Language Detection based on Unigram Analysis and Decision Trees

Sofia Bastrup
LTH Lund, Sweden
`dat00sob@ludat.lth.se`

Christina Pöpper
ETH Zürich, Switzerland
`poepperc@student.ethz.ch`

Abstract

This document describes the process of implementing a decision tree for language detection. First, text profiles are computed for each text in the training set and out of this a decision tree is built. Finally evaluation is made with query texts. To compare the performance of different approaches, a few variations of trees are implemented; sets of 26 respectively 56 characters as attributes are compared, furthermore a 'direct-child'-tree versus a 'neighbours'-tree are implemented for comparison. The 'direct-child' tree gave the greatest number of correct answers, in contrast to the 'neighbours' tree that did not give any incorrect answers (however many no results). Suggestions of enhancements of performance is given. The authors conclude that the language detector gives comparatively good results given that the implementation only considers unigrams.

1 Introduction

The goal of this project was to write a *language detector*, i.e. a system to identify the language of a given text automatically - out of a predefined number of possible languages.

Applications of automatic language detection involve language processing such as automatic retrieval of texts in the desired language (e.g. from the world wide web) as well as studies of language use (e.g. estimation of language frequencies in the internet).

Several broad approaches exist to the problem of language detection.

One obvious technique is to use a lexicon for each possible language and compare the words in the sample text with those in the lexica to find the lexicon with the highest correlation. This involves huge numbers of data to be managed and processed as well as difficulties when dealing with highly inflected languages. On account of the drawbacks of the lexicon method, grammatical words are used as discriminant in [Gi2].

Another technique is to use the alphabet. But, as stated in [Gi1], the alphabet is not very useful, because accented characters are not as frequent as needed and they often belong to several alphabets. Thus, this clue does not really allow to discriminate the right language.

A further approach is the use of n-gram analysis. *n-grams* are sequences of n letters. The basic idea is to compute, from a training set, a profile for each considered language based on the probability of letter sequences. For a given text the language with the nearest profile is selected.

On account of the drawbacks of the lexicon and alphabet methods, we decided to implement the last of the mentioned techniques: *language detection by n-gram analysis*.

2 Decision Tree Approach

The process of language detection can – in our case – be divided into three steps. In the first step, the frequencies and probabilities of the letters (unigrams) and combinations of letters (n-grams) of the training set are computed. This gives a profile for each language and has do be done only once. As the second step, we decided to build up a decision tree out of these profiles. Finally, the third step is to evaluate the decision tree for a given text.

In our implementation we only consider unigrams (i.e. letters) so far.

2.1 Training set

The implemented language detection method depends entirely on the quality of the training set. For obtaining the training set we used the Europarl Corpus of European Parliament Proceedings ([EPP]), version 1, for the following ten languages: *danish, dutch, english, finnish, french, german, italian, portuguese, spanish,* and *swedish,* each consisting of texts with 17-22 million words, corresponding to 60-80 MB each.

2.2 Language/text profile

The profile for each text in the training set (corresponding to a certain language) as well as the profile for the query text is given by the probabilities of all n-grams (in our implementation: unigrams). In order to deal with these probabilities and to transform the training set profiles into a decision tree, the probabilities are converted into intervals. Eight intervals were used corresponding to the probabilities of letters:

interval	corresponding probability	interval	corresponding probability
0	0 - 0.001	4	0.09 - 0.12
1	0.001 - 0.03	5	0.12 - 0.15
2	0.03 - 0.06	6	0.15 - 0.18
3	0.06 - 0.09	7	$>$ 0.18

Interval 0 is not only assigned to letters not occuring at all, but also to letters occurring with a tiny probability of smaller than 0.001 (0.1%) in order to take loan words from different languages into account and prevent these from contaminating the profile of a language.

After this step, a typical profile for a language looks like this example for danish (containing only the common character set, see also section 2.3.4): [letter:interval]

a:2	b:1	c:1	d:2	e:6	f:1	g:2	h:1
i:3	j:1	k:2	l:2	m:2	n:3	o:2	p:1
q:0	r:4	s:2	t:3	u:1	v:1	w:0	x:0
y:1	z:0						

2.3 The decision tree

2.3.1 General building and structure

A decision tree is used for making classifications. The input for the tree is given by examples consisting of values of the set of attributes. Each internal node represents a test of the attribute value. The branches of the node are labeled with the different interval values. When reaching a leaf node the classification of the example is returned.

In our implementation of the decision tree the examples are represented by texts of different languages - a set of training texts when building the tree and a set of test texts to evaluate the tree. The classification attribute is the language of the text.

The attributes are the text profiles that are calculated out of every text in the training set. The values of the attributes are the intervals which correspond to the probabilities of the unigrams.

2.3.2 The implemented trees

Two different implementations are used in the language detector. We call them *direct-child-tree* and *neighbours-tree*.

Direct-child-tree: For the direct-child-tree, the idea is to take the profiles of the training texts and assign each of them exactly one path through the tree. In the final leaf, the language of the text is stored. I.e. for every internal node, containing a letter (the chosen attribute), the interval corresponding to this letter in the training text gives the child were we have to go on recursively.

In a first version of an implementation we only had one example for each language in the training set. This tree had a lot of leaves where no language could be determined. This happened because the query text has to have a profile much as same as the training text to be detected (in a path where all the attributes have to be used, the two profiles have to be identical). To overcome this problem we split up the training text into about 300 smaller text for each language. Then a query text has a higher probability of having a similar profile as one of the 300 training text profiles.

Neighbours-tree: Besides the direct-child-tree described in the previous paragraphs, we implemented a second approach, the *neighbours-tree*. The determining motive behind this approach is the fact that the direct-child tree (in its first version) is strict in respect to the profiles: the intervals of the n-grams of a query text has to correspond to the profile of a training text for this language. As described, the more training texts for each language are used for the direct-child-tree, the greater is the probability that a similar profile exists. But the problem remain, especially when the intervals are small and the probabilities lie at the border of the intervals (e.g. letter *a* may have probability 0.089 in the training text corresponding to interval 3, in the query text the probability may be 0.091 corresponding to interval 4, but the languages may be easily identical).

In the neighbours-tree we compensate for this by using only the profile of one text (of 60-80 MB size; concatenation of all texts of the same language used for the direct-child-tree) for each language and by assigning the language to the child of the interval as well as to both of the child's neighbours. Thus, texts varying only slightly in the intervals may be recognized as the same language nevertheless.

In Figure 1, a typical node in a neighbours-tree is depicted. Let's say, the languages danish and finnish are still possible and attribute *e* is chosen. The interval for *e* is 6 for danish and 4 for finnish. Then this part of the tree will look like this figure. Only for interval 5 both languages are still possible.

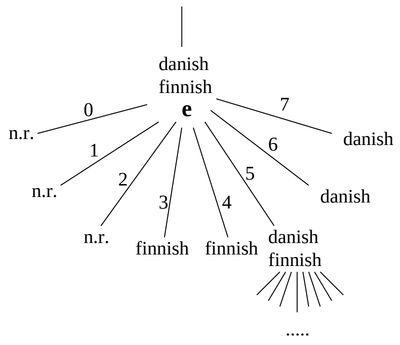

Figure 1: Typical node with children in the neighbours-tree. n.r. = no result. Letter e is chosen as attribute for this node.

The drawback of this approach are the many overlappings of letter intervals for the languages. The tree gets very big, because for many branches a great number of recursive steps has to be made. Because of limited memory, we had to limit the height of the tree to 8.

2.3.3 Choose attribute

When building a decision tree an appropriate attribute has to be chosen in each internal node. The method for choosing the attribute (we call it 'choose attribute method') is of importance concerning the depth and effectiveness of the tree. A "perfect" attribute is one that splits the set of examples into new sets in which all examples have the same classification. Choosing the right attribute gives a compact tree with a smaller search depth.

Two different choose attribute methods were implemented.

In the 'direct-child'-tree, information theory is used. The goal is to choose the attribute as similar as possible to the "perfect" attribute. To choose an attribute calculations of how much information is gained are made for all the attributes. This is compared to how much information there is currently. When building the tree the information content is calculated by counting how many examples there are in the set of each classification of the goal attribute. In case a node is reached when there are no more attributes in the set to choose from, but there are still examples not having the same classification, then a majority value is determined. This means that the language having the greatest number of examples in the set is returned as the classification of the text.

When building the 'neighbours'-tree there is only one example for each language (see section 3.2). Therefore, a simple choose attribute method is enough. Among the examples it calculates the difference between the highest and the lowest value for each attribute. The attribute with the largest difference is chosen.

2.3.4 Variation

Special characters: In our first approach we only considered 26 characters of the alphabet, the letters that all languages included in our research have in common. To develop our program further we extended the set of attributes to 56 different unigrams (containing letters such as $\hat{a}$, $\grave{e}$ and $\acute{o}$). This set now includes all special characters that exist in the considered european languages.

3 Results

Our test set consists of 33 different texts in the ten considered languages - at least three texts for each language. The (mainly contemporary) test texts were obtained from miscellaneous sources, from newspaper articles to scientific reports and literature. The sizes vary between about 270 and 67,600 words (or from 1.7 to 430 KB). To our surprise, we could not detect a correlation between the size of the text in this range and the correctness of the result. We suppose, that this correlation requires a greater number of test.

In the following, the reporting of the results is split up into the two implemented kinds of trees.

3.1 The 'direct-child-tree'

For the direct-child-tree each branch is labeled only with one value of the attribute. This in comparison with the 'neighbour-tree', where each branch is assigned three adjacent values.

Our results are as follows (33 tests):

tree	correct	false	no result
1	24	5	4
2	22	9	2
3	24	6	3
4	24	7	2

Four slightly different trees were implemented to compare the results. The first two compared the choose attribute methods. The method that uses information theory (tree 1) only appeared to be slightly better than the simple method (tree 2). This improved method gave two more correct answers and four fewer incorrect answers.

The second variation in the 'direct-child' approach is the tree with 26 attributes respectively the one where also the special characters are included as attributes. Both trees use the information theory when choosing attribute. Comparing those two trees (tree 1 and 3) showed surprisingly not any big difference at all in performance of classification. They both gave 24 correct answers. Furthermore the 56-characters-tree actually gave one more incorrect answer, which makes the 26-characters-tree the tree that showed best performance in the research. Still there was such small difference between those trees that it is reasonable to assume that in a more extensive research, those trees would yield the same performance.

Tree 4 used 56 characters and the simple choose attribute method.

All four trees determine a majority value when there are no more attributes.

3.2 The 'neighbours-tree'

The recursion depth for our version of the neighbours-tree is 8, bounded above by the size of memory on the used computer with a limited student account. Due to the height 8 of the tree (the maximal height is 26 or 56 respectively depending on the number of considered characters), many branches do not contain a result in the final leaf, as several languages are still possible, but can not be determined any further. Hence, we got many 'no result'.

We tested the neighbours-tree for the 26 character set and our first chose attribute method (a majority language would make no sense here as there is only one text for each language).

Our results are as follows (33 tests):

correct	false	no result
5	0	28

What strikes is the fact that there are no false answers - either the result is correct or there is no result at all.

More than likely, the number of 'no results' would decrease if the number of recursion steps could be increased.

4 Enhancements and Alternatives

An obvious enhancement is to consider n-grams, i.e. combinations of letters, and not only unigrams, because n-grams potentially contain much more information about the language. Many languages have bi- and trigrams that are representative for them, such as the english '*th*' or the german '*sch*'.

An alternative to the implemented decision tree approach is to use a *vector approach*. Thus, each language would be assigned a vector with 26 or more elements, depending on the number of characters taken into account. The result of the language detection will be the language whose vector has the smallest distance to the vector of the given text. The definition of distance depends on the chosen method. We did not implement this approach and thus can not compare the decision tree and vector approaches.

One more alternative turns up: Would the language detector show any improvement in performance if the value zero would represent no occurrences at all of a character in the text, instead of the interval 0 - 0.001?

There is obviously a difference between the letters that exist in a language only rarely (at few occasions) and the letters that do not exist at all. As for example the letters ä and ö in Swedish that do not appear in any latin language. Those characters can give a unique profile to the language they appear in. Still, no big difference in performance was detected between the 26- and 56-characters-trees. One way to make a distinction between these trees would be to represent only no occurrences with the value zero. There comes though a risk with this approach. The letters that exist only rarely in the language, could in one specific text be assigned the value 0 because in this text there are no occurrences at all. This is the reason why we chose to assign the value zero to those letters with a probability of 0 - 0.001. Anyway some emphasis on special characters would with high probability yield better performance.

5 Conclusion

A surprise for us during the development of the language detector was the fact that the enhancement from the simple character set to special characters did not yield the wished improvement. A reason for this may be that the special characters are not treated in any specific way.

Still far from a complete and satisfying program we were nevertheless surprised by the comparatively good results yielded by the very simply approach with unigrams. We are confident that an extension to two- and three-grams will improve the prediction significantly.

Acknowledgements

The authors gratefully acknowledge the very useful suggestions and help for this work by Pierre Nugues and Richard Johansson.

References

[CT] William B. Cavnar & John M. Trenkle. *N-Gram-Based Text Categorization*. Environmental Research Institute of Michigan.

[EPP] Europarl Corpus of European Parliament Proceedings. *http://www.isi.edu/ koehn/europarl*.

[Gi1] Emmanuel Giguet. *Categorization according to Language: A step toward combining Linguistic Knowledge and Statistic Learning*. Université de Caen, France.

[Gi2] Emmanuel Giguet. *Multilingual Sentence Categorization according to Language*. Université de Caen, France.

[GN] Gregory Grefenstette & Julien Nioche. *Estimation of English and non-English Langage Use on the WWW*. Xerox Research Centre Europe, France.