Language Identification
L101 Project

Augustin Zidek (*az317*)

January 12, 2016

Abstract

Language identification is the task of identifying the language a given document is written in. It can be achieved with very good accuracy using sophisticated machine learning methods. This report describes various features and distance metrics that were used to perform this task while using the nearest prototype method. Although simple, very high classification accuracy and performance was achieved on the EuroGOV corpus of the `naacl2010-langid` dataset.

Contents

1 Introduction

There are billions of pages in the Internet written in hundreds of human languages. However, not all of them declare their language in the metadata. Since this information is valuable, for instance to deliver search engine results in the language specified by the user, it is necessary to be able to reliably determine language of web pages.

There are many approaches available ranging from deterministic methods based on letter frequencies, word frequencies, analysis of special letters unique to each language to machine learning approaches which use various document features (e.g. the histogram of used characters) and are trained on huge datasets where the languages are known.

Baldwin and Lui [1] showed in their paper that deterministic classification methods could work surprisingly well compared to sophisticated machine learning methods. Moreover, they compiled a huge dataset `naacl2010-langid` that will be used in this work and the performance of my classification algorithms will be compared to the performance of algorithms by Baldwin and Lui.

In this paper, character unigram and bigram histograms are used to identify language of documents. Although I originally proposed to use words as features, this had to be discarded, since there are many languages that don't have clear boundaries between words and I suspected this would to lead to poor results. While Baldwin and Lui use histograms trained on the datasets, histograms obtained from Wikipedia crawling were used in this paper. This lead to better language prototype histograms and more robust performance on unseen datasets.

The `R` programming language was used to do the analysis and performance of the various algorithms was evaluated using the Python evaluation script provided with the `naacl2010-langid` dataset.

2 Datasets

2.1 The `naacl2010-langid` dataset

The `naacl2010-langid` was created by Baldwin and Lui and accompanied their paper. It contains three corpora which are described in Table 1 below. Meta files are provided for each of the corpora containing rows with filename, encoding, document language and partition number (to enable reconstruction of 10-fold cross-validation which Baldwin and Lui used in their paper).

	Docs	**Langs**	**Source**	**Doc length [bytes]**
EuroGOV	1,500	10	EuroGOV document collection with long documents	17460.5 ± 39353.4
TCL	3,174	60	Manually sourced by the Thai Computational Linguistics Laboratory	2623.2 ± 3751.9
Wikipedia	4,963	67	Sourced from Wikipedia dump, sampled in bias-preserving manner	1480.8 ± 4063.9

Table 1: The characteristics of the `naacl2010-langid` dataset

While the EuroGOV and Wikipedia corpora had all documents encoded in UTF-8, this wasn't the case for the TCL which had documents in 12 various encodings. To make histogram construction easier, I preprocessed the TCL corpus and converted all the documents to use the UTF-8 encoding.

Moreover, the TCL corpus had the meta file with the document list in wrong order compared to the standard listing of the TCL corpus folder using R. This was addressed by correctly sorting the meta file for the TCL corpus.

I also found spurious files in the Wikipedia dataset – there are 37 files with their size smaller than 10 bytes and containing mostly non-letter character. Therefore these files would not be correctly classifiable even by humans and therefore I would suggest them to be removed. However, to make the results consistent with Baldwin and Lui, I kept the files in the corpus.

2.2 Unigram and bigram histograms

In order to have representative (prototype) histograms of unigrams (letters) and bigrams in various languages, I used the datasets provided by Denny Vrandečić [2]. These datasets were created on the Wikipedia dump cleaned using WikiExtractor. Two datasets were used – one with unigram (letter) histograms and one with bigram histograms for all the languages needed.

Using these histograms gave the advantage of having very representative language models, since the entire Wikipedia was used to provide data for these histograms.

Although punctuation and other non-letter characters were removed from the histograms, they still had very long tails. For instance, the unigram histogram for English had 7,455 characters with the sum of the frequencies of the top 50 characters being about 1000 times the sum of the frequencies of the remaining characters. However, the histograms were much more flat for languages like Chinese or Japanese which have a greater number of characters in their alphabets.

Also, the histograms were slightly messy since Wikipedia contains many pages which contain fragments of texts in foreign languages – e.g. all pages on foreign languages contain example sentences in that language. I addressed this by reducing the prototype histograms to contain only top n features and I used $n = 500$, since that gave the best results across the corpora.

3 Document representation

The documents were represented either as unigram or bigram histograms computed from the individual UTF-8 characters. Unigrams are single-letter tokens and bigrams are two-letter tokens in this context. E.g. the unigram representation of the document “Hello World!!! áá 42” would be

Char	Freq
á	2
d	1
e	1
h	1
l	3
o	2
r	1
w	1

Table 2: Unigram representation of the document “Hello World áá 42!!!”

Since everything was UTF-8 encoded, the regular expression `[^\\p{L}]` was used to find all non-letter characters and they were discarded. This proved to work well even for languages with non-Latin alphabets, such as Japanese.

However, if removal of non-letter characters led to an empty string, punctuation was also permitted. If even this led to an empty string, space characters and the vertical bar character (“|”) was also permitted. This was a workaround to make the system work even with documents containing no letters as observed in the Wikipedia corpus.

The bigram representation was similar, but bigrams were used. The same non-letter removal algorithm was used. Therefore, e.g. for the document “He helps” the following bigram histogram would be generated:

Char	Freq
eh	1
el	1
he	2
lp	1
ps	1

Table 3: Bigram representation of the document “He helps”

I also experimented with combined methods where both unigram and bigram histograms were used, but this only led to marginal improvements in the accuracy, therefore the results were not included in the report.

4 Distance measures

The 7 following distance measures were used and compared:

1. **Cosine similarity (cos):** The cosine of the angle between the two histogram vectors. Missing histogram values were interpreted as having frequency 0 in that dimension.
2. **Symmetric Kullback-Leibler divergence (kl):** A relative entropy measure defined as: $D(x \parallel y) = \frac{1}{2}\sum_i (x_i(\log_2 x_i - \log_2 y_i) + y_i(\log_2 y_i - \log_2 x_i))$
3. **Skew divergence (skew):** As defined by Baldwin and Lui: a variant of Kullback-Leibler divergence, whereby y is smoothed by linear interpolation with x using a smoothing factor α: $D_\alpha(x \parallel y) = \sum_i [x_i(\log_2 x_i - \log_2(\alpha x_i + (1-\alpha)y_i))]$. The value $\alpha = 0.99$ was used in all experiments as recommended by [1] and verified by my own experiments.
4. **Minkowski L1 distance (mink1):** Minkowski distance with $p = 1$ calculated as $D(x \parallel y) = \sum_i |x_i - y_i|$.
5. **Minkowski L2 distance (mink2):** Minkowski distance with $p = 2$ calculated as $D(x \parallel y) = \left(\sum_i |x_i - y_i|^2\right)^{1/2}$. While very intuitive, it proved to have surprisingly good performance.
6. **Out-of-place distance (oop):** As defined by Baldwin and Lui: a ranklist-based distance metric where the distance between x and y is calculated as: $D(x \parallel y) = \sum_t |R(x,t) - R(y,t)|$ where $R(d,t)$ is the rank of the term t in the document d. The terms are sorted in the descending order of the frequency of the terms. Missing terms were handled by assigning them frequency 0.
7. **Voting distance (vote):** Majority voting on the distance measures 1–5. OOP was omitted since its classification results were rather poor and it is more convenient to have odd number of voters. The five algorithms were run and then the resulting language was chosen according to the language which was chosen by the majority of the algorithms.

5 Methodology

The document classification works as follows:

1. Compute the histogram H_i for the document i.
2. Compare the histogram H_i to each of the histograms computed from Wikipedia W_j and assign the document i to the language of the W_j which is the closest to H_i, that is $i = \text{argmin}_j \; D(H_i \parallel W_j)$.

The results of the various algorithms were evaluated using the same methodology as by Baldwin and Lui using the Python evaluation script provided with the `naacl2010-langid` dataset.

A small modification was, however, done to the evaluation script as it crashed when a certain language that was supposed to be present in the dataset was not assigned to a single document.

The performance of the system enabled classification of about 50 documents per second. Considering this was done in the `R` programming language which is interpreted, it is decent performance. An easy way of increasing the performance would be to rewrite the system in a compiled language (e.g. C++ or Java) which could lead to even 100 times better performance [3].

6 Results

The results are presented and discussed separately for each dataset. In each case unigrams and bigram results are presented together in combination with all the distance measures described above.

The prototype histograms were limited to contain only the top 500 values and the impact of the number of values is investigated in a later section.

I evaluated the models using the same methodology as Baldwin and Lui and report macro-averaged precision (P_M), recall (R_M) and F-score (F_M). I also report $A = P_\mu / R_\mu / F_\mu$ where P_μ, R_μ and F_μ are micro-averaged precision, recall and F-score, respectively. While A indicates performance per document, the macro-averaged scores indicate performance per language.

6.1 EuroGOV

The results for the EuroGOV corpus are in Table 4 below.

Model	Histogram	P_M	R_M	F_M	A
cos	unigram	0.961	0.964	0.963	0.961
cos	bigram	0.984	0.982	0.983	0.981
kl	unigram	0.971	0.956	0.963	0.962
kl	bigram	0.978	0.976	0.977	0.974
skew	unigram	0.980	0.979	0.980	0.977
skew	bigram	0.974	0.970	0.972	0.968
mink1	unigram	0.972	0.969	0.971	0.966
mink1	bigram	0.981	0.976	0.979	0.978
mink2	unigram	0.961	0.964	0.963	0.961
mink2	bigram	0.984	0.982	0.983	0.981
oop	unigram	0.852	0.766	0.807	0.743
oop	bigram	0.888	0.821	0.853	0.798
vote	unigram	0.972	0.969	0.971	0.966
vote	bigram	**0.986**	**0.984**	**0.985**	**0.983**

Table 4: Results for the EuroGOV dataset using prototype histograms with the top 500 features

In this case, I obtained very good results which are in the best case on par with SVM used by Baldwin and Lui. The best score is achieved by the voting meta-algorithm. This is not very surprising, considering the fact that results of the individual classificators are very good as well and the Cosine and Minkowski L2 are almost on par with the SVM classificator giving the best scores for Baldwin and Lui. The voting algorithm performs so well because small errors caused by single algorithms are smoothed out.

The reason the performance is so good in this case is that the documents are quite long and therefore the nature of the language statistically matters. Moreover, the number of languages in this corpus is small (only 10 European languages are used).

We can also observe that with the exception of the Skew distance measure, all models were improved by using bigrams instead of unigrams. This was expected as the bigram histograms for different languages provide more diversity, especially in languages with short alphabets. This is partially due to different vocabulary but more importantly to the fact, that the number of unique bigrams is of the order of the number of unigrams squared.

6.2 TCL

The results for the TCL corpus are in Table 5 below.

Model	Histogram	P_M	R_M	F_M	A
cos	unigram	**0.888**	0.910	0.899	0.942
cos	bigram	0.876	0.928	0.901	0.939
kl	unigram	0.879	0.890	0.884	0.944
kl	bigram	0.875	0.901	0.888	0.746
skew	unigram	0.884	0.911	0.897	**0.948**
skew	bigram	0.806	0.854	0.829	0.618
mink1	unigram	0.850	0.885	0.867	0.758
mink1	bigram	0.821	0.804	0.812	0.625
mink2	unigram	**0.888**	0.910	0.899	0.942
mink2	bigram	0.867	0.928	0.896	0.939
oop	unigram	0.000	0.017	0.000	0.001
oop	bigram	0.829	0.860	0.844	0.891
vote	unigram	0.876	0.907	0.892	0.946
vote	bigram	0.880	**0.926**	**0.902**	0.939

Table 5: Results for the TCL dataset

In this corpus we see more diversity in the performance of the algorithms and also decrease in the performance of all of the algorithms. This is due to two facts:

1. The documents in this corpus are shorter on average, therefore the character histograms are not representative enough. Same applies for bigram histograms.
2. There are documents in 60 languages in this corpus. This is 6 times the number of languages in the EuroGOV corpus and that makes the classification algorithms much more sensitive to errors.

The results in this case are not as good as results obtained by Baldwin and Lui. Most likely it is due to the fact that they used byte as the histogram token and that effectively made their document length 1–4 times longer, since UTF-8 characters consist of 1 to 4 bytes [4].

We see that in this corpus it is no longer the case that bigrams outperform the unigrams. It is most likely due to the short document length, as explained above.

It is also worth noticing that the unigram OOP algorithm performed extremely badly in this case. This was caused most likely by the large number of characters (500) in the prototype histograms which introduced huge penalisation for each document and hence the subtle differences caused by individual languages were insignificant. We see the bigram OOP had much better performance since there much more bigrams are actually significant. This did not happen to Baldwin and Lui, since their prototype histograms were computed from these corpora, hence they contained less completely unrelated characters.

6.3 Wikipedia

The results for the Wikipedia corpus are in Table 6 below.

Model	Histogram	P_M	R_M	F_M	A
cos	unigram	0.550	0.620	0.583	0.712
cos	bigram	0.607	0.698	0.650	0.783
kl	unigram	0.485	0.485	0.485	0.194
kl	bigram	0.638	**0.734**	**0.683**	0.792
skew	unigram	0.406	0.267	0.323	0.119
skew	bigram	0.639	0.695	0.666	0.727
mink1	unigram	0.550	0.530	0.540	0.613
mink1	bigram	**0.661**	0.448	0.534	0.591
mink2	unigram	0.548	0.620	0.582	0.712
mink2	bigram	0.598	0.698	0.644	0.783
oop	unigram	0.045	0.016	0.024	0.004
oop	bigram	0.515	0.507	0.511	0.434
vote	unigram	0.542	0.627	0.581	0.703
vote	bigram	0.610	0.709	0.656	**0.797**

Table 6: Results for the Wikipedia dataset

The results for Wikipedia are the worst of the three corpora. There are two reasons which were already discussed in the context of TCL: even shorter documents and even more languages (67 in this case). However, compared to Baldwin and Lui, their results on codepoint tokens are either comparable or worse, and hence the fact I used more representative prototype histograms showed.

Also, there is a significant improvement compared to results obtained by Baldwin and Lui. While their per document (micro-averaged) best score is significantly higher than mine (0.869 compared to 0.797), their per language (macro-averaged) scores are lower.

This could be explained by the fact they achieved their best results using bytes as tokens, while I used characters. When using bytes, the consistency across documents of a single language could decrease due to the nature of character encoding schemes.

While certain trends were observable in the previous two corpora, the results are much more diverse in this case. However, we see again trend in better bigram performance to unigram performance. Also, the same problem as in TCL in case of unigram OOP happened.

6.4 Impact of prototype length on the performance

As mentioned above, all the reported results were conducted with prototype histograms of length limited to 500, i.e. the 500 most frequent tokens were kept in each. In this section the impact of the size of the prototype histograms will be investigated. The trend was the same for all the datasets, therefore only results for the EuroGOV dataset are reported. Only results for Cosine, Minkowski L1, Skew and OOP are reported, as the results for Minkowski L2 are similar to Minkowski L1, the results of KL are similar to Skew.

In this case only the classification accuracy, i.e. the percentage of correctly classified documents is reported.

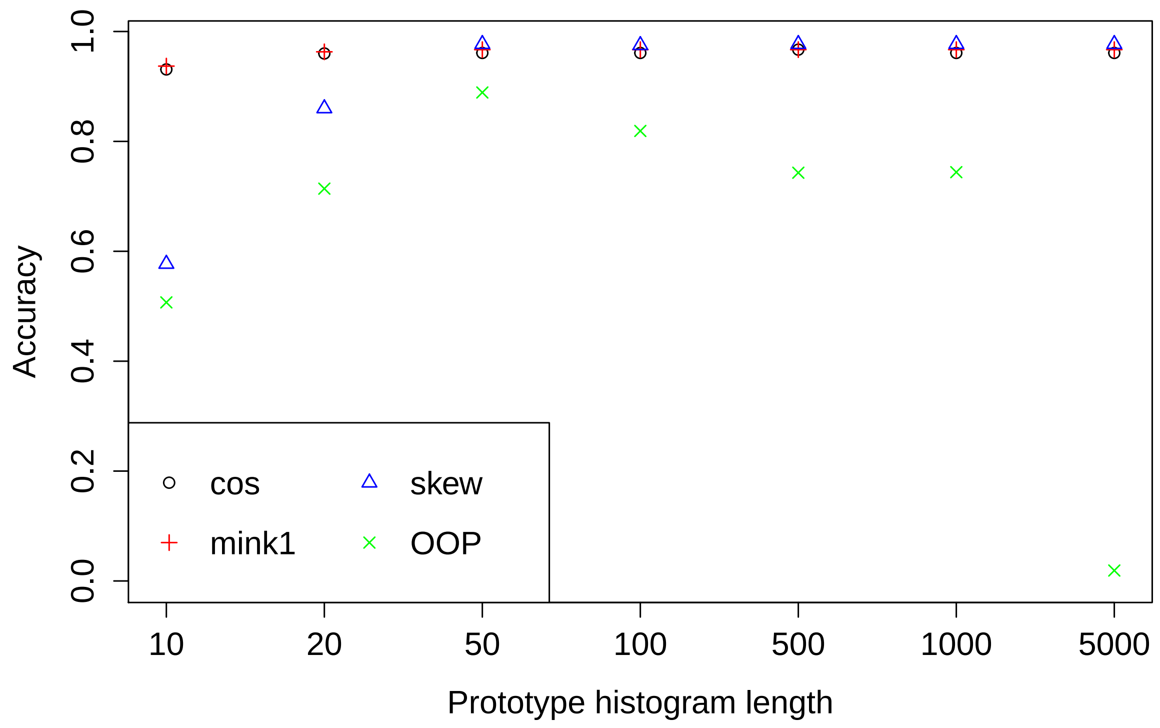

Figure 1: The classification accuracy depending on the prototype histogram length

We see that Cosine and Minkowski L1 measures are not very sensitive to the length of the prototype histograms. But even these two measures achieve slightly lower performance when the prototype histogram length is below 50.

The Skew measure needs certain prototype histogram length and then reaches plateau, achieving comparable or slightly better accuracy to Cosine and Minkowski L1 measures. The OOP measure is the most problematic one, as it reaches its maximum accuracy at around 50 and then its accuracy decreases again.

7 Conclusion

I evaluated 7 distance measures used on unigram and bigram histograms in order to detect document language. I achieved better classification results than Baldwin and Lui for the EuroGOV dataset and my solution would be more robust on unseen data, as it was trained solely on this dataset.

I also confirmed their results and showed that classifying shorter documents into a larger number of languages is much more challenging as shown by the classification results of the TCL and Wikipedia corpora. I showed 4 other metrics which Baldwin and Lui didn't use and the Minkowski L1 and L2 distance measures proved to perform very well while being conceptually very simple and intuitive.

I used prototype histograms obtained by parsing Wikipedia. These histograms could provide very valuable, especially if more carefully cleaned to contain only characters (or bigrams) that occur in the language they represent. The prototype histograms, since being created using

Wikipedia, contained a lot of spurious tokens, as Wikipedia contains a lot of articles containing fragments of foreign words or even sentences.

Moreover, I pointed out certain problems with the provided dataset that should be addressed: buggy evaluation script and a lot of documents which don't contain any letter characters whatsoever.

The voting meta-algorithm proved to be an easy yet powerful extension that might be used as one of the methods to increase the accuracy. It would be interesting to include more classifiers in the voting algorithm, such as classifiers working on byte features or more diverse, such as SVM or kNN.

References

[1] T. Baldwin and M. Lui, "Language identification: The long and the short of the matter," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 229–237, Association for Computational Linguistics, 2010.

[2] D. Vrandecic, "Letter frequency." http://simia.net/letters/, 2016. [Online; accessed 2016-01-10].

[3] S. B. Aruoba and J. Fernández-Villaverde, "A comparison of programming languages in economics," tech. rep., National Bureau of Economic Research, 2014.

[4] J. D. Allen *et al.*, *The Unicode Standard.* Addison-Wesley, 2007.

A Code used

```
# Clean up from the previous run
rm(list=ls())
setwd("c:/Users/augustin/Cambridge/Machine_Learning_for_Language_Processing/
    Project/")
wd <- getwd()

library(readr)      # I/O library
library(snow)       # Parallel computing
library(foreach)    # For each loop
library(doSNOW)     # Foreach Parallel Adaptor for snow
library(stringr)    # String processing library
library(tictoc)     # Code timing
library(beepr)      # beep() function
library(seewave)    # For computing Kullback-Leibler divergence
cluster <- makeCluster(4)
clusterExport(cluster, c("get_closest_language", "merge_histograms", "norm_vec",
                         "clean_string"))
clusterExport(cluster, c("kl.dist"))
registerDoSNOW(cluster)

# Read the entire dataset and save it as R datatypes for faster loading
read_and_preprocess_dataset <- function() {
    EuroGOV_URL   <- paste(wd, "/naacl2010-langid/EuroGov/", sep="")
    EuroGOV_files <- paste(EuroGOV_URL, list.files(EuroGOV_URL), sep="")
    EuroGOV_lst   <- parLapply(cluster, EuroGOV_files, read_file)
    EuroGOV       <- data.frame(cbind(EuroGOV_lst))
    EuroGOV_meta  <- read.table("naacl2010-langid/EuroGOV.meta", sep="\t")
    EuroGOV       <- cbind(EuroGOV, EuroGOV_meta)
    colnames(EuroGOV) <- c("string", "filename", "encoding", "lang", "part")
    EuroGOV$char_hist <- parLapply(cluster, EuroGOV$string, character_histogram)
    EuroGOV$bigr_hist <- parLapply(cluster, EuroGOV$string, bigram_histogram)
    save(EuroGOV, file="naacl2010-langid/EuroGOV.Rda")

    TCL_URL       <- paste(wd, "/naacl2010-langid/TCL/", sep="")
    TCL_files     <- paste(TCL_URL, list.files(TCL_URL), sep="")
    TCL_lst       <- parLapply(cluster, TCL_files, read_file)
    TCL           <- data.table(cbind(TCL_lst))
    TCL_meta      <- read.table("naacl2010-langid/TCL.meta", sep="\t")
    TCL           <- cbind(TCL, TCL_meta)
    colnames(TCL) <- c("string", "filename", "encoding", "lang", "part")
    # TCL doesn't have all strings in UTF-8, convert
    for (i in 1:nrow(TCL)) {
        TCL$string[i] <- iconv(as.character(TCL$string[i]),
                               from=as.character(TCL$encoding[i]), to="UTF-8")
    }
    TCL$char_hist <- parLapply(cluster, TCL$string, character_histogram)
    TCL$bigr_hist <- parLapply(cluster, TCL$string, bigram_histogram)
    save(TCL, file="naacl2010-langid/TCL.Rda")

    Wiki_URL      <- paste(wd, "/naacl2010-langid/Wikipedia/", sep="")
    Wiki_files    <- paste(Wiki_URL, list.files(Wiki_URL), sep="")
    Wiki_lst      <- parLapply(cluster, Wiki_files, read_file)
    Wikipedia     <- data.table(cbind(Wiki_lst))
    Wikipedia_meta<- read.table("naacl2010-langid/Wikipedia.meta", sep="\t")
    Wikipedia     <- cbind(Wikipedia, Wikipedia_meta)
    colnames(Wikipedia) <- c("string", "filename", "encoding", "lang", "part")
```

```r
    Wikipedia$char_hist <- parLapply(cluster, Wikipedia$string,
        character_histogram)
    Wikipedia$bigr_hist <- parLapply(cluster, Wikipedia$string, bigram_histogram)
    save(Wikipedia, file="naacl2010-langid/Wikipedia.Rda")
}


# Read in all the data from the preprocessed files
load("naacl2010-langid/EuroGOV.Rda")
load("naacl2010-langid/TCL.Rda")
load("naacl2010-langid/Wikipedia.Rda")

# Read in the histograms for the EuroGOV documents
load_small_histogram_dataset <- function() {
    histograms <- read.csv("Letter_histograms-EuroGOV.csv", encoding="UTF-8",
                           dec=".", header=TRUE, stringsAsFactors=FALSE)
    histograms_dfs <- list(length=ncol(histograms) - 1)
    for (i in 2:ncol(histograms)) {
        histograms_dfs[[i - 1]] <- data.table(histograms[, 1], histograms[, i])
        names(histograms_dfs[[i - 1]]) <- c("char", "freq")
    }
    names(histograms_dfs) <- names(histograms)[2:ncol(histograms)]
    return(histograms_dfs)
}


load_wikipedia_histograms <- function(hist_folder, limit, is_freq_limit) {
    # Load all the histograms
    folder     <- paste(wd, hist_folder, sep="")
    file_list  <- paste(folder, list.files(folder, pattern="*.txt"), sep="")
    histograms <- parLapply(cluster, file_list, read.csv, encoding="UTF-8",
                            sep="␣", stringsAsFactors=FALSE, header=FALSE)
    histograms <- parLapply(cluster, histograms, data.frame)
    # Load the list of language names
    languages  <- sub(pattern=".txt", replacement="", fixed=TRUE,
                      x=sub(pattern=".*/", x=file_list, replacement=""))
    # Set the column names to the language codes
    names(histograms) <- languages
    # Remove non-letters
    histograms <- parLapply(cluster, histograms, remove_non_letters)

    if (is_freq_limit) {
        # Filter out letters above the given frequency
        histograms <- parLapply(cluster, histograms, remove_infrequent, n=limit)
    }
    else {
        # Keep only the top <limit> letters
        histograms <- parLapply(cluster, histograms, keep_top_n, n=limit)
    }

    for(i in 1:length(histograms)) {
        names(histograms[[i]]) <- c("char", "freq")
    }
    return(histograms)
}

remove_infrequent <- function(histogram, n) {
    return(histogram[histogram[, 2] > n, ])
```

```
}

keep_top_n <- function(histogram, n) {
    return(histogram[1:min(nrow(histogram), n), ])
}

remove_non_letters <- function(histogram) {
    return(histogram[grepl(pattern="[\\p{L}]",
                          as.character(histogram[, 1]), perl=TRUE), ])
}

find_misaligned_files_in_dataset <- function(dataset, dataset_files) {
    for (i in 1 : nrow(dataset)) {
        if (!grepl(as.character(dataset$filename[i]),
                   dataset_files[i], fixed=TRUE)) {
            print(paste(i, dataset_files[i], dataset$filename[i]))
        }
    }
}

find_files_with_empty_histograms <- function(dataset) {
    for (i in 1 : nrow(dataset)) {
        if (str_replace_all(dataset$string[[i]],
                            "[^\\p{L}\\p{P}\\p{Z}|]", "") == "") {
            print(paste("Problematic␣file:", i))
        }
    }
}

clean_string <- function(str) {
    require(stringr)
    str_cleaned <- str_replace_all(str, "[^\\p{L}]", "")
    # If there are not any letter characters in the document, try punctuation
    if(str_cleaned == "") {
        str_cleaned <- str_replace_all(str, "[^\\p{L}\\p{P}]", "")
        # If still, try also spaces and "|" character
        if (str_cleaned == "") {
            str_cleaned <- str_replace_all(str, "[^\\p{L}\\p{P}\\p{Z}|]", "")
        }
    }
    return(str_cleaned)
}

# Single character (unigram) histogram,
character_histogram <- function(str) {
    str_cleaned <- clean_string(str)
    # Return letter histogram, lowercase only
    histogram <- data.frame((table(strsplit(tolower(str_cleaned), ""))))
    names(histogram) <- c("char", "freq")
    return (histogram)
}

# Bigram histogram
bigram_histogram <- function(str) {
    str_cleaned <- clean_string(str)
    unigrams <- strsplit(tolower(str_cleaned), "")[[1]]
    bigrams <- paste(unigrams[1 : length(unigrams) - 1],
                     unigrams[2 : length(unigrams)], sep="")
```

```r
    # Return letter histogram, lowercase only
    histogram <- data.frame(table(bigrams))
    names(histogram) <- c("char", "freq")
    return (histogram)
}

norm_vec <- function(x) {
    return(sqrt(sum(x^2)))
}

merge_histograms <- function(hist1, hist2) {
    merged <- merge(hist1, hist2, by="char", all=TRUE)
    merged[is.na(merged)] <- 0
    return(merged)
}

# Compute cosine distance of two histograms
distance_cos <- function(x, y) {
    merged <- merge_histograms(x, y)
    sum <- sum(merged$freq.x * merged$freq.y)
    cos_sim <- sum / (norm_vec(x$freq) * norm_vec(y$freq))
    return(1 - cos_sim)
}

# Normalise by L1, then compute average coeficient distance
distance_minkowski1 <- function(x, y) {
    merged <- merge_histograms(x, y)
    x_len <- norm_vec(merged$freq.x)
    y_len <- norm_vec(merged$freq.y)
    merged$freq.x <- merged$freq.x / x_len
    merged$freq.y <- merged$freq.y / y_len
    dist_sum <- abs(merged$freq.x - merged$freq.y)
    return(sum(dist_sum))
}

# Normalise by L2, then compute average coeficient distance
distance_minkowski2 <- function(x, y) {
    merged <- merge_histograms(x, y)
    x_len <- norm_vec(merged$freq.x)
    y_len <- norm_vec(merged$freq.y)
    merged$freq.x <- merged$freq.x / x_len
    merged$freq.y <- merged$freq.y / y_len
    dist_sum <- abs(merged$freq.x - merged$freq.y)^2
    return(sqrt(sum(dist_sum)))
}

distance_kl <- function(x, y) {
    merged <- merge_histograms(x, y)
    return(kl.dist(merged$freq.x, merged$freq.y)$D)
}

distance_skew <- function(x, y) {
    merged <- merge_histograms(x, y)
    return(kl.dist(merged$freq.x, 0.99*merged$freq.y + 0.01*merged$freq.x)$D1)
}

# A ranklist-based distance metric
distance_out_of_place <- function(x, y) {
    x_sorted_by_freq <- x[with(x, order(-x$freq)), ]
    y_sorted_by_freq <- y[with(y, order(-y$freq)), ]
    x_sorted_by_freq$rank <- row(x_sorted_by_freq)[, 1]
    y_sorted_by_freq$rank <- row(y_sorted_by_freq)[, 1]
    merged <- merge_histograms(x_sorted_by_freq, y_sorted_by_freq)
    oop <- sum(abs(merged$rank.x - merged$rank.y))
    return(oop)
}

# Compute the closest language using the provided histograms
get_closest_language <- function(hist, dist_function, histograms) {
    min_distance = dist_function(hist, histograms[[1]])
    min_dist_lang = names(histograms)[1]
    for (i in 2:length(histograms)) {
        distance <- dist_function(hist, histograms[[i]])
        if (distance < min_distance) {
            min_distance <- distance
            min_dist_lang <- names(histograms)[i]
        }
    }
    return(min_dist_lang)
}

# Compute the closest language using the provided histograms
get_closest_language2 <- function(hist_tuple, dfun1, dfun2, alpha,
                                  histograms1, histograms2) {
    hist1 <- hist_tuple[[1]]
    hist2 <- hist_tuple[[2]]
    min_distance = alpha * dfun1(hist1, histograms1[[1]]) +
                   (1 - alpha) * dfun2(hist2, histograms2[[1]])
    min_dist_lang = names(histograms1)[1]
    for (i in 2:length(histograms1)) {
        distance <- alpha * dfun1(hist1, histograms1[[i]]) +
                    (1 - alpha) * dfun2(hist2, histograms2[[i]])
        if (distance < min_distance) {
            min_distance <- distance
            min_dist_lang <- names(histograms1)[i]
        }
    }
    return(min_dist_lang)
}


save_prediction_file <- function(corpus, predictions, filename) {
    prediction_table <- data.frame(corpus$filename, unlist(predictions))
    write.table(prediction_table, file=filename, sep="\t",
                row.names=FALSE, col.names=FALSE, quote=FALSE)
}

count_prediction_score <- function(corpus, predictions) {
    corpus_len <- nrow(corpus)
    no_of_matches <- length(predictions[predictions == corpus$lang])
    print(paste("Number␣of␣matches␣is", no_of_matches,
                "Success␣rate", no_of_matches/corpus_len))
}


evaluate <- function(corpus, hist_index, distance_fun, histograms, filename) {
    predictions <- vector(length=nrow(corpus))

    print("Computing␣predictions")
    tic()
    predictions <- parLapply(cluster, corpus[[hist_index]], get_closest_language,
                             dist_function=distance_fun, histograms=histograms)
    toc()

    count_prediction_score(corpus, predictions)
    save_prediction_file(corpus, predictions, filename)
    # Done, beep!
    beep(2)
}

eval_vote <- function(corpus, result_files, filename) {
    predictions <- lapply(result_files, read.table, sep="\t")
    predictions <- data.frame(predictions)
    predictions <- predictions[, c(FALSE, TRUE)]

    final_predictions <- vector(length=nrow(predictions))
    for(i in 1:nrow(predictions)) {
        final_predictions[i] <- names(which.max(table(unlist(predictions[i, ]))))
    }
    print("Predictions␣computed")
    count_prediction_score(corpus, final_predictions)
    save_prediction_file(corpus, final_predictions, filename)
    beep(2)
}

# Unigrams
histograms <- load_small_histogram_dataset()
histograms <- load_wikipedia_histograms("/unigrams_EuroGOV/", 500, FALSE)
evaluate(EuroGOV, 6, distance_cos, histograms, "EuroGOV-Char_hist-Cosine-
    top500.dat")
evaluate(EuroGOV, 6, distance_minkowski1, histograms, "EuroGOV-Char_hist-
    minkowski1-top500.dat")
evaluate(EuroGOV, 6, distance_minkowski2, histograms, "EuroGOV-Char_hist-
    minkowski2-top500.dat")
evaluate(EuroGOV, 6, distance_out_of_place, histograms, "EuroGOV-Char_hist-OOP-
    top500.dat")
evaluate(EuroGOV, 6, distance_kl, histograms, "EuroGOV-Char_hist-kl-top500.dat")
evaluate(EuroGOV, 6, distance_skew, histograms, "EuroGOV-Char_hist-skew-
    top500.dat")
EuroGOV_res <- list("Results/EuroGOV-Unigr_hist-cos-top500.dat",
                    "Results/EuroGOV-Unigr_hist-mink1-top500.dat",
                    "Results/EuroGOV-Unigr_hist-mink2-top500.dat",
                    "Results/EuroGOV-Unigr_hist-skew-top500.dat",
                    "Results/EuroGOV-Unigr_hist-kl-top500.dat")
eval_vote(EuroGOV, EuroGOV_res, "EuroGOV-Unigr_hist-ALL.dat")

histograms <- load_wikipedia_histograms("/unigrams_TCL/", 500, FALSE)
evaluate(TCL, 6, distance_cos, histograms, "TCL-Unigr_hist-Cos-top500.dat")
evaluate(TCL, 6, distance_minkowski1, histograms, "TCL-Unigr_hist-mink1-
    top500.dat")
evaluate(TCL, 6, distance_minkowski2, histograms, "TCL-Unigr_hist-mink2-
    top500.dat")
evaluate(TCL, 6, distance_out_of_place, histograms, "TCL-Unigr_hist-OOP-
    top500.dat")
```

```
evaluate(TCL, 6, distance_kl, histograms, "TCL-Unigr_hist-kl-top500.dat")
evaluate(TCL, 6, distance_skew, histograms, "TCL-Unigr_hist-skew-top500.dat")
TCL_res <- list("Results/TCL-Unigr_hist-cos-top500.dat",
                "Results/TCL-Unigr_hist-mink1-top500.dat",
                "Results/TCL-Unigr_hist-mink2-top500.dat",
                "Results/TCL-Unigr_hist-skew-top500.dat",
                "Results/TCL-Unigr_hist-kl-top500.dat")
eval_vote(TCL, TCL_res, "TCL-Unigr_hist-ALL.dat")

histograms <- load_wikipedia_histograms("/unigrams_Wiki/", 500, FALSE)
evaluate(Wikipedia, 6, distance_cos, histograms, "Wiki-Unigr_hist-Cos-top500.dat"
    )
evaluate(Wikipedia, 6, distance_minkowski1, histograms, "Wiki-Unigr_hist-mink1-
    top500.dat")
evaluate(Wikipedia, 6, distance_minkowski2, histograms, "Wiki-Unigr_hist-mink2-
    top500.dat")
evaluate(Wikipedia, 6, distance_out_of_place, histograms, "Wiki-Unigr_hist-OOP-
    top500.dat")
evaluate(Wikipedia, 6, distance_kl, histograms, "Wiki-Unigr_hist-kl-top500.dat")
evaluate(Wikipedia, 6, distance_skew, histograms, "Wiki-Unigr_hist-skew-
    top500.dat")
Wiki_res <- list("Results/Wiki-Unigr_hist-cos-top500.dat",
                 "Results/Wiki-Unigr_hist-mink1-top500.dat",
                 "Results/Wiki-Unigr_hist-mink2-top500.dat",
                 "Results/Wiki-Unigr_hist-skew-top500.dat",
                 "Results/Wiki-Unigr_hist-kl-top500.dat")
eval_vote(Wikipedia, Wiki_res, "Wiki-Unigr_hist-ALL.dat")


# Bigrams
histograms <- load_wikipedia_histograms("/bigrams_EuroGOV/", 500, FALSE)
evaluate(EuroGOV, 7, distance_cos, histograms, "EuroGOV-Bigr_hist-cos-top500.dat"
    )
evaluate(EuroGOV, 7, distance_minkowski1, histograms, "EuroGOV-Bigr_hist-
    minkowski1.dat")
evaluate(EuroGOV, 7, distance_minkowski2, histograms, "EuroGOV-Bigr_hist-
    minkowski2-top500.dat")
evaluate(EuroGOV, 7, distance_out_of_place, histograms, "EuroGOV-Bigr_hist-OOP-
    top500.dat")
evaluate(EuroGOV, 7, distance_skew, histograms, "EuroGOV-Bigr_hist-skew.dat")
evaluate(EuroGOV, 7, distance_kl, histograms, "EuroGOV-Bigr_hist-kl-top500.dat")
EuroGOV_res2 <- list("Results/EuroGOV-Bigr_hist-cos-top500.dat",
                     "Results/EuroGOV-Bigr_hist-mink1-top500.dat",
                     "Results/EuroGOV-Bigr_hist-mink2-top500.dat",
                     "Results/EuroGOV-Bigr_hist-skew-top500.dat",
                     "Results/EuroGOV-Bigr_hist-kl-top500.dat")
eval_vote(EuroGOV, EuroGOV_res2, "EuroGOV-Bigr_hist-ALL.dat")

histograms <- load_wikipedia_histograms("/bigrams_TCL/", 500, FALSE)
evaluate(TCL, 7, distance_cos, histograms, "TCL-Bigr_hist-cos-top500.dat")
evaluate(TCL, 7, distance_minkowski1, histograms, "TCL-Bigr_hist-mink1-top500.dat
    ")
evaluate(TCL, 7, distance_minkowski2, histograms, "TCL-Bigr_hist-mink2-top500.dat
    ")
evaluate(TCL, 7, distance_out_of_place, histograms, "TCL-Bigr_hist-OOP-top500.dat
    ")
evaluate(TCL, 7, distance_skew, histograms, "TCL-Bigr_hist-skew-top500.dat")
evaluate(TCL, 7, distance_kl, histograms, "TCL-Bigr_hist-kl-top500.dat")
```

```r
TCL_res2 <- list("Results/TCL-Bigr_hist-cos-top500.dat",
                 "Results/TCL-Bigr_hist-mink1-top500.dat",
                 "Results/TCL-Bigr_hist-mink2-top500.dat",
                 "Results/TCL-Bigr_hist-skew-top500.dat",
                 "Results/TCL-Bigr_hist-kl-top500.dat")
eval_vote(TCL, TCL_res2, "TCL-Bigr_hist-ALL.dat")

histograms <- load_wikipedia_histograms("/bigrams_Wiki/", 500, FALSE)
evaluate(Wikipedia, 7, distance_cos, histograms, "Wiki-Bigr_hist-cos-top500.dat")
evaluate(Wikipedia, 7, distance_minkowski1, histograms, "Wiki-Bigr_hist-mink1-
    top500.dat")
evaluate(Wikipedia, 7, distance_skew, histograms, "Wiki-Bigr_hist-skew-top500.dat
    ")
evaluate(Wikipedia, 7, distance_kl, histograms, "Wiki-Bigr_hist-kl-top500.dat")
eval_all_and_vote(EuroGOV, 7, histograms, "EuroGOV-Bigr_hist-ALL.dat")
Wiki_res2 <- list("Results/Wiki-Bigr_hist-cos-top500.dat",
                  "Results/Wiki-Bigr_hist-mink1-top500.dat",
                  "Results/Wiki-Bigr_hist-mink2-top500.dat",
                  "Results/Wiki-Bigr_hist-skew-top500.dat",
                  "Results/Wiki-Bigr_hist-kl-top500.dat")
eval_vote(Wikipedia, Wiki_res2, "Wiki-Bigr_hist-ALL.dat")


evaluate(Wikipedia, 7, distance_minkowski2, histograms, "Wiki-Bigr_hist-mink2-
    top500.dat")
evaluate(Wikipedia, 7, distance_out_of_place, histograms, "Wiki-Bigr_hist-OOP-
    top500.dat")

h10   <- load_wikipedia_histograms("/unigrams_EuroGOV/", 10, FALSE)
h20   <- load_wikipedia_histograms("/unigrams_EuroGOV/", 20, FALSE)
h50   <- load_wikipedia_histograms("/unigrams_EuroGOV/", 50, FALSE)
h100  <- load_wikipedia_histograms("/unigrams_EuroGOV/", 100, FALSE)
h500  <- load_wikipedia_histograms("/unigrams_EuroGOV/", 500, FALSE)
h1000 <- load_wikipedia_histograms("/unigrams_EuroGOV/", 1000, FALSE)
h5000 <- load_wikipedia_histograms("/unigrams_EuroGOV/", 5000, FALSE)
# Examine what impact has the histogram length on the performance
for (i in list(h10, h20, h50, h100, h500, h1000, h5000)) {
    print(paste("New␣histogram␣length"))
    evaluate(EuroGOV, 6, distance_cos, i, "res")
    evaluate(EuroGOV, 6, distance_minkowski1, i, "res")
    evaluate(EuroGOV, 6, distance_skew, i, "res")
    evaluate(EuroGOV, 6, distance_out_of_place, i, "res")
}

results <- read.csv("results.csv", row.names=1, header=TRUE, encoding="UTF-8",
    sep="␣")

plot(unlist(results[1, ]), xlab="Prototype␣histogram␣length", ylab="Accuracy",
     xaxt="n", ylim=c(0, 0.98))
points(unlist(results[2, ]), pch=3, col="red")
points(unlist(results[3, ]), pch=2, col="blue")
points(unlist(results[4, ]), pch=4, col="green")
axis(1, at=1:7, labels=c("10", "20", "50", "100", "500", "1000", "5000"))
legend("bottomleft", legend = c("cos", "mink1", "skew", "OOP"),
       col = c("black", "red", "blue", "green"), pch = c(1, 3, 2, 4), ncol=2)

plot(unlist(results[1, ]), xlab="Prototype␣histogram␣length", ylab="Accuracy",
     xaxt="n", ylim=c(0.85, 0.98))
```

```r
points(unlist(results[2, ]), pch=3, col="red")
points(unlist(results[3, ]), pch=2, col="blue")
axis(1, at=1:7, labels=c("10", "20", "50", "100", "500", "1000", "5000"))
legend("bottomright", legend = c("cos", "mink1", "skew"),
       col = c("black", "red", "blue"), pch = c(1, 3, 2))

stopCluster(cluster)
```