Ben-Gurion University of the Negev

Faculty of Natural Science

Department of Computer Science

STATISTICS-BASED METHODS FOR ERRORS DETECTION AND CORRECTION IN MODERN HEBREW

Thesis submitted as part of the requirements for the

M.Sc. degree of Ben-Gurion University of the Negev

by

Yoni Lev

December 2011

Subject: Statistics-Based Methods for Errors Detection and Correction in Modern Hebrew

Writen By: Yoni Lev

Advisor: Prof. Michael Elhadad

Department: Computer Science

Faculty: Natural Science

Ben-Gurion University of the Negev

Signatures:

Author: Yoni Lev	Date
Advisor: Prof. Michael Elhadad	Date
Dept. Committee Chairman	Date

Abstract

This work focuses on the problem of correcting errors in a modern Hebrew text. Even though they have existed for a long time, error correction systems for Hebrew suffer from two main problems: They do not recognize all the valid words in our language and they fail to distinguish valid words in invalid contexts. This work includes the development of two types of methods, each designed to overcome one of the two principal problems.

The first method determines whether an unknown word is an error with a precision of 0.79 and a recall of 0.78. The samples that were used were extracted from the web and contained real errors in addition to valid words. Our method managed to distinguish between the two while achieving better results than other previous methods.

The second method uses the context in which an error appears to determine which word out of a predefined set of words was actually intended. The method was adapted from English with adjustments appropriate to a rich morphological language such as Hebrew. It achieves an accuracy of 96% over 9 different errors and 98.6% while covering 85% of the tested samples. In addition, the method was able to correct authentic common Hebrew errors in an unedited text with very high precision.

Acknowledgements

This work was accomplished with a great support from many different people. First, I would like to thank my research advisor Prof. Michael Elhadad whose guidance and help accompanied me throughout the entire process. Without his vast contribution, this work would not have completed.

I would also like to thank my colleagues and friends in Ben Gurion University. Their day to day advices helped me overcome many challenging obstacles. Particularly, I would like to thank the NLP team for their professional and patient assistance.

Finally, special thanks go to my friends and family for their endless support during the last two years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this research we focus on the problem of correcting spelling and grammar mistakes in a modern Hebrew text. The ability to correct mistakes in a text is very useful for several different applications. It makes the task of writing correctly easier, as fewer errors slip into the document. Additionally, error correction is an integral part of other common Natural Language Processing (NLP) applications such as Optical Character Recognition (OCR), machine translation, speech recognition and search engines queries.

A common program that corrects mistakes in a text is called a *spell checker*. A spell checker extracts the words from the text and checks whether they are valid. Invalid words are marked and a correction is offered for the user. In complete automatic systems the correction is made automatically without any action by the user. The first spell checkers handled only simple spelling mistakes. However, recently developed spell checking systems are capable of fixing grammatical errors as well.

Spell checkers are available for many languages, though the ones that are designed for Hebrew are either limited or proprietary. Two common systems that were developed for Hebrew are the free *Hspell* spell checker [15] and the system embedded in the commercial *Mircrosoft Word* 2010[1]. Both systems suffer from the same two problems: (1) They do not recognize all the words in our language and (2) they fail to distinguish valid words in invalid contexts. Consider the following incorrect sentence:

> *שחקתי טניס אם פדרר.
>
> **sihakti tenis im federer.*
>
> *I played tennis if Federer.

The two systems flag the word פדרר as an error, despite the fact that it is the name of a very famous tennis player. They also fail to flag the word אם as an error ,although it is clear that the user intended to write the word עם *im* (with) instead. Our main interest in this

[1]http://office.microsoft.com

work is to allow Hebrew spell checkers to cope with problems of these types.

1.1 Categorizing the errors

We can categorize the different errors in a text in several different ways. Naber [29] suggests four different categories: (1) Spelling errors - errors that result in words that do not exist in the language, e.g., רישון *rishon* (misspelling of the word "first"). (2) Grammar errors - errors that cause the sentence not to comply with the grammar of the language, e.g., חלונות גבוהות *halonot gvohot* (high windows, when the word "high" is inflected as feminine instead of masculine). (3) Style errors - using uncommon words or too complicated sentences in the given context, e.g., using slang words in a formal letter. (4) Semantic errors - sentences that contain incorrect information which is neither a style error, grammar error, nor a spelling error, e.g., אוניברסיטת בן- גוריון, תל-אביב *universitat ben-gurion, tel-aviv* (Ben-gurion university, Tel-Aviv, while it is actually in Be'er-Sheva).

Verberne [39] makes a different classification. Based on Kukich's research [22] she offers a two-dimensional classification. The two dimensions are (1) the cause of the error and (2) the result of the error. The cause can be either a typing error, a cognitive-phonetic lapse, or a grammatical mistake. The result can be either a non-word error - error that results in a word that does not exist in the language, e.g., רישון *rishon* (misspelling of the word "first"); a real-word syntactic error - an error that results in a real word and leads to an ungrammatical sentence, e.g., על/אל תחצה את הכביש *al/al tahze et ha-kvish* (on/don't cross the road); a real-word semantic error - an error that results in a real word and leads to a grammatical sentence but with a different meaning from the one intended, e.g., הוא אכל ברק/מרק *hu ahal barak/marak* (he ate a lightning/soup); a *structural error* - an error that violates the inherent coherence relations in a text, e.g., יש לי שני ילדים: אברהם, יצחק ויעקב *yesh li shnei yeladim: havrahm, itzhak ve-yaakov* (I have two children: Abraham, Isaac and Jacob); or a *pragmatic error* - an error that reflects some anomaly related to the goals and plans of the discourse, e.g., באר-שבע נמצאת בצפון *beer-sheva nimtzet be-tzfon* (Be'er- Sheva is located in the north, while it is actually in the south). Pragmatic and structural errors are remain beyond the scope of Verberne's research as they reflect an error of thought and not necessarily an unintended error.

In this research we will categorize the errors by relying on the classification made by Verberne [39]. However, in contrast to her work, we will not divide the errors according to their cause since it is not always possible to determine it conclusively and a classification in this manner will lead to an ambiguous result. For example, if the writer used the word אם *im* (if) instead of עם *im* (with), we can classify it as a phonetic mistake (because the two words sound the same) and as a typing mistake (because the letter א is next to the letter ע on the keyboard).

We will divide the different errors into two main categories: (1) *non-word errors*, i.e., errors that result in words that do not exist in the language; and (2) *real-word errors*, i.e., errors that result in words that do exist in the language. Real-word errors will be classified as *syntactic real-word errors* or *semantic real-word errors*.

Syntactic real-word errors refer to errors that result in a real word and lead to an ungrammatical sentence. We can distinguish two types of syntactic real-word errors according to the *Part Of Speech* (POS) of the resulting word. A *Part Of Speech* (POS) is "a traditional class of words distinguished according to the kind of idea denoted and the function performed in a sentence"[2], e.g., the POS of the word "airplane" is Noun. A real-word error may lead to an ungrammatical sentence if (1) it has an unacceptable POS according to the language rules, e.g., על/אל תחצה את הכביש *al/al tahze et ha-kvish* (on/don't cross the road); or if (2) it has an acceptable POS but the word is inflected improperly according to the language rules, e.g., חלונות גבוהות *halonot gvohot* (high windows, when the word "high" is inflected as feminine instead of masculine). Semantic real-word errors refer to errors that result in a real word and lead to a grammatical sentence but with a different meaning from the one intended, e.g., הוא אכל ברק/מרק *hu ahal barak/marak* (he ate a lightning/soup).

In cases where we cannot determine unequivocally whether the sentence is ungrammatical or has a different meaning from the one intended, we will not classify the real words in the sentence as errors. Finally, the task of correcting pragmatic errors and structural errors as defined above requires further study and lies outside the scope of this work.

1.2 Non-word errors

Non-word errors are errors that result in words that do not exist in a language. Kukich [22] distinguished three different types of non-word errors: (1) typographic errors (2) cognitive errors and (3) phonetic errors. Typographic errors are errors that accrue due to incorrect typing, e.g., כדוגל/כדורגל *kadugel/kaduregel* (footall/football). The assumption is that the writer knows the correct spelling of the word but simply fails to type it correctly. Cognitive errors are errors that arise due to misconception or lack of knowledge of the writer, e.g., לוחים/לוחות *luhim/luhut* (boards - validly inflected)/(boards - invalidly inflected). In this type of error the writer does not know the correct spelling of the word. Phonetic errors are a subcategory of cognitive errors. They refer to errors in which the intended word (the correct word) sounds exactly like the used one, e.g., רישון/ראשון *rishon/rishon* (first/misspelling of the word "first"). We are not always able to make the distinction between the different

[2]Merriam-Webster online dictionary

categories. When a writer writes רישון instead of ראשון, we can not determine unequivocally that the mistake occurred because the writer mistyped the word or because he genuinely thought this is how the word is spelled.

1.2.1 Non-word errors detection

There are two main approaches to detection of non-word errors [26][32][22]. The first one is simply to check every word in a *dictionary.* A dictionary (or a *lexicon*) is a list of words that are the vocabulary of a system's language. If a word is not in the dictionary it is considered as a potential error. It is not mandatory to keep every word in the dictionary. Natural languages follow certain rules, and by these rules different words can be derived from other words, e.g., the words *doubts*, *doubted* and *doubting* can be derived from the word doubt by adding the suffixes "s", "ed" and "ing" respectively [24].

As for Hebrew, a language with a productive morphology based on relatively simple rules, it would be redundant to keep all the inflected forms of all lexemes in the dictionary [41]. Some of those morphological rules allow words to appear with several prefix particles [20]. These include the definite article ה *ha* (the), e.g., הילד *ha-yeled* (the boy); prepositions such as ב *be* (in), כ *ke* (as), ל *le* (to) and מ *me* (from), e.g., בעיר *be-ir* in a city; subordinating conjunctions such as ש *she* (that) and כש *kshe* (when), e.g., כשקפצתי *kshe-kafazti* (when I jumped); relativizers such as ש *she* (that), e.g, הילד שראיתי *ha-yeled she-raiti* (the boy that I saw); the coordinating conjunction ו *vav* (and), e.g., וילד *ve-yeled* (and a child). A word might have several prefix particles, e.g., ושמהים *ve-she-me-ha-yam* (and that from the sea), which leads to millions of possible tokens in a text. Instead of maintaining a dictionary with millions of entries, the *Hspell* spell checker version 1.0 [15] consists of only 329K words within their category of possible prefix particles and a rather small list of possible prefixes for each category under several constraints. This technique allows the system to recognize more than 38M different tokens. Another example is the *Haifa Lexicon of Contemporary Hebrew* [20][19] which holds only 22K base form words with their morpho-syntactic features and their lexical properties. By using the additional information of each base form word over 473K words can be derived and by adding prefix particles more than 100M valid tokens can be created.

Most of the systems that detect errors by using a dictionary allow the addition of new words to the word list. This is due to the fact that the vocabulary of natural languages is growing constantly and new words (mainly proper names) are being used frequently in a common text. For example, when we analysed an arbitrary text consisting of 333K words instances (words with two or more Hebrew letters) taken from articles in *Walla* website[3], nearly 21K were not in the *Hspell* version 1.0 [15] lexicon (6.2%). Another observation was

[3]http://www.walla.co.il

that out of all the different words types in our *Walla* corpus (words with two or more Hebrew letters), 33% were not in the lexicon. Adler [1] reported that in his newswire corpus 4.5% of the 27M token instances were unknown tokens, i.e., were not in the lexicon. He added that for less edited text, such as random web text, the percentage is about 7.5%. Goldberg and Elhadad [11] found genres of Hebrew text where nearly 5% of the word instances are borrowed from foreign languages and appear in their *transliterated* version, i.e., the letters were changed systematically from a foreign language to Hebrew. Most of these words will not appear in a common dictionary. Systems that do not modify their dictionary in line with language development will mistakenly flag intended words as errors.

The second approach to detection of non-word errors is to use *n-grams* analysis. A simple n-gram is a sequence of *n* letters in a word or *n* words in a sentence. 1-grams are referred to as unigrams, 2-grams as bigrams and 3-grams as trigrams. For example, the word "peace" contains the unigrams "p", "e", "a","c", "e"; the bigrams "pe", "ea", "ac", "ce"; and the trigrams "pea", "eac", "ace". A *positional n-gram* is a n-gram that includes also the positions of the letters in the word. The letters in the positional n-gram do not have to be consecutive. For example, the positional bigram of the word "peace" at positions 2,4 is "ec".

In general, n-grams analysis methods work by checking if each simple or positional n-gram in a word is statically reasonable. Words that contain infrequent n-grams are considered potential errors. The frequencies of the n-grams can be determined by counting their appearances in a corpus [42], in a dictionary [35], or even in the examined document itself [28].

1.2.2 Non-word errors correction

Correcting non-word errors is not a simple task, as the system has to guess the intended word in the original sentence. The problem consists of three sub problems [22]: First, as discussed earlier, the error has to be detected. Then a set of possible corrections is generated. Finally, ranking the possibilities has to be done in order to decide what is the most probable correction. Some of the systems leave the decision to the writer and give him the entire list of possible corrections as a suggestion for amending the error. Complete automatic systems will replace the error with the most probable correction from that list.

Most of the prior error correcting techniques focused on isolated words [43], i.e., not taking into account semantic or syntactic information from the text apart from the word itself. These various techniques use string matching algorithms, hashing based on different properties, n-grams analysis, explicit error correction rules, probabilities derived from corpora and/or other supervised machine learning methods. For further details see Kukich's survey [22]. Clearly this approach, which ignores the context of the misspelled word, will be limited. For example, the following sentences contain the same non-word error that resulted from different words (the misspelled word is before the slash and the correct word is after):

- הלכתי ביום רישון/ראשון

  halahti be-yom rishon/rishon

  I went on Sunday

- אין לי רישון/רישיון

  ein li rishon/rishayon

  I don't have a driving licence

- התקן רישון/ריסון המתאים לילד

  hetken risun/risun matim le-yeladim

  Appropriate restraint device for children

If the system chooses the correction only on the basis of the misspelled word itself, the correction of the non-word רישון in all the above sentences will be the same, and for at least two of the three sentences it will be wrong. Nevertheless, Kukich reports that "...while all techniques fall short of perfect correction accuracy when only the first guess is reported, most researchers report accuracy levels above 90% when the first three guesses are considered". Given this report, the next step for improving an isolated word system would be exploiting the contextual information of the misspelled word in order to make the selection of the correct word from the top ranked correction candidates.

Church and Gale [6] reported such an improvement when ranking the possible corrections using the immediate context of the misspelled word. Their system used only the word before and the word after the misspelled word to estimate the context conditional probabilities of each possible correction. The different corrections were ranked using these probabilities and a probability that was given to the correction word itself.

1.3 Real-word errors

Real-word errors are errors that occur when a correctly spelled word replaces the intended word. The result can be either an ungrammatical sentence (a syntactic real-word error) or a grammatical sentence with a different meaning from the one intended (a semantic real-word error). Here are a few examples of real-word errors that were found on the web (the error is before the slash and the correction is after):

- הסמים הוחזקו בתוך גרב שחורה/שחור

  ha-samim huhzeku betoh gerev shhora/shahor

The drugs were kept in a black sock (the word "black" is inflected as feminine instead of masculine)

- אז מה עם/אם הוא שיחק במכבי

  az ma im/im hu sihek be-macabi

  So what with/if he played for Maccabi

- תודה אל/על הפרגון

  toda al/al ha-firgun

  Thank you to/for the favour

The first sentence contains a syntactic real-word error while the last two contain a semantic real-word error.

We [34][18] can distinguish six different types of reasons for writing a different word from the one intended:

1. **Typographic:** the replacement word resulted from a typing mistake.
2. **Phonetic:** the replacement word sounds like the intended one.
3. **Grammatical:** the replacement word resulted from lack of grammar knowledge.
4. **Frequency disparity:** the replacement word occurs very frequently.
5. **Learners' errors:** the replacement word resulted from an inter-linguistic disparity.
6. **Idiosyncratic:** the replacement word resulted from an unknown reason.

Similarly to non-word errors, determining the reason for the error can be problematic as we can associate a given real-word error to several categories, e.g., writing the word הוא *hu* (he) instead of the word היא *hi* (she) can be classified as a grammatical mistake (wrong gender inflection) and as a typographic mistake (the letter ו is next to the letter י on the keyboard).

Systems that do not consider words that are in the dictionary as potential errors will fail to flag real-word errors. In order to detect such errors, a system can not rely only on a list of words, rather it has to use also the surrounding context of the words. The term used for these kind of systems is *context sensitive systems*.

1.3.1 The frequency of real-word errors

Several studies have shown that these kinds of errors are surprisingly common. According to Peterson [33], almost one out of six mistyped word will be a real word. His research is based on the assumption that most typing mistakes are caused by transposition of two adjacent letters, one extra letter, one missing letter, or one wrong letter [7]. He artificially inserted these four types of errors into every possible position in common words and counted how many of them resulted in a real-word. We preformed a similar experiment to see if the problem arises with modern Hebrew. We counted how many artificial misspellings of 10K arbitrary words were real words. The dictionary that we used to determine if a string was a real word was generated automatically from the *Hspell* project[4]. The words were taken with replacements from articles in *Walla*. We did not include words that were not in our dictionary. The misspellings were created according to the common four types of typing mistakes, by first allowing the insertion of every extra letter and the substitution of a letter with every other letter. Later on we restricted the insertion of an extra letter and the substitution of a letter with another only to adjacent letters on the keyboard. In both runs the results were even more radical than Peterson's, with real-words ratios of 0.178 and 0.188 respectively.

In another research a corpus of short compositions written by pupils at age 15 was assembled [31]. In this corpus about 40% of the misspelled words were real words [25]. In another corpus of essays, written by students as applications [40], the result was 30% [22], while in two other studies of children's compositions [37][3] the results were 26% and 29% respectively, as reported by Mitton [26].

Mitton [26] concludes that "real-word errors account for about a quarter to a third of all spelling errors" and according to Kukich [22] they can be even more than 50% of all textual errors in some applications.

1.3.2 Real-word errors detection and correction

There are three main approaches to detection and correction of real-word errors [29][36][5]: (1) syntax based, (2) rule based and (3) statistics based.

Syntax Based In a syntax based system a syntactic analysis is performed for each sentence, i.e., the grammatical tree structure of each sentence is determined. With the use of this analysis, ungrammatical sentences can be detected. The task of determining the grammatical tree structure of a sentence is called *parsing* and a program that performs it is called a *parser*.

An example of such a system is the one that was developed by IBM [16] in the early eighties. Their system searches for grammar anomalies by parsing each sentence in the text. With a set of grammar rules the system tries to determine the syntactic parts of each sentence. If the system fails to parse the sentence it relaxes some of the rules and tries to parse it once again. This process is repeated until the system can parse the entire sentence. A sentence that the system fails to parse without relaxing some of the rules is considered

[4]http://ivrix.org.il/projects/spell-checker - Hspell version 1.0 was used

[5][29][36] attribute these approaches to an implementation of a grammar checker, i.e., an implementation of a comprehensive program, with part of its task being to detect and correct real-word errors.

ungrammatical. In addition, by knowing which rules are being relaxed and how, the system can suggest corrections.

Rule Based Rule based systems detect errors by matching the given text to a set of rules. The set of rules are developed manually by using linguistic and common errors knowledge. These rules may include invalid words or POS patterns, style violation rules or any other linguistic rules set by the developers. If a sentence violates one or more rules, the error is flagged and the correction can be inferred from those rules. For further review see [29].

Statistics Based Statistics based systems use a large corpus of well written text to infer the probability that a given sequence of words is valid. The probability can be calculated from the corpus in a straightforward way by counting n-grams appearances or by using these n-grams as positive or negative examples in the process of machine learning. The guideline for this approach is that an uncommon or a never seen n-gram might be an error when observed in a given text.

Correcting the error can be done by finding a more frequent n-gram than the one used, that is still suitable in the given context. The n-grams may be established from the words themselves and/or from other lexical features extracted from them.

Mays et al. [23] addressed the problem by using words trigrams. They calculated the probabilities of a each dictionary word given a sequence of two previous dictionary words. The product of the trigram probability of each word in the sentence multiplied by a factor for general level of errors expectation in the text gave a probability score for each sentence. Their method replaced one word in the sentence with another dictionary word by means of a single character transformation (insertion, deletion substitution or transposition) and calculated the score for all the possible sentences. The sentence with the highest score was considered as the correction for the original one. They tested their method on 8,628 sentences containing one real-word error that was created artificially by a single character transformation. The results gave a detection score of 76% and correction score of 74%. For a comprehensive review of the earlier works see [22] and [27].

The advantage of a syntax based system is that with an accurate parser it will detect all the ungrammatical sentences. However, a problem arises when a sentence contains semantic real-word errors. These errors will be left undetected because the sentence remains grammatically correct and the parser will be able to parse it.

Naber [29] points out many advantages of the rule based approach: The system can check the text while it is being typed; it can be designed in such a way that rules can be turned on and off individually; the system has the ability to offer a message explaining the error because it can tell which rule was violated; it is possible to allow users to create their own set of rules; and it can be built incrementally: starting with just one rule and then extending it rule by rule. However, there are also several fundamental downsides to this approach. According to Naber himself [29], because the rules are generated manually "a rule-based checker will never be complete... there will always be errors it does not find". Generating a comprehensive set of rules for languages with a complex grammar system and many irregularities will be a particularly difficult task. In addition, every language has its own set of rules, which makes an existing rule based system suitable for only the language it was designed for.

As for statistics based systems, Naber [29] claims that they "bear the risk that their results are difficult to interpret", which makes it difficult to explain the error after detecting it. Another problem that Naber raises is that in this approach "someone has to set a threshold which separates the uncommon but correct constructs from the uncommon and incorrect ones". According to him, the idea of using a threshold is inconsistent with the perception that sentences are usually either correct or incorrect. On the other hand, this approach is language independent, i.e., a given algorithm can be applied for different languages by changing the corpus that the system uses. Additionally, the rules of detection and correction are automatically generated from a corpus and they are likely to encompass more cases than manually generated ones.

In this work we will address the problem of detecting and correcting real-word errors using a statistics based approach. Nowadays, most of the NLP tools use statistical machine learning algorithms combined with large scale corpora, which yields extremely good results. In addition, this technique is much more general and comprehensive than the rule based or the syntax based techniques.

The Confusion Sets Approach

A common approach to the problem is to refer to it as a word disambiguation task [13], when the ambiguity among words is modelled by *confusion sets*. A confusion set is a set of words that are likely to be confused with each other, e.g., the words אם *im* (if) and עם *im* (with). When the examined document contains a word from a confusion set the task is to decide which word from that confusion set was actually intended. Choosing the correct word can be done statistically according to the observed context, i.e., a system can be trained over well formed sentences that contain the confusion set words to infer which word is the most probable for the observed context.

Most of the works on the confusion set problem have focused on a small number of confusion sets, when each set contained only two or three words [30]. These confusion sets were selected manually on the basis of their frequency in use and included homophone errors (e.g., peace, piece), grammatical errors (e.g., among, between) and typographic errors (e.g., maybe, may be).

Creating confusion sets can be done automatically as well. Fossati and Di Eugenio [8] used a large set of word/misspelled pairs and a set of features to estimate the probability that a word e was transformed into a word w because of a spelling mistake, i.e., $P(w|e)$. For each dictionary word w they created a confusion set C_w assembled from every other dictionary word e that yielded $P(w|e)$ greater than a predefined threshold. Pedler and Mitton [30] used a string matching algorithm to measure the similarity of two dictionary words. For each dictionary word w, they created a confusion set C_w out of all the dictionary words whose value of similarity to w exceeded a predefined threshold. The result of the two methods was thousands of asymmetrical confusion sets, i.e., if a word e is in the confusion set C_w it does not imply that w is in the confusion set W_e. When using these type of confusion sets, the selection of the intended word is made from the confusion set that corresponds to the observed word. Thus, the selection is made from only the words that might have led to this particular observed word.

1.4 Modern Hebrew

Modern Hebrew is one of the two official languages spoken in the state of Israel, along with Arabic. Today, it is spoken by 7M people, making it the mother tongue for about half of the population. Some aspects of modern Hebrew make NLP tasks very challenging [12]. We will elaborate on those that are relevant for our work.

Auffixation As mentioned above, modern Hebrew allows words to appear with several different prefix particles, such as prepositions, conjunctions and articles. In addition pronominal elements often appear as suffixes. This auffixation property might lead to a relatively high number of token types in a text. For example, consider a text which contains only the following words: בית *bait* (house), הבית *ha-bait* (the house), כלב *kelev* (dog) and הכלב *ha-kelev* (the dog). The number of token types in Hebrew is 4, while in English it is 3. We compared the number of token types in an arbitrary news paper text written in Hebrew and in English. The text was taken from Ha'aretz newspaper[6] and from newspaper text from the Brown corpus (Brown University Standard Corpus of American English). Out of 100K token instances, the Hebrew text contained 24K token types, while the English text contained only 13K token types. The high number of token types in a text leads to data sparseness, which makes the task of inferring a word from its context more difficult.

[6]http://www.haaretz.co.il/

Morphology Hebrew has a very productive morphological structure, which is based on a root+template system. According to Goldbelg and Elhadad [12], it results in many distinct word forms and a high out-of-vocabulary rate. Hence, assembling a comprehensive dictionary for Hebrew is not an easy task, and additional techniques for detecting non-words errors should be considered.

Script Hebrew script has two main forms: *dotted* and *undotted.* The dotted Hebrew script contains diacritical signs (*Niqud*) used to represent vowels or to distinguish between alternative pronunciations of words. The undotted script, which is mostly used nowadays, does not contain these signs. Instead, additional letters represent some of the vowels. The undotted writing system yields many homographs words, i.e., different words with similar spelling. For example, the string אם represents five different words in Hebrew[7]: (1) *im* - if; (2) *em* - mother; (3) *em* - the letter M; (4) *om* - nut; (5) *om* - nation. Adler [1] reports an average of 2.7 possible morphological analyses per token in Hebrew, while only 1.4 in English. The multiple meanings of a token increases the number of different contexts in which that token can appear, thus making the confusion set disambiguation task more complex.

Grammar Hebrew grammar forces morphological agreement between Adjectives and Nouns, which should agree in gender, number and definiteness. These agreement rules open the door to many potential syntactic real-word errors. For example, the collocation גרב שחורה *gerev shhora* (black sock, when the Adjective "black" is inflected as feminine) is unacceptable in Hebrew, because the word "sock" in Hebrew is a masculine Noun. The collocation העורך דין *ha-oreh din* (the lawyer) is also illegal in Hebrew, since this collocation is a construct state (*Smichut*) and the definite article ה can appear only with the second noun. These types of mistakes are very common in Hebrew, even among native speakers.

The field of error detection and correction in Hebrew falls short in comparison to other more common languages such as English. As far as we know, this is the first study to evaluate the computational ability to handle non-words and real-word errors in Hebrew. Accommodating the above methods to Hebrew, when they were originally developed for English, can be a rather challenging task, especially with the interesting linguistics features discussed in this section.

[7]according to the *Rav-Milim* online dictionary http://www.ravmilim.co.il/

Chapter 2

Objectives

As discussed above, common Hebrew spell checkers suffer from two main problems: (1) They falsely flag real-words that do not appear in their dictionary as errors and (2) they do not detect real-words errors. Our goal in this work is to improve the performance of a spell checker for both these cases. More specifically, we will create and evaluate two types of classifiers for non-word errors and for real-word errors that can be embedded into a simple spell checker. We will examine the performance of each classifier on arbitrary text containing real errors, i.e., errors introduced inadvertently by real people.

Non-word Error Classifier Common spell checkers flag non-word errors based only on a predefined dictionary. Each word in the observed document that does not appear in this list is flagged as an error. This leads to many false alarms by the spell checker, since a lot of these words are in fact valid words. As we reported earlier, the percentage of out of dictionary words in a modern text can reach more than 7%. A simple system will incorrectly mark as errors all these words. We aim to develop a classifier that given a word that does not appear in the dictionary will classify it as an error or as a valid word based on morphological properties.

Real-word Error Classifier Our second purpose in this work is to classify real-word errors. Real-word errors account for 25% to more than 50% of the errors in some applications [22][25]. Spell checkers that use only a dictionary for error detection will fail to detect these errors. We wish to apply the confusion set approach to modern Hebrew and to examine its effectiveness. We will develop a classifier that, given a small set of words and a sentence in which one of those words appear in, selects from the set the word that was intended.

Chapter 3

Resources

3.1 Dictionary

We used the word list of the *Hspell* spell checker version 1.0 [15]. The word list consists of 329K words with their category of possible prefix particles and a list of possible prefixes for each category under several constraints, which yields more than 38M different tokens. This word list was designed to be compliant with the official spelling rules published by the Academy of the Hebrew Language (the supreme institute by law for the Hebrew Language) for the undotted Hebrew script.

3.2 Morphological Analysis

We used the morphological analysis system developed by Adler [1]. Given a Hebrew text, the system assigns a full set of morphological features for each word, extracts noun phrases, and recognizes entity names (persons, locations, organizations, temporal and number expression). The system achieves accuracy of 93% for word segmentation and POS tagging.

3.3 Transliterated Classifier

Transliteration is the process of changing a word into corresponding letters of another language. We used the method of Goldberg and Elhadad [11] to identify transliterated words in Hebrew text. Their method is context free and works on words alone. It achieves 80% precision and 82% recall.

3.4 Corpora

This section describes the two corpora we used as part of this work. The two corpora were assembled from text found in the *Walla* website and were both analysed morphologically using Adler's system [1].

3.4.1 The Walla Corpus

The Walla Corpus consists of arbitrary articles and news taken from the *Walla* website. This text is edited and written in modern Hebrew. The corpus contains about 31M token instances and 497K token types.

3.4.2 The TB Corpus

The TB Corpus consists of arbitrary *TalkBacks* taken from the *Walla* website. A TalkBack (TB) is a short response written by a web user to an article, a story, a blog or a news report. The TB system in the *Walla* website does not check for errors in the text, thus the text is unedited and contains many spelling, grammar and style errors. Because many of the TBs are just short phrases, we excluded TBs with less than ten words. The corpus contains nearly 2M token instances and 148K token types.

Corpus	TInst	TDicInst	TInstRatio	TType	TDicType	TTypeRatio
Walla	31,138,698	23,662,568	0.76	497,782	304,333	0.61
TB	1,971,987	1,576,063	0.80	147,749	98,843	0.67

Table 3.1: Statistics of the corpora used in this work.

The statistical information of the tokens in these corpora is summarized in Table 3.1. **TInst** - number of token instances; **TDicInst** - number of token instances that are in the dictionary (see Section 3.1); **TInstRatio** - the ratio between TDicInst and TIns, i.e., the probability that a token instance is in the dictionary; **TType** - number of token types; **TDicType** - number of token types that are in the dictionary; **TTypeRatio** - the ratio between TDicType and TType, i.e., the probability that a token type is in the dictionary.

Figure 3.1 illustrates difference between the two corpora by comparing the frequencies of the most common POS tags as annotated by the morphological analysis system (see Section 3.2).

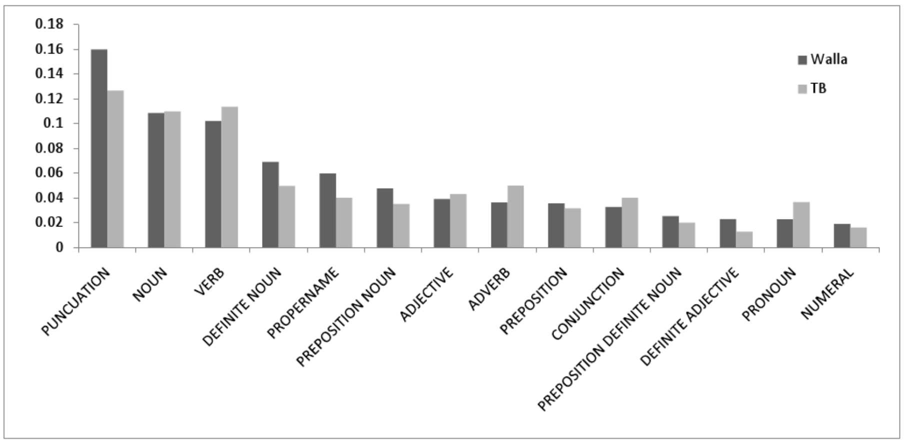

Figure 3.1: Frequencies of the most common POS tags in the used corpora.

Chapter 4

Learning Methods

The concept of learning refers to the ability to identify the class of a new sample based on a training set of data. A learning algorithm receives as input training data and gives as output a function that classifies new samples to their most probable class, i.e., *a classifier*. When training data consist of samples labelled with their known class, the process is called *supervised learning*. Learning with unlabelled training data is called *unsupervised learning*.

In the following sections we will describe three supervised learning methods that are used in this work.

4.1 Naive Bayes

Given an example x, we can describe it by a set of feature variables $F_1, F_2, \ldots, F_n$. A probabilistic classifier will classify the example according to the conditional probability $P(C|F_1, F_2, \ldots, F_n)$, when C is the class variable. A *Naive Bayes classifier* is a probabilistic classifier that makes use of the Bayesian theory with a strong (naive) independence assumption - every feature F_i is conditionally independent of every other feature F_j. According to the Bayes' rule:

$$P(C|F_1, F_2, \ldots, F_n) = \frac{P(C)P(F_1, F_2, \ldots, F_n|C)}{P(F_1, F_2, \ldots, F_n)} \tag{4.1}$$

$P(F_1, F_2, \ldots, F_n)$ does not depend on C and therefore is considered as constant. If we apply the definition of the conditional probability to Equation 4.1 we can rewrite it as:

$$P(C|F_1, F_2, \ldots, F_n) = \alpha P(C)P(F_1|C)P(F_2|C, F_1)\ldots p(F_n|C, F_1, F_2 \ldots F_n - 1) \tag{4.2}$$

From the conditional independent assumption follows that $P(F_i|C, F_j) = P(F_i|C)$ for every $i \neq j$, thus Equation 4.2 can be rewritten as:

$$P(C|F_1, F_2, \ldots, F_n) = \alpha P(C) \prod_{i=1}^{n} P(F_i|C) \tag{4.3}$$

Usually, a Naive Bayes classifier will select the class c that maximizes the right hand size of Equation 4.3. According to this rule, the classification of a sample x with features values $F_1(x), F_2(x), \ldots, F_n(x)$ can be written as the following function:

$$\begin{aligned} Classify(x) &= Classify(F_1(x), F_2(x), \ldots, F_n(x)) \qquad (4.4) \\ &= \arg\max_{c} P(C = c) \prod_{i=1}^{n} P(F_i = F_i(x)|C = c) \end{aligned}$$

The probabilities $P(F_i|C), P(C)$ can be estimated from the training set using the following equations:

$$P(C = c) = \frac{\text{number of samples labelled as c}}{\text{number of samples}} \tag{4.5}$$

$$P(F_i = f_i|C = c) = \frac{\text{number of examples labelled as c with feature } F_i = f_i}{\text{number of examples labelled as c}} \tag{4.6}$$

According to equation 4.6, if an unseen sample has a feature F_i with a value f_i that was not seen in the training set then $P(F_i = f_i|C = c) = 0$ and thus $P(C|F_1, F_2, \ldots, F_n) = 0$. To overcome this problem there are several smoothing techniques [10].

4.2 SVM

In its base form the *Support Vector Machine* (SVM) is a binary classification algorithm that given a sample will classify it into one of only two possible classes. The samples are represented as vectors by predefined features. Given a set of training samples labelled as one of two classes, the SVM searches for a boundary between the two groups of vectors according to their spatial location. This boundary will function as a decision threshold for classification of new samples.

The simplest version of the SVM algorithm separates the vectors with a hyperplane, i.e., *linear classification*. Among the possible hyperplanes that can divide the vectors, the one with the maximal gap between the two groups of vectors will be selected, i.e, the hyperplane with the maximal *margin*. Vapnik [38] showed that increasing the margin reduces the probability for misclassifying an unseen sample.

The SVM algorithm can also be applied in cases where linear separation can not be carried out without an error. In such cases the SVM uses some fixed mapping, known as a *kernel*, to map the vectors into a different space with a potentially higher dimension. This transformation allows the algorithm to separate the vectors in a different space. The transformation may be non-linear, which can lead to a non-linear separation in the original space, i.e., *non-linear classification*.

Another technique to separate training data that is not linearly separable is to use *soft margins*. Soft margins permit misclassification of some training samples by the hyperplane. In this modification, the model uses a cost parameter for the misclassified samples in the process of selecting a maximum-margin hyperplane. This parameter controls the trade-off between allowing training errors and finding a large-margin hyperplane.

Extending the SVM algorithm for multi-class classification can be done by combining several binary SVM classifiers or by considering all data in the optimization formulation. Previous work by Hsu and Lin [17] shows that combining binary classifiers may be more useful for practical use.

4.3 Perceptron

The *Perceptron* is a binary classifier inspired by the structure and the functionality of a single nueorn. The classic Perceptron classifies a sample, represented by a real-valued feature vector x, to a class $y \in \{-1, 1\}$ by using a hyperplane:

$$Classify(x) = \begin{cases} 1 & \text{if } w \cdot x + b \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{4.7}$$

Where w is a vector of real-valued weights, b is a bias constant and $w \cdot x$ is the dot product of w and x.

The training of the Perceptron starts with an initial weights vector w. It predicts the class of a new sample vector x to be $\hat{y} = Classify(x)$. If the prediction is different from the correct label y, the weights are updated as follows: $w \leftarrow w + yx$. The weights will remain unchanged if the prediction is correct. This process is repeated for every sample in the training set. When the training data is linear separable, repeating this training algorithm in cycles will converge to a weights vector which classifies correctly all of the training samples.

Freund and Schapire [9] algorithm, the *Voted Perceptron*, computes the weights when the data are not separable or if one does not want to wait until convergence is achieved. The training of the Voted Perceptron is similar to the training of the Perceptron, only in addition to the latter's algorithm the Voted Perceptron keeps all the weights vectors that were computed during training with the number of iterations that they "survived", i.e., the number of correct classifications until a misclassification. The prediction of a sample after training is done by voting when each weights vector votes for a class by using Equation 4.7 and the votes are weighted by the survival time of this vector.

Multi-class classification with the Voted Perceptron can be done by training several binary classifiers and combining their predictions into a single prediction [9].

Chapter 5

Non-word Error Classifier

In this chapter we will describe and evaluate a non-word error classifier. Given a word which is not in the dictionary, a non-word error classifier determines whether the word should be flagged as an error.

5.1 Related Works

We will elaborate on three methods for detecting non-word errors using ngrams analysis. These methods do not detect errors by looking up the words of the examined text in the dictionary. For further review see [22][27].

Riseman and Hanson [35] used positional n-grams to detect and correct non-words. They partitioned a dictionary into several sub-dictionaries according to the length of the words. For each sub-dictionary of l letters words and for each of the possible letters positions i,j they created a binary matrix D^l_{ij}. The value $D^l_{ij}[k][h]$ was defined as one only if there was a word in the sub-dictionary of l letters words that contained the letter k in the ith position and the letter h in the jth position. Given a word $w{=}c_1c_2\cdots c_n$ their method classified it as an error if there were i,j such that $D^n_{ij}[c_i][c_j] = 0$. Similarly, they applied their method using positional trigrams which led to better results. Their method was able to detect 98.6% of the errors in their test set.

Zamora et al. [42] used a collection of 50K word/misspelling pairs to compute the error probability of a trigram. They defined it as $P = E/(E + V)$ where E is number of times the word of the trigram was a misspelling and V is the number of times it was a valid word. Words that contained two adjacent trigrams whose error probabilities exceeded a predefine threshold were considered errors. They found that the error probability did not distinguish effectively between the valid words and the misspellings.

Morris and Cherry [28] generated a table of trigrams frequencies from the given document, i.e., the document in which they searched for errors. They used this table to compute an "index of peculiarity" for each word in the given document. The index of a word was defined as the root mean square of the indices of each trigram of the token. The index for a trigram xyz given the bigram frequencies $f(xy), f(yz)$ and the trigram frequencies $f(xyz)$ was defined as $\frac{1}{2}log(f(xy)-1)+\frac{1}{2}log(f(yz)-1)-log(f(xyz)-1)$. Their experiments indicated that misspelled tokens tend to have high indices of peculiarity, thus detecting errors can be done by setting a threshold for separating valid words from non-words. A word will be considered an error if its index of peculiarity is higher than this threshold.

5.2 Method

Our non-word error classifier is based on the SVM algorithm. We used the *LIBSVM* library [5], which is a free implementation of that algorithm and has many features. The library provides a simple Python interface where we could easily link it with our program.

In order to train and evaluate this supervised classifier we created a set of non-dictionary words. The words were taken arbitrarily from the TB corpus without repetitions and were annotated manually by a native Hebrew speaker as non-words or as real words. We chose only words that met the following conditions:

- The word has more than one letter.
- The word does not contain non-Hebrew letters.
- The word does not contain a digit.
- The word does not contain illegal characters, e.g., *"!"*, *"@"*, *"#"*, *"%"*, *"?"*, *","*[1]

Words that do not follow these conditions can be easily classified with a set of simple rules. The final set contained nearly 7K annotated words.

Total Words	Non-words	Real-words	Non-words Ratio	Real-words Ratio
6826	3543	3283	0.52	0.48

Table 5.1: Statistics of the annotated non-dictionary words set

We examined the effect of different types of features on the classifier. The features included the length of the word, the bigrams and trigrams of the word, the positions of the letters in the word, the number of occurrences of each letter in the word, the POS of the word and the classification of the word as transliterated. Our hypothesis was that many transliterated words and proper names might not appear in the dictionary but would be considered real words. In addition, we believed that many non-words would have unreasonable letters sequences or placements while some real-words would have similar letters patterns. We

[1] We allowed words with *"""*, *"."*,*"’"*, as they can be part of a real-word.

trained several classifiers with different features and we selected the features that yielded the best result. Those features are summarized in Table 5.2 with an example according to the word טובב *tovv* (misspelling of the word "good"). Only features with positive value appear in the example column.

Feature	Description	Example
length<w>	the length of the word w normalized by 20	length<טובב>=0.2
count<c>	the number of occurecnces of the character c normalized by 7	count<ט>=0.14 count<ו>=0.14 count<ב>=0.28
middleChar<c>	a boolean feature that indicates if the character c appears in the middle of the word	middleChar<ו>=1 middleChar<ב>=1
startsWith<c>	a boolean feature that indicates if the character c appears at the beginning of the word	startsWith<ט>=1
startsWith<c_1><c_2>	a boolean feature that indicates if the bigram c_1c_2 appears at the beginning of the word	startsWith<ט><ו>=1
endsWith<c>	a boolean feature that indicates if the character c appears at the end of the word	endsWith<ב>=1
endsWith<c_1><c_2>	a boolean feature that indicates if the bigram c_1c_2 appears at the end of the word	endsWith<ב><ב>=1
has<c_1><c_2>	a boolean feature that indicates if the bigram c_1c_2 appears in the word	has<ט><ו>=1 has<ו><ב>=1 has<ב><ב>=1
has<c_1><c_2><c_3>	a boolean feature that indicates if the trigram $c_1c_2c_3$ appears in the word	has<ט><ו><ב>=1 has<ו><ב><ב>=1
transliterated<w>	a boolean feature that indicates if w is transliterated according to a transliterated classifier (see Section 3.3).	the word was not classified as transliterated by the classifier
pos<p>	a boolean feature that indicates if p is the POS of the word	pos<adjective>=1

Table 5.2: Features used for the non-words SVM classifier

At first we used the SVM with a linear kernel and the default parameters as defined by the *LIBSVM* to determine the most effective features. After setting the features, as described above, we conducted several more tests with different kernels and different parameters. The results of the classifier with a polynomial kernel, a RBF kernel and a sigmoid kernel were significantly worse. Lowering the cost parameter on the other hand, improved the results. The best results were achieved while using a linear kernel and a cost parameter equal to 0.1.

5.3 Experiments

We compared our most effective SVM classifier (linear kernel and a cost parameter equal to 0.1) to five other classifiers:

- **Baseline Classifier** classifies each word as the most common label as seen in the training set.
- **Positional Bigrams Classifier** classifies words by the method of Riseman and Hanson [35] while using positional bigrams. It uses our dictionary (See Chapter 3.1) for creating the binary matrices.
- **Positional Trigrams Classifier** is similar to the previous classifier only it uses positional trigrams.
- **Peculiarity Index Classifier** is a classifier based on the method of Morris and Cherry [28]. The frequencies of the bigrams and trigrams are computed from the entire TB corpus. During training the threshold is set to the index that maximizes the results over the train set.
- **Error Probability Classifier** is based on the method of Zamora et al. [42]. The error probabilities are calculated from the training set. During training each value in $\{0.001, 0.002, \ldots, 1\}$ is tested as a threshold. The threshold for classification is set to the value that maximized the results over the train set.

Each classifier that we developed was evaluated by using a *10-fold cross-validation*. We partitioned the data set randomly into 10 subsets. We used 9 sets for training the classifier and one set for testing it. The size of the train set was 6144 and the size of the test set was 682. We repeated this process ten times, each time using a different set as a test set. The average of the results gave a single estimation of the classifier performance.

Next, we added a threshold parameter to the SVM classifier. Classification with a threshold is done according to the decision value returned by the SVM model. This value is defined by the distance of the sample from the hyperplane. We made sure to give positive decision values to samples that were tested positive by the classifier and negative decision values to samples that were tested negative by simply multiplying the decision value by -1. We compared the performance of the SVM classifier to the two classifiers that use a threshold: (1) the Peculiarity Index classifier and (2) the Error Probability classifier. We set different values for each classifier's threshold and we used 10-fold cross-validation for evaluation.

5.4 Results

The task of a non-word error classifier is to flag all the non-words in a given text and not to flag all the real words. This can lead to two types of errors: (1) the classifier did not flag a non-word and (2) the classifier flagged a real word. A classification is *true positive* (TP) if the classifier flagged a non-word; *false positive* (FP) if the classifier flagged a real word; *true negative* (TN) if the classifier did not flag a real word; *false negative* (FN) if the classifier did not flag a non-word.

We evaluate the performance of a classifier by four parameters: (1) *Precision* is defined as $TP/(TP + FP)$, i.e., the ratio between the number of times the classifier flagged a non-word and the total number of flagged words; (2) *Recall* is defined as $TP/(TP+FN)$, i.e, the ratio between the number of times the classifier flagged a non-word and the total number of non-words; (3) *F-measure* is defined as $(2 \cdot Precision \cdot Recall)/(Precision + Recall)$, i.e., the harmonic mean of precision and recall - a measure that combines those two parameters; (4) *Accuracy* is defined as $(TP + TN)/(TP + TN + FP + TN)$, i.e., the ratio between the number of times the classification was correct and the total number of classifications.

Classifier used	Precision	Recall	F-measure	Accuracy
Baseline	0.52	1.00	0.68	0.52
Positional Bigrams	0.69	0.30	0.42	0.57
Positional Trigrams	0.51	0.62	0.56	0.49
Peculiarity Index	0.52	1.00	0.68	0.52
Error Probability	0.72	0.80	0.75	0.73
SVM	0.79	0.78	0.78	0.78

Table 5.3: Non-word error classifiers results

The performance of each classifier is shown in Table 5.3. Our SVM classifier outperformed significantly the five other classifiers in precision, f-measure and accuracy, with results of 0.79, 0.78 and 0.78 respectively. The baseline classifier and the peculiarity index classifier labelled each word as a non-word, thus achieved recall of 1. This yielded a low precision of 0.52 which is the proportion of the non-words in our data set.

Figure 5.1 illustrates the precision and recall performances of the Peculiarity Index classifier, the Error Probability classifier and the SVM classifier. For every recall the SVM classifier achieves better precision than the other two classifiers.

Figure 5.2 illustrates the *Receiver Operating Characteristic* (ROC) curve of the SVM classifier. The ROC curve is a plot of the *True Positive Rate* (TPR) and the *False Positive Rate* (FPR) as the decision threshold of the classifier is varied. The TPR is defined as $TP/(TP + FN)$ and FPR is defined as $FP/(FP + TN)$. The ROC curve gives a single measure to the classifier performance which is defined by the *Area Under Curve* (AUC). The linear line in the figure illustrates the performance of a completely random classifier.

The AUC of the SVM classifier is 0.86.

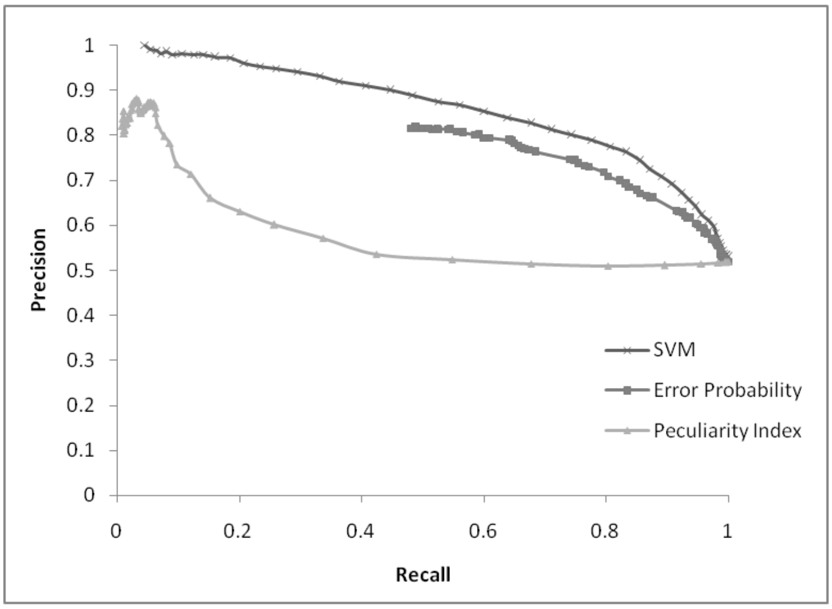

Figure 5.1: Precision and recall performance of the non-word error SVM classifier

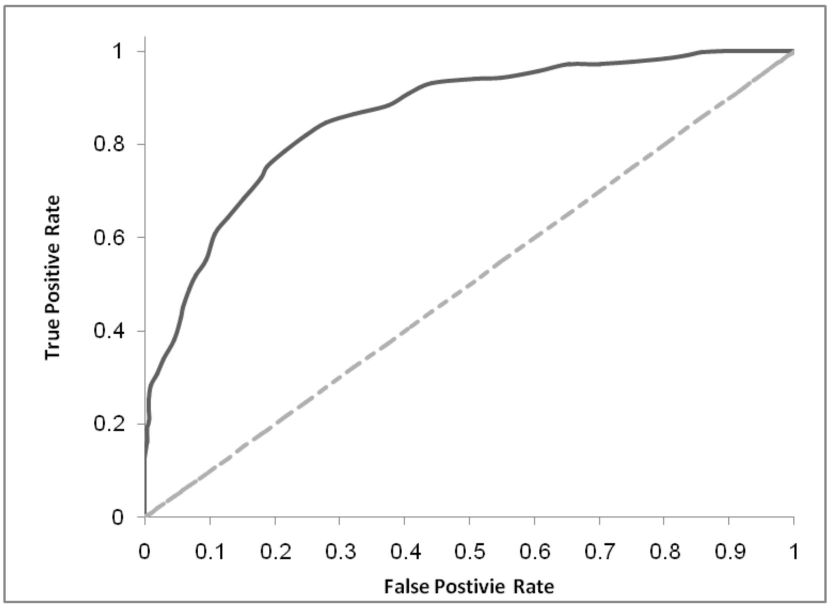

Figure 5.2: The ROC curve of the non-word error SVM classifier

5.5 Discussion

As reported above, our SVM classifier is able to determine if a non-dictionary word is a non-word or a real word with 0.79 precision and 0.78 recall. The results were the highest among five other classifiers which were based on previous non-dictionary spell checking methods.

The results indicate that it is not sufficient to use n-grams analysis of dictionary words for non-words classification. The positional n-grams method of Riseman and Hanson [35] and the peculiarity index method of Morris and Cherry [28] do not use any information from the errors in the train set, and consequently did not improve the baseline classifier. In contrast to these classifiers, the error Probability technique of Zamora et al. [42], extracts its probabilities from the errors in the train set. The SVM classifier uses the errors as training samples. The results of these two classifiers were much higher than the results of the other classifiers.

Most of the non-words that were classified correctly by the SVM classifier include words containing letters at illegal positions, e.g., סןבלים (the letter ן can appear only at the end of a word); and words containing irregular sequences of letters, e.g., תדבריייי (the trigram ייי is an indication for the error) . The learner automatically inferred these Hebrew properties from the training data. The majority of the real words that were classified correctly include proper names, e.g., קופרמן *kuperman* (Cooperman); and transliterated words, e.g., צ'אנסים *chansim* (chances), as we predicted above.

By inserting a threshold to the decision of the SVM classifier one can increase the precision at the cost of recall and vice versa. The classifier can reach a recall of 1 with a precision of 0.53. Adding this outcome to the result of the base line classifier indicates that the SVM classifier is able to detect all the errors in a text, while reducing the false alarms that are performed by a regular spell checker. The classifier can also reach a precision of one with a recall greater than zero, i.e., detect some of the errors with zero false alarms. A system can adjust the threshold's value as needed.

Considering the difficulty of the task and the rather small number of train samples, the performance of our SVM classifier is very impressive. By substantially increasing the training size, we believe that the classifier can achieve even better accuracy as more patterns of non-words and real words can be inferred.

Chapter 6

Real-word Error Classifier

In this chapter we will describe and evaluate a real-word error classifier that is based on the confusion set approach. Given a small set of words which are likely to be confused with each other and a sentence that contains one of those words, a real-word error classifier determines which word out of the set of words was actually intended.

6.1 Related Works

Golding [13] used two types of features for creating a Naive Bayes classifier: *context words* and *collocations*. Context words are individual words that appear around a confusion set word and imply the existence of that particular word. For example, if the confusion set is {desert,dessert} then the words "chocolate" and "delicious" in context imply the existence of the word "dessert", while the words "sun" and "sand" imply the existence of the word "desert". Collocations are sequences of words and/or POS of words next to a confusion set word. They express a pattern of syntactic elements next to a particular word.

Golding's system extracted features that characterized the contexts in which each confusion set word tended to occur. First every possible feature was extracted from a training corpus, i.e., one context word feature for every distinct word within k words from a confused word and one collocation feature for every sequence of l words/POS next to a confused word, when l and k are predefined. Then, features that had insufficient data or were uninformative discriminators according to the training were pruned. Classification was done by using the Naive Bayes formula, when features that were interdependent with other features were dropped to insure conditional independence among features.

In a following work, Golding and Roth [14] applied the *Winnow* learning algorithm for the task. The Winnow algorithm is very similiar to the Perceptron algorithm, only instead of using an additive weights update scheme, the weights are updated multiplicatively. They used similar technique to those applied in Golding's previous work [13] to extract features.

However, in order to exploit the ability of the Winnow algorithm to deal with a large number of features, they modified the method to do only minimal pruning. Golding and Roth tested the Winnow based classifier and the previous Naive Bayes classifier on 21 confusion sets, which included homophone confusions, grammatical errors and typographic errors. Each classifier was trained over 80% of the Brown corpus and was tested over the remaining 20%. The Winnow based classifier outperformed the Naive Bayes classifier, with an accuracy of 96.4% averaged over the 21 sets.

Carlson et al. [4] simplified the Winnow based system of Golding and Roth [14] by using only a single layer architecture. They reduced the learning time and the memory requirement of the system, while maintaining the performance. They managed to do so by using a larger training corpus and a *prediction confidence*, which is a value that represents how confidence is the prediction. Their system made a prediction only when the prediction confidence exceeded a predefined threshold. They tested their system on 256 confusion sets which were generated automatically by using simple edits in both the character space and the phoneme space. The averages for all 265 confusion sets reached an accuracy of 99% while making a prediction for over 85% of the occurrences.

Banko and Brill [2] explored the effect of the training data size on the confusion set task. They tested four different machine learning algorithms: (1) Winnow (2) Perceptron (3) Naive Bayes and (4) a very simple memory based classifier. The Winnow, Perceptron and the Naive Bayes were trained using similar features as in Goldings's work, i.e., context words and collocations of words and/or POS. The memory based classifier used only the word before and the word after the target word as features. They tested the performances of the different classifiers as a function of the training set size. The result was a log-linear effect. The Winnow, Perceptron and the Naive Bayes classifiers improved their accuracy from about 90% to over 95% when the training set was increased from 10M words to 1B words.

Ingason et al. [21] studied the issue of morphological richness and ambiguity in the task of context sensitive spelling correction. They applied the confusion sets approach on Icelandic "whose morphology is both rich and highly ambiguous". In addition to the use of context words and collocations as features, they also extracted *context lemmas*, which are the base forms of words near a confusion set word. By using a Naive Bayes learner their system achieved an average accuracy of 87.2% over 11 two-words confusion sets.

6.2 Method

We created 9 confusion sets in order to evaluate the ability to correct three different types of errors:

Syntactic real-word errors that resulted in a different POS

1. אם *im* (if), עם *im* (with)
2. לא *lo* (no), לו *lo* (to him)
3. אל *al* (don't), על *al* (on)

Syntactic real-word errors that resulted in a missing or an unnecessary definite

4. משחק *mishak* (game), המשחק *ha-mishak* (the game)
5. משפט *mishpat* (trial), המשפט *ha-mishpat* (the trial)
6. שבוע *shavua* (week), השבוע *ha-shavua* (this week)

Syntactic real-word errors that resulted in a different gender

7. גדול *gadol* (big, S-M[1]), גדולה *gdola* (big, S-F)
8. האחרון *ha-haharon* (the last, S-M), האחרונה *ha-hahrona* (the last, S-F)
9. חזק *hazak* (strong, S-M), חזקה *hazaka* (strong, S-F)

The first three confusion sets were selected as they are common mistakes in Hebrew. The rest of the confusion sets simply consist of highly frequent words as seen in the Walla corpus.

Algorithm 6.1 generic real-word error classifier

```
Input: $trainCorpus$, $confusionSet$, $features = [f_1, \ldots, f_m]$
Output: a real-word error classifier
1:  $trainSet \leftarrow \{\}$
2:  for each $sent$ in $trainCorpus$ do
3:    for each $word$ in $sent$ do
4:      if $word$ in $confusionSet$ then
5:        $sample \leftarrow [f_1(word, sent), \ldots, f_m(word, sent)]$
6:        $label \leftarrow word$
7:        add $(sample, label)$ to $trainSet$
8:      end if
9:    end for
10: end for
11: use a supervised learning algorithm to infer a classifier from $trainSet$
```

We developed several classifiers for each confusion set. Each classifier was trained over the Walla corpus, which is considered a corpus of well formed text. The training set contained more than 28.7M tokens. An occurrence of a word which belongs to the confusion set is considered a positive sample to this word and a negative sample to all the other words in the confusion set. Algorithm 6.1 describes the general procedure. Given a well formed text, a confusion set and a set of features, it creates a classifier with the use of supervised learning.

[1](S=singular, P=plural, M=masculine, F=feminine)

The samples are represented by a vector of features, when $f_i(word, sent)$ is the value of the feature f_i of the word w in the sentence *sent*. Each sample is labelled according to the observed word from the confusion set.

Next we describe the set of features we used for representing the context of the target word. We made a few principal changes from Golding's [14] common context words and collocations approach.

As the confusions sets in this work do not cover semantic real-word errors, we assumed that words that were more than one word from the target word to be irrelevant to the decision of the classifiers. Thus, context words features were left out of our method.

The collocations features were defined only according to the two tokens before and after the target word. Because of the high token instance/type ratio of Hebrew, which can lead to data sparseness, we did not use the tokens themselves for the collocations features. Instead, we used the base form of those tokens, i.e.,*lemmas*. The lemmas were computed by Adler's morphological analysis system (see Chapter 3.2). In addition, the common POS tag was replaced by the complete morphological analysis label returned by Adler's system, which includes among other the gender, number and definiteness of nouns and adjectives. We assumed that this complete analysis would be more effective in capturing linguistic patterns in the context of our confusion sets.

Let us denote the complete morphological analysis label of the ith word after the target word by m_{+i}, the complete morphological analysis label of the ith word before the target word by m_{-i}, the lemma of the ith word after the target word by l_{+i} and the lemma of the ith word before the target word by l_{-i}. For example, for the sentence היא תעבוד _ שר הבריאות *hi taavod _ sar ha-briut* (She will work _ the Minister of Health), when the blank stands for the target word עם *im* (with), we get:

m_{-2} = :PRONOUN-F,S,3: (a feminine singular third person pronoun)
m_{-1} = :VERB-F,S,3,FUTURE: (a feminine singular third person verb)
m_{+1} = :NOUN-M,S,CONST: (a masculine singular construct noun)
m_{+2} = DEF:NOUN-F,S,ABS: (a definite feminine singular absolute noun)
l_{-2} = הוא *hu* (he)
l_{-1} = עבד *avad* (worked)
l_{+1} = שר *sar* (minister)
l_{+2} = בריאות *briut* (health)

For every sentence in the train set that contained a confusion set word we extracted the following binary features (**pp**-second before; **p**-before; **nn**-second next; **n**-next; **L**-lemma; **M**-morphological analysis label):

(1) ppLpL<l_{-2}><l_{-1}> (2) ppLpM<l_{-2}><m_{-1}> (3) ppMpL<m_{-2}><l_{-1}>

(4) ppMpM<m_{-2}><m_{-1}> (5) pLnL<l_{-1}><l_{+1}> (6) pLnM<l_{-1}><m_{+1}>

(7) pMnL<m_{-1}><l_{+1}> (8) pMnM<m_{-1}><m_{+1}> (9) nLnnL<l_{+1}><l_{+2}>

(10) nLnnM<l_{+1}><m_{+2}> (11) nMnnL<m_{+1}><l_{+2}> (12) nMnnM<m_{+1}><m_{+2}>

(13) pL<l_{-1}> (14) nL<l_{+1}> (15) pM<m_{-1}> (16) nM<m_{+1}>

Sentences with less than two words before or after the target word received a special mark for every missing word. Finally, to reduce the number of features, we pruned those that appeared less than five times in the training set. Table 6.1 summarizes the number of different features extracted from these words after pruning.

Confusion Sets	Features
אם *im* (if)	16763
עם *im* (with)	54479
לא *lo* (no)	78522
לו *lo* (to him)	14154
אל *al* (don't)	9217
על *al* (on)	102709
משחק *mishak* (game)	8273
המשחק *ha-mishak* (the game)	9200
משפט *mishpat* (trial)	1129
המשפט *ha-mishpat* (the trial)	3622
שבוע *shavua* (week)	2174
השבוע *ha-shavua* (this week)	5125
גדול *gadol* (big, S-M)	4939
גדולה *gdola* (big, S-F)	2948
האחרון *ha-haharon* (the last, S-M)	5960
האחרונה *ha-hahrona* (the last, S-F)	3301
חזק *hazak* (strong, S-M)	1746
חזקה *hazaka* (strong, S-F)	1021

Table 6.1: Number of extracted features

We examined the performance of three types of classifiers using the above features: (1) a Naive Bayes based classifier (2) a Perceptron based Classifier and (3) a SVM based classifier.

The Naive Bayes classifier This classifier is based on the Naive Bayes algorithm (see Chapter 4.1) with a small modification. Classifying the observed word w in a sentence *sent* according to Equation 6.1, yielded poor results.

$$Classify(w, sent) = \arg\max_{w} P(C = w) \prod_{i=1}^{n} P(F_i = f_i(w, sent)|C = w) \quad (6.1)$$

In order to improve the classifier performance we gave greater weight to the proportion of the target words in the train set, i.e, to the probability $P(C = w)$. We did it by simply raising the probability $P(C = w)$ by the power of the number of features. Our final Naive Bayes classifier classifies the target word w in a sentence *sent* according to Equation 6.2.

$$\begin{aligned} Classify(w, sent) &= \arg\max_{w} P(C = w)^n \prod_{i=1}^{n} P(F_i = f_i(w, sent) | C = w) \\ &= \arg\max_{w} \prod_{i=1}^{n} P(F_i = f_i(w, sent), C = w) \end{aligned} \tag{6.2}$$

The probability $P(F_i = f_i(w, sent)|C = w)$ was calculated according to the *Expected Likelihood Estimate* (ELE). The ELE is a simple smoothing technique that estimates the probability of a sample with c occurrences in a training set with N samples instances and B possible samples types by $(c + 0.5)/(N + B/2)$.

The Perceptron classifier This classifier is based on the Voted Perceptron algorithm (see Chapter 4.3). We used the free Python implementation of the algorithm by Leif Johnson[2].

The SVM classifier This classifier is based on the SVM algorithm (see Chapter 4.2) by using the *LIBSVM* [5] library. We used the SVM with a linear kernel and the default parameters as defined by the *LIBSVM*.

6.3 Experiments

Well Formed Text We created a test set consisting of 2.4M tokens taken from the part of the Walla corpus that was not used for training. We will refer to this test set as the **Walla Test Set**. As mentioned above, the Walla corpus is considered well formed, thus every instance of a word from a confusion set is actually a label sample, i.e., the sample is the features values defined by the context of the observed word and the label is the observed word itself. We tested each of the three classifiers on this test set and we compared their results to two other classifiers: (1) a **Baseline Classifier** and (2) a **Word-POS Classifier**. The Baseline classifier classifies each sample as the most frequent word as seen in the train set. The Word-POS Classifier is a voted perceptron based classifier which uses similar features to the other three classifiers, only instead of lemmas and complete morphological analysis labels it uses words and simple POS labels.

Artificial Errors We assembled 500 sentences from the Walla Test Set for every confusion set. The sentences were chosen in such a way that each sentence contained exactly one instance of some word from the corresponding confusion set. Then, the instance of the word from the confusion set was replaced with the other confusion set word, which yielded sentences with exactly one artificial error. The new sentences were analysed morphologically

[2]Copyright (c) 2009 Leif Johnson <leif@leifjohnson.net>

again using the system of Adler [1]. Finally, the replaced word was changed back to the original word so the sentences contained the correct target word. We will refer to this test set as the **Walla Artificial Test Set**. We tested each of the 5 classifiers discussed above on this test set.

Training Size We tested the effect of the size of the training set on the Naive Bayes classifier, Perceptron Classifier and the SVM classifier. We trained the classifiers using smaller portions of the training set. Each classifier was tested on the Walla Test Set.

Prediction Confidence For each classifier we added a confidence parameter. This parameter is calculated during classification and gives a real value for the confidence of the decision made by the classifier. We defined it as the ratio between the two largest probabilities $\prod_{i=1}^{n} P(F_i = f_i(w, sent), C = w)$ for the Naive Bayes classifier, as the difference between the two largest values of the output function for the Perceptron classifier, and as the absolute value of the decision value for the SVM classifier. We tested each of these classifiers on the Walla Test Set again, except this time we kept the confidence parameter for each classification. We measured the performance of these three classifiers only according to their k highest confidence predictions. We set k to 85%, 90% and 95% of the number of test set samples.

Real Errors We used the TB corpus as a test set. This text is unedited and contains many real errors. A classifier can not be evaluated automatically on this test set as it contains sentences in which the target word is not the observed word. However, by manually examining the samples that were labelled as different words from the observed words, we can estimate the precision of a classifier. We used the confidence parameter discussed above to sort the samples that were labelled differently from the observed words in descending order. Then, we manually checked whether the classification of the top samples was correct, i.e., the classifier corrected a real-word error. We tested only our most effective classifiers on this test set.

Table 6.2 provides details on the number of instances of each confusion set word in the different data sets. These numbers include occurrences of homograph words. The data are organized by the different confusion sets. **Train** - the training set; **WTS** - the Walla Test Set; **WATS** - the Walla Artificial Test Set; **TBTS** - the TB Test Set

Confusion Sets	Train	WTS	WATS	TBTS
אם *im* (if)	54986	3852	130	8673
עם *im* (with)	160155	13145	370	10047
לא *lo* (no)	258669	18878	440	39348
לו *lo* (to him)	35112	2535	60	4508
אל *al* (don't)	24372	2342	25	1850
על *al* (on)	304857	25706	475	18228
משחק *mishak* (game)	19250	1750	209	1201
המשחק *ha-mishak* (the game)	26512	2236	291	688
משפט *mishpat* (trial)	3953	429	148	219
המשפט *ha-mishpat* (the trial)	12048	960	352	331
שבוע *shavua* (week)	4633	402	123	237
השבוע *ha-shavua* (this week)	12684	1273	377	76
גדול *gadol* (big, S-M)	12297	861	275	1254
גדולה *gdola* (big, S-F)	7472	646	225	493
האחרון *ha-haharon* (the last, S-M)	15622	1502	317	381
האחרונה *ha-hahrona* (the last, S-F)	8157	782	183	312
חזק *hazak* (strong, S-M)	3575	332	317	389
חזקה *hazaka* (strong, S-F)	2203	189	183	204

Table 6.2: Number of instances of confusion words in the train and test sets

6.4 Results

Well Formed Text Table 6.3 reports the accuracy of the different real-word classifiers tested on the Walla Test Set. **Average Accuracy** is the average accuracy of the method used over the different confusion sets; **Weighted Average Accuracy** is the average accuracy of the method used over the different confusion sets, when each accuracy of a confusion set is weighted proportionally according to its frequency in the Walla Test Set.

The SVM classifier outperforms the other classifiers for almost every confusion set. It achieves average accuracy of 96.0% and weighted average accuracy of 97.7%. The Naive Bayes classifier and the Perceptron classifier fall short behind with average accuracies of 94.2%/95.3% and weighted accuracy of 96.0%/97.5% respectively.

The Perceptron classifier is more accurate than the Word-POS classifier for seven out of the nine confusion sets. These two classifiers use the same learning algorithm, the first with lemmas and/or complete morphological analysis features and the latter with words and/or POS. The results are significant mainly for the last three confusion sets, which involve an inflection error.

Artificial Errors The accuracy of the Naive Bayes, the Perceptron and the SVM classifiers was significantly lower when tested on the Walla Artificial Test Set. The average accuracy of the three classifiers dropped 3.1%/3.4%/3.5% respectively while the weighted average accuracy dropped 0.5%/0.7%/1.1% respectively. The most radical change in the results was a drop of 16.4%-18.8% from the accuracy of the three classifiers on the fifth confusion

Confusion Set	Baseline	Word-POS	Naive Bayes	Perceptron	SVM
אם, עם	77.3	97.1	96.1	97.3	97.5
לא, לו	88.2	98.6	97.5	98.6	98.9
אל, על	91.7	97.4	95.8	97.8	97.8
משחק, המשחק	56.1	94.9	93.6	95.0	95.3
משפט, המשפט	69.1	92.9	91.4	93.2	93.2
שבוע, השבוע	76.0	94.0	92.1	93.6	94.4
גדול, גדולה	57.1	82.5	93.9	94.9	95.0
האחרונה, האחרון	65.8	91.5	96.9	97.1	98.3
חזק, חזקה	63.7	80.8	90.4	90.2	93.9
Average Accuracy	71.7	92.2	94.2	95.3	96.0
Weighted Average Accuracy	83.4	96.8	96.0	97.5	97.7

Table 6.3: Accuracies of real-word error classifiers tested on the Walla test set

set. Despite these changes, the SVM classifier remains the most accurate classifier for every confusion set. Table 6.4 summarizes the results. Because this test set consists of exactly 500 occurrences of each confusion set, the weighted average accuracy was calculated according to the occurrences in the Walla Test Set.

Confusion Set	Baseline	Word-Pos	Naive Bayes	Perceptron	SVM
אם, עם	74.0	96.0	95.2	96.6	95.8
לא, לו	88.0	98.0	96.4	98.2	98.4
אל, על	95.0	98.4	97.6	98.6	98.0
משחק, המשחק	58.2	89.6	89.6	90.4	93.0
משפט, המשפט	70.4	74.6	75.0	74.4	75.0
שבוע, השבוע	75.4	91.8	90.4	91.4	91.4
גדול, גדולה	55.0	81.2	90.0	92.8	92.0
האחרונה, האחרון	63.4	90.4	95.8	95.6	96.2
חזק, חזקה	63.4	80.8	89.8	89.0	92.6
Average Accuracy	71.4	89.0	91.1	91.9	92.5
Weighted Average Accuracy	83.8	96.1	95.5	96.8	96.6

Table 6.4: Accuracies of real-word error classifiers tested on the Walla artificial test set

Training Size In Figure 6.1 we illustrate the effect of the size of the training set on the classifiers. Each point in the graph is the average accuracy over the different confusion sets. The learning curves appear to be log-linear (base 5) from 217K training data words up to 28.7 training data words.

Prediction Confidence The accuracy of the Naive Bayes, Perceptron and SVM classifiers was significantly improved when we considered only the samples that the classifiers were most confident about. Table 6.5 shows the performance per confusion set. The *willingness* parameter is the proportion of samples for which the classifier made a prediction out of the total number of test samples. The selection of the samples is done according to the confidence value. For example, 85% willingness with 100 test samples, means that the classifier was

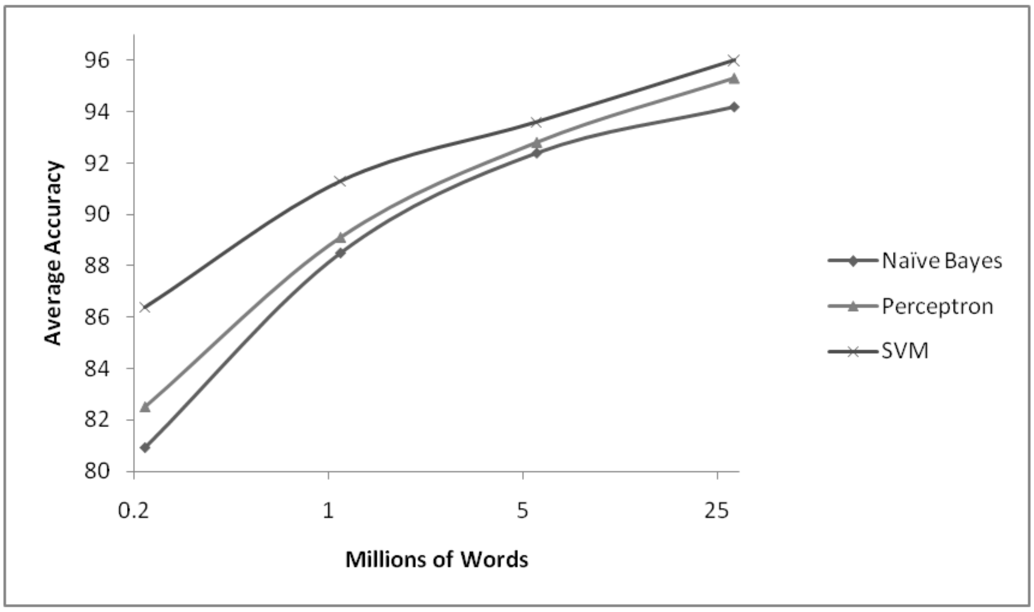

Figure 6.1: Learning curves of real-word error classifiers

tested only on the predictions for the 85 most confident samples. **N** is the Naive Bayes classifier, **P** is the Perceptron classifier and **S** is the SVM classifier.

	95% willingness			90% willingness			85% willingness		
Confusion Set	N	P	S	N	P	S	N	P	S
אם, עם	98.1	98.8	99.0	98.9	99.3	99.3	99.4	99.5	99.4
לא, לו	98.8	99.6	99.7	99.4	99.7	99.8	99.6	99.8	99.9
אל, על	97.7	98.9	99.0	98.6	99.3	99.4	99.2	99.5	99.6
משחק, המשחק	95.5	96.9	97.1	97.0	97.8	97.9	98.1	98.4	98.4
משפט, המשפט	92.8	94.3	95.0	94.3	95.0	96.1	95.5	95.8	96.1
שבוע, השבוע	94.0	95.0	95.9	95.4	96.7	96.8	96.1	97.3	97.5
גדול, גדולה	95.9	96.8	96.9	97.1	97.9	98.2	98.0	98.8	98.4
האחרונה, האחרון	98.5	99.0	99.3	99.2	99.5	99.7	99.6	99.7	99.8
חזק, חזקה	92.7	92.5	96.0	94.0	94.9	97.6	95.7	97.3	98.4
Average Accuracy	96.0	96.9	97.5	97.1	97.8	98.3	97.9	98.5	98.6
Weighted Average Accuracy	97.8	98.7	98.9	98.6	99.1	99.3	99.1	99.4	99.4

Table 6.5: Accuracies of real-word error classifiers using confidence value tested on the Walla test set

Real Errors Our most effective classifier, the SVM classifier, was selected for this experiment. The classifier performed extremely well on the first confusion set. Out of 100 flagged samples, 98 were real errors and the classification made by the classifier was the right correction for the error. In one of the two mistakes made by the classifier, there was an error in the word following the confusion word. The results were quite promising as well for the second and third confusion sets, with a precision of 90% and 77.8% respectively.

The performance of the classifier for confusion sets involving definiteness or gender inflection errors was significantly poorer. As for the first type of confusion sets, the classifier made some correct classifications. Out of the 60 samples tested, 11 were correct. We note that some of the mistakes made by the classifier resulted in a valid sentence. As for the second type, none of the 60 classifications were correct. Table 6.6 summarizes the results.

Confusion Set	Precision
אם, עם	98.0% (98/100)
לא, לו	90.0% (45/50)
אל, על	77.8% (28/36)
משחק, המשחק	40.0% (8/20)
משפט, המשפט	15.0% (3/20)
שבוע, השבוע	0.0% (0/20)
גדול, גדולה	0.0% (0/20)
האחרונה, האחרון	0.0% (0/20)
חזק, חזקה	0.0% (0/20)

Table 6.6: Precision of real-word error classifier

6.5 Discussion

Our three real-word classifiers achieved promising results over 9 different confusion sets, when the SVM based classifier outperformed the Naive Bayes and the Voted Perceptron learners.

Using the common well formed text test, the SVM classifier reached an average accuracy of 96% and a weighted average accuracy of 97.7% over the different sets. The accuracy of this classifier was the highest over the first three confusion sets, which represented a syntactic real-word error that resulted in a different POS. Over these three sets the average accuracy rises to 98%, while over the three sets representing a syntactic real-word error that resulted in a missing or an unnecessary definite the average accuracy drops to 94.3% and over the three sets representing a syntactic real-word error that resulted in a different gender the average accuracy drops to 95.7%. It is important to emphasize that the first three confusion sets words occurred a lot more frequently in the training corpus than the rest of the confusion sets words, which yielded a larger training set for the classifier.

The Perceptron classifier achieved better accuracy then the Word-POS classifier over seven out of the nine confusion sets. These classifiers differ only in the set of features used. The Word-POS classifier, which uses words and/or POS features, lacks the necessary morphological knowledge to classify confusion sets that involve different inflections with high accuracy. The results favoured the Perceptron classifier over the other confusion sets as well. The additional morphology knowledge with the use of lemmas instead of the words themselves as features, seem to be more useful for a confusion set learner in a rich morphological language such as modern Hebrew.

Despite falling behind the SVM classifier and the Perceptron classifier on accuracy, our Naive Bayes method should be taken into consideration. In more realistic cases, where there are thousands of confusion sets, instead of training a classifier for each of the sets, this method simply calculates the conditional probability of every confusion set word. Thus, in cases where there are overlaps among different sets, this method is much more efficient. An interesting study would be to embed this method into a spell checker and to examine its ability to sort the correction candidates suggested by the spell checker for non-words errors.

The accuracy of the classifiers dropped when the morphological analysis of the text was performed after inserting artificial errors. The SVM classifier remained the most accurate classifier according to the average accuracy, with a score of 92.5%. When the average accuracy is measured over 8 of the 9 confusion sets, the score rises to 94.7%, as the accuracy over the confusion set {משפט, המשפט} was extremely low. Since the samples in this test set differ from the ones in the Walla Test Set only in the morphological analysis labels, we can conclude that they are the reason for the decline in the accuracy. The morphological analysis system used in this work selects a tag for a word according to a probability conditioned by the two tags that precede it or by the tag that precedes it and by the one that follows [1]. The errors that were inserted yielded unpredicted tags for some of the words near the target words, thus affecting the performance of the classifier. As for the drastic decline in the performance over the confusion set {משפט, המשפט}, we believe that it is mainly because these two confusion words can appear in many similar contexts, e.g., בית _ החליט *beit _ hehlit* (_ house decided). The results of this experiment give us a rough upper bound on the recall of the SVM classifier over these specific real-words errors, as all of the samples in this test set are instances of these errors. One might reconsider selection of a certain confusion set according to the outcome of this test.

The learning curve of the classifiers appeared to be log-linear. The outcome is similar to the one in the work of Banko and Brill [2]. According to this result, we predict that scaling the training size up to 150M words will yield an average accuracy of over 98%. Nevertheless, enlarging the training size should be done carefully as it requires extra memory and increases the running time of the classifier, as more features need to be calculated.

Inserting a confidence parameter into the classification and predicting only samples in which this parameter is high enough, increased the accuracy of the classifiers substantially. The SVM classifier reached an average accuracy of 98.6% and a weighted averaged accuracy of 99.4% while ignoring only 15% of the samples. Its accuracy over two confusion sets reached nearly 100%. In order to make the classifiers applicable for real-world systems they need to reach an accuracy of 98%-99% [4]. For less common errors the accuracy has to be even higher, as the probability for a false alarm by a system increases. We have shown that by using the decision value returned by the SVM model, the classifier reaches this goal for 7 out of 9 confusion sets, at the cost of lowering the recall.

Our last experiment over an unedited text with real errors validates the ability of the confusion set approach to detect and correct real-word errors. Despite the differences between the domains of the training text and the test text, the SVM classifier managed to detect and correct 5 types of real errors, while achieving very high precision over the first three confusion sets.

The significant gap in the results over the various confusion sets can be explained by several reasons. First, the number of test samples is not equal, e.g., the confusion set {אם, עם} has 18,720 instances in the TB Test Set while the confusion set {חזק, חזקה} has only 593 instances. As a result, the chances for an error to occur in the test set is much lower for some of the confusion sets. Second, the prevalence of the errors is not equal. The first three confusion sets were selected according to well known Hebrew mistakes in which the confusion set words are short, frequent, sound the same and also can be transformed to the other confusion set word with a single letter replacement. The other confusion sets represent grammatical mistakes which are less common among Hebrew writers.

As for the disappointing results over the confusion set {האחרונה, האחרון}, a set for which the accuracy reached a 99.8% with 85% willingness, we note that when using a threshold that yields these results, the classifier made only 2 false predictions.

The lack of Hebrew error corpora makes it difficult to truly evaluate the performance of a real-word error classifier. In this work we used common methods for evaluating which gave us only an estimation of the ability to detect and correct errors by the classifiers. However, when we tested the SVM classifier over real errors we saw that the results were not exactly as they were predicted by these tests. A realistic evaluation must be based on a large amount of real text, where its real errors are labelled with their corrections. From such corpora the detection/correction of an error by the classifier can be tested, the different errors can be extracted, confusion sets can be assembled and the profitability of artificial confusion sets can be measured.

Chapter 7

Conclusions

In this work we have examined the ability to detect and correct errors in a modern Hebrew text. The work includes the development of two types of classifiers, each designed to overcome one of the two principal problems pertaining to simple spell checkers: (1) Valid words that do not appear in the system's dictionary are flagged as errors and (2) they do not flag real-word errors. In addition, as part of the development of these classifiers, corpora of edited and unedited modern Hebrew text were assembled.

The first type of classifier that we developed was a non-word error classifier. This classifier provides a solution to the first problem of flagging all the unfamiliar words as errors. With the use of the SVM learning algorithm and features based only upon the present word, this classifier is able to determine whether an out of dictionary word is an error with a precision of 0.79 and a recall of 0.78. The samples that were used for training and evaluating were extracted from the corpus of unedited text and contained real errors in addition to valid words that a common Hebrew spell checker flagged as errors. Our classifier managed to distinguish between the two classes while achieving better results than other previous methods.

The second problem we addressed involves errors resulting in valid words which a simple spell checker does not flag. By implying the common confusion set approach with slight adjustments appropriate to a rich morphological language such as Hebrew, we have created real-word error classifiers based on different learning techniques. Out of all the different classifiers we explored in this work, the SVM based classifier achieved the best results. This classifier was able to determine which word from the confusing set should appear in a given context with an average accuracy of 96% over 9 different sets. By adding a confidence parameter to the decision of the classifier, the average accuracy rose to 98.6% while covering 85% of the tested samples. It should also be noted that we expect that by increasing the size of the training set, better results can be reached by the classifier, as we measured a log-linear effect.

In another experiment the SVM classifier was able to correct authentic real-word errors in an unedited text with high precision for sets containing common Hebrew real-word errors. Over one of the most frequent real-word errors in Hebrew, confusing אם *im* (if) with עם *im* (with), the precision reached 98%. The classifier had some success in correcting errors involving a missing or a redundant definite. However, over real-word errors involving a gender inflection all the flags made by the classifier were false alarms.

We can conclude that the confusion set approach is highly applicable for modern Hebrew. Nevertheless, the selection of a confusion set should be careful to consider not only the theoretical accuracy over a well formed text, but also the evaluation of the classifier over an unedited text.

7.1 Future Work

We believe that the two types of classifier that were developed in this work have great potential in detecting and/or correcting various types of errors. Further work should continue to explore these classifiers and their capabilities as follows:

Training Data The training data of the non-word error classifier should be increased to include millions of samples. Such samples can be extracted automatically from websites that store versions of their previous contents, e.g., Wikipedia. In addition, the training data of the real-word error classifiers should be scaled up to 1 billion words while checking its effectiveness.

Different Errors Further ways to create different confusion sets automatically should also be explored. Thousands of different confusion sets can easily be assembled by selecting similar strings. There is a need to develop a method that prunes unlikely mistakes.

Using a Parser In this work we failed to detect errors that involved wrong usage of gender inflection. We believe that a parser might be the missing link for our real-word error classifier for errors of that type.

Error Corpora This research emphasizes the need for an annotated error corpora for modern Hebrew. With such corpus more information about errors people tend to make could be extracted and our classifiers could be evaluated in a more accurate way. We hope that the results above will encourage the creation of such a corpus, as it will be a significant gain for the field of error detection/correction in Hebrew.

Bibliography

[1] M.M. Adler. *Hebrew morphological disambiguation: An unsupervised stochastic word-based approach*. PhD thesis, Citeseer, 2007.

[2] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33. Association for Computational Linguistics, 2001.

[3] G. Brooks, T. Gorman, and L. Kendall. *Spelling it Out: The Spelling Abilities of 11-and 15-year-olds*. National Foundation for Educational Research, 1993.

[4] A.J. Carlson, J. Rosen, and D. Roth. Scaling up context-sensitive text correction. *Urbana*, 51:61801, 2001.

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

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

[7] F.J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[8] D. Fossati and B. Di Eugenio. I saw tree trees in the park: How to correct real-word spelling mistakes. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, pages 896–901. Citeseer, 2008.

[9] Y. Freund and R.E. Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[10] W.A. Gale and K.W. Church. Poor estimates of context are worse than none. In *Proceedings of the workshop on Speech and Natural Language*, pages 283–287. Association for Computational Linguistics, 1990.

[11] Y. Goldberg and M. Elhadad. Identification of transliterated foreign words in hebrew script. *Computational Linguistics and Intelligent Text Processing*, pages 466–477, 2008.

[12] Y. Goldberg and M. Elhadad. Easy first dependency parsing of modern hebrew. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 103–107. Association for Computational Linguistics, 2010.

[13] A.R. Golding. A bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, volume 3, pages 39–53, 1995.

[14] A.R. Golding and D. Roth. A winnow-based approach to context-sensitive spelling correction. *Machine learning*, 34(1):107–130, 1999.

[15] N. Har'el and D. Kenigsberg. Hspell-the free hebrew spell checker and morphological analyzer. In *Israeli Seminar on Computational Linguistics*, 2004.

[16] G.E. Heidorn, K. Jensen, L.A. Miller, R.J. Byrd, and M.S. Chodorow. The epistle text-critiquing system. *IBM Systems Journal*, 21(3):305–326, 1982.

[17] C.W. Hsu and C.J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[18] J.H. Huang and D. Powers. Large scale experiments on correction of confused words. In *Australian Computer Science Communications*, volume 23, pages 77–82. IEEE Computer Society, 2001.

[19] A. Itai and S. Wintner. Language resources for hebrew. *Language Resources and Evaluation*, 42(1):75–98, 2008.

[20] A. Itai, S. Wintner, and S. Yona. A computational lexicon of contemporary hebrew. In *Proceedings of The fifth international conference on Language Resources and Evaluation (LREC-2006), Genoa, Italy.* Citeseer, 2006.

[21] A.K.I.S.B. Jóhannsson, E. Rögnvaldsson, H. Loftsson, and S. Helgadóttir. Context-sensitive spelling correction and rich morphology.

[22] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439, 1992.

[23] E. Mays, F.J. Damerau, and R.L. Mercer. Context based spelling correction. *Information processing & management*, 27(5):517–522, 1991.

[24] M. McIlroy. Development of a spelling list. *Communications, IEEE Transactions on*, 30(1):91–99, 1982.

[25] R. Mitton. Spelling checkers, spelling correctors and the misspellings of poor spellers* 1. *Information processing & management*, 23(5):495–505, 1987.

[26] R. Mitton. *English spelling and the computer.* Citeseer, 1996.

[27] R. Mitton. Spellchecking by computer. *Journal of the Simplified Spelling Society*, 20(1):4–11, 1996.

[28] R. Morris and L.L. Cherry. Computer detection of typographical errors. *IEEE Transactions on Professional Communication*, (1):54–56, 1975.

[29] D. Naber et al. A rule-based style and grammar checker. *Master's thesis, University of Bielefeld*, 2003.

[30] J. Pedler and R. Mitton. A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation (LREC'10)*, 2010.

[31] M.L. Peters. *Success in Spelling: a study of the factors affecting improvement in spelling in the junior school.* Cambridge Institute of Education (Cambridge), 1970.

[32] J.L. Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980.

[33] J.L. Peterson. A note on undetected typing errors. *Communications of the ACM*, 29(7):633–637, 1986.

[34] D.M.W. Powers. Learning and application of differential grammars. 1997.

[35] E.M. Riseman and A.R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *Computers, IEEE Transactions on*, C-23(5):480 – 493, may 1974.

[36] K.F. Shaalan. Arabic gramcheck: A grammar checker for arabic. *Software: Practice and Experience*, 35(7):643–665, 2005.

[37] C.M. Sterling. Spelling errors in context. *British Journal of Psychology*, 74(3):353–364, 1983.

[38] V.N. Vapnik. Statistical learning theory. 1998.

[39] S. Verberne. Context-sensitive spell checking based on word trigram probabilities. *Master's thesis, University of Nijmegen, February-August*, 2002.

[40] A.M. Wing and A.D. Baddeley. Spelling errors in handwriting: A corpus and a distributional analysis. *Cognitive processes in spelling*, pages 251–285, 1980.

[41] S. Wintner. Hebrew computational linguistics: Past and future. *Artificial Intelligence Review*, 21(2):113–138, 2004.

[42] EM Zamora, J.J. Pollock, and A. Zamora. The use of trigram analysis for spelling error detection. *Information Processing & Management*, 17(6):305–316, 1981.

[43] Y. Zhao and K. Truemper. Effective spell checking by learning user behavior. *Applied Artificial Intelligence*, 13(8):725–742, 1999.