Information Retrieval (2ID10)

Assignment 5: Language Identification Module

Assignment Report

Group 23
Martin van Wingerden, 0533967
Mark Kurvers, 0569661
Dennis Bolio, 0571662

Table of contents

List of figures

List of tables

1 Introduction

For the course *Information Retrieval (2ID10)* given at the *Eindhoven University of Technology* during the spring of 2006 the students have to carry out an assignment related to information retrieval. Each lecture of the course discusses a different subject of information retrieval and proposes at least one assignment. At the end of the lecture series, each group of three students chooses an assignment to work on. This document describes the solution of group 23 for assignment 5: *Language Recognition* [ASS5].

The layout of the document is as follows: the assignment is described in detail in chapter 2. The relation of the assignment to the literature presented in the course is explained in chapter 3. Chapter 4 gives an overview of the architecture we used, including the algorithms. In chapter 5 some implementation details are highlighted. In chapter 6 the user interface of our implementation is described, along with a short tutorial. Chapter 7 discusses the performance of our solution by testing it against a test set. Finally, in chapter 8 we draw conclusion about the assignment and our solution to it.

2 Assignment description

In this chapter we discuss both the assignment description and our interpretation of the assignment. The assignment [ASS5] was part of the fourth lecture of the course given by Wessel Kraaij from TNO [LEC4].

Basically the assignment is to build an application that is able to read a piece of text (a number of sentences) and determine in which of the 21 European languages the text is written. Below we present a description of the problem, the assignment in separate parts and the proposed steps by Wessel Kraaij.

Problem:
Develop a language identification module using generative character models for the 21 European languages.

Assignment:

- Develop and test the language identification module on the European Union constitution corpus which can be found at [OPUS]
- Test the language identification module on some non European languages (e.g. taken from the KDE corpus on the same site)
- Adjust the classifier in such a way that it classifies them as "unknown" instead of the best matching EU language.

Proposed steps:

1. Download the EU constitution in 21 languages from [OPUS].
2. Split each language corpus in a training, test and evaluation set.
3. Train classifiers for each language using character trigram models smoothed with character bigram models.
4. Optimize the smoothing parameter on the test data.
5. Compute the accuracy of the classifier, by taking 100 sentences from the evaluation set of each language.
6. After adjusting the classifier, test the accuracy of the classifier on the same evaluation set plus 100 sentences from 5 non European languages. Present results per language.

We have split the corpus of each language in different sets in the following way:

- Entire text of the corpus as training set.
- As test set we have taken random sentences from the corpus as well as text sentences from the Internet.
- The evaluation set consists out of 100 randomly selected sentences from the corpus of each language.

3 Literature discussion

In this chapter we discuss the literature we used for our assignment. In the first paragraph we briefly discuss the literature we used in modeling our solution and algorithms. The second paragraph discusses alternative solutions. In the last paragraph we make some links to other topics in the field of information retrieval.

3.1 Related literature

In the proposed steps of the assignment it is stated that a classifier should be trained using character trigram models smoothed with character bigram models. This language identification method is classified as an *N*-gram counting algorithm. In lecture 4 of the course given by Wessel Kraaij this algorithm is treated.

Because of this, the main literature used is the slides of that lecture: [ASS4]. The algorithm we used in our solution is based on the Basic Ranking Formula of slide 15 from [ASS4].

There is some literature that is related to *N*-gram algorithms (and thus related to our solution). We will mention two of these for the interested reader. Both papers contain many references to other papers in the field of *N*-gram counting and language recognition algorithm.

The paper "*Statistical Identification of Language*" by Ted Dunning [DUNN] gives a detailed description of *N*-gram counting. Different classification methods and even practical results are presented in that paper. In section 3.1 of [ROSE] a short description of *N*-gram counting is presented along with many references to other papers.

3.2 Alternative solution discussion

In the assignment description (chapter 2 and [ASS5]) it is explicitly stated that the language recognition algorithm should use an *N*-gram method. More specifically: classify using character trigram models smoothed with character bigram models. This means we didn't have to make a comparison and choice between different algorithms presented in the literature. We will however give brief descriptions of other language recognition algorithms in this paragraph.

In [ROSE] section 3 an overview is given of the different statistical language recognition algorithms. Besides the *N*-grams models, decision tree models, linguistically motivated models, exponential models and adaptive models exist. Below we will briefly discuss two of these alternative methods. For more detailed descriptions and references to these algorithms we direct the interested reader to section 3 of [ROSE].

In a **decision tree model** the language is modeled as a tree. The tree is build by recursively partitioning the training data (called the history) of the language. At each leaf of the tree a question can be asked about the probability of occurrence of the next word in the sentence. This probability is calculated using the history at that leaf.
Linguistically motivated models are models which assign probabilities to linguistic properties of languages. This can be done for instance by describing the language in a context free grammar where sentences are generated by starting at a non-terminal and repeatedly apply transitions. Each transition is assigned a certain probability.

3.3 Links to other IR topics

The link of our project to other IR topics lies mainly in the field of data mining and data parsing. According to results presented in [DUNN], the more training data in the database, the more accurate the results of the algorithm are. So it is very important to have large quantities of training data so that the algorithm will perform better in the end.

The second topic to which our assignment is related is that of data parsing. As suggested in the assignment, we used the EU constitution in 21 languages from [OPUS]. This website presents the information in the XML format, so a parser was needed to extract the sentences of the different languages.

4 Architecture overview

The architectural approach we used in our solution is described in the following chapter. The first paragraph discusses the architecture of the language identification module. In the second paragraph the algorithm we use is described in detail.

4.1 Architecture of the solution

After we had decided on the assignment that we were going to do, we started thinking about the architecture of the application that we were going to develop. We created the following schema of different components that would make up our language recognition application.

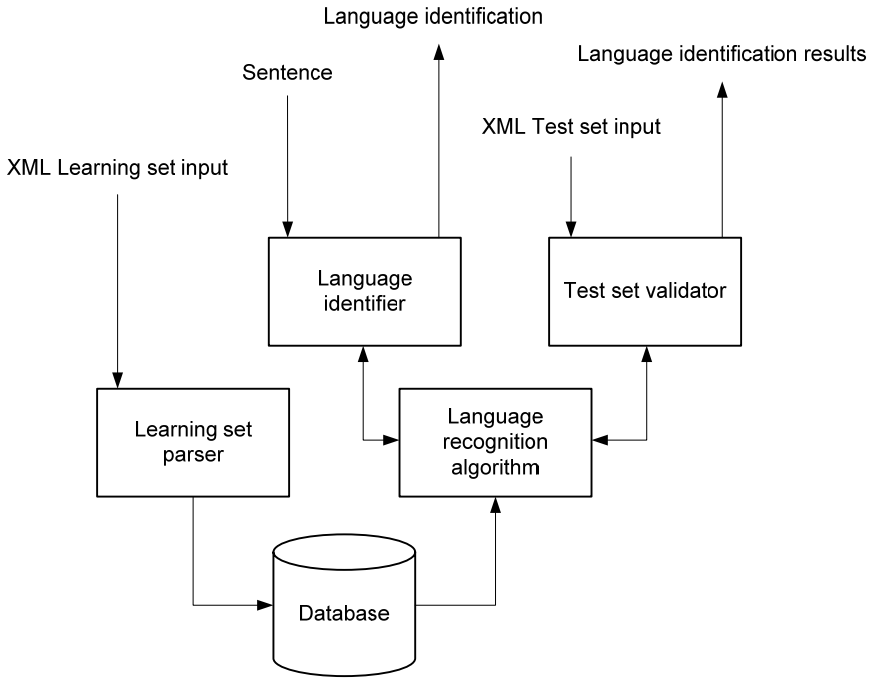

Figure 1: Overview architecture of the solution

The different components of the architecture are briefly described below.

Database
The database is the central component of the application, as it stores all statistical information used for language recognition. It stores all trigrams and bigrams that have been encountered, along with the frequencies of these trigrams and bigrams in each language. All this statistical data is put in the database by the learning set parser.

Learning set parser
The learning set parser takes as input an XML document of which the language is known. It then reads all bigrams and trigrams from the words of the XML document, and increases the frequencies of these bigrams and tigrams for that language in the database. If it encounters new trigrams or bigrams, then it will add them to the database.

Language recognition algorithm
The main component used for language recognition is the language recognition algorithm. This component contains an implementation of the algorithm as presented in [referentie naar slides van lecture 4]. The trigram and bigram data that is used by the algorithm is taken from the database.

Language identifier
This is the user application part where a user can input a sentence which will then be identified. This part uses the language recognition algorithm module to identify the language of the sentence.

Test set validator
The test set validator contains a randomly selected set of 100 sentences of each language. The validator then lets the language recognition algorithm identify all these sentences and checks the results. It then displays the results of the language recognition algorithm so that its accuracy can be determined.

4.2 Language identification algorithm

The algorithm that is used for the language identification is classified as an *N*-gram counting algorithm. For details on *N*-gram algorithms we direct the reader to [LEC4], [DUNN] and [ROSE]. The basic formula for the algorithm is:

$$P(T_1, T_2, \cdots T_n \mid D) = \prod_{j=1}^{n} P(T_j \mid D)$$

This formula calculates the probability of a certain set *T*, given a certain dictionary *D*. This probability is calculated, by multiplying all probabilities of elements of that set given a certain dictionary.

In our algorithm, the set is a sentence in a certain language, and the elements of that set are the trigrams of that sentence. The dictionary in our case is the statistical information of all languages about the frequency of occurrence of that trigram in that language.

Because data can be sparse which can lead to zero probabilities, smoothing is added. The trigram probability is smoothed with bigram probabilities. This is added in the formula:

$$P(T_1, T_2, \cdots T_n \mid D) = \prod_{j=1}^{n} \lambda P(T_j \mid D) + (1-\lambda) P(T_j \mid C)$$

Here, λ is the smoothing constant, and C is the dictionary of statistical information of all languages about the frequency of occurrence of bigrams in that language. Now the probability of a certain trigram occurring in a certain language is multiplied by λ. This means that the weight factor of that probability is also λ. Added to this probability is the

probability of the bigrams occurring in that language. This probability is multiplied by (1 - λ), giving the bigram probability a weight factor of 1 - λ. For instance, if λ = 0.2 then the trigram probability has a weight of 20% in the total probability, and the bigram probability has a weight of 80% in the total probability.

The last step in the algorithm is of practical value. Since sentences can contain many trigrams, and probabilities of trigrams occurring are very low, the total probabilities for large sentences can end up being smaller than $1 * 10^{-300}$. Therefore, the slides offer a practical solution, to transform the formula into:

$$\log P(T_1, T_2, \cdots T_n \mid D) = \sum_{j=1}^{n} \log\left[\lambda P(T_j \mid D) + (1-\lambda) P(T_j \mid C)\right]$$

The probabilities now are expressed as negative numbers, where the higher the number (the closer to 0), the higher the probability that a sentence is of that language. As a last step we choose to normalize these results, so that they can be compared against results of other sentences. We did this by dividing the result by the number of trigrams the sentence contains.

These formulas are translated into the following algorithm which is used by our application:

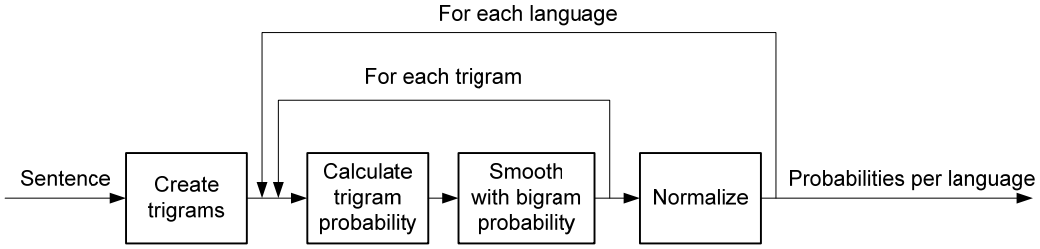

Figure 2: Block diagram of the language identification algorithm

The algorithm first splits all words into trigrams. Then for each language it will calculate for each trigram the probability that this trigram occurs in that language. The probabilities of each trigram are then smoothed by the probability that the bigrams of that trigram occur in that language. This smoothing is parameterized by a constant that determines the weight factor of the trigram probability and the bigram probability. Finally, the results are normalized and the algorithm gives a list of known languages and the probability for each language that the test sentence is of that language.

5 Implementation description

This chapter discusses the implementation details. In the first paragraph implementation choices such as chosen programming language are justified. In the second paragraph we discuss some of the issues we encountered during the implementation. The third paragraph gives an overview of the implementation. The last paragraph gives a mapping of the source code files to the blocks presented in chapter 5. For a detailed description of the source code we direct the interested reader to the well documented source code itself.

5.1 Implementation choices

The most important implementation choices made are discussed below.

Using PHP as programming language
We decided to use PHP because the assignment was ideally suited to be implemented as a web application. So that leaves a few choices open such as ASP, .NET and PHP. Because PHP has lots of easy to use string manipulation functions and database connectivity this was an obvious choice. Moreover, two of our team members already had lots of experience programming in PHP.

Using multibyte character encoding (UTF-8 encoding)
Our first implementation was a very direct implementation of the suggestions made in the slides, that solution worked quite well but only supported ASCII characters so there was no way to recognize a Greek text and all East-European languages were not recognized either. After we discovered the source of the problem (using ASCII instead of UTF-8) we totally rebuild our application to support, the so called mulitbyte characters using the in PHP included multibyte library.

Using MySQL as database
When using PHP as programming language, using MySQL as database is a straightforward choice. MySQL is an easy to use, fast and freely available database management system. The build-in functions of PHP to use a MySQL database also are very easy to use. We used a MySQL database to save all the trigrams and bigrams we extracted from the European constitution. In the application we check for every bigram and trigram we find, how many times it occurs.

Using DomXML to parse the input
Because the files of the learning set to train our application were coded in XML we needed to build a tool to parse the files into the MySQL database. We decided to use the DomXML parser for this purpose because it is supported in PHP version 5 and provides a rich set of functions to read and manipulate XML files. We wrote some almost trivial XSLT sheets to process the files.

5.2 Implementation issues

In this paragraph we will discuss some implementation issues and what our solutions to them.

Corrupt XML files
The first issue we had were corrupt XML files from the EU constitution corpus at [OPUS]. There where some missing closing tags so we had to correct these files manually. Examples are C2004310DE.01018601.xml, line: 3576, 3580 & 3583.

Usage of multibyte character encoding (UTF-8 encoding)
None of our group member had previous experience using UTF-8 so we had to figure out how to use UTF-8 in the database; in the files and how to use the multibyte string manipulation functions from PHP. But that was a very interesting search and we learned new things in that search.

Multiple languages in a file
We discovered that in some of the XML files of [OPUS] multiple languages do occur. These are sentences like: “The following have also signed this Final Act, in their capacity as candidate States for accession to the European Union, having been observers to the Conference:” and that in all 21 languages. This leads to corruption of the learning set but also to corruption of the test set. We didn’t solve this problem.

5.3 Implementation details

As already presented in chapter 5, our architecture consists out of several blocks. Each of these blocks will be explained by a program flow diagram illustrating how that particular block is implemented.

5.3.1 Program flow of the learning set parser

To train the program in recognizing the different languages we have build a module called “Learning set parser”. The parser reads XML files taken from the EU constitution corpus found at [OPUS] and extracts the trigrams and bigrams from the files. The trigrams and bigrams are stored in a database. Given below is a figure illustrating the program flow of this process.

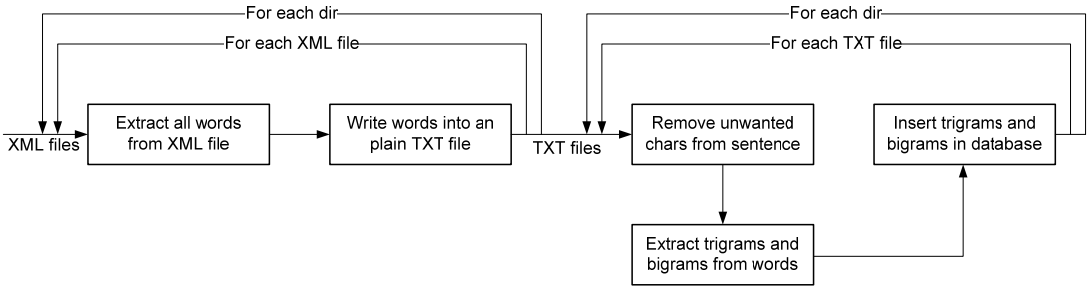

Figure 3: Program flow of the learning set parser

The program first recursively travels through each directory to extract all the text nodes, which are the actual words, from the XML files. These words are written into temporary plain TXT file, one TXT file for each XML file. For each TXT file, the program then removes unwanted characters from the words, extracts the trigrams and bigrams from the words and stores their occurrence frequency into the database. A similar program is used for randomly extracting the 100 test-sentences for each language.

5.3.2 Program flow of classifying a sentence

Below a picture is presented containing the program flow of classifying a sentence to a language. This flow corresponds to the blocks “Language identifier” and “Language recognition algorithm” from the architecture overview presented in chapter 4.

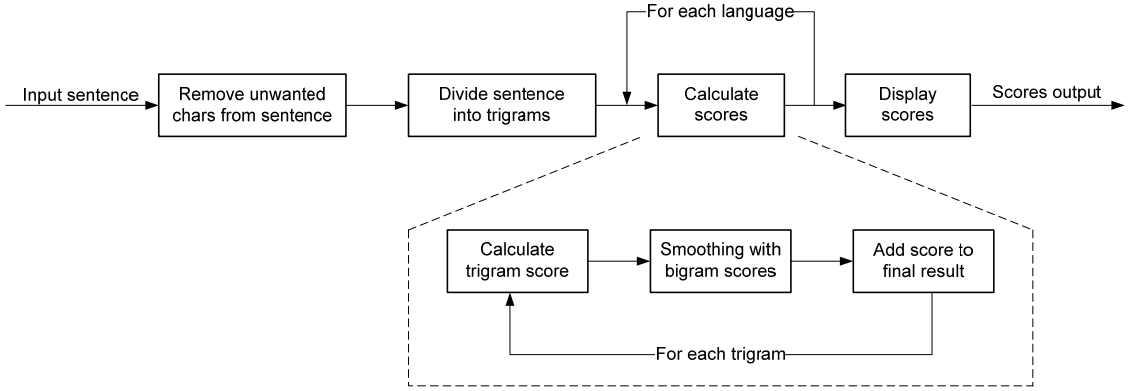

Figure 4: Program flow of classifying a sentence

Remove unwanted chars from sentence: All unwanted punctuation marks are removed and the sentence is converted to lower-case characters only.

Divide sentence into trigrams: The lower-case sentence is converted into an array of trigrams. Only words containing three or more characters are converted. Each word is divided into (charcount(word) – 2) trigrams. For example the word “hello” is converted into the following three trigrams: “hel”, “ell” and “llo”.

Calculate scores: After the dividing the sentence in trigrams, the program calculates the score for each language. For details of the score calculation procedure we direct the reader to chapter 4 where the score algorithm is explained. Basically the algorithm calculates for each trigram the probability of occurring in the language and smoothes this with the occurrences probability of the two bigrams contained in the trigram.

Display scores: The score for each language is displayed and if the scores allow it, the program classifies the sentence to a language.

5.3.3 Program flow of the test set validator

To test the performance of our solution we build a block called “Test set validator”. The block takes as input 100 random sentences of each language and thus knows in which language each sentence is. Then for each sentence the language is calculated using the same method as in the previous paragraph. By comparing the calculated language to the actual language of the sentence the accuracy of the program can be calculated. Below a figure with the program flow of the “Test set validator” block is given.

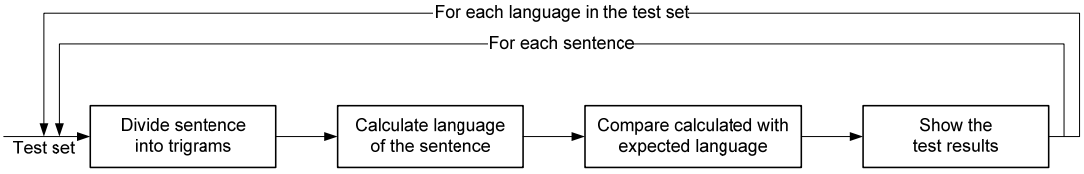

Figure 5: Program flow of the test set validator

5.4 Mapping of blocks to source code files

The actual implementation consists out of a number of files which will be discussed in this paragraph. In the root directory of the implementation the files “index.php”, “test_set.php” and “debug_scores..php” are found. The directory also contains the following directories: “data”, “hulp”, “images”.

index.php implements the language recognition module itself, the blocks “Language identifier” and “Language recognition algorithm” are contained in this file. This file is the core of the implementation and is loaded as the user browses to the application.

testset.php is developed to automate the testing of 100 random sentences of each language. This file implements both the blocks “Test set validator” and “Language recognition algorithm”.

debug_scores.php is used to render the scores of the individual trigrams of a sentence into a table.

The directory **data** contains several files (PHP and CSS stylesheets) containing functionality used in the three PHP files explained above. An example of this is the database handling and uniform look and feel of the interface.

hulp directory contains files with the implementation of the block “Learning set parser” for both the extraction of the learning set and the test set. Also separate parser applications are provided for the 5 non-European languages.

Finally the directory **images** contains images that are used to render the user interface.

6 Graphical User Interface

This chapter will describe the graphical user interface of the language identifier. The language identifier is the main user module. A user can input a sentence here, and the module will use the language identification algorithm to identify the language of the sentence. The startup screen looks as follows:

Figure 6: Startup and main screen of the application

The text field contains room the insert the sentence of which the language will be identified. The smoothing constant (as described in paragraph 4.2) can be set from 0 to 1 in steps of 0.05. It defaults to 0.85. Finally, the user can press the “Identify Language” button to let the language identification algorithm identify the language.

When the language identification is done, the results are displayed as follows:

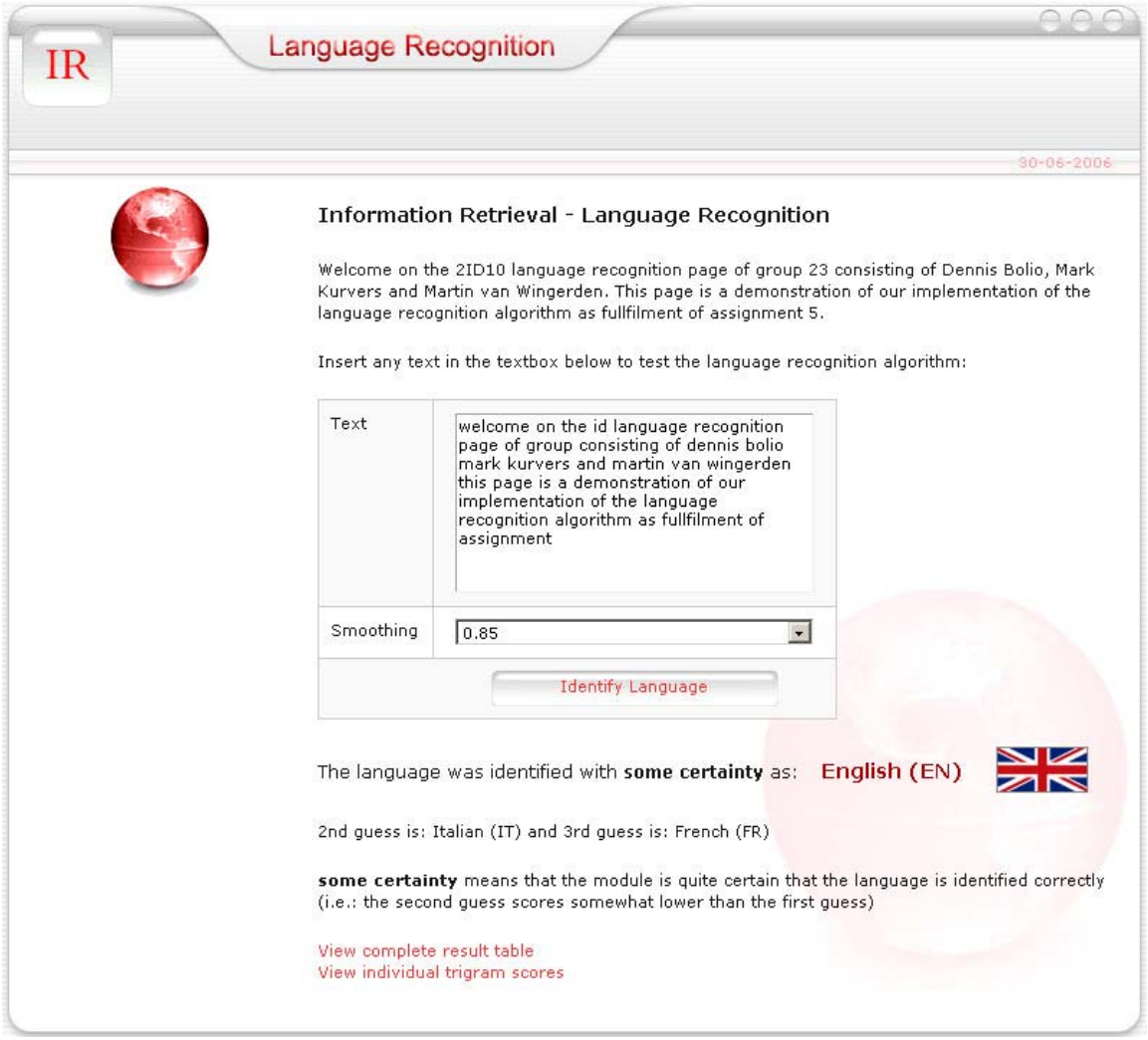

Figure 7: Screen after successful identification of the language

If the language could be identified, it shows the result in red along with the flag of the country. It also gives a classification of how certain the identifier is about the result. Currently, there are three certainty levels:

Certainty level	*Description*
Uncertainty	This means that the module is almost certain that the language is identified correctly (i.e.: the second guess scores much lower than the first guess)
Some certainty	This means that the module is quite certain that the language is identified correctly (i.e.: the second guess scores somewhat lower than the first guess)
Large certainty	This means that the identified language is not reliable (i.e.: the second guess score was close to the first guess score)

Table 1: Different identification certainty levels

The identifier uses the difference in scores of the first and second guess as a measure of how certain it is of the result. The thresholds for some certainty and large certainty are parameters of the language identifier. Apart from this, the identifier also displays the 2nd guess and 3rd guess of the language identification module.

If the user is interested, he or she can view all scores by clicking the "View complete result table" link in the bottom. This will display a list of all 21 known languages sorted by their score. The higher the score, the higher the probability that the sentence is in that language. The lowest possible score is -16, which is reached if none of the trigrams or bigrams of the sentence occur in that language. This happens for instance, with an unknown character set, such as Russian. The results of a Russian sentence are shown below:

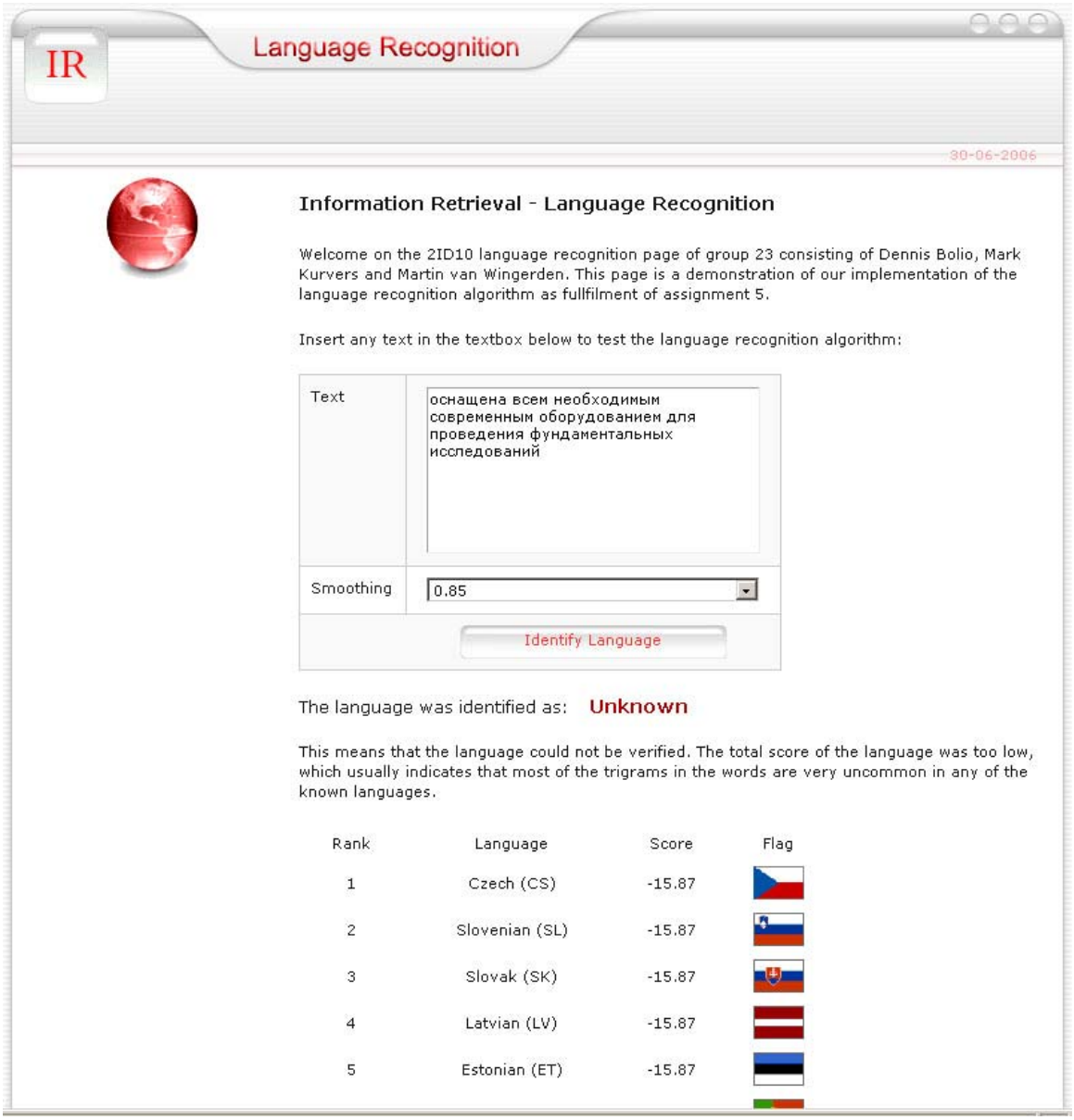

Figure 8: Screen after unsuccessful identification of the language

As can be seen, scores are very low, and if the best score gets below a certain threshold, the language of the sentence is identified as unknown.

Finally, it is possible for the user to view the scores of the individual trigrams of all languages. For displaying purpose, we have inputted the sentence “La vache qui rit”, which means “the laughing cow” and is a French brand of cheese. This sentence is incorrectly identified with large certainty as Irish Gaelic. When clicking on the “View individual trigram scores”, a new window opens displaying a table of all trigrams and the score of each trigram for each language.

Trigram	cs	da	de	el	en	es	et	fi	fr	ga	hu	it	lt	lv	mt	nl	pl	pt	sk	sl	sv
vac	-3.09	-3.75	-3.66	-4.93	-3.37	-3.02	-2.98	-3.04	-3.45	-2.8	-3.36	-3.43	-3.2	-3.06	-3.94	-2.79	-3.36	-3.24	-2.95	-3.08	-3.49
ach	-2.83	-4.2	-2.3	-4.53	-2.94	-3.06	-4.36	-4.46	-3.29	-1.63	-4.1	-3.3	-3.71	-4.08	-4.39	-2.79	-2.45	-3.41	-2.68	-3.88	-3.22
che	-2.95	-3.25	-1.96	-4.37	-2.54	-3.34	-3.43	-3.61	-2.96	-2.33	-3.52	-2.33	-4.11	-4.29	-3.84	-2.57	-2.95	-3.46	-2.95	-3.81	-2.99
qui	-3.88	-5.18	-4.58	-5.62	-3.12	-2.9	-3.6	-3.71	-2.54	-3.3	-5.55	-3.12	-3.67	-3.87	-3.88	-3.35	-5.53	-2.94	-3.89	-5.38	-5.37
rit	-3.11	-2.93	-2.65	-4.69	-2.52	-2.85	-2.55	-2.61	-2.53	-2.68	-3.27	-2.38	-2.53	-2.81	-2.47	-2.83	-3.75	-2.73	-3	-2.82	-2.73

Figure 9: Individual trigram score screen

This option can be used to identify problems with the identifier, or problems with the training data, as it shows why some trigram scores low in that language, or higher in another language.

7 Test results

This chapter contains the test results of our language identification application. First the method that was used for testing will be described, and then the results of the tests are described.

7.1 Test method used

For testing we’ve used the method as described in the assignment in [LEC4] and [ASS5]. We’ve created an application that randomly selects 100 sentences for each of the 21 languages from the EU constitution.

After the test set was created, we created an application that uses the language identification module to test each sentence and verify the results. After a complete run, the application displays the results for each language, showing the accuracy of the language identification module for that language. If a sentence was wrongly identified, it also displays what it was identified as.

7.2 Test results of the 21 European languages

The results of the test set application are displayed in the table below:

Language	*Score*
Czech (CS)	100 %
Danish (DA)	97 %
German (DE)	98 %
Greek (EL)	100 %
English (EN)	100 %
Spanish (ES)	99 %
Estonian (ET)	99 %
Finnish (FI)	99 %
French (FR)	98 %
Irish Gaelic (GA)	100 %
Hungarian (HU)	100 %
Italian (IT)	99 %
Lithuanian (LT)	100 %
Latvian (LV)	98 %
Maltese (MT)	99 %
Dutch (NL)	94 %
Polish (PL)	98 %
Portuguese (PT)	98 %
Slovak (SK)	98 %
Slovenian (SL)	96 %
Swedish (SV)	99 %
Total	**98,5 %**

Table 2: Test results of the 21 European languages

As can be seen from the table, only 31 sentences out of the 2100 were misidentified. While we thought this wasn't a bad score, a closer inspection of the results revealed that many of the languages that were "misidentified" were actually correctly identified, but were wrongly taken into the test set. The problem is that many of the XML files of [OPUS] contain a few lines that are in all languages. Since the test sentences are taken at random, there is a chance that a few of these sentences ended up in the test set for some languages.

When these sentences are ignored, only the following sentences show up that are really misidentified:

Sentence	*Language*	*Identified as*
die sozialpartner und der autonome soziale dialog	German (DE)	Italian (IT)
la collecte le stockage le traitement l analyse et l échange d informations pertinentes	French (FR)	English (EN)
o parlamento europeu é plenamente informado	Portuguese (PT)	Spanish (ES)

Table 3: Mismatch sentences during testing of European languages

So the real accuracy of our language identification module is 99.9 % (3 out of 2100 sentences misidentified).

7.3 Test results of 5 non-European languages

Part of the assignment also is to test 100 sentences of non EU languages. Suggested was that these languages are taken from the KDE translations, which also available at [OPUS]. We chose the following five languages:

- Norwegian (NN)
- Russian (RU)
- Turk (TR)
- Xhosa (XH)
- Zulu (ZU)

The results are a bit disappointing:

Language	*Score*
Norwegian (NN)	1 %
Russian (RU)	97 %
Turk (TR)	59 %
Xhosa (XH)	0 %
Zulu (ZU)	0 %
Total	**52,3 %**

Table 4: Test results of 5 non-European languages

The Russian language is almost completely recognized as being an unknown language. This is not very surprising since the Russian character set is much different from the EU character sets. Turk is recognized as unknown almost 60% of the time, but the other languages are never recognized as being unknown.

However, many of the sentences that were identified as being of a known language were identified with uncertainty. When we also count identification of languages with uncertainty as being correct (i.e. count them as unknown), we get much higher scores:

Language	*Score*
Norwegian (NN)	71 %
Russian (RU)	99 %
Turk (TR)	82 %
Xhosa (XH)	74 %
Zulu (ZU)	79 %
Total	**81 %**

Table 5: Test results of 5 non-European languages, second version

Still, this is nowhere near the accuracy of known languages but quite a bit better than the earlier result. A cause for the bad scores could be that the threshold below which a language is identified as unknown, is set too low. A solution could be to raise the threshold to a higher value. The downside of this is of course that it will also cause some sentences that are now correctly identified, to be identified as unknown.

8 Conclusions

We conclude this document by stating some final remarks and conclusions about the project.

The cooperation in the group was done without any problems. There was hardly any division of work during the project because we worked most of the time at the university together at the assignment.

About optimizing the smoothing parameter we can conclude that value of 0.85 works well in most cases. This parameter is however easy adjustable from the graphical user interface.

In general we can conclude that the application is in general good in identifying the 21 European languages. This we find quite remarkable because we used relative little data as learning set: about 3000 records, 300 kilobyte of data in the bigram table and 28000 records, 2800 kilobyte of data in the trigram table.

Identifying non-European languages seems more difficult for the program but this is due to the constant that determines which scores yield the language identification “Unknown”. Changing this constant can increase the performance of the program in identifying the non-European languages as “Unknown” but can at the same time also reduce the good performance of identifying the European languages.

The final conclusion we can draw is that the assignment is carried out successfully. The group work was carried out without problems or differences. The result of the assignment is an application that performance quite well.

9 References

[ASS5] Wessel Kraaij, *Assignment 5: Language identification is an important basic component for multilingual search engines*, Assignment, 18-04-2006, http://www.win.tue.nl/~laroyo/2ID10/2005-2006/Assignments/assignment5.html

[DUNN] Ted Dunning, *Statistical Identification of Language*, 10-03-1994

[LEC4] Wessel Kraaij, *Language Modelling*, 4th lecture of course 2ID10, 18-04-2006, http://www.win.tue.nl/~laroyo/2ID10/2005-2006/Slides/college-tue-2006-kraaij.pdf

[OPUS] Jörg Tiedemann, Lars Nygaard, *OPUS - an open source parallel corpus*, http://logos.uio.no/opus/

[ROSE] Ronald Rosenfeld, *Two Decades of Statistical Language Modeling: Where Do We Go From Here?*, Proceedings of the IEEE, 88(8), 2000