TEL AVIV UNIVERSITY

Raymond and Beverly Sackler
Faculty of Exact Sciences
The Blavatnik School of Computer Science

Tools to aid OCR of Hebrew character manuscripts

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
In

Computer Science

by

Alex Zhicharevich

Thesis Supervisor:

Professor Nachum Dershowitz

February 2011

Acknowledgments

I would like to thank my thesis advisor, Nachum Dershowitz for helping me focus on the right problems, providing me with ideas and helpful insights when I needed and his guidance throughout my work.
I would also like to express my appreciation to my wife Meital, my family and friends for supporting me through the tougher moments.

Abstract

Digitalization of historical and cultural documents can provide researchers new options for conducting research on variety of subjects. Although OCR systems are the common method for digitalization processes, they are sometimes not enough due to the poor performance of those on documents that include handwriting, low contrast, writing style shifts and various other characteristics of manuscripts. For those documents, the OCR needs to be post-processed to allow successful utilization of the data contained in the documents.

The thesis proposes various methods for such post-processing, using techniques from the field of natural language and statistical language modeling. The methods proposed for language classification, document segmentation and text searching. The methods are designed to handle very noisy texts and tuned to work on Hebrew language, as part of the project of the digitalization of the Cairo Genizah – a collection of ancient and medieval Jewish work. The methods are tested on both real and artificially reduced documents.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

The Cairo Genizah is a combination of important scholarly works, community records and ledgers, business and marriage contracts, personal letters and more. Among them are original manuscripts in the hand of famous medieval scholars and personalities. The digitalization of those manuscripts can open various research possibilities for cultural and historical researchers. Nonetheless, this process poses challenges to the traditional digitalization processes of scanning and recognition of text by an optical character recognition (OCR) engine. Although virtually all the documents written in Hebrew script, the fact that the manuscripts are handwritten, when handwriting can vary in style and clearance, presents a big challenge to OCR systems. Other properties of the documents, such as poor quality of the manuscripts, multiple languages, incomplete pages and other challenges, make it impossible for an OCR engine to produce results that can satisfy the minimal demands for any research. Hence, common method for handling noisy texts is using some statistical, language oriented post-processing on the result to increase accuracy.

The post-processing of a text relies on properties of the language the text is written in. The tools present a scale up in the level of processing of the text, from geometric features of written figures to the linguistic meanings of those characters as building blocks for words. It is common, for example, to correct erroneous words by matching them to some known words in a known vocabulary or to measure the probability of some character combination in a language. Identification of the text language is a preliminarily for those methods, in case the language is not given and cannot be implied from the script/encoding of the characters. In the case of the Cairo Genizah, the texts appear in a variety of Hebrew-script languages, such as Aramaic, Hebrew, Judeo-Arabic, Ladino and more. The degree of similarity between those languages also varies from languages similar in style like Hebrew and Aramaic, to languages that share only the script with the others, like Judeo-Arabic. Moreover, many of the documents are mixtures of paragraphs in different languages, presenting a further challenge for the application of post-processing tools on them since a straight forward classification of language cannot be used.

Even after successful identification of the language of every part of the text, the application of traditional correction methods is not straight forward. Due to the low accuracy of the results produced by the OCR process, correction of text using single word lookup is not satisfying. On the other hand, a significant part of the Genizah documents are transcriptions of some known Jewish texts, which we can look for in pre-prepared repositories. Therefore, an application of approximate string matching techniques for searching the noisy text in this repository can be useful for such post-processing.

For the post-processing of OCR on Cairo Genizah documents we present a multi stage scheme:

1. Identification of the document language.
2. Segmenting the document into monolingual fragments in the case of a multi-lingual document.
3. Searching the text in a corpus according to language recognized.

1.1 The Cairo Genizah

The Cairo Genizah is a collection of over 350,000 Jewish manuscripts found in the loft of the ancient Ben Ezra Synagogue in Fustat (medieval Cairo), to the south-west of the modern city used as a repository between the 11th and 19th centuries. The dark, sealed room in the arid Egyptian climate contributed to the preservation of the documents, the earliest of which may date back to the eighth and ninth centuries. The Genizah texts are written in various languages, especially Hebrew, Arabic and Aramaic, mainly on vellum and paper, but also on papyrus and cloth. They represent the most important discovery of new material for every aspect of scientific Hebrew and Jewish studies in the Middle Ages. In addition to containing Jewish religious texts, such as Biblical, Talmudic and later Rabbinic works (some in the original hands of the authors), the Genizah gives a detailed picture of the economic and cultural life of the North African and Eastern Mediterranean regions, especially during the 10th to 13th centuries. Its documents reveal a wealth of information about this previously little known period in Jewish history. Today, a large portion of the Genizah's documents are available at Cambridge University Library and at the Jewish Theological Seminary in New York. Smaller collections are spread out in university library collections across the globe, among them London, Oxford, Manchester, Paris, Geneva, Vienna, Budapest, St Petersburg, New York, Philadelphia, Washington and Jerusalem; some are housed in private collections.

1.2. Corpora collection

The algorithms proposed below use statistical properties of the languages in which Genizah documents are largely written. A significant effort was made for collecting statistics on those languages, which are not commonly used nowadays, and digital copies of documents in those languages are not widespread. The corpora collected for Hebrew contains the *Torah* – the Pentateuch, and the *Mishnah* - the first major written redaction of the Jewish oral traditions, which is also the first major work of Rabbinic Judaism. For Aramaic, the corpus contains the *Jerusalem Talmud* - a collection of rabbinic notes on the Mishnah, which was compiled in the Land of Israel during the 4th-5th century. The Talmud, as a commentary on the Mishnah, contains significant number of Hebrew quotes, so it is not pure Aramaic. Another Aramaic book is *Targum Onkelos*, an official Aramaic translation of the Torah. For Judeo-Arabic, later works were collected such as *More Nevuchim* (The Guide for the Perplexed) by Maimonides, the *Kuzari* by Rabi Yehuda Halevy, and *Hamaspik Ovdey Hashem* by Maimonides son.

Other collections were obtained for further experiments, among which are the *Zohar* in Aramaic, which is the foundational work in the literature of Jewish mystical thought known as Kabbalah, the *Shulkhan Arukh* which is the most authoritative legal code of Judaism and other Jewish religious work. A full list of the components of the corpora is listed in Appendix 1.

For the use of the collection as a statistical reference, it was processed to clean of irrelevant characters, unneeded lines and various punctuation signs. It was then tokenized and n-gram statistics were collected.

1.3 Related work

Much work has been conducted in the field of OCR post-processing, most using statistical approaches over N-grams or vocabularies. The methods over vocabularies contain approximate string matching techniques for searching lists of all known words of a language such as proposed by Chen et al (2010). Statistical methods use probabilities over character combinations for correcting OCR errors, combined with confusion matrices (Kukish, 1992). Kolak and Resnik (2005) advice the use of statistical methods in the case of low density languages, where massive document sets for producing vocabulary are not available. Methods for using word n-grams for such a process were also introduced. However, little work has been done on using those methods on multilingual documents. Approximate string matching methods for strings against corpora were surveyed by Navarro (2001) and include dynamic programming algorithms, filtering techniques and approaches using final automata.

Work on language classification has been widely studied (Hughes et al, 2006), mostly as a classification problem. Two approaches dominate the work in this area, word based and character based. Word-based approaches represent the text as a vector of words and use supervised classification techniques for the identification of language. The character based approaches do this by comparing n-gram probability distributions over each language and the text (Hakkinen and Tian 2001).

The processing of multilingual documents was addressed by Giguet (1996), who addressed the problem using grammatical words and end of word characters. The processing was sentence-wise, and actually the segmentation process was not issued. Related work on segmentation of text, usually of semantic nature, was pioneered by Hearst (1993) and used sliding window techniques. Follow-up works utilized lexical chains techniques, clustering, dynamic programming and other techniques (Choi, 2000).

1.4 Structure

The rest of this thesis is structured as follows: Chapter 2 describes the method for language classification of documents. Chapter 3 describes the extension of the method for segmenting multi-lingual documents into monolingual fragments. Chapter 4 presents the algorithm for searching noisy texts in a corpus. Each of those chapters includes a short background, description of the algorithm and experiments performed to test the algorithm. Chapter 5 contains conclusions and discusses further possible research directions.

CHAPTER 2

LANGUAGE CLASSIFICATION

An important step in the digitalization process of manuscripts is language identification. Apart from using the language to help catalogue the manuscripts, recognizing the language is a crucial part for OCR processes. An OCR post-processing algorithm (described in further in this work) assumes knowledge of the language of the manuscript for choosing the appropriate corpus to scan.

2.1 N-Gram approach

An obvious fact is that different languages, even if utilizing the same character set, have different distributions of letter occurrences. Therefore, gathering statistics on the typical distribution of letters in each language may lead us to reveal the language of a manuscript, by comparing its letter distribution to the distributions of known. A simple distribution of the letters may not be enough, so a common technique in NLP is using n-grams which means computing the distributions of all possible combinations of n letters. Obviously, the number of possible combinations grows exponentially with n, so usually the value of n does not exceed 4.

The classification process can be described by the following procedure

1. Collect n-gram statistics for all relevant languages.
2. Compute n-gram distribution on the manuscript.
3. Compute the distance of the manuscript's distribution from each language using some distance function.
4. Classify the manuscript as the language with the minimal distance.

The first task in computing the n-gram distributions is choosing n. In our experiments, we tried unigram, bigram and trigram. The characters we considered were all Hebrew alphabet letters, including “sofiot” (variants of letters that appear at the end of the words). The only additional character used was the space character, under the assumption that different languages can have different word lengths (for languages with shorter words the space character will have higher appearance count) and that different languages tend to have different letters ending a word (and then bigrams or trigrams containing those letters followed by a space will appear more often). Specifically, when a human tries to identify Aramaic texts, he may do it by looking for words ending by alef ('א'), a property strongly correlated with this language. The probability function for an n-gram i is given by

$$P(i) = \frac{Count(i)}{\sum_{j \epsilon all\ Ngrams} Count(j)}$$

It is easy to see that the denominator, which is the sum of all appearances of all n-grams in the text, is just the length of the text (minus n). The formula implies that an n-gram that was not spotted in the text has a zero probability, a fact that can be true for some n-grams (for example n-gram which contains a letter that appears only at the end of the word followed by a character which is not space), but is not generally correct. There are techniques that smooth the distribution function, giving unseen n-grams a probability larger than zero, but we chose not to address this problem by smoothing but by adapting the distance function to handle such distributions.

The second missing detail in the algorithm is the distance function. Let A be the alphabet and A^n the set of all n-grams over A. Since the distribution function is discrete, we can actually represent it as a vector of probabilities over A^n, and transform the problem into a vector distance problem. We tried the following three distance functions:

- **Cosine similarity** – this function is basically the cosine of the angle between two vectors, measuring how similar are the directions of the two vectors. The value is computed using the formula

$$Cosine(d1, d2) = \frac{d1 \cdot d2}{\|d1\| * \|d2\|} = \frac{\sum_{i \epsilon A^n}\left(P_{d1}(i) * P_{d2}(i)\right)}{\sqrt{\sum_{i \epsilon A^n}\left(P_{d1}(i)\right)} * \sqrt{\sum_{i \epsilon A^n}\left(P_{d2}(i)\right)}}$$

The function is a similarity measure rather than a distance measure, therefore when classifying a manuscript, the language with the highest similarity value is taken (opposed to the minimal distance for other functions). It is also symmetric and normalized to values between zero and one.

- **KL Divergence** - the Kullback–Leibler divergence, often referred to as information gain, is a measure between two distributions, originated from information theory. The function is defined as following

$$\mathrm{KL}(\mathrm{d1}, \mathrm{d2}) = \sum_{i \epsilon A^n}\left(P_{d1}(i) * \ln\left(\frac{P_{d1}(i)}{P_{d2}(i)}\right)\right)$$

Note that there are several problems using this measure for classification purposes. First, the function is not symmetric therefore we need to choose whether d1 is the corpus language distribution or the manuscript distribution. It is common to look at the KL divergence as a measure to how much a sample distribution d2 differs from the "true" distribution, therefore, we used (after some testing) d1 as the corpus distribution. Another challenge is the presence of zero probabilities. If $P_{d1}(i) = 0$ or $P_{d2}(i) = 0$ then $\ln\left(\frac{P_{d1}(i)}{P_{d2}(i)}\right)$ is undefined. We chose to ignore all n-grams not present in one of the distributions, which can of course distort the distance (for example, if the manuscript and language have no n-gram in common, the distance will be zero although it should be infinity) but simplifies the function to match our needs.

- **Euclidean distance** – this is the straight forward approach for measuring distances between vectors. $Dist(d1, d2) = \sqrt{\sum_{i \epsilon A^n} (P_{d1}(i) * P_{d2}(i)\ )^2}$

2.1.1 Unknown Classification

For shorter fragments, we can expect poorer performance. On the other hand, if we allow an "Unknown" classification, we can reduce the error rate for some such fragments. To determine when the classification should be set to "Unknown" we need some certainty measure for the classification. We can then set some threshold and classifications with certainty above the threshold may be considered certain and below the threshold will be considered uncertain or "unknown". This can be helpful in many cases, especially when the classification precision is of high importance. Using this method, "unknown" fragments can be further analyzed (maybe manually) and classified fragments can enjoy very high certainty.

To get this certainty measure, we can look at the cosine similarities of fragments to their closest language. We obviously expect them to grow as the fragment length grows. For extracting the certainty measure we can use two methods:

Absolute distance – If the distance of the fragment to the classified language is very high, we can be more certain of the classification. Here we assume that mistakenly classified fragments will have lower similarity than the correct ones, as presented in Figure 1. We then use regression to learn a function of the threshold dependency on length. We tried establishing a linear logarithmic function of the form $threshold = a + b * \ln(c * length + d)$ where a, b, c and d are parameters to be determined by regression.

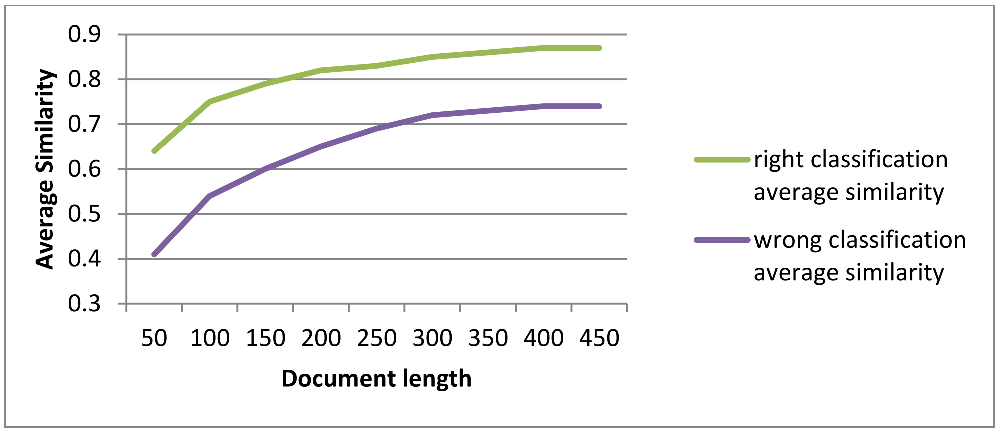

Figure 1: The difference between the average cosine distance of correctly classified texts and mistakenly classified texts

We can see in Figure 1 that the accurately classified texts are classified with much higher similarity then the mistaken ones so it looks possible to compute some threshold under which we can say the classification is not certain.

Relative distance – Here we rely on the intuition that when a document is classified correctly, its cosine similarity to the correct language is much higher than its similarity to other languages. We can define a variable *offset* that will stand for the difference between the cosine similarity of the fragment to the closest language and the document's average similarity to all considered languages. More formally,

$$offset = \max_{l \epsilon \text{languages}} Cosine(l, document) - \text{Average}_{l \epsilon \text{languages}}\, Cosine(l, document)$$

Figure 2 shows that *offset* indeed is significantly larger when the classification is correct, so we can use it as the certainty threshold as we can see that for wrongly classified documents the offset is always in the range of 0.04-0.05

Here, we will not set the threshold as a function of the length, but use the variance of the similarity distances. For each document we can compute the standard deviation *std(document)* of the cosine distances from each language. We will say that the classification is certain if *offset>a*std* for some constant *a.*

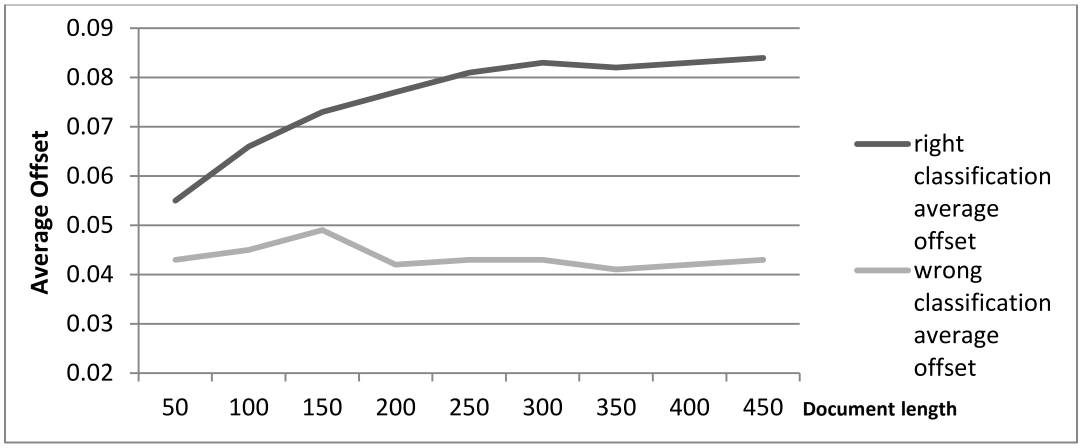

Figure 2: The average of the offset measure (the difference between the maximum similarities to the average similarity) of correctly and mistakenly classified documents.

2.1.2 SMALL AND NOISY DOCUMENTS

Classifying OCR processed manuscripts poses several unique challenges not encountered when handling traditional language classification of documents. One of these challenges is handling a significant amount of noise in OCR outputs. Another challenge is the frequency of extremely small texts, some with fewer than 50 characters. The significance of small documents classification rises when handling the problem of multilingual document segmentation described below. The length of the documents and the noise rates can make some statistic measures less efficient due to distorted distributions or insignificance of statistics on small samples.

Several methods (Kukich, 1992) have been proposed for error correction using n-grams and transition probabilities – the probabilities of one letter following another. Here, we are not interested in error correction, but in the adjustment of the classifying procedure to handle noisy texts. For noise representation we introduce the *"$"* character to stand for a character unrecognized by the OCR system. We do not discuss error recognition here and assume that errors are recognized and represented by *"$"*. A conservative OCR system could only output characters which have high probability of correctness and output the rest as "$", so all misidentification mistakes can be reduced to this notion. There is also no assumption that the word boundaries will not be misidentified, so "$" can be produced instead of a space character.

Several methods are proposed:

Ignoring unrecognized n-grams – Here we do not count the n-gram containing "$" in the cosine similarity measures. This requires no change from the regular pattern since those n-grams do not appear in the language model anyway. Here we assume there are enough bigrams left in the text to successfully identify its language.

Remove unrecognized characters – We can also remove the "$" fragment from the text before starting any analysis. On the one hand, it looks natural to ignore all noise, but on the other hand we lose the information that noise was indeed produced. For example, 'א$ב' will transform to 'אב' which may distort the n-gram distributions.

Error correction – Given an unknown character we can try correcting it using trigrams. When observing "$" surrounded by a character *L* on its left and *R* on its right, we can look for the most common trigram in each language containing *L* in the beginning and *R* at the end. It looks natural to do this and enhances the statistical power of the n-gram distribution. On the other hand, it does not scale well for high noise rates since there is no solution for two or more consecutive *"$"* characters.

Averaging n-gram probabilities – When encountering "$", we can use averaging to estimate the probability of the n-gram containing it. For instance the probability of the bigram 'א$' will be the average probability of all bigrams starting with 'א' in a certain language. This can of course scale to higher n-grams and integrates the noisy information into the computation.

**Replacing the '$'** – We can try to replace '$' by some other character without relying on the language model. We do that by looking at the character *L* before it, and searching the given text for another appearance of it. The character appearing after *L* in the closest appearance to the '$' character will be the one we will choose to replace it with. This is a quite heuristic and is not a statistic error correction, relying on replacing an unknown bigram can be predicted using similar bigrams close to it in the text will enhance the statistical significance of recognized bigrams.

Top n-grams – When looking at noisy text, we can say that more weight should be given to the corpus statistics since it is error free. Moreover, since the text is short we expect to see only a small portion of the common n-grams in the text. Therefore we may look only on the k most common n-grams in the corpus, assuming that they must appear in the text regarding noise and length.

Higher or lower n-gram space – So far bigrams showed superior performance. When the error rate rises and text length drops, the more distinctive n-grams such as trigrams may produce higher success rates, while on the other hand, unigrams would need shorter text sample for robust statistics so are also reconsidered.

2.2 Experiments and results

2.2.1 TEST SETTINGS

The success of language classification can depend heavily on the properties of the test set. For the task of classifying manuscripts, there are several properties to be considered:

Text length – manuscripts can be of different lengths, from a small number of sentences up to a whole page that contains multiple paragraphs. It is clear that the variance of the distributions of smaller texts is much higher, so the probability of a statistical model extracted from a short text to differ from the language model is higher. Therefore, we can expect a lower accuracy on shorter texts. For our experiments we tested various text lengths to measure the influence of this parameter.

OCR error rate – Assuming that the classified text is a result of some noisy process, we expect that high rate of noise will reduce the classification success rate. This parameter was also tested and we present how we address highly erroneous documents.

Language set – Although our languages share the same character set, they can still significantly differ from one another. For example, Hebrew and Judeo-Arabic are completely different, with little chance that a Hebrew speaker will understand Judeo-Arabic even a little. On the other hand, some languages can share the same character set due to common origins, which will resemble in the high similarity between them that can make the classification task more difficult. Such are Hebrew and Aramaic that have many similar words or a word in one language that is some variant or cognate of a word in the other language. Needless to say that as the set of languages grows the classification task becomes more difficult.

2.2.2 TEST RESULTS

To test the distance function we begin by selecting 300 documents, 100 in each language, and try to classify those using bigrams with each of the above mentioned distance functions. For this purpose we use prepared error-free text. Each document is 300 characters long.

	Cosine	KL	Euclidian
Overall	0.94	0.81	0.94
Hebrew	0.93	0.78	0.94
Aramaic	0.89	0.72	0.89
Judeo-Arabic	1.00	0.94	1.00

Table 1: Classification accuarcy of distance function

From the results two facts arise clearly: The cosine and Euclidean functions have higher accuracy than KL and the Judeo-Arabic language is much easier to spot then Hebrew and Aramaic.

Three hundred characters are about four sentences which is a pretty short text. For similar languages like Hebrew and Aramaic, it may be too short to get a good classification. We also want to try out trigrams in order to gain better statistics. To test this, we classified texts of various lengths, using unigrams, bigrams and trigrams. We tried it only on Hebrew and Aramaic since we saw that Judeo-Arabic is distinguishable pretty easily.

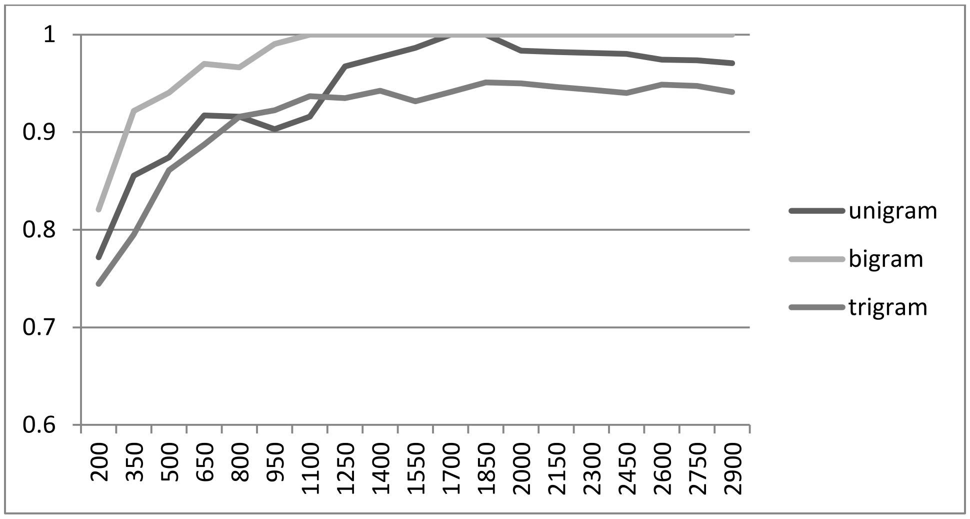

Figure 3: Classification accuracy of different n-grams

From Figure 3, we can see that generally bigrams are the best method on all lengths. For texts longer than 1000 characters the performance is perfect. On short texts trigrams have low performance, which rises as the text size grows, but does not reach the bigram performance even on long texts. Perhaps on really long texts, the statistical power of trigrams would be more significant, but on page sized texts it is inferior. Unigrams have poorer performance then bigrams even on the shortest texts.

2.2.3 "UNKNOWN" CLASSIFICATION TESTS

By allowing classification to return an "unknown" result, we obviously reduce the error rate. On the other hand, since the unknown classification is not a correct classification, it also reduces the success rate. To establish a fair measure, we can score a successful classification as 1, an unknown classification by 0 and wrong classification by -1. It is a "neutral" score since right and wrong classifications weigh the same. For error sensitive classification the weight of the error should increase.

For absolute threshold we estimated the threshold function as $threshold = a + b * \ln(c * length + d)$ where

a = 5.31E-02;

b = 1.29E-01;

c = 8.27E-01;

d = -1.18E+01;

Increasing *a* will make classification more error sensitive (lower error rate and lower success rate) and decreasing it will give higher success rate (and error rate).
For relative threshold, we set $threshold = a * std$, where *a=0.8.* As a grows, the classification is more error sensitive (lower error rate), and as *a* reduces the success rate grows.

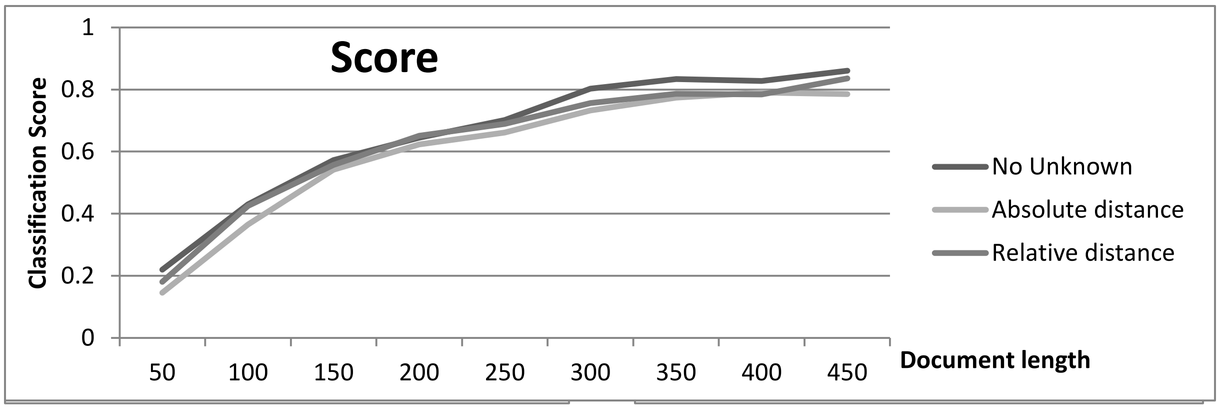

Figure 4: The success rate, the error rate and the classification score of every method of unknown classification

Naturally, "unknown" classification methods reduce both the error rate and success rate. We can see that the relative distance method is superior to the absolute distance, with significantly lower error rates on almost every length and equal success rate. We can also notice that for neutral classification score, the regular classification is superior to all methods. Only when we measure the classification with an error sensitive score, do the "unknown" classification methods become relevant.

2.2.4 Noisy texts

A test to measure the performance of all noise reduction methods was done on various document lengths. The error rate was simulated using the '$' character, that randomly replaced text characters according to some error rate.

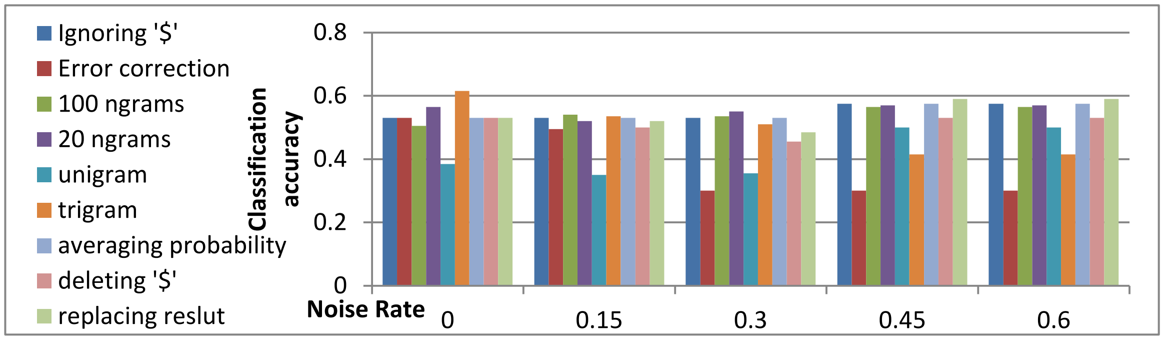

Figure 5: The performance of all noise reduction methods on 40 character length documents

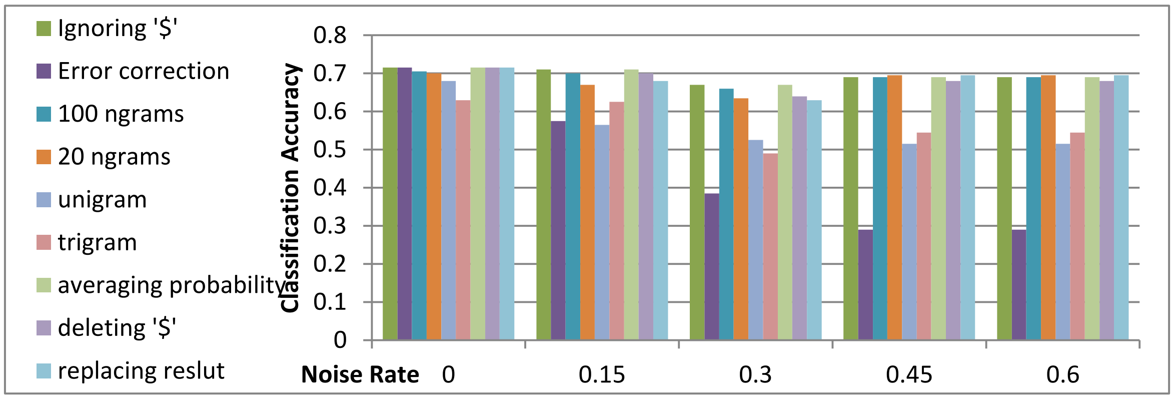

Figure 6: The performance of all noise reduction methods on 100 character length documents

From Figures 5 and 6 we can see that usually, just ignoring the unrecognized character, relying on the statistics of the recognized text, is the straightforward and best result. Trigrams perform well only on short noise-free texts, and reducing the bigram to the top 100 performs well also, usually not very different by from the ignoring methods. Taking the top 20 bigrams performs well only on very noisy texts as we can expect, presenting poorer performance on other cases, it looks suitable only when the amount of noise is extremely high. Error correction methods do not perform well, except the enhancement of replacing '$' with a character of neighboring bigram, which seems like a useful feature for high noise rates.

CHAPTER 3

SPLITTING BI-LINGUAL TEXTS

In the previous chapter we saw methods for language classification of manuscripts. The methods work under the assumption that each manuscript is monolingual, and their behavior on multilingual texts is unpredictable. As stated before, Genizah manuscripts contain mixed texts which cannot be strictly classified to any one language. When we look at texts of Jewish biblical philosophy or interpretation, we will usually find Aramaic texts with lots of quotations in Hebrew. Classifying such texts as one language is rather useless so instead of classification we are interested in a more general problem of splitting the text into monolingual fragments, classifying each fragment to its language.

3.1 Background

The problem of fragmenting multilingual texts into monolingual fragments was not addressed much, although it is a natural generalization of the language classification problem (Hughes 2006). Several related methods can be useful for this type of problems. The notion of structured learning is the generalization of the classification task (Daume and Marcu, 2005) to extend the target to complex structures such as sequences or trees. Theoretical general methods use Markov models or support vector machines for such predictions, usually using massive datasets for learning. The segmentation problem, as a simple case of structured learning, that results classification sequences can be addressed using those very general methods.

More specific methods deal with segmentation of text, such as automatic paragraph detection. All such methods use two modules, one classification model and a second segment-boundary searching model. The most popular approach is the sliding-window technique that looks for the most rapid change in classification scores for detecting boundaries. The general scheme of the designed algorithm should not be affected by noise unlike some sliding window approaches which apply classification to very small windows, a technique shown to be inefficient for classification of noisy documents.

3.2 Algorithm Outline

For the splitting task, we assume that no dictionary is available and only n-gram statistics of each language are known. We want the algorithm to work even if the language shifts every few sentences so we do not assume anything about the length of each fragment (We of course cannot count several words as a language shift). In the general case there is also no assumption that the sentences in the text are marked, so it can be a one long sentence as well. The algorithm has 4 major steps:

1. Split the text to fragments.
2. Calculate characteristics for each fragment.
3. Classify each fragment.
4. Refine classification result and output final results.

3.2.1 Splitting the text

As stated, we do not assume the documents are split into sentences or paragraphs. So the splitting is done in the naïve way of segmenting the text into fixed size fragments. Obviously, the language cannot shift in the middle of a word, so we do perform adjustment of the fragments sizes to fall between words. If sentences are marked in the text and we assume that the language cannot shift in the middle of the sentence, then the adjustment described is done at the level of a sentence.

The selection of fragment size should depend on the language shift frequency. Nonetheless, each fragment is classified using statistical properties, so it has to be long enough to have some statistical significance. On the other hand, if it is too long, the language transitions will be spotted less accurately, and if a fragment contains two language shifts, the algorithm will not be able to classify the inner fragment (for example if a fragment starts with Hebrew, shifts to Aramaic and ends in Hebrew, the algorithm can classify it as Aramaic or Hebrew, or split it into two languages in the post-processing stage but cannot spot all three fragments). Moreover, the post-processing phase is computationally more expensive, and its complexity grows proportionally with the fragment length so we cannot choose a long fragment size.

3.2.2 Feature extraction

The core of the algorithm is the classification of the fragments produced by the first step of the algorithm. Classification problems are usually reduced to vector classification, so there has to be a method of representing each fragment as a vector of features. Naturally, the selection of features is critical for successful classification, regardless of the classification algorithm.

N-gram distance – The first and obvious feature is the classification of the fragment using the methods described in Chapter 2. However, the fragments are significantly smaller than the texts that were classified in the previous chapter, so we can expect the accuracy to be much lower. The features in this case will be the cosine distance from each language model rather than a single feature with the result language. This is rather natural, since we want to preserve the distances from each language model in order to combine it with other features further on. For each fragment f and language l, we can compute $Distance_l = Dist(l, f)$ where $Dist(l, f)$ represents the cosine distance of the bigram distributions of l and f.

Neighboring fragments language – We expect that languages in a document are not shifting too frequently. It is a reasonable assumption, since paragraphs tend to be monolingual and usually at least several sentences in a row are in the same language to present some idea. Therefore, if we are sure that a fragment is in some language, there is a high chance that the adjacent fragment will be in the same language as well. One way to express such dependencies is by post-processing the results to reduce noise. Other way is by combining the classification results of neighboring fragments as features in the classification of the fragment. Of course, not only neighboring fragments can be considered, and all fragments under some distance from the fragment can help in classification. For example if we have a classification results of HHHAHHH (where H stands for Hebrew fragment and A for Aramaic fragment), it looks possible that the A is noise and should be H. On the other hand, if the result is HAHAHAH, there is no reason to turn the middle A to H. Some parameter should be estimated to be the threshold for the distance between fragments under which they will be considered neighbors. We denote Neighbor(f,i) = if i is positive then the i'th fragment after f. If i is negative the i'th fragment before f. If i=0 Neighbor(f,i) = f. So for each fragment f and language l we can compute $NeighborDist_{l,f}(i) = Dist(l, Neighbor(f,i))$

Whole document language - Another feature to be considered is the cosine distance of the whole document from each language model. This feature tends to smooth and reduce noise from the classification output. Note that for a monolingual document the algorithm is expected to output a single fragment (the whole document) classified as the right language. So for each language l we calculate $DocumentDist_l = Dist(l, text)$

Clustering – a major drawback of the features presented so far is the fact that they resemble statistical similarity between language models and very short text fragments. To increase classification accuracy, we would like to classify longer texts. In order to do so, we can cluster similar fragments together and then classify the whole cluster as a single unit. It will obviously be longer than a single fragment, so the classification will be more accurate, but there is no guarantee that the clustered fragments will actually be monolingual.

The clustering is done using complete linkage hierarchical clustering. The idea is to perform an iterative process where in each iteration we unify the two closest clusters. Initially, all fragments are clusters containing only one fragment .In every iteration, we unify the two closest clusters, where complete linkage stands for the metric used for calculating distance between two clusters. The distance between clusters is the maximal distance between any elements in the two clusters. More formally $Dist(C1, C2) = Max_{f1\epsilon C1, f2\epsilon C2}(Dist(f1, f2))$ where $c1\epsilon C1$ stands for fragment *f1* belong to cluster *C1*. We end this iterative process when the minimal distance between two clusters rises above some threshold *T* (which in turn means there are two fragments that the distance between them exceeds the threshold). In the end of the process we can compute the distance between each cluster and each language model. For a fragment *f* we denote the cluster that contains *f* as *Clus(f)* and for each fragment *f* and language *l* we can calculate features $Clus_l = Dist\big(l, Clus(f)\big)$. The threshold *T* for stopping the clustering process represents the threshold between gaining bigger clusters which can be better classified on one hand, and risking getting clusters that are not monolingual on the other hand and is established empirically,

Since the point of clustering is to get a longer text for classification, then the bigger the cluster gets the more positive we are in its classification. Therefore, the size of the cluster is another feature we want to consider in order to give more significance to the $Clus_l$ features for longer clusters. So, for each fragment *f* we denote by $ClusSize = |Clus(f)|$ the number of fragments assigned to the same cluster as *f*.

3.2.3 Classification phase

After the features have been extracted, the classification step is rather straight-forward. We can either use some known supervised learning method, such as learning the problem on a test set and producing a classifier, or we can try establishing some manual scoring formula using the features and classify by the language getting the highest score.

3.2.4 POST- PROCESSING

We now want to refine the fragment splitting procedure. We do it in the following way:

We look at the results of the splitting procedure and recognize all language shifts. For each shift we try to find the position where the shift takes place (at word granularity). We unify the two fragments and then try to re-split the fragment at N points. For every such point we look at the cosine distance of the words before the point from the language to which the first fragment was classified to and the cosine distance of the words after the point to the language to which the second fragment was classified to. For example, suppose the fragment *A1….An* was classified as Hebrew and the fragment *B1….Bm* which appeared right after it in the text was classified as Aramaic. We look at the text *A1…An,B1…Bm* and try to split it at N points (say N =3). So we try to split it in*to F1=A1…A(n+m)/3* and *to F2=A(n+m)/3…Bm* (suppose ((n+m)/3)<n). We look at cosine distance of *F1* to Hebrew and *F2* to Aramaic since those were the languages to which the fragments were originally classified to. Then we try to look at *F1 = A1…A(2*(n+m)/3)* and *F2 = A(2*(n+m)/3)…Bm,* and so on. We take the split point with the smallest product of the two values. The choice for N is a tradeoff between accuracy and computation efficiency. When N is higher, we check more transition points, but for large fragments it can be computationally expensive.

3.3 Noise Reduction

As for language classification, the segmentation algorithm can be extended to handle noisy documents. As the splitting and shift recognition phases are not expected to be noise sensitive, the classification phase of each segment is the best stage for handling noise. We test the segmentation success rate on all noise correction methods presented in the noise handling section for classification.

3.4 Experiments and results

3.4.1 Test settings

We want to test the algorithm with well-defined parameters and evaluation factors. For this purpose we created artificially mixed documents, containing fragments from two different languages (we can do it using Hebrew and Aramaic, which are difficult to distinguish, Hebrew and Judeo-Arabic where classification is easy and the fragmentation is the main challenge, or do it with three languages). The fragments were produced using a procedure that accepts two parameters: The desired document length d and the average fragment length l – where a fragment is a continuous text block of only one language. Obviously $l < d$. The procedure iteratively chooses a number in the range *[l-20,l+20]* and takes a substring of this size from a corpus of one language. The substring is adjusted to contain whole words only. It repeats this on all corpora of all other languages and then restarts with the first language until the whole text reaches the size of *d*.

Obviously l and d are of significance. For very small *l*, it will be very difficult to fragment the document exactly, since the text blocks will not be long enough for statistic tests. As for *d*, it is clear that the average number of fragments inside the document is $n = \frac{d}{l}$. As *n* grows larger it is more difficult for the splitting algorithm to be right for all fragments, and since *n* grows with *d,* we will expect to see a higher absolute error rate.

3.4.2 Success Measures

Obviously, the splitting procedure will not be perfect, and we cannot expect it to precisely split the document to the original fragments. Given that, we want to establish some measures for the quality of the splitting result. We would like the measure to produce some kind of score for the algorithm output, which can indicate

whether a certain feature or parameter in the algorithm improves it or not. However, the result quality is not well defined since it is not clear what is more important: detecting the fragment's boundaries accurately, classifying each fragment correctly or even splitting the document into the exact number of fragments. For example, given a long document in Hebrew with a small fragment in Aramaic, is it better to return that it actually is a long document in Hebrew with an Aramaic fragment but misidentify the fragment's location or rather recognize the Aramaic fragment perfectly but classify it as Judeo-Arabic.
We established three evaluation measures, with which we test the algorithm accuracy:

Correct word percentage – The most intuitive measure is simply measuring the percentage of words classified correctly. Since the "atomic" block of the text is words (or sentences in some cases described further), which are certainly monolingual, this measure will resemble the algorithm accuracy pretty good for most cases. It is however not enough, since in some cases it does not reflect the quality of the splitting. Assume a long Hebrew document with several short sentences in Aramaic. If the Hebrew is 95% of the text, a result that classifies the whole text as Hebrew will get 95%, but it is actually a pretty useless result and we may prefer a result that identifies the Aramaic fragments but errs on more words (say classifies the two Hebrew sentences before and after the Aramaic sentence also as Aramaic).

Fragment count ratio (FCR) – This measure estimates the algorithm sensitivity to language shifts. It counts the difference between the real number of fragments to the number of fragments returned by the algorithm. To normalize it is divided by the number of real fragments. Obviously, $\mathrm{FCR} \in [-1 \dots 1]$. It will indeed resemble the problem previously described, since if the entire document will be classified as Hebrew the FCR score will be very low as the actual number of fragments is much higher than one.

Splitting edit-distance (ED) – Counting the number of fragments (FCR) will allow the evaluation of the sensitivity of the splitting in the algorithm. However, it does not resemble the quality of the classification stage output. Going back to the same example, FCR will return the same result even if the algorithm will recognize the Aramaic fragment as Judeo-Arabic. In order to evaluate if the algorithm classifies correctly, we define the following measure: label each language in the language set by 1…n. such that each document can be represented by a vector representing the languages of its fragments. The ED will be the edit-distance between the vectors of the actual fragment decomposition to the vector produced by the split of the algorithm (this measure is not normalized so it supposed to grow with *d/l*). For instance, given a document which contains Hebrew text, then Aramaic, then Hebrew and Judeo-Arabic, it will be presented as *HAHJ*. If the algorithm misidentified the Aramaic fragment it will return *HJ* so the ED will be the edit distance between *HAHJ* and *HJ,* which is 2. If it will misclassify the Judeo-Arabic as Aramaic and produce *HAHA,* the ED will be 1. We can notice that if the language set contains only two languages, there is no point to the ED measure, since it will return the absolute value of the FCR measure. Due to the fact that each character in the classification language vector is different from the character following it (if they are the same they would be unified to the same character), the edit distance on binary vectors is just the length difference up to ±1.

Therefore we will only use this measure when the language set contains more than two languages.

3.4.3 NAÏVE SPLITTING

To get a reference on each feature of the algorithm, we will run a naïve algorithm on the documents. The basic algorithm will simply split the document, classify each fragment in the way documents are classified, and output the result. We want to test how *d* and *l* affect each classification parameter using this naïve scheme.

d	l	d/l	Correct words	FCR	ED
500	50	10.00	0.729	1.24	1.36
500	100	5.00	0.827	-0.48	0.56
500	150	3.33	0.847	-1.33	1.33
500	200	2.50	0.869	-1.77	1.77
500	250	2.00	0.902	1.45	1.45
1000	50	20.00	0.729	-2.64	2.68
1000	100	10.00	0.824	-0.62	0.9
1000	150	6.67	0.856	-1.52	1.56
1000	200	5.00	0.850	-2.75	2.79
1000	250	4.00	0.880	-2.61	2.61
1500	50	30.00	0.718	4.4	4.48
1500	100	15.00	0.813	-1.1	1.42
1500	150	10.00	0.841	-2.21	2.33
1500	200	7.50	0.859	-3.67	3.69
1500	250	6.00	0.882	-3.46	3.5
2000	50	40.00	0.716	5.93	5.95
2000	100	20.00	0.818	-0.82	1.54
2000	150	13.33	0.838	-3.28	3.36
2000	200	10.00	0.860	-4.47	4.49
2000	250	8.00	0.874	-4.63	4.63

Table 2: The splitting results of artificially mixed texts from three languages. The *d* and *l* parameters are the length of the document and the fragment respectively, and *d/l* is the average number of fragments in a document. For each *d* and *l* we calculated the average evaluation measures.

From Table 2 we can see how the measures behave on various document and fragment lengths. First of all, it is easy to see that as *l* grows, the correct word percentage grows as well, regardless of document length. This is rather intuitive, since longer fragments are easier to recognize and classify. The FCR is obviously strongly dependent on the number of fragments, and if the number of fragments in a document grows it is harder to accurately estimate it. We can notice that, although the algorithm splits the document into fragment of 40 characters, if the average fragment length is 50 characters, the algorithm underestimates the number of fragments (splits it into fewer fragments than needed). When the average fragment length is 100 or more, the algorithm overestimates the number of fragments. The last observation is that ED is very close to FCR, probably due to the low rate of misclassification, so further test will consider only the correct word percentage and the fragment count ratio.

3.4.4 Feature evaluation

3.4.4.1 Neighboring fragments

The first enhancement to consider is the way a fragment's classification is affected by neighboring fragments. To do that, we begin by checking if adding the cosine distance of the closest fragments will enhance the algorithm performance. We define $Score_{f,l} = Dist(l,f) + a * \left(NeighborDist_{l,f}(1) + NeighborDist_{l,f}(-1)\right)$. For the test, we set *a=0.4*

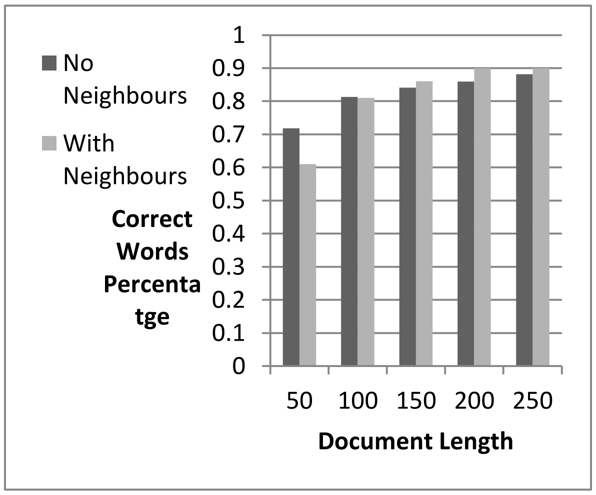

Figure 8: The word percentage of the algorithm with considering neighbors and without them as a function of l (d was chosen to 1500).

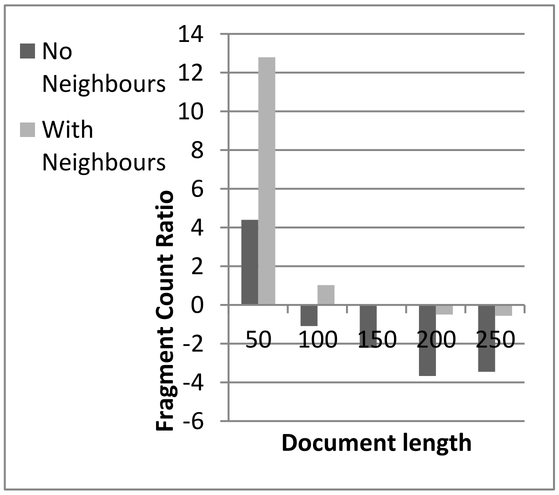

Figure 7: The fragment count ratio of the algorithm with considering neighbors and without them as a function of l (d was chosen to 1500).

We can see that on long fragment lengths, the neighboring fragments improve classification, while on shorter ones, classification without the neighbors was superior. It is not surprising that by using neighbors the splitting procedure tends to split the text into longer fragments, which has a beneficial effect only if fragments actually are longer. We can also see from Fig. 7 that the FCR is now positive with *l=100* which means the algorithm underestimates the number of fragments even when each fragment is 100 characters long. By further experiments, we can see that the *a* parameter is not significant, and we fix it to be 0.3.

As expected, looking at neighboring fragment can improve results in most cases. The next question to be asked is if farther neighbors can improve it also. We try the following scoring function: $Score_{f,l} = Dist(l,f) + \sum_{k=1}^{N}(\frac{a}{k})\left(NeighborDist_{l,f}(k1) + NeighborDist_{l,f}(-k)\right)$. N stands for the longest distance of neighbors to consider in the score. The parameter a is set to 0.3.

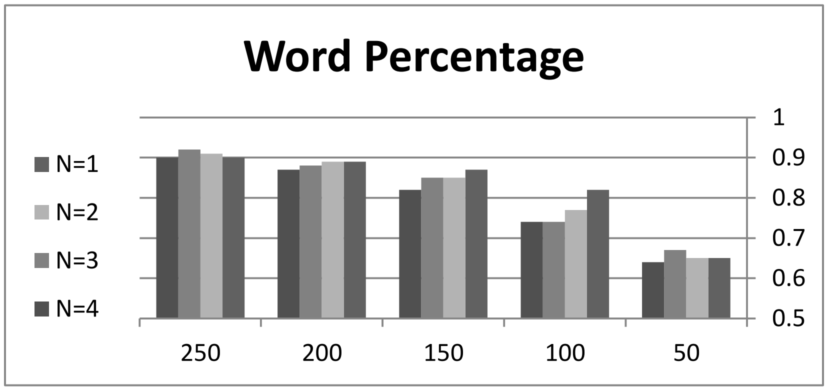

Figure 9: The word percentage of the classification for different values of N

We can see that increasing N does not have a significant impact on the algorithm performance, and on shorter fragment lengths performance drops with N. We conclude that there is no advantage at looking at distant neighbors and looking at the closest fragments is enough.

3.4.4.2 Clustering

The next thing we test is how the clustering method described above can enhance the algorithm. As stated before, the clustering refines a fragment's classification by classifying similar fragments in the same document together which can allow for more accurate classification, since texts are longer. There are several parameters to consider: since the clustering method is hierarchical, there needs to be some similarity score below which we stop clustering. We set this similarity to 0.55, meaning two fragments that have lower similarity then 0.55 cannot be clustered together.

Fragment 1	Fragment 2	Cosine similarity
לעני אלהים סח יב אדני יתן אמר המבשרות צבא	את תמר אחתו יג לג ועתה אל ישם אדני המלך	0.56
אישתא עלתא הוא קורבן דמתקבל ברעווא קודם	למדבחא עלתא הוא קודם לאתקבלא ברעווא קורבנא	0.78
לבדו מת כי על פי אבשלום היתה שומה מיום ענתו	על אדמת עמי קוץ שמיר תעלה כי על כל בתי משוש	0.52
לעיני בני עמי יהבתה לך קבר מיתך וסגיד אברהם	כספא דמי חקלא סב מיני ואקבר ית מיתי תמן	0.49
אל בית ישראל ואפתח את פי ויאכילני את המגלה	את המגלה הזאת אשר אני נתן אליך ואכלה ותהי	0.61

Table 3: Fragments couples and their cosine similarity

To gain some perspective Table 3 demonstrates fragment couples with their cosine similarities. We can see that fragments with over 0.6 similarity usually have common words (even long ones), a fact that makes it reasonable to assume they are in the same language. When similarity drops, the fragments look more random and we do not want to cluster them together for classification.

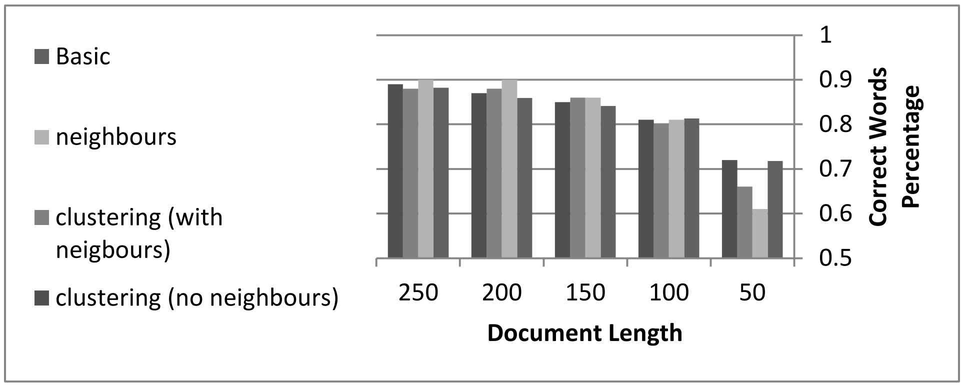

Figure 10: The word percentage rate of the algorithm with and without a clustering phase.

As seen in Figure 10, the clustering phase does not modify results dramatically, compared to the other features of the algorithm. It can be explained by the fact that clustered fragments were already correctly classified, where the mistaken fragments that needed their classification corrected were not clustered because of their anomaly.

3.4.4.2 Post-Processing

Another thing we test is the post-processing of the results splitting to refine the initial fragment choice. We try to move the transition point from the original position to a more accurate position using the technique described above. We note that it cannot affect the FCR measure, since we only move the transition points without changing the classification. As shown in Figure 4.5, it does improve the performance for every value of *l*.

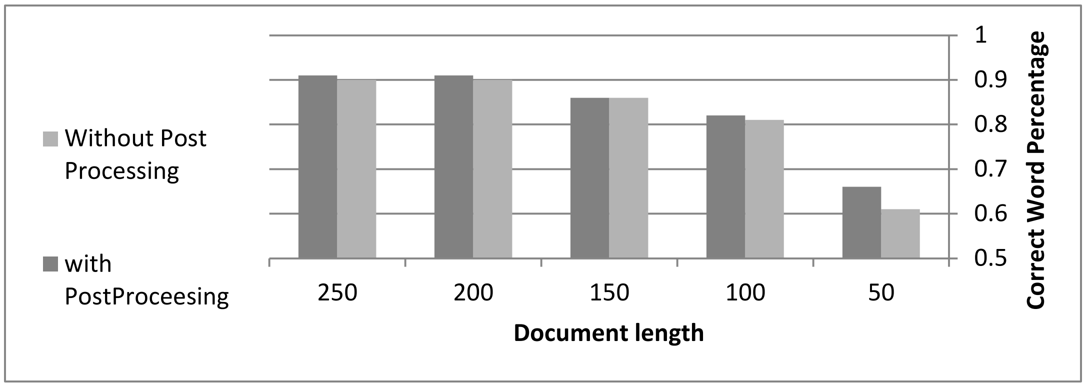

Figure 11: The correct words percentage of the algorithm with and without post-processing (the N value was set to N=5) as a function of l

3.4.5. Sentence accuracy

To test the success rate on sentences, we do the same procedure as for words, but the classification and mixed fragment creation works at sentence granularity. For simplicity, we mark 8 consecutive words of the same language as a sentence and mark the end of it by '.'. In the artificial test creation phase, each fragment contains several language of each language (instead of creating fragments with the number of characters, we now create it according to the number of sentences). In the splitting phase, we do not split it at arbitrary word, since it is certain that each sentence is monolingual. Therefore, we skip the refinement stage at the end of the algorithm and test how good the sentence rate classification is (what percentage of the sentences were recognized correctly), and the improvement of the algorithm using neighboring fragment data. We note that for short fragment length documents each fragment contains only one sentence, so the most we can expect on those documents is the accuracy of language classification on sentence length (about 30 characters) texts. The results are given in Table 4, and we can see that for low *l* values, the success rate is even lower than the word percentage, since it uses only language classification of sentences (the neighboring data only decreases accuracy in this case since neighbors are surely have different language). For longer fragments, the classification rises above the words percentage.

l	Correct words percentage	Correct sentence percentage (no neighbors)	Correct sentence percentage (with neighbors)
50	0.72	0.68	0.59
100	0.81	0.84	0.81
150	0.84	0.87	0.88
200	0.86	0.87	0.92
250	0.88	0.88	0.93

Table 4: The percentage of the sentences correctly identified by the algorithm, with and without neighboring fragments data, compared to the percentage of correct words percentage.

3.4.6. NOISE REDUCTION

To test the noise reduction, we artificially introduce noise in the text by randomly replacing some letters with the *"$"* character. We denote the desired noise rate by *P* and for each letter independently replace it with the "$" character with probability *P*. Since the replacement of each character of the text is mutually independent, we can expect a normal distribution of the error positions in the text and the correction phase described above does not assume anything about the error creation process. The error creation does not assign different probabilities for different characters in the text unlike natural OCR systems or other noisy processing.

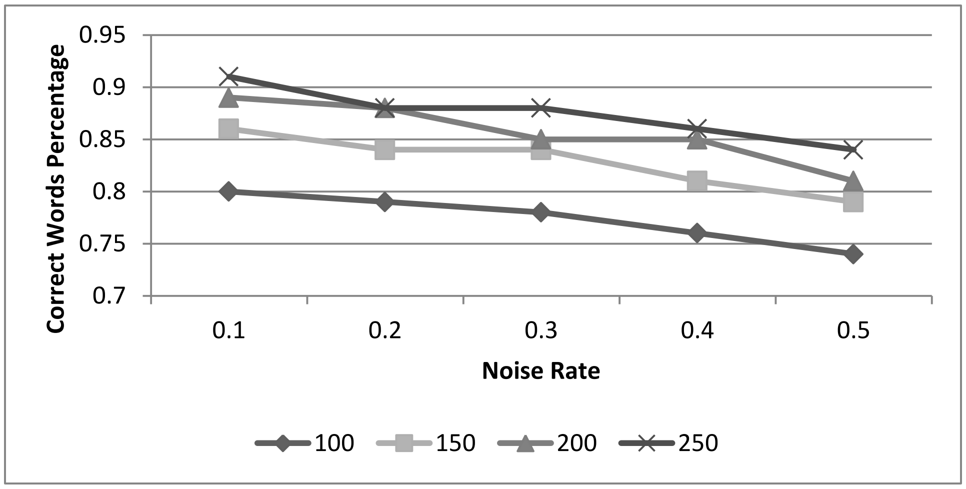

Figure 12: The algorithm word accuracy as a function of the noise rate P. Each line shows the reduce in accuracy for every fragment length

Not surprisingly Figure 12 illustrates that the accuracy decreases as the error rate rises. However, it does not significantly drop even for very high error rate, and obviously we cannot expect that the error reducing process will perform better than the algorithm performs on errorless text.

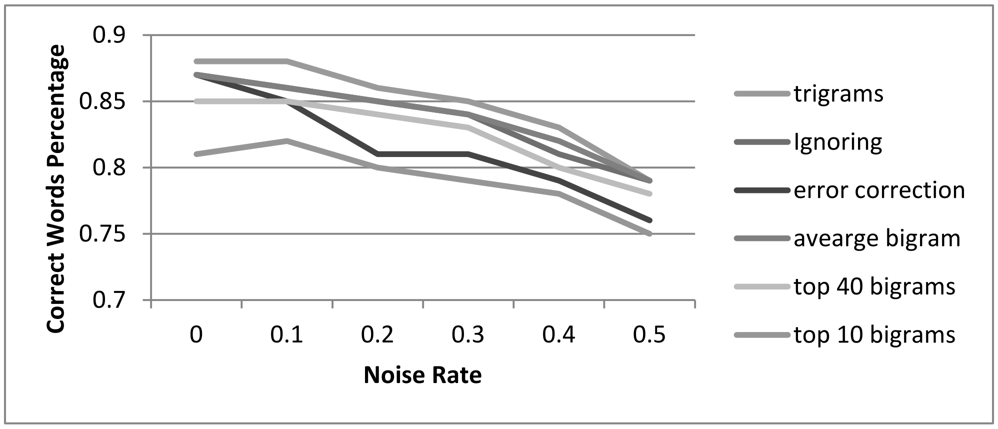

Figure 13: The performance of the correction methods above as for each error rate.

Figure 13 illustrates the performance of each method. It seems that looking at the most common n-grams does not help nor is correcting the unrecognized character. Ignoring the unrecognized character, using either bigrams or trigrams, and estimating the missing unrecognized bigram probability, show the best (and pretty similar) results.

CHAPTER 4

CORPUS SEARCHING ERROR CORRECTION

The ability to identify the language of documents opens possibilities for creating simple catalogues and enhances search options for digitalized documents. As for text produced by an OCR process, it opens up the option of post-processing the text to enhance OCR accuracy. This is especially important for extremely noisy OCR processes, where the produced text cannot be used without further improvement.

A common technique in OCR post-processing is approximate string matching. We assume the text is a part of some big known corpus, and the problem is reduced to finding the correct sub-text in the corpus that corresponds to the processed document. In processing Hebrew manuscripts in the Genizah corpus, it highly likely that the manuscript is part of some known book, and searching for it can reveal the text that the OCR could not accurately recognize. Assuming we identified the language using the techniques from the previous chapters, we can use those language models also to improve the OCR result. This is relevant for extremely noisy texts, for which searching approximate in the corpus may not provide with good results.

4.1 Background

String searching is a well-studied problem in computer science with many established algorithms and strong theoretical background. The basic problem of finding a string in a long text has well known solutions such as the Knuth-Morris-Pratt (KMP) and Rabin-Karp algorithms, which are linear in the text size. The problem of approximate string matching is a generalization of this problem, where the goal is to find a substring of the text that best matches some search pattern, where matches are ranked using some distance functions. Approximate string matching has applications in variety of fields in computer science especially computational biology, information retrieval and spell checking.

4.1.1 Edit-Distance

To deal with approximate matching, we first need to define what an approximate match is. This of course depends on the application, but one of the most popular similarity measures for strings is edit-distance, also known as Levenstein distance. The edit distance between two strings is defined by the minimal number of edit operations needed to be performed on one string for it to exactly match the other. The edit operations allowed in the basic form of edit distance are insertion, deletion and substitution of letters. For example, given the words "train" and "ruins" we can perform 3 edit operations: deleting the 't' letter (getting rain), substitution of 'a' by 'u' (getting ruin) and inserting the 's' character at the end to get "ruins". Hence, the edit distance is 3.

Edit distance is highly suitable for OCR correction purposes, since the allowed edit operations are pretty consistent with the errors an OCR engine may perform. It is frequent for an OCR to skip a character or to recognize some irrelevant symbol as a letter (insertion and deletion operations), and of course confuse one character with another (substitution). We can use edit distance to approximate the probability the OCR engine will produce one string from the other, in the sense that the lower the edit distance is , the higher the probability for the OCR to produce one string as an output on the second string. A generalized version of the edit distance problem assigns different weights for insertion of each character, deletion of each character and substitution of each character with each other character. This can match the different probabilities of mistakes made by the OCR engine (it is more likely for the OCR to replace two characters that have geometric similarity). Other string distance functions do not reflect the nature of OCR engines. The popular Hamming distance allows only substitution of letters, but gives infinite distance for two strings with different lengths, which will assign zero probability for an OCR to miscalculate the length of its input (which is way above zero obviously). The generalization of Levenstein distance, Damerau-Levenstein, is popular in spelling correction because of the additional edit operation of substitution of two letters. As opposed to human typing, which have high probability of confusing character order, OCR scans the text linearly, so this function does not suit our requirements. Other string distance function popular in natural language processing usually use phonetic or semantic word properties, while OCR usually uses geometric properties of characters.

The edit distance between two strings is usually computed using a dynamic programming procedure. The computation complexity is *O(n*m)* where *n* and *m* are the lengths of the two strings.

4.1.2 APPROXIMATE STRING MATCHING METHODS

There are several approaches to the approximate string matching problem. Some of them are mainly theoretical in nature, where the practical ones are dynamic programming, filtering and indexing.

Dynamic programming techniques are a search generalization of the distance computation method, by trying to compute distance from every possible starting point in the text. The run time of those methods is usually close to *O(n*m)* where n is the text length and m is the pattern length. The main drawback of those methods is the large space requirements that dynamic programming matrices need to manage. For large corpora, this can make it inapplicable, since there are searches of patterns of thousands of characters in texts of tens of millions of characters.

Filtering methods use heuristics to eliminate indexes in the text that cannot be the best solution. There is usually some fast scanning of the text, which will make the search phase more efficient. The search phase usually makes use of dynamic programming techniques, so worst case scenarios will be similar to the dynamic programming complexity. Those methods are much more convenient for our purpose.

Indexing methods preprocess the text, which in turn enhances the matching procedure. Those methods are suitable for applications that perform multiple searches on the same text. The indexing is usually computationally expensive and has high memory consumption and the search algorithms are complex and difficult to modify.

4.2 Error rate estimation

Applying string matching techniques for correction of OCR process has unique properties, due to the unknown accuracy of the OCR process. Although edit-distance is suitable for estimating the probability of a string being produced by the OCR, there is still no guarantee that the closest substring in a text in terms of edit distance is the actual string. In other words, we want to estimate how efficient application of string matching techniques to OCR correction problem is and what OCR accuracy is needed for it to be efficient.

4.2.1 Single Letter alignment

The first thing we test is the application of string matching techniques, under the assumption the OCR can accurately recognize only one character of the alphabet. We first try exact matching, meaning we assume the OCR recognizes the letter perfectly. For this purpose, we choose an arbitrary string in the text, mask it, in the sense we leave only one letter in the string and turn all other characters into the '\$' sign, and search for it back in the text. We denote by $\text{Align}_l(n)$ the average number of alignments for substrings of length n (over 500 random strings) which was masked to contain only the letter l. We used the Bible, which is 1505034 characters long, as the text to search in. We expect the number of alignments to rise as the text grows.

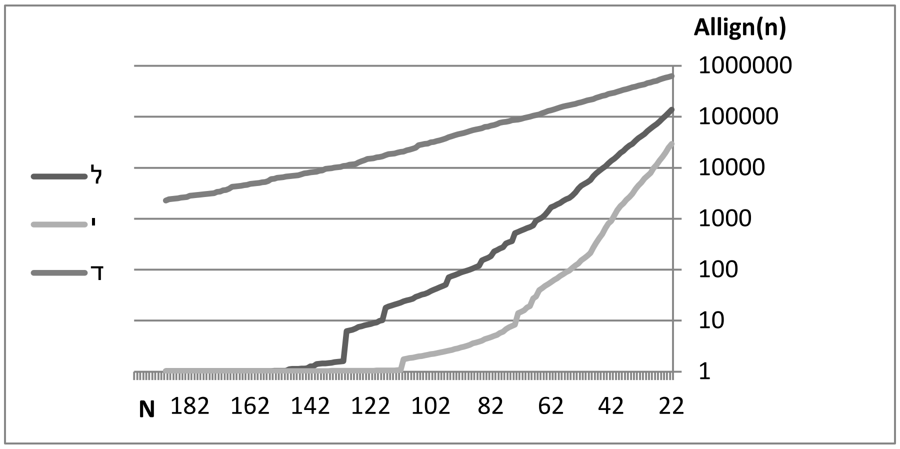

Figure 14: The number of possible matches for string masked by three different letters, as a function of the string length.

As shown in Figure 5.1, we can see that for frequent letters (such as ‘י’ which has 10% frequency), the search yields a single match on strings longer than 110 characters. For rare characters, even on 200 character strings, it still has over 1000 correct matchings, making the search irrelevant. So if OCR recognizes only rare characters we demand much longer documents.

4.2.1.1 Single letter with errors

After testing the straight forward approach of exact matching, we try to extend it to inexact substrings. We still assume the availability of only one letter. Suppose the substring is a copy of some fragment of the original text, while errors can be taken place in the copying process. We can look at a probability matrix of copying errors. Suppose we have a probability of P_1 to get a letter *l* in the copy even when the original text contained something else. It means that we have a probability of only $1 - P_1$ to get another character (denote it by *$*) in the copy when there appeared *$* in the original text. Symmetrically, we define by P_2 the probability of getting *$* in the copy when the original text contained *l,* and it means we only have $1 - \mathrm{P}_2$ probability to copy the letter *l* correctly.

For the test, we set *P1=0* and *P2=0.05*. This means that the OCR does not produce “false positives” and identify *l* where it did not appear. It does have a 0.05 probability to miss a character *l* and produce something else. For example given a string “אדבאדהאדבאדא” which contains 4 occurences of the character 'ד', the probability of getting “$ד$$ד$$ד$$ד$” (exactly correct OCR) is 0.95^4 , since the probability of correct recognition of the letter is 0.95. The probability for “$$$$ד$$ד$$ד$” is $0.95^3 \cdot 0.05$ (3 correct identifications and one mistake). This can be the nature of a very conservative OCR, that identifies some character only when there is very high probability it is actually it and therefore does not produce false positives.

The test we produced selects an arbitrary string, leaving only a character l in the string, while with probability *P2* we replace the appearances of *l* into *$*. Then the new pattern is searched in the text and the match with the highest probability (as defined above) is returned. We are interested to know when the returned match is actually the string that was selected and masked. We use the character 'ל' as *l,* which is an average letter with 0.05% appearance.

Pattern length	Percentage of correct matches	Average rank of original string
500	1.000	1.000
350	1.000	1.000
250	0.995	1.025
200	1.000	1.000
160	0.985	1.015
120	0.945	14.385
100	0.860	21.610

80	0.675	514.93
60	0.375	1760.85

Table 5: The percentage of correct matches of the patterns with errors searched in the Bible. The second rows shows the average rank (in probability terms) of the correct string

As shown in Table 5, for fragments of 200 characters and longer, there is high probability for the best match to actually be the correct fragment. For fragments longer than 100, the results are reasonable; below that results are poor, so we cannot expect string matching to show good performance. When the error rate rises, the success rate drops as shown in Figure 15. The performance is reasonable for error rates below 25%.

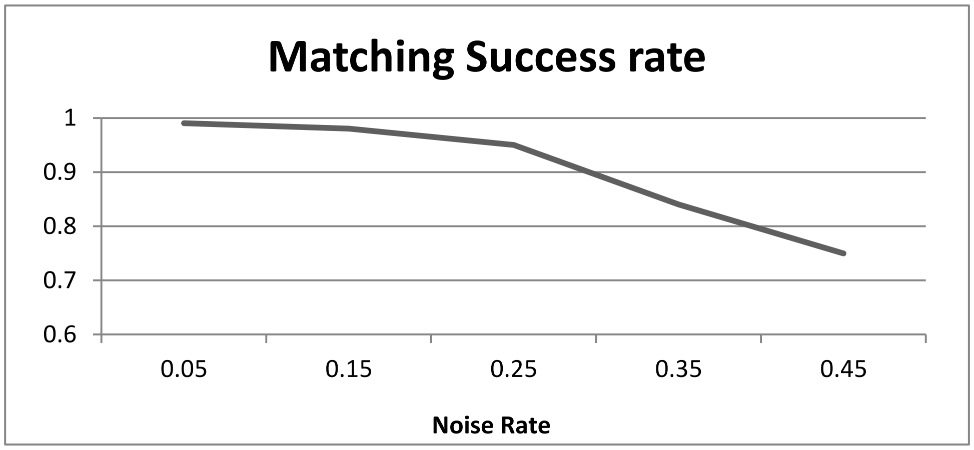

Figure 15: The percentage of correct matches of the patterns with errors searched in the bible as a function of the OCR error rate. The patterns are 200 characters long

4.2.2 MULTI LETTER ALIGNMENT

When scaling to several letters, we obviously expect the searching success rate to increase. It is obvious due to the higher rate of recognized characters but also due the veracity of symbols needed to be matched. Figure 16 illustrates the success rate for matching patterns containing various amounts of different letters, and we can see accuracy increases with the increase in number of letters.

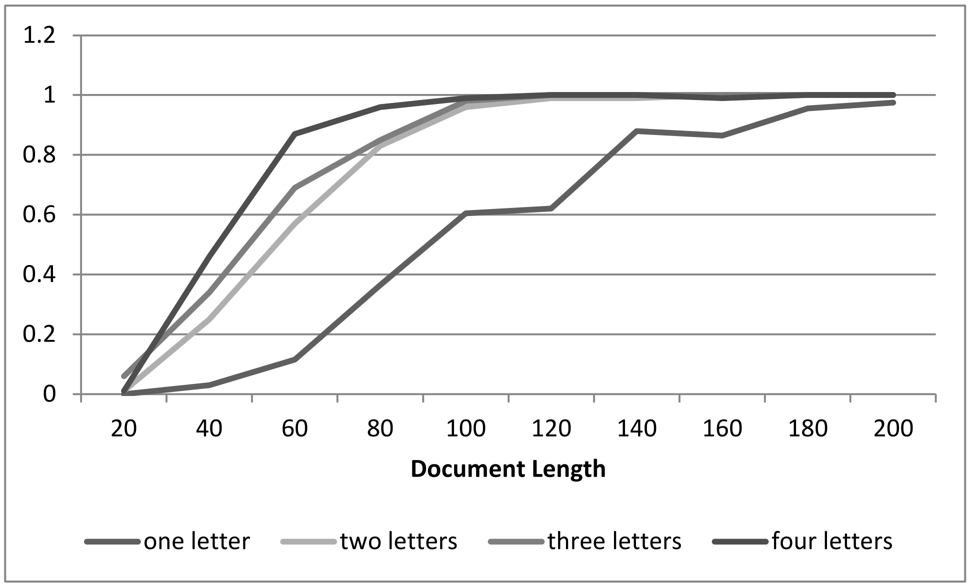

Figure 16: The percentage of a single match rates for a pattern in for various available letter numbers as a function of fragment length.

4.3 Proposed Algorithm

We propose an algorithm for post-processing OCR results by approximately searching a corpus of text. The algorithm has to deal with relatively large patterns and corpora, and the fact that the pattern is extremely noisy, so the search result may be considerably different from the pattern.

4.3.1 THE INPUT

The corpus is a long text (several millions of characters) denoted by T. The OCR results are given by the following: for each character in the OCR'ed text P_i, we get a set of characters $C_{i1} \dots C_{i,k}$ and a set of probabilities $P(C_{i,1}) \dots P(C_{i,k})$ which stand for the probability that the character I of the pattern (the scanned text) is $C_{i,k}$. Notice that the $\sum_{n=1}^{k} P(C_{i,n})$ do not necessary add to 1, since some of the probabilities are neglected. For most characters only one option $C_{i,k}$ is given.

We assume that the accuracy of the OCR is pretty low, which means that only about half of the top scored letters produced by the OCR are correct. The probabilities of each character are not high (most of them below 0.5, which is just a good guess), and in addition insertions and deletions of characters can occur frequently. On the other hand, the pattern is assumed to be quite long so, we can use that fact to perform a more accurate search.

4.3.2 The Algorithm

The proposed algorithm is actually a filter algorithm, where the filtering is heuristic, and uses substring of the pattern. The length of the corpus and fragment makes it very difficult to run a classic dynamic programming algorithm. The fact that the input is so noisy will make most filter algorithms inefficient.

The algorithm proposed follows the following scheme:

1. Clean the corpus and the pattern from characters that are poorly recognized.
2. Iteratively choose a substring of the pattern and search for it in the corpus
3. Combine the results into the one best result

The first stage handles the cleaning of the pattern from the noisy characters. We estimate the probability of each character to be the average of the probabilities attached to the character at all its occurences. More formally, $\mathrm{P(l)} = \mathrm{Average_i}\ \mathrm{P}(\mathrm{C_{i,k}})|\mathrm{P}(\mathrm{C_{i,k}}) > 0\ and\ \mathrm{C_{i,k}} = \mathrm{l}$. If the OCR engine proposed the character l as an option for the I'th place of the pattern, we consider it in the average of the probabilities of l. The estimation P(l) is an approximation for how accurate the engine handles the character l. The estimation is a combination for the OCR recall and precision on the character. After the estimation of accuracies for all characters we pick a threshold below which all letters will be neglected.

The threshold should be estimated empirically and depends on the accuracy of the OCR. As shown in the previous section, even one character with reasonable error rate can be enough for searching a fragment of even medium length. Therefore the threshold should be set high enough for the extra recognized characters to increase rather than decrease the success of the search. All characters below the threshold are replaced by an "unknown" character both in the pattern and the corpus.

The second step is iterative search. In each iteration, we select a substring of the pattern and search for it in the corpus. The length of the substring should be the shortest possible, such that, using the characters chosen in step 1, the search is expected to return one result only. In other words, we would like to search for a sub-pattern long enough so the search will return match (or matches) in the corpus with high probability. When searching, we use the dynamic programming technique, assuming the sub-pattern is relatively short for efficient search. The result of each search is an interval [a..b], where a and b are the starting and ending indexes of the match in the corpus. We define the following variables:

l- the sub-pattern length, **L**- pattern length

r – the index of the sub-pattern in the pattern

a,b – the indexes in the corpus that define the match of the sub-pattern

Insert ratio - a parameter estimating the frequency of insertion mistakes made by the OCR engine.

From a, b, we define an interval [i1..i2] which will estimate the position of the whole pattern in the string. Since the sub-pattern is in indexes [a..b] we naturally would expect to find the whole pattern at indexes [a-r…b+(L-(r+l))], by extending the match of the interval by the distance of the sub-pattern the pattern boundaries. Since we expect some insertion mistakes we extend the interval to [a-(1+*InsertRatio)**r…b+(1+*InsertRatio*)(L-(r+l))] to estimate the alignment of the pattern with the mistakes. The estimated interval is the result of each search iteration.

In the third step we combine all intervals into a single result. The combination algorithm is as follows:

```
1. Initialize $IntervalsList \leftarrow \emptyset$
2. For each [i1,..i2] in Results
   2.1. If $IntervalsList$ contains Interval such that $Interval.union \cap [i1..i2] \neq \emptyset$
        2.1.1. $Interval.union \leftarrow Interval.union \cup [i1 \ldots i2]$
        2.1.2. $Interval.intersection \leftarrow Interval.intersection \cap [i1 \ldots i2]$
        2.1.3. $Interval.count \leftarrow Interval.count + 1$
   2.2. Else
        2.2.1. $IntervalsList \leftarrow IntervalsList \cup Interval(intersection = [i1..i2], union = [i1 \ldots i2], count = 1)$
3. $IntMax \leftarrow The\ interval\ in\ IntervalList\ with\ maximal\ count$
4. [a..b] = The search of pattern in Intmax.Intersection
5. Return [a..b]
```

The combination algorithm is based on the fact that the corpus is significantly longer than the pattern. Given an interval returned, we compare it to all other intervals returned by other searches. If two intervals intersect, they're union is considered to be one (longer) interval. If an interval does not intersect with others, it is considered new. The algorithm tries to unify all intervals and counts the number of intervals joined in each union and we expect only one interval to be counted a significant number of times and classify all others as noise. We then search for the fragment back in the interval to return the final result.

4.4 Testing

The testing was done on several outputs of an OCR system operated on Genizah fragments. The input was as described in the section 2, where the number of iterations was 20 and the sub-pattern length was 50. The minimum accuracy, under which characters were omitted, was set to 0.5.

Original Text	OCR result	OCR output with prob > 0.5	Algorithm Output	Best Match
שמייםליהוהוהארץנתןלבנ יאדםלאהמתיםיהללויהול אכלירדידומהואנחנונברךי המעתהועדעולםהללויהא הבתיכיישמעיהוהאתקולי תחנוניכיהטהאזנוליובימי אקראאפפוניחבלימותומצ רישאולמצאוניצרהויגוןאמ צאובשםיהוהאקראאנהיהו המלטהנפשיחנוןיהוהוצדי קואלהינומרחםשמרפתאי םיהוהדלתיולייהושיעשובי נפשילמנוחיכייהוהגמלע ליכיכיחלצתנפשיממותאת עינימןדמעהאתרגלימדחי אתהלךלפנייהוהבארצות החייםהאמנתיכיאדבראני	שסיסלידודארץכען לבני אדסלאדתיילוידולאכליו רדידומואשנונברךידעת דעולסדלוידאדזבעיכיש עידודארוליעחנוניכידסר אזנויוביסיארראאוניחבל ישתומצרישאולצאוניצדו גוןאמצאובשםידואררא אנאידודלטכשינוןידודוצ דידואדינוסרחסשוערת איסיזוזדלייייליידושיעשו בינשילסנוחיכיידודגמ לעיכיכיחלצתנשיממועא עעיניסדסעדאערגליאע דלךלניידודבארדחייפד אסנעיכיאדבראניגליאע דל	ש$י$$ידוד$רץ$$ן$ב $יאדס$אד$יי$וידולא כליורדידו$ו$$נו$$$ך יד$$ד$$לסדלוידאד$ ב$יכי$$ידוד$$ו$י$ח נו$י$יד$ראזנויו$י$יא $ר$או$י$בלי$$ו$$ר יש$ו$צאוני$דוגו$א$ $א$$שמי$וא$ראא$ אידודל$$שי$$ןידודו $די$ואדינו$רחסשו$ ר$$י$י$ו$ד$י$יי$יידו שיעשובי$$י$$נוחיכי כיידוד$$לעיכי$יחל$ $$שי$$ו$א$עינ$$ד $$דא$רג$יא$דלךל$ יידו$$$רד$י$פדא$$ $יכי$דברא$יג$יא$ד ל$	והארץנתןלבניאדםל אהמתיםיהללויהולא כלירדידומהואנחנונ ברךיהמעתהועדעול םהללויהאהבתיכיי שמעיהוהאתקולית חנוניכיהטהאזנוליוב ימיאקראאפפוניחבל ימותומצרישאולמצא וניצרהויגוןאמצאוב שםיהוהאקראאנהי הוהמלטהנפשיחנוןי הוהוצדיקואלהינומר חםשמרפתאיםיהוה דלתיולייהושיעשובי נפשילמנוחיכייהוה גמלעליכיכיחלצתנפ שיממותאתעינימןד מעהאתרגלימדחיא תהלךלפנייהוהבאר צותהחייםהאמנתיכי אדבראני	ליהוה והארץ נתן לבני אדם לא המתים יהללו יה ולא כל ירדי דומה ואנחנו נברך יה מעתה ועד עולם הללו יה אהבתי כי ישמע יהוה את קולי תחנוני כי הטה אזנו לי ובימי אקרא אפפוני חבלי מות ומצרי שאול מצאוני צרה ויגון אמצא ובשם יהוה אקרא אנה יהוה מלטה נפשי חנון יהוה וצדיק ואלהינו מרחם שמר פתאים יהוה דלתי ולי יהושיע שובי נפשי למנוחיכי כי יהוה גמל עליכי כי חלצת נפשי ממות את עיני מן דמעה את רגלי מדחי אתהלך לפני י
ענייתימאדאניאמרתיבחפז יכלהאדםכזבמהאשיבליהו הכלתגמולוהיעליכוסישועו תאשאובשםיהוהאקראנד ריליהוהאשלםנגדהנאלכל עמויקרבעינייהוההמותהל חסידיואנהיהוהכיאניעבדך אניעבדךבןאמתךפתחתל מוסרילךאזבחזבחתודהוב שםיהוהאקראנדריליהוהא	עניסאדאניאסדעיבהזיכלד אדסכוזככלדזאדסכוזכללד אשדודכלעגמולודיעליכוסי שועועאשאובשסידזודזאד ראכדרילידודאשלנגדדנאל כלעמויבעיניידודדמותדלחי דיואנידודכיאניעכדךאניעב דךבןאמתךעחתלוסרילדא זבחזבהתודדובשסידודאד זראנדרילידודאשלנגדדלכ	$$ י$$ד$נ$א$$$יב$ז$ כ$ד$דסכוז$כ$דז$ד סכוז$$$ד$שדודכ$$ $$ו$ודי$$יכוסישו$ו$ אשאובש$יד$וד$א$$ $$$ר$י$וד$ש$נ$ד ד$$לכל$$ויבעי$יידוד ד$ו$דל$ידיוא$ידודכי א$י$כדךאני$בדךבן$	יתימאדאניאמרתיב חפזיכלהאדםכזבמ האשיבליהוהכלתגמ ולוהיעליכוסישועות אשאובשםיהוהאקר אנדריליהוהאשלםנג דהנאלכלעמויקרבעי נייהוההמותהלחסיד יואנהיהוהכיאניעבד ךאניעבדךבןאמתךפ	אני אמרתי בחפזי כל האדם כזב מה אשיב ליהוה כל תגמולוהי עלי כוס ישועות אשא ובשם יהוה אקרא נדרי ליהוה אשלם נגדה נא לכל עמו יקר בעיני יהוה המותה לחסידיו אנה יהוה כי אני עבדך אני עבדך

שלםנגדהנאלכלעמובחצרותביתיהוהבתוככיירושלםהללויההללואתיהוהכלגויםשבחוהוכלהאמיםכיגברעלינוחסדוואמתיהוהלעולםהללויההודוליהוהכיטובכילעולם	לעובהותביתידודבעוככייושלסבתלדזודזבעככושלסדללוידדללזאעידודכלגוידזללזאעלוזכלגושכחזדוכלדאומיסכיגכעליוחסדוואעיעיעולסדללוידشוכלדאומיסכיגכעליוחסדווא$	$$ך$$$לו$ריל$$זב $ו$'שס$ודדו$$ב$$ א$$ר$$ד$ילידודאש ל$$דדלכ$$וב$ו$בי$ $דוד$$וככייוש$$ב$ $ד$וד$$$ככוש$$ד$ $וידדלל$$$'י$ודכ$$ו ידזלל$$$לו$כ$$וש$ $$דוכלד$ו$'י$כיג$$ל יו$$דווא$$יו$ולסד$ל וי$شוכלד$ו$'י$כיג$$ ליו$$דווא$	תחתלמוסרילךאזבחזבחתודהובשםיהוהאקראנדריליהוהאשלםנגדהנאלכלעמובחצרותביתיהוהבתוכיירושלםהללויההללואתיהוהכלגויםשבחוהוכלהאמיםכיגברעלינוחסדוואמתיהוהלעולםהללויההודוליהוהכיטובכילעולםחסדויא	בן אמתך פתחת למוסרי לך אזבח זבח תודה ובשם יהוה אקרא נדרי ליהוה אשלם נגדה נא לכל עמו בחצרות בית יהוה בתוככי ירושלם הללו יה הללו את יהוה כל גוים שבחוהו כל האמים כי גבר עלינו חסדו ואמת יהוה לעולם הללו יה הודו
כבשיביאקרבנולחטאתנקבהתמימהיביאנהוסמךאתידועלראשהחטאתושחטאתהלחטאתבמקוםאשרישחטאתהעלהולקחהכהןמדםהחטאתבאצבעוונתןעלקרנתמזבחהעלהואתכלדמהישפךאליסודהמזבחואתכלחלבהיסירכאשריוסר	לישייאררונרזתמיריבין|ורעיהעלרשןרמושמאררחמברושרישמדעלרייין|ודןחרןרעבאצכיןזעלידמנץרלרואכדמרזושרליסזדורמזבחואכלחלסישיומרו	$$$ ייִ$$ר$נר$$$'י$$בין|ו $$$ו$$ןרש$ע$$$$ $$$$$$$$ו$$$$$$ $$$$$$ן$$ן$$ן$$'י$ $ו$$$$$$$$$$$$ו$ $$$$$$$$$$$$$$ שיו$$$$$$$$ו$$ז$$ $$$	לואחותתובלקין|נעמהויאמרלמךלנשיועדהוצלהשמעןקולינשילמךהאזנהאמרתיכיאישהר	להויאמרואליואישבעלשערואזורעוראזורבמתניוויאמראליההתשביהואוישלחאליו
החטאתבמקוםהעלהולקחהכהןמדמהבאצבעוונתןעלקרנתמזבחהעלהואתכלדמהישפךאליסודהמזבחואתכלחלבהיסירכאשרהוסרחלבמעלזבחהשלמיםוהקטירהכהןהמזבחהלריחניחחליהוהוכפרעליוהכהןונסלחלוואם	ריחטאבץזרעליוררכדןמדמריבבעוןעלרריזברערזאידכלדמריישךאליסורזבחיורכלרלבריסיבשרזסלבמזזרשיוררמיררכרמזבדלחגיההיבירג	$$$$$$$$$$$ יו$ר$$ן$$$$$$$$עוו $$$$$$$$ז$ר$$$$$ $$$$$$שך$$$$$$$$ $$$$$$$ר$$$$$$ $'י$ש$$$$$$$$ז$שי $$$$$$$$$'ר$$ר$ '$$$$$$$$	מימאשרמעללרקיעויהיכןויקראאלהיםלרקיעשמיםויהיערבויהיבקריוםשניויאמראלהים'	גידביזרעאלוירכביהואוילךיזרעאלהכייורםשכבשמהואחזיהמלךיהודהיר
והקטירהמזבחהעלאשייהוהחטאתהואוכפרעליוהכהןעלחטאתואשרחטאמאחתמאלהונסלחלווהיתהלכהןכמנחהוידבריהוהאלמשהלאמרנפשכיתמעלמעלוחטאהבשגגהמקדשייהוהוהביאאתאשמוליהוהאיל	וירזסרדימזרשייחטריזברעליורדןעמעורחזמאיחמלרונלוורירלככהנחרןוידביאמירןאמנכישכירסלמעלוזץטארבמרייורבירזזיויל	ו$$$$$$$$$$$$'יי$ $$'י$$$ע$$$$$ן$$ $ור$$$$$$$$$$$$$$ו $$'ר$$$$נ$$$ו$$$ '$$$$$$$$$'יש$'י$$ יו$$$רב$$$$$$ע$$ וי$ז$$'יב$$	והארץהיתהתהווהבווחשךעלפניתהוםורוחאלהיםמרחפתעלפניהמיםויאמראלהיםיהיאורויהיאוריראאל	ויעלמשםבארשבעויראאליויהוהבלילהההואויאמראנכיאלהיאברהםאביךאל

Table 6: The algorithm results for several fragments. The columns show the original text, the text as transcripted by the OCR, the transcript after the character omission, the algorithm results and the best edit distance match of the string to the corpus

As seen in Table 6, the algorithm presents good results and approximates the original fragment well for reasonable OCR performance. The results are similar to the best edit distance matching. For extremely noisy OCR output, the results are not accurate, since edit distance cannot be a good approximation for the result and other methods need to be applied.

CHAPTER 5

CONCLUSIONS AND FURTHER RESEARCH

5.1 Conclusions

This thesis presented methods for three slightly different but connected problems. A statistical algorithm for language classification of Hebrew script documents was presented, using bigram distributions. The algorithm showed over 95% accuracy for most of the documents, rising to perfect 100% performance on documents longer than 800 characters. It also showed a method for higher precision the error rate by allowing the classification algorithm to return an unknown result.

An algorithm for segmenting multilingual documents into monolingual fragments was introduced, reaching about 90% percent accuracy on 100-200 character length language shifts. The accuracy for more frequent language shifts was about 70%. Several methods were presented and compared for generalizing the method to handle noisy texts.

Finally, a heuristic filter algorithm for approximate string matching was described. It showed good accuracy results, running significantly faster than classic edit distance algorithms.

5.2 Further research

While the language classification problem has well established methods, language segmentation has many open questions. The algorithm proposed was extremely sensitive to the language shift rate, so a method for approximating this rate can significantly increase performance by smarter parameter tuning. Another method for increasing performance can be a machine learning approach for parameter estimation, a method that was not tried thoroughly enough in this work. A different direction can be to revise the shift smoothing process, by trying a different way than trying constant shift points.

The probabilistic approximate search algorithm needs to be further tested on larger datasets, with patterns of various lengths and noise rates. A more accurate selection of the sub-patterns can be considered. Another direction can be an OCR ad hoc tuning such as considering specific substitution matrixes and error rates and test the change in performance.

REFERENCES

1. Hearst, Marti A. 1993. TextTiling: A quantitative approach to discourse segmentation. Technical Report Sequoia 93/24, Computer Science Division,
2. Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, Frédéric Saubion. On Evaluation Methodologies for Text Segmentation Algorithms. In Proceedings of ICTAI (2)'2007. pp.19~26
3. F.Y.Y. Choi, "Advances in domain independent linear text segmentation", in Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, 2000, pp.
4. Hakkinen, J. Jilei Tian .N-gram and decision tree based language identification for written words. Automatic Speech Recognition and Understanding, 2001. ASRU '01.
5. X. Tong, D. Evans. A Statistical Approach to Automatic OCR Error Correction in Context. In Proceedings of the Fourth Workshop on Very Large Corpora, pages 88-100,Copenhagen, Denmark, August 1996
6. K. Kukich. Techniques for automatically correcting words in text. In ACM Computing Surveys, vol. 24, no. 4, 1992.
7. Hughes B, Baldwin T, Bird S, Nicholson J, and MacKinlay A. Reconsidering language identification for written language resources. In Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006), 485–488, 2006, Genoa, Italy
8. Friedberg Genizah Project website, http://www.genizah.org/TheCairoGenizah.aspx
9. OCR Post-Processing for Low-Density Languages", Okan Kolak, Philip Resnik, Proceedings of HLT-EMNLP '05, 2005
10. S. Chen, D. Misra, and G. R. Thoma. Eficient automatic OCR word validation using word partial format derivation and language model. In L. Likforman-Sulem and G. Agam, editors,Document Recognition and Retrieval XVII, 2010.
11. E. Giguet. 1996. The stakes of multilinguality: Multilingual text tokenisation in natural language diagnosis.In Proceedings of the PRICAI Workshop on Future Issues for Multilingual Text Processing.
12. Navarro, Gonzalo 2001. "A guided tour to approximate string matching". *ACM Computing Surveys*
13. Hal Daum´e III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. InInternational Conference on Machine Learning (ICML), 2005.

Appendix 1 – collected corpora

English name	**Hebrew name**
Targum Onkelus, Targum Unkelus	תרגום אונקלוס
Jerusalem Talmud, Talmud Yerushalmi	תלמוד ירושלמי
Torah, Pentateuch, Five books of Moses	תורה, חומש
Mishnah	משנה
The Guide for the Perplexed, Moreh Nevukhim	מורה נבוכים
Kozari	הכוזרי
Maspik Ovdei Hashem, A Comprehensive Guide for the Servants of God	המספיק לעובדי השם
Tanakh	תנך
Bible commentaries	פרשנות
Chazal	חז"ל
Zohar	זהר
Geonim	גאונים
Rishonim	ראשונים
Achronim	אחרונים
Mishnah commentaries	פרשנות משנה
Minhag, Customs books	מנהג
Mizvot, Commandments book	ספרי מצוות
Machshava, Thought books	מחשבה
Maimonides	רמבם
Tur	טור

Shulchan Aruch (Code of Jewish Law)	שולחן ערוך