BITS – F464 MACHINE LEARNING

ASSIGNMENT REPORT

ON

IMPLEMENTATION OF

NAÏVE BAYES CLASSIFIER ALGORITHM

2012A7PS017H – A DHANVI SRIRAM

2012A7PS803H – SHUBHAM SINGHAL

2012A7PS118H - K SAI KRISHNA

2012A7PS066H – CH ROHIT KUMAR

Table of Contents

1. CODE DESCRIPTION

This algorithm has been implemented using 'Java' programming language and Net-Beans IDE. The following are the classes used and their respective objects….

CLASSES :

1. **NaiveBayes.java** : This is our main driver class which has the over-all description of the program flow. This class calls the necessary functions, implemented in their respective classes. The program execution must end with this class only.

2. **TrainingData.java** : This class contains the instance variables and methods to read the training data from the file and also initialize the required '*Vocabulary*' for the given training data set.

This class contains a constructor which initializes the '*folder*' variable which points to the folder in which the training documents are present. Also this class contains the '*initVocab()*' method which returns the vocabulary (set of all distinct words for the given set of documents) as an array list of String objects.

Apart from reading the data and creating the vocabulary, the *initVocab()* method of this class also assigns the 'target value' to each file. This , it does in the following way :

There is declared an array of String objects- '*targetValues*'---

String[] targetValue = {"like","dislike","like","dislike","dislike","like"};

Also, there is another *HashMap*, which associates every File object with a String object.

datahm = new HashMap<File,String>();

datahm.put(f,targetValue[(i)%6]);

Thus, as we can see, the *initVocab()* function associates every file with a String, which is nothing but its target classification.

The '[(i)%6]' just ensures that the files are given target classifications in a cyclical order. Thus the ordering of files in the document becomes important in this implementation. We should take care that the files which we '*like*' are placed in the document in such a way that the index assigned to them (in an increasing order) leaves a remainder of either 0,1 or 5 with 6. Again, this implementation changes with changing the '*targetValue*' array.

3. **NaiveBayesLearner.java** : This is the class which contains the actual implementation of the learning algorithm. The following is the list of instance variables and their purposes :

1. *ArrayList<String> Vocabulary = new ArrayList<String>()* : contains the vocabulary returned by the initVocab() function.

2. *String[] V = new String[2]* : contains the target values "like" and "dislike"

3. *ArrayList<File> likefolder* : list of files declared as "like".

4. *ArrayList<File> dislikefolder* : list of files declared as "dislike"

5. *File likeFile* : collection of all words which occur in the likefolder

6. *File dislikeFile* : collection of all words which occur in the likefolder.

7. *File file = null* : to read any file object.

8. *FileReader fr = null* : to read any file object.

9. *double like_prob,dislike_prob* : absolute probablilities of liking or disliking any file, which in our case are 0.5 and 0.5 respectively because the number of files assigned the targetValue "like" and the number of files assigned the targetValue "dislike" are same.

10.*BufferedReader br = null* : to read any file object.

11.*HashMap<String,Double> likehm* : to associate each word occurring in the 'likeFile' with a probability.

12.*HashMap<String,Double> dislikehm* : to associate each word occurring in the 'dislikeFile' with a probability.

The constructor of this class initializes these instance variables. The most important function of this class is the *learn()* function which takes the set of files as the input, and initializes the '*likehm*' and '*dislikehm*' objects. The flow of the code is on the same lines as that of the algorithm given in the textbook. This function is helped by some 'utility' functions, which are : 1.*mergeFiles()* and *wordCount()*.

mergeFiles() basically takes a set of Files and a destination file as the input and merges the set of files into the destination file. The function header is given below :

private void mergeFiles(ArrayList<File> folder, File mergedFile);

The working code for this function has been obtained from the following url : http://www.programcreek.com/2012/09/merge-files-in-java/

wordCount() function is very simple. It returns the total number of words encountered in a file. The function header is given below :

private int wordCount(File file)throws FileNotFoundException

The successful execution of all the above mentioned functions completes the learning process of the algorithm and the control returns to the '*main*' function. The main function the calls the *'naiveBayesClassifier()'* function with the input as the document to be tested. This function finally classifies the input document as 'like' or 'dislike'. The function header is given below :

public String naiveBayesClassifier(File testDoc)throws FileNotFoundException

This function declares a String variable ‘prediction’ and initializes it to “dislike”.

Another variable called ‘*wordList*’ is declared which is an array list of all the words which are contained in the list ‘*Vocabulary’*. Two variables ‘*tlike*’ and ‘*tdislike*’ are declared to store the probabilities of the final target classifications.

tlike = log10(like_prob(like_pie(wordList)));*

tdislike = log10(dislike_prob(dislike_pie(wordList)));*

Here, the *like_pie()* and *dislike_pie()* are simple utility functions which calculate the product term in the following expression :

$$v_{NB} = argmax\ P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

The logarithms appear in the above equations to handle probabilities in the order of very high ‘negative’ powers of 10. (they are not found in the original algorithm).

Finally, if ‘*tlike*’ comes out to be greater than ‘*tdislike*’, then the prediction is changed to ‘like’, else the default prediction value, which is ‘dislike’ is returned.

Also note the last lines of the main function.

PrintWriter like_writer = new PrintWriter(new File("C:\\Users\\Sriram\\Documents\\NetBeansProjects\\NaiveBayes\\likefile.txt"));

like_writer.print("");

like_writer.close();

These lines simply ensure that the likefile.txt and the dislikefile.txt are completely cleaned but not deleted, and are made ready for the next execution.

2. EXPERIMENTATION

We have considered two kinds of documents for the training & testing of the algorithm. Consider documents on '**cricket**', classified as '*like*' and documents on '**international politics**', classified as '*dislike*'. As required by the algorithm, the cricket-related documents are stored in the first, third and the sixth files , the politics related documents have been stored in the remaining files. The document to be tested is stored in a file named '*testfile*'. The following urls have been referred to obtain the data.

1. http://mea.gov.in/articles-in-indian-media.htm?55/Articles_in_Indian_Media for intl politics.

2. http://www.abcofcricket.com/Article_Library/art50/art59/art59.htm and http://www.britannica.com/EBchecked/topic/142911/cricket for cricket.

The algorithm has been trained and tested for both the documents and the results were consistently correct.

The dataset used for training and the testfile are included in the zip file.

The program returns the following outputs :

For a 'cricket' document as input :

the classification for the new document is like(cricket).

For an 'international politics' document as input :

the classification for the new document is dislike(international politics)

3. ERRORS AND TROUBLE-SHOOTING

1. One possible cause of error (misprediction) is improper training, which can be rectified by providing more number of documents.

2. Another cause of error can be improper file reading, especially the test document. In such cases we must delete the previous testfile and create a new testfile in the same folder and run the program again.

4. CONCLUSIONS

The program has been tested for various documents and snippets from many documents, relating to '**cricket**' and '**International politics**'. In every test run, the program correctly classified the test document, Except for cases where second type error occurred.(refer section 3).

In addition, the program has also been trained and tested for '**Classification of poems**' as being a '**happy poem**' or a '**sad poem**' , based on the words occurring in it. In this case, the program showed misclassifications whenever the test poem consisted of an extremely large quantity of new words, which is an effect of first kind of error.(refer section 3). The data sources of the poems are given in section 5 (5,6).

Thus we conclude that with proper training and efficient file handling mechanism, this implementation of the **Naïve Bayes Classifier** can produce appreciably accurate results.

5. BIBILOGRAPHY

1. http://www.programcreek.com/2012/09/merge-files-in-java/

2. http://mea.gov.in/articles-in-indian-media.htm?55/Articles_in_Indian_Media

3. http://www.abcofcricket.com/Article_Library/art50/art59/art59.htm

4. http://www.britannica.com/EBchecked/topic/142911/cricket

5. http://100.best-poems.net/happiness.html

6. http://www.scrapbook.com/poems/cat/67.html