007) show that selectional preference models produced from distributional thesauri are the most promising; this is encouraging as the technique could be applied to a language without a manual thesaurus and it would also be worthwhile comparing the use of raw data directly, both from the British National Corpus (BNC) and from Google's Web 1T corpus since web counts have been shown to outperform the Clark and Weir models (Clark and Weir, 2002) on a pseudo-disambiguation task (Keller and Lapata, 2003).

Ringlstetter et al. (2007) describe a series of experiments where word bigram counts from domain-dependent web corpora are used to improve a practical text correction system. To guarantee the efficiency, bigram counts for arbitrary bigrams over large dictionaries are kept in main memory, using special techniques. It was shown that bigram counts improve simpler scores based on word similarity and word frequency only.

Ringlstetter et al. (2007) conclude:

> It seems interesting to see what improvements can be obtained using the large list of n-grams recently made available by Google.

Some of the researchers (e.g., Krishnakumaran and Zhu, 2007; Hassan et al., 2007; Trnka and McCoy, 2007; McCarthy et al., 2007) did not mention how efficiently they manage to store and use the Web 1T data set. Evert (2010) introduces *Web1T5-Easy*, a simple indexing solution that allows interactive searches of the Web 1T 5-gram database and a derived database of quasi-collocations.

Islam and Inkpen (2009a) describe how the Google Web 1T 5-gram data set can efficiently be used and stored. Islam and Inkpen (2009b) use 3-grams of this data set to detect and correct real-word spelling errors and also use n-grams to only correct real-word spelling errors (Islam and Inkpen, 2009c,d), preposition errors (Islam and Inkpen, 2010b). Islam and Inkpen (2010a) use a 5-gram language model with the help of this data set for automatic choice of near-synonyms. Islam et al. (2008) use corpus-based semantic similarity and word segmentation to database schema matching where they use the BNC to generate n-grams for the words processed by their method. They use the Second Order Co-occurrence Pointwise Mutual Information (SOC-PMI) word similarity method (Islam and Inkpen, 2006) that uses PMI values to sort lists of important neighbor words of the two target words from the BNC. The SOC-PMI word similarity method considers the words that are common in both lists and aggregate their PMI values (from the opposite list) to calculate the relative semantic similarity. Having no off-the-shelf list of n-grams for the BNC means their method needs to generate it first in order to process any specific word and it goes without saying that this approach is not time efficient. As a result the method proposed by Islam et al. (2008) cannot be used on the fly. Efficient using of the Web 1T data can solve the problem of having no n-gram lists for the BNC.

Chapter 3

Unsupervised Approaches to Near-Synonym Choice

In this chapter, an unsupervised statistical method for automatic choice of near-synonyms using the Google Web 1T n-gram data set is presented and compared with the state-of-the-art. Figure 3.1 shows the flow diagram of the near-synonym choice problem, where the task is to select the best near-synonym from a set of near-synonyms that could be used in a particular context. As a set of near-synonyms is given, we do not need any

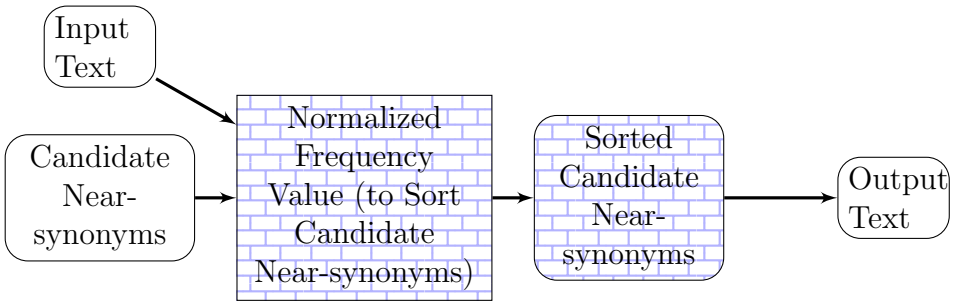

Figure 3.1: Flow diagram of near-synonym choice.

procedure to generate candidate words/near-synonyms, which distinguishes Figure 3.1 from Figure 1.3 on page 10. Thus, the task is to sort the candidate near-synonyms. We use the normalized frequency value function to sort the near-synonyms. The method uses

the normalized frequency value function in two phases, if required, and uses the Google Web 1T data set in a 'back off fashion' that increases the performance of the method. The proposed method does not require any human-annotated knowledge resources. We also compare it to using a standard 5-gram language model built from the Google Web 1T data set. Our evaluation experiments show that the two-phase method significantly outperforms the 5-gram language model and three previous methods on the same task. This work is applicable to an intelligent thesaurus, machine translation, and natural language generation.

3.1 Introduction

Choosing the wrong near-synonym can convey unwanted connotations, implications, or attitudes. In machine translation and natural language generation systems, the choice among near-synonyms needs to be made carefully. In addition to paying attention to lexical nuances, when choosing a word we need to make sure it fits well with the other words in a sentence. In this chapter we investigate how the collocational properties of near-synonyms can help with choosing the best words. This problem is difficult because the near-synonyms have senses that are very close to each other, and therefore they occur in similar contexts. We build a strong representation of the context in order to capture the more subtle differences specific to each near-synonym.

The work we present here can be used in an intelligent thesaurus. A writer can access a thesaurus to retrieve words that are similar to a given word, when there is a need to avoid repeating the same word, or when the word does not seem to be the best choice in the context. A standard thesaurus does not offer any explanation about the differences in nuances of meaning between the possible word choices.

This work can also be applied to a natural language generation system (Inkpen and Hirst, 2003) that needs to choose among near-synonyms. Inkpen and Hirst (2003) in-

cluded a preliminary collocation module that reduces the risk of choosing a near-synonym that does not fit with the other words in a generated sentence (i.e., violates collocational constraints). The work presented in this chapter allows for a more comprehensive near-synonym collocation module.

The task we address in this chapter is the selection of the best near-synonym that should be used in a particular context. Inkpen (2007) argues that the natural way to validate an algorithm for this task would be to ask human readers to evaluate the quality of the algorithm's output, but this kind of evaluation would be very laborious. Instead, Inkpen (2007) validates her algorithms by deleting selected words from sample sentences, to see whether the algorithms can restore the missing words. That is, Inkpen (2007) creates a *lexical gap* and evaluates the ability of the algorithms to fill the lexical gap. Two examples from (Inkpen, 2007) are presented in Figure 3.2. All the near-synonyms of the original word, including the word itself, become the choices in the solution set (see the figure for two examples of solution sets). The task is to automatically fill the gap with the best choice in the particular context. We present a method that can be used to scoring the choices. For our particular task, we choose only the highest scoring near-synonym. In order to evaluate how well our method works we consider that the only correct solution is the original word. This will cause our evaluation scores to underestimate the performance of our method, as more than one choice will sometimes be a perfect solution. Moreover, what we consider to be the best choice is the typical usage in the corpus, but it may vary from writer to writer. Nonetheless, it is a convenient way of producing test data in an automatic way. To verify how difficult the task is for humans, Inkpen (2007) performed experiments with human judges on a sample of the test data.

The near-synonym choice method that we propose here uses the Google Web 1T n-gram data set (Brants and Franz, 2006). This chapter is organized as follow: Section 3.2 presents a brief overview of the related research. In Section 3.3, we describe the 5-gram

Sentence: This could be improved by more detailed consideration of the processes of ..... propagation inherent in digitizing procedures.
Original near-synonym: error
Solution set: mistake, blooper, blunder, boner, contretemps, error, faux pas, goof, slip, solecism

Sentence: The day after this raid was the official start of operation strangle, an attempt to completely destroy the ..... lines of communication.
Original near-synonym: enemy
Solution set: opponent, adversary, antagonist, competitor, enemy, foe, rival

Figure 3.2: Examples of sentences with a lexical gap, and candidate near-synonyms to fill the gap.

language model. The two-phase method is described in Section 3.4. Evaluation and experimental results are discussed in Section 3.5. We summarize in Section 3.6.

3.2 Related Research

Turney et al. (2003) addressed the multiple-choice synonym problem: given a word, choose a synonym for that word, among a set of possible solutions. In this case the solutions contain one synonym and some other (unrelated) words. They achieves high performance by combining classifiers. Clarke and Terra (2003) addressed the same problem as Turney *et al.*, using statistical associations measures computed with counts from the Waterloo terabyte corpus. In our case, all the possible solutions are synonyms of each other, and the task is to choose one that best matches the context: the sentence in which the original synonym is replaced with a gap. It is much harder to choose between words that are near-synonyms because the context features that differentiate a word from other words might be shared among the near-synonyms.

Inkpen's (2007) unsupervised method is based on the mutual information scores between a near-synonym and the content words in the context filtering out the stopwords[1].

[1]We do not filter out stopwords or punctuation in our method.

The *pointwise mutual information* (PMI) between two words x and y compares the probability of observing the two words together (their joint probability) to the probabilities of observing x and y independently (the probability of occurring together by chance) (Church and Hanks, 1991).

$$PMI(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

The probabilities can be approximated by: $P(x) = C(x)/N$, $P(y) = C(y)/N$, $P(x, y) = C(x, y)/N$, where C denote frequency counts and N is the total number of words in the corpus. Therefore,

$$PMI(x, y) = \log_2 \frac{C(x, y) \cdot N}{C(x) \cdot C(y)}$$

where N can be ignored in comparisons, since is it the same in all the cases. Inkpen (2007) models the context as a window of size $2k$ around the gap (the missing word): k words to the left and k words to the right of the gap. If the sentence is
$s = \cdots w_1 \cdots w_k \quad Gap \quad w_{k+1} \cdots w_{2k} \cdots$, for each near-synonym NS_i from the group of candidates, the score is computed by the following formula:

$$Score(NS_i, s) = \Sigma_{j=1}^{k-1} PMI(NS_i, w_j) \; + \Sigma_{j=k+1}^{2k} PMI(NS_i, w_j).$$

In a supervised learning method, Inkpen (2007) trains classifiers for each group of near-synonyms. The classes are the near-synonyms in the solution set. Each sentence is converted into a vector of features to be used for training the supervised classifiers. Inkpen used two types of features. One type of features consists in the scores of the left and right context with each class (i.e., with each near-synonym from the group). The number of features of this type is equal to twice the number of classes: one feature for the score between the near-synonym and the part of the sentence at the left of the gap, and one feature for the score between the near-synonym and the part of the sentence

at the right of the gap. The second type of features is formed by the words in the context windows. For each group of near-synonyms, Inkpen used as features the 500 most frequent words situated close to the gaps in a development set. The value of a word feature for each training example is 1 if the word is present in the sentence (at the left or at the right of the gap), and 0 otherwise. Inkpen trained classifiers using several machine learning algorithms to see which one is best at discriminating among the near-synonyms.

There has been quite a lot of work in unsupervised learning of word clusters based on n-grams (e.g., Brown et al., 1992; Pereira et al., 1993; Lin, 1998). Our task has similarities to the word sense disambiguation task. Our near-synonyms have senses that are very close to each other. In Senseval, some of the fine-grained senses are also close to each other, so they might occur in similar contexts, while the coarse-grained senses are expected to occur in distinct contexts. In our case, the near-synonyms are different words to choose from, not the same word with different senses.

In fact, the works that address exactly the same task are that of Edmonds (1997) and Inkpen (2007), as far as we are aware. Edmonds (1997) gives a solution based on a lexical co-occurrence network that included second-order co-occurrences, whereas Inkpen (2007) uses a much larger corpus and a simpler method, and obtains better results than that of Edmonds (1997). The SemEval 2007 lexical substitution task (McCarthy and Navigli, 2009) is somewhat similar to the near-synonym choice task, except for some subtle differences. In near-synonym choice, a set of candidates is given and the task is to find the best candidate from the set, whereas in SemEval 2007, the task is two-fold: 1) finding the set of candidate substitutes for the word, and then 2) finding the best candidate (excluding the original word). That is, the lexical substitution task allows a representation of meaning that does not need a pre-defined list of senses and the systems are free to select these words/phrases. So, the main difference between the two tasks is the question of whether the set of candidates is given or not. Another difference is that

the near-synonym choices have very similar meanings, while in the lexical substitution task the choices are for different meaning. In SemEval 2007 task, some of the researchers (e.g., Hassan et al., 2007; Giuliano et al., 2007; Yuret, 2007; Dahl et al., 2007; Hawker, 2007) use the Web 1T in a very simple way. Our method is more complex (it has two phases), and it can directly be applied to the second step of the lexical substitution task.

3.3 The n-gram Language Model

A *language model* is usually formulated as a probability distribution $P(S)$ over strings S , and attempts to reflect how frequently a string S occurs as a sentence. The most widely-used language models, by far, are n-gram language models (Chen and Goodman, 1998). We introduce these models by considering the case $n = 5$; these models are called 5-gram models. For a sentence S composed of the words $w_1 \cdots w_p$, without loss of generality we can express $P(S)$ as

$$\begin{aligned} P(S) &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_p|w_1\cdots w_{p-1}) \\ &= \prod_{i=1}^{p} P(w_i|w_1\cdots w_{i-1}) \end{aligned} \tag{3.1}$$

For n-gram models where n >2, we condition the probability of a word on the identity of the last n-1 words. Generalizing equation 3.1 to n >2, we get

$$P(S) = \prod_{i=1}^{p+1} P(w_i|w_{i-n+1}^{i-1}) \tag{3.2}$$

where w_i^j denotes the words $w_i \cdots w_j$. To estimate $P(w_i|w_{i-n+1}^{i-1})$, the frequency with which the word w_i occurs given that the words w_{i-n+1}^{i-1} precede the current word w_i, we simply count how often the n-gram w_{i-n+1}^{i} occurs in some text and normalize by the total number of occurrences of any word in the same context. Let $C(w_{i-n+1}^{i})$ denote the

number of times the n-gram w_{i-n+1}^{i} occurs in the given text. Then

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{C(w_{i-n+1}^{i})}{\sum_{w_i} C(w_{i-n+1}^{i})} \tag{3.3}$$

Notice that $C(w_{i-n+1}^{i-1}) \geq \sum_{w_i} C(w_{i-n+1}^{i})$. To understand the inequality of $C(w_{i-n+1}^{i-1})$ and $\sum_{w_i} C(w_{i-n+1}^{i})$, assume that both i and n are 5. Then, $C(w_{i-n+1}^{i-1})$ becomes $C(w_1w_2w_3w_4)$, which is actually the frequency of a specific 4-gram, $w_1w_2w_3w_4$, and $\sum_{w_i} C(w_{i-n+1}^{i})$ becomes $\sum_{w_5} C(w_1w_2w_3w_4w_5)$, which is the sum of the frequencies of all the 5-grams that start with the 4-gram, $w_1w_2w_3w_4$. Thus, in general, $C(w_1w_2w_3w_4)$ is equal to $\sum_{w_5} C(w_1w_2w_3w_4w_5)$; but, for some specific cases, it is possible that $C(w_1w_2w_3w_4)$ is greater than $\sum_{w_5} C(w_1w_2w_3w_4w_5)$. For example, all the n-grams ($2 \leq n \leq 5$) that appear less than 40 times have been filtered out from the Web 1T n-grams. This means all the 5-grams (starting with the 4-gram $w_1w_2w_3w_4$) that appear less than 40 times have not been included in $\sum_{w_5} C(w_1w_2w_3w_4w_5)$. Thus, when we deal with the Web 1T 5-grams and 4-grams, it is obvious that $C(w_1w_2w_3w_4)$ is greater than or equal to $\sum_{w_5} C(w_1w_2w_3w_4w_5)$. Thus, in general, we can say that $C(w_{i-n+1}^{i-1}) \geq \sum_{w_i} C(w_{i-n+1}^{i})$. This also supports the idea of using the missing count (equation 3.4) in the smoothing formula (equation 3.5).

We use a smoothing method loosely based on the *one-count* method described in (Chen and Goodman, 1996). Because tokens that appears less than 200 times and n-grams that appear less than 40 times have been filtered out from the Web 1T, we use n-grams with missing counts instead of n-grams with one counts (Yuret, 2007). The missing count is defined as:

$$M(w_{i-n+1}^{i-1}) = C(w_{i-n+1}^{i-1}) - \sum_{w_i} C(w_{i-n+1}^{i}) \tag{3.4}$$

The corresponding smoothing formula is:

$$P(w_i|w_{i-n+1}^{i-1}) =$$

$$\frac{C(w_{i-n+1}^{i}) + (1+\alpha_n)M(w_{i-n+1}^{i-1})P(w_i|w_{i-n+2}^{i-1})}{C(w_{i-n+1}^{i-1}) + \alpha_n M(w_{i-n+1}^{i-1})} \tag{3.5}$$

Yuret (2007) optimized the parameters $\alpha_n > 0$ for $n = 2 \cdots 5$ on the Brown corpus to yield a cross entropy of 8.06 bits per token. The optimized parameters are: $\alpha_2 = 6.71$, $\alpha_3 = 5.94$, $\alpha_4 = 6.55$, $\alpha_5 = 5.71$

Thus, incorporating the smoothing formula in equation 3.2, we get

$$P(S) =$$

$$\prod_{i=1}^{p+1} \frac{C(w_{i-n+1}^{i}) + (1+\alpha_n)M(w_{i-n+1}^{i-1})P(w_i|w_{i-n+2}^{i-1})}{C(w_{i-n+1}^{i-1}) + \alpha_n M(w_{i-n+1}^{i-1})} \tag{3.6}$$

3.3.1 Our Task

Our task is to find the best near-synonym from a set of candidates that could fill in the gap in an input text, using the Google Web 1T data set. Let us consider an input text

T which after tokenization[2] has p $(2 \leq p \leq 9)$ words[3], i.e.,

$$T = \{\ldots\ w_{i-4}\, w_{i-3}\, w_{i-2}\, w_{i-1}\, \boxed{w_i}\, w_{i+1}\ w_{i+2}\, w_{i+3}\, w_{i+4}\ \ldots\}$$

where $\boxed{w_i}$ (in position i) indicates the gap and w_i in $\boxed{w_i}$ denotes a set of m near-synonyms (i.e., $w_i = \{s_1, s_2, \cdots, s_j, \cdots, s_m\}$) . We take into account at most four words before the gap and at most four words after the gap. Our task is to choose the $s_j \in w_i$ that best matches with the context. In other words, the position i is the gap that needs to be filled with the best suited member from the set, w_i.

We construct m strings $(S_1 \cdots S_m)$ replacing the gaps in position i with $s_j \in w_i$ as follows:

$$S_1 = \cdots w_{i-4} w_{i-3} w_{i-2} w_{i-1} s_1 w_{i+1} w_{i+2} w_{i+3} w_{i+4} \cdots$$

$$S_2 = \cdots w_{i-4} w_{i-3} w_{i-2} w_{i-1} s_2 w_{i+1} w_{i+2} w_{i+3} w_{i+4} \cdots$$

$$\vdots$$

$$S_m = \cdots w_{i-4} w_{i-3} w_{i-2} w_{i-1} s_m w_{i+1} w_{i+2} w_{i+3} w_{i+4} \cdots$$

Using equation 3.7 in Section 3.3.2, we calculate $P(S_1), \cdots, P(S_m)$. The index of the

[2]We need to tokenize the input sentence to make the n-grams formed using the tokens returned after the tokenization consistent with the Google n-grams. The input sentence is tokenized in a manner similar to the tokenization of the Wall Street Journal portion of the Penn Treebank. Notable exceptions include the following:

- Hyphenated word are usually separated, and hyphenated numbers usually form one token.
- Sequences of numbers separated by slashes (e.g., in dates) form one token.
- Sequences that look like urls or email addresses form one token.

[3]If the input text has more than 9 words then we keep at most four words before the gap and at most four words after the gap to make the length of the text 9. We choose these numbers so that we could maximize the number of n-grams to use, given that we have up to 5-grams in the Google Web 1T data set.

target synonym, j, will be $\underset{j\in\{1\cdots m\}}{\text{argmax}}\ P(S_j)$.

3.3.2 The Language Model Used for Our Task

For the specified task, we simplify the 5-gram model described in Section 3.3.1. From equation 3.2, it is clear that the maximum number of products possible is 10 as $2 \leq p \leq 9$. Among these products, we can omit the products $P(w_{i-1}|w_{i-5}^{i-2})$, $P(w_{i-2}|w_{i-6}^{i-3})$, $P(w_{i-3}|w_{i-7}^{i-4})$, $P(w_{i-4}|w_{i-8}^{i-5})$, and $P(w_{i+5}|w_{i+1}^{i+4})$ because these product items have the same values for all $j \in \{1 \cdots m\}$. Thus, the five product items that we consider are: $P(w_i|w_{i-4}^{i-1})$, $P(w_{i+1}|w_{i-3}^{i})$, $P(w_{i+2}|w_{i-2}^{i+1})$, $P(w_{i+3}|w_{i-1}^{i+2})$, and $P(w_{i+4}|w_{i}^{i+3})$. Applying this simplification in equation 3.2 and equation 3.6, we get

$$\begin{aligned} P(S) &= \prod_{i=5}^{p} P(w_i|w_{i-n+1}^{i-1}) \\ &= \prod_{i=5}^{p} \frac{C(w_{i-n+1}^{i}) + (1+\alpha_n)M(w_{i-n+1}^{i-1})P(w_i|w_{i-n+2}^{i-1})}{C(w_{i-n+1}^{i-1}) + \alpha_n M(w_{i-n+1}^{i-1})} \end{aligned} \tag{3.7}$$

Equation 3.7 is actually used in a recursive way for n=5,4,3,2,1 (i.e., if the current n-gram count is zero, then it is used with a lower n-gram, and so on). For example, $P(w_5|w_1w_2w_3w_4)$ is a function of $C(w_1w_2w_3w_4w_5)$ and $P(w_5|w_2w_3w_4)$; if $C(w_1w_2w_3w_4w_5) > 0$ then we do not consider $P(w_5|w_2w_3w_4)$. This is a backoff language model.

3.4 The Two-Phase Method

Our task is to choose the $s_j \in w_i$ that best matches with the contexts. In other words, the position of i is the gap that needs to be filled with the best suited member from the set, w_i. First, we discuss how we categorize different n-grams based on the gap position and how we find the *normalized frequency value*. Then, we discuss the procedure to find the best choice near-synonym using the n-gram data set.

3.4.1 Categorizing n-gram Type Based on the Gap Position

We categorize an n-gram type (we call it, k) based on the position of the gap in the n-gram. When we consider the 5-gram data set, there might be five types of 5-grams (i.e., $k \in \{1, 2, 3, 4, 5\}$) based on the position of the gap in the 5-gram. For example, if the gap position is the last position in the 5-gram then we call it type 1 (i.e., $k = 1$). Thus, a 5-gram, $w_{i-4}\, w_{i-3}\, w_{i-2}\, w_{i-1}\, \boxed{s_j}$, could be represented incorporating k as $w_{(i-4)k}\, w_{(i-3)k}\, w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}}$. All the five types of 5-grams are as follows:

$$w_{(i-4)k}\, w_{(i-3)k}\, w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}} \quad (k = 1)$$

$$w_{(i-3)k}\, w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}}\, w_{(i+1)k} \quad (k = 2)$$

$$w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k} \quad (k = 3)$$

$$w_{(i-1)k}\, \boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k}\, w_{(i+3)k} \quad (k = 4)$$

$$\boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k}\, w_{(i+3)k}\, w_{(i+4)k} \quad (k = 5)$$

Similarly, all the four types of 4-grams are:

$$w_{(i-3)k}\, w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}} \qquad (k = 1)$$

$$w_{(i-2)k}\, w_{(i-1)k}\, \boxed{s_{jk}}\, w_{(i+1)k} \qquad (k = 2)$$

$$w_{(i-1)k}\, \boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k} \qquad (k = 3)$$

$$\boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k}\, w_{(i+3)k} \qquad (k = 4)$$

The three types of 3-grams are as follows:

$$w_{(i-2)k}\, w_{(i-1)k} \boxed{s_{jk}} \qquad (k = 1)$$

$$w_{(i-1)k} \boxed{s_{jk}}\, w_{(i+1)k} \qquad (k = 2)$$

$$\boxed{s_{jk}}\, w_{(i+1)k}\, w_{(i+2)k} \qquad (k = 3)$$

And the two types of 2-grams are:

$$w_{(i-1)k} \boxed{s_{jk}} \qquad (k = 1)$$

$$\boxed{s_{jk}}\, w_{(i+1)k} \qquad (k = 2)$$

3.4.2 Normalized Frequency Value

We determine the *normalized frequency value* of each candidate choice for the gap position with respect to all other candidates. If we have m candidate choices for the gap position, i, which are $\{s_1, s_2, \cdots, s_j, \cdots, s_m\}$, and their frequencies $\{f_1, f_2, \cdots, f_j, \cdots, f_m\}$, where f_j is the frequency of a n-gram (where $n \in \{5, 4, 3, 2\}$ and any candidate near-synonym choice s_j is a member of the n-gram), then we determine the normalized frequency value of any candidate near-synonym choice s_j as the frequency of the n-gram containing s_j, over the maximum frequency among all the candidate near-synonym choices for that position.

$$F(s_j) = \frac{f_j}{\max(f_1, f_2, \cdots, f_j, \cdots, f_m)} \tag{3.8}$$

Now, based on the types of n-gram, k, equation 3.8 can be written as:

$$F(s_{jk}) = \frac{f_{jk}}{\max(f_{1k}, f_{2k}, \cdots, f_{jk}, \cdots, f_{mk})} \tag{3.9}$$

3.4.3 Determining the Choice Word (Phase 1)

In Phase 1, we first use the Google 5-gram data set to find the best choice near-synonym. If the 5-gram data set fails to generate a choice then we move forward to the 4-gram data set, the 3-gram data set, or the 2-gram data set if needed. We apply Phase 2 only if the n-gram (where $n \in \{5, 4, 3, 2\}$) data set fails to generate at least one choice.

Determining the Choice Word using the 5-gram Data Set

First, we determine f_{11}, which is the frequency of the 5-gram $w_{(i-4)1}\, w_{(i-3)1}\, w_{(i-2)1}\, w_{(i-1)1}\, \boxed{s_{11}}$ with a specific type $k = 1$, where the last word of the 5-gram is the first synonym choice among m. Similarly, we determine f_{j1}, for all $j \in \{2 \cdots m\}$. Now, we determine $F(s_{j1})$ using equation 3.9, where $k = 1$. In the same way, we determine $F(s_{j2})$, $F(s_{j3})$, $F(s_{j4})$ and $F(s_{j5})$. Now, index of synonym

$$j = \begin{cases} j & \text{if } |\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})| = 1 \\ 0 & \text{if } |\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})| > 1 \end{cases} \tag{3.10}$$

For simplicity, we assume that $\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})$ returns the set of values[4] of $j \in \{1 \cdots m\}$ for which $\sum_{k=1}^{5} F(s_{jk})$ for all $j \in \{1 \cdots m\}$ attains its maximum value. If the expression $\sum_{k=1}^{5} F(s_{jk})$ returns 0 for all $j \in \{1 \cdots m\}$, then $\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})$ will return the set $\{1 \cdots m\}$. Again, if $\sum_{k=1}^{5} F(s_{jk})$ has unique value for more than one candidates, then $\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})$ will return a set containing the indices of those candidates. Thus, in general, if $\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})$ returns a single index, it means that

[4]Even if it returns a single value, we assume it returns a set containing a single value, though the standard argmax returns a value not a set for single value.

the synonym choice is the word $s_j \in w_i$, we return s_j and exit. If $\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{5} F(s_{jk})$ returns a set of indices containing at least two indices means, then we need to go to the next step to try with 4-grams. This is how ties in the method, if any, are resolved.

Note that in equation 3.10, the values of $\sum_{k=1}^{5} F(s_{jk})$ for all $j \in \{1 \cdots m\}$ are known and from that only the j that maximizes $\sum_{k=1}^{5} F(s_{jk})$ is returned. The values of j with their corresponding expression values, $\sum_{k=1}^{5} F(s_{jk})$ for all $j \in \{1 \cdots m\}$, in descending order can be used to score the choices, if required. For our particular task, we use only the j that maximizes $\sum_{k=1}^{5} F(s_{jk})$ to choose the highest scoring near-synonym.

Determining the Choice Word using the n-gram Data Set where $n \in \{5, 4, 3, 2\}$

We could generalize equation 3.10 by the fact that the upper limit of the summation for this equation is actually $n-1$, where $n \in \{5, 4, 3, 2\}$ is the n-gram that we use. Thus, equation 3.10 can be generalized as:

$$j = \begin{cases} j & \text{if } |\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{n} F(s_{jk})| = 1 \\ 0 & \text{if } |\underset{j \in \{1 \cdots m\}}{\text{argmax}} \sum_{k=1}^{n} F(s_{jk})| > 1 \end{cases} \tag{3.11}$$

Here, we first try equation 3.11 with $n = 5$ and having $j \neq 0$ means that the synonym choice is the word $s_j \in w_i$; we return s_j and exit. Otherwise, we try equation 3.11 with all possible decreasing n until we get $j \neq 0$, and then we return s_j and exit. Otherwise, we go to Phase 2.

3.4.4 Determining the Choice Word (Phase 2)

The question of why we use Phase 2 is best understood by the example " $\cdots$ Klepper suggests that VNRs □ major opportunities for $\cdots$" (Table 3.6) where the gap, □, need to be filled up by any of {*give*, *provide*, *offer*}; but, there is no such 5-gram (e.g., *Klepper*

suggests that VNRs □, *suggests that VNRs* □ *major* and so on), 4-gram (e.g., *suggests that VNRs* □, *that VNRs* □ *major* and so on), 3-gram (e.g., *that VNRs* □, *VNRs* □ *major* and so on), 2-gram (e.g., *VNRs* □). The reason of the unavailability of such n-grams is that "VNRs" is not a very common word in the Google Web 1T data set.

To solve this issue is straightforward. We follow Phase 1 with some small changes: instead of trying to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where only $s_{jk} \in w_i$ is changed while keeping all of $\{\cdots, w_{(i-2)k}, w_{(i-1)k}, \cdots\}$ unchanged, we try to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where $s_{jk} \in w_i$, as well as any but the first member of $\{\cdots, w_{(i-2)k}, w_{(i-1)k}, \cdots\}$ are changed while keeping the remaining of $\{\cdots, w_{(i-2)k}, w_{(i-1)k}, \cdots\}$ unchanged.

3.5 Evaluation and Experimental Results

3.5.1 Comparison to Edmonds's and Inkpen's methods

In this section we present results of the proposed method explained in Section 3.3 and 3.4. We compare our results with those of Edmonds (1997) and Inkpen (2007). Edmonds (1997) solution used the texts from the year 1989 of the Wall Street Journal (WSJ) to build a lexical co-occurrence network for each of the seven groups of near-synonyms from Table 3.9. The network included second-order co-occurrences. Edmonds used the WSJ 1987 texts for testing, and reported accuracies only a little higher than the baseline. Inkpen's (2007) method is based on mutual information, not on co-occurrence counts. Inkpen's counts are collected from a much larger corpus.

For comparison purposes, in this section we use the same test data (WSJ 1987) and the same groups of near-synonyms. The seven groups of near-synonyms used by Edmonds are listed in the first column of Table 3.9. The near-synonyms in the seven groups were chosen to have low polysemy. This means that some sentences with wrong senses of

near-synonyms might be in the automatically produced test set, but hopefully not many.

Before we look at the results, we mention that the accuracy values we compute are the percentage of correct choices when filling in the gap with the winning near-synonym. The expected solution is the near-synonym that was originally in the sentence, and it was taken out to create the gap. This measure is conservative; it does not consider cases when more than one solution is correct.

Some examples of correct and incorrect choices, using the Google 5-grams, 4-grams, 3-grams, and 2-grams (in Phase 1) are shown in Table 3.1, Table 3.2, Table 3.3, and Table 3.4, respectively. Because the Google 5-grams, 4-grams and 3-grams fail to

CORRECT CHOICE:
$\cdots$ viewed widely as a *mistake* $\rightarrow$ mistake [error, mistake, oversight] and a major $\cdots$
$\cdots$ analysts expect stocks to *settle* $\rightarrow$ settle [settle, resolve] into a steady $\cdots$
INCORRECT CHOICE:
$\cdots$ would be a political *mistake* $\rightarrow$ error [error, mistake, oversight] to criticize the $\cdots$
$\cdots$ its energies on the *material* $\rightarrow$ substance [material, stuff, substance] as well as $\cdots$

Table 3.1: Examples of correct and incorrect choices using Google 5-grams. Italics indicate the proposed near-synonym choice returned by the method, arrow indicates the original near-synonym, square brackets indicate the solution set.

generate any suggestion for the examples shown in Table 3.4, we try with Google 2-grams.

Table 3.5, Table 3.6, and Table 3.7 shows some examples of correct and incorrect choices using Google 5-grams (in Phase 2), 4-grams (in Phase 2), and 3-grams (in Phase 2), respectively. Because the Google 5-grams, 4-grams, 3-grams, 2-grams, 5-grams (in Phase 2), and 4-grams (in Phase 2) fail to generate any suggestion for the examples shown in Table 3.7, we try with Google 3-grams (in Phase 2).

CORRECT CHOICE:
··· Sometimes that *task* → task [job, task, duty] is very straightforward ···
··· carry a heavier tax *burden* → burden [responsibility, burden, obligation, commitment] during 1987 because ···
INCORRECT CHOICE:
··· 23 , and must *provide* → give [give, provide, offer] Washington-based USAir at ···
··· Phog Allen – to *resolve* → settle [settle, resolve] the burning question ···

Table 3.2: Examples of correct and incorrect choices using Google 4-grams. For these examples, Google 5-grams fail to generate any suggestion.

CORRECT CHOICE:
··· until his committee 's *oversight* → oversight [error, mistake, oversight] panel completes its ···
··· pressing the government too *hard* → hard [difficult, hard, tough] , fearful that ···
INCORRECT CHOICE:
··· firm that could make *hard* → difficult [difficult, hard, tough] block trades efficiently ···
··· First Boston Corp. will *provide* → offer [give, provide, offer] 12,607,350 in the ···

Table 3.3: Examples of correct and incorrect choices using Google 3-grams. For these examples, Google 5-grams and 4-grams fail to generate any suggestion.

Table 3.8 shows some examples of cases where Google 5-grams, 4-grams, 3-grams, 2-grams, 5-grams (in Phase 2), 4-grams (in Phase 2) and 3-grams (in Phase 2) fail to generate any suggestion. That is, these are some of the examples where our proposed method does not provide any suggestion.

For each case, our method returns either a choice (which is either correct or incorrect) or *no suggestion* which means that the observed word is indeed a correct one. Figure 3.3 shows the number of cases where either a correct choice or an incorrect choice is generated for different combinations of n-grams used. To give an example, using only 5-grams, each

CORRECT CHOICE:
$\cdots$ in Mr. O'Brien 's *job* $\rightarrow$ job [job, task, duty] as chief equity $\cdots$ $\cdots$ association of a feared *substance* $\rightarrow$ substance [material, stuff, substance] with a feared $\cdots$
INCORRECT CHOICE:
$\cdots$ Latin staff is n't *hard* $\rightarrow$ tough [difficult, hard, tough] enough about compelling $\cdots$ $\cdots$ by the SEC commonly *resolve* $\rightarrow$ settle [settle, resolve] the charges by $\cdots$

Table 3.4: Examples of correct and incorrect choices using Google 2-grams. For these examples, Google 5-grams, 4-grams and 3-grams fail to generate any suggestion.

CORRECT CHOICE (in Second Phase):
$\cdots$ agreed that it 's *difficult* $\rightarrow$ difficult [difficult, hard, tough] these days to $\cdots$ $\cdots$ contenders for the Dior *job* $\rightarrow$ job [job, task, duty] . $\cdots$
INCORRECT CHOICE (in Second Phase):
$\cdots$ to the government 's *commitment* $\rightarrow$ obligation [responsibility, burden, obligation, commitment] to try to $\cdots$ $\cdots$ We 'll *offer* $\rightarrow$ give [give, provide, offer] you help in $\cdots$

Table 3.5: Examples of correct and incorrect choices using Google 5-grams in Phase 2. For these examples, Google 5-grams, 4-grams, 3-grams and 2-grams fail to generate any suggestion.

of the 14504 cases either generate a correct or an incorrect suggestion. It also means that in the next 5-4-gram combination, we only process the remaining of the 16612 cases[5]. Figure 3.3 validates the intuition behind using a combination of n-grams rather using only n-grams (e.g., 5-grams) by showing that while 5-grams generate suggestions for only 14504 cases, a combination of 5-4-3-2-5′-4′-3′-grams[6] generates the same for 30663 cases out of the total of 31116 cases.

Figure 3.4 breaks down the numbers shown in Figure 3.3 into correct choices and

[5]We use the result of the previous 5-grams in 5-4-gram combination, thus we only use 4-grams in this combination.

[6]Apostrophe (′) is used to denote the n-grams used in Phase 2. x-y-$\cdots$-z-gram means that we use x-grams, y-grams, $\cdots$ and z-grams.

CORRECT CHOICE (in Second Phase):
··· military budget also reflects *resolve* → resolve [settle, resolve] to rebuild the ···
··· Klepper suggests that VNRs *provide* → provide [give, provide, offer] major opportunities for ···
INCORRECT CHOICE (in Second Phase):
··· changed by his 1985 *error* → oversight [error, mistake, oversight] hearings on allegedly ···
··· do with any dewy *material* → stuff [material, stuff, substance] about hope springing ···

Table 3.6: Examples of correct and incorrect choices using Google 4-grams in Phase 2. For these examples, Google 5-grams, 4-grams, 3-grams, 2-grams and 5-grams (in Phase 2) fail to generate any suggestion.

CORRECT CHOICE (in Second Phase):
··· company regards SAS 's *commitment* → commitment [responsibility, burden, obligation, commitment] " as firm ···
··· first recommended U.S. " *material* → material [material, stuff, substance] aid " to ···
INCORRECT CHOICE (in Second Phase):
··· But the patrolmen 's *duty* → job [job, task, duty] and their basic ···
··· THE *responsibility* → BURDEN [responsibility, burden, obligation, commitment] OF PROOF in ···

Table 3.7: Examples of correct and incorrect choices using Google 3-grams in Phase 2. For these examples, Google 5-grams, 4-grams, 3-grams, 2-grams, 5-grams (in Phase 2) and 4-grams (in Phase 2) fail to generate any suggestion.

incorrect choices. For example, using only 5-grams, we get correct suggestions for 11492 cases and incorrect suggestions for 3012 cases, out of 14504 cases. Figure 3.4 also shows that a combination of 5-4-3-2-5′-4′-3′-grams generates correct suggestions for 22041 cases and incorrect suggestions for 8622 cases, out of 30663 cases.

Figure 3.5 shows the number of cases where a correct choice, an incorrect choice, or *no suggestion* is generated for different combinations of n-grams used. To give an example, using only 5-grams, we get correct suggestions for 11492 cases, incorrect suggestions for

NO CHOICE:
··· two exchanges ' " → commitment [responsibility, burden, obligation, commitment] to making serious ···
··· He said ECD 's → material [material, stuff, substance] is a multi-component ···
··· Safe Rides , teen-agers → give [give, provide, offer] rides to other ···
··· guilty plea does n't → resolve [settle, resolve] the issue for ···
··· sees Mr. Haig 's → tough [difficult, hard, tough] line toward Cuba ···
··· The 1980 Intelligence → Oversight [error, mistake, oversight] Act requires that ···
··· thought Mr. Cassoni 's → job [job, task, duty] will be made ···

Table 3.8: Examples of sentences where *no suggestion* is returned using Google n-grams. That is, for these examples, Google 5-grams, 4-grams, 3-grams, 2-grams, 5-grams (in Phase 2), 4-grams (in Phase 2) and 3-grams (in Phase 2) fail to generate any suggestion.

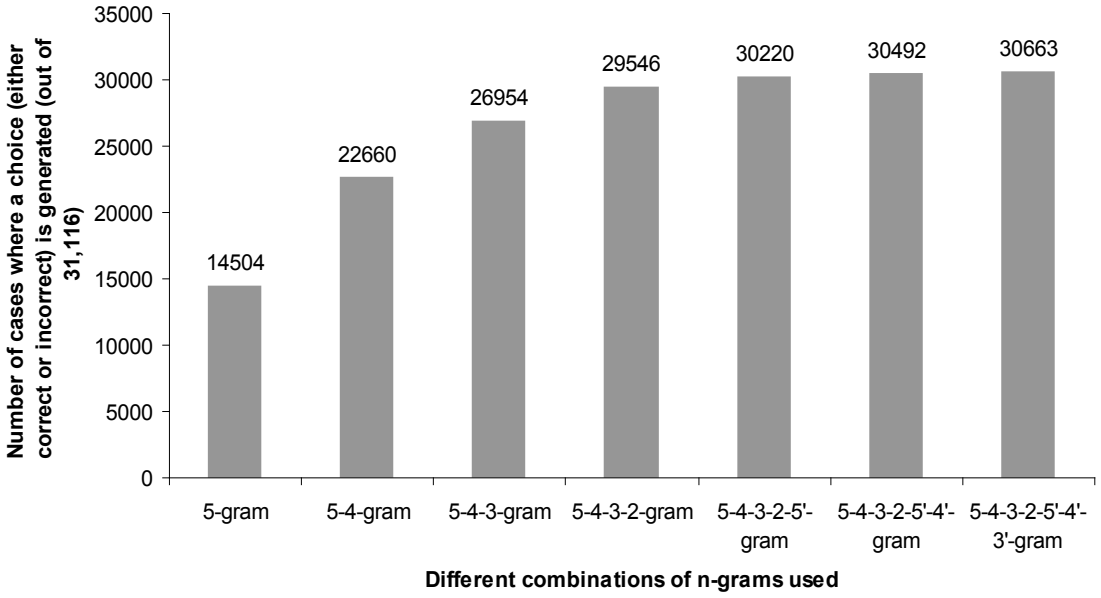

Figure 3.3: Number of cases where a choice (either correct or incorrect) is returned for different combinations of n-grams used.

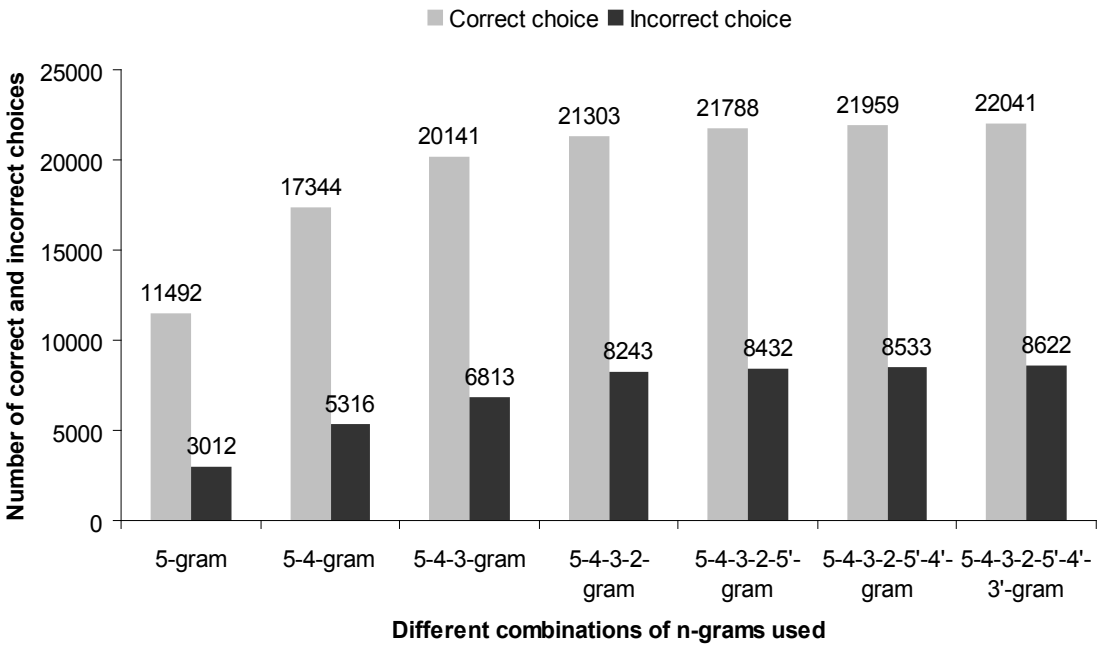

Figure 3.4: Number of correct and incorrect choices returned for different combinations of n-grams used.

3012 cases, and *no suggestion* for 16612 cases; whereas using a combination of 5-4-3-2-5′-4′-3′-grams, we get correct suggestions for 22041 cases, incorrect suggestions for 8622 cases, and *no suggestion* for 453 cases. Figure 3.6 shows the numbers shown in Figure 3.5 into percentage points. For example, using only 5-grams, we get correct suggestions for 36.9% cases, incorrect suggestions for 9.7% cases, and *no suggestion* for 53.4% cases; whereas using a combination of 5-4-3-2-5′-4′-3′-grams, we get correct suggestions for 70.8% cases, incorrect suggestions for 27.7% cases, and *no suggestion* for 1.5% cases.

The performance among different combinations of n-grams is measured using *Precision* (P), *Recall* (R) and *F-measure* (F_1). Figure 3.7 shows *precision*, *recall* and *F-measure* for different combinations of n-grams used. We get highest *precision* (0.792) when using only 5-grams, which is obvious because 5-grams use the maximum possible context (4 words) and as a result the chance of getting the highest ratio between the

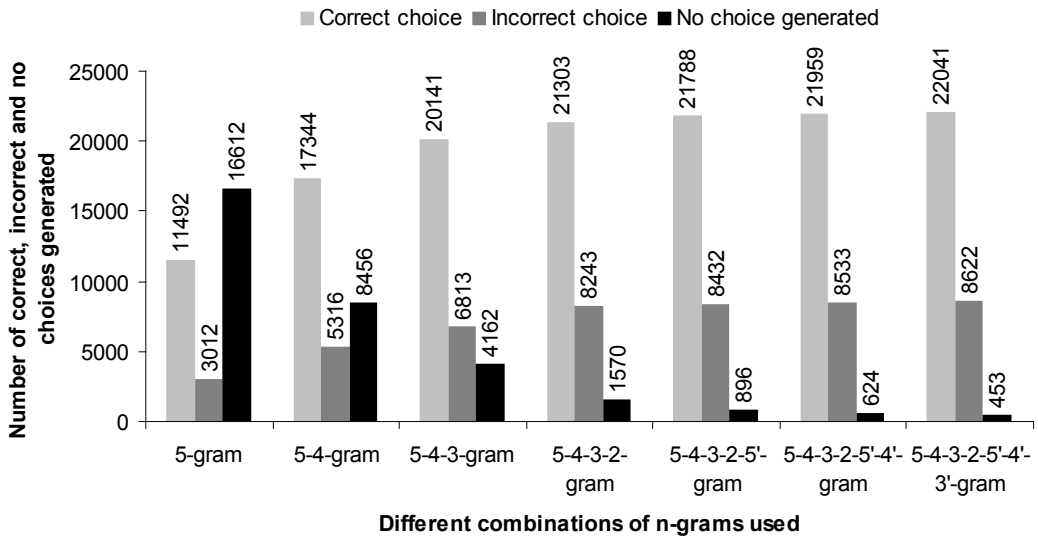

Figure 3.5: Number of correct choices, incorrect choices and *no suggestion* returned for different combinations of n-grams used.

number of correct suggestions returned and the number of suggestions returned increases. But the *recall* at this level is very poor (only 0.369). Figure 3.7 demonstrates how *recall* gets better using different combinations of n-grams while keeping *precision* as high as possible. Using a combination of 5-4-3-2-grams, we get a significant improvement of *recall*, but after that (i.e., a combination of 5-4-3-2-grams to a combination of 5-4-3-2-5′-4′-3′-grams), we get only a 0.023 *recall* increase. As a result, the question of whether it makes any sense to try these later combinations arises. We try to answer this issue in Figure 3.8, which shows the number of cases processed[7] for different combinations of n-grams used. Figure 3.8 shows that we need to process only 3090 more cases when we move from a combination of 5-4-3-2-grams to a combination of 5-4-3-2-5′-4′-3′-grams, and

[7]When we use only 5-grams, we process all 31116 cases (where we do not get a suggestion for 16612 cases) and then when we use 4-grams along with 5-grams for 5-4-grams combination, we process these unsolved 16612 cases again, thus totalling the number of cases processed to 47728 for 5-4-grams combination.

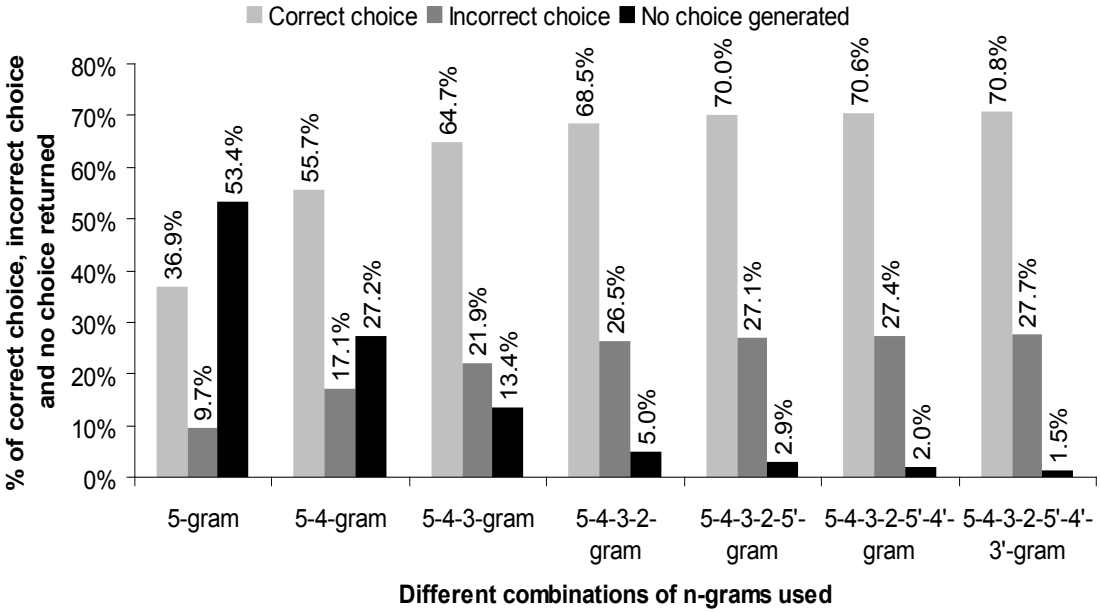

Figure 3.6: Percentage of correct choices, incorrect choices and *no suggestion* returned for different combinations of n-grams used.

we get 738 correct suggestions, 379 incorrect suggestions and 453 *no suggestion*. Thus, it is worth taking into account these later combinations.

Table 3.9 presents the comparative results for the seven groups of near-synonyms. The second last row averages the accuracies for all the test sentences, i.e., these are calculated as the number of correct choices returned over total number of sentences (i.e., 31116). The last row averages the accuracies for all the groups averages, i.e., these are calculated as the sum of the accuracies (in percentage) of all the seven groups over the number of groups (i.e., 7). The second column shows how many test sentences we collected for each near-synonym group. The third column is for the baseline algorithm that always chooses the most frequent near-synonym. The fourth column presents the results reported in (Edmonds, 1997). The fifth column presents the result of Inkpen's (2007) supervised method when using boosting (decision stumps) as machine learning method

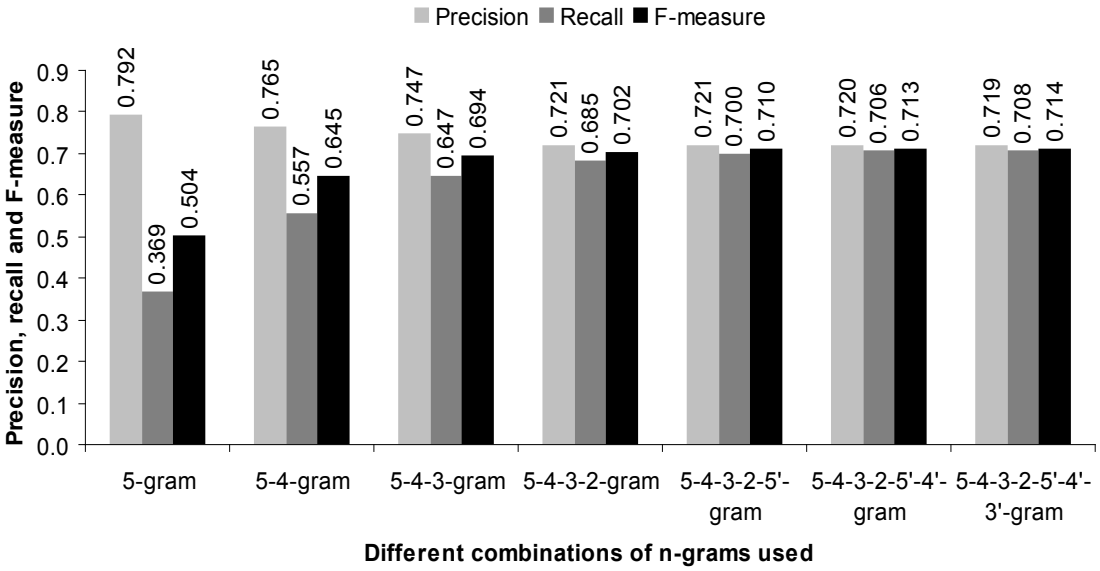

Figure 3.7: Precision, recall and F-measure for different combinations of n-grams used.

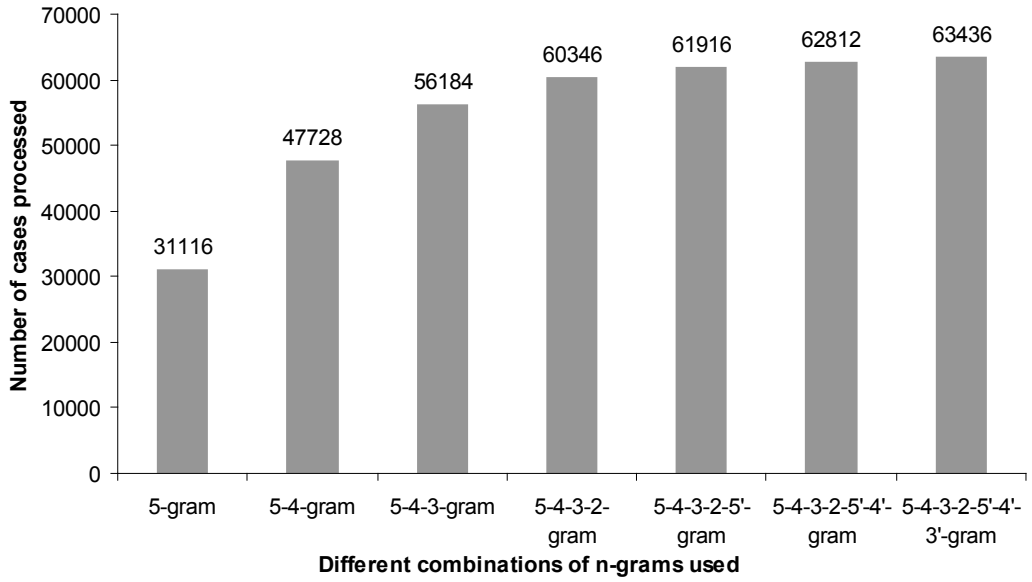

Figure 3.8: Number of cases processed for different combinations of n-grams used.

Test Set	Number of Cases	Accuracy					
		Base-Line	Edmonds's Method	Inkpen's Method (Superv.)	Inkpen's Method (Unsup.)	5-gram Model (Unsup.)	Two-Phase Method (Unsup.)
difficult, hard, tough	6,630	41.7%	47.9%	57.3%	59.1%	63.2 %	**69.4**%
error, mistake, oversight	1,052	30.9%	48.9%	70.8%	61.5%	78.7 %	**83.3**%
job, task, duty	5,506	70.2%	68.9%	**86.7**%	73.3%	78.2%	85.5%
responsibility, burden, obligation, commitment	3,115	38.0%	45.3%	66.7%	66.0%	72.2%	**77.9**%
material, stuff, substance	1,715	59.5%	64.6%	71.0%	72.2%	70.4%	**77.3**%
give, provide, offer	11,504	36.7%	48.6%	56.1%	52.7%	55.8%	**60.1**%
settle, resolve	1,594	37.0%	65.9%	75.8%	**76.9**%	70.8%	74.5%
ALL (average over all sentences)	31,116	44.9%	53.5%	65.2%	61.7%	65.3%	**70.8**%
ALL (average from group averages)	31,116	44.8%	55.7%	69.2%	66.0%	69.9%	**75.4**%

Table 3.9: Comparison among the two proposed methods, a baseline algorithm, Edmonds's method, and Inkpen's unsupervised and supervised method.

and PMI+500 words as features. The sixth column presents the result of Inkpen's (2007) unsupervised method when using word counts in PMI, and the last column is for our proposed unsupervised method using the Google n-grams. We show in bold the best accuracy figure for each data set. We notice that the automatic choice is more difficult for some near-synonym groups than for the others.

Our method performs significantly better than the baseline algorithm, Edmond's method, and Inkpen's unsupervised and even supervised methods. For all the results presented in this chapter, statistical significance tests were done using the paired t-test, as described in (Manning and Schütze, 1999), page 209. Error analysis reveals that incorrect choices happen more often when the context is weak, that is, very short sentences or sentences with very few content words.

On average, our method performs 31 percentage points better than the baseline algorithm, 20 percentage points better than Edmonds's method, 9 percentage points better than Inkpen's unsupervised method, and 6 percentage points better than Inkpen's su-

Test set	J1-J2 Agreement	J1 Accuracy	J2 Accuracy	Inkpen's Accuracy	5-gram Accuracy	Two-Phase Accuracy
difficult, hard, tough	72%	70%	76%	53%	62%	70%
error, mistake, oversight	82%	84%	84%	68%	70%	78%
job, task, duty	86%	92%	92%	78%	80%	90%
responsibility, burden, obligation, commitment	76%	82%	76%	66%	76%	84%
material, stuff, substance	76%	82%	74%	64%	56%	64%
give, provide, offer	78%	68%	70%	52%	52%	58%
settle, resolve	80%	80%	90%	77%	66%	74%
All (average)[8]	78.5%	79.7%	80.2%	65.4%	66%	74%

Table 3.10: Experiments with two human judges on a random subset of the experimental data set, and the results of the methods on the same subset.

pervised method. An important advantage of our method is that it works on any group of near-synonyms without training, whereas Edmonds's method requires a lexical co-occurrence network to be built in advance for each group of near-synonyms and Inkpen's supervised method requires training for each near-synonym group. 453 cases among 31116, where our method failed to provide any suggestion are due to the appearances of some very uncommon proper names or nouns, contractions (e.g., n't), hyphens between two words (e.g., teen-agers), single and double inverted commas in the n-grams, and so on. Preprocessing to tackle this issues would improve the results.

3.5.2 Experiment with human judges

Inkpen (2007) asked two human judges, native speakers of English, to guess the missing word in a random sample of the experimental data set (50 sentences for each of the 7 groups of near-synonyms, 350 sentences in total). The results in Table 3.10 show that the agreement between the two judges is high (78.5%), but not perfect. This means the

[8]Here, each of the seven groups has equal number of sentences, which is 50. Thus, the average from all 350 sentences and the average from group averages are the same.

task is difficult even if some wrong senses in the automatically-produced test data might have made the task easier in a few cases.

The human judges were allowed to choose more than one correct answer when they were convinced that more than one near-synonym fits well in the context. They used this option sparingly, only in 5% of the 350 sentences. In future work, we plan to allow the system to make more than one choice when appropriate (e.g., when the second choice has a very close score to the first choice).

Taking the accuracy achieved by the human judges as an upper limit, Inkpen (2007) concluded that the automatic method has room for improvement of approximately 10-15 percentage points. Our proposed method achieves approximately 10 percentage points improvement.

Figures 3.9, 3.10, 3.11, 3.12, 3.13, and 3.14 show the results of our method on the same 350 sentences used in the experiment with human judges.

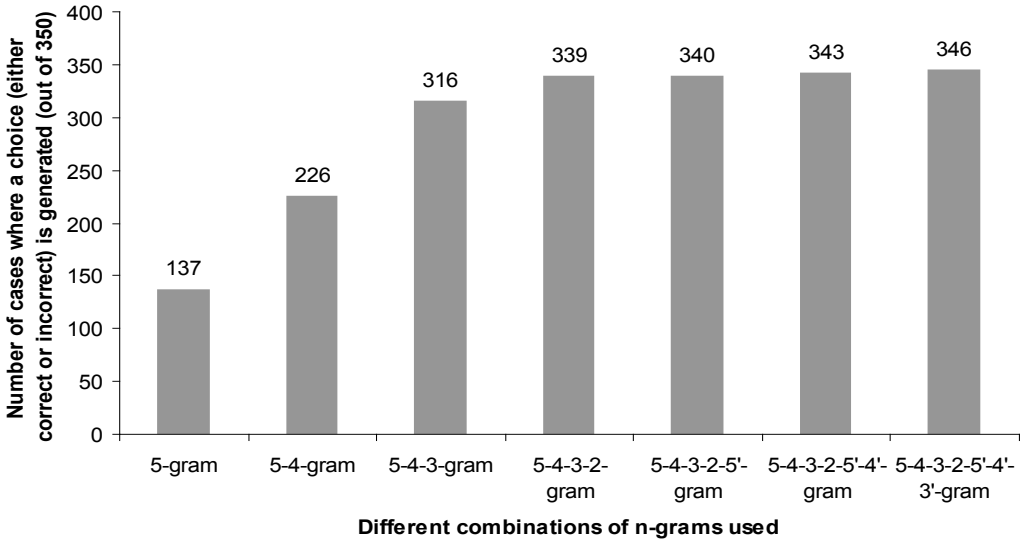

Figure 3.9: Number of cases where a choice (either correct or incorrect) is returned for different combinations of n-grams used, for the 350 sentences used in the experiment with human judges.

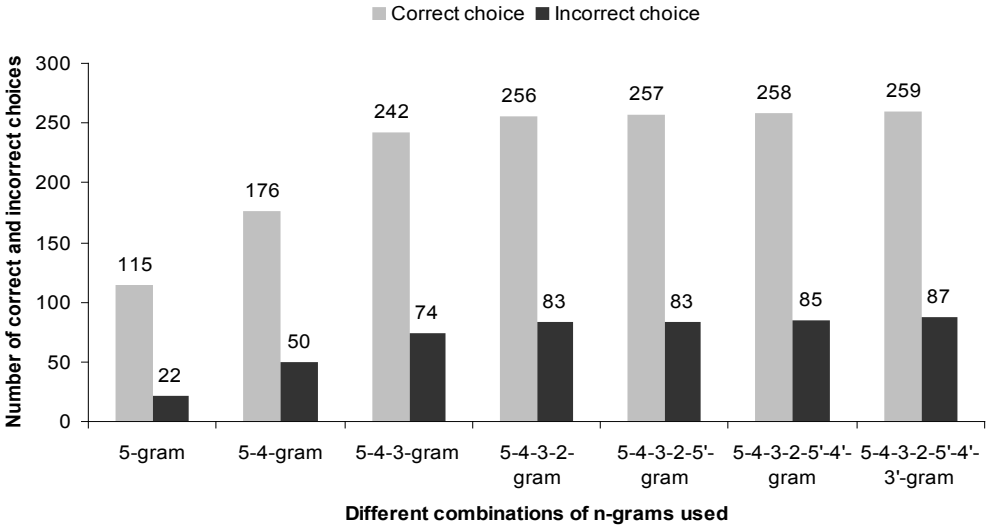

Figure 3.10: Number of correct and incorrect choices returned for different combinations of n-grams used, for the 350 sentences.

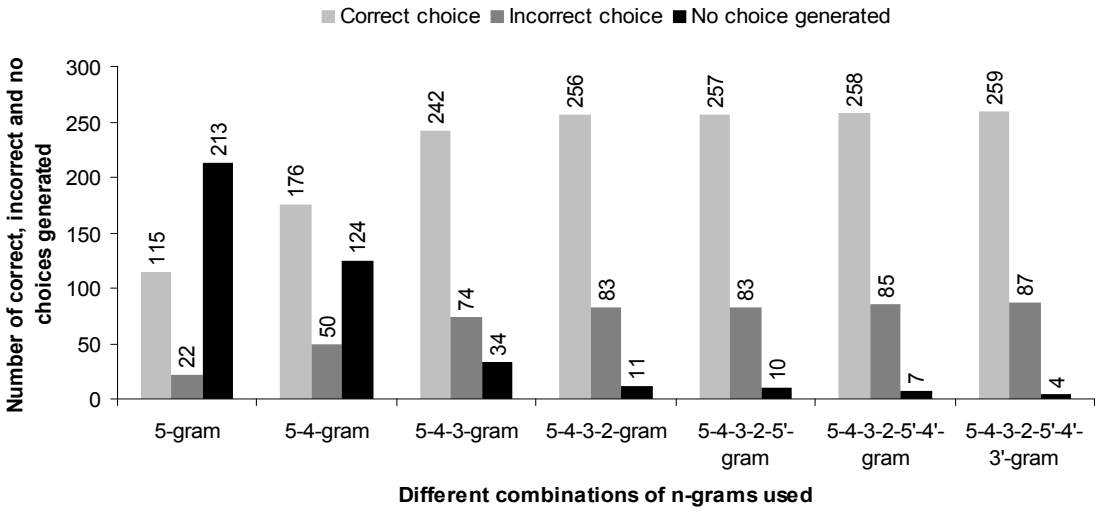

Figure 3.11: Number of correct choices, incorrect choices and *no suggestion* returned for different combinations of n-grams used, for the 350 sentences.

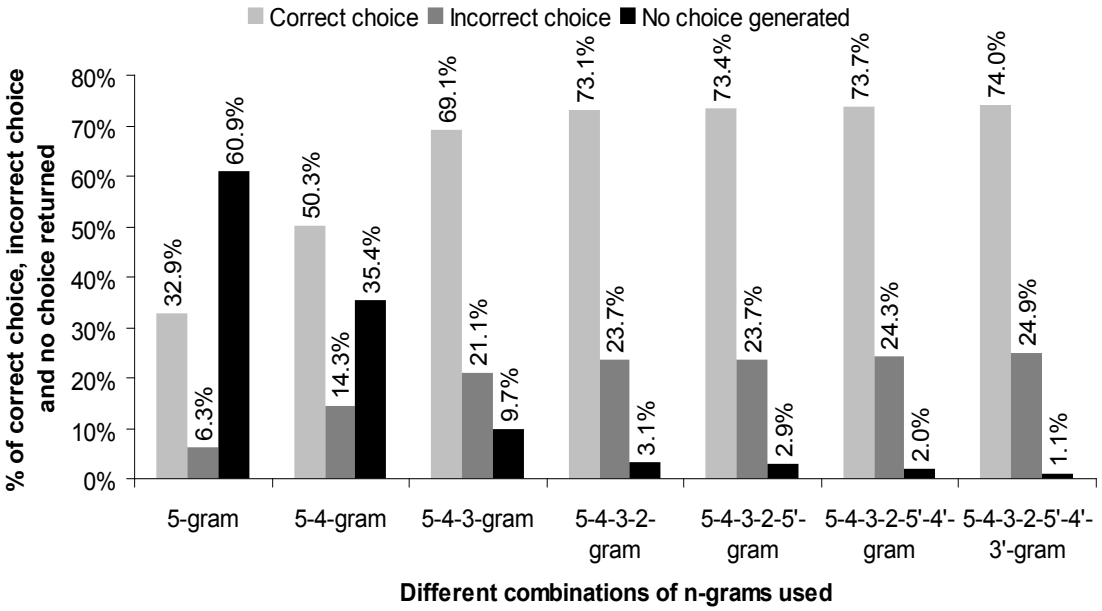

Figure 3.12: Percentage of correct choices, incorrect choices and *no suggestion* returned for different combinations of n-grams used, for the 350 sentences.

3.5.3 Comparison between 5-gram language model and two-phase method

To compare between 5-gram language model and two-phase method, we will use the same notation used in Section 3.3. Let us consider a sentence S composed of the tokens $w_1 \cdots w_5$, where the gap position or the token of interest is w_5, without loss of generality we can express $P(S)$ using equation 3.1 as

$$\begin{aligned} P(S) &= \prod_{i=1}^{5} P(w_i|w_1 \cdots w_{i-1}) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)P(w_4|w_1w_2w_3)P(w_5|w_1w_2w_3w_4) \end{aligned} \tag{3.12}$$

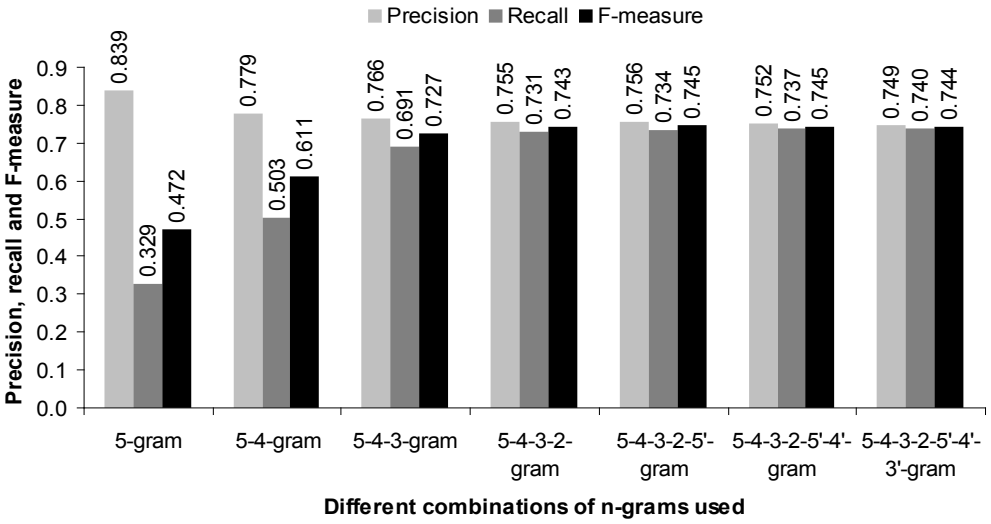

Figure 3.13: Precision, recall and F-measure for different combinations of n-grams used, for the 350 sentences.

Using equation 3.3 in equation 3.12, we get:

$$P(S) = \frac{C(w_1)}{N} \times \frac{C(w_1w_2)}{\sum_{w_2} C(w_1w_2)} \times \frac{C(w_1w_2w_3)}{\sum_{w_3} C(w_1w_2w_3)} \times \frac{C(w_1w_2w_3w_4)}{\sum_{w_4} C(w_1w_2w_3w_4)} \times \frac{C(w_1w_2w_3w_4w_5)}{\sum_{w_5} C(w_1w_2w_3w_4w_5)} \tag{3.13}$$

where N is the number of unigrams in the n-gram data set.

In two-phase method, if $C(w_1w_2w_3w_4w_5)$ exists in the n-gram data set, then the desired problem can be handled using the 5-gram data set. In 5-gram language model, if $C(w_1w_2w_3w_4w_5)$ exists in the n-gram data set, then the desired problem can be handled as $C(w_1w_2w_3w_4)$, $C(w_1w_2w_3)$, $C(w_1w_2)$, and $C(w_1)$ also exist. If w_1 is not a very frequent token in the n-gram data set then two-phase method can handle the desired problem using the 4-gram data set if $C(w_2w_3w_4w_5)$ exists in the data set. But, this is not the case for the 5-gram language model as all the product terms in equation 3.13 are related with the

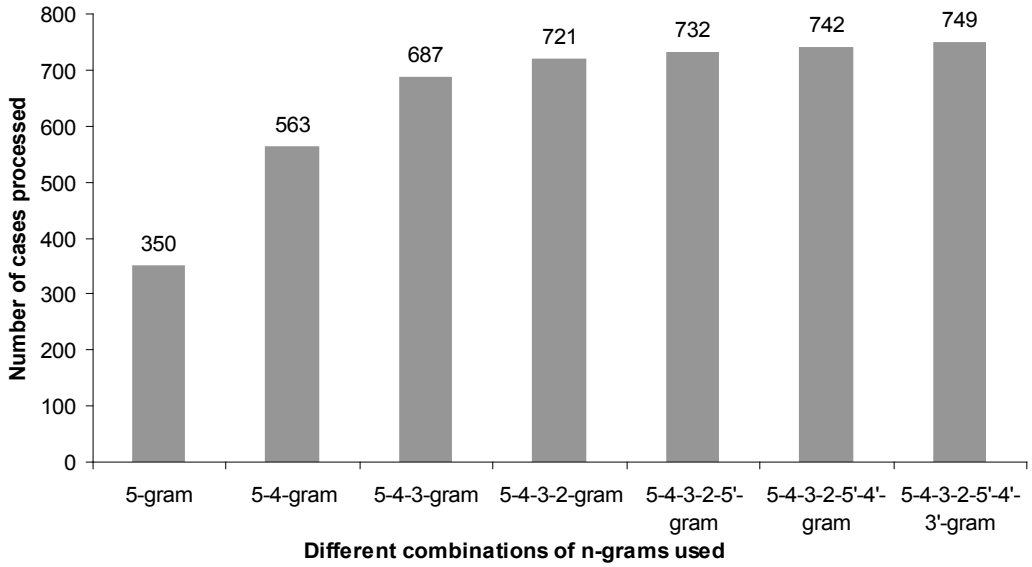

Figure 3.14: Number of cases processed for different combinations of n-grams used, for the 350 sentences.

not-very-frequent token w_1. Now, if w_2 is also a not-very-frequent token then again two-phase method can handle the desired problem using the 3-gram data set if $C(w_3w_4w_5)$ exists in the data set. Similarly, two-phase method can successfully be applied if $C(w_4w_5)$ exists in a bigram data set even if w_1, w_2, and w_3 are all not-very-frequent tokens. In general, for all these cases, 5-gram language model or any n-gram language model is not applicable without smoothing the model with some parameters $\alpha_n > 0$ for $n = 2 \cdots 5$ that needs to be optimized on a corpus.

If w_4 is a not-very-frequent token, then the second phase of the two-phase method is easily applicable using the 5-gram data set if $C(w_1w_2w_3 - w_5)$ exists, where '-' can be substituted with any available word in that position. The idea of using the second phase, where the not-very-frequent token is substituted with any available word in that position, actually restores the context value by at least 80% for the 5-gram data set; considering that each word in the 5-gram data set while used collectively contains equal

context value. Any n-gram language model, in such cases, can not restore almost no context value as it can only use unigram data set.

Again, while classical 5-gram language model uses 5-gram, 4-gram, tri-gram, bigram, and unigram data sets, two-phase approach uses 4-gram data set or the lower n-gram data sets only if the 5-gram data set does not have the required 5-gram data. Thus, the question of how two-phase approach does not lose context value with respect to 5-gram language model needs to be addressed. When a specific 5-gram (e.g., $w_1w_2w_3w_4w_5$) exists in the 5-gram data set, it means that the 4-gram (i.e., $w_1w_2w_3w_4$), derived by removing the fifth word from the 5-gram, tri-gram (i.e., $w_1w_2w_3$), bigram (i.e., w_1w_2), and unigram (i.e., w_1) also exist. Thus, a 5-gram contains the maximum context value with respect to the lower n-grams and the context value does not increase by collectively using the lower n-grams.

3.6 Summary

We present an unsupervised statistical method of choosing the best near-synonym in a context. We compare this method with three previous methods (Edmonds's method and two of Inkpen's method) and a standard 5-gram language model, and show that the performance improved considerably. It is interesting that our unsupervised method performs better than a supervised learning method.

Future work includes an intelligent thesaurus and a natural language generation system that has knowledge of nuances of meaning of near-synonyms. We plan to include a near-synonym sense disambiguation module to ensure that the thesaurus does not offer alternatives for wrong senses of words. We also plan to test the SemEval 2007 data, with a new method for the first step.

Chapter 4

Unsupervised Approaches to Correcting Preposition Errors

In this chapter, the same unsupervised statistical method discussed in Chapter 3 is used for automatic correction of preposition errors using the Google Web 1T n-gram data set and compared to the state-of-the-art. Figure 4.1 shows the flow diagram of correcting preposition errors, where the task is to choose the best preposition from a set of prepositions that could be used in a particular context. It is also shown that the

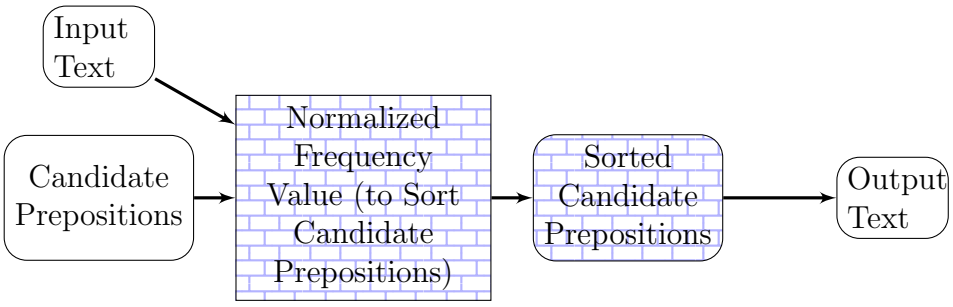

Figure 4.1: Flow diagram of correcting preposition errors.

use of the Web 1T n-gram data set in a back-off fashion increases the performance of the method. The proposed method can be applied to English language texts, including non-native (L2) ones in which preposition errors are known to be numerous. The method

can be applied to other languages for which Google n-grams are available.

4.1 Introduction

Prepositions are known to be one of the most frequent sources of errors for L2 English speakers. Bitchener et al. (2005) found that preposition errors accounted for 29% of all the errors made by intermediate to advanced English as a Second Language (ESL) students. As a result, it seems desirable to focus on this problematic and very common parts of speech, in developing a system for automatic error correction in English writing.

Prepositions are challenging for learners because they can appear to have an idiosyncratic behavior which does not follow any predictable pattern even across nearly identical contexts. That is, the choice of a preposition for a given context also depends upon the intention of the writer. For example, "they sat near the beach", "at the beach", "on the beach", "by the beach" are all grammatically correct. Prepositions are so difficult to master because they perform so many complex roles. In English, prepositions appear in adjuncts, they mark the arguments of predicates, and they combine with other parts of speech to express new meanings. The preposition error correction method that we propose here uses the Web 1T n-gram data set (Brants and Franz, 2006).

This chapter is organized as follow: Section 4.2 presents an overview of the related work. Our proposed method is described in Section 4.3. Evaluation and experimental results are discussed in Section 4.4. We conclude in Section 4.5.

4.2 Related Research

To the best of our knowledge, almost all of the methods for correcting preposition error are based on supervised approaches. A method for correcting preposition errors for French as a second language is presented in (Matthieu Hermet and Szpakowicz, 2008).

It is unsupervised and it uses counts collected from the Web in a simple way, in order to rank the candidates. Japkowicz and Wiebe (1991) describe a system of translation of locative prepositions between English and French by introducing knowledge representations of conceptualizations of objects, and a method for translating prepositions based on these conceptual representations that can also be used to detect conceptual errors and conceptual ambiguities. Li et al. (2005) present the task of machine translation of prepositions using a semantic framework for the interpretation of the surrounding elements of a preposition in the source language. This framework reduces the set of possible prepositions in the target language. Eeg-olofsson and Knutsson (2003) used 31 handcrafted matching rules to detect extraneous, omitted, and incorrect prepositions in Swedish text written by native speakers of English, Arabic, and Japanese. The rules, which were based on the kinds of errors that were found in a training set of text produced by non-native Swedish writers, targeted spelling errors involving prepositions and some particularly problematic Swedish verbs. In a test of the system, 11 of 40 preposition errors were correctly detected.

Izumia et al. (2004) train a maximum entropy classifier to recognize various errors using contextual features. Their results for different error types are: for omission — precision 75.7%, recall 45.67%; for replacement — precision 31.17%, recall 8%; but there is no break-down of results by individual parts of speech. Therefore we do not know what were the results for prepositions only. Chodorow et al. (2007) present an approach to preposition error detection which also uses a model based on a maximum entropy classifier trained on a set of contextual features, together with a rule-based filter. They report 80% precision and 30% recall. Gamon et al. (2008) use a complex system including a decision tree and a language model for preposition errors.

Felice and Pulman (2008) propose a classifier-based supervised approach to correct preposition error in native (L1) English and non-native (L2) English that uses a corpus

of grammatically correct English to train a maximum entropy classifier on examples of correct usage. The L1 source they use is the British National Corpus (BNC). They represent training and testing items as vectors of values for linguistically-motivated contextual features. Their feature vectors include 13 feature categories for prepositions. We will compare our results to their results, on the same test data.

Bergsma et al. (2009) present a unified view of web-scale approaches to lexical disambiguation where they use the counts of 5-grams, 4-grams, trigrams and bigrams in a supervised method on the task of preposition selection. They also use an unsupervised version of their supervised method where they produce a score for each candidate by summing the (unweighted) log-counts of all context patterns (i.e., 5-grams, 4-grams, trigrams and bigrams) using that candidate, and the candidate with the highest score is taken as the solution. While they use the counts of all the n-grams ($n \in \{5, 4, 3, 2\}$), we use the counts of any subsequent (n-1)-grams only if we do not get any suggestion using the counts of n-grams. As a result, our method is computationally more efficient. Their supervised method obtained 75.4% accuracy and unsupervised method obtained 73.7% accuracy. We do not directly compare our results to their results because of the unavailability of their test data set.

4.3 Proposed Method

Our task is to find the best preposition from a set of candidates that could fill in the gap in an input text, using the Google Web 1T data set. The gap is where a preposition was in the original text. In this case, we know what is the expected solution, the original preposition in this L1 text. First, we use the Google 5-gram data set to find the best choice from a set of candidate prepositions. If the 5-gram data set fails to generate a choice, then we move to the 4-gram data set, the 3-gram data set, or the 2-gram data set, if the preceding data set fails to generate at least one choice. Let us

consider an input text T which after tokenization has p ($2 \leq p \leq 9$) words, i.e., $T = \{\ldots w_{i-4}\, w_{i-3}\, w_{i-2}\, w_{i-1}\, \boxed{w_i}\, w_{i+1}\, w_{i+2}\, w_{i+3}\, w_{i+4}\, \ldots\}$, where $\boxed{w_i}$ (in position i) indicates the gap and w_i in $\boxed{w_i}$ denotes a set of m prepositions (i.e., $w_i = \{s_1, s_2, \cdots, s_j, \cdots, s_m\}$). We take into account at most four words before the gap and at most four words after the gap. Our task is to choose the $s_j \in w_i$ that best matches with the context. In other words, the position of i is the gap that needs to be filled with the best suited member from the set, w_i.

We use the same procedures discussed in Section 3.4.1 (page 41) and Section 3.4.2 (page 42) to categorize different n-grams based on the gap position and to find the *normalized frequency value* respectively. Then, we use the procedures discussed in Section 3.4.3 (page 43) and Section 3.4.4 (page 44) to find the best choice preposition using the n-gram ($n \in \{5, 4, 3, 2\}$) data set.

4.4 Evaluation and Experimental Results

We restrict our candidate preposition set to the nine most frequent prepositions in the BNC: ***of***, ***to***, ***in***, ***for***, ***on***, ***with***, ***at***, ***by***, and ***from***, same as Felice and Pulman (2008) to ensure the conformity, for direct comparison. Felice and Pulman (2008) tested their model on a section of the British National Corpus (BNC) with test set size 536,193. Felice and Pulman (2008) mentioned that their model's performance compared favorably to the best results in the literature, although direct comparisons were hard to draw since different groups trained and tested on different preposition sets and on different types of data (British vs. American English, BNC vs. news reports, and so on). To directly compare with Felice and Pulman (2008), we also use the same test set size (536,193 cases) from the BNC. Our best result to date is 75.64% accuracy. Figure 4.2 relates our results to others reported in the literature on comparable task. The baseline refers to always choosing the most frequent option, namely ***of***. It should be noted that Gamon

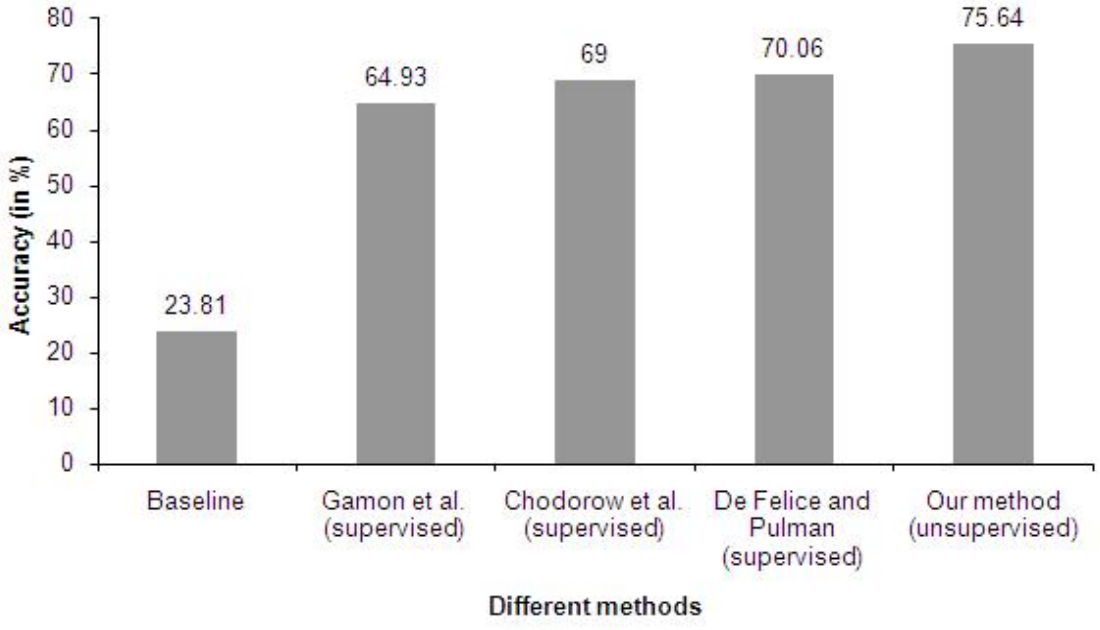

Figure 4.2: Performance of different methods on L1 prepositions.

et al. (2008) report more than one figure in their results, as there are two components to their model: one determining whether a preposition is needed, and the other deciding what the preposition should be. The figure reported here refers to the later task, as it is the most similar to the one we are evaluating. Chodorow et al. (2007) also discuss some modifications to their model which can increase accuracy; the result noted here is the one more directly comparable to our own approach.

4.4.1 Further discussion and analysis

To assess the method's performance on the L1 data, it is important to consider factors such as performance on individual prepositions, the relationship between test set size and accuracy, and the kinds of errors made by the model.

Table 4.1 shows the method's performance on individual prepositions together with the size of their test sets. A detailed picture of the method's errors can be observed by

Table 4.1: L1 results — individual prepositions.

Prepositions	Test set size	Accuracy
of	135,161	94.45%
to	111,834	86.12%
in	97,558	75.73%
for	42,428	56.81%
with	34,953	57.37%
on	32,628	58.54%
by	31,278	54.44%
at	25,652	63.45%
from	24,701	45.24%

Table 4.2: Confusion matrix for L1 data — prepositions.

Target Prep	Confused with								
	of	to	in	for	with	on	by	at	from
of		17.00%	43.67%	14.67%	7.33%	6.33%	5.00%	3.67%	2.33%
to	35.10%		30.92%	11.92%	7.09%	4.19%	5.15%	2.09%	3.54%
in	51.32%	14.04%		12.25%	5.49%	5.81%	5.60%	2.85%	2.64%
for	32.88%	22.10%	25.78%		7.23%	2.73%	4.09%	2.86%	2.32%
with	32.55%	17.79%	31.71%	8.56%		3.69%	2.52%	1.01%	2.18%
on	30.87%	15.71%	29.02%	5.91%	6.84%		4.25%	3.33%	4.07%
by	33.86%	13.86%	28.60%	8.25%	7.54%	3.16%		2.63%	2.11%
at	18.67%	18.40%	32.00%	11.47%	6.40%	8.00%	2.40%		2.67%
from	29.39%	14.97%	27.73%	8.50%	6.28%	3.51%	5.36%	4.25%	

looking at the confusion matrix in Table 4.2, which shows, for each preposition, what the method's incorrect decision was.

Table 4.3 shows some examples of instances where the method's chosen preposition differs from that found in the original text. In most cases, the method's suggestion is also grammatically correct, but the overall meaning of the phrases changes somewhat.

For each case, our method returns either a choice (which is either correct or incorrect) or *no suggestion*. Figure 4.3 shows the number of cases where either a choice (correct or incorrect) or *no suggestion* is generated for different combinations of n-grams used. To give an example, using only 5-grams, each of the 316950 cases either generate a

Table 4.3: Examples of method's errors on preposition L1 task

Method's choice	**Correct phrase**
The connections and friendships **with** Surrealism can also be	friendships **of** Surrealism
He wants to escape **from** the world	escape **to** the world
hand were essential ingredients **to** the success of The	ingredients **in** the success
may not be enough **to** a theoretician.	enough **for** a theoretician
saw the twentieth century **in** their eyes but they	century **with** their eyes
figure begins to work **with** us further, now less	work **on** us
as a sympathetic appraisal **of** a critic who is	appraisal **by** a critic
Indeed, an article **of** this length will frequently	an article **at** this length
they seem to proceed **on** his own mind entirely,	proceed **from** his own

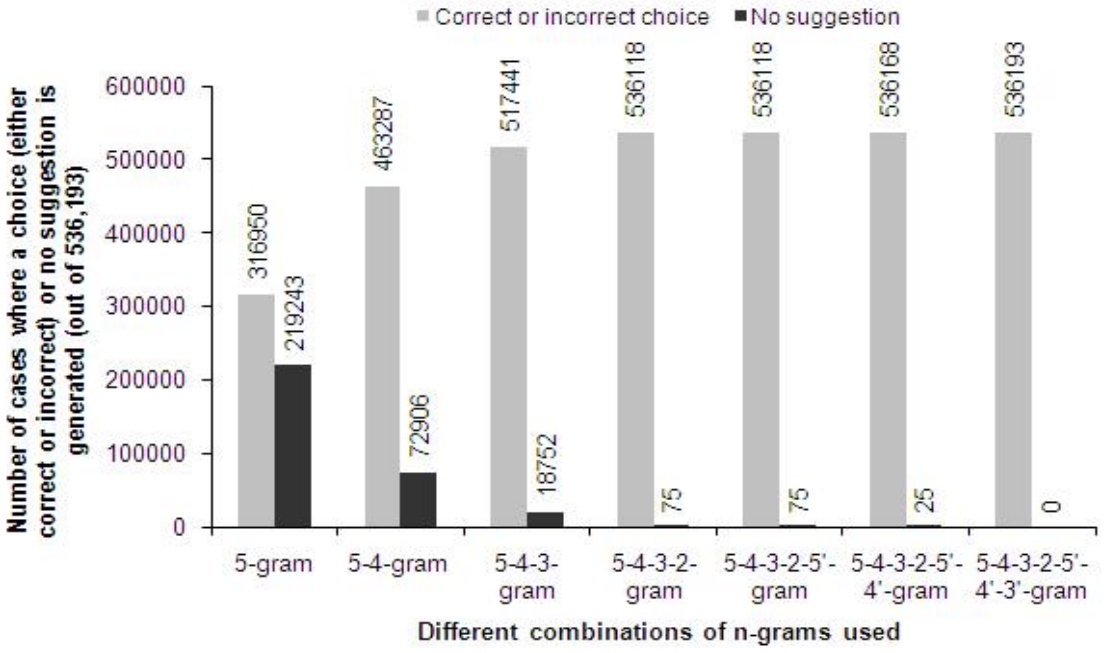

Figure 4.3: Number of cases where a choice (either correct or incorrect) and *no suggestion* is returned for different combinations of n-grams used.

correct or an incorrect suggestion. It also means that in the next 5-4-gram combination, we only process the remaining of the 219243 cases[1]. Figure 4.3 validates the intuition behind using a combination of n-grams rather than using only n-grams (e.g., 5-grams),

[1]We use the result of the previous 5-grams in 5-4-gram combination, thus we only use 4-grams in this combination.

by showing that while 5-grams generate suggestions for only 316950 cases, a combination of 5-4-3-2-5′-4′-3′-grams[2] generates suggestion for all 536193 cases.

Figure 4.4 breaks down the numbers shown in Figure 4.3 into correct choices, incorrect

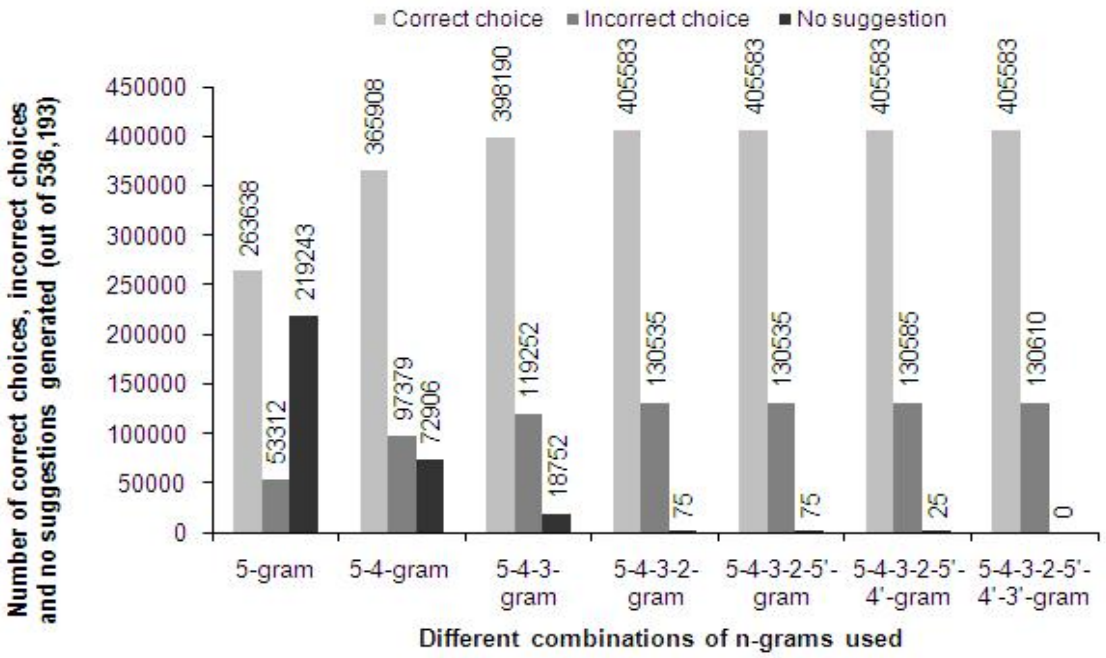

Figure 4.4: Number of correct choices, incorrect choices and *no suggestions* returned for different combinations of n-grams used.

choices and *no suggestion*. To give an example, using only 5-grams, we get correct suggestions for 263638 cases, incorrect suggestions for 53312 cases, and *no suggestion* for 219243 cases; whereas using a combination of 5-4-3-2-5′-4′-3′-grams, we get correct suggestions for 405583 cases, incorrect suggestions for 130610 cases, and *no suggestion* for 0 cases.

The performance among different combinations of n-grams is measured using *Precision* (P) and *Recall* (R). Figure 4.5 shows *precision* and *recall* for different combinations of n-grams used. We get the highest *precision* (83.18%) when using only 5-grams, which

[2]Apostrophe (′) is used to denote the n-grams used in Phase 2. x-y-$\cdots$ z-gram means that we use x-grams, y-grams, $\cdots$ and z-grams.

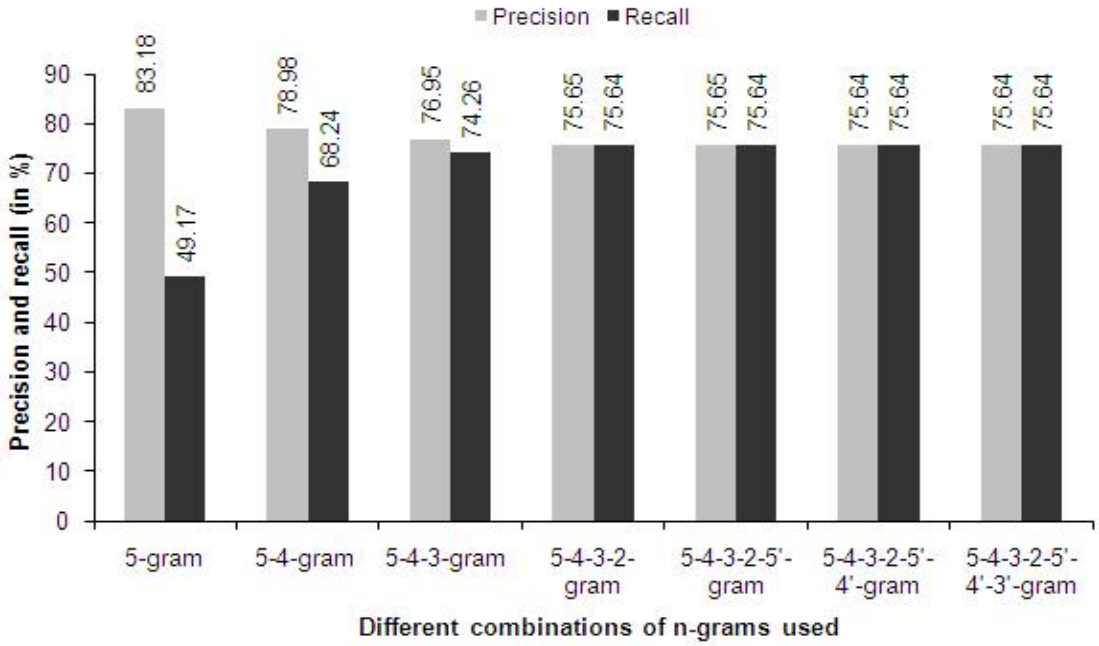

Figure 4.5: Precision and recall for different combinations of n-grams used.

is obvious because 5-grams use the maximum possible context (4 words) and as a result the chance of getting the highest ratio between the number of correct suggestions returned and the number of suggestions returned increases. But the *recall* at this level is very poor (only 49.17%). Figure 4.5 demonstrates how *recall* gets better using different combinations of n-grams while keeping *precision* as high as possible. Using a combination of 5-4-3-2-5′-4′-3′-grams, we achieve equal precision and recall (which is also the accuracy of the method). Thus, the equal precision, recall and accuracy ensure that for each preposition, we get one and only one suggestion. This is not the case for other approaches.

Using a combination of 5-4-3-2-grams, we get a significant improvement of *recall*, but after that (i.e., a combination of 5-4-3-2-grams to a combination of 5-4-3-2-5′-4′-3′-grams), we do not get any improvement. As a result, the question of whether it makes any sense to try these later combinations arises. We try to answer this issue in Figure 4.6,

which shows the number of cases processed[3] for different combinations of n-grams used.

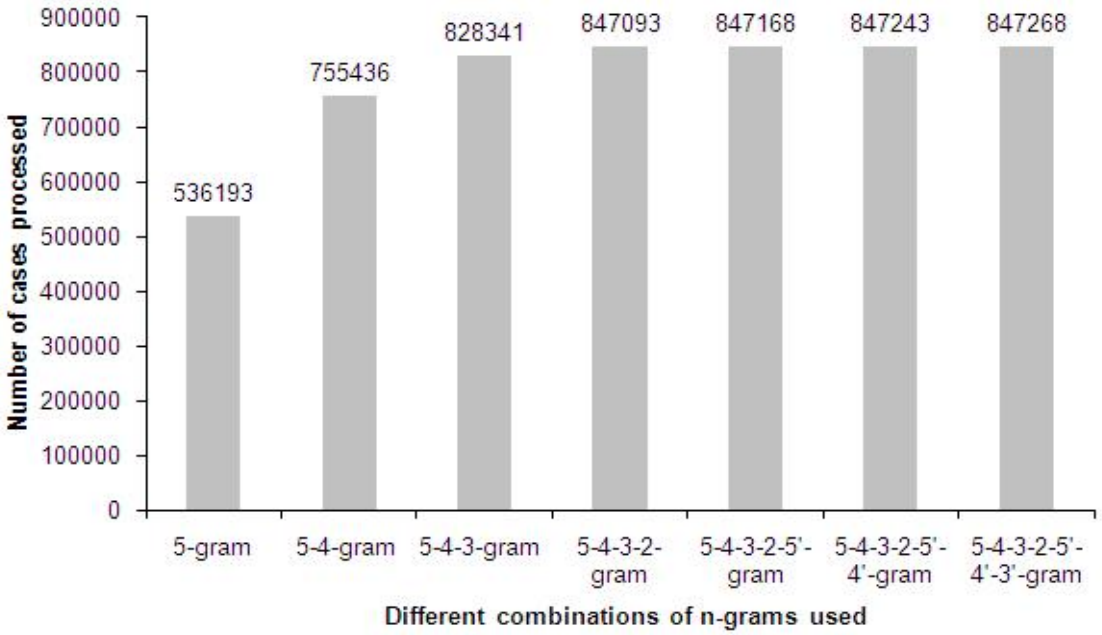

Figure 4.6: Number of cases processed for different combinations of n-grams used.

Figure 4.6 shows that we need to process only 175 more cases when we move from a combination of 5-4-3-2-grams to a combination of 5-4-3-2-5′-4′-3′-grams. Though for this data set we do not get any new correct suggestions, there is always some chance to provide some correct suggestions. Thus, it is worth taking into account these later combinations.

4.5 Summary

We use the unsupervised statistical method presented in Chapter 3 for correcting preposition errors. We compare the result of this method with that of the three previous

[3]When we use only 5-grams, we process all 536193 cases (where we do not get a suggestion for 219243 cases) and then when we use 4-grams along with 5-grams for 5-4-grams combination, we process these unsolved 219243 cases again, thus totalling the number of cases processed to 755436 for 5-4-grams combination.

supervised methods and show that the performance is comparable or even better. It has been estimated that about 750M people use English as a second language, as opposed to 375M native English speakers (Crystal, 1997), while as much as 74% of writing in English is done by non-native speakers (Gamon et al., 2008). Because the Web 1T data set is a representation of both native and non-native English, we can say that our proposed unsupervised method is also equally applicable to L2 English texts. For proprietary reason, we cannot test our method to the L2 data set that Felice and Pulman (2008) and Chodorow et al. (2007) use. Also, at the moment there is no other adequate test set publicly available to test the method on a L2 data set. In future, we plan to test our method on a L2 data set, if one becomes publicly available.

Chapter 5

Real-Word Spelling Errors

In this chapter, we present a method for correcting real-word spelling errors using the Google Web 1T n-gram data set. Figure 5.1 shows the flow diagram of correcting real-word spelling error, where the task is to generate a list of candidate words for the erroneous word and then select the best candidate word from the list, that should be used in a particular context. Here, a normalized and modified version of the Longest Common

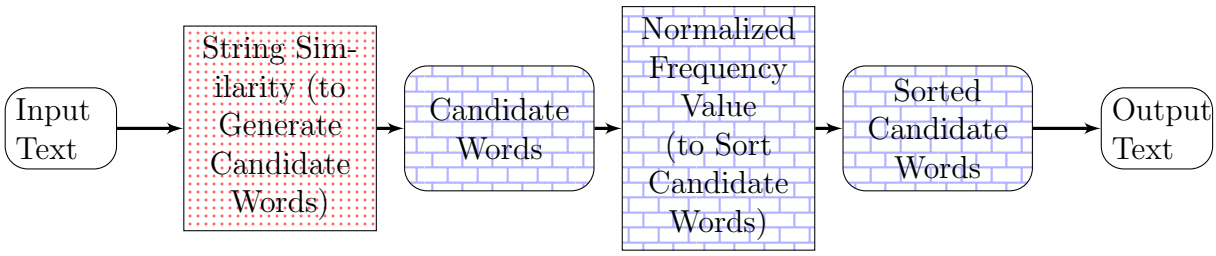

Figure 5.1: Flow diagram of correcting real-word spelling errors.

Subsequence (LCS) string matching algorithm (Hirschberg, 1977) is used to generate a list of candidate words and then normalized frequency value function is used to choose the best candidate word. Our method is focused mainly on how to improve the correction recall (the fraction of errors corrected) while keeping the correction precision (the fraction of suggestions that are correct) as high as possible. Evaluation results on a

standard data set show that our method performs very well.

5.1 Introduction

Real-word spelling errors may be caused by the writer's ignorance of the correct spelling of the intended word or by typing mistakes. Such errors generally go unnoticed by most spellcheckers as they deal with words in isolation, accepting them as correct if they are found in the dictionary, and flagging them as errors if they are not. This approach would be sufficient to correct the non-word error *myss* in "It doesn't know what the *myss* is all about." but not the real-word error *muss* in "It doesn't know what the *muss* is all about." To correct the latter, the spell-checker needs to make use of the surrounding context such as, in this case, to recognise that *fuss* is more likely to occur than *muss* in the context of *all about*. Ironically, errors of this type may even be caused by spelling checkers in the correction of non-word spelling errors when the *auto-correct* feature in some word-processing software sometimes silently change a non-word to the wrong real word (Hirst and Budanitsky, 2005), and sometimes when correcting a flagged error, the user accidentally make a wrong selection from the choices offered (Wilcox-O'Hearn et al., 2008). An extensive review of real-word spelling correction is given in (Pedler, 2007; Hirst and Budanitsky, 2005) and the problem of spelling correction more generally is reviewed in (Kukich, 1992).

In this chapter, we present a method for correcting real-word spelling errors using the Google Web 1T n-gram data set (Brants and Franz, 2006), and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm (details are in section 5.3.1 on page 80). Our intention is to focus on how to improve the correction recall while maintaining the correction precision as high as possible. The reason behind this intention is that if the recall for any method is around 0.5, this means that the method fails to correct around 50 percent of the errors. As a result, we can

not completely rely on these type of methods, for that we need some type of human interventions or suggestions to correct the remaining of the uncorrected errors. Thus, if we have a method that can correct almost 90 percent of the errors, even generating some extra candidates that are incorrect is more helpful to the human.

This chapter is organized as follow: Section 5.2 presents a brief overview of the related work. Our proposed method is described in Section 5.3. Evaluation and experimental results are discussed in Section 5.4. We conclude in Section 5.5.

5.2 Related Research

Work on real-word spelling correction can roughly be classified into three categories: methods based on semantic information or human-made lexical resources, methods based on machine learning, and methods based on probability information. Our proposed method falls into the third category.

5.2.1 Methods Based on Semantic Information

The 'semantic information' approach first proposed by Hirst and St-Onge (1998) and later developed by Hirst and Budanitsky (2005) detected semantic anomalies, but was not restricted to checking words from predefined confusion sets. This approach was based on the observation that the words that a writer intends are generally semantically related to their surrounding words, whereas some types of real-word spelling errors are not.

5.2.2 Methods Based on Machine Learning

Machine learning methods are regarded as lexical disambiguation tasks and confusion sets are used to model the ambiguity between words. Normally, the machine learning and statistical approaches rely on pre-defined confusion sets, which are sets (usually pairs) of

commonly confounded words, such as {*their*, *there*, *they're*} and {*principle*, *principal*}. Golding and Roth (1999), an example of a machine-learning method, combined the Winnow algorithm with weighted-majority voting, using nearby and adjacent words as features. Another example of a machine-learning method is that of Carlson et al. (2001).

5.2.3 Methods Based on Probability Information

Mays et al. (1991) proposed a statistical method using word-trigram probabilities for detecting and correcting real-word errors without requiring predefined confusion sets. In this method, if the trigram-derived probability of an observed sentence is lower than that of any sentence obtained by replacing one of the words with a spelling variation, then they hypothesize that the original is an error and the variation is what the user intended.

Verberne (2002) proposed a trigram-based method for real-word errors without explicitly using probabilities or even localizing the possible error to a specific word. This method simply assumes that any word trigram in the text that is attested in the British National Corpus (Burnard, 2000) is correct, and any unattested trigram is a likely error. When an unattested trigram is observed, the method then tries the spelling variations of all words in the trigram to find attested trigrams to present to the user as possible corrections. The evaluation of this method was carried out on only 7100 words of the Wall Street Journal corpus, with 31 errors introduced (i.e., one error in every approximately 200 words) obtaining a recall of 0.33 for correction, a precision of 0.05 and a F-measure of 0.086.

The work that is closely related to the problem addressed in this chapter is that of Wilcox-O'Hearn et al. (2008), where the authors analyze the advantages and limitations of Mays et al. (1991)'s method, and present a new evaluation of the algorithm, designed so that the results can be compared with those of other methods, and then construct and evaluate some variations of the algorithm that use fixed-length windows.

They consider a variation of the method that optimizes over relatively short, fixed-length windows instead of over a whole sentence (except in the special case when the sentence is smaller than the window), while respecting sentence boundaries as natural breakpoints. To check the spelling of a span of d words requires a window of length d+4 to accommodate all the trigrams that overlap with the words in the span. The smallest possible window is therefore 5 words long, which uses 3 trigrams to optimize only its middle word. They assume that the sentence is bracketed by two *BoS* and two *EoS* markers (to accommodate trigrams involving the first two and last two words of the sentence). The window starts with its left-hand edge at the first *BoS* marker, and the Mays et al. (1991)'s method is run on the words covered by the trigrams that it contains; the window then moves d words to the right and the process repeats until all the words in the sentence have been checked. As Mays et al. (1991)'s algorithm is run separately in each window, potentially changing a word in each, Wilcox-O'Hearn et al. (2008)'s method as a side-effect also permits multiple corrections in a single sentence. Contrary to Wilcox-O'Hearn et al. (2008)'s method, which is supervised and uses a trigram model, our proposed method is unsupervised and uses n-gram (where $n \in \{5, 4, 3, 2\}$) in a back-off fashion.

5.3 Proposed Method

The proposed method first tries to determine some probable candidates and then finds the best one among the candidates. We consider a string similarity function and a frequency value function in our method. The following sections present a detailed description of the frequency value function, followed by the procedure to determine some probable candidates along with the procedure to find the best candidate.

5.3.1 Similarity between Two Strings

We use the *Longest Common Subsequence* (*LCS*) (Hirschberg, 1977) measure with some normalization and small modifications for our string similarity measure. Kondrak (2005) showed that edit distance and the length of the *longest common subsequence* are special cases of n-gram distance and similarity, respectively. Melamed (1999) normalized *LCS* by dividing the length of the *longest common subsequence* by the length of the longer string and called it *Longest Common Subsequence Ratio* (*LCSR*). But *LCSR* does not take into account the length of the shorter string which sometimes has a significant impact on the similarity score.

We use the same three different modified versions of *LCS* that (Islam and Inkpen, 2008; Islam et al., 2008; Islam and Inkpen, 2009e) used, along with another modified version of *LCS* (i.e., Consecutive LCS ending at the last character), and then take a weighted sum of these. We obtained better experimental results (precision and recall) for schema matching on a sample of data using the modified versions than when using the original LCS, or other string similarity measures (Islam et al., 2008). The theoretical reason behind this is as follows:
If the two strings that we are comparing do not have any Consecutive Longest Common Subsequence starting at character 1 or starting at any character n, or ending at the last character, then the standard LCS and our modified version will get the same score. But, if the two strings do have any Consecutive Longest Common Subsequence starting at character 1 or starting at any character n, or ending at the last character, then our modified LCS string matching algorithm will get better normalized score than the standard LCS.

Islam and Inkpen (2008) normalized the *longest common subsequence* so that it takes into account the length of both the shorter and the longer string and called it *Normalized*

Longest Common Subsequence (*NLCS*) which is:

$$v_1 = NLCS(s_1, s_2) = \frac{len(LCS(s_1, s_2))^2}{len(s_1) \times len(s_2)} \tag{5.1}$$

We normalize equation (5.1) in the following way as it gives better similarity value, as well as it is more computationally efficient:

$$v_1 = NLCS(s_1, s_2) = \frac{2 \times len(LCS(s_1, s_2))}{len(s_1) + len(s_2)} \tag{5.2}$$

While in classical *LCS*, the common subsequence needs not be consecutive, in spelling correction, a consecutive common subsequence is important for a high degree of matching. Islam and Inkpen (2008) used *Maximal Consecutive Longest Common Subsequence* starting at character 1, $MCLCS_1$ and *Maximal Consecutive Longest Common Subsequence* starting at any character n, $MCLCS_n$. $MCLCS_1$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching must be from first character (character 1) for both strings. $MCLCS_n$ takes two strings as input and returns the shorter string or maximal consecutive portions of the shorter string that consecutively match with the longer string, where matching may start from any character (character n) for both of the strings. They normalized $MCLCS_1$ and $MCLCS_n$ and called it *Normalized* $MCLCS_1$ ($NMCLCS_1$) and *Normalized* $MCLCS_n$ ($NMCLCS_n$), respectively.

$$v_2 = NMCLCS_1(s_1, s_2) = \frac{len(MCLCS_1(s_1, s_2))^2}{len(s_1) \times len(s_2)} \tag{5.3}$$

$$v_3 = NMCLCS_n(s_1, s_2) = \frac{len(MCLCS_n(s_1, s_2))^2}{len(s_1) \times len(s_2)} \tag{5.4}$$

Similarly, we normalize equation (5.3) and (5.4) in the following way:

$$v_2 = NMCLCS_1(s_1, s_2) = \frac{2 \times len(MCLCS_1(s_1, s_2))}{len(s_1) + len(s_2)} \tag{5.5}$$

$$v_3 = NMCLCS_n(s_1, s_2) = \frac{2 \times len(MCLCS_n(s_1, s_2))}{len(s_1) + len(s_2)} \tag{5.6}$$

Islam and Inkpen (2008) did not consider consecutive common subsequences ending at the last character, though $MCLCS_n$ sometimes, if not always, covers this. We argue that the consecutive common subsequence ending at the last character is as significant as the consecutive common subsequence starting at the first character. So, we introduce the *Maximal Consecutive Longest Common Subsequence* ending at the last character, $MCLCS_z$ (Algorithm 1). Algorithm 1, takes two strings as input and returns the shorter string or the maximal consecutive portions of the shorter string that consecutively matches with the longer string, where matching must end at the last character for both strings. We normalize $MCLCS_z$ and call it *Normalized* $MCLCS_z$ ($NMCLCS_z$).

$$v_4 = NMCLCS_z(s_1, s_2) = \frac{2 \times len(MCLCS_z(s_1, s_2))}{len(s_1) + len(s_2)} \tag{5.7}$$

We take the weighted sum of these individual values v_1, v_2, v_3, and v_4 from equation (5.2), (5.5), (5.6) and (5.7), respectively, to determine the string similarity score, where α_1, α_2, α_3, α_4 are weights and $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. Therefore, the similarity of the two strings, $S_1 \in [0, 1]$ is:

$$S_1(s_1, s_2) = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 \tag{5.8}$$

We heuristically set equal weights for most of our experiments[1]. Theoretically, $v_3 \geq v_2$

[1]We use equal weights in several places in this thesis in order to keep the system unsupervised. If development data would be available, we could adjust the weights.

and $v_3 \geq v_4$.

To give an example, consider $s_1 = albastru$ and $s_2 = alabasteru$, then using equation (5.1), (5.3), (5.4) and (5.7):

$LCS(s_1, s_2) = albastru$

$MCLCS_1(s_1, s_2) = al$

$MCLCS_n(s_1, s_2) = bast$

$MCLCS_z(s_1, s_2) = ru$

$NLCS(s_1, s_2) = 8^2/(8 \times 10) = 0.8$

$NMCLCS_1(s_1, s_2) = 2^2/(8 \times 10) = 0.05$

$NMCLCS_n(s_1, s_2) = 4^2/(8 \times 10) = 0.2$

$NMCLCS_z(s_1, s_2) = 2 \times 2/(8 + 10) = 0.22$

The string similarity, $S_1 = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 = 0.25 \times 0.8 + 0.25 \times 0.05 + 0.25 \times 0.2 + 0.25 \times 0.22 = 0.318$

Again, using equation (5.2), (5.5), (5.6) and (5.7):

$NLCS(s_1, s_2) = 2 \times 8/(8 + 10) = 0.89$

$NMCLCS_1(s_1, s_2) = 2 \times 2/(8 + 10) = 0.22$

$NMCLCS_n(s_1, s_2) = 2 \times 4/(8 + 10) = 0.44$

$NMCLCS_z(s_1, s_2) = 2 \times 2/(8 + 10) = 0.22$

Therefore, the string similarity is, $S_1 = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4 = 0.25 \times 0.89 + 0.25 \times 0.22 + 0.25 \times 0.44 + 0.25 \times 0.22 = 0.443$. The previous two examples show that string similarity using equations (5.2), (5.5), (5.6) and (5.7) gives better similarity value than that of using equations (5.1), (5.3), (5.4) and (5.7).

Algorithm 1: $MCLCS_z$ (Maximal Consecutive LCS ending at the last character)

input : s_1, s_2 /* s_1 and s_2 are input strings where $|s_1| \leq |s_2|$ */
output: *str* /* *str* is the Maximal Consecutive LCS ending at the last character */

```
1  str ←NULL
2  c ← 1
3  while |s1| ≥ c do
4  |  x ← SubStr(s1, −c, 1) /* returns cth character of s1 from the end */
5  |  y ← SubStr(s2, −c, 1) /* returns cth character of s2 from the end */
6  |  if x = y then
7  |  |  str ← SubStr(s1, −c, c)
8  |  else
9  |  |  return str
10 |  end
11 |  increment c
12 end
```

5.3.2 Normalized Frequency Value

We determine the normalized frequency value of each candidate word for a single position with respect to all other candidates for the same position. If we find n replacements of a word w_i which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$, and their frequencies $\{f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in}\}$, where f_{ij} is the frequency of a n-gram (where $n \in \{5, 4, 3, 2\}$ and any candidate word w_{ij} is a member of the n-gram), then we determine the normalized frequency value of any candidate word w_{ij} as the frequency of the n-gram containing w_{ij}, over the maximum frequency among all the candidate words for that position.

$$F(w_{ij}) = \frac{f_{ij}}{\max(f_{i1}, f_{i2}, \cdots, f_{ij}, \cdots, f_{in})} \tag{5.9}$$

5.3.3 Determining Candidate Words (Phase 1)

Our task is to correct real-word spelling error from an input text using Google Web 1T data set. First, we use Google 5-gram data set to find candidate words of the word having spelling error. If the 5-gram data set fails to generate at least one candidate word then we move forward to 4-gram data set or 3-gram data set or 2-gram data set if the preceding data set fails to generate at least one candidate word. Let us consider an input text T which after tokenization has m words, i.e., $T = \{w_1, w_2, \ldots, w_i, \ldots, w_m\}$, where w_i $(i > 1)$[2] is the word having the spelling error. First, we discuss how we find the candidates for the word marked as a spelling error and, then we discuss the procedure to find the most relevant single candidate from several candidates.

Determining candidate words using the 5-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 5-gram w_{i-4} w_{i-3} w_{i-2} w_{i-1} w_i (where $m \geq i \geq 5$) in the Google Web 1T 5-grams, if the said 5-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0. The intuition behind using the *cut off frequency* is the fact that, if the word is misspelled, then the correct one should have a higher frequency than the misspelled one in the context. Thus, using the *cut off frequency*, we isolate a large number of candidates that we do not need to process.

2. We find all the 5-grams (where only w_i is changed while w_{i-4}, w_{i-3}, w_{i-2} and w_{i-1}

[2]It means that we do not consider the first word as the word with spelling error. Though, our method could have worked for the first word too. We did not do it here due to efficiency reasons. Google n-grams are sorted based on the first word, then the second word, and so on. Based on this sorting, all Google 5-grams, 4-grams, 3-grams and 2-grams are stored in 117, 131, 97 and 31 different files, respectively. For example, all the 117 Google 5-gram files could have been needed to access a single word instead of accessing just one 5-gram file, that we do for any other words. This is because when the first word needs to be corrected, it might be in any file among those 117 5-gram files.

are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 5-gram $w_{i-4}\ w_{i-3}\ w_{i-2}\ w_{i-1}\ w_{ij}$. If there is no such 5-gram (having frequency above the *cut off frequency*), our task is two-fold: we set *matched*←1, if there is at least one 5-gram, and we jump to step 5 or 6 or 7 that is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5.8) and then assign a weight using the following equation (5.10) only to the words that return the string similarity value greater than 0.5.

$$weight(w_i, w_{ij}) = \beta S_1(w_i, w_{ij}) + (1 - \beta)F(w_{ij}) \tag{5.10}$$

   Equation (5.10) is used to ensure a balanced weight between the string similarity function and the probability function, where β refers to how much importance we give to the string similarity function with respect to the frequency function[3].

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number[4] of words as the candidate words[5].

5. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram w_{i-3}

[3]We give more importance to string similarity function with respect to frequency value function throughout the section of 'determining candidate words' to have more candidate words so that the chance of including the target word into the set of candidate words gets higher. For this reason, we heuristically set β=0.85 in equation (5.10) instead of setting β=0.5.

[4]For our experiment, we heuristically set this number as 10. If we lower this number (say 2 or 3) then there is a chance that sometimes the most appropriate candidate (solution word) fails to be included in the candidate list.

[5]Sometimes the top candidate word might be either a plural form or a past participle form of the original word. Or even it might be a high frequency function word (e.g., *the*). We omit these type of words from the candidacy.

w_{i-2} w_{i-1} w_i w_{i+1} if $m-1 \geq i \geq 4$. Otherwise, we go to next step.

6. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram w_{i-2} w_{i-1} w_i w_{i+1} w_{i+2} if $m-2 \geq i \geq 3$. Otherwise, we go to next step.

7. If this step is not visited yet then we follow step 1 to step 4 with the 5-gram w_{i-1} w_i w_{i+1} w_{i+2} w_{i+3} if $m-3 \geq i \geq 2$. Otherwise, we go to next step.

8. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

9. If we find more than one word in step 4, we go to section 5.3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

Determining candidate words using the 4-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 4-gram w_{i-3} w_{i-2} w_{i-1} w_i (where $m \geq i \geq 4$) in the Google Web 1T 4-grams, if the said 4-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 4-grams (where only w_i is changed while w_{i-3}, w_{i-2} and w_{i-1} are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 4-gram w_{i-3} w_{i-2} w_{i-1} w_{ij}. If there is no such 4-gram, our task is two folds: we set *matched*$\leftarrow$1, if there is at least one 4-gram having any frequency and we jump to step 5 or 6 that is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5.8) and then assign a weight using equation (5.10) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If this step is not visited yet then we follow step 1 to step 4 with the 4-gram w_{i-2} w_{i-1} w_i w_{i+1} if $m - 1 \geq i \geq 3$. Otherwise, we go to next step.

6. If this step is not visited yet then we follow step 1 to step 4 with the 4-gram w_{i-1} w_i w_{i+1} w_{i+2} if $m - 2 \geq i \geq 2$. Otherwise, we go to next step.

7. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

8. If we find more than one word in step 4, we go to section 5.3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

Determining candidate words using the 3-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 3-gram w_{i-2} w_{i-1} w_i (where $m \geq i \geq 3$) in the Google Web 1T 3-grams, if the said 3-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 3-grams (where only w_i is changed while w_{i-2} and w_{i-1} are unchanged), if any, having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is

the frequency of the 3-gram w_{i-2} w_{i-1} w_{ij}. If there is no such 3-gram, our task is two folds: we set *matched*←1, if there is at least one 3-gram having any frequency and we jump to step 5, if it is yet to visit.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5.8) and then assign a weight using equation (5.10) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If this step is not visited yet then we follow step 1 to step 4 with the 3-gram w_{i-1} w_i w_{i+1} if $m - 1 \geq i \geq 2$. Otherwise, we go to next step.

6. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

7. If we find more than one word in step 4, we go to section 5.3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit.

Determining candidate words using the 2-gram data set

We use the following steps:

1. We define the term *cut off frequency* for word w_i as the frequency of the 2-gram w_{i-1} w_i (where $m \geq i \geq 2$) in the Google Web 1T 2-grams, if the said 2-gram exists. Otherwise, we set the *cut off frequency* of w_i as 0.

2. We find all the 2-grams (where only w_i is changed while w_{i-1} is unchanged) having frequency greater than the *cut off frequency* of w_i (determined in step 1). Let us consider that we find n replacements of w_i which are $R_1 = \{w_{i1}, w_{i2}, \cdots, w_{in}\}$ and

their frequencies $F_1 = \{f_{i1}, f_{i2}, \cdots, f_{in}\}$ where f_{ij} is the frequency of the 2-gram $w_{i-1}\ w_{ij}$. If there is no such 2-gram, we set *matched*←1, if there is at least one 2-gram having any frequency.

3. For each $w_{ij} \in R_1$, we calculate the string similarity between w_{ij} and w_i using equation (5.8) and then assign a weight using equation (5.10) only to the words that return the string similarity value greater than 0.5.

4. We sort the words found in step 3 that were given weights, if any, in descending order by the assigned weights and keep only a fixed number of words as the candidate words.

5. If we find exactly one word in step 4, then return that word as the suggestion word and exit.

6. If we find more than one word in step 4, we go to section 5.3.5. Otherwise, if *matched*=1 then return *no suggestion* and exit. Otherwise, we proceed to phase 2 (section 5.3.4).

5.3.4 Determining Candidate Words (Phase 2)

The question of why we use phase 2 is best understood by the example "$\cdots$ by releasing the WPPSS *retort*." (Table 5.1) where *retort* is the observed word that needs to be corrected. But, there is no such 5-gram where *retort* can be changed by following the condition of having the string similarity between *retort* and w_i be at least 0.5 and keeping "by releasing the WPPSS" unchanged. Similarly, there is no such 4-gram, 3-gram and 2-gram where *retort* can be changed by following the same previous condition and keeping "releasing the WPPSS", "the WPPSS" and "WPPSS" unchanged respectively. The reason of the unavailability of such n-grams is that "WPPSS" is not a very common word in the Google Web 1T data set.

To solve this issue is straightforward. We follow phase 1 with some small changes: instead of trying to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where only w_i is changed while keeping all of $\{\cdots, w_{i-2}, w_{i-1}\}$ unchanged, we try to find all the n-grams ($n \in \{5, 4, 3, 2\}$) where w_i, as well as any but the first member of $\{\cdots, w_{i-2}, w_{i-1}\}$ are changed while keeping the remaining of $\{\cdots, w_{i-2}, w_{i-1}\}$ unchanged.

5.3.5 Determining the Suggestion Word

We use this section only if we have more than one candidate word found in section 5.3.3 or section 5.3.4. Let us consider that we find n candidate words of w_i in section 5.3.3 or section 5.3.4 which are $\{w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots, w_{in}\}$. For each w_{ij}, we use the string similarity value between w_{ij} and w_i (already calculated using equation (5.8)) and the normalized frequency value of w_{ij} (already calculated using equation (5.9)) and then calculate the weight value using equation (5.10) by setting $\beta = 0.5$. We find the index, $\bar{j}$, of the word having the maximum weight value which is the target *suggestion word*, $w_{i\bar{j}}$, by the following equation:

$$\bar{j} = \underset{j \in n}{\operatorname{argmax}}\ weight(w_i, w_{ij}) \tag{5.11}$$

5.4 Evaluation and Experimental Results

We used as test data the same data that Wilcox-O'Hearn et al. (2008) used in their evaluation of Mays et al. (1991) method, which in turn was a replication of the data used by Hirst and St-Onge (1998) and Hirst and Budanitsky (2005) to evaluate their methods.

The data consisted of 500 articles (approximately 300,000 words) from the 1987−89 *Wall Street Journal* corpus, with all headings, identifiers, and so on removed; that is, just a long stream of text. It is assumed that this data contains no errors; that is, the

Wall Street Journal contains no malapropisms or other typos. In fact, a few typos (both non-word and real-word) were noticed during the evaluation, but they were small in number compared to the size of the text.

Malapropisms were randomly induced into this text at a frequency of approximately one word in 200. Specifically, any word whose base form was listed as a noun in WordNet (but regardless of whether it was used as a noun in the text; there was no syntactic analysis) was potentially replaced by any spelling variation found in the lexicon of the *ispell* spelling checker[6]. A *spelling variation* was defined as any word with an *edit distance* of 1 from the original word; that is, any single-character insertion, deletion, or substitution, or the transposition of two characters, that results in another real word. Thus, none of the induced malapropisms were derived from closed-class words, and none were formed by the insertion or deletion of an apostrophe or by splitting a word. Though Wilcox-O'Hearn et al. (2008) mentioned that the data contained 1402 inserted malapropisms, there were only 1391 malapropisms.

Because it had earlier been used for evaluating Mays et al. (1991)'s trigram method, which operates at the sentence level, the data set had been divided into three parts, without regard for article boundaries or text coherence: sentences into which no malapropism had been induced; the original versions of the sentences that received malapropisms; and the malapropized sentences. In addition, all instances of numbers of various kinds had been replaced by tags such as <INTEGER>, <DOLLAR VALUE>, and <PERCENTAGE VALUE>. Actual (random) numbers or values were restored for these tags. Some spacing anomalies around punctuation marks were corrected. A detailed description of this data can be found in (Hirst, 2008; Wilcox-O'Hearn et al., 2008).

Some examples of successful and unsuccessful corrections, using Google 5-grams, are shown in Table 5.1. Some of the malapropisms created were "unfair" in the sense that no

[6]Ispell is a fast screen-oriented spelling checker that shows you your errors in the context of the original file, and suggests possible corrections when it can figure them out.

SUCCESSFUL CORRECTION:
$\cdots$ chance to mend his *fencers* $\rightarrow$ fences [fences] with Mr. Jefferies $\cdots$
$\cdots$ breathtaking, the vast *steep* $\rightarrow$ sweep [sweep] of the land $\cdots$
$\cdots$ employees is the largest *employee* $\rightarrow$ employer [employer] in Europe and $\cdots$
SUCCESSFUL CORRECTION (in Second Phase):
$\cdots$ by releasing the WPPSS *retort* $\rightarrow$ report [report].
$\cdots$ reported, the Clore *grout* $\rightarrow$ group [group] in October placed $\cdots$
$\cdots$ tests comparing its potpourri *covert* $\rightarrow$ cover [cover] with the traditional $\cdots$
FALSE POSITIVE CORRECTION:
$\cdots$ can almost see the *firm* $\rightarrow$ fire [farm] issue receding.
$\cdots$ the Senate to support *aim* $\rightarrow$ him [aid] for the Contras $\cdots$
$\cdots$ change or kill any *storm* $\rightarrow$ form [story] , and he $\cdots$
FALSE NEGATIVE:
I trust that the *contract* [contrast] between the American $\cdots$
$\cdots$ as much as <DOLLAR_VALUE> *billion* [million].
$\cdots$ company's Lynx golfequipment *busyness* [business], which Mr. $\cdots$

Table 5.1: Examples of successful and unsuccessful corrections using Google 5-grams. Italics indicate the observed word, arrow indicates the correction, square brackets indicate the intended word.

automatic procedure could reasonably be expected to see the error. The canonical case is the substitution of *million* for *billion*, or vice versa[7]; another is *employee* for *employer*, or vice versa, in many (but not all) contexts. In some cases, the substitution was merely a legitimate spelling variation of the same word (e.g., *labour* for *labor*).

Table 5.2 shows some examples of successful and unsuccessful corrections using Google 4-grams where Google 5-grams fail to generate any suggestion. In Table 5.3 and Table 5.4, some examples of successful and unsuccessful corrections using Google 3-grams (where Google 5-grams and 4-grams fail to generate any suggestion) and using Google 2-grams (where Google 5-grams, 4-grams and 3-grams fail to generate any suggestion) are shown respectively.

[7]One such example is shown in FALSE NEGATIVE section in Table 5.1. There are 40 malapropisms in the data set related with *million/billion*.

SUCCESSFUL CORRECTION:
··· one of a corporate *raiser* → raider [raider]'s five most ···
··· Bonderman's penchant for bright *pick* → pink [pink] or yellow socks ···
··· of rumors about insider *tracing* → trading [trading] in Pillsbury options ···
SUCCESSFUL CORRECTION (in Second Phase):
··· , for the Belzberg *brothels* → brothers [brothers]' First City ···
··· typical of Iowa's *food* → mood [mood] a month before ···
··· four makers of snowmobiles *tern* → turn [turn] out about 23 ···
FALSE POSITIVE CORRECTION:
··· I'm uncomfortable *tacking* → talking [taking] a lot of ···
He said the *vales* → values [valves] were sold to···
··· to approve a formal *teat* → test [text] of their proposed ···
FALSE NEGATIVE:
A lot of the *optimisms* [optimists] were washed out ···
··· to service the <DOLLAR_VALUE> *billion* [million] Polo/Ralph ···
··· published its support of *patent* [patient]-funded research ···

Table 5.2: Examples of successful and unsuccessful corrections using Google 4-grams. For these examples, Google 5-grams fail to generate any suggestion.

For each error, our method returns either a suggestion (which is either correct or wrong) or *no suggestion*. Figure 5.2 shows the number of errors where either a suggestion or *no suggestion* is generated for different combinations of n-grams used. To give an example, using only 5-grams, each of 505 errors either generate a suggestion or *no suggestion*. It also means that in the next 5-4-gram combination, we only process 886 errors[8] (i.e., 1391-505). Figure 5.2 validates the intuition behind using a combination of n-grams rather using only n-grams (e.g., 5-grams) by showing that while single 5-gram generates either a suggestion or *no suggestion* for only 505 errors, a combination of 5-4-3-2-5′-4′-3′-2′-gram generates the same for all 1391 errors.

Figure 5.3 breaks down the numbers shown in Figure 5.2 into *true positive*, *false*

[8]We use the result of the previous 5-grams in 5-4-gram combination, thus only use 4-grams in this combination.

SUCCESSFUL CORRECTION:
··· Disappointment turned to dire *traits* → straits [straits] when Vestron's ···
··· pay <DOLLAR_VALUE> million in *destitution* → restitution [restitution] related to contract ···
··· believes that nationhood per *sec* → se [se] is obsolete.
SUCCESSFUL CORRECTION (in Second Phase):
··· its Ally & Gargano *unity* → unit [unit] settled their suit ···
··· 23 5/8 after rising 23 3/8 *prints* → points [points] Tuesday.
FALSE POSITIVE CORRECTION:
··· working on improving his *lent* → talent [left].
··· company feels the 23 *mate* → rate [date] is for the ···
··· going to happen, *calming* → claiming [calling] the move just ···
FALSE NEGATIVE:
··· which built up tremendous *leadership* [readership] and advertising in ···
··· town saloon after the *battle* [cattle] roundup.
··· be this rock-*firm* [film] orgy, an ···

Table 5.3: Examples of successful and unsuccessful corrections using Google 3-grams. For these examples, Google 5-grams and 4-grams fail to generate any suggestion.

positive and *false negative*. For example, using only 5-grams, we get 493 suggestions, 472 out of them are correct and 21 are incorrect, along with 12 *no suggestion*. Figure 5.3 also shows that a combination of 5-4-3-2-5′-4′-3′-2′-gram generates 1343 suggestions, 1222 out of them are correct and 121 are incorrect, along with 48 *no suggestion*.

The performance is measured using *Precision* (P), *Recall* (R) and *F-measure* (F). Figure 5.4 shows *precision*, *recall* and *F-measure* for different combinations of n-gram used. We get highest *precision* (0.96) when using only 5-gram which is obvious because 5-gram uses maximum possible context words (which is four) and as a result the chance of getting highest ratio between the number of correct suggestions returned and the number of suggestions returned increases. But the *recall* at this level is very poor (only 0.34). Figure 5.4 demonstrates how *recall* gets better using different combinations of n-gram while keeping *precision* as high as possible. Using 5-gram to a combination

SUCCESSFUL CORRECTION:
··· raider or Zurn management *making* → taking [taking] shares out of ···
··· Silesia, said Solidarity *advisor* → adviser [adviser] Jacek Kuron.
··· that lack strong domestic *inventor* → investor [investor] interest, such ···
SUCCESSFUL CORRECTION (in Second Phase):
Mr. Mutert *votes* → notes [notes] that investors have ···
FALSE POSITIVE CORRECTION:
··· relieved if Super Tuesday *toughs* → tours [coughs] up a front ···
··· replaced the supposedly flawed *vales* → values [valves], known technically ···
··· be reserved about indiscriminate *clays* → ways [plays] in some groups ···
FALSE NEGATIVE:
··· a Drexel Burnham Lambert *petals* [metals] economist.
··· aimed at clearing out *overstuffing* [overstaffing] left by previous ···
··· NMS rose (8.4 *billion* [million]) than fell ···

Table 5.4: Examples of successful and unsuccessful corrections using Google 2-grams. For these examples, Google 5-grams, 4-grams and 3-grams fail to generate any suggestion.

of 5-4-3-2-gram, we get a significant improvement of *recall* but after that (i.e., a combination of 5-4-3-2-gram to a combination of 5-4-3-2-5′-4′-3′-2′-gram), we get only 0.01 *recall* increase. As a result, the question of whether it makes any sense to try these later combinations arises. We try to answer this issue by using Figure 5.5 which shows the number of errors processed[9] for different combinations of n-gram used. Figure 5.5 shows that we need to process only 48 more errors when we move from a combination of 5-4-3-2-gram to a combination of 5-4-3-2-5′-4′-3′-2′-gram, and we get 16 correct suggestions, 5 wrong suggestion and 2 *no suggestion*. Thus, it is worth taking into account these later combinations.

[9]When we use only 5-grams, we process all 1391 errors (where we do not get either a suggestion or *no suggestion* for 886 errors) and then when we use 4-grams along with 5-grams for 5-4-gram combination, we process these unsolved 886 errors again, thus totalling the number of errors processed to 2277 for 5-4-gram combination.

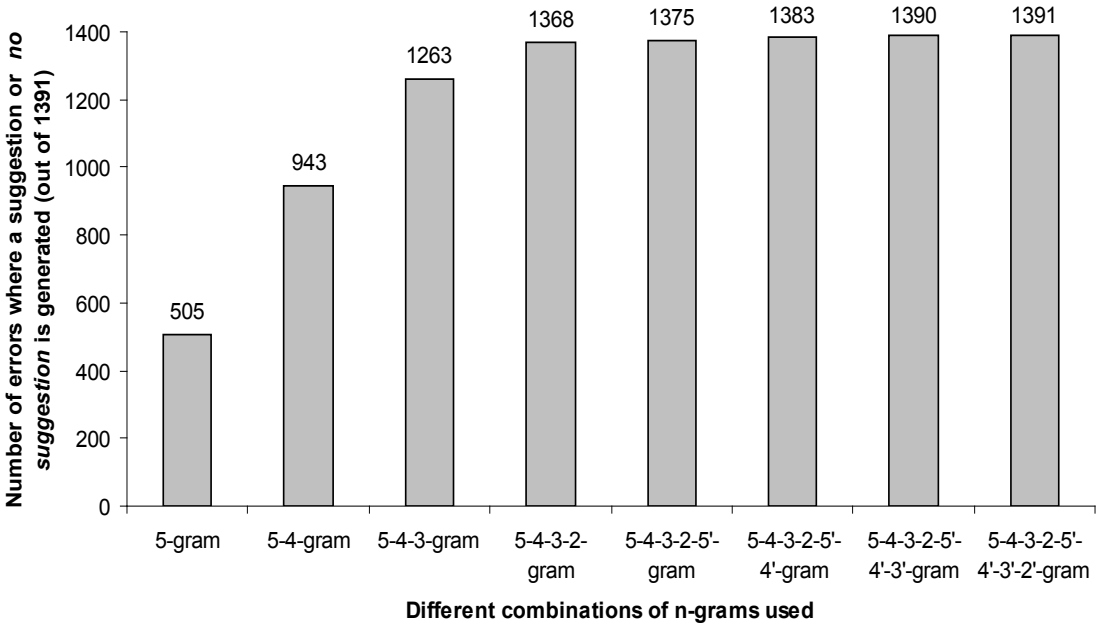

Figure 5.2: Number of errors where a suggestion or *no suggestion* is generated for different combinations of n-grams used. Apostrophe ($'$) is used to denote the n-grams used in phase 2. x-y-$\cdots z$-gram means that we use x-grams, y-grams, $\cdots$ and z-grams.

We cannot directly compare our results with the correction results from previous work, because in that work the correction was run on the results of the detection module, cumulating the errors, while our correction module ran on the correctly-flagged spelling errors. Still, we indirectly try to compare our results with the previous work. Table 5.5 shows our method's results on the described data set compared with the results for the trigram method of Wilcox-O'Hearn et al. (2008) and the lexical cohesion method of Hirst and Budanitsky (2005). The data shown here for trigram method are not from (Wilcox-O'Hearn et al., 2008), but rather are later results following some corrections reported in (Hirst, 2008)[10]. That the corrected result of Wilcox-O'Hearn et al. (2008) can detect

[10]The result (detection *recall*=0.544, detection *precision*=0.528, correction *recall*=0.491, correction *precision*=0.503) mentioned in (Hirst, 2008) seems to have some inconsistency in correction *recall* and correction *precision*. Detection true positives (762) and detection false positives (681) can be calculated from detection *precision* and *recall*. Correction true positives and correction false positives are 688 and 755, respectively, given that the correction *recall* is 0.491. Thus, correction *precision* is 0.477.

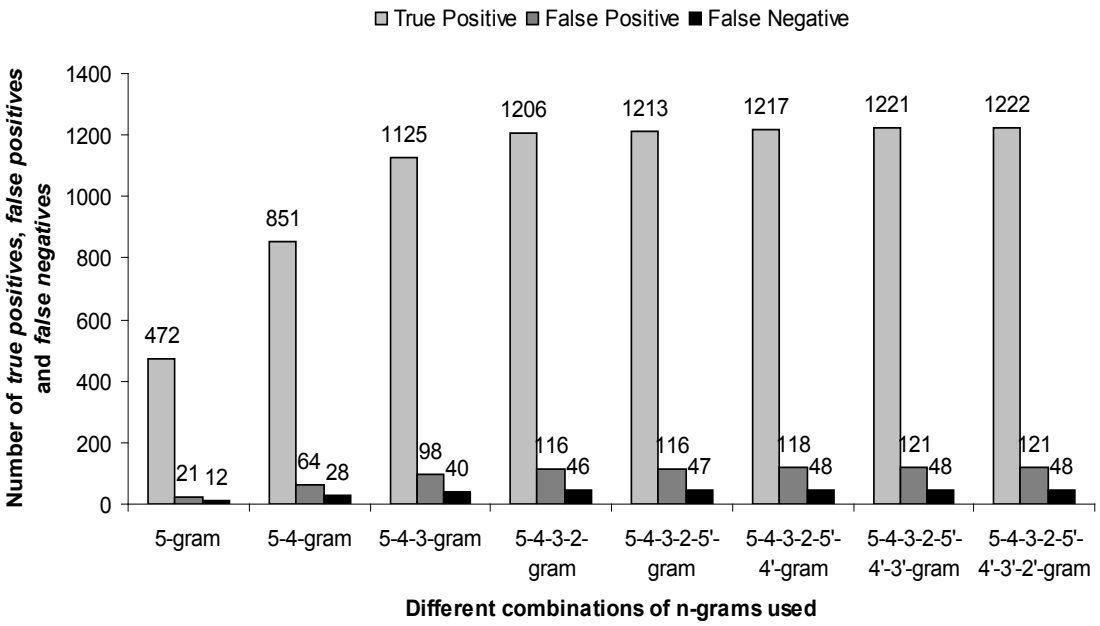

Figure 5.3: Number of true positives, false positives and false negatives for different combinations of n-grams used.

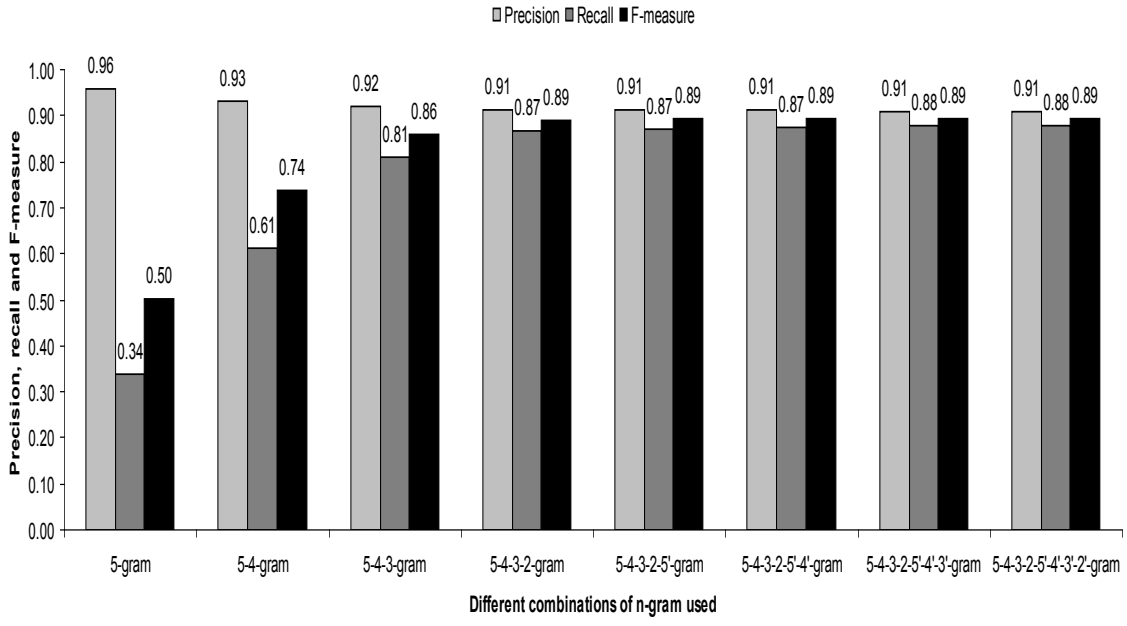

Figure 5.4: Precision, recall and F-measure for different combinations of n-grams used.

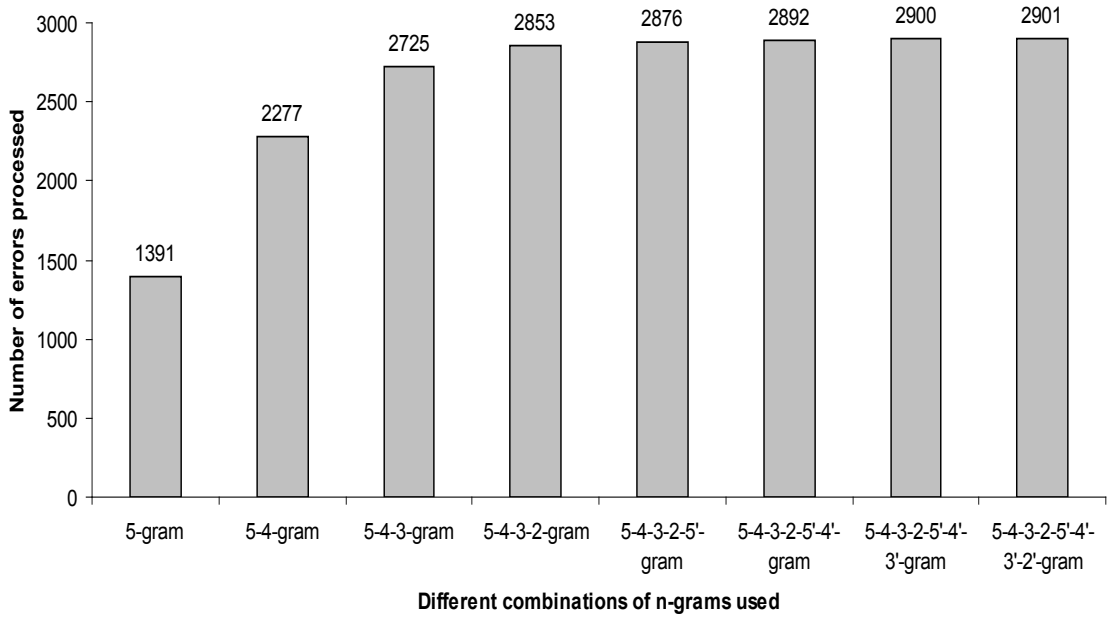

Figure 5.5: Number of errors processed for different combinations of n-grams used.

762 errors and thus correct 688 errors out of these 762 detected errors means each of the correction *precision*, *recall* and F-measure is 0.9. It is obvious that the performance of correcting the remaining of the undetected errors will not be the same as correcting the detected errors because these errors are difficult to correct since they are difficult to detect in the first place. Still, the correction performance of our proposed method is comparable to the correction performance of the method that runs on the results of the detection module, cumulating the errors.

Moreover, considering the fact[11] that 85 malapropisms out of 1391 created were "unfair" in the sense that no automatic procedure could reasonably be expected to see the error (Hirst, 2008), to have a method generating 1222 correct suggestions along with 121 wrong suggestions and 48 *no suggestion* could be useful.

[11]Though we did not consider this fact in the evaluation procedure.

Detection			correction		
R	P	F	R	P	F
Lexical cohesion(Hirst and Budanitsky, 2005)					
0.306	0.225	0.260	0.281	0.207	0.238
Trigrams(Wilcox-O'Hearn et al., 2008)					
0.544	0.528	0.536	0.491	0.477	0.484
Google n-grams					
-	-	-	0.88	0.91	0.89

Table 5.5: A comparison of recall, precision, and F-measure for three methods of malapropism detection and correction on the same data set.

5.5 Summary

Our purpose in this chapter was the development of a high-quality correction module. The Google n-grams proved to be very useful in correcting real-word errors. When we tried with only 5-grams the precision (0.96) was good, though the recall (0.34) was too low. Having sacrificed a bit of the precision score, our proposed combination of n-grams method achieves a very good recall (0.88) while maintaining the precision at 0.91. Our attempts to improve the correction recall while maintaining the precision as high as possible are helpful to the human correctors who post-edit the output of the real-word spell checker. If there is no postediting, at least more errors get corrected automatically. Our method could also correct misspelled words, not only malapropism, without any modification. In future work, we plan to add a detection module and extend our method to allow for deleted or inserted words, and to find the corrected strings in the Google Web 1T n-grams. In this way we will be able to correct grammar errors too.

Chapter 6

Text Correction using n-grams

In this chapter, a more general unsupervised statistical method for automatic text correction using the Google Web 1T n-gram data set is presented. Figure 6.1 shows the flow diagram of this generic approach, where the task is to generate a list of candidate texts and then choose the best candidate text. Here, all the three basic operations: *insert*, *delete*, and *replace* are used to detect and correct errors in a text, contrary to Chapters 3, 4, and 5, where only the *replace* operation is used. In the candidate texts generating procedure, we use rules generated from 5-grams, by using all the three operations and a normalized and modified version of the Longest Common Subsequence (LCS) string matching algorithm. To sort candidate texts, we use the normalized frequency value function, along with some other similarity functions, namely common-word similarity, non-common word similarity, and common-word order similarity, unlike in Chapters 3, 4, and 5, where only the normalized frequency value function is used.

6.1 Introduction

This is the approach that corrects a text containing multiple errors of both syntactic and semantic nature and uses the *insert* and *delete* operations, along with the *replace*

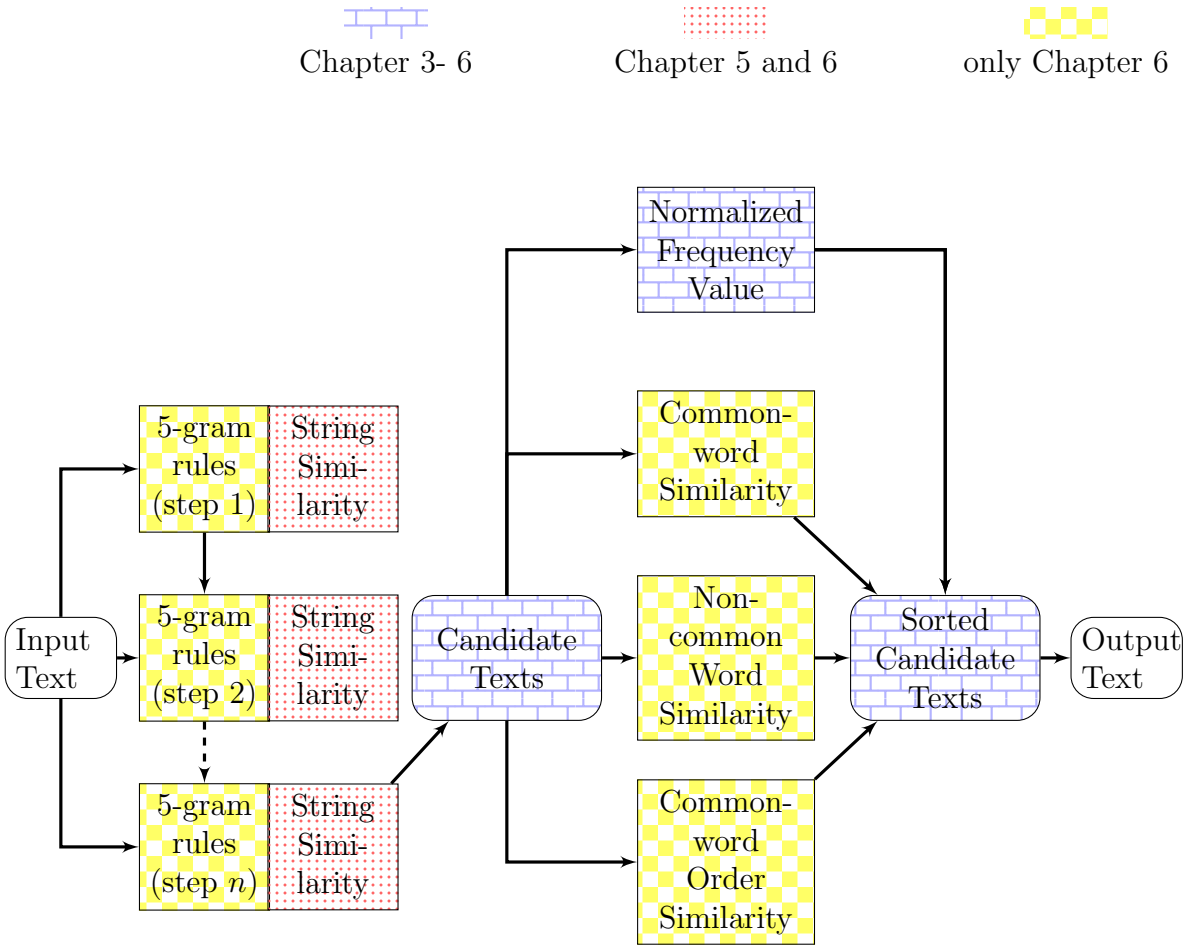

Figure 6.1: Flow diagram of the text correction approach.

operation. The basic idea of our proposed text correction approach is to first generate a set of probable candidates and then to sort those candidates based on some functions that help to decrease the number of *false positives* and *false negatives*. How we sort the candidates or how we set the balance between original input text and each candidate text are key issues of our proposed approach.

6.2 Proposed Method

The proposed method first determines some probable candidates and then finds the best one among the candidates or sorts them based on some weights. We consider three *false positive decreasing* functions (or similarity functions) and one *false negative decreasing* function (or frequency value function) in our method. The following sections present a detailed description of each of these functions followed by the procedure to determine some probable candidates, along with the procedure to sort the candidates.

6.2.1 False Positive Decreasing Functions

Similarity between Two Strings

The string similarity function that we use in this chapter to determine the similarity between two strings is described in detail in section 5.3.1 on page 80.

Common Word Similarity between Texts

If the two texts have some words in common, we can measure how similar the two texts are based on their common words. Let us consider a pair of texts, T_1 and T_2, which have m and n tokens, respectively, i.e., $T_1 = \{p_1, p_2 \ldots, p_m\}$ and $T_2 = \{r_1, r_2 \ldots, r_n\}$ and $n \geq m$. Otherwise, we switch T_1 and T_2. We count the number of p_i's (say, δ) for which $p_i = r_j$, for all $p \in T_1$ and for all $r \in T_2$. That is, there are δ tokens in T_1 that exactly match with T_2, where $\delta \leq m$. Thus, common word similarity, $S_2 \in [0, 1]$ is:

$$S_2(T_1, T_2) = \frac{2\delta}{m + n} \tag{6.1}$$

For example:

T_1 : *Many consider Maradona as the best player in soccer history.*

T_2 : *Maradona is one of the best soccer player.*

There are 5 tokens (δ) in T_1 that exactly match with T_2.

$$S_2(T_1, T_2) = \frac{2 \times 5}{8 + 10}, \text{ i.e., } S_2 = 0.56$$

Common-Word Order Similarity between Texts

If the two texts have some words in common, we can measure how similar the order of the common-words is in the two texts (if these words appear in the same order, or almost the same order, or very different order). While word-order mistakes do not figure among the most frequent error types, they are not uncommon (Lee, 2009). Although syntactic information (here approximated by the order of the common-words) has low importance for the semantic processing of short texts according to Wiemer-Hastings (2000), we incorporate the word order as an option and to make our method more generic. We can use it when we consider the importance of syntactic information by setting its weight factor. Let us consider a pair of sentences, T_1 and T_2, which have m and n tokens, respectively, i.e., $T_1 = p_1, p_2 \ldots, p_m$ and $T_2 = r_1, r_2 \ldots, r_n$ and $n \geq m$. Otherwise, we switch T_1 and T_2. We count the number of p_i's (say, δ) for which $p_i = r_j$, for all $p \in T_1$ and for all $r \in T_2$. That is, there are δ tokens in T_1 that exactly match with T_2, where $\delta \leq m$. We remove all δ tokens from T_1 and put them in X and from T_2 in Y, in the same order as they appear in the sentences. So, $X = \{x_1, x_2 \ldots, x_\delta\}$ and $Y = \{y_1, y_2 \ldots, y_\delta\}$. We replace X by assigning a unique index number for each token in X starting from 1 to δ, i.e., $X = \{1, 2 \ldots, \delta\}$. Based on this unique index numbers for each token in X, we also replace Y where $X = Y$. We propose a measure for measuring the common-word order similarity of two texts as:

$$S_3(T_1, T_2) = 1 - \frac{|x_1 - y_1| + |x_2 - y_2| \ldots + |x_\delta - y_\delta|}{|x_1 - x_\delta| + |x_2 - x_{\delta-1}| \ldots + |x_\delta - x_1|} \tag{6.2}$$

That is, common-word order similarity is determined by the normalized difference of common-word order (the denominator is the largest possible dissimilarity value, the worse order of pairwise elements in X). (6.3) demonstrates three individual cases of (6.2)[1].

$$S_3(T_1, T_2) = \begin{cases} 1 - \frac{2\sum_{i=1}^{\delta} |x_i - y_i|}{\delta^2} & \text{if } \delta \text{ is even} \\ 1 - \frac{2\sum_{i=1}^{\delta} |x_i - y_i|}{\delta^2 - 1} & \text{if } \delta \text{ is odd and } \delta > 1 \\ 1 & \text{if } \delta \text{ is odd and } \delta = 1 \end{cases} \tag{6.3}$$

For example:

T_1 : *Many consider Maradona as the best player in soccer history.*

T_2 : *Maradona is one of the best soccer player.*

There are 5 tokens (δ) in T_1 that exactly match with T_2. We remove all 5 tokens from both T_1 and T_2 to X and Y in the same order as they appear in the sentences. So, X = {*maradona, the, best, player, soccer*} and Y = {*maradona, the, best, soccer, player*}. We replace X by assigning a unique index number for each token in X starting from 1 to 5, i.e., X = {1, 2, 3, 4, 5}. Based on this unique index numbers for each token in X, we also replace Y where $X = Y$. That is, Y = {1, 2, 3, 5, 4}.

[1]The denominator is different for δ even or odd. For example, if δ is even and $X = \{1, 2, 3, 4, 5, 6\}$ then the denominator is $|1-6| + |2-5| + |3-4| + |4-3| + |5-2| + |6-1|$
$= 5 + 3 + 1 + 1 + 3 + 5$
$= 2(1 + 3 + 5)$
$= 2(1 + 3 + \cdots 2n - 1)$ [When $n = 3$]
$= 2n^2$ [As the sum of n odd numbers starting from 1 is n^2]
$= 2\frac{\delta^2}{2^2}$ [As $2n = \delta$]
$= \frac{\delta^2}{2}$
If δ is odd and $X = \{1, 2, 3, 4, 5, 6, 7\}$ then the denominator is $|1-7| + |2-6| + |3-5| + |4-4| + |5-3| + |6-2| + |7-1|$
$= 6 + 4 + 2 + 0 + 2 + 4 + 6$
$= 2(2 + 4 + 6)$
$= 2(2 + 4 + \cdots 2n)$ [When $n = 3$]
$= 2n(n + 1)$ [As the sum of n even numbers starting from 2 is $n(n + 1)$]
$= (\delta - 1) \times (\frac{\delta - 1}{2} + 1)$ [As $2n = \delta - 1$]
$= \frac{\delta^2 - 1}{2}$

$$S_3(T_1, T_2) = 1 - \frac{2\sum_{i=1}^{5} |x_i - y_i|}{5^2 - 1}, \text{ i.e., } S_3 = 0.83 \text{ as } \delta \text{ is odd and } \delta > 1.$$

Non-Common Word Similarity between Texts

If the two texts have some non-common words, we can measure how similar the two texts are based on their non-common words. Let us consider a pair of texts, T_1 and T_2, which have m and n tokens, respectively, i.e., $T_1 = \{p_1, p_2 \ldots, p_m\}$ and $T_2 = \{r_1, r_2 \ldots, r_n\}$ and $n \geq m$. Otherwise, we switch T_1 and T_2. If there are δ tokens in T_1 that exactly match with T_2, then there are $m-\delta$ and $n-\delta$ non-common words in texts T_1 and T_2, respectively. We remove all δ common tokens from both T_1 and T_2. So, $T_1 = \{p_1, p_2 \ldots, p_{m-\delta}\}$ and $T_2 = \{r_1, r_2 \ldots, r_{n-\delta}\}$. We construct a $(m - \delta) \times (n - \delta)$ *string similarity matrix* (say, $M = (S_{1_{ij}})_{(m-\delta)\times(n-\delta)}$) using the following process: we assume any token $p_i \in T_1$ has τ characters, i.e., $p_i = \{c_1 c_2 \ldots c_\tau\}$ and any token $r_j \in T_2$ has η characters, i.e., $r_j = \{c_1 c_2 \ldots c_\eta\}$ where $\tau \leq \eta$. In other words, η is the length of the longer token and τ is the length of the shorter token. We calculate the following based on equation 5.2, 5.5, 5.6, 5.7, and 5.8:

$$v_1 \leftarrow NLCS(p_i, r_j)$$
$$v_2 \leftarrow NMCLCS_1(p_i, r_j)$$
$$v_3 \leftarrow NMCLCS_n(p_i, r_j)$$
$$v_4 \leftarrow NMCLCS_z(p_i, r_j)$$
$$S_{1_{ij}} \leftarrow \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + \alpha_4 v_4$$

That is, $S_{1_{ij}}$ is a weighted sum of v_1, v_2, v_3, and v_4 where α_1, α_2, α_3, α_4 are weights and $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. We set equal weights for our experiments.

We put $S_{1_{ij}}$ in row i and column j position of the matrix for all $i = 1 \ldots m - \delta$ and

$j = 1 \ldots n - \delta$.

$$M = \begin{pmatrix} S_{1_{11}} & S_{1_{12}} & \ldots & S_{1_{1j}} & \ldots & S_{1_{1(n-\delta)}} \\ S_{1_{21}} & S_{1_{22}} & \ldots & S_{1_{2j}} & \ldots & S_{1_{2(n-\delta)}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ S_{1_{i1}} & S_{1_{i2}} & \ldots & S_{1_{ij}} & \ldots & S_{1_{i(n-\delta)}} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ S_{1_{(m-\delta)1}} & S_{1_{(m-\delta)2}} & \ldots & S_{1_{(m-\delta)j}} & \ldots & S_{1_{(m-\delta)(n-\delta)}} \end{pmatrix}$$

After constructing the *string similarity matrix*, M, we find out the maximum-valued matrix-element, $S_{1_{ij}}$. We add this matrix element to a list (say, ρ and $\rho \leftarrow \rho \cup S_{1_{ij}}$) if $S_{1_{ij}} \geq 0$. We remove all the matrix elements of i'th row and j'th column from M. We remove the row and column in order to remove the pair with maximum similarity. This makes the computation manageable: in the next steps fewer words are left for matching. We repeat the finding of the maximum-valued matrix-element, $S_{1_{ij}}$ adding it to ρ and removing all the matrix elements of the corresponding row and column until either $S_{1_{ij}} = 0$, or $m - \delta - |\rho| = 0$, or both.

We sum up all the elements in ρ and divide by $n - \delta$ to get non-common word similarity, $S_4 \in [0, 1)$:

$$S_4(T_1, T_2) = \frac{\sum_{i=1}^{|\rho|} \rho_i}{n - \delta} \tag{6.4}$$

6.2.2 False Negative Decreasing Function

Normalized Frequency Value of a Candidate Text

We determine the normalized frequency value of a candidate text (how we determine candidate texts is discussed in detail in Section 6.2.3) with respect to all other candidate texts. A candidate text having higher normalized frequency value is more likely a strong

candidate for the correction, though not always. Let us consider that we have $\tilde{n}$ candidate texts for the input text T: $\{T_1, T_2, \ldots T_i \ldots, T_{\tilde{n}}\}$

$$T_1 = \{w_{11}, w_{12}, \ldots w_{1j} \ldots w_{(1)(m_1)}\}$$
$$T_2 = \{w_{21}, w_{22}, \ldots w_{2j} \ldots w_{(2)(m_2)}\}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$T_i = \{w_{i1}, w_{i2}, \ldots w_{ij} \ldots w_{(i)(m_i)}\}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$T_{\tilde{n}} = \{w_{\tilde{n}1}, w_{\tilde{n}2}, \ldots w_{\tilde{n}j} \ldots w_{(\tilde{n})(m_{\tilde{n}})}\}$$

where w_{ij} is the jth token of the candidate text, T_i, and m_i means that the candidate text T_i has m_i tokens. It is important to note that the number of tokens each candidate text has may be different from the rest. The number of 5-grams in any candidate text, T_i is $m_i - 4$ (as T_i has m_i tokens). Again, let us consider that F_i is the set of frequencies of all the 5-grams that T_i has; f_{ij} is the frequency of the jth 5-gram of the candidate text, T_i, and $N = \{1, 2, \ldots, \tilde{n}\}$. That is:

$$F_1 = \{f_{11}, f_{12}, \ldots f_{1j} \ldots f_{(1)(m_1-4)}\}$$
$$F_2 = \{f_{21}, f_{22}, \ldots f_{2j} \ldots f_{(2)(m_2-4)}\}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$F_i = \{f_{i1}, f_{i2}, \ldots f_{ij} \ldots f_{(i)(m_i-4)}\}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$F_{\tilde{n}} = \{f_{\tilde{n}1}, f_{\tilde{n}2}, \ldots f_{\tilde{n}j} \ldots f_{(\tilde{n})(m_{\tilde{n}}-4)}\}$$

Here, $\{f_{11}, f_{21}, \ldots f_{i1} \ldots f_{\tilde{n}1}\}$, $\{f_{12}, f_{22}, \ldots f_{i2} \ldots f_{\tilde{n}2}\}$, $\{f_{1j}, f_{2j}, \ldots f_{ij} \ldots f_{\tilde{n}j}\}$ and $\{f_{(1)(m_i-4)}, f_{(2)(m_i-4)}, \ldots f_{(i)(m_i-4)} \ldots f_{(\tilde{n})(m_i-4)}\}$ are the sets of 5-gram frequencies for all $\tilde{n}$ candidate texts that are processed in the first step, in the second step, in the jth step and in the $(m_i - 4)$th step[2] respectively. Note that $(m_i - 4)$th step is the last step for candidate text T_i having m_i tokens. We calculate the normalized frequency value of a candidate text as the summation of all the 5-gram frequencies of the candidate text over the summation of the maximum frequencies in each step that the candidate text may have. Thus the normalized frequency value of T_i represented as $S_5(T_i) \in [0, 1]$ is :

$$S_5(T_i) = \frac{\sum_{j=1}^{m_i-4} f_{ij}}{\max_{k\in N} f_{k1} + \max_{k\in N} f_{k2} + \ldots + \max_{k\in N} f_{k(m_i-4)}}$$

$$= \frac{\sum_{j=1}^{m_i-4} f_{ij}}{\sum_{l=1}^{m_i-4} \max_{k\in N} f_{kl}} \tag{6.5}$$

To give an example, consider T_8 in Table 6.6 on page 125 from the walk-through example, where T_8 has m_i=6 tokens.

$$S_5(T_8) = \frac{\sum_{j=1}^{2} f_{8j}}{\sum_{l=1}^{2} \max_{k\in N} f_{kl}} = \frac{7447 + 1267}{7447 + 15049} = 0.387$$

The values of f_{81}, $\max_{k\in N} f_{k1}$ and f_{82}, $\max_{k\in N} f_{k2}$ can be found in Table 6.3 on page 123 and in Table 6.4 on page 123, respectively.

[2]By the first step, we mean the step when we process the first possible 5-grams in the input text. Similarly, by the second step, we mean the step when we process the next possible 5-grams (by removing the first token from the 5-grams used in first step and adding an extra word from the input text or other way, which is discussed in detail in Section 6.2.3) in the input text, and so on.

6.2.3 Determining Candidate Texts

Let us consider an incorrect input text, T which after tokenization has m tokens, i.e., $T = \{w_1, w_2 \ldots, w_m\}$. We use the assumptions discussed in Section 1.2 on page 6. Our approach consists in going from left to right according to a set of rules that are listed in Table 6.1 and Table 6.2. The idea here is to first use all the possible rules using the three basic operations: *Insert*, *Replace* and *Delete*. We also use *No Operation* to mean that we not use any *Insert*, *Replace* or *Delete* operation rather we directly use the next token from T to list a 5-gram rule. Using these 5-gram rules we ultimately generate 5-grams and their frequencies in several steps from the Google Web 1T 5-grams.

5-gram Rules Used in Step 1

Table 6.1 lists all possible 5-gram rules generated from the said operations and assumptions. We use each of these 5-gram rules to generate a set of 5-grams and their frequencies by trying to match the 5-gram rule with the Google Web 1T 5-grams. The set of 5-grams and their frequencies for a specific 5-gram rule can be empty or can have one or more 5-grams and their frequencies. We take the decision of how many candidate 5-grams generated from each 5-gram rule we keep for further processing (say, $\bar{n}$) .

5-gram Rule #1 in Table 6.1 says that we take the first five tokens from T to generate a 5-gram and try to match with the Google Web 1T 5-grams to generate the only candidate 5-gram and its frequency, if there is any matching.

In 5-gram Rule #2, we take the first four tokens from T and try to insert each word from a list of words (our goal here is to determine this list of words; it might be empty) in between first word (w_1) and second word (w_2) to generate a list of 5-grams and try to match with the Google Web 1T 5-grams to generate a set of 5-grams and their frequencies. All I's and R's in Table 6.1 and Table 6.2 function similarly to variables and all $w_i \in T$ function similarly to constants. Thus, 5-gram Rule #2 (w_1 I_1 w_2 w_3 w_4)

can generate a list of 5-grams and their frequencies based on all the possible values of I_1. We sort these 5-grams in descending order by their frequencies and only keep the top $\bar{n}$ 5-grams and their frequencies if $\bar{n}$ is smaller than the number of generated 5-grams. Otherwise, we keep all the generated 5-grams and their frequencies.

In 5-gram Rule #12, we take the first five tokens from T and try to replace the second token (w_2) by R_2. Thus, 5-gram Rule #12 ($w_1\ R_2\ w_3\ w_4\ w_5$) can generate a list of 5-grams and their frequencies based on all the possible values of R_2, a set of all replaceable words of w_2. We determine the string similarity between w_2 and each member of R_2 using (5.8) and sort the list of 5-grams and their frequencies in descending order by these string similarity values and only keep $\bar{n}$ 5-grams and their frequencies if $\bar{n}$ is smaller than the number of generated 5-grams. Otherwise, we keep all the generated 5-grams and their frequencies.

In 5-gram Rule #22, we take the first six tokens from T and remove the second one (i.e., w_2). Thus, 5-gram Rule #22 ($w_1\ w_3\ w_4\ w_5\ w_6$) generates a 5-gram and tries to match it with the Google Web 1T 5-grams to generate the only candidate 5-gram and its frequency, if there is any matching.

In Table 6.1 and Table 6.2, R_i means that it replaces word $w_i \in T$. Apparently it seems that in Table 6.1 there is no difference between the last 5-gram rule of *Single Insert* (rule#5) and the last 5-gram rule of *Single Replace* (rule#15); but this is not the case as the next token (which is in T) that joins to the last 5-gram of *Single Insert* is w_5, whereas the next token that joins to the last 5-gram of *Single Replace* is w_6. Again, candidate 5-grams for *Single Insert* are chosen based on frequency of the candidate 5-grams, whereas candidate 5-grams for *Single Replace* are chosen based on string similarity between the original token and each of the candidate tokens. Similarly, the next token that joins to the last 5-gram of *Consecutive Double Insert* is w_4, whereas the next token that joins to the last 5-gram of *Consecutive Double Replace* is w_6.

Table 6.1: List of all possible 5-gram rules in step 1

Rule #	5-gram Rule	Generated from (Operation Name)	Rule #	5-gram Rule	Generated from (Operation Name)
			39	$w_1\ I_1\ w_3\ w_4\ w_5$	Consecutive Single
1	$w_1\ w_2\ w_3\ w_4\ w_5$	No Operation	40	$w_1\ w_2\ I_1\ w_4\ w_5$	Insert+Single Delete
			41	$w_1\ w_2\ w_3\ I_1\ w_5$	
2	$w_1\ I_1\ w_2\ w_3\ w_4$		42	$w_1\ w_3\ R_4\ w_5\ w_6$	
3	$w_1\ w_2\ I_1\ w_3\ w_4$		43	$w_1\ w_2\ w_4\ R_5\ w_6$	
4	$w_1\ w_2\ w_3\ I_1\ w_4$	Single Insert	44	$w_1\ w_3\ w_4\ R_5\ w_6$	Single Delete +
5	$w_1\ w_2\ w_3\ w_4\ I_1$		45	$w_1\ w_2\ w_4\ w_5\ R_6$	Single Replace
			46	$w_1\ w_3\ w_4\ w_5\ R_6$	
			47	$w_1\ w_2\ w_3\ w_5\ R_6$	
6	$w_1\ I_1\ w_2\ I_2\ w_3$		48	$w_1\ R_2\ w_3\ w_5\ w_6$	
7	$w_1\ w_2\ I_1\ w_3\ I_2$	Double Insert	49	$w_1\ w_2\ R_3\ w_4\ w_6$	Single Replace +
8	$w_1\ I_1\ w_2\ w_3\ I_2$		50	$w_1\ R_2\ w_3\ w_4\ w_6$	Single Delete
9	$w_1\ I_1\ I_2\ w_2\ w_3$	Consecutive	51	$w_1\ I_1\ w_2\ R_3\ w_4$	
10	$w_1\ w_2\ I_1\ I_2\ w_3$	Double	52	$w_1\ w_2\ I_1\ w_3\ R_4$	Single Insert +
11	$w_1\ w_2\ w_3\ I_1\ I_2$	Insert	53	$w_1\ I_1\ w_2\ w_3\ R_4$	Single Replace
12	$w_1\ R_2\ w_3\ w_4\ w_5$				
13	$w_1\ w_2\ R_3\ w_4\ w_5$		54	$w_1\ R_2\ w_3\ I_1\ w_4$	Single Replace +
14	$w_1\ w_2\ w_3\ R_4\ w_5$	Single Replace	55	$w_1\ w_2\ R_3\ w_4\ I_1$	Single Insert
15	$w_1\ w_2\ w_3\ w_4\ R_5$		56	$w_1\ R_2\ w_3\ w_4\ I_1$	
16	$w_1\ R_2\ w_3\ R_4\ w_5$		57	$w_1\ I_1\ R_2\ w_3\ w_4$	Consecutive Single
17	$w_1\ w_2\ R_3\ w_4\ R_5$	Double Replace	58	$w_1\ w_2\ I_1\ R_3\ w_4$	Insert+Single
18	$w_1\ R_2\ w_3\ w_4\ R_5$		59	$w_1\ w_2\ w_3\ I_1\ R_4$	Replace
19	$w_1\ R_2\ R_3\ w_4\ w_5$	Consecutive	60	$w_1\ R_2\ I_1\ w_3\ w_4$	Consecutive Single
20	$w_1\ w_2\ R_3\ R_4\ w_5$	Double	61	$w_1\ w_2\ R_3\ I_1\ w_4$	Replace + Single
21	$w_1\ w_2\ w_3\ R_4\ R_5$	Replace	62	$w_1\ w_2\ w_3\ R_4\ I_1$	Insert
22	$w_1\ w_3\ w_4\ w_5\ w_6$		63	$w_1\ R_3\ w_4\ w_5\ w_6$	
23	$w_1\ w_2\ w_4\ w_5\ w_6$		64	$w_1\ w_2\ R_4\ w_5\ w_6$	Consecutive Single
24	$w_1\ w_2\ w_3\ w_5\ w_6$	Single Delete	65	$w_1\ w_2\ w_3\ R_5\ w_6$	Delete + Single
25	$w_1\ w_2\ w_3\ w_4\ w_6$		66	$w_1\ w_2\ w_3\ w_4\ R_6$	Replace
26	$w_1\ w_3\ w_5\ w_6\ w_7$		67	$w_1\ w_3\ I_1\ w_4\ w_5$	
27	$w_1\ w_2\ w_4\ w_6\ w_7$		68	$w_1\ w_3\ w_4\ I_1\ w_5$	
28	$w_1\ w_3\ w_4\ w_6\ w_7$	Double Delete	69	$w_1\ w_2\ w_4\ I_1\ w_5$	Single Delete +
29	$w_1\ w_2\ w_4\ w_5\ w_7$		70	$w_1\ w_2\ w_4\ w_5\ I_1$	Single Insert
30	$w_1\ w_2\ w_3\ w_5\ w_7$		71	$w_1\ w_3\ w_4\ w_5\ I_1$	
31	$w_1\ w_3\ w_4\ w_5\ w_7$		72	$w_1\ w_2\ w_3\ w_5\ I_1$	
32	$w_1\ w_4\ w_5\ w_6\ w_7$		73	$w_1\ R_2\ w_4\ w_5\ w_6$	Consecutive Single
33	$w_1\ w_2\ w_5\ w_6\ w_7$	Consecutive	74	$w_1\ w_2\ R_3\ w_5\ w_6$	Replace + Single
34	$w_1\ w_2\ w_3\ w_6\ w_7$	Double	75	$w_1\ w_2\ w_3\ R_4\ w_6$	Delete
35	$w_1\ w_2\ w_3\ w_4\ w_7$	Delete			
36	$w_1\ I_1\ w_2\ w_4\ w_5$				
37	$w_1\ w_2\ I_1\ w_3\ w_5$	Single Insert +			
38	$w_1\ I_1\ w_2\ w_3\ w_5$	Single Delete			

Limits for the Number of Steps, n_{tp}

Before going into how we use the 5-gram rules in next steps, we need to figure out what maximum and minimum number of steps we need for an input text, T having m tokens. Our second assumption is that there should be at least three words in an input text. Taking this assumption into consideration, it is obvious that if the value of m is 3 (number of words would be also 3) then only rules # 6, 7, 8, 9, 10 and 11 in Table 6.1 can be used to generate 5-grams. 5-grams generated from rule # 11 in Table 6.1 can not be used in the next step as after the last word (w_3); we might have at most two consecutive errors and all the 5-grams, if any, generated using this rule have those errors (i.e., I_1 and I_2). 5-grams generated from rules # 7 and 8 in Table 6.1 can be used in the next step to test whether we can insert another word or not (by rule # 7 in Table 6.2). But 5-grams generated from rules # 6, 9 and 10 in Table 6.1 can be used in the next two steps (by rule # 2 in Table 6.2) to test whether we can insert a word in each of the two next steps, provided that each previous step generates at least one 5-gram. Thus, if $m = 3$ we might need at most 3 steps. Now, if $m = 4$ then for the added word (i.e., w_4) we need three extra steps to test rules # 14, 2 and 7 in Table 6.2, in order, on top of the previous three steps (for the first three words) provided that each previous step generates at least one 5-gram. That is, each extra token in T needs at most three extra steps. We generalize the maximum number of steps needed for an input text having m tokens as:

$$\begin{aligned} \text{Max } n_{tp} &= 3+(m-3)\times 3 \\ &= 3m-6 \end{aligned} \tag{6.6}$$

If any step fails to generate at least one 5-gram then we do not need to use (actually we cannot use) the next step; this is the reason why we consider what minimum number of

steps we can have for an input text T having m tokens. Taking the second assumption into consideration, it is obvious that, if the value of m is 3, then only rules # 6, 7, 8, 9, 10 and 11 in Table 6.1 can be used to generate 5-grams. 5-grams generated from rule # 11 in Table 6.1 can not be used in the next step as according to the second assumption after the last word (w_3) we might have at most two consecutive errors, and all the 5-grams, if any, generated using this rule have those errors (i.e., I_1 and I_2). Thus, the minimum number of steps is ensured if all the other rules (i.e., rules # 6, 7, 8, 9 and 10 in Table 6.1) including rule # 11 in step 1, do not generate any 5-gram. This means that, if $m = 3$, then we might need at least 0 steps[3]. Now, if $m = 4$ then for the added word (i.e., w_4) we need only an extra step to test rule # 14 in Table 6.2 on top of the previous single step (for the first three tokens). That is, each extra token in T needs at least one extra step provided that each previous step for each extra token generates at least one 5-gram.[4] We generalize the minimum number of steps needed for an input text having m tokens as:

$$\text{Min } n_{tp} = m-3 \tag{6.7}$$

From (6.6) and (6.7), it turns out that total number of steps,

$n_{tp} \in [m-3,\ 3m-6]$.

In (6.6), the maximum number of steps, $3m - 6$, also means that the maximum number of tokens possible in a candidate text is $3m - 2$. Thus, an input text having $3m - 2$ tokens can have at most $2m - 2$ errors to be handled and m correct words, assuming $m \geq 3$ (the second assumption on page 6 of Chapter 1). For example, an input text having seven tokens can have at most four erroneous tokens.

[3]We call a step successful if it generates at least one 5-gram. Thus, if we try to generate some 5-grams in step 1 and if we fail to generate any, then the number of step, n_{tp} is 0, though we do some processing for step 1.

[4]If we omit the assumption that each previous step for each extra token generates at least one 5-gram then to determine the minimum number of steps needed is very straight forward which is 0.

Table 6.2: List of All Possible 5-gram Rules in Step 2 to Step $3m-6$

Rule #	5-gram Rule	Generated from (Operation Name)	Case Number
1	$- \; - \; - \; w_i \; w_{i+1}$	No Operation	1: if the last word in step 1 is in T
2	$- \; - \; - \; w_i \; I_j$	Single Insert	
3	$- \; - \; - \; w_i \; w_{i+2}$	Single Delete	
4	$- \; - \; - \; w_i \; w_{i+3}$	Double Delete	
5	$- \; - \; - \; w_i \; R_{i+1}$	Single Replace	
6	$- \; - \; w_i \; I_j \; w_{i+1}$	No Operation	2: if the second last word in step 1 is in T and the last word is either an inserted or a replaced word
7	$- \; - \; w_i \; I_j \; I_{j+1}$	Single Insert	
8	$- \; - \; w_i \; I_j \; w_{i+2}$	Single Delete	
9	$- \; - \; w_i \; I_j \; R_{i+1}$	Single Replace	
10	$- \; - \; w_i \; R_{i+1} \; w_{i+2}$	No operation	
11	$- \; - \; w_i \; R_{i+1} \; I_j$	Single Insert	
12	$- \; - \; w_i \; R_{i+1} \; w_{i+3}$	Single Delete	
13	$- \; - \; w_i \; R_{i+1} \; R_{i+2}$	Single Replace	
14	$- \; w_i \; I_j \; I_{j+1} \; w_{i+1}$	No Operation	3: if the third last word in step 1 is in T and the last two words are either inserted or replaced words or both
15	$- \; w_i \; R_{i+1} \; R_{i+2} \; w_{i+3}$	No Operation	
16	$- \; w_i \; I_j \; R_{i+1} \; w_{i+2}$	No operation	
17	$- \; w_i \; R_{i+1} \; I_j \; w_{i+2}$	No Operation	

5-gram Rules used in Step 2 to Step $3m-6$

Table 6.2 lists all possible 5-gram rules generated from the said operations and assumptions for step 2 to step $3m-6$. We use step 2 (i.e., the next step) only if step 1 (i.e., the previous step) generates at least one 5-gram from 5-gram rules listed in Table 6.1. Similarly, we use step 3 (i.e., the next step) only if step 2 (i.e., the previous step) generates at least one 5-gram from the 5-gram rules listed in Table 6.2, and so on. We describe how we use the next step using step 2 as the next step and step 1 as the previous step. This is true for all other steps. At the beginning of step 2, we first delete all duplicate 5-grams generated in step 1, as different 5-gram rules in step 1 might generate some unique 5-grams. In Table 6.2, '$-$' means that it might be any word that is in T or an inserted word (an instance of I's) or a replaced word (an instance of R's) in the previous step. To give a specific example of how we list the 5-gram rules in Table 6.2, consider

that rule #2 ($w_1\ I_1\ w_2\ w_3\ w_4$) in Table 6.1 generates at least one 5-gram in step 1. We take the last four words of this 5-gram (i.e., $I_1\ w_2\ w_3\ w_4$) and add the next word from T (in this case w_5) that is not used in the rule that the 5-gram is generated from in order to form a new rule in step 2 (which is $I_1\ w_2\ w_3\ w_4\ w_5$). The general form of this rule ($-\ -\ -\ w_i\ w_{i+1}$) is listed as rule #1 in Table 6.2. Note that in step 1, I_1 of rule #2 in Table 6.1 acts similarly to a variable, but in step 2 we use only a single instance of I_1, which acts similarly to a constant.

We categorize all the 5-grams generated in step 1 (i.e., the previous step) into three different cases. Case 1 groups each 5-gram in step 1 having its last word in T, and the status of the remaining four words does not matter. Case 2 groups each 5-gram in step 1 having its second last word in T, the last word not in T, and the status of the remaining three words does not matter. Case 3 groups each 5-gram in step 1 having its third last word in T, the last two words not in T, and the status of the remaining two words does not matter. For each 5-gram generated in step 1 and grouped in case 1, we can form five different 5-gram rules in step 2 (rules #1 to 5 in Table 6.2). For each 5-gram generated in step 1 and grouped in case 2, we can form eight different 5-gram rules in step 2 (rules #6 to 13 in Table 6.2), and for each 5-gram generated in step 1 and grouped in case 3, we can form four different 5-gram rules in step 2 (rules #14 to 17 in Table 6.2).

We stop when we fail to generate any 5-gram in the next step from all the 5-gram rules of the previous step.

The n-grams in the Web 1T data set are sorted in lexicographically ascending order, and split into files of at most 10,000,000 entries. Finding which n-gram file to search depends on the number of n-gram files. All the 5-grams are in 118 files. Thus, to find the exact file (in the worst case) requires $\log_2 118$, or 7 attempts per 5-gram search. The number of 5-grams in each 5-gram file are 10^7, which leads to $\log_2 10^7$, or 23 attempts per 5-gram search.

Determining the Limit of Candidate Texts

Before going into how to form candidate texts, we try to answer the question of what the limit of the number of candidate texts is. There might be a case when no 5-gram is generated in step 1; this means that the minimum $\tilde{n}$ possible is 0. Table 6.1 shows that there are 15 5-gram rules (rules without any I's or R's or both) in step 1 that generate at most one 5-gram per 5-gram rule. It turns out that the remaining 5-gram rules can generate at most $\bar{n}$ 5-grams per 5-gram rule. Thus, the maximum number of candidate texts, $\tilde{n}$, that can be generated having only a single step (i.e., $n_{tp} = 1$) is:

$$\begin{aligned}\text{Max } \tilde{n} = &(\text{number of 5-gram rules in a step} - \\ &\text{number of 5-gram rules in the same step without any } I\text{'s or } R\text{'s or both}) \times \bar{n} + \\ &\text{number of 5-gram rules in the same step without any } I\text{'s or } R\text{'s or both} \end{aligned} \tag{6.8}$$

$$= (75 - 15) \times \bar{n} + 15$$

$$= 60\bar{n} + 15 \tag{6.9}$$

At most $2\bar{n} + 3$ 5-grams[5] (rules #1 to 5 in Table 6.2) can be generated in step 2 from a single 5-gram generated in step 1 having the last word is in T. There may be at most 54 such 5-grams in step 1. At most $2\bar{n} + 2$ 5-grams[6] (rules #6 to 13 in Table 6.2) can be generated in step 2 from a single 5-gram generated in step 1 having the second last word in T and the last word being either an inserted or a replaced word. There may be

[5]Using equation 6.8 for step 2 with case number 1 (Table 6.2), where the number of 5-gram rules is 5 and the number of 5-gram rules without any I's or R's at the last word position is 3.

[6]Using equation 6.8 for step 2 with case number 2 (Table 6.2), where the number of 5-gram rules is 8 and the number of 5-gram rules without any I's or R's at the last word position is 4. Among these 8 5-gram rules, the first 4 5-gram rules have an inserted word at the second last word position, and the last 4 5-gram rules have a replaced word at the second last word position. But, at any specific instance, we might have either an inserted word or a replaced word at the second last word position, not the both. Thus, we should consider the number of 5-gram rules as 4 instead of 8 and the number of 5-gram rules without any I's or R's at the last word position as 2 instead of 4.

at most 17 such 5-grams in step 1. Again, at most a single 5-gram (rules #14 to 17 in Table 6.2) can be generated in step 2 from a single 5-gram generated in step 1 having the third last word in T and the last two words being either inserted or replaced words or a combination of both. There may be at most 4 such 5-grams in step 1. Thus, the maximum number of candidate texts, $\tilde{n}$, that can be generated having two steps (i.e., $n_{tp} = 2$) is:

$$\text{Max } \tilde{n} = 54(2\bar{n} + 3) + 17(2\bar{n} + 2) + 4 \tag{6.10}$$

We generalize the maximum number of candidate texts, $\tilde{n}$, that can be generated for different values of n_{tp} as:

$$\text{Max } \tilde{n} \approx \begin{cases} 60\bar{n} + 15 & \text{if step=1} \\ 54 \times 3^0(2\bar{n} + 3) + 17(2\bar{n} + 2) + 4 & \text{if step=2} \\ 54 \times 3^1(2\bar{n} + 3) + 108 \times 3^0\bar{n}(2\bar{n} + 2) + 34\bar{n} & \text{if step=3} \\ 54 \times 3^2(2\bar{n} + 3) + 108 \times 3^1\bar{n}(2\bar{n} + 2) + 216 \times 3^0\bar{n}^2 & \text{if step=4} \\ 54 \times 3^3(2\bar{n} + 3) + 108 \times 3^2\bar{n}(2\bar{n} + 2) + 216 \times 3^1\bar{n}^2 & \text{if step=5} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ 54 \times 3^{n_{tp}-2}(2\bar{n} + 3) + 108 \times 3^{n_{tp}-3}\bar{n}(2\bar{n} + 2) + 216 \times 3^{n_{tp}-4}\bar{n}^2 & \text{if step=}n_{tp} \end{cases} \tag{6.11}$$

Simplifying (6.11):

$$\text{Max } \tilde{n} \approx \begin{cases} 60\bar{n} + 15 & \text{if } n_{tp} = 1 \\ 142\bar{n} + 200 & \text{if } n_{tp} = 2 \\ 216\bar{n}^2 + 574\bar{n} + 486 & \text{if } n_{tp} = 3 \\ 3^{n_{tp}-4}(864\bar{n}^2 + 1620\bar{n} + 1458) & \text{if } n_{tp} \geq 4 \end{cases} \tag{6.12}$$

Equation (6.12) also represents the maximum number of 5-grams that can be generated in each step. For example, $60\bar{n} + 15$ is the maximum number of 5-grams that can be generated in step 1, $142\bar{n} + 200$ is that of step 2 and $3^{n_{tp}-4}(864\bar{n}^2 + 1620\bar{n} + 1458)$ is that of step n_{tp}. Theoretically, Max $\tilde{n}$ seems a large number but practically $\tilde{n}$ is much smaller than Max $\tilde{n}$. This is because not all theoretically possible 5-grams are in the Google Web 1T 5-grams data set, and because fewer 5-grams generated in any step have an effect in all the subsequent steps.

Forming Candidate Texts

Algorithm 2 describes how a list of candidate texts can be formed from the list of 5-grams in each step. That is, the output of Algorithm 2 is $\{T_1, T_2, \ldots T_i \ldots, T_{\tilde{n}}\}$. The algorithm works as follows: Taking the last four words of each 5-gram in step 1, it tries to match with the first four words of each 5-gram in step 2. If it matches then concatenating the last word of the matched 5-gram in step 2 with the matched 5-gram in step 1 generates a temporary candidate text for further processing, to lead to an ultimate candidate text. If a 5-gram in step 1 does not match with at least a single 5-gram in step 2 then the 5-gram in step 1 is a candidate text. One 5-gram in step 1 can match with several 5-grams in step 2, thus generating several temporary candidate texts. After that, we take the last four words of each of the temporary candidate texts and try to match with the

Algorithm 2: forming candidate texts

```
   input  : n_tp, list of 5-grams in each step  /* n_tp is total number of steps */
   output: candidate_list                       /* list of candidate text */
 1 candidate_list ← NULL
 2 for each 5-gram of step 1 do
 3     k ← 1
 4     candidate_text[k] ← 5-gram of step 1
 5     for i ← 2 to n_tp do
 6         j ← 1
 7         for each k do
 8             for each 5-gram of step i do
 9                 next_5-gram← 5-gram of step i
10                 temp_candidate_text[j] ← candidate_text[k]
11                 str_1 ← last four words of temp_candidate_text[j]
12                 str_2 ← first four words of  next_5-gram
13                 if str_1 = str_2 then
14                     temp_candidate_text[j] ← temp_candidate_text[j] . last word of
                       next_5-gram        /* '.'  is used to concatenate */
15                 end
16                 increment j
17             end
18         end
19         decrement j
20         for each j do
21             candidate_text[j] ← temp_candidate_text[j]
22         end
23         k ← j
24     end
25     for each k do
26         candidate_list ← candidate_list + candidate_text[k]
27     end
28 end
```

first four words of each of the 5-grams in step 3. If any of the temporary candidate text does not match with at least one of the 5-grams in step 3, this means that the temporary candidate text is a candidate text. Again, if it matches, then concatenating the last word of the matched 5-gram in step 3 with the matched temporary candidate text generates a temporary candidate text that needs to be further processed in the subsequent steps, following the same procedure. We continue this process till we cover all the steps.

6.2.4 Sorting Candidate Texts

It turns out from 6.2.3 that, if the input text is T, then the total $\tilde{n}$ candidate texts are $\{T_1, T_2, \ldots T_i \ldots, T_{\tilde{n}}\}$. Now, the task is to sort the candidate texts based on their chance to lead to a correction. First, we determine the correctness value, S for each candidate text using (6.13), a weighted sum of (6.1), (6.3), (6.4) and (6.5), and then we sort in descending order by the correctness values. In (6.13), it is obvious that $\beta_1+\beta_2+\beta_3+\beta_4 = 1$ to have $S \in (0, 1)$.

$$S(T_i) = \beta_1 S_2(T_i, T) + \beta_2 S_3(T_i, T) + \beta_3 S_4(T_i, T) + \beta_4 S_5(T_i) \tag{6.13}$$

By assuming that we try to preserve the semantic meaning of the input text as much as possible (the weak assumption on page 6), we are intentionally trying to keep the candidate texts and the input text as close (both semantically and syntactically) as possible. Thus, we set much preference on the similarity of common words and non-common words between input text and each candidate text. Though we set low preference on the normalized frequency value of each candidate text, it is one of the most crucial parts of the method, that helps to identify and correct the error.

If we only rely on the normalized frequency value of each candidate text, then we have

to deal with an increasing number of *false positives*: the method detects an input text as incorrect, while, in reality, it is not. On the contrary, if we only rely on the similarity of common words, non-common words, and so on, between input text and each candidate text, then we have to deal with an increasing number of *false negatives*: the method detects an input text as correct, while, in reality, it is not.

6.2.5 Discussion on Assumptions

The first assumption means that we consider the first token as a word without any error. Though, our method could have worked for the first word too. We did not do it here due to efficiency reasons. Google n-grams are sorted based on the first word, then the second word, and so on. Based on this sorting, all Google 5-grams, 4-grams, 3-grams and 2-grams are stored in 117, 131, 97 and 31 different files, respectively. For example, all the 117 Google 5-gram files could have been needed to access a single word instead of accessing just one 5-gram file, as we do for any other words. This is because when the first word needs to be corrected, it might be in any file among those 117 5-gram files.

The second assumption ensures that we could use at least some 5-gram rules in step 1 (Table 6.1 on page 112). If we look at all the 75 5-gram rules in Table 6.1, it is clear that we need at least three words to form a 5-gram rule (rules # 6, 7, 8, 9, 10, 11, 21, 52, 53, 59, and 62 in Table 6.1). All the 5-gram rules in different steps in Chapter 6 are based on these three assumptions. The number of errors that can be corrected in a single text is almost double of the number of correct words that the text has; this is because of these strict assumptions. It goes without saying that related work has even stricter assumptions.

Table 6.3: List of all possible 5-grams generated in step 1

No.	5-gram	Frequency	Generated from Step 1's Rule#
P1-1	he was very good at	3542	56
P1-2	he was a very good	7447	60
P1-3	he was very good to	612	18
P1-4	he is very very good	246	54
P1-5	he thought was very good	50	57
P1-6	he was very good in	1050	56
P1-7	he was very very good	240	54
P1-8	he said was very good	130	57
P1-9	he was not very good	3099	60

Table 6.4: List of all possible 5-grams generated in step 2

No.	5-gram	Frequency	Generated from Step 2's Rule#
P2-1	was not very good idea	59	5
P2-2	was very good to eat	61	9
P2-3	was very very good and	388	2
P2-4	was very good to hear	509	9
P2-5	was very good to me	2931	7
P2-6	was very good in the	2423	7
P2-7	is very very good item	197	5
P2-8	was very good to us	15049	7
P2-9	was a very good idea	6535	2
P2-10	was a very good teacher	1267	3
P2-11	was a very good teacher	1267	5
P2-12	was a very good year	13680	2
P2-13	was very very good to	122	5
P2-14	thought was very good value	47	5
P2-15	was very very good all	471	2
P2-16	thought was very good for	67	2
P2-17	was very good at a	98	9
P2-18	is very very good the	54	5
P2-19	was very good in this	719	7
P2-20	thought was very good and	133	2
P2-21	was very good at it	3235	7
P2-22	was not very good the	223	5
P2-23	was a very good team	760	5
P2-24	is very very good and	1010	2
P2-25	was very good at the	1314	7
P2-26	was very good at teaching	163	9
P2-27	is very very good search	1497	2
P2-28	was very good in a	436	9
P2-29	was not very good and	3909	2
P2-30	was not very good at	14088	2

Table 6.5: List of all possible 5-grams generated in step 3

No.	5-gram	Frequency	Generated from step 3's rule#
P3-1	very very good all delivered	464	7
P3-2	very very good search engine	1499	7
P3-3	a very good idea of	14363	7
P3-4	not very good and the	1735	7
P3-5	was very good and the	11076	7
P3-6	a very good year for	12210	7
P3-7	not very good and a	188	9
P3-8	not very good idea to	62	7
P3-9	was very good value at	157	9
P3-10	was very good for me	1678	7
P3-11	was very good for the	2623	7
P3-12	a very good year in	1833	7
P3-13	not very good at all	8368	7
P3-14	not very good at it	20624	7
P3-15	not very good at a	134	9
P3-16	was very good and ate	41	9
P3-17	was very good value and	330	7
P3-18	not very good idea since	43	9
P3-19	was very good and I	4117	7
P3-20	a very good year ahead	110	9
P3-21	not very good idea which	53	7
P3-22	not very good and I	1478	7
P3-23	very very good to me	1322	7
P3-24	a very good idea the	40	9
P3-25	not very good the first	96	7
P3-26	not very good at teaching	230	9
P3-27	very very good and I	267	7
P3-28	a very good idea to	26071	7
P3-29	very very good and it	84	9
P3-30	was very good and a	707	9
P3-31	very very good to be	53	9
P3-32	was very good value for	2366	7
P3-33	was very good for a	1225	9
P3-34	a very good year to	326	9
P3-35	very very good and the	216	7
P3-36	very very good to us	62	7

Table 6.6: List of all candidate texts generated from step 1, 2 and 3

No.	Candidate text	Sum of frequency	Generated from (5-gram Numbers)
T_1	he was very good at a	3640	P1-1, P2-17
T_2	he was very good at it	6777	P1-1, P2-21
T_3	he was very good at the	4856	P1-1, P2-25
T_4	he was very good at teaching	3705	P1-1, P2-26
T_5	he was a very good idea of	28345	P1-2, P2-9, P3-3
T_6	he was a very good idea the	14022	P1-2, P2-9, P3-24
T_7	he was a very good idea to	40053	P1-2, P2-9, P3-28
T_8	he was a very good teacher	8714	P1-2, P2-10
T_9	he was a very good year for	33337	P1-2, P2-12, P3-6
T_{10}	he was a very good year in	22960	P1-2, P2-12, P3-12
T_{11}	he was a very good year ahead	21237	P1-2, P2-12, P3-20
T_{12}	he was a very good year to	21453	P1-2, P2-12, P3-34
T_{13}	he was very good to eat	673	P1-3, P2-2
T_{14}	he was very good to hear	1121	P1-3, P2-4
T_{15}	he was very good to me	3543	P1-3, P2-5
T_{16}	he was very good to us	15661	P1-3, P2-8
T_{17}	he is very very good item	443	P1-4, P2-7
T_{18}	he is very very good the	300	P1-4, P2-18
T_{19}	he is very very good and I	1523	P1-4, P2-24, P3-27
T_{20}	he is very very good and it	1340	P1-4, P2-24, P3-29
T_{21}	he is very very good and the	1472	P1-4, P2-24, P3-35
T_{22}	he is very very good search engine	3242	P1-4, P2-27, P3-2
T_{23}	he thought was very good value at	254	P1-5, P2-14, P3-9
T_{24}	he thought was very good value and	427	P1-5, P2-14, P3-17
T_{25}	he thought was very good value for	2463	P1-5, P2-14, P3-32
T_{26}	he thought was very good for me	1795	P1-5, P2-16, P3-10
T_{27}	he thought was very good for the	2740	P1-5, P2-16, P3-11
T_{28}	he thought was very good for a	1342	P1-5, P2-16, P3-33
T_{29}	he thought was very good and the	11259	P1-5, P2-20, P3-5
T_{30}	he thought was very good and ate	224	P1-5, P2-20, P3-16
T_{31}	he thought was very good and I	4300	P1-5, P2-20, P3-19
T_{32}	he thought was very good and a	890	P1-5, P2-20, P3-30
T_{33}	he was very good in the	3473	P1-6, P2-6
T_{34}	he was very good in this	1769	P1-6, P2-19
T_{35}	he was very good in a	1486	P1-6, P2-28
T_{36}	he was very very good and I	895	P1-7, P2-3, P3-27
T_{37}	he was very very good and it	712	P1-7, P2-3, P3-29
T_{38}	he was very very good and the	844	P1-7, P2-3, P3-35
T_{39}	he was very very good to me	1684	P1-7, P2-13, P3-23
T_{40}	he was very very good to be	415	P1-7, P2-13, P3-31
T_{41}	he was very very good to us	424	P1-7, P2-13, P3-36
T_{42}	he was very very good all delivered	1175	P1-7, P2-15, P3-1
T_{43}	he said was very good	130	P1-8
T_{44}	he was not very good idea to	3220	P1-9, P2-1, P3-8
T_{45}	he was not very good idea since	3201	P1-9, P2-1, P3-18
T_{46}	he was not very good idea which	3211	P1-9, P2-1, P3-21
T_{47}	he was not very good the first	3418	P1-9, P2-22, P3-25
T_{48}	he was not very good and the	8743	P1-9, P2-29, P3-4
T_{49}	he was not very good and a	7196	P1-9, P2-29, P3-7
T_{50}	he was not very good and I	8486	P1-9, P2-29, P3-22
T_{51}	he was not very good at all	25555	P1-9, P2-30, P3-13
T_{52}	he was not very good at it	37811	P1-9, P2-30, P3-14
T_{53}	he was not very good at a	17321	P1-9, P2-30, P3-15
T_{54}	he was not very good at teaching	17417	P1-9, P2-30, P3-26

Table 6.7: List of all candidate texts sorted by S ($\beta 1 = 0.5$, $\beta 2 = 0$, $\beta 3 = 0.4$ and $\beta 4 = 0.1$)

No.	Candidate text	S	S_2	S_4	S_5
T_8	he was a very good teacher	0.349	0.444	0.222	0.387
T_7	he was a very good idea to	0.241	0.214	0.128	0.824
T_{16}	he was very good to us	0.237	0.250	0.106	0.696
T_9	he was a very good year for	0.232	0.214	0.142	0.686
T_4	he was very good at teaching	0.223	0.250	0.204	0.164
T_5	he was a very good idea of	0.217	0.214	0.128	0.583
T_{52}	he was not very good at it	0.216	0.214	0.078	0.778
T_{10}	he was a very good year in	0.211	0.214	0.142	0.472
T_{12}	he was a very good year to	0.208	0.214	0.142	0.441
T_{11}	he was a very good year ahead	0.207	0.214	0.142	0.437
T_1	he was very good at a	0.205	0.250	0.160	0.161
T_{54}	he was not very good at teaching	0.204	0.214	0.153	0.358
T_3	he was very good at the	0.199	0.250	0.132	0.215
T_6	he was a very good idea the	0.197	0.214	0.152	0.288
T_2	he was very good at it	0.196	0.250	0.104	0.301
T_{35}	he was very good in a	0.190	0.250	0.148	0.066
T_{53}	he was not very good at a	0.190	0.214	0.120	0.356
T_{13}	he was very good to eat	0.190	0.250	0.156	0.029
T_{51}	he was not very good at all	0.189	0.214	0.075	0.526
T_{33}	he was very good in the	0.188	0.250	0.120	0.154
T_{15}	he was very good to me	0.188	0.250	0.118	0.157
T_{14}	he was very good to hear	0.185	0.250	0.138	0.049
T_{43}	he said was very good	0.183	0.300	0.078	0.017
T_{17}	he is very very good item	0.175	0.250	0.121	0.019
T_{18}	he is very very good the	0.174	0.250	0.121	0.013
T_{29}	he thought was very good and the	0.169	0.214	0.097	0.231
T_{49}	he was not very good and a	0.168	0.214	0.117	0.148
T_{34}	he was very good in this	0.167	0.250	0.085	0.078
T_{46}	he was not very good idea which	0.163	0.214	0.125	0.066
T_{48}	he was not very good and the	0.163	0.214	0.096	0.180
T_{44}	he was not very good idea to	0.163	0.214	0.124	0.066
T_{45}	he was not very good idea since	0.161	0.214	0.120	0.065
T_{28}	he thought was very good for a	0.157	0.214	0.120	0.027
T_{32}	he thought was very good and a	0.156	0.214	0.118	0.018
T_{47}	he was not very good the first	0.152	0.214	0.096	0.070
T_{27}	he thought was very good for the	0.152	0.214	0.099	0.056
T_{30}	he thought was very good and ate	0.152	0.214	0.111	0.004
T_{38}	he was very very good and the	0.152	0.214	0.108	0.017
T_{50}	he was not very good and I	0.151	0.214	0.068	0.174
T_{39}	he was very very good to me	0.149	0.214	0.097	0.034
T_{41}	he was very very good to us	0.147	0.214	0.097	0.008
T_{40}	he was very very good to be	0.147	0.214	0.097	0.008
T_{21}	he is very very good and the	0.146	0.214	0.090	0.030
T_{31}	he thought was very good and I	0.144	0.214	0.070	0.088
T_{37}	he was very very good and it	0.143	0.214	0.087	0.014
T_{22}	he is very very good search engine	0.142	0.214	0.071	0.066
T_{26}	he thought was very good for me	0.142	0.214	0.078	0.036
T_{42}	he was very very good all delivered	0.141	0.214	0.080	0.024
T_{36}	he was very very good and I	0.141	0.214	0.080	0.018
T_{23}	he thought was very good value at	0.139	0.214	0.078	0.005
T_{25}	he thought was very good value for	0.138	0.214	0.066	0.050
T_{20}	he is very very good and it	0.137	0.214	0.069	0.027
T_{24}	he thought was very good value and	0.136	0.214	0.071	0.008
T_{19}	he is very very good and I	0.135	0.214	0.062	0.031

6.3 A Walk-Through Example

To see how the proposed method works, it is useful to use an example with a relatively small number of tokens m. Consider an input text, T='*he wss very good tea teacher*' with six tokens. This example covers a *Single Replace*, *Single Insert* and a *Single Delete*.

We set the value of how many candidate 5-grams generated from each 5-gram rule we keep for further processing, $\bar{n}$, as 2. Table 6.3 lists all possible 5-grams generated in step 1 using 5-gram rules listed in Table 6.1. $Pi{-}j$ in Table 6.3, 6.4 and 6.5 stands for 5-gram number j generated in step i. Step 1 generates nine 5-grams; though from (6.12), it turns out that it could have generated at most 135 5-grams. Table 6.4 shows that step 2 generates 30 5-grams, whereas theoretically it could have been 484. Similarly, Table 6.5 shows that step 3 generates 36 5-grams, and in theory, it could have been 2498.

For this example, we have three steps and from (6.7) and (6.6) it is obvious that $n_{tp} \in [3, 12]$. Again, as this example has three steps means that using $n_{tp} = 3$ in (6.12) it could have theoretically at most 2498 candidate texts, though practically it has only 54. The reason behind this massive difference between theoretical and practical number is two-fold. First, not all theoretically possible 5-grams are in the Google Web 1T 5-grams data set. Second, fewer 5-grams generated in any step have an effect in all the subsequent steps.

In Table 6.4, P2-10 and P2-11 are same generated from different rules in step 2. As said before, we delete all duplicate 5-grams generated in any step using different 5-gram rules. Table 6.6 lists all the candidate texts generated using Algorithm 2. It also lists the sum up of the frequencies of the 5-grams the candidate text is generated from as well as the 5-gram numbers in different steps that are used to generate the candidate text.

Finally, Table 6.7 lists all the candidate texts sorted in descending order by the value of S using (6.13). It also lists the value of S_2, S_4 and S_5 using (6.1), (6.4) and (6.5) respectively. In (6.13), we heuristically set $\beta_1 = 0.5$, $\beta_3 = 0.4$ and $\beta_4 = 0.1$ to ensure

that we assign more weight to common word similarity and then to non-common word similarity and less weight to the 5-gram probability value. The intuition of assigning more weight to S_2 and S_4 is that it helps to provide higher ranking to the candidate texts that are close to the given input text.

In Table 6.7, the candidate text '*he was very very good to me*' appears a long way beneath the candidate text '*he was very good idea to*', though the first one has more semantic meaning than the later one. The reason is that our target is to correct the given input text keeping it as intact as possible, not to make the candidate text more meaningful sacrificing the closeness towards the input text.

Assuming an imaginary candidate text, which is '*he wss very good tea teacher*', implies that for this candidate text our method says that there is no error, but actually there is. Therefore, the imaginary candidate text, '*he wss very good tea teacher*', would be a *false negative* from the error detection point of view. The frequency value function (S_5) generates a 0 value for this candidate. This is why we call this function the *false negative decreasing* function. Again, for this candidate text, the *false positive decreasing* functions will generate maximum possible values.

On the other hand, that all the candidate texts are different from the input text implies that based on each of these candidate texts, our method says that there are errors, and actually there are. Therefore, these candidate texts are called *true positives* from the error detection point of view. Only one of these candidate texts is the target solution. Therefore, that target candidate text is called *true positive* from the error correction point of view. The remaining of the candidate texts are called *false positives* from the error correction point of view. That is, among all the candidate texts, only one is the *true positive* and the remaining are the *false positives* from the error correction point of view. We sort them and find out the target one from all these candidate texts. This is why we call all the similarity functions (S_2,S_3, and S_4) as *false positive decreasing*

functions.

6.4 Evaluation and Experimental Results

6.4.1 Evaluation on WSJ Corpus

Because of the lack of a publicly available data set containing multiple errors in short texts[7], we generate a new evaluation data set[8], utilizing the 1987-89 *Wall Street Journal* corpus. It is assumed that this data contains no errors. We select 50 short texts from this corpus and artificially introduce some errors, so that it requires to perform some combinations of insert, and/or delete, and/or replace operation to get back to the correct texts. To generate the incorrect texts, we artificially insert prepositions and articles, delete articles, prepositions, auxiliary verbs, and replace prepositions with other prepositions, singular noun with plural noun (e.g., *spokesman* with *spokesmen*), articles with other articles, real words with real-word spelling errors (e.g., *year* with *tear*), real words with spelling errors (e.g., *there* with *ther*), verb forms with other verb forms (e.g., *shows* with *show*), auxiliary verbs with other auxiliary verbs (e.g., *are* with *is*). To generate real-word spelling errors (which are in fact semantic errors) and spelling errors, we use the same procedure as Hirst and Budanitsky (2005). The average number of tokens in a correct text and an incorrect text are 7.48 and 6.14, respectively. The average number of corrections required per text is 2.02. We keep some texts without inserting any error, to test the robustness of the system (we got only a single false positive). This decreases the number of errors per text, though one fifth of the data set has three errors per text.

The performance is measured using *Recall* (R), *Precision* (P), *F-measure* and *Accuracy* (Acc). We asked two human judges, both native speakers of English and graduate students in Natural Language Processing, to correct those 50 texts. It is obvious that a

[7]We assume a text containing not more than ten tokens to be a short text.
[8]We will make the data set publicly available.

text can be corrected in many different ways. We consider only the text which is in the WSJ corpus as the solution, in order to make the evaluation process easier. The agreement between the two judges is low (the detection agreement is 54.8% and the correction agreement is 54.4%). Among the five real-word spelling errors, our method and one of the judges corrected all the five errors, whereas, the other judge corrected four errors. Table 6.8 shows three examples of test texts.

Table 6.8: Some examples.

Ex.1	Incorrect	It is quiet time year
	Correct	It is a quiet time of the year
	Judge 1	It is a quiet time of year
	Judge 2	It is quiet this time of year
	Our Method	It is a quiet time of the year
Ex.2	Incorrect	for the first mime try settle
	Correct	for the first time to try to settle
	Judge 1	for the first time try to settle
	Judge 2	for the first time I try to settle
	Our Method	for the first time try to
Ex.3	Incorrect	The agency also plan set a
	Correct	The agency also plans to set up a
	Judge 1	The agency also plan to set a
	Judge 2	The agency also plans to set a
	Our Method	The agency also plans to set up a

The results in Figure 6.2 and Figure 6.3 show that our method gives better *recall* value for both detection and correction, whereas human judges give better *precision* value for both detection and correction. Since a majority of the words in the evaluation data set are correct, the *baseline* is to propose no correction, achieving 72.9% accuracy (Figure 6.4). Taking this *baseline* accuracy as a lower limit and the accuracy achieved by the human judges as an upper limit, we conclude that the automatic method has room for improvement of approximately 5 percentage points.

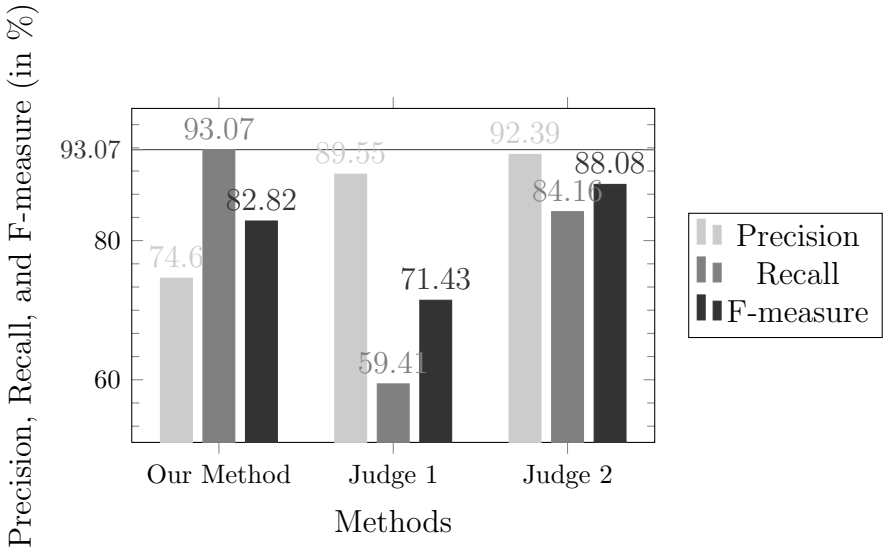

Figure 6.2: Results on the WSJ corpus for detection.

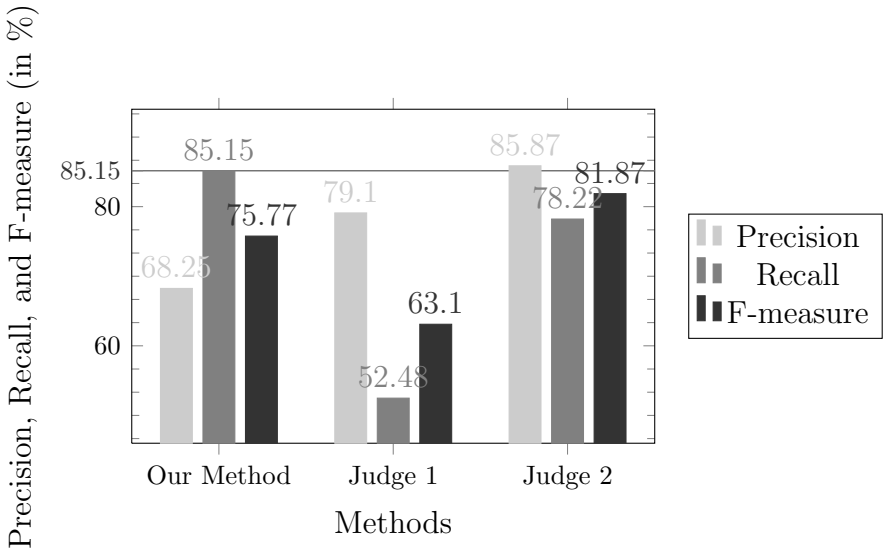

Figure 6.3: Results on the WSJ corpus for correction.

6.4.2 Evaluation on JLE Corpus

We also evaluate the proposed method using the NICT JLE corpus (Izumi et al., 2005; Izumia et al., 2004), to directly compare with (Lee, 2009; Lee and Seneff, 2008). The

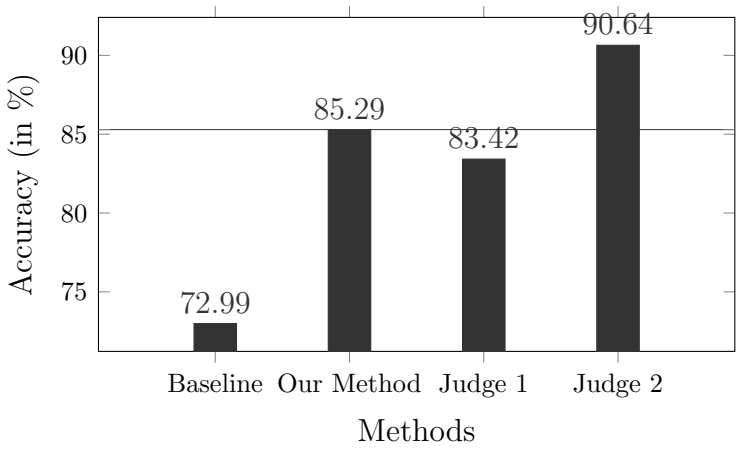

Figure 6.4: Accuracy on the WSJ corpus.

National Institute of Information and Communications Technology (NICT) in Japan assembled the Japanese Learners of English (JLE) corpus, known as NICT JLE corpus, which consists of 2 million words from 1200 Japanese students. These students were interviewed for the Standard Speaking Test (SST), an English-language proficiency test conducted in Japan, and graded on a scale between 1 and 9. The 15-minute interview includes informal chat, picture description, role-playing and story telling. These interviews were transcribed; for 167 of these interviews, grammatical errors in 45 categories have been manually annotated and corrected. For example,

I belong to two baseball <`n_num crr`="*teams*">*team*< /`n_num`>.

where `n_num` is the error label for a noun number error, "team" is the word that should be removed, and `crr` is the word (in this case, "teams") that should be inserted.

There are <`at crr`= "a">< /`at`> *lot of books.*

where `at` is the error label for an article error, there is no word that should be removed, and `crr` is the word (in this case, "a") that should be inserted. Sentence segmentation (Reynar and Ratnaparkhi, 1997) was performed on the interviewee turns in these transcripts, discarding sentences with only one word. Thus, JLE corpus has 15,637 sentences

Table 6.9: Results on the JLE corpus.'—' means that the result is not mentioned in Lee and Seneff (2008).

	Detection			**Correction**			**Acc.**
	R	**P**	**F-measure**	**R**	**P**	**F-measure**	
Lee (S-V)	—	83.93	—	80.92	81.61	—	98.93
Lee (AAC)	—	80.67	—	42.86	68.00	—	98.94
Our (S-V)	98.92	96.84	97.87	97.85	95.79	96.81	98.74
Our (AAC)	97.92	94.00	95.92	95.83	92.00	93.88	97.48

with annotated grammatical errors and their corrections (with over 153,000 words). The three most-frequent error classes are articles, noun number and prepositions, followed by a variety of errors associated with verbs. We generated a test set of 477 sentences for subject-verb (S-V) agreement errors, and another test set of 238 sentences for auxiliary agreement and complementation (AAC) errors by retaining the verb form errors, but correcting all other error types. Lee and Seneff (2008) generated the same number of sentences of each category.

Lee and Seneff (2008) used the *majority baseline*, which is to propose no correction, since the vast majority of verbs were in their correct forms. Thus, Lee and Seneff (2008) achieved a *majority baseline* of 96.95% for S-V agreement and 98.47% for AAC. Based on these numbers, it can be determined that Lee and Seneff (2008) had only 14 or 15 errors in S-V agreement data set and 3 or 4 errors in AAC data set. Our data set has a *majority baseline* of 80.5% for S-V agreement and 79.8% for AAC. It means that we have 93 errors in S-V agreement data set and 48 errors in AAC data set. The small number of errors in their data set is the reason why they get high accuracy even when they have moderate *precision* and *recall*. For example, if their method fails to correct 2 errors out of the 3 errors in S-V agreement data set (i.e., if true positive is 1 and false positives are 2), then their *recall* would be 33.3%, even then their accuracy would be 99.16%. Table 6.9 shows that our method generates consistent *precision*, *recall*, and accuracy.

6.5 Handling Data Sparsity

To handle data sparsity, specially when we do not find enough 5-grams to generate at least a single candidate text, we could use 4-gram rules (Table 6.10 and 6.11) and then tri-gram rules (Table 6.12 and 6.13). We do not generate any bigram rules as these do not fit with the assumptions stated in Section 1.2 on page 6. This is the end of first phase. If we fail to generate at least one candidate text in the first phase, we use the same technique used in Section 3.4.4 on page 44 first with 5-gram rules and then, if needed, with 4-gram and tri-gram rules. This later part is known as second phase. This way the proposed text correction approach is analogous to the two-phased approach described in Chapter 3.

Table 6.10: List of All Possible 4-gram Rules in Step 1

Rule #	4-gram Rule	Generated from (Operation Name)	Rule #	4-gram Rule	Generated from (Operation Name)
			27	$w_1\ I_1\ w_3\ w_4$	Consecutive Single
1	$w_1\ w_2\ w_3\ w_4$	No Operation	28	$w_1\ w_2\ I_1\ w_4$	Insert+Single Delete
2	$w_1\ I_1\ w_2\ w_3$		29	$w_1\ w_3\ R_4\ w_5$	
3	$w_1\ w_2\ I_1\ w_3$		30	$w_1\ w_2\ w_4\ R_5$	Single Delete +
4	$w_1\ w_2\ w_3\ I_1$	Single Insert	31	$w_1\ w_3\ w_4\ R_5$	Single Replace
5	$w_1\ I_1\ w_2\ I_2$	Double Insert	32	$w_1\ R_2\ w_3\ w_5$	Single Replace + Single Delete
6	$w_1\ I_1\ I_2\ w_2$	Consecutive	33	$w_1\ I_1\ w_2\ R_3$	Single Insert +
7	$w_1\ w_2\ I_1\ I_2$	Double Insert			Single Replace
8	$w_1\ R_2\ w_3\ w_4$				
9	$w_1\ w_2\ R_3\ w_4$	Single Replace	34	$w_1\ R_2\ w_3\ I_1$	Single Replace +
10	$w_1\ w_2\ w_3\ R_4$				Single Insert
11	$w_1\ R_2\ w_3\ R_4$	Double Replace	35	$w_1\ I_1\ R_2\ w_3$	Consecutive Single
			36	$w_1\ w_2\ I_1\ R_3$	Insert+Single Replace
12	$w_1\ R_2\ R_3\ w_4$	Consecutive Double	37	$w_1\ R_2\ I_1\ w_3$	Consecutive Single
13	$w_1\ w_2\ R_3\ R_4$	Replace	38	$w_1\ w_2\ R_3\ I_1$	Replace + Single Insert
14	$w_1\ w_3\ w_4\ w_5$		39	$w_1\ R_3\ w_4\ w_5$	Consecutive Single
15	$w_1\ w_2\ w_4\ w_5$	Single Delete	40	$w_1\ w_2\ R_4\ w_5$	Delete + Single
16	$w_1\ w_2\ w_3\ w_5$		41	$w_1\ w_2\ w_3\ R_5$	Replace
17	$w_1\ w_3\ w_5\ w_6$		42	$w_1\ R_2\ w_4\ w_5$	Consecutive Single
18	$w_1\ w_2\ w_4\ w_6$	Double Delete	43	$w_1\ w_2\ R_3\ w_5$	Replace + Single Delete
19	$w_1\ w_3\ w_4\ w_6$				
20	$w_1\ w_4\ w_5\ w_6$	Consecutive			
21	$w_1\ w_2\ w_5\ w_6$	Double			
22	$w_1\ w_2\ w_3\ w_6$	Delete			
23	$w_1\ I_1\ w_2\ w_4$	Single Insert + Single Delete			
24	$w_1\ w_3\ I_1\ w_4$	Single Delete +			
25	$w_1\ w_3\ w_4\ I_1$	Single Insert			
26	$w_1\ w_2\ w_4\ I_1$				

Table 6.11: List of All Possible 4-gram Rules in Step 2 to Step Max(n_{tp})

Rule #	4-gram Rule	Generated from (Operation Name)	Case Number
1	$- \; - \; w_i \; w_{i+1}$	No Operation	
2	$- \; - \; w_i \; I_j$	Single Insert	
3	$- \; - \; w_i \; w_{i+2}$	Single Delete	1: if the last word in step 1 is in T
4	$- \; - \; w_i \; w_{i+3}$	Double Delete	
5	$- \; - \; w_i \; R_{i+1}$	Single Replace	
6	$- \; w_i \; I_j \; w_{i+1}$	No Operation	
7	$- \; w_i \; I_j \; I_{j+1}$	Single Insert	
8	$- \; w_i \; I_j \; w_{i+2}$	Single Delete	
9	$- \; w_i \; I_j \; R_{i+1}$	Single Replace	2: if the second last word in step 1 is in
10	$- \; w_i \; R_{i+1} \; w_{i+2}$	No Operation	T and the last word is either an inserted
11	$- \; w_i \; R_{i+1} \; I_j$	Single Insert	or a replaced word
12	$- \; w_i \; R_{i+1} \; w_{i+3}$	Single Delete	
13	$- \; w_i \; R_{i+1} \; R_{i+2}$	Single Replace	
14	$w_i \; I_j \; I_{j+1} \; w_{i+1}$	No Operation	
15	$w_i \; R_{i+1} \; R_{i+2} \; w_{i+3}$	No Operation	3: if the third last word in step 1 is in T
16	$w_i \; I_j \; R_{i+1} \; w_{i+2}$	No Operation	and the last two words are either inserted
17	$w_i \; R_{i+1} \; I_j \; w_{i+2}$	No Operation	or replaced words or both

Table 6.12: List of All Possible 3-gram Rules in Step 1

Rule #	3-gram Rule	Generated from (Operation Name)	Rule #	3-gram Rule	Generated from (Operation Name)
1	$w_1\ w_2\ w_3$	No Operation	14	$w_1\ I_1\ w_3$	Consecutive Single Insert+Single Delete
2	$w_1\ I_1\ w_2$	Single Insert	15	$w_1\ w_3\ R_4$	Single Delete + Single Replace
3	$w_1\ w_2\ I_1$				
4	$w_1\ I_1\ I_2$	Consecutive Double Insert			Single Insert + Single Replace
5	$w_1\ R_2\ w_3$	Single Replace			Single Replace + Single Insert
6	$w_1\ w_2\ R_3$				
		Double Replace	16	$w_1\ I_1\ R_2$	Consecutive Single Insert+Single Replace
7	$w_1\ R_2\ R_3$	Consecutive Double Replace	17	$w_1\ R_2\ I_1$	Consecutive Single Replace + Single Insert
8	$w_1\ w_3\ w_4$	Single Delete	18	$w_1\ R_3\ w_4$	Consecutive Single
9	$w_1\ w_2\ w_4$		19	$w_1\ w_2\ R_4$	Delete + Single Replace
10	$w_1\ w_3\ w_5$	Double Delete	20	$w_1\ R_2\ w_4$	Consecutive Single Replace + Single Delete
11	$w_1\ w_4\ w_5$	Consecutive			
12	$w_1\ w_2\ w_5$	Double Delete			
		Single Insert + Single Delete			
13	$w_1\ w_3\ I_1$	Single Delete + Single Insert			

Table 6.13: List of All Possible 3-gram Rules in Step 2 to Step Max(n_{tp})

Rule #	3-gram Rule	Generated from (Operation Name)	Case Number
1	$-\ w_i\ w_{i+1}$	No Operation	
2	$-\ w_i\ I_j$	Single Insert	
3	$-\ w_i\ w_{i+2}$	Single Delete	1: if the last word in step 1 is in T
4	$-\ w_i\ w_{i+3}$	Double Delete	
5	$-\ w_i\ R_{i+1}$	Single Replace	
6	$w_i\ I_j\ w_{i+1}$	No Operation	
7	$w_i\ I_j\ I_{j+1}$	Single Insert	
8	$w_i\ I_j\ w_{i+2}$	Single Delete	
9	$w_i\ I_j\ R_{i+1}$	Single Replace	2: if the second last word in step 1 is in
10	$w_i\ R_{i+1}\ w_{i+2}$	No Operation	T and the last word is either an inserted
11	$w_i\ R_{i+1}\ I_j$	Single Insert	or a replaced word
12	$w_i\ R_{i+1}\ w_{i+3}$	Single Delete	
13	$w_i\ R_{i+1}\ R_{i+2}$	Single Replace	
14	$I_j\ I_{j+1}\ w_{i+1}$	No Operation	
15	$R_{i+1}\ R_{i+2}\ w_{i+3}$	No Operation	3: if the last two words are either inserted
16	$I_j\ R_{i+1}\ w_{i+2}$	No Operation	or replaced words or both
17	$R_{i+1}\ I_j\ w_{i+2}$	No Operation	

6.6 Summary

In this chapter, we generate 5-grams from a set of rules in different steps and then merge the 5-grams of different steps that exclusively share at least four tokens, in order to generate candidate texts (because we slide the 5-gram window to the right to traverse the sentence). Then, we use some similarity functions that decrease the normalized weight of the candidate texts which are false positives. Similarly, the frequency value function is used to lessen the normalized weight of the false negative candidate texts.

In Chapters 3, 4, and 5, where we used *replace* operation, our task was to find the best choice (or word) of a near-synonym, or a preposition, or a real-word spelling correction from a given list of candidate choices or words. When we use *replace* operation in Chapter 6, our task is three-fold. First, we determine a list of replaceable candidates (if we are in need to replace a token for correction purpose) which is not given in this case and, second, to generate a list of candidate texts by applying all/any of the remaining two text correction operations (if we are in need to insert word(s) or delete token(s) for correction purpose) and finally, to find the best text from the list of candidate texts. That is, in Chapter 3, 4, and 5, the best matching text is generated by applying only the *replace* operation, whereas, in Chapter 6, all the candidate texts are generated by applying any or some or all of the three text correction operations. This is the reason why in this chapter we do not directly use the same sorting approach that we used in Chapter 3, 4, and 5.

Chapter 7

Conclusions and Future Directions

7.1 Conclusion

The proposed text correction approach, which is unsupervised, can deal with multiple errors of both syntactic and semantic nature. The number of errors that can be corrected in a single text is almost double of the number of correct words that the text has. This large magnitude of error coverage, in terms of number, can have a positive impact on applications such as correcting Optical Character Recognition (OCR) errors, automatically-marking (based on grammar and semantics) subjective examination papers, normalizing short messages, detecting and correcting speech recognition errors, and so on.

7.2 Future Work

It goes without saying that a major drawback of our proposed approach is the dependence on the availability of enough n-grams. The challenge is how to tackle the data sparsity problem, while keeping the approach unsupervised. To tackle the data sparsity problem, one approach might be to generate rules and their frequencies off-line from the Web

1T n-grams and then to use those rules along with the proposed approach. The rules generation process would be unsupervised. To make the rules simple, one approach might be to only replace nouns/proper nouns and adjectives of 5-grams with their tags (e.g., <NOUN> and <ADJ>). To give a simple example, we can generate a rule "*he was a* <NOUN> *of* " and its frequency by tagging parts-of-speech (only noun and adjective) of all the 5-grams in Web 1T that match "*he was a* *.* *of* " where *.* is tagged as <NOUN>. Thus, the frequency of the rule would be the sum of the frequencies of all the 5-grams that match with "*he was a* *.* *of* " where *.* is tagged as <NOUN>.

For long texts, there might be two problems. Due to the data sparsity, there might be a break of steps while generating candidate texts which, in turn, generates incomplete candidate texts with respect to the input text. Again, we show that the maximum number of candidate texts that can be generated for a long input text is theoretically a big number, though practically most of the times this is not the case. This is because the maximum number of candidate texts is a quadratic polynomial function of two arguments, the number of steps and the number of candidate 5-grams generated from each 5-gram rule that we keep for further processing. To solve the first problem, we could use an approach that would treat the incomplete candidate texts generated because of the break of steps as a part of final candidate texts and try to generate another part of the incomplete candidate texts independently and then merge those independent parts of candidate texts to generate the final candidate texts. To solve the second problem, we could intentionally split the long input texts into smaller parts and then treat them independently and finally merge them. The merging procedure would definitely be a crucial step for this kind of approach.

Let us consider another input text: "*This is powerful tea.*" Some of the top candidate texts generated using our proposed approach for this input text are: "*This is a powerful idea*", "*This is a powerful rear*", and "*This is a great tea.*" It is obvious that the first

two candidate texts emphasize the word '*powerful*' and the last one emphasizes the word '*tea*'. The word '*powerful*' does not semantically fit with the word '*tea*'; it means that we need to choose either '*powerful*' or '*tea*' as the focus word. It is hard, even for humans, to choose the correct focus word from this short context. When we have enough context, the question is: how can we choose the focus word? Topic detection methods can be employed for this task.

It is possible to propose or use some other *false positive decreasing* functions and *false negative decreasing* functions. We could even think of using a function that would help choosing the focus word from the context.

Given that we have different functions that reduce the number of *false positives* and *false negatives*, the question is: how can we choose the best weight factors for each of the functions keeping the approach unsupervised? If desired, for specific tasks regarding the correction of specific kinds of errors, it is possible to choose the weights based on some training data; in this case the approach will no longer be unsupervised.

Bibliography

Andersen, O. E. (2007). Grammatical error detection using corpora and supervised learning. In Nurmi, V. and Sustretov, D., editors, *Proceedings of the 12th Student Session of the European Summer School for Logic, Language and Information*.

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

Atwell, E. and Elliot, S. (1987). Dealing with ill-formed english text. In Garside, R., Sampson, G., and Leech, G., editors, *The computational analysis of English: a corpus-based approach*, London. Longman.

Atwell, E. S. (1987). How to detect grammatical errors in a text without parsing it. In *Proceedings of the third conference on European chapter of the Association for Computational Linguistics*, pages 38–45, Morristown, NJ, USA. Association for Computational Linguistics.

Bergsma, S., Lin, D., and Goebel, R. (2009). Web-scale n-gram models for lexical disambiguation. In *IJCAI*, pages 1507–1512, Pasadena, California.

Bitchener, J., Young, S., and Cameron, D. (2005). The effect of different types of corrective feedback on ESL student writing. *Journal of Second Language Writing*, 14:191–205.

Brants, T. and Franz, A. (2006). Web 1T 5-gram corpus version 1.1. Technical report, Google Research.

Brants, T. and Franz, A. (2009). Web 1T 5-gram, 10 European languages version 1. Technical report, Linguistic Data Consortium, Philadelphia.

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

Brown, P. F., deSouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based *n*-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Burnard, L. (2000). *Reference Guide for the British National Corpus (World Edition)*. www.natcorp.ox.ac.uk/docs/userManual/urg.pdf.

Burstein, J., Chodorow, M., and Leacock, C. (2004). Automated essay evaluation: the criterion online writing service. *AI Mag.*, 25(3):27–36.

Cao, L. (2008). *Data Mining for Business Applications*. Springer.

Cao, L. (2010). Domain-driven data mining: Challenges and prospects. *IEEE Transactions on Knowledge and Data Engineering*, 22:755–769.

Carlson, A. and Fette, I. (2007). Memory-based context-sensitive spelling correction at web scale. In *Proceedings of the IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cincinatti, OH.

Carlson, A. J., Rosen, J., and Roth, D. (2001). Scaling up context-sensitive text correction. In *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*, pages 45–50. AAAI Press.

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318.

Chen, S. F. and Goodman, J. T. (1998). An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University.

Chodorow, M. and Leacock, C. (2000). An unsupervised method for detecting grammatical errors. In *Proceedings of NAACL'00*, pages 140–147.

Chodorow, M., Tetreault, J. R., and Han, N.-R. (2007). Detection of grammatical errors involving prepositions. In *SigSem '07: Proceedings of the Fourth ACL-SIGSEM Workshop on Prepositions*, pages 25–30, Morristown, NJ, USA. Association for Computational Linguistics.

Choueka, Y. and Lusignan, S. (1985). Disambiguation by short contexts. *Computers and the Humanities*, 19:147–158.

Church, K. and Hanks, P. (1991). Word association norms, mutual information and lexicography. *Computational Linguistics*, 16(1):22–29.

Church, K. W. and Gale, W. A. (1991). Probability scoring for spelling correction. *Statistics and Computing*, 1(2):93–103.

Clark, S. and Weir, D. (2002). Class-based probability estimation using a semantic hierarchy. *Computational Linguistics*, 28(2):187–206.

Clarke, C. L. A. and Terra, E. (2003). Frequency estimates for statistical word similarity measures. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pages 165–172, Edmonton, Canada.

Corston-Oliver, S., Gamon, M., and Brockett, C. (2001). A machine learning approach to the automatic evaluation of machine translation. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 148–155, Toulouse, France. Association for Computational Linguistics.

Crystal, D. (1997). *English as a Global Language*. Cambridge University Press.

Curran, J. R., Murphy, T., and Scholz, B. (2007). Minimising semantic drift with mutual exclusion bootstrapping. In *Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 172–180.

Dahl, G., Frassica, A.-M., and Wicentowski, R. (2007). SW-AG: Local context matching for English lexical substitution. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 304–307, Prague, Czech Republic. Association for Computational Linguistics.

Dan, E. B., Flickinger, D., Oepen, S., Walsh, A., and Baldwin, T. (2004). Arboretum: Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL/ICALL Symposium: NLP and Speech Technologies in Advanced Language Learning Systems*, pages 83–86.

Domeij, R., Knutsson, O., Carlberger, J., and Kann, V. (1999). Granska an efficient hybrid system for Swedish grammar checking. In *Proc. 12th Nordic Conference in Computational Linguistics, NoDaLiDa-99*, pages 49–56.

Edmonds, P. (1997). Choosing the word most typical in context using a lexical co-occurrence network. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 507–509, Madrid, Spain.

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

Evert, S. (2010). Google Web 1T 5-grams made easy (but not for the computer). In *Proceedings of the 6th Web as Corpus Workshop (WAC-6)*, Los Angeles, California.

Felice, R. D. (2008). *Automatic error detection in non-native English*. PhD thesis, Oxford University, UK.

Felice, R. D. 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 (Coling 2008)*, pages 169–176, Manchester, UK. Coling 2008 Organizing Committee.

Fellbaum, C., editor (1998). *WordNet: An electronic lexical database*. MIT Press.

Foster, J. and Vogel, C. (2004). Parsing ill-formed text using an error grammar. *Artif. Intell. Rev.*, 21(3-4):269–291.

Fouvry, F. (2003). Constraint relaxation with weighted feature structures. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 23–25, Nancy, France.

Gale, W. A. and Church, K. W. (1990). Estimation procedures for language context: Poor estimates are worse than none. In *Proceedings Computational Statistics*, pages 69–74. Physica-Verlag, Heidelberg.

Gale, W. A., Church, K. W., and Yarowsky, D. (1994). Discrimination decisions for 100,000-dimensional spaces. In *Journal of Operations Research*, pages 429–450. Kluwer Academic Publishers.

Gamon, M., Aue, A., and Smets, M. (2005). Sentence-level mt evaluation without reference translations: Beyond language modeling. In *European Association for Machine Translation (EAMT)*, pages 103–111.

Gamon, M., Gao, J., Brockett, C., Klementiev, A., Dolan, W. B., Belenko, D., and Vanderwende, L. (2008). Using contextual speller techniques and language modeling for ESL error correction. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008)*, pages 449–456.

Giuliano, C., Gliozzo, A., and Strapparava, C. (2007). FBK-irst: Lexical substitution task exploiting domain and syntagmatic coherence. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 145–148, Prague, Czech Republic. Association for Computational Linguistics.

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

Golding, A. R. and Roth, D. (1999). A winnow-based approach to context-sensitive spelling correction. *Machine Learning*, 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*, pages 71–78, Morristown, NJ, USA. Association for Computational Linguistics.

Granger, S., Kraif, O., Ponton, C., Antoniadis, G., and Zampa, V. (2007). Integrating learner corpora and natural language processing: A crucial step towards reconciling technological sophistication and pedagogical effectiveness1. *ReCALL*, 19(3):252–268.

Hassan, S., Csomai, A., Banea, C., Sinha, R., and Mihalcea, R. (2007). UNT: SubFinder: Combining knowledge sources for automatic lexical substitution. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 410–413, Prague, Czech Republic. Association for Computational Linguistics.

Hawker, T. (2007). USYD: WSD and lexical substitution using the Web1T corpus. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 446–453, Prague, Czech Republic. Association for Computational Linguistics.

Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *J. ACM*, 24:664–675.

Hirst, G. (2008). An evaluation of the contextual spelling checker of Microsoft Office Word 2007. http://ftp.cs.toronto.edu/pub/gh/Hirst-2008-Word.pdf.

Hirst, G. and Budanitsky, A. (2005). Correcting real-word spelling errors by restoring lexical cohesion. *Natural Language Engineering*, 11(1):87–111.

Hirst, G. and St-Onge, D. (1998). *WordNet: An electronic lexical database*, chapter Lexical chains as representations of context for the detection and correction of malapropisms, pages 305–332. The MIT Press, Cambridge, MA.

Hockenmaier, J. (2003). *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. PhD thesis, School of Informatics, University of Edinburgh.

Ide, N. and Véronis, J. (1998). Word sense disambiguation: The state of the art. *Computational Linguistics*, 24(1):1–41.

Inkpen, D. (2007). A statistical model for near-synonym choice. *ACM Transactions on Speech and Language Processing*, 4(1):1–17.

Inkpen, D. Z. and Hirst, G. (2003). Near-synonym choice in natural language generation. In *Proceedings of the International Conference RANLP-2003 (Recent Advances in Natural Language Processing)*, pages 204–211, Borovets, Bulgaria.

Ishioka, T. and Kameda, M. (2006). Automated Japanese essay scoring system based on articles written by experts. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 233–240, Morristown, NJ, USA. Association for Computational Linguistics.

Islam, A. and Inkpen, D. (2006). Second order co-occurrence PMI for determining the semantic similarity of words. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 1033–1038, Genoa, Italy.

Islam, A. and Inkpen, D. (2008). Semantic text similarity using corpus-based word similarity and string similarity. *ACM Trans. Knowl. Discov. Data*, 2:10:1–10:25.

Islam, A. and Inkpen, D. (2009a). Managing the Google Web 1T 5-gram data set. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE09)*, pages 1–5, Dalian, China.

Islam, A. and Inkpen, D. (2009b). Real-word spelling correction using Google Web 1T 3-grams. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009*, pages 1241–1249, Singapore. Association for Computational Linguistics.

Islam, A. and Inkpen, D. (2009c). Real-word spelling correction using Google Web 1T n-gram data set. In Cheung, D. W.-L., Song, I.-Y., Chu, W. W., Hu, X., and Lin,

J. J., editors, *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009*, pages 1689–1692, Hong Kong. ACM.

Islam, A. and Inkpen, D. (2009d). Real-word spelling correction using Google Web 1T n-gram with backoff. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE09)*, pages 1–8, Dalian, China.

Islam, A. and Inkpen, D. (2009e). Semantic similarity of short texts. In Nicolov, N., Angelova, G., and Mitkov, R., editors, *Recent Advances in Natural Language Processing V*, Current Issues in Linguistic Theory 309, pages 227–236. John Benjamins Publishing Company. Selected papers from Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP) 2007.

Islam, A. and Inkpen, D. (2010a). Near-synonym choice using a 5-gram language model. *Research in Computing Science*, 46:41–52.

Islam, A. and Inkpen, D. (2010b). An unsupervised approach to preposition error correction. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'10)*, pages 1–4, Beijing.

Islam, A., Inkpen, D., and Kiringa, I. (2008). Applications of corpus-based semantic similarity and word segmentation to database schema matching. *The VLDB Journal*, 17(5):1293–1320.

Izumi, E., Uchimoto, K., and Isahara, H. (2005). Error annotation for corpus of Japanese learner English. In *Proceedings of the Sixth International Workshop on Linguistically Interpreted Corpora (LINC-2005)*, pages 71–80, Jeju Island, Korea.

Izumia, E., Uchimotoa, K., and Isaharaa, H. (2004). SST speech corpus of Japanese

learners' English and automatic detection of learners' errors. *ICAME Journal*, 28:31–48.

Japkowicz, N. and Wiebe, J. M. (1991). A system for translating locative prepositions from English into French. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*, ACL '91, pages 153–160, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kaplan, A. (1950). *An experimental study of ambiguity and context*. Published as Kaplan, Abraham (1955), An experimental study of ambiguity and context, Mechanical Translation, 2(2), 39-46.

Keller, F. and Lapata, M. (2003). Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29(3):459–484.

Klein, M. and Nelson, M. L. (2009). Correlation of term count and document frequency for Google n-grams. In et al., B. M., editor, *Proceedings of the 31st European Conference on Information Retrieval*, volume 5478/2009 of *Lecture Notes in Computer Science*, pages 620–627. Springer Berlin / Heidelberg.

Kondrak, G. (2005). N-gram similarity and distance. In *Proceedings of the 12h International Conference on String Processing and Information Retrieval*, pages 115–126, Buenos Aires, Argentina.

Koutsoudas, A. K. and Korfhage, R. (1956). M.T. and the problem of multiple meaning. *Mechanical Translation*, 2(2):46–51.

Krishnakumaran, S. and Zhu, X. (2007). Hunting elusive metaphors using lexical resources. In *NAACL 2007 Workshop on Computational Approaches to Figurative Language*.

Kukich, K. (1992). Technique for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439.

Lee, J. and Seneff, S. (2008). Correcting misuse of verb forms. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 174–182, Columbus, Ohio, USA. The Association for Computer Linguistics.

Lee, J. S. Y. (2009). *Automatic Correction of Grammatical Errors in Non-native English Text*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

Li, H., Japkowicz, N., and Barriere, C. (2005). English to Chinese translation of prepositions. In Kégl, B. and Lapalme, G., editors, *Advances in Artificial Intelligence*, volume 3501 of *Lecture Notes in Computer Science*, pages 43–54. Springer Berlin / Heidelberg.

Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics*, pages 768–774, Morristown, NJ, USA. Association for Computational Linguistics.

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

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts.

Masterman, M. (1961). Semantic message detection for amchine translation, using an interlingua. In *1961 International Conference on machine Translation of Languages and Applied Language Analysis*. Her Majesty's Stationery Office, London, 1962, pages 437-475.

Matthieu Hermet, A. D. and Szpakowicz, S. (2008). Using the web as a linguistic resource to automatically correct lexico-syntactic errors. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco.

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

McCarthy, D. and Navigli, R. (2009). The English lexical substitution task. *Language Resources and Evaluation*, 43(2):139–159.

McCarthy, D., Venkatapathy, S., and Joshi, A. K. (2007). Detecting compositionality of verb-object combinations using selectional preferences. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 369–379, Prague.

Melamed, I. D. (1999). Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25(1):107–130.

Michaud, L. N. and McCoy, K. F. (2001). Error profiling: Toward a model of English acquisition for deaf learners. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 394–401, Toulouse, France. Association for Computational Linguistics.

Michaud, L. N., McCoy, K. F., and Pennington, C. A. (2000). An intelligent tutoring system for deaf learners of written English. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies*, pages 92–100, New York, NY, USA. ACM.

Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.

Murphy, T. and Curran, J. (2007). Experiments in mutual exclusion bootstrapping. In *Proceedings of the Australasian Language Technology Workshop 2007*, pages 66–74, Melbourne, Australia.

Nulty, P. and Costello, F. (2009). Using lexical patterns in the Google Web 1T corpus to deduce semantic relations between nouns. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions (SEW-2009)*, pages 58–63, Boulder, Colorado. Association for Computational Linguistics.

Park, J. C., Palmer, M., and Washburn, G. (1997). An English grammar checker as a writing aid for students of English as a second language. In *Proceedings of the fifth conference on Applied natural language processing*, pages 24–24, Morristown, NJ, USA. Association for Computational Linguistics.

Pedler, J. (2007). *Computer Correction of Real-word Spelling Errors in Dyslexic Text*. PhD thesis, Birkbeck, London University.

Pereira, F., Tishby, N., and Lee, L. (1993). Distributional clustering of english words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190.

Reynar, J. C. and Ratnaparkhi, A. (1997). A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19, Morristown, NJ, USA. Association for Computational Linguistics.

Ringlstetter, C., Hadersbeck, M., Schulz, K. U., and Mihov, S. (2007). Text correction using domain dependent bigram models from web crawls. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2007) Workshop on Analytics for Noisy Unstructured Text Data*, pages 47–54, Hyderabad, India.

Seneff, S. (1992). Tina: a natural language system for spoken language applications. *Comput. Linguist.*, 18(1):61–86.

Sjöbergh, J. (2005). Chunking: an unsupervised method to find errors in text. In Werner, S., editor, *Proceedings of the 15th NoDaLiDa conference*, pages 180–185.

Thomas, J. (2004). Using computers in correcting written work. *Teaching English with Technology*, 4(3).

Tomokiyo, L. M. and Jones, R. (2001). You're not from 'round here, are you? naive bayes detection of non-native utterance text. In *Proc. NAACL*, Pittsburgh, PA.

Trnka, K. and McCoy, K. F. (2007). Corpus studies in word prediction. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 195–202, Tempe, Arizona, USA.

Turney, P., Littman, M., Bigham, J., and Shnayder, V. (2003). Combining independent modules to solve multiple-choice synonym and analogy problems. In *Proceedings of the International Conference RANLP-2003 (Recent Advances in Natural Language Processing)*, pages 482–489, Borovets, Bulgaria.

Verberne, S. (2002). Context-sensitive spell checking based on word trigram probabilities. Master's thesis, University of Nijmegen.

Vogel, C. and Cooper, R. (1995). Robust chart parsing with mildly inconsistent feature structures. In Schter, A. and Vogel, C., editors, *Nonclassical Feature Systems*, volume 10. Centre for Cognitive Science, University of Edinburgh. Working Papers in Cognitive Science.

Wagner, J., Foster, J., and van Genabith, J. (2007). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pages 112–121.

Wang, C. and Seneff, S. (2004). High-quality speech translation for language learning. In *Proc. of InSTIL*, Venice, Italy.

Weaver, W. (1949). *Translation*. Reprinted in Locke, William N. and Booth, A. Donald (1955) (Eds.), Machine translation of languages, John Wiley & Sons, New York, 15-23.

White, M., Rajkumar, R., and Martin, S. (2007). Towards broad coverage surface realization with CCG. In *Proceedings of the Workshop on Using Corpora for NLG: Language Generation and Machine Translation (UCNLG+MT)*.

Wiemer-Hastings, P. (2000). Adding syntactic information to lsa. In *Proceedings of the 22nd Annual Conference Cognitive Science Society*, pages 989–993.

Wilcox-O'Hearn, L. A., Hirst, G., and Budanitsky, A. (2008). Real-word spelling correction with trigrams: A reconsideration of the mays, damerau, and mercer model. In Gelbukh, A., editor, *Proceedings, 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2008) (Lecture Notes in Computer Science 4919, Springer-Verlag)*, pages 605–616, Haifa.

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*, pages 88–95, Morristown, NJ, USA. Association for Computational Linguistics.

Yuret, D. (2007). KU: Word sense disambiguation by substitution. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 207–214, Prague, Czech Republic. Association for Computational Linguistics.

Appendix A

Text Correction using Different Assumptions

Here, we try to visualize the text correction scenario where we use the same approach used in chapter 6 with less strict assumptions compared to strict assumptions discussed in Section 1.2 on page 6. From now on, we will use 'assumption set 2' to refer less strict assumptions and 'assumption set 1' to refer strict assumptions. We only mention those parts that differ from Section 6.2.3 where we used assumption set 1. Here, we use less strict assumption or assumption set 2 which are discussed in the following section.

A.1 Assumption Set 2

Less Strict Assumptions

- The first token is a word.
- There should be at least three words in an input text.
- There might be at most one error in between two words. We also assume that there might be at most one error after the last word.

Weak Assumption

- We try to preserve the intended semantic meaning of the input text as much as possible.

A.2 5-gram Rules Used in Step 1

Table A.1, analogous to Table 6.1, lists all possible 5-gram rules generated from all the three operations and assumption set 2.

A.3 Limits for the Number of Steps, n_{tp}

We figure out what maximum and minimum number of steps we need for an input text. Taking the second assumption (from assumption set 2) into consideration, it is obvious that if the value of m is 3 (the number of words would also be 3) then only rules # 6, 7 and 8 in Table A.1 can be used to generate 5-grams. 5-grams generated from rule # 7 and 8 in Table A.1 can not be used in the next step as after the last word (w_3); we might have at most one error and all the 5-grams, if any, generated using these rules have this error (i.e., I_2). 5-grams generated from rules # 6 in Table A.1 can be used in the next step (by rule # 2 in Table A.2) to test whether we can insert a word in the next step, provided that the previous step generates at least one 5-gram. Thus, if $m = 3$ we might need at most 2 steps. Now, if $m = 4$, then for the added word (i.e., w_4) we need two extra steps to test rules # 5 and 2 in Table A.2, in order, on top of the previous two steps (for the first three words), provided that each previous step generates at least one 5-gram. That is, each extra token in T needs at most two extra steps. We generalize the maximum number of steps needed for an input text having m tokens as:

Table A.1: List of all possible 5-gram rules in step 1

Rule#	5-gram Rule	Generated from	Rule#	5-gram Rule	Generated from
1	$w_1\ w_2\ w_3\ w_4\ w_5$	No Operation	26	$w_1\ w_3\ I_1\ w_4\ w_5$	
2	$w_1\ I_1\ w_2\ w_3\ w_4$		27	$w_1\ w_3\ w_4\ I_1\ w_5$	
3	$w_1\ w_2\ I_1\ w_3\ w_4$		28	$w_1\ w_2\ w_4\ I_1\ w_5$	Single Delete +
4	$w_1\ w_2\ w_3\ I_1\ w_4$	Single Insert	29	$w_1\ w_2\ w_4\ w_5\ I_1$	Single Insert
5	$w_1\ w_2\ w_3\ w_4\ I_1$		30	$w_1\ w_3\ w_4\ w_5\ I_1$	
6	$w_1\ I_1\ w_2\ I_2\ w_3$		31	$w_1\ w_2\ w_3\ w_5\ I_1$	
7	$w_1\ w_2\ I_1\ w_3\ I_2$	Double Insert	32	$w_1\ w_3\ w_4\ w_5\ R_6$	
8	$w_1\ I_1\ w_2\ w_3\ I_2$		33	$w_1\ w_2\ w_3\ w_5\ R_6$	
9	$w_1\ R_2\ w_3\ w_4\ w_5$		34	$w_1\ w_3\ R_4\ w_5\ w_6$	
10	$w_1\ w_2\ R_3\ w_4\ w_5$		35	$w_1\ w_2\ w_4\ R_5\ w_6$	Single Delete +
11	$w_1\ w_2\ w_3\ R_4\ w_5$	Single Replace	36	$w_1\ w_3\ w_4\ R_5\ w_6$	Single Replace
12	$w_1\ w_2\ w_3\ w_4\ R_5$		37	$w_1\ w_2\ w_4\ w_5\ R_6$	
13	$w_1\ R_2\ w_3\ R_4\ w_5$		38	$w_1\ R_2\ w_3\ w_5\ w_6$	
14	$w_1\ w_2\ R_3\ w_4\ R_5$	Double Replace	39	$w_1\ w_2\ R_3\ w_4\ w_6$	Single Replace +
15	$w_1\ R_2\ w_3\ w_4\ R_5$		40	$w_1\ R_2\ w_3\ w_4\ w_6$	Single Delete
16	$w_1\ w_3\ w_4\ w_5\ w_6$		41	$w_1\ I_1\ w_2\ R_3\ w_4$	
17	$w_1\ w_2\ w_4\ w_5\ w_6$		42	$w_1\ w_2\ I_1\ w_3\ R_4$	Single Insert +
18	$w_1\ w_2\ w_3\ w_5\ w_6$	Single Delete	43	$w_1\ I_1\ w_2\ w_3\ R_4$	Single Replace
19	$w_1\ w_2\ w_3\ w_4\ w_6$		44	$w_1\ R_2\ w_3\ I_1\ w_4$	Single Replace +
20	$w_1\ w_3\ w_5\ w_6\ w_7$		45	$w_1\ w_2\ R_3\ w_4\ I_1$	Single Insert
21	$w_1\ w_2\ w_4\ w_6\ w_7$		46	$w_1\ R_2\ w_3\ w_4\ I_1$	
22	$w_1\ w_3\ w_4\ w_6\ w_7$		47	$w_1\ I_1\ w_2\ w_4\ w_5$	
23	$w_1\ w_2\ w_4\ w_5\ w_7$	Double Delete	48	$w_1\ w_2\ I_1\ w_3\ w_5$	Single Insert +
24	$w_1\ w_2\ w_3\ w_5\ w_7$		49	$w_1\ I_1\ w_2\ w_3\ w_5$	Single Delete
25	$w_1\ w_3\ w_4\ w_5\ w_7$				

Table A.2: List of All Possible 5-gram Rules in Step 2 to Step $2m-4$

Rule#	5-gram Rule	Generated from	Case Number
1	$---w_iw_{i+1}$	No Operation	
2	$---w_iI_j$	Single Insert	1: if the last word
3	$---w_iw_{i+2}$	Single Delete	in step 1 is in T
4	$---w_iR_{i+1}$	Single Replace	
5	$--w_iI_jw_{i+1}$	No Operation	2: if the second last word in step 1 is in
6	$--w_iR_{i+1}w_{i+2}$	No operation	T and the last word is either an inserted or a replaced word

$$\text{Max } n_{tp} = 2+(m-3)\times 2 = 2m-4 \tag{A.1}$$

Again, the minimum number of steps is ensured if rules # 6 to 8 (Table A.1) in step 1 do not generate any 5-gram. This means that, if $m = 3$, we might need at least 0 steps. Now, if $m = 4$ then for the added word (i.e., w_4) we need only an extra step to test rule # 5 in Table A.2 on top of the previous single step (for the first three tokens). That is, each extra token in T needs at least one extra step, provided that each previous step for each extra token generates at least one 5-gram. We generalize the minimum number of steps needed for an input text having m tokens as:

$$\text{Min } n_{tp} = m-3 \tag{A.2}$$

In (A.1), the maximum number of steps, $2m-4$, also means that the maximum number of tokens possible in a candidate text is $2m$. Thus, an input text having $2m$ tokens can have at most m errors to be handled and m correct words, assuming $m \geq 3$ (the second assumption on page 158).

A.4 5-gram Rules used in Step 2 to $2m-4$

Table A.2, analogous to Table 6.2, lists all possible 5-gram rules generated from all the three operations and assumption set 2 for step 2 to step $2m-4$. We use step 2 (i.e., the next step) only if step 1 (i.e., the previous step) generates at least one 5-gram from 5-gram rules listed in Table A.1. Similarly, we use step 3 (i.e., the next step) only if step 2 (i.e., the previous step) generates at least one 5-gram from the 5-gram rules listed in Table A.2, and so on. In Table A.2, '$-$' means that it might be any word that is in T, or an inserted word (an instance of I's), or a replaced word (an instance of R's) in the previous step. To give a specific example of how we list the 5-gram rules in Table A.2, consider that rule #2 ($w_1\ I_1\ w_2\ w_3\ w_4$) in Table A.1 generates at least one 5-gram in step 1. We take the last four words of this 5-gram (i.e., $I_1\ w_2\ w_3\ w_4$) and add the next word from T (in this case w_5), in order to form a new rule in step 2 (which is $I_1\ w_2\ w_3\ w_4\ w_5$). The general form of this rule ($-\ -\ -\ w_i\ w_{i+1}$) is listed as rule #1 in Table A.2. In step 1, I_1 in rule #2 acts like a variable, but in step 2 we use only a single instance of I_1, which acts like a constant. We categorize all the 5-grams generated in step 1 (i.e., the previous step) into two different cases. Case 1 groups each 5-gram in step 1 having its last word in T. Case 2 groups each 5-gram in step 1 having its second last word in T, and the last word not in T. We stop when we fail to generate any 5-gram in the next step from all the 5-gram rules of the previous step.

A.5 Determining the Limit of Candidate Texts

There might be a case when no 5-gram is generated in step 1; this means that the minimum $\tilde{n}$ possible is 0. Table A.1 shows that there are 11 5-gram rules (rules without any I's or R's) in step 1 that generate at most one 5-gram per 5-gram rule. It turns out that the remaining 5-gram rules can generate at most $\bar{n}$ 5-grams per 5-gram rule. Thus,

the maximum number of candidate texts, $\tilde{n}$, that can be generated having only a single step (i.e., $n_{tp} = 1$) is:

$$\begin{aligned}\text{Max } \tilde{n} = &(\text{ number of 5-gram rules in a step } - \text{number of 5-gram rules in the same} \\ &\text{step without any } I\text{'s or } R\text{'s}) \times \bar{n} + \text{number of 5-gram rules in the same step} \\ &\text{without any } I\text{'s or } R\text{'s} \qquad \text{(A.3)} \\ &= (49 - 11) \times \bar{n} + 11 = 38\bar{n} + 11 \qquad \text{(A.4)}\end{aligned}$$

At most $2\bar{n} + 2$ 5-grams (rules #1 to 4 in Table A.2) can be generated in step 2 from a single 5-gram generated in step 1 having the last word in T. There may be at most 33 such 5-grams in step 1. At most 1 5-gram (rules #5 and 6 in Table A.2) can be generated in step 2 from a single 5-gram generated in step 1 having the second last word in T and the last word being either an inserted or a replaced word. There may be at most 16 such 5-grams in step 1. The maximum number of candidate texts, $\tilde{n}$, that can be generated having two steps (i.e., $n_{tp} = 2$) is:

$$\text{Max } \tilde{n} = 33(2\bar{n} + 2) + 16 \times 1 \qquad \text{(A.5)}$$

We generalize Max $\tilde{n}$, analogous to equation 6.11, for different values of n_{tp} as:

$$\text{Max } \tilde{n} \approx \begin{cases} 38\bar{n}+11 & \text{if step} = 1 \\ 33\times 2^0(2\bar{n}+2)+16 & \text{if step} = 2 \\ 33\times 2^1(2\bar{n}+2)+66\times 2^0\bar{n} & \text{if step} = 3 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ 33\times 2^{n_{tp}-2}(2\bar{n}+2)+66\times 2^{n_{tp}-3}\bar{n} & \text{if step} = n_{tp} \end{cases} \qquad \text{(A.6)}$$

Table A.3: Some examples.

	Example 1	Example 2
Incorrect	All funding decisions is made the	What his believed to the next
Correct	All funding decisions are made by the	What is believed to be the next
Judge 1	All funding decisions is made by the	What is believed to be next
Judge 2	All funding decisions are made the	What he believed to be the next
Our Method	All funding decisions are made by the	What is believed to be the next

Simplifying (A.6),analogous to equation 6.12:

$$\text{Max } \tilde{n} \approx \begin{cases} 38\bar{n}+11 & \text{if step} = 1 \\ 66\bar{n}+82 & \text{if step} = 2 \\ 2^{n_{tp}-3}(198\bar{n}+132) & \text{if step} \geq 3 \end{cases} \tag{A.7}$$

A.6 Evaluation on WSJ Corpus

We use 34 out of the 50 short texts used in Section 6.4.1. To be fair, we omit 16 short texts from the data set which contain consecutive errors. In the same way, we omit the scores of those 16 examples for the two human judges. The average number of tokens in a correct text and an incorrect text are 7.44 and 6.32, respectively. The average number of corrections required per text is 1.76. The agreement between the two judges is low (the detection agreement is 53.85% and the correction agreement is 50.77%). Table A.3 shows three examples of test texts. The results in Table A.4 show that our method gives comparable *recall* value for both detection and correction, whereas human judges give better *precision* value for both detection and correction. Since a majority of the words in the evaluation data set are correct, the *baseline* is to propose no correction, achieving 76.28% accuracy. Taking this *baseline* accuracy as a lower limit and the accuracy achieved by the human judges as an upper limit, we conclude that the automatic method using

Table A.4: Results on the WSJ corpus using assumption set 2.

	Detection			**Correction**			**Acc.**
	R	**P**	**F_1**	**R**	**P**	**F_1**	
Our Method	90.0	75.00	81.82	78.33	65.28	71.21	84.98
Judge 1	65.0	88.64	75.00	58.33	79.54	67.31	86.56
Judge 2	90.0	93.10	91.53	83.33	86.21	84.75	92.89

assumption set 2 realizes about half of the possible improvement between the baseline and the human expert upper bound (76%-84%-92%, respectively).

A.7 Summary

It is obvious that text correction approach discussed in Chapter 6 using assumption set 2 can not correct any text having two consecutive errors. The number of steps, n_{tp}, required using assumption set 2 is in $[m-3,\ 2m-4]$ compared to $[m-3,\ 3m-6]$ when using assumption set 1. Again, the maximum number of candidate texts that can be generated for different values of n_{tp} using assumption set 2 is $2^{n_{tp}-3}(198\bar{n}+132)$ compared to $3^{n_{tp}-4}(864\bar{n}^2 + 1620\bar{n} + 1458)$ when using assumption set 1.

Appendix B

Managing the Google Web 1T 5-gram Data Set

Here, we describe how the Google Web 1T 5-gram data set, contributed by Google Inc., can efficiently be used and managed as a context window. We present an efficient way of accessing all the 5-grams containing the context words for a specific word of interest from the stored files. We measure the maximum access and processing efficiency achievable for any word of interest. It is expected that this data will be useful for a variety of Research and Development projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, as well as for other uses. We also compare results (access time and memory requirements) on the task of accessing all the 5-grams containing the context words for a list of words, on both the processed and the original organization of the data set.

B.1 Introduction

The Web 1T 5-gram corpus contains so much data that many machines with average amounts of memory are unable to even load it. We propose a method for splitting the

enormous list of 5-grams, released in 118 large files, down to a more manageable size based on user requirements.

We use the 5-grams from the Web 1T corpus such that the middle token is the term and the two tokens on either side form the context. Several researchers tried to find out the advantages of this context definition by initiating the practical question of what minimum value of window size would, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word (Weaver, 1949). A well-known early experiment by Kaplan (1950) attempted to answer this question at least in part, by presenting ambiguous words in their original context and in a variant context providing one or two words on either side to seven translators. Kaplan (1950) observed that sense resolution given two words on either side of the word was not significantly better or worse than when given the entire sentence (Ide and Véronis, 1998). The same phenomenon has been reported by several researchers: e.g., Masterman (1961); Koutsoudas and Korfhage (1956) on Russian, and Choueka and Lusignan (1985) on French. This phenomenon indicates that the Web 1T 5-grams can be a useful tool for natural language processing. Whenever we talk about window size, we consider the word of interest in the middle of the window and context words on either side of the word of interest. For example, a window of size 5 will have two context words on either side of the middle word (which is actually the word of interest); but we can not use the Web 1T 5-grams as a window of 5 as all the 5-gram files (total 118 files) are sorted based on the first word in the 5-grams and then the second word in the 5-grams and so on.

We wish to find out what maximum n-gram access and processing efficiency can be achieved for any word of interest. Searching in the original Web 1T dataset is dependent on the total number of n-grams of all the words, including the word of interest. Our goal is to make this search independent of the total number of n-grams of all the words, and thus the only dependency will be on the number of n-grams for that word of interest;

this ensures the maximum access and processing efficiency achievable for any word of interest.

Two of our primary assumptions (used in Section B.2) are that we need to store less than or equal to n (for our experiment, $n = 100,000$) 5-grams in each single file and we plan to read only a single file to process all the 5-grams for a specific word of interest. We omit the first assumption and add a new assumption (used in Section B.3) that we store only all the 5-grams of unique word in a single file.

This chapter is organized as follow: in Section B.2, we show how the Web 1T 5-gram data set can be used as a balanced window and how this data can be stored, accessed or processed efficiently with respect to time and memory space and we present experimental results. In Section B.3, we find out what maximum n-gram access and processing efficiency can be achieved. We summarize contributions and conclude in Section B.4.

B.2 Processing the Web 1T 5-grams

As an example, consider a sentence consisting of 10 words (w_1 to w_{10}) (Figure B.1) that generates six 5-grams. Figure B.1 details the process of how Google's 5-grams are constructed. Now, if w_4 in Figure B.1 is our word of interest then considering the 5-gram starting with word w_4 (5-gram#4 in Figure B.1), we are actually considering four context words (w_5 to w_8) on the right side of the word w_4. According to the classical/conventional window size concept, we should consider the 5-gram starting with word w_2 (5-gram#2 in Figure B.1) where w_4 is in the middle and there are two words on either side of the word w_4. The problem here is that as the Web 1T 5-grams are sorted based on the first words in the 5-grams, we need to search the middle words in 118 large files of 5-grams. That is, to find the context words of a single word, we need to search all the 5-grams (1,176,470,663), which is not time efficient at all, especially in the applications where

$$\overbrace{w_1\ w_2\ w_3\ w_4\ w_5}^{5-gram\#1}\ \underbrace{w_6\ w_7\ w_8\ w_9\ w_{10}}_{5-gram\#6}$$

5-gram#	word1	word2	word3	word4	word5	frequency
1	w_1	w_2	w_3	w_4	w_5	1
2	w_2	w_3	w_4	w_5	w_6	1
3	w_3	w_4	w_5	w_6	w_7	1
4	w_4	w_5	w_6	w_7	w_8	1
5	w_5	w_6	w_7	w_8	w_9	1
6	w_6	w_7	w_8	w_9	w_{10}	1

Figure B.1: How Google's 5-grams are constructed.

we are in need of finding context words of a specific word of interest on the fly. So, we process all the Web 1T 5-grams as described in the next sections, in order to make the procedure of finding context words time efficient.

B.2.1 Preprocessing the Web 1T 5-grams

Some words in the Web 1T 5-grams start with special characters or numbers. We replace the words that start with these characters with blank spaces in all the 5-grams. As a result after the preprocessing step, it is not always the case that all the 5-grams have the exactly five words. We also replace all the times, dates, and numbers with blank spaces. In this way, if the middle word of a 5-gram is eliminated then we eliminate the whole 5-gram. We also transform all the 5-grams into lower case; this creates many duplicate 5-grams (5-grams having the same words). We merge all the duplicate 5-grams into a single 5-gram by adding their frequencies. In this way we managed to reduce the number of 5-grams from 1,176,470,663 to 356,611,338 and the file size of all the 118 5-gram files from 33.68 GB to 9.48 GB. Using the same technique, we managed to reduce the number of unigrams from 13,588,391 to 2,870,385.

We re-sort all the 118 preprocessed 5-gram files based on the middle words and restore

the sorted 5-gram files to place the middle words in the first position. Thus, the 118 re-sorted 5-gram files have 356,611,338 5-grams requiring 9.48 GB of storage.

B.2.2 Calculating Unique Two Characters and Their Frequencies using 5-grams

We devise an algorithm (Algorithm 3) that generates all the unique two-character tokens/words (taking the first two characters from the first word in each 5-gram) and their frequencies from all the 5-grams of 118 re-sorted files. Figure B.2 shows a snap shot

Algorithm 3: calculating unique two characters and their frequencies using 5-grams

```
   input : input files    /* input files are all 118 re-sorted 5-gram files
           */
   output: output file    /* output file contains unique two characters and
           their frequencies */
 1 for each sorted file do
   |                        /* total 118 re-sorted files */
 2 |   for each 5-gram do
 3 |   |   word_1 ← first word of the 5-gram
 4 |   |   str ← SubStr(word_1, 2)    /* returns first two characters of word_1
   |   |   */
 5 |   |   if str ∈ list then    /* list contains unique two characters (i.e.,
   |   |   str) and their frequencies (i.e., list.str) */
 6 |   |   |   increment list.str
 7 |   |   else
 8 |   |   |   list.str ← 1          /* initialize frequency for str */
 9 |   |   end
10 |   end
11 end
12 for each str ∈ list do
13 |   write str and list.str to output file
14 end
```

of the output file generated by Algorithm 3. For example, *na* 1,793,071 means the number of 5-grams (in the re-sorted files) having first words starting with *na* (the first

two characters) is $1,793,071$. Algorithm 3 generates 962 unique two-character tokens[1] using Google's all 118 re-sorted 5-gram files. Figure B.2 shows such nine entries from 962 possible entries. That is, the output file of Algorithm 3 contains the first two unique characters of all 5-grams with their frequencies.

token	frequency	token	frequency
..	...	*nf*	44,130
na	1,793,071	*ng*	31,181
nb	34,454	*nh*	40,302
nc	70,151	*ni*	669,766
nd	23,766	..	...
ne	4,223,689		

Figure B.2: A snapshot of output file generated by Algorithm 3.

Now we need to make a decision of what maximum number of 5-grams[2] (say N) we will store in a single file so that we can have efficient search results to find context words for a specific word of interest. Based on the value of N, we split the output file generated by Algorithm 3 into two files. One file (say *filesName*) contains only the words (not frequencies) having frequencies less than N. Another file (say *nextInFile*) also contains only the words (not frequencies) having frequencies greater than or equal to N. Using the snap shot in Figure B.2 as an example, *filesName* will have *nb*, *nc*, *nd*, *nf*, *ng*, *nh* and *nextInFile* will have *na*, *ne* and *ni* when $N = 100{,}000$.

B.2.3 Calculating Unique $p+1$ Characters and Their Frequencies using 5-grams

We execute Algorithm 4 with *nextInFile* and $p = 2$ (p denotes the number of characters). Algorithm 4 generates all the unique $(p+1)$-character tokens and their frequencies from

[1]First character $\in \{a \cdots z\}$ and the second character $\in \{a \cdots z, 0 \cdots 9, '\}$. So, there are total $26 \times 37 = 962$ unique two-character tokens possible.

[2]For the Web 1T 5-gram we use $N = 100{,}000$ as the state-of-the-art computing device can process these numbers of 5-grams in less than a second.

those 5-grams of 118 re-sorted files having the first two characters of the first words in *nextInFile*.

Algorithm 4: calculating unique $p+1$ characters and their frequencies using 5-grams

```
   input : p, input files and current nextInFile      /* input files are all 118
           re-sorted 5-gram files and p ≥ 2 */
   output: output file  /* output file contains unique p+1 characters and
           their frequencies */
 1 listFN ← nextInFile  /* listFN contains all the tokens in nextInFile */
 2 for each sorted file do
 3 |
 4 |   for each 5-gram do
 5 |   |   word_1 ← first word of the 5-gram
 6 |   |   if SubStr(word_1, p) ∈ listFN then
 7 |   |   |   str ← SubStr(word_1, p+1)
 8 |   |   |   if list.str > 0 then                          /* str is in list */
 9 |   |   |   |   increment list.str
10 |   |   |   else
11 |   |   |   |   list.str ← 1
12 |   |   |   end
13 |   |   end
14 |   end
15 end
16 for each str ∈ list do
17 |   write str and list.str to output file
18 end
```

Using Figure B.2 as an example, Algorithm 4 will not process any 5-gram having first word started with *nb*, *nc*, *nd* and so on. That is, it will process *na*, *ne* and *ni*. Figure B.3 shows the output of Algorithm 4 with $p = 2$ and *nextInFile* (assuming *nextInFile* contains only the word *na*).

We split the output file generated by Algorithm 4 and add the words having frequencies less than N into the file *filesName* and we store words having frequencies greater than or equal to N by overwriting the file *nextInFile*.

Now, *filesName* will have *nb*, *nc*, *nd*, nf, *ng*, *nh*, *naa*, *nab*, *nac*, *nad*, *nae*, naf, *nag*,

token	frequency	token	frequency	token	frequency
...	...				
naa	5,029	*naj*	2,196	*nas*	84,686
nab	5,454	*nak*	71,359	*nat*	756,338
nac	15,210	*nal*	4,167	*nau*	26,168
nad	14,560	*nam*	421,422	*nav*	125,127
nae	1,685	*nan*	58,106	*naw*	1,821
naf	2,968	*nao*	4,196	*nax*	1,318
nag	14,239	*nap*	39,560	*nay*	4,062
nah	3,651	*naq*	632	*naz*	13,965
nai	35,347	*nar*	55,307	...	...

Figure B.3: A snapshot of output file generated by Algorithm 4 when $p = 2$.

nah, *nai*, *naj*, *nak*, *nal*, *nan*, *nao*, *nap*, *naq* and *nar* and *nextInFile* will have only *nam*. We carry on executing Algorithm 4 with current *nextInFile* and incremented p (i.e., this time $p = 3$). Figure B.4[3] shows the output of Algorithm 4 with $p = 3$ and *nextInFile* (*nextInFile* contains only *nam*).

token	frequency	token	frequency	token	frequency
nam	5,474	~~*namd*~~	~~186~~	*namp*	673
~~*nam′*~~	~~6~~	*name*	397,776	~~*namq*~~	~~2~~
~~*nam1*~~	~~1~~	~~*namf*~~	~~11~~	~~*namr*~~	~~83~~
~~*nam2*~~	~~2~~	~~*namg*~~	~~43~~	~~*nams*~~	~~178~~
~~*nam3*~~	~~1~~	~~*namh*~~	~~36~~	~~*namt*~~	~~87~~
~~*nam4*~~	~~1~~	*nami*	11,469	*namu*	454
~~*nam6*~~	~~2~~	~~*namj*~~	~~19~~	~~*namv*~~	~~22~~
~~*nam7*~~	~~1~~	~~*namk*~~	~~35~~	~~*namw*~~	~~29~~
~~*nam9*~~	~~2~~	~~*naml*~~	~~95~~	~~*namx*~~	~~3~~
nama	1,576	*namm*	515	~~*namy*~~	~~42~~
namb	795	~~*namn*~~	~~128~~	~~*namz*~~	~~2~~
namc	1,181	*namo*	492		

Figure B.4: A snapshot of output file generated by Algorithm 4 when $p = 3$.

[3]We cross entries like *nam′*, *nam*1, *namd* and so on to reduce the total number of files. We merge small files (less than 200 5-grams) to their immediate parent file. For this example their parent file is *nam*. Now, if the first four characters of the word of interest are *namd*, we first search a file with *namd* and obviously we find no file with this name. We then chop the last character from *namd* (chopping *d* from *namd* generates *nam*) and search in *nam* file. This way we try to balance the file size.

We follow the same steps of executing Algorithm 4 until *nextInFile* is empty. For the Web 1T 5-gram data set, we get an empty *nextInFile* when $p = 12$.

Finally *filesName* contains all the file names that need to be created[4].

B.2.4 Creating Files and Assigning 5-grams to Files

We execute Algorithm 5 which practically creates all the files needed based on *filesName* entries and assigns each 5-gram to the appropriate file (the entries in *filesName* are the name of the files) traversing all 118 re-sorted 5-gram files.

Algorithm 5: creating files and assigning 5-grams to the files

```
   input  : input files and filesName      /* input files are all 118 re-sorted
            5-gram files */
   output: output files       /* the number of output files is equal to the
            number of tokens in filesName and the names of these files
            are the token names in filesName */
 1 listFN ← filesName
 2 for each sorted file do
 3 |
 4 |   for each 5-gram do
 5 |   |   word_1 ← first word of the 5-gram
 6 |   |   len ← length(word_1)
 7 |   |   not_found ← TRUE
 8 |   |   while len > 1 AND not_found = TRUE do
 9 |   |   |   if SubStr(word_1, len) ∈ listFN then
10 |   |   |   |   fname ← SubStr(word_1, len)
11 |   |   |   |   not_found ← FALSE
12 |   |   |   |   Open fname in append mode
13 |   |   |   |   write 5-gram to fname
14 |   |   |   else
15 |   |   |   |   decrement len
16 |   |   |   end
17 |   |   end
18 |   end
19 end
```

[4] *filesName* contains 22,743 entries after running Algorithm 4 on the Web 1T 5-grams.

B.2.5 Efficient Search in the Web 1T 5-grams

To find all the 5-grams for a specific word of interest, we need to search only in a single file among those 22,743 files created using Algorithm 5 in Section B.2.4. The good thing about these files is that the maximum number of 5-grams in each file[5] is less than 100,000, and as a result it is possible to process all the 5-grams for a specific word with time efficiency. Algorithm 6 is used to locate the specific file that have all the 5-grams for a specific word and then to access all the 5-grams for that word to further process as per requirement[6].

Algorithm 6: finding the specific file and all the 5-grams for a specific word

```
   input  : word                          /* word is the word of interest */
   output: fname    /* fname is the file that contains all the 5-grams of
           word */
 1 listFN ← filesName
 2 len ← length(word)
 3 not_found ← TRUE
 4 while len > 1 AND not_found = TRUE do
 5 |   if SubStr(word, len) ∈ listFN then
 6 |   |   fname ← SubStr(word, len)
 7 |   |   not_found ← FALSE
 8 |   else
 9 |   |   decrement len
10 |   end
11 end
12 Open fname in read mode
13 for each 5-gram in fname do
14 |   word_1 ← first word of the 5-gram
15 |   if word = word_1 then
16 |   |   do as per requirement
17 |   end
18 end
```

[5]By changing the value N in Section B.2.2 it is possible to change the size of the maximum number of 5-grams each file can have.

[6]We can insert code for what we plan to do with these 5-grams in line 16 of Algorithm 6.

B.2.6 Comparing Efficiency in Time and Memory Space

We use Miller and Charles (1991) data set, a well known data set used to judge different natural language processing (NLP) tasks, to analyze the access time and main memory requirements on run time and to process a specific task[7]. Though there are 30 pairs of words in this data set, the number of distinct words are 39. We experimented on two different machines. The first machine is a dual core Intel® Xeon™ having CPU speed of 3.20 GHz and main memory of 4 GB (we refer it as Dual Core Machine). The second machine is an IBM ThinkCentre M51 machine with Intel® Pentium 4 processor having CPU speed of 3.40 GHz and main memory of 1GB (we refer it as ThinkCentre). Table B.1 shows all the test results on Miller and Charles (1991) data set.

Figure B.5 shows the time[8] (in seconds) needed to access all the 5-grams of the 39 words on the ThinkCentre and the Dual Core Machine.

Figure B.5 shows that 38 words out of 39 (97.4%) were accessed in less than a second, 33 words (84.6%) were accessed with in 0.5 second, 20 words (51.3%) were accessed with in 0.2 second on the Dual Core Machine. The total access time for all the 39 words on this machine is 12 seconds[9], an average of 0.31 seconds per word. Figure B.5 also shows that 31 words out of 39 (79.5%) were accessed in less than a second, 9 words (23.1%) were accessed with in 0.2 second on the ThinkCentre. The total access time for all the 39 words on this machine is 25.05 seconds[10], an average of 0.64 seconds per word which is almost double compared to the Dual Core Machine. The first word, which is the word *car*, takes longer time to be accessed (see Figure B.5) or to be processed (see Figure B.6)

[7]As a specific task for processing, we access all the 5-grams of a specific word of interest and then from those 5-grams we find all the unique context/neighbor words and the pair frequencies. By pair we mean the word of interest and one context/neighbor word.

[8]We apply *linear search*, which simply examines each element of the file in order. To calculate the search time we use the Benchmark module of Perl. This module comes bundled with Perl, and can be imported into any Perl script through the "use" command.

[9]Though we need 11.97 seconds to access all the 39 words, we need to load *filesName* which takes 0.03 second: the total is 12 seconds.

[10]It includes the time needed to load *filesName* which is 0.13 second.

Table B.1: Experimental Results on Miller and Charles Data Set

Word Number	Word	Time to access all the 5-grams (in seconds)			Time to access all the 5-grams & to find all the context words and the pair frequencies (in seconds)			Number of searched 5-grams		Number of target 5-grams	Memory required to run (in bytes)
		Dual Core Machine		Think Centre (TER 0.67)	Dual Core Machine		Think Centre (TER 0.67)	TER 0.67	TER 1		
		TER 0.67	TER 1		TER 0.67	TER 1					
1	car	1.66	0.07	3.39	4.35	3.65	5.5	272,920	272,532	272,532	455,583
2	automobile	0.28	0.01	0.58	0.42	0.18	0.69	47,395	11,863	11,863	57,993
3	gem	0.19	0.00	0.38	0.30	0.14	0.47	31,891	9,342	9,342	58,488
4	jewel	0.40	0.00	0.11	0.13	0.12	0.19	8,263	7,936	7,936	53,606
5	journey	0.13	0.01	0.27	0.33	0.27	0.45	23,996	19,508	19,508	93,099
6	voyage	0.37	0.00	0.76	0.46	0.07	0.80	60,527	4,684	4,684	37,016
7	boy	0.40	0.02	0.81	1.08	0.93	1.42	67,943	67,391	67,391	209,030
8	lad	0.01	0.00	0.03	0.03	0.03	0.05	2,471	1,837	1,837	18,424
9	coast	0.51	0.02	1.08	1.12	0.82	1.55	86,362	59,030	59,030	170,044
10	shore	0.15	0.00	0.33	0.33	0.23	0.48	27,400	15,837	15,837	78,535
11	asylum	0.09	0.00	0.17	0.15	0.07	0.23	15,656	5,057	5,057	35,086
12	madhouse	0.01	0.00	0.03	0.02	0.00	0.03	2,123	179	179	3,057
13	magician	0.39	0.01	0.81	0.46	0.04	0.83	70,113	2,497	2,497	21,479
14	wizard	0.12	0.00	0.26	0.28	0.21	0.39	22,324	14,471	14,471	75,424
15	midday	0.54	0.00	1.11	0.61	0.01	1.11	91,534	992	992	10,069
16	noon	0.10	0.00	0.20	0.17	0.08	0.27	18,058	5,806	5,806	31,747
17	furnace	0.03	0.00	0.06	0.06	0.05	0.09	4,952	3,104	3,104	23,627
18	stove	0.05	0.00	0.11	0.10	0.07	0.14	8,340	4,747	4,747	28,568
19	food	0.97	0.05	2.05	2.68	2.28	3.44	171,215	170,079	170,079	305,285
20	fruit	0.32	0.01	0.66	0.63	0.39	0.91	57,135	28,403	28,403	108,827
21	bird	0.40	0.01	0.86	0.85	0.59	1.20	71,947	40,779	40,779	152,996
22	cock	0.39	0.02	0.77	0.97	0.83	1.22	60,695	60,424	60,424	92,293
23	tool	0.50	0.02	1.06	1.40	1.18	1.78	87,837	87,022	87,022	222,354
24	implement	0.14	0.01	0.28	0.41	0.34	0.53	26,026	26,022	26,022	82,762
25	brother	0.35	0.01	0.73	0.72	0.46	1.03	63,185	34,244	34,244	119,188
26	monk	0.13	0.00	0.28	0.19	0.06	0.33	24,507	3,630	3,630	32,736
27	oracle	0.12	0.01	0.27	0.32	0.27	0.44	21,346	19,624	19,624	84,541
28	rooster	0.05	0.00	0.09	0.08	0.03	0.11	8,580	1,924	1,924	18,374
29	cemetery	0.13	0.00	0.27	0.25	0.15	0.36	22,317	10,286	10,286	64,289
30	woodland	0.05	0.00	0.09	0.11	0.08	0.16	9,048	5,286	5,286	39,352
31	hill	0.44	0.02	0.91	1.22	1.06	1.56	74,961	73,998	73,998	251,043
32	slave	0.55	0.01	1.14	0.77	0.24	1.28	98,079	16,573	16,573	72,514
33	forest	0.42	0.01	0.88	0.95	0.69	1.31	73,224	48,829	48,829	174,405
34	graveyard	0.17	0.00	0.38	0.21	0.03	0.39	32,559	1,262	1,262	15,320
35	chord	0.14	0.00	0.28	0.19	0.05	0.30	24,158	3,384	3,384	23,465
36	smile	0.54	0.00	1.14	0.71	0.14	1.22	98,610	10,149	10,149	56,234
37	glass	0.49	0.03	1.03	1.36	1.15	1.72	84,160	83,244	83,244	206,989
38	string	0.47	0.02	0.98	1.16	0.90	1.56	87,937	66,204	66,204	236,570
39	crane	0.13	0.00	0.28	0.20	0.09	0.33	23,252	5,425	5,425	43,547

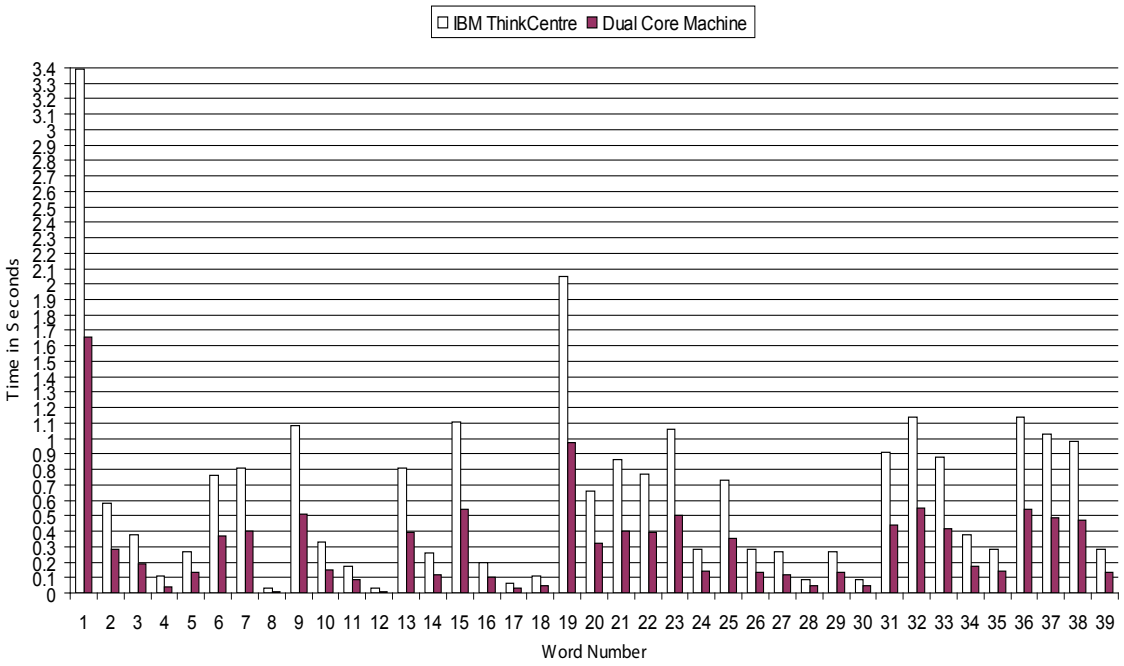

Figure B.5: Time (in seconds) needed to access all the 5-grams of the 39 words on the ThinkCenter and the Dual Core Machine.

because it has more 5-grams than the other words (as shown in Figure B.7)[11].

Figure B.6 shows the time (in seconds) needed to access all the 5-grams and then to find all the context words and the pair frequencies for all the 39 words on the ThinkCentre and the Dual Core Machine.

Figure B.6 shows that 31 words out of 39 (79.5%) were processed within one second, 23 words (59%) were processed within 0.5 second, and 12 words (30.8%) were processed within 0.2 second on the Dual Core Machine. The total processing time for all the 39 words on the Dual Core Machine is 26.11 seconds, an average of 0.67 seconds per word. Figure B.6 also shows that 24 words out of 39 (61.5%) were processed within one second, and 7 words (17.9%) were processed within 0.2 second on the ThinkCentre. The total processing time for all the 39 words on the ThinkCentre is 36.34 seconds, an average of 0.93 seconds per word.

[11]Figure B.8 shows that the word *car* also requires more memory.

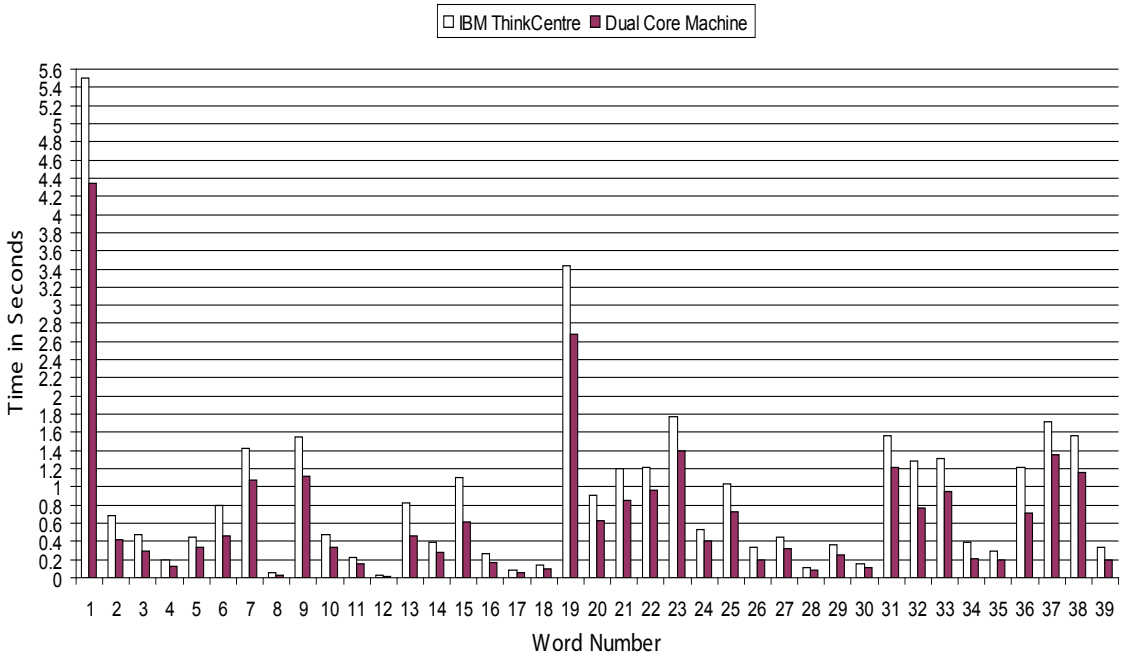

Figure B.6: Time (in seconds) needed to access all the 5-grams and then to find all the context words and the pair frequencies for all the 39 words of interest on the ThinkCenter and the Dual Core Machine.

Figure B.7 shows a comparison between the number of 5-grams we search through (we call them searched 5-grams) versus the number of 5-grams we needed (we call them target 5-grams).

The number of searched 5-grams and the number of target 5-grams are independent of the computing devices. For example, to have all the target 5-grams for word *boy* (i.e., 67,391), we need to search 67,993 5-grams. The ratio between target 5-grams and searched 5-grams (we call it Time Efficiency Ratio (TER)) is a key issue for efficiency in terms of time:

$$TER = \frac{\text{number of target 5-grams}}{\text{number of searched 5-grams}}$$

The higher the time efficiency ratio the better. Having the time efficiency ratio equal to 1 is an ideal case. The average time efficiency ratio for all the 39 words using the original

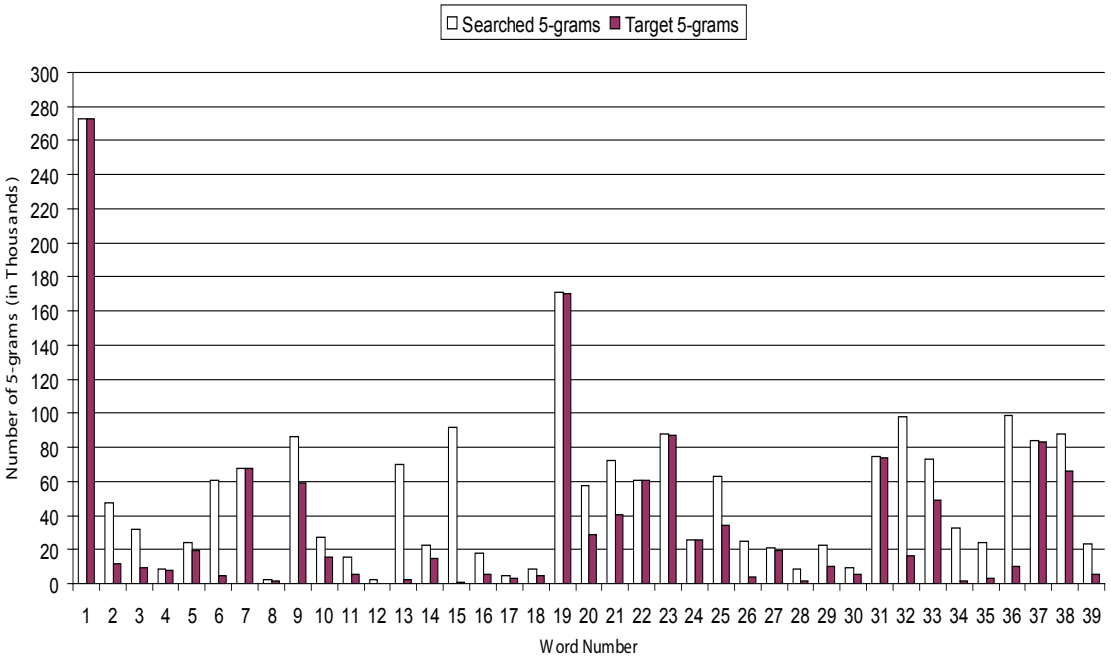

Figure B.7: Comparison between target 5-grams versus searched 5-grams for all the 39 words.

118 files is 0.001108 (this is the base case). The average time efficiency ratio for all the 39 words using 22,743 files is 0.63. To have an ideal average time efficiency ratio, we need to have a distinct 5-gram file for each unique word (unigram) possible. That is, for the Web 1T 5-gram data set, we have to have more than 2.8 millions 5-gram files to achieve an ideal time efficiency ratio. In Section B.3, we study the feasibility and practicability of generating this enormous number of files.

Figure B.8 shows the amount of main memory required to access all the 5-grams and to find all the context words and the pair frequencies. It shows that 27 words out of 39 (69.2%) need less than 100 KB main memory, and the average is 99 KB of main memory per word. It is notable that the required runtime memory is also independent of the computing device.

Using the 118 re-sorted 5-gram files, it takes 16,037 seconds on the Dual Core Machine to access all the 5-grams for the 39 words in a single pass over all the 5-gram files, taking

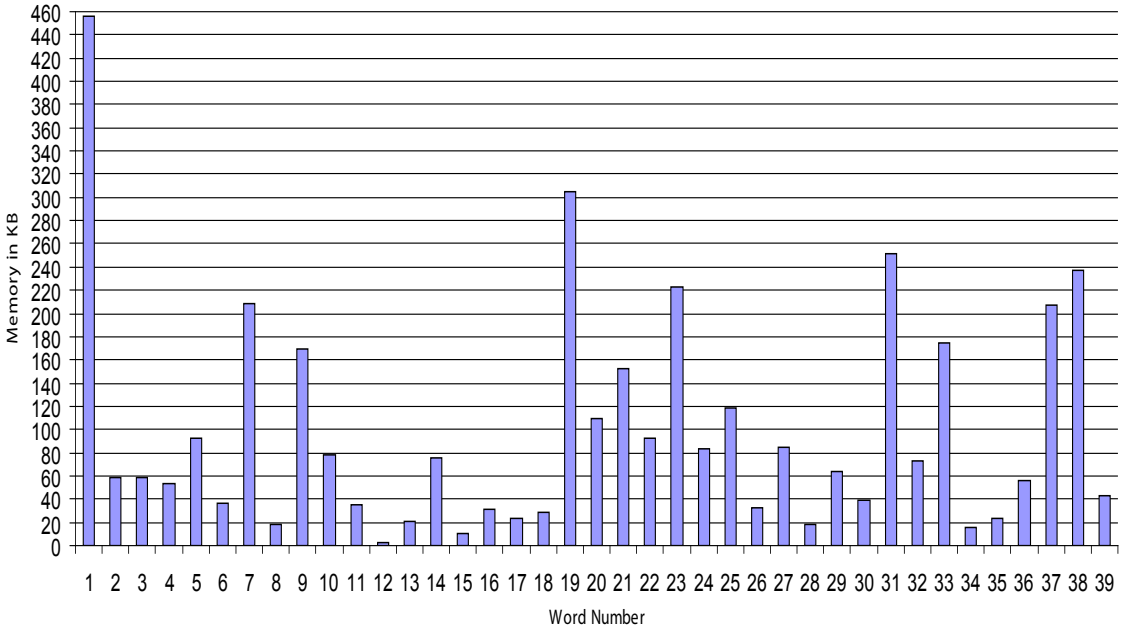

Figure B.8: Main memory required on the fly to access all the 5-grams and to find all the context words and the pair frequencies for all the 39 words of interest.

an average of 411.2 seconds whereas it takes 2,370 seconds if we use only one word, the word $fruit$ (word number 20 in Table B.1). Again, using the same 118 re-sorted 5-gram files, the ThinkCentre takes 22,266.24 seconds to access all the 5-grams for the 39 words in a single pass over all the 5-gram files, taking an average of 570.9 seconds whereas it takes 4,302.94 seconds if we use only one word, $fruit$.

We use the Web 1T 5-grams in uncompressed form, trading off disk space for speed. The largest file in *filesName* is *new*[12] having 1,310,730 5-grams and using 32,338,060 bytes (32.34 MB) of memory. The size of *filesName* is 124,262 bytes (0.12 MB). So, the

[12]All the 1,310,730 5-grams in *new* start with the word *new*. We mentioned two of our assumptions that we store less than or equal to N (for our experiment, $N = 100,000$) 5-grams in a single file and to process all the 5-grams for a specific word of interest, we open and read only a single file. For this example (*new* file), we trade off the first assumption in favor of the other as the execution time required to open and read more than one file is much longer than that of a single file. Again, keeping on the first assumption means we are to split the *new* file into 14 different files and search each of the 14 files to process all the 5-grams for the word *new*.

maximum main memory required for any word is 32.46 MB. It takes 8.55 seconds on the Dual Core Machine to access all the 5-grams for word *new* and 18.42 seconds to access all the 5-grams and then to find all the context words and the pair frequencies. ThinkCentre takes 15.94 seconds to access all the 5-grams for word *new* and 23.11 seconds to access all the 5-grams and then to find all the context words and the pair frequencies. We can conclude that the time required by the Dual Core Machine to access all the 5-grams for any word is less than or equal to 8.55 seconds and to access all the 5-grams and then to find all the context words and the pair frequencies is less than or equal to 18.42 seconds (when N=100,000) and the time required by the ThinkCentre to access all the 5-grams for any word is less than or equal to 15.94 seconds and to access all the 5-grams and then to find all the context words and the pair frequencies is less than or equal to 23.11 seconds (when N=100,000).

B.3 Making the time efficiency ratio (TER) equal to 1

To have an ideal TER, we use a new assumption (instead of the first assumption used in Section B.2) that we store only all the 5-grams of each unique word in a single file. We also use the second assumption used in Section B.2 that we read only a single file to process all the 5-grams for a specific word of interest. That is, each single file contains only all the target 5-grams for a specific word of interest, and hence we do not need to search for the word of interest inside a file. We can directly find the file of interest having all the target 5-grams.

B.3.1 Creating Files and Assigning 5-grams to Files when TER = 1

We execute Algorithm 7 which creates unique files to store all the 5-grams associated with each unique words and assigns each 5-gram to the appropriate file traversing all 118 re-sorted 5-gram files. Algorithm 7 generates 2,870,385 files from the 118 re-sorted files and all these files are stored in a subfolder sf ($sf \in \{\{a \cdots z\} \times \{0 \cdots 9, ', a \cdots z\}\}$) under the folder f ($f \in \{a \cdots z\}$). So, there will have $26 \times 37 = 962$ unique subfolders under 26 folders. For example, all the 5-grams of word *brother* are in subfolder *br* under folder *b* (the path of the file is */home/*$\cdots$*/b/br/brother*).

Algorithm 7: creating unique files to store all the 5-grams associated with each unique words and assigning 5-grams to the files

```
   input  : input files      /* input files are 118 re-sorted 5-gram files */
   output: output files      /* the number of output files is equal to the
          number of unique words in the first position in 5-grams
          (first word of a 5-gram) in all 118 re-sorted files and the
          names of the output files are the unique word names */
 1 for each sorted file do
 2     for each 5-gram do
 3         word_1 ← first word of the 5-gram
 4         char_1 ← SubStr(word_1, 1)
 5         char_2 ← SubStr(word_1, 2)
 6         fname ← /home/···/char_1/char_2/word_1      /* set the location of
           file word_1 in subfolder char_2 under folder char_1 */
 7         Open fname in append mode
 8         write 5-gram to fname
 9     end
10 end
```

B.3.2 Efficient Search in the Web 1T 5-grams when TER = 1

To find all the 5-grams for a specific word of interest, we need to access[13] only a single file among those 2,870,385 files, distributed under 962 different subfolders, created using

[13]Actually we do not need to search as all the 5-grams in the file are target 5-grams.

Algorithm 7 in Section B.3.1. The good thing about these files is that each file is solely dedicated for a unique word; thus we are saving search time for 5-grams in a file. Algorithm 8 is used to locate the specific file that has all the 5-grams for a specific word and then to access all the 5-grams for that word to further process as per requirement[14].

Algorithm 8: finding the specific file and all the 5-grams for a specific word when all the 5-grams for each word is in a unique file

```
   input  : word                          /* word is the word of interest */
   output: fname    /* fname is the file that contains all the 5-grams of
           word */
1  char_1 ← SubStr(word, 1)
2  char_2 ← SubStr(word, 2)
3  fname ← /home/···/char_1/char_2/word
4  Open fname in read mode
5  for each 5-gram in fname do
6  |  do as per requirement
7  end
```

B.3.3 Comparing Efficiency in Time and Memory Space when TER = 1

We use the same Miller and Charles (1991) data set on the Dual Core Machine[15]. Table B.1 shows all the test results on Miller and Charles (1991) data set when TER = 1. Figure B.9

shows a comparison between the time (in seconds) needed to access all the 5-grams of the 39 words on the Dual Core Machine using the 22,743 files from Section B.2 versus using 2,870,385 files. It shows that all the 39 words (100%) were accessed in less than

[14]We can insert code for what we plan to do with these 5-grams in line 6 of Algorithm 8.

[15]We failed to generate 2,870,385 files on the ThinkCentre, though its operating system, Windows XP, using NTFS (New Technology File System) file system should have supported to generate $2^{32} - 1$ files maximum. Again, we failed to copy 2,870,385 files generated on the Dual Core Machine using Linux operating system to the ThinkCentre.

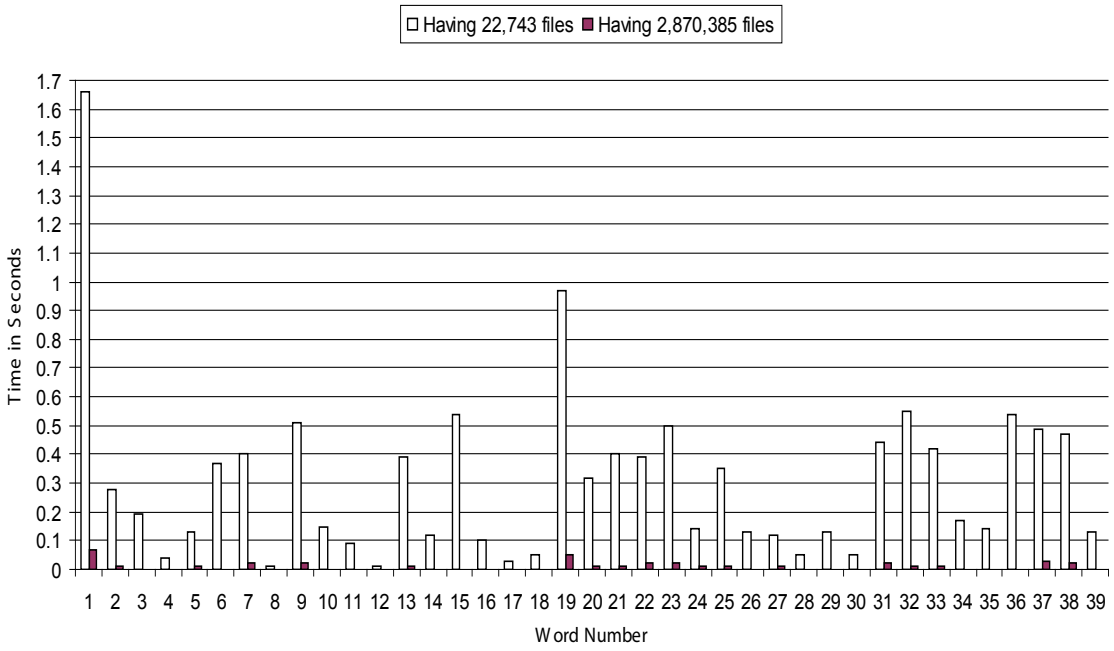

Figure B.9: Comparison between time (in seconds) needed to access all the 5-grams of the 39 words on the Dual Core Machine having 22,743 files versus having 2,870,385 files.

0.1 seconds[16]. The total access time for all the 39 words on the Dual Core Machine is 0.37 seconds, an average of 0.0095 seconds per word (i.e., 96.95% less average access time compared to using 22,743 files). The reason for having this high efficiency in access time is that we do not need to search the target 5-grams when we use 2,870,385 files, whereas using 22,743 files we need to search the target 5-grams from the searched 5-grams, even if sometimes the target 5-grams are equal or almost equal to the searched 5-grams[17]. Again, we do not need to load *filesName*, thus saving 0.03 seconds when we use 2,870,385 files.

Figure B.10 shows a comparison between time (in seconds) needed to access all the

[16]There are 20 words in Figure B.9 having 0.00 second access time as the Benchmark module of Perl returns only 2 digits after decimal point and it goes without saying that we could have had a non zero digit at the third digit position after the decimal point.

[17]To access all the 5-grams having 22,743 files needs to execute lines 13 and 14 of Algorithm 6 which are not required for the method having 2,870,385 files.

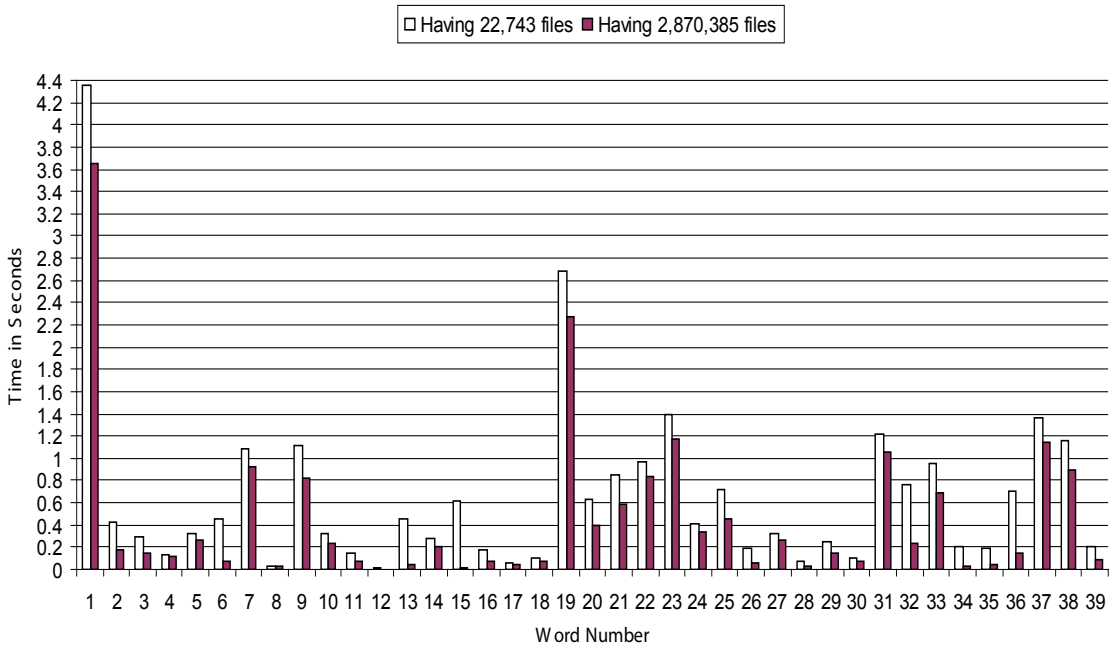

Figure B.10: Comparison between time (in seconds) needed to access all the 5-grams and then to find all the context words and the pair frequencies for all the 39 words on the Dual Core Machine using 22,743 files versus using 2,870,385 files.

5-grams and then to find all the context words and the pair frequencies for all the 39 words on the Dual Core Machine having 22,743 files versus having 2,870,385 files. The total processing time for all the 39 words on the Dual Core Machine is 18.28 seconds, an average of 0.468 seconds per word (i.e., 30.15% less average processing time compared to using 22,743 files). We get higher efficiency in time when we are to only access all the 5-grams than to access all the 5-grams and then to find all the context words and the pair frequencies. This is because to access all the 5-grams having 22,743 files requires the execution of lines 13 and 14 of Algorithm 6, which are not required for method having 2,870,385 files. But to access all the 5-grams and then to find all the context words and the pair frequencies requires to execute lines 13 and 14 of Algorithm 6 when having 22,743 files and only line 13 of Algorithm 6 when having 2,870,385 files.

Figure B.11 shows a comparison between searched 5-grams when using 22,743 files

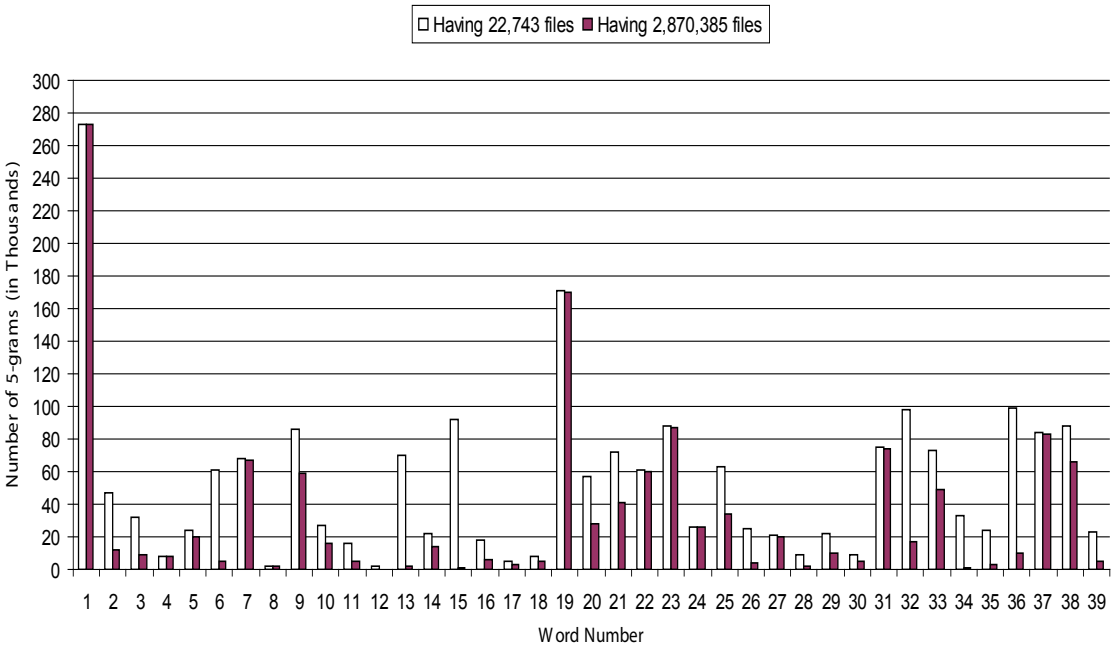

Figure B.11: Comparison between searched 5-grams using 22,743 files versus searched 5-grams using 2,870,385 files for all the 39 words on the Dual Core Machine.

versus searched 5-grams using 2,870,385 files for all the 39 words on the Dual Core Machine. All the searched 5-grams are actually the target 5-grams when we use 2,870,385 files.

The single largest file with respect to the memory size is *new* having 1,310,730 5-grams and using 32,338,060 bytes (32.34 MB) of memory when we use 2,870,385 files. So, the maximum main memory required for any word is 32.34 MB. It takes 0.39 seconds on the Dual Core Machine to access all the 5-grams and 17.12 seconds[18] to access all the 5-grams and then to find all the context words and the pair frequencies for the word *new*. We can consider these values as the upper limits.

The main memory requirement using 2,870,385 files are similar to the method using

[18]We need 95.44% less time to only access all the 5-grams and 7.06% less time to access all the 5-grams and then to find all the context words and the pair frequencies for the word *new* compared to using 22,743 files.

22,743 files except that we need not to load *filesName*, thus saving 124,262 bytes (i.e., 0.12 MB) of memory.

Table B.2 shows the actual size of the files and the size required on disk for different

Table B.2: File Size on Disk

No. of files	No. of folders	Actual size (in bytes)	Size on disk (in bytes)
118	N/A	9,483,356,102	9,483,418,624
22,743	N/A	9,483,356,102	9,495,010,304
2,870,385	988	6,865,896,655	9,514,983,424

number of files[19]. It shows that we need 0.12%, 0.33% and 0.21% increase of secondary storage when we switch from 118 files to 22,743 files , from 118 files to 2,870,385 files and from 22,743 files to 2,870,385 files. The increase of secondary storage is much smaller than the increase in the number of files. From Table B.1, it is clear that the main memory required is independent of the number of files.

B.4 Summary

It is expected that Google Inc. will release a larger data set than the Web 1T in the future and our proposed method is general enough to handle a larger data set than the Web 1T. Assuming that the rate of increase of the number of n-grams is much smaller than the rate of increase of the number of words (collected from web pages) over which their observed frequency are counted, our proposed approach is easily applicable to the future Web 10T, Web 100T or even Web 1Q (Q=Quadrillion). We are interested in the question of what is the best way of grouping n-grams to access and process these n-grams. We find that if the computing device supports millions of files, then creating unique files to store all the 5-grams associated with each unique word provides the best access time efficiency

[19]The actual file size using 2,870,385 files is less than the other two versions because we do not need to store the word of interest as the file name itself represents it; thus saving disk space.

and processing efficiency. Otherwise, the method described in Section B.2 provides a reasonable access time efficiency and processing efficiency. The main contribution of this work is that we provide a method for efficient access to a resource that is otherwise impractical to use in applications specially when context words need to be accessed. It is interesting to note that though we only experimented on the 5-gram data set, the proposed methods are also applicable to Web 1T bigrams, trigrams, and 4-grams.

In conclusion, we reorganized the 5-gram files for faster access. We measured the maximum main memory required to access all the 5-grams for any word on the fly. We also measured the maximum time required to access all the 5-grams and to find all the context words and the pair frequencies for any word, varying the number of files. The time needed to access all the n-grams containing context words for a specific word of interest depends only on the number of n-grams the word has and it is completely independent of the total number of n-grams of all the words. We measured the best access time and the best processing time that can be achieved for any word on specific computing devices.

This thesis was typeset with LaTeX 2_ε, using a modified version of the University of Ottawa Ph.D. thesis tex files, `thesis-frontpages.tex`, `thesis-preamble.tex`, `thesis-headings.tex`, `mastersource.tex`, and `uottawa-thesis.tex`.