

UNIVERSITY of CALIFORNIA
SANTA CRUZ

LANGUAGE IDENTIFICATION ON THE WORLD WIDE WEB

A project report submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Katia Hayati

June 2004

The project report of Katia Hayati is approved:

Professor Raymie Stata

Professor David Helmbold

Abstract

Language Identification on the World Wide Web

by

Katia Hayati

We report on our experiments with adapting a classical language identification method for use on a Web corpus. We show how to refine its feature selection stage and its similarity measure by using the Fisher discriminant function and the cosine similarity metric respectively, and report on the improvements to the performance of the classifier. Lastly, we describe how to use Web-specific information, namely inlinks, to improve the performance of the classifier on very short Web documents.

Contents

Abstract	ii
1 Introduction	1
2 Data and methodology	2
3 Cavnar and Trenkle's algorithm	3
3.1 The method	3
3.3 Implementation and results	5
3.4 Discussion	5
4 Using the Fisher discriminant function	5
4.1 The Fisher discriminant function	6
4.2 The training phase	7
4.3 Classifying new documents	7
4.4 Algorithm formulation	8
4.6 Experimental results	9
5 Using Web information	9
5.1 Top-level domains	9
5.3 Outlinks	11
5.4 Inlinks	11
5.6 Experimental results	12
6 Related work	13
7 Conclusion	15
References	17

1 Introduction

Automatically identifying the language of a Web page is an important problem in information retrieval. We initially approached this problem wanting to implement a simple and efficient language recognizer for Web documents. One method which is popular for its performance and simplicity is Cavnar and Trenkle's n -gram based algorithm [1]. However, when we implemented this algorithm and tested it on Web documents, we found the results to be disappointing. The accuracy of the classifier on our Web corpus was significantly lower than the accuracy the authors report for Usenet data. We conjectured that the feature selection stage of the algorithm should be refined, and (consequently) its similarity function modified. We also noticed that the algorithm performed especially badly on short documents. When we improved the algorithm as planned, we still found that our classifier, while more accurate than the original, often failed on very short documents. This fact led to experiments with incorporating extra, Web-specific information into our classifier.

In this project, we report on our experiments and describe the various classifiers we implemented. We show how using the Fisher discriminant function [2] to select features reduces the number of errors by a factor of about two. Further, we show that using information about the pages which point to a page to be classified doubles the accuracy of the classifier on very short documents (less than 25 characters).

The rest of this paper is organized as follows. First, we describe our data and experimental setting. Then we describe our experiments with Cavnar and Trenkle's algorithm [1]. In Section 4 we explain how to use the Fisher discriminant function to improve the performance of the base classifier. Then in Section 5 we describe experiments that exploit information about top-level

domains, outlinks and inlinks. We show that using inlink information doubles the performance of the classifier on short Web pages. Section 6 describes related work, and Section 7 concludes.

2 Data and methodology

We obtained a sample of 1359 webpages spanning eleven languages (Danish, German, English, Spanish, Finnish, French, Italian, Dutch, Norwegian, Portuguese, Swedish) from the Internet Archive¹. We manually classified them according to language. We restricted ourselves to pages in the Windows-1252 character set [12] (a superset of the ISO-8859-1 character set, also called Latin-1). This is because unlike other character sets such as Euc-JP (a Japanese character set), Latin-1 does not determine a language. By this we mean that if a page uses the portion of Euc-JP which is not ASCII it is certainly written in Japanese. No such conclusion can be drawn for a page that uses the Latin-1 character set. There are also very good solutions [4, 7] to the problem of automatically identifying a character set, so it seems reasonable to focus on the Windows-1252 character set.

From these pages we removed all HTML markup (as well as scripts, etc.) and non-alphabetic characters. We also normalized spaces so there would be exactly one space between words. We divided the documents into training and testing sets. The performance of the algorithms was recorded on the testing sets.

Because the classification was manual, a certain bias against very short pages was introduced. This is because we needed at least a few words to be able to correctly assign a language to a Web page. In order to test the performance of our methods on very short pages (a few words at most), we had to go back and specifically look for short pages to introduce in our corpus.

¹<http://www.archive.org>

We formed a second corpus of pages under 50 characters long (the length of the page is calculated after normalization). These pages were selected from a sample of pages that were linked to from weblog entries on LiveJournal² over a period of eight hours. Most of these short pages were in English, but there were a few pages in other languages. Although these pages are biased towards English, we believe they are acceptable as a sample. This is because (as will be explained later) our final algorithm does not take any information into account beyond the text of documents. Gathering a more diverse corpus of short documents is a hard task that we leave for future work.

3 Cavnar and Trenkle’s algorithm

In this section we describe Cavnar and Trenkle’s method [1], and explain how we implemented it. It should be noted that strictly speaking, the authors originally used their method to determine the provenance of Usenet postings. The authors conflated a message being in a particular language with the message being posted in the appropriate newsgroup in the `soc.` hierarchy.

3.1 The method

An n -gram can be described as a sliding window over a set of characters. For example, in the text fragment “the quick”, the 3-grams are “the”, “he “, “e q”, “qui”, “uic”, “ick”, “ckt”, “kth”. Notice that the n -grams wrap around, so that there are always as many n -grams as characters in the text.

The method is quite simple. To each language in the corpus L_i corresponds a set of training documents D_i . For $n = 1..5$, we find the set of all character n -grams appearing in each D_i .

²<http://www.livejournal.com>

We form a ranked list G_i of the C most common n -grams in each D_i . In this way each language L_i is represented by a list of its most common n -grams G_i .

To classify a new document d , we find its C most common n -grams for n in 1..5 and form the list G . We then compare G to each of the G_i and find the closest one.

Let $G = (g_1, g_2, \dots, g_C)$, $G_i = (g_{i_1}, \dots, g_{i_C})$, and define $index(g, G)$ to be the index of g in list G , or $C + 1$ if $g \notin G$. To measure the similarity of two lists, we use the following formula:

$$d(G, G_i) = \sum_{j=1}^C |index(g_j, G_i) - j| \quad (0.1)$$

That is, we sum up the difference between the ranks of each n -gram in both lists. Note that this measure is not symmetric, so is a pseudo-distance rather than a distance.

We now formalize the algorithm.

Algorithm 3.2 (BASE algorithm) *Let $L = \{L_1, \dots, L_k\}$ be the set of known languages, and let $D = \{D_1, \dots, D_k\}$ be the set of training documents associated with each language.*

TRAINING PHASE: *To train the algorithm:*

1. For $i = 1..k$: compute G_i , the list of C most common n -grams in D_i for n in 1..5.

CATEGORIZATION: *Input: d , a document to classify. Output: $L_i \in L$, a guess as to the language of d .*

1. Find G , the list of C most common n -grams in d for $n = 1..5$.
2. For $i = 1..k$: compute $s_i = d(G, G_i)$.
3. Find i such that s_i is minimal, and return L_i .

3.3 Implementation and results

Cavnar and Trenkle report that a good value of C is about 400. We chose C to be 399. The classifier had an accuracy of 86.8%.

3.4 Discussion

We examined the documents which caused the classifier to produce an erroneous answer. We found that most of the mistakes were either made on short documents, or consisted of assigning to a document a language which was linguistically close to the correct language (e.g., Spanish and Italian). To understand this phenomenon, we examined the representative n -grams for sets of close languages (French, Spanish and Italian is an example of such a family). We observed that those lists shared many common n -grams, as may be expected. Therefore, our technique for selecting representative n -grams did not yield n -grams which had sufficient discriminatory power to allow the classifier to distinguish close languages. This led us to consider other schemes for feature selection.

4 Using the Fisher discriminant function

In this section we present a method for improving the baseline classifier described in Section 3. We use the Fisher discriminant function [2] to choose representative n -grams for all the languages, and compare new documents to the reference by using the cosine similarity measure.

Calculating the Fisher discriminant for all the n -grams in every language for $n = 1..5$ would be possible but is computationally prohibitive. This is especially true since the function must be recomputed from scratch if a new language is introduced in the corpus (on the other hand, in

Cavnar and Trenkle’s method only the profile for the new language must be computed). For this reason we have chosen to fix an n . It has been shown [10, 11] that $n = 3$ is a good choice. From now on when we speak of n -grams we will mean 3-grams. It would be interesting to see if allowing n to range from 1 to 5 would produce better results.

4.1 The Fisher discriminant function

The idea here is to select from all the n -grams appearing in every training document those n -grams which have the most discriminatory power to distinguish between languages. Intuitively, a “good” n -gram is common in one language but rare in the remainder of the languages.

Let us define the Fisher discriminant function F more formally. First, we need to fix some notation. As before, let $L = \cup L_i$ be the set of all languages in consideration, and let $D = \cup D_i$ be the union of the training documents for each language. Let \mathcal{A} be the set of all 3-grams appearing in D . Let $g \in \mathcal{A}$ and $d \in D$. Let $\#(g, d)$ denote the number of occurrences of g in d . We define the *frequency* of g in d as follows:

$$freq(g, d) = \frac{\#(g, d)}{|d|} \quad (0.2)$$

Now we can define the *normalized frequency* of g in d :

$$Freq(g, d) = \frac{freq(g, d)}{\sqrt{\sum_{h \in \mathcal{A}} freq(h, d)^2}} \quad (0.3)$$

The vector $(Freq(g_1, d), \dots, Freq(g_k, d))$ is the vector $(freq(g_1, d), \dots, freq(g_k, d))$ normalized to have length 1.

Following Chakrabarti [2], we can define the mean normalized frequency of an n -gram n in a set of training documents D_i :

$$\mu_i(g) = \frac{\sum_{d \in D_i} Freq(g, d)}{|D_i|}. \quad (0.4)$$

Finally, we can define $F(g)$, the Fisher discriminant of g , as:

$$F(g) = \frac{\sum_{L_i, L_j} (\mu_i(g) - \mu_j(g))^2}{\sum_i \frac{\sum_{d \in D_i} (Freq(g, d) - \mu_i(g))^2}{|D_i|}}. \quad (0.5)$$

4.2 The training phase

To train the classifier, we compute $F(g)$ for every n -gram in every training document. Then we sort the n -grams in descending order of their discriminant, and select the top 1000 as representatives, obtaining the vector $(g_1, g_2, \dots, g_{1000})$. Then for each language L_i , we form the vector \mathbf{v}_i where:

$$v_{ij} = \frac{\sum_{d \in D_i} \#(g_j, d)}{\sum_{d \in D_i} |d|} \quad (0.6)$$

To obtain the *representative vector* for language L_i , we normalize v_i and get:

$$\mathbf{r}_i = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \quad (0.7)$$

4.3 Classifying new documents

To classify a new document d , we form its representative vector as follows:

$$\mathbf{r} = (Freq(g_1, d), Freq(g_2, d), \dots, Freq(g_{1000}, d)) \quad (0.8)$$

We compare \mathbf{r} to each reference \mathbf{r}_i by computing their dot product $\mathbf{r} \cdot \mathbf{r}_i$. This is known as the *cosine similarity measure*, because $\mathbf{v}_1 \cdot \mathbf{v}_2 = \cos(\theta(v_1, v_2)) \cdot \|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\|$. If both vectors have unit length, the dot product measures the angle between them, which is understood as representing their similarity. Since a smaller angle means a larger cosine (at least in the first quadrant, where we work), we want to find the reference vector \mathbf{r}_m which maximizes the dot product. Then \mathbf{r} will be closest to \mathbf{r}_m , and we assign language L_m to d .

4.4 Algorithm formulation

We present a precise formulation of the NFC (for n -grams, Fisher and cosine) algorithm.

Algorithm 4.5 (NFC algorithm) *As before, let $L = \{L_1, \dots, L_k\}$ be the set of known languages, $D = \{D_1, \dots, D_k\}$ the set of training documents associated with each language.*

TRAINING PHASE *To train the classifier:*

1. Find \mathcal{A} , the set of all 3-grams in D .
2. For $i = 1..k$:
 - (a) For $d \in D_i$:
 - i. For $g \in \mathcal{A}$: find $\text{Freq}(g, d)$
 - (b) For $g \in \mathcal{A}$: find $\mu(g, D_i)$ using the information just computed.
3. For $g \in \mathcal{A}$: compute $F(g)$ using the information computed about μ and Freq .
4. Sort \mathcal{A} in descending order of F , and select the top 1000 to form $\mathbf{R} = (g_1, \dots, g_{1000})$.
5. For $i = 1..k$:
 - (a) Compute an approximation of \mathbf{v}_i by forming $C = D_{i1} + D_{i2} + \dots + D_{im}$, the concatenation of all the documents in D_i , and computing

$$\tilde{\mathbf{v}}_i = (\text{Freq}(g_1, C), \dots, \text{Freq}(g_{1000}, C)). \quad (0.9)$$

- (b) Compute $\mathbf{r}_i := \tilde{\mathbf{v}}_i / \|\tilde{\mathbf{v}}_i\|$.

CATEGORIZATION *Input: d , a document to classify. Output: $L_i \in L$, a guess as to the language of d . As above, $\mathbf{R} = (g_1, \dots, g_{1000})$ is the vector of reference 3-grams.*

1. Compute $\mathbf{v} = (\text{Freq}(g_1, d), \dots, \text{Freq}(g_{1000}, d))$.
2. Compute $\mathbf{r} = \mathbf{v} / \|\mathbf{v}\|$.
3. For $i = 1..k$: compute $\text{score}(i) = \mathbf{r} \cdot \mathbf{r}_i$.
4. Find i such that $\text{score}(i)$ is maximized, and return L_i .

4.6 Experimental results

Using this method, we achieved an accuracy of 93.9% on our sample. In other words, the number of errors is halved in comparison to Cavnar and Trenkle's algorithm. Although there were not very many short documents in our sample, we observed that the classifier performed badly on them. We decided to investigate the use of Web-only information to improve the performance of the classifier on very short documents.

5 Using Web information

To improve the performance of the Fisher discriminant-based classifier of Section 4, we considered adding Web-specific information to help the classifier. We considered top-level domains, outlinks and inlinks. We describe the results in this section.

5.1 Top-level domains

The algorithm

We conjectured that top-level domains are a good predictor of language. To test this hypothesis, we implemented a very simple classifier based on top-level domains only. To train the classifier, we find the most common language for every top-level domain in the training documents.

Then, to classify a new document, we assign the language corresponding to its domain. We give a precise algorithm below.

Algorithm 5.2 (TOP-LEVEL DOMAIN algorithm) *Let d be a webpage. Let $site(d)$ represent the site part of the URL of d . Define $tld(d)$, the top-level domain of d , as the part of $site(d)$ after the rightmost period.*

TRAINING PHASE *To train the algorithm:*

1. Compute $T = \{tld(d) : d \in D\}$, the set of all top-level domains in D .
2. For each $t \in T$, for each $L_i \in L$: compute $F(t, L_i) = |\{d \in D_i : tld(d) = t\}|$.
3. For each $t \in T$, find L_i such that $F(t, L_i)$ is maximized, and let $L(t) = L_i$.

CATEGORIZATION *Let d be a document to classify.*

1. Return $L(tld(d))$.

We use a naive definition of top-level domain. One could implement a more accurate definition (for example, considering `co.uk` and `ac.uk` as two different domains), but we chose to retain this definition because it is very easy to compute and does not require any knowledge of the classifier.

Results

Using this method we achieved an accuracy of 69.2%. So the TOP-LEVEL DOMAIN classifier is a quick and dirty way of getting a reasonable estimate for the language of a Web page, but is not adequate for our purposes. One could investigate combining a classifier based on top-level domains and the NFC classifier in a naive Bayesian way. We leave this for further investigation.

5.3 Outlinks

Using a sample of 799 links, we assessed that roughly 95% of all links are between pages written in the same language. Let P be a page which is too short for the classifier to give a reliable answer. We consider the pages P points to (its outlinks) O_1, \dots, O_n . We compute the language of each O_i using the base classifier, and select the most common language by majority vote and assign it to P .

Unfortunately, in preliminary experiments this method did not work as well as we hoped, because short pages are unlikely to contain links. We did not pursue this line of investigation further.

5.4 Inlinks

We then turned to inlinks of a page P , that is pages which point to P . The hope is that if a page is short, there may nonetheless be longer pages which point to it, and which could be leveraged to infer the language of the short page.

The sample of webpages we originally collected did not contain many very short pages. This is in part due to the bias introduced by the manual classification of the pages, because we had less trouble confidently assigning a language to longer pages. In order to test the performance of our algorithm on very short pages, we used the sample of short Web documents described in section 2. We divided this sample into two subsamples: one contained documents whose length was less than 25 characters, and the other one contained documents whose length was between 25 and 50 characters.

We did not retrain the NFC algorithm on this new sample. We believe that to be effective, our language recognizer must be versatile enough to be trained once and then used anywhere. We

very carefully selected our original training set to have many documents in each language, a feature which would be impossible to reproduce by random samplings of the Web.

We give a more formal description of the INLINKS algorithm.

Algorithm 5.5 (INLINKS algorithm) *Input:* d , a webpage to be classified. *Output:* $L_i \in L$, a guess as to the language of d .

1. If $|d| > MIN_LENGTH$ for some minimum length MIN_LENGTH , return $NFC(d)$.
2. Find $Inlinks(d)$, a set of pages which point to d such that

$$|Inlinks(d)| \leq MAX_INLINKS \quad (0.10)$$

(we chose this set to contain at most 10 pages) and such that the length of each document in $Inlinks(d)$ is at least MIN_REF_LENGTH . In practice we found this set using the Google API³ and a query of the form `link: url(d)`.

3. Compute the multiset $R = \{NFC(r) : r \in Inlinks(d)\} \cup \{NFC(d)\}$.
4. Return $L_i \in L$ such that L_i is the element of R with the largest multiplicity.

We chose MIN_REF_LENGTH to be 150 characters. We leave the investigation of the relationship between the optimal MIN_REF_LENGTH and MIN_LENGTH for future work.

5.6 Experimental results

On the first sample (less than 25 characters), we first ran the NFC classifier. The accuracy was 33.8%. Then we used the INLINKS classifier. To find the pages pointing to the pages in our

³<http://api.google.com>

Lenght in characters	Accuracy of NFC	Accuracy of INLINKS
< 25	33.8%	71.0%
$\geq 25, < 50$	43.3%	70.9%
< 50	38.0%	70.0%

Table 5.1: Accuracy of INLINKS compared to NFC

sample, we used the Google API and limited the maximum number of inlinks to 10 per page. We then fetched those pages, classified them, and selected the most common language of these pages as the result. Using this method, we achieved an accuracy of 71.0%, or more than double the accuracy of NFC. We observed that the classifier mainly made mistakes on pages which had few or no inlinks.

On the second sample (length between 25 and 50 characters), the NFC classifier had an accuracy of 43.3%, and the INLINKS classifier achieved an accuracy of 70.9%. So we again see a substantial gain by using inlink information, though the difference is not as large as in the sample with shorter pages.

6 Related work

A number of methods have been developed to identify the language of a document. We mention some categories and representative works. We base these descriptions on the papers by Sibun and Reynar [10] and Souter et al [11], as they both contain a survey of existing methods.

Dictionary methods are a commonly used approach. They basically consist in building a list of commonly found words (usually stopwords such as “the”) in languages and scanning unknown texts for those words and assigning a language to them based on those words. They work moderately well, but are not automatic in that the words to be included in dictionaries are chosen by linguists. In a similar vein, unique strings have been considered as a way to classify documents [11].

This method performs poorly because unique strings found in training documents might not appear in testing documents.

Hidden Markov Models [3] are another line of research. They have been found to be slightly less efficient than the n -gram method. They are also not easy to implement. The basic idea behind HMMs is to compute reference Markov chains for the training sets. Then when a text comes in, we must compute the probability that a certain model produced the output that we see (this is the “hidden” part: the model is hidden but the output is what we have to classify). The model with the highest probability is chosen as an answer.

Many probabilistic methods have also been considered. Sibun and Reynar [10] give an overview. These methods are shown to perform well on relatively long documents, but generally not on documents shorter than one line (about 80 characters).

Most recent work in language identification has focused on speech language recognition and image language recognition, because it was deemed [6] that Cavnar and Trenkle’s method solves the problem of textual language identification.

Kikui [4] developed an approach for language and character set identification on the Web. The approach is to use some bytes to make a few guesses about possible language encodings. For each such possible encoding, the text is decoded and a statistical language model is used to find the most likely language for this encoding. This is done until a satisfactory answer is found. This method solves a slightly different problem than this one, since it identifies a character encoding first and then determines a language based on this information.

The idea that links between Web documents carry interesting information has been used by the HITS [5] and PageRank [9] algorithms. We derived the idea of using inlinks from these works.

7 Conclusion

In this paper, we report on our experiments with a classical method of language identification based on n -gram frequencies, and show that it did not perform as well as expected on our sample of web pages. We show that refining the n -gram selection method by using the Fisher discriminant improves the accuracy of the classifier. However, the resulting classifier does not perform well on short webpages. We incorporate web-specific information (inlinks) into the classifier and test it on a new sample of short webpages. The new information more than doubles the accuracy of the classifier on pages with less than 25 characters, and slightly less than doubles the accuracy on pages with less than 50 characters.

We do not investigate incorporating top-level domain information into our classifier, perhaps for use on pages where our classifier has low confidence. It would be interesting to see if this information could be leveraged in a useful way. We also do not investigate the optimization of the inlink classifier, by fine-tuning the number of inlinks required and their length.

Finding inlinks statically is costly. We envision that our classifier could be used by a crawler while crawling, perhaps maintaining confidence information and link information as it finds pages. One could also imagine implementing the classifier as a recursive graph algorithm on the web graph (or the graph of collected web pages). We have begun investigation in this direction, by starting to integrate our classifier with Nutch [8], an open-source search engine.

References

- [1] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, NV, U.S.A., 1994.
- [2] Soumen Chakrabarti. *Mining the Web: Analysis of Hypertext and Semi Structured Data*. Morgan Kaufmann, 2002.
- [3] Ted Dunning. Statistical identification of language. Technical Report MCCS 94-273, New Mexico State University, 1994.
- [4] Gen-Itiro Kikui. Identifying the coding system and language of online documents on the internet. In *Proceedings of the 16th International Conference on Computational Linguistics*. Morgan Kaufmann, 1996.
- [5] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, September 1999.
- [6] Gianni Lazzari, Robert Frederking, and Wolfgang Minker. *Multilingual Information Management: Current Levels and Future Abilities*, volume XIV–XV of *Linguistica Computazionale*, chapter 7. Insituti Editoriali e Poligrafici Internazionali, 2001.
- [7] Shanjian Li and Katsuhiko Momoi. A composite approach to language/encoding detection. Technical report, Netscape Communications Corp, 2002.
- [8] The Nutch project. <http://nutch.org>.
- [9] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the Web, 1999.
- [10] Penelope Sibun and Jeffrey C. Reynar. Language identification: Examining the issues. In *5th Symposium on Document Analysis and Information Retrieval*, pages 125–135, Las Vegas, NV, U.S.A., 1996.
- [11] Clive Souter, Gavin Churcher, Judith Hayes, John Hughes, and Stephen Johnson. Natural language identification using corpus-based models. *Hermes, Journal of Linguistics*, 13, 1994.

[12] The Windows-1252 character set. <http://www.microsoft.com/globaldev/reference/sbcs/1252.htm>.