Language Identifier

1 Introduction

The problem statement: Implement (without using any specialized libraries) a Language Identifier in Java that can identify the language of a document.

2 Approach

2.1 Supported Languages

My approach is to build a multi-class classifier, where the classes are the following languages:

- English (UK)
- German
- Spanish
- French
- Italian
- Latin
- Lithuanian
- Hungarian
- Danish
- Bulgarian

Corpus used: `http://corpora.uni-leipzig.de/`

Note: The Identifier supports adding/removing languages without any change in the code.

2.2 Classification Algorithm Used

In my implementation of the Language Identifier, I have used the *multinomial Naive Bayes text classification* algorithm (on words) with 10 classes. The most likely class is then returned as the predicted language for the document. *Reference*:

`http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html`

Smoothing: Add-one smoothing has been used to deal with words that are not present in the training set.

2.3 Key Ideas Behind The Choice Of Algorithm

Intuition: A sentence is more likely to be of language L if most of the words in the sentence belong to L.

Availability of frequencies: The frequencies of words in the corpus of each language are readily available in text files.

Huge training set: Naive Bayes gives high accuracy due to the hugeness of the training data.

Nature of words: Most of the words are unique to a particular language or a language-group. In many cases, if a word is shared by two/more languages, a slight variation in spelling is present. Hence, the probability that a word is in language L is highest if it actually belongs to L or a similar language.

Nature of text found commonly: Mixing up of languages within a single text-document is very rare. Hence, probability of getting the correct language is very high.

Speed: Naive Bayes gives a fast method of classification as compared to other algorithms as training the classifier is very fast.

2.4 Applicability of Approach

Suppose the document contains only the words "Esta es mi casa" and, say, we have the following data from a corpus (word(frequency)):

Spanish: Esta(54), es(930), mi(67), casa(47)
Italian: Esta(0), es(2), mi(182), casa(122)

Let the total of frequencies of all words in Spanish be 10000 and let Italian have the same number. Further, for the purpose of demonstration, let the size of the vocabulary of Spanish and Italian combined be 1000. *Since the products of the probabilities will be very small*, we take log of the probabilities to prevent underflow. We denote log probabilities by 'P'.
The prior probabilities are:

$P(Spanish) = \log 0.5 = -0.301029996$
$P(Italian) = \log 0.5 = -0.301029996$

Using add-one smoothing,

$P(Esta|Spanish) = \log\left(\frac{54+1}{10000+1000}\right) = -2.301029996$

$P(es|Spanish) = \log\left(\frac{930+1}{10000+1000}\right) = -1.072475659$

$P(mi|Spanish) = \log\left(\frac{67+1}{10000+1000}\right) = -2.209011525$

$P(casa|Spanish) = \log\left(\frac{47+1}{10000+1000}\right) = -2.360214787$

which gives

$P_S = P(Spanish) + P(Esta|Spanish) + P(es|Spanish) + P(mi|Spanish) + P(casa|Spanish)$
$= -8.243761963$

and,

$P(Esta|Italian) = \log\frac{0+1}{10000+1000} = -4.041393119$

$P(es|Italian) = \log\frac{2+1}{10000+1000} = -3.564271865$

$P(mi|Italian) = \log\frac{182+1}{10000+1000} = -1.778941586$

$P(casa|Italian) = \log\frac{122+1}{10000+1000} = -1.951487581$

which gives

$P_I = P(Italian) + P(Esta|Italian) + P(es|Italian) + P(mi|Italian) + P(casa|Italian)$
$= -11.637124147$

Clearly, $P_S > P_I$ and the classifier will return *Spanish* as the predicted language. The actual data in the corpus is much larger and more accurate. Hence, this gives a much larger difference in the log probabilities of different languages when used with actual statistical data.

3 Implementation Details

3.1 Training The Classifier

The list of names of languages and their corresponding word-resource file-names are stored in the file, languages.txt. The word-resource files are kept in the resources directory. For every language, the program reads the corresponding word-file taken from the downloaded corpus. Each line is split, (word,

word-id) is added to the language's word → word-id HashMap and (word, frequency) is added to the language's word → frequency HashMap. At the same time, all words are added to the universal Set (vocabulary) and the count of words in each language is maintained. To make the classifier memory-efficient, only the words from line 101 to 5100 in every file are trained upon. Training starts from 101 because the first 100 words are symbols that are *equally common in all languages*. Only the top 5000 words of a word-file are trained upon because the top 5000 are also the 5000 most frequent words for that language. *Additionally*, the user is allowed to specify a custom training size-per-language as a command-line argument.

3.2 Pre-processing The Test Document

The document is read into a String and the String is stripped of the punctuation marks: comma, full-stop, semicolon, colon, exclamation mark, question mark, parentheses, ampersand, double-quotes and forward-slash. These are some common punctuation marks present in text and have high frequencies for *all* languages. Hence, they need not be considered for the purpose of classification. The String is then split into tokens.

3.3 Classifying The Document

Multinomial Naive Bayes is applied on the array containing the tokens in the String. The language having the highest probability is returned as the predicted language for the document.

4 Running The Program

To run from source,

1. Compile the source using the command, *javac *.java*
2. Run the Identifier using the command, *java Identifier*

To run from the executable jar file, use the command
java -jar Identifier.jar