International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 03, March-2015

Multilingual Text Classification

Sonam Mittal
Computer Science Dept.
B K Birla Institute of Engineering & Technology
Pilani, Rajasthan, India

Prof. Praveen Dhyani
Executive Director
Banasthali Vidyapith
Jaipur, India

***Abstract* - Identifying the language used for a document will typically be the first step to most of the Natural Language Processing tasks. Among the wide variety of language identification methods discussed in the literature, the ones employing the Canvar and Trenkle (1994) approach to text categorization based on character n-gram frequencies have been particularly successful. Multilingual Text Classification using N-gram techniques seems to have produced very interesting results in the field of text categorization not only for the languages like English and French but equally good for more difficult to classify languages like Spanish, Italian, German and Russian.**

Keywords— Multilingual Text, N-gram, tf-idf, frequency, similarity, classification, prediction, classifier

1. INTRODUCTION

Automated text categorization is a supervised learning task, defined as assigning category labels to new documents based on the likelihood suggested by a set of labeled documents. Classifying the language of the documents requires several essential steps like preprocessing the text to obtain terms, identifying important terms and a classifier (in this case Naïve-Bayes classifier is used) [2].

Language classification is an important task for today's World Wide Web where an increasing number of documents are in languages other than English. Language classification finds use in area like search engine indexing, text mining, spam filtering and other applications that apply language specific algorithms. Such classification is a key step in processing a large document streams and is a data intensive task [3].

2. METHODOLOGY

The proposed system is able to predict the language of an incoming document. The languages taken for this system are English, Spanish, and Italian taken from [4]. The system is first trained (with 20% corpus) and is then tested for 80% of the corpus. The efficiency of the system comes out to be 99%. The documents are in XML file format. The motive behind choosing Spanish and Italian was their extreme closeness in words and hence it makes the classification task even more challenging. The whole task of classification consists of four phases:

- Document Preprocessing
- TF-IDF Analysis
- Training the Model
- Testing the Model

First we will see each of these phases in terms of the functions performed and then we will present the approach to execute the code.

2.1 Document Preprocessing

Preprocessing of the document is a key step in almost every kind of document classification system whether it be content based classification, topic based classification or syntactic type of classification. Preprocessing of the documents to get rid of the stop-words itself makes the later steps easier and more efficient. In this phase, we use *regular expressions* to remove the noise, which is special symbols, single characters, numbers from 0 to 9, special characters, backslashes, and multiple white spaces and tabs.

2.2 TF-IDF (Term Frequency – Inverse Document Frequency) Analyser

In this phase, the document level profiles (the term for each document and their frequency) are used to calculate the TF-IDF value for identified terms. The TF-IDF value represents how important a term is to a document in a collection or corpus (Wikipedia). It can be considered as a weighting factor for the terms in a document with respect to other terms of the same document. TF-IDF is the product of two statistics, term frequency and the inverse document frequency. Term frequency is defined as the number of times the term appears in the document. In order to have the normalized document frequency, we use

$$tf = \frac{term\ frequency}{highest\ term\ frequency}$$

The inverse document frequency is a measure of whether the term is common or rare across all the documents. It is obtained by:

$$idf = log\frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ the\ term}$$

Finally, the tf-idf is calculated by:

TF-IDF = tf × idf

The result of this phase is present in the sub folder TFIDF Analyser results under the Profiles folder. The value of TF-IDF is used as a classifier.

Based on the TF-IDF values of the term, we select a set of say 25 terms with highest frequency (or TF-IDF value) for each category. These are also called the keywords for the category. This collection of the highest frequency terms with their tf-idf values is called dictionary for each category. It is used in the phase of testing where the category of the test document is predicted.

2.3 Training the Model

20% of the collection is used for training the model. The corpus used for training is stored in the folder called training set with a sub folder for each category. For each of the documents used for training, the given XML is fetched and generate its profile explained in the previous two sections (pre-processing, tf-idf).

A file called probabilitymap.csv is generated having three rows (one for each category) and the number of columns is the product of number of categories and number of keywords (if 25 keywords are given, there will be 25*3 columns in the csv file). The calculation of the probabilities is explained in the next section. In the probabilitymap.csv, each cell denotes the probability of the keyword to belong to the particular category (category is extracted from the row number 0: It, 1: En, 2:Es). This file is used in the testing phase for predicting the value of the target document.

2.4 Testing or Predicting

In order to predict the category of the test file, the first step is to pre-process it in the same way as training documents were done to remove noise and create its document level profile. For each term of the test document, first we see if it exists in the three dictionaries of the categories. If it exists then we calculate the possibility of it to belong to one of the category. The probability of each term of the test file to belong to one of the language categories is given as follows: (Bayes Rule)

$$Pr(Category|Term) = \frac{[Pr(Term|Category) * Pr(Category)]}{Pr(Term)}$$

Pr(Term) in the corpus is static. Once we obtain the probability of each term, we can calculate the probability of the whole document to belong to each category. Let w_1, w_2,…w_n be the probability of n terms in the test document. Then the probability of the whole document is nothing but the log of the product of all these individual probabilities.

The result of this phase is shown on the console. The test document should be in XML format and should contain the actual language category of the document. The test file name should be of the form es.xml, en.xml, it.xml. These types of file names will help us to compare their original language category to the predicted one.

2.5 Basic Architecture

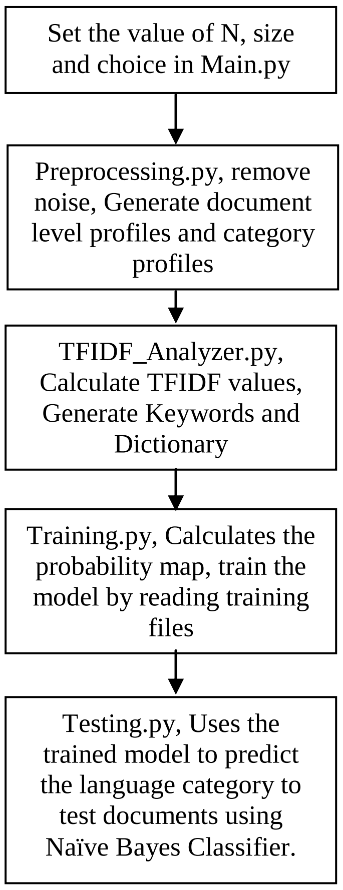
Set the value of N, size and choice in Main.py
Preprocessing.py, remove noise, Generate document level profiles and category profiles
TFIDF_Analyzer.py, Calculate TFIDF values, Generate Keywords and Dictionary
Training.py, Calculates the probability map, train the model by reading training files
Testing.py, Uses the trained model to predict the language category to test documents using Naïve Bayes Classifier.

Fig 1: Architecture of Language Classification System

3. ORGANIZATION AND EXECUTION OF THE PROGRAM

3.1 Organization

The code for this multilingual classifier is organized into 4 different files (just as the phases described in the previous section). The file preprocessing.py is responsible for the pre-processing part of the program. It has modules or functions for removing noise, calculating n-grams (if value of n>0) and to generate the document level profile as explained in the preprocessing section. The file TFIDF analyzer does the part for calculating the TF-IDF values for the pre-processed files. It has modules to count the terms and their respective frequencies. Training.py reads the files from each category sub-folder (en, es, it) under the parent folder called Training set. It calls the modules from the folder Testing set. The format of test-file names are already described in the previous section.

3.2 Execution

The given program can be run in two modes:

- Normal Mode: In this mode, there is only one fixed value for N-grams (the variable N) and size of keywords dictionary (size) which is to be initialized in the Main.py. This mode can be selected by setting the value of variable choice to 0.
- Mega Run Mode: When choice variable is set to 1, the code runs in mega run mode. In this mode, there is a list of N-gram values and a list of sizes for the dictionary i.e. the code executes with various different configurations and hence is very time consuming (with the given values, it takes 10 hours to complete).
- System Requirements : Linux (Ubuntu 13.04 is used)
- Packages: numpy, pyngram
- The documents should be tested/predicted are to be placed in the folder Testing Set, should be XML files, should have two initial characters of the original language category just before the .XML part of the file name. Some examples of test files are: hello es.xml, irtest it.xml etc.

3.3 Observations

The efficiency of the system is 65% when no n-gram is applied and the size of keywords is 25. With 4 n-grams the efficiency increases to 96% with the same size of keywords. 10 fold cross validation can give better results. Logically, the larger the size of keyword list better is the efficiency. Value of n in n-grams can significantly change the performance of the system but as per the observations, 3 and 4 are the best values to be considered.

REFERENCES

1. N Gram Based Text Categorization – William B. Canvar, John M. Trenkle, 1994.
2. Is Naïve Bayes a Good Classifier for Document Classification – S.L. Ting, W.H. Ip, Albert H.C. Tsang, International Journal of Software Engineering & Its Applications, Vol. 5 No. 3, July 2011.
3. Multilingual Text Categorization using character N gram – Suzuki M., Yamagishi N. , Yi Ching Tsai, Hirasawa S., Soft Computing in Industrial Applications, 2008.
4. http://optima.jrc.it/Acquis/JRC-Acquis 3.0/corpus/