Language Classification and Segmentation of Noisy Documents in Hebrew Scripts

Nachum Dershowitz
School of Computer Science
Tel Aviv University
Ramat Aviv, Israel
nachumd@tau.ac.il

Alex Zhicharevich
School of Computer Science
Tel Aviv University
Ramat Aviv, Israel
alex.zhicharevich@gmail.com

Abstract

Language classification is a preliminary step for most natural-language related processes. The significant quantity of multilingual documents poses a problem for traditional language-classification schemes and requires segmentation of the document to monolingual sections. This phenomenon is characteristic of classical and medieval Jewish literature, which frequently mixes Hebrew, Aramaic, Judeo-Arabic and other Hebrew-script languages. We propose a method for classification and segmentation of multi-lingual texts in the Hebrew character set, using bigram statistics. For texts, such as the manuscripts found in the Cairo Genizah, we are also forced to deal with a significant level of noise in OCR-processed text.

1. Introduction

The identification of the language in which a given test is written is a basic problem in natural-language processing and one of the more studied ones. For some tasks, such as automatic cataloguing, it may be used stand-alone, but, more often than not, it is just a preprocessing step for some other language-related task. In some cases, even English and French, the identification of the language is trivial, due to non-identical character sets. But this is not always the case. When looking at Jewish religious documents, we often find a mixture of several languages, all with the same Hebrew character set. Besides Hebrew, these include Aramaic, which was once the lingua franca in the Middle East, and Judeo-Arabic, which was used by Jews living all over the Arab world in medieval times.

Language classification has well-established methods with high success rates. In particular, character n-grams, which we dub *n-chars*, work well. However, when we looked at recently digitized documents from the Cairo Genizah, we found that a large fraction contains segments in different languages, so a single language class is rather useless. Instead, we need to identify monolingual segments and classify them. Moreover, all that is available is the output of mediocre OCR of handwritten manuscripts that are themselves of poor quality and often seriously degraded. This raises the additional challenge of dealing with significant noise in the text to be segmented and classified.

We describe a method for segmenting documents into monolingual sections using statistical analysis of the distribution of n-grams for each language. In particular, we use cosine distance between character unigram and bigram distributions to classify each section and perform smoothing operations to increase accuracy.

The algorithms were tested on artificially produced multilingual documents. We also artificially introduced noise to simulate mistakes made in OCR. These test documents are similar in length and language shifts to real Genizah texts, so similar results are expected for actual manuscripts.

2 Related Work

Language classification is well-studied, and is usually approached by character-distribution methods (Hakkinen and Tian, 2001) or dictionary-based ones. Due to the lack of appropriate dictionaries for the languages in question and their complex morphology, the dictionary-based approach is not feasible. The poor quality of the results of OCR also precludes using word lists.

Most work on text segmentation is in the area of topic segmentation, which involves semantic features of the text. The problem is a simple case of structured prediction (Bakir, 2007). Text tiling (Hearst, 1993) uses a sliding-window approach.

Similarities between adjacent blocks within the text are computed using vocabularies, counting new words introduced in each segment. These are smoothed and used to identify topic boundaries via a cutoff function. This method is not suitable for language segmentation, since each topic is assumed to appear once, while languages in documents tend to switch repeatedly. Choi (2000) uses clustering methods for boundary identification.

3. Language Classification

Obviously, different languages, even when sharing the same character set, have different distribution of character occurrences. Therefore, gathering statistics on the typical distribution of letters may enable us to uncover the language of a manuscript by comparing its distribution to the known ones. A simple distribution of letters may not suffice, so it is common to employ n-chars (Hakkinen and Tian, 2001).

Classification entails the following steps: (1) Collect n-char statistics for relevant languages. (2) Determine n-char distribution for the input manuscript. (3) Compute the distance between the manuscript and each language using some distance measure. (4) Classify the manuscript as being in the language with the minimal distance.

The characters we work with all belong to the Hebrew alphabet, including its final variants (at the end of words). The only punctuation we take into account is inter-word space, because different languages can have different average word lengths (shorter words mean more frequent spaces), and different languages tend to have different letters at the beginnings and ends of words. For instance, a human might look for a prevalence of words ending in *alef* to determine that the language is Aramaic. After testing, bigrams were found to be significantly superior to unigrams and usually superior to trigrams, so bigrams were used throughout the classification process. Moreover, in the segmentation phase, we deal with very short texts on which trigram probabilities will be too sparse.

We represent the distribution function as a vector of probabilities. The language with smallest cosine distance between vectors is chosen, as this measure works well in practice.

4. Language Segmentation

For the splitting task, we use only n-char statistics, not presuming the availability of useful wordlists. We want the algorithm to work even if the languages shift frequently, so we do not assume anything about the minimal or maximal length of segments. We do not, of course, consider a few words in another language to constitute a language shift. The algorithm comprises four major steps: (1) Split text into arbitrary segments. (2) Calculate characteristics of each segment. (3) Classify each. (4) Refine classifications and output final results.

4.1 Splitting the Text

Documents are not always punctuated into sentences or paragraphs. So, splitting is done in the naïve way of breaking the text into fixed-size segments. As language does not shift mid-word (except for certain prefixes), we break the text between words. If sentences are delineated and one ignores possible transitions mid-sentence, then the breaks should be between sentences.

The selection of segment size should depend on the language shift frequency. Nonetheless, each segment is classified using statistical properties, so it has to be long enough to have some statistical significance. But if it is too long, the language transitions will be less accurate, and if a segment contains two shifts, it will miss the inner one. Because the post-processing phase is computationally more expensive, and grows proportionally with segment length, we opt for relatively short initial segments.

4.2 Feature Extraction

The core of the algorithm is the initial classification of segments. Textual classification is usually reduced to vector classification, so there each segment is represented as a vector of features. Naturally, the selection of features is critical for successful classification, regardless of classification algorithm. Several other features were tried such as hierarchical clustering of segments and classification of the clusters (Choi, 2000) but did not yield significant improvement.

N-char distance – The first and most obvious feature is the classification of the segment using the methods described in Section 3. However, the segments are significantly smaller than the usual documents, so we expect lower accuracy than usual for language classification. The features are the cosine distance from each language model. This is rather natural, since we want to preserve the distances from each language model in order to combine it with other features later on. For each segment *f* and language *l*, we compute $Distance_1 = Dist(l, f)$, the cosine distance of their bigram distributions.

Neighboring segments language – We expect that languages in a document do not shift too frequently, since paragraphs tend to be monolingual and at least several sentences in a row will be in the same language to convey some idea. Therefore, if we are sure about one segment, there is a high chance that the next segment will be in the same language. One way to express such dependency is by post-processing the results to reduce noise. Another way is by combining the classification results of neighboring segments as features in the classification of the segment. Of course, not only neighboring segments can be considered, but all segments within some distance can help. Some parameter should be estimated to be the threshold for the distance between segments under which they will be considered neighbors. We denote by (negative or positive) $Neighbor(f,i)$ the i-th segment before/after f. If $i=0$, $Neighbor(f,i) = f$. For each segment f and language l, we compute $NDist_{l,f}(i) = Dist(l, Neighbor(f, i))$.

Whole document language - Another feature is the cosine distance of the whole document from each language model. This tends to smooth and reduce noise from classification, especially when the proportion of languages is uneven. For a monolingual document, the algorithm is expected to output the whole document as one correctly-classified segment.

4.3 Post-processing

We refine the segmentation procedure as follows: We look at the results of the splitting procedure and recognize all language shifts. For each shift, we try to find the place where the shift takes place (at word granularity). We unify the two segments and then try to split the segment at N different points. For every point, we look at the cosine distance of the text before the point from the class of the first segment, and at the distance of the text after the point to the language of the second segment. For example, suppose a segment $A_1 \ldots A_n$ was classified as Hebrew and segment $B_1 \ldots B_m$, which appeared immediately after was classified Aramaic. We try to split $A_1 \ldots A_n, B_1 \ldots B_m$ at any of N points, N=2, say. First, we try $F_1 = A_1 \ldots A_{(n+m)/3}$ and $F_2 = A_{(n+m)/3+1} \ldots B_m$ (supposing $(n+m)/3 < n$). We look at cosine distance of F_1 to Hebrew and of F_2 to Aramaic. Then, we look at $F_1 = A_1 \ldots A_{2(n+m)/3}$ and $F_2 = A_{2(n+m)/3+1} \ldots B_m$. We choose the split with the best product of the two cosine distances. The value of N is a tradeoff between accuracy and efficiency. When N is larger, we check more transition points, but for large segments it can be computationally expensive.

4.4 Noise Reduction

OCR-processed documents display a significant error rate and classification precision should be expected to drop. Relying only on n-char statistics, we propose probabilistic approaches to the reduction of noise. Several methods (Kukich, 1992) have been proposed for error correction using n-chars, using letter-transition probabilities. Here, we are not interested in error correction, but rather in adjusting the segmentation to handle noisy texts.

To account for noise, we introduce a $ sign, meaning “unknown” character, imagining a conservative OCR system that only outputs characters with high probability. There is also no guarantee that word boundaries will not be missed, so $ can occur instead of a space.

Ignoring unrecognized n-chars – We simply ignore n-chars containing $ in the similarity measures. We assume there are enough bigrams left in each segment to successfully identify its language.

Error correction – Given an unknown character, we could try correcting it using trigrams, looking for the most common trigram of the form a$b. This seems reasonable and enhances the statistical power of the n-char distribution, but does not scale well for high noise levels, since there is no solution for consecutive unknowns.

Averaging n-char probabilities – When encountering the $, we can use averaging to estimate the probability of the n-char containing it. For instance, the probability of the bigram $*x* will be the average probability of all bigrams starting with *x* in a certain language. This can of course scale to longer n-chars and integrates the noisy into the computation.

Top n-chars – When looking at noisy text, we can place more weight on corpus statistics, since they are error free. Therefore, we can look only at the N most common n-chars in the corpus for edit distance computing.

Higher n-char space – So far we used bigrams, which showed superior performance. But when the error rate rises, trigrams may show a higher success rate.

5. Experimental Results

We want to test the algorithm with well-defined parameters and evaluation factors. So, we created artificially mixed documents, containing segments from pairs of different languages (Hebrew/Aramaic, which is hard, Hebrew/Judeo-Arabic, where classification is easy and segmentation is the main challenge, or a mix of all three). The segments are produced using two parameters: The desired document length *d* and the average monolingual segment length *k*. Obviously, $k < d$. We iteratively take a random number in the range [*k*–20:*k*+20] and take a substring of that length from a corpus, rounded to whole words. We cycle through the languages until the text is of size *d*. The smaller *k*, the harder to segment.

5.2 Evaluation Measures

Obviously splitting will not be perfect and we cannot expect to precisely split a document. 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 to the algorithm output, using which we 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 segment's boundaries accurately, classifying each segment correctly or even split the document to the exact number of segments. For example, given a long document in Hebrew with a small segment in Aramaic, is it better to return that it actually is a long document in Hebrew with Aramaic segment but misidentify the segment's location or rather recognize the Aramaic segment perfectly but classify it as Judeo-Arabic. There are several measures for evaluating text segmentation (Lamprier et al., 2007).

Correct word percentage – The most intuitive measure is simply measuring the percentage of the 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 is pretty useless and we may prefer a result that identifies the Aramaic segments but errs on more words.

Segmentation error (SE) estimates the algorithm's sensitivity to language shifts. It is the difference between the correct number and that returned by the algorithm, divided by correct number. Obviously, SE is in the range [–1:1]. It will indeed resemble the problem previously described, since, if the entire document is classified as Hebrew, the SE score will be very low, as the actual number is much greater than 1.

5.3 Experiments

Neighboring segments – The first thing we tested is the way a segment's classification is affected by neighboring segments. We begin by checking if adding the distance of the closest segments enhances performance. Define $Score_{f,l} = Dist(l,f) + a\left(NDist_{l,f}(1) + NDist_{l,f}(-1)\right)$. For the test we set *a*=0.4.

From Figures 1 and 2, one can see that neighboring segments improve classification of short segments, while on shorter ones classification without the neighbors was superior. It is not surprising that when using neighbors the splitting procedure tends to split the text to longer segments, which has good effect only if segments actually are longer. We can also see from Figure 3 that the SE measure is now positive with *k*=100, which means the algorithm underestimates the number of segments even when each segment is 100 characters long. By further experiments, we can see that the *a* parameter is insignificant, and fix it at 0.3.

As expected, looking at neighboring segments can often improve results. The next question is if farther neighbors also do. Let: $Score_{f,l} = Dist(l,f) + \sum_{k=1}^{N}\left(\frac{a}{k}\right)\left(NDist_{l,f}(k1) + NDist_{l,f}(-k)\right)$. Parameter *N* stands for the longest distance of neighbors to consider in the score. Parameter *a* is set to 0.3.

We see that increasing *N* does not have a significant impact on algorithm performance, and on shorter segment lengths performance drops with *N*. We conclude that there is no advantage at looking at distant neighbors.

Post-processing – Another thing we test is the post-processing of the splitting results to refine the initial segment choice. We try to move the transition point from the original position to a more accurate position using the technique described above. We note is cannot affect the SE measure, since we only move the transition points without changing the classification. As shown in Figure 4, it does improve the performance for all values of *l*.

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

Not surprisingly, Figure 5 illustrates that the accuracy reduces as the error rate rises. However, it does not significantly drop even for a very high error rate, and obviously we cannot expect that the error reducing process will perform better then the algorithm performs on errorless text. Figure 6 illustrates the performance of each method. It looks like looking at most common n-chars does not help, nor trying to correct the unrecognized character. Ignoring the unrecognized character, using either bigrams or trigrams, or estimating the missing unrecognized bigram probability show the best and pretty similar results.

6. Conclusion

We have described methods for classifying texts, all using the same character set, into several languages. Furthermore, we considered segmented multilingual texts into monolingual components. In both cases, we made allowance for corrupted texts, such as that obtained by OCR from handwritten manuscripts. The results are encouraging and will be used in the Friedberg Genizah digitization project (www.genizah.org).

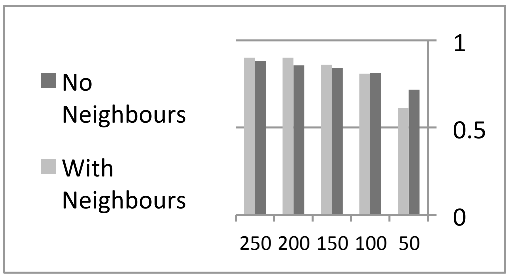

Figure 1: Correct word percentage considering neighbors and not, as a function of segment length *k* (document length was 1500).

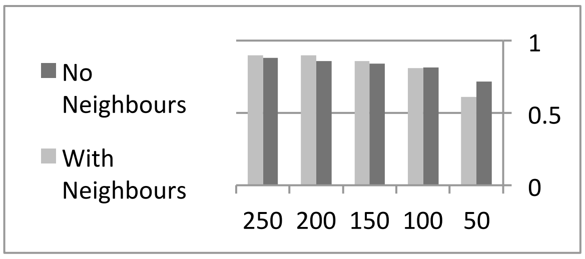

Figure 2: Segmentation error considering neighbors or not (k =1500).

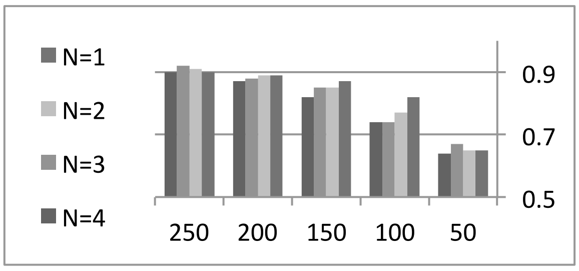

Figure 3: Correct word percentage for various resplitting values N as a function of *k*.

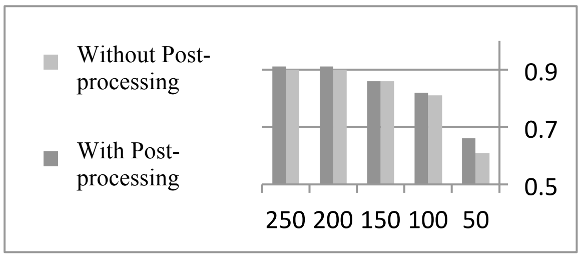

Figure 4: Correct word percentage with and without post-processing.

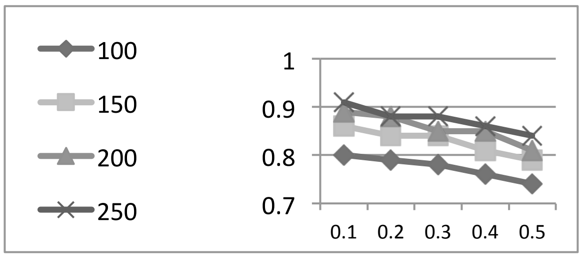

Figure 5: Word accuracy as a function of noise.

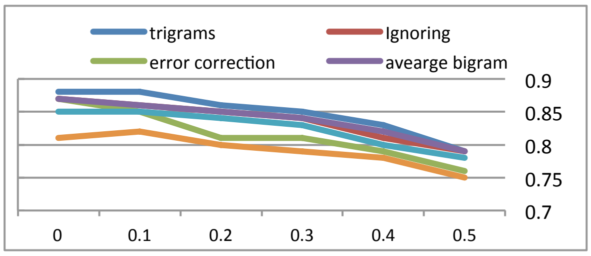

Figure 6: The performance of suggested correction methods for each error rate.

Acknowledgement

We thank the Friedberg Genizah Project for supplying data to work with and for their support.

References

Gökhan Bakır. 2007. *Predicting Structured Data*. MIT Press, Cambridge, MA.

Freddy Y. Y. Choi. 2000. Advances in domain independent linear text segmentation. *Proc. 1st North American Chapter of the Association for Computational Linguistics Conference,* pp. 26-33.

Juha Häkkinen and Jilei Tian. 2001. *N*-gram and decision tree based language identification for written words. *Proc. Automatic Speech Recognition and Understanding (ASRU '01)*, Italy, pp. 335-338.

Marti A. Hearst. 1993. TextTiling: A quantitative approach to discourse segmentation. Technical Report, Sequoia 93/24, Computer Science Division.

Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, and Frédéric Saubion. 2007. On evaluation methodologies for text segmentation algorithms. *Proc. 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI),* volume 2, pp. 19-26.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys* 24(4): 377-439.