



BAYESIAN NAÏVE BAYES

Artificial Intelligence DNET4



NOVEMBER 11, 2015

REN KAI TAM
R00095982

Contents

Bayesian Naïve Bayes Classification Introduction.....	2
High level overview of the process	2
BayesianSample Class	2
TestingModel Class	2
Technical Code Description (Methods).....	3
BayesianSample Class	3
LoadingFilesFromDirectory	3
CaptureFileContents	3
AddWord.....	3
CalculateProbabilityOfEachWord	4
CreateModel	4
TestingModel Class	4
CaptureFileContents	4
CalculatePVOfDocument.....	5
DetermineThePosOrNegReview	5
PopulateHashMap.....	5
SplitString.....	5
Main	6
Result	6
Improve Word Parsing	6
Large IMDB.....	6
Small IMDB.....	6
Optimization	7

You must include a signed declaration asserting the originality of the work, e.g “I, the undersigned, declare the following to be 100% original and my own work, except where indicated otherwise.

Signed: _____”.

A handwritten signature in black ink, appearing to be 'K. V. S.', written over a horizontal line.

Bayesian Naïve Bayes Classification Introduction

This assignment is using Bayesian to calculate the frequency to each word within either positive or negative datasets and use the frequency of each word to calculate the probability of the word have a positive or negative meaning.

High level overview of the process

First there are 2 classes in the Java project; BayesianSample.class and TestingModel.class. There are also 2 predetermined dataset that marked as positive and negative folder that contained 2001 IMDB review of random movies. Below explain the process in each class and also explaining the flow process of how the model is created and tested using both classes.

BayesianSample Class

This class is responsible to create positive and negative model, within these models it contain the word and its probability value. First a collection of files will be load into a file array (either positive or negative directory). Then a method will read each word within a file into a Set, simple parsing will be done before adding each word into the Set. The reason of using set is to get a unique word within an IMDB review, eliminate duplicated words in the process.

In the positive dataset all the word will be loaded into a positive HashMap and also into the Set which is the vocabulary (vocabulary contain all positive and negative words). During the process of adding word if the same word occurred more than once, instead of discarding the word HashMap is set to increase the frequency of the word by 1.

After the loading process is finished, the calculation of the probability for each word will begin and this probability value will be replace the frequency number in the HashMap. Then within a for-loop to loop through the HashMap to write into a file (either positive or negative text file). This text file is the model and will be used in the test data set.

TestingModel Class

Once the model is created, then use the unseen test data to run against the positive and negative model. To do this, first there are 2 folders within the test day, positive and negative. One contain positive IBDM review and the other is negative IMDB review.

First load the all the positive file into a File array, then looping through each word in each document and sum up the probability value if it exists in the positive model. Then run this again to the negative model to sum up the probability of the negative value if the word exists in the negative model. The final steps is to compare the 2 probability values, positive and negative to see which have the greater value. The one with the greater value is consider the categories.

Technical Code Description (Methods)

BayesianSample Class

LoadingFilesFromDirectory

```
private static void loadFilesFromDirectory(String dirName, Set<String> vocabulary) {

    String path = ".\\" + S_IMDB + "\\" + dirName + "\\";

    String fileName;
    File folder = new File(path);

    // Creates an array of File from the specified path and directory
    File[] listOfFiles = folder.listFiles();

    // Iterates through all the files in the directory
    for (int i = 0; i < listOfFiles.length; i++)
    {
        if (listOfFiles[i].isFile())
        {
            fileName = listOfFiles[i].getName();

            // Open the specified file and adds it's contents into the set
            captureFileContents(path+fileName, vocabulary);
        }
    }
    System.out.println(dirName + " dir, Number of files" + listOfFiles.length);
}
```

Using for-loop to loop through to check is each element within the file array is a file. If it is, get the file name and pass the file name and the directory into the captureFileContents method and after that it prints out the number of files within the directory.

CaptureFileContents

```
private static void captureFileContents(String fileName, Set<String> vocabulary) {

    try {
        Scanner reader = new Scanner(new File(fileName));

        // Read each string from the file and add to the set
        while (reader.hasNext())
        {
            String word = reader.next().trim().toLowerCase();
            //System.out.println(word);

            if (simpleParsing(word, fileName))
                addWord(word, fileName);

            totalNumberOfWords++;
        }
        reader.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

I have modify the method a little. This method using “Scanner” class takes in the files name and read each word within the file. This word is then trim and change to lower case this prevent the adding same word that in different form (i.e. Set is case sensitive).

If the word passed the simple parsing it will be added to the vocabulary and add to positive or

negative HashMap, then increasing word count for each word. Close the scanner after it’s finished. All the code is wrapped in a try catch block.

AddWord

```
private static void addWord(String word, String fileName){
    if(vocabulary.add(word))
    {
        if(fileName.contains("pos"))
            positive.put(word, (double) 1);

        if(fileName.contains("neg"))
            negative.put(word, (double) 1);
    }
    else if(!(vocabulary.add(word)))
    {
        if(fileName.contains("pos"))
            positive.put(word, (positive.containsKey(word) ? positive.get(word) : 0)+1);

        if(fileName.contains("neg"))
            negative.put(word, (negative.containsKey(word) ? negative.get(word) : 0)+1);
    }
}
```

This method passed in a word and the file name and path. If vocabulary successfully added the word that means that word is a new word so it will be added into the positive or negative HashMap depending on which directory it’s in. If

vocabulary adding new word return a false it means that the words already exists, then increase the frequency value in the HashMap by 1.

CalculateProbabilityOfEachWord

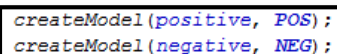
```
private static void calculateProbabilityOfEachWord(HashMap<String, Double> posOrNeg) {  
    for(String word : posOrNeg.keySet())  
    {  
        double pV = (Double.parseDouble(posOrNeg.get(word).toString())+1)/((totalNumberOfWords) + 2);  
        double logOfPv = Math.log(pV);  
        posOrNeg.put(word, logOfPv);  
    }  
}
```

This method calculate probability value of each word, by using the naïve Bayes Probability $P(\text{word} \mid \text{class}) = \log(\text{count}(\text{word} \mid \text{class}) / \text{count}(\text{total Words}) + 2)$. Then it remove the frequency and add the probability value to the HashMap. This mathematical function is wrapped within a enhance for-loop to get the key of the HashMap.

CreateModel

```
private static void createModel(HashMap<String, Double> posOrNeg, String fileName) {  
    try {  
        File file = new File(fileName);  
  
        if (!file.exists())  
            file.createNewFile();  
  
        FileWriter fw = new FileWriter(file.getAbsolutePath());  
        BufferedWriter bw = new BufferedWriter(fw);  
  
        for(String word : posOrNeg.keySet())  
        {  
            bw.write(word.toString() + " " + posOrNeg.get(word).toString());  
            bw.newLine();  
        }  
        bw.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Create model method writing each HashMap (positive and negative) into each separate text file. Image on the bottom right corner, is how the method is called. Final string “POS” and “NEG” is the name of the model that will be created during the process here in this method.



First check to see if the file “POS”

exists in the current directory, if it’s not then create the file. The using FileWriter class and BufferedWriter class to write to the file. Using enhance for-loop to loop through the HashMap, word.toString() get the key and posOrNeg.get(word).toString() this get the frequency of the word. And bw.newLine() put the next loop into a new line. And close the buffered writer it’s finished.

TestingModel Class

First loading use the method to load the file into the array using the same method in BayesianSample class (loadFileFromDirectory). Within the method it use the method CaptureFileContents.

CaptureFileContents

```
private void captureFileContents(String fileName) {  
    int totalNumberOfWords = 0;  
    double posPV = 0, negPV = 0;  
    try {  
        Scanner reader = new Scanner(new File(fileName));  
  
        // Read each string from the file and add to the set  
        while (reader.hasNext()) {  
            String word = reader.next().trim().toLowerCase();  
  
            posPV += calculatePVOFDocument(word, positive);  
            negPV += calculatePVOFDocument(word, negative);  
  
            totalNumberOfWords++;  
        }  
        reader.close();  
    } catch (FileNotFoundException e) {  
        e.printStackTrace();  
    }  
    determineThePosOrNegReview(posPV, negPV);  
}
```

This method uses Scanner class to get each word and within the while loop, the word is compared with the HashMap, if the word is found then the probability value add together.

The word is trim to no spaces and also change it to lowercase for the whole word before doing the addition.

Then both positive and negative value is compared using the method call “determineThePosOrNegReview”

CalculatePVOfDocument

```
private double calculatePVOfDocument(String word, HashMap<String, Double> posOrNeg) {  
    if (posOrNeg.containsKey(word))  
        return Double.parseDouble(posOrNeg.get(word).toString());  
    return 0;  
}
```

If the contained in the (i.e. positive HashMap) if it's calculating the positive probability of the word.

This method have a return value which returns a double. Using parse double to get the numeric value from the HashMap instead of string value, it parse the string to Double value.

DetermineThePosOrNegReview

```
private void determineThePosOrNegReview(double posPV, double negPV) {  
    if (posPV > negPV)  
        numberOfPosFile++;  
    else if (negPV > posPV)  
        numberOfNegFile++;  
}
```

This method compared the 2 values increase the values. If the positive probability of a file is higher than the negative probability then increase the

number of positive file. Vice versa.

PopulateHashMap

```
private void populateHashMap(String fileName){  
    File file = new File(fileName);  
    FileReader fr;  
  
    try{  
        fr = new FileReader(file);  
        BufferedReader br = new BufferedReader(fr);  
        String line;  
  
        while ((line = br.readLine()) != null)  
            splitString(line, fileName);  
  
        br.close();  
        System.out.println("Done Populating hashmap " + fileName);  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

This method read the model text file, using bufferedReader to read each line of the file then using while loop then using splitString method to separate the strings.

SplitString

```
private void splitString(String line, String fileName) {  
    String keyValue[] = line.split(" ");  
  
    if (keyValue.length == 2)  
        if (fileName.equalsIgnoreCase(POS))  
            positive.put(keyValue[0], Double.parseDouble(keyValue[1]));  
        else if (fileName.equalsIgnoreCase(NEG))  
            negative.put(keyValue[0], Double.parseDouble(keyValue[1]));  
        else  
            System.out.println("No such file Exists in the directory");  
    else  
        System.out.println("Error length");  
}
```

This method takes in the line and the file name. The file name is allow to identify which HashMap to load the model into, either to positive or negative.

Main

```
public TestingModel() {
    //Load Model
    populateHashMap(POS);
    populateHashMap(NEG);

    loadFilesFromDirectory("pos");
    System.out.println("Pos - Number of Pos File " + numberOfPosFile + " " + ((numberOfPosFile*100)/1000) + "%");
    System.out.println("Pos - Number of Neg File " + numberOfNegFile + " " + ((numberOfNegFile*100)/1000) + "%\n\n");

    numberOfNegFile = 0;
    numberOfPosFile = 0;
    loadFilesFromDirectory("neg");
    System.out.println("Neg - Number of Pos File " + numberOfPosFile + " " + ((numberOfPosFile*100)/1000) + "%");
    System.out.println("Neg - Number of Neg File " + numberOfNegFile + " " + ((numberOfNegFile*100)/1000) + "%");
}

public static void main(String[] args) {
    // TODO Auto-generated method stub
    new TestingModel();
}
```

In the constructor first it call the populateHashMap method to load the positive and negative model.

The load the positive unseen data in and calculate the

probability of all the positive data. Then reset the number of files for the negative and positive file in the negative unseen data directory and do the print out. Percentage also calculated in both unseen data print out.

Result

```
Done Populating hashmap pos.txt
Done Populating hashmap neg.txt
pos dir, Number of files1000
Pos - Number of Pos File 729 72%
Pos - Number of Neg File 271 27%
```

The first 2 lines shows that the response of successfully repopulate the model into each HashMap. Result of the unseen positive directory, the model recognised 72% of unseen positive file and 29% it recognised the 27% of negative files.

```
neg dir, Number of files1000
Neg - Number of Pos File 295 29%
Neg - Number of Neg File 704 70%
```

For the negative directory it recognised 70% of the negative files and 29% of positive files which is a little bit worst.

To improve overall model, word parsing can be done better, I have reduced down the unique word in the vocabulary from over 50000 + words down to 38097 words. But some numbers and symbol are still exists within the model. Or run large data set that have more words and data to improve the quality of the model.

Improve Word Parsing

Large IMDB

```
Done Populating hashmap pos.txt
Done Populating hashmap neg.txt
pos dir, Number of files1000
Pos - Number of Pos File 825 82%
Pos - Number of Neg File 175 17%
```

More complex word parsing gave better result. Positive directory, it recognised 82% of positive files and 17% of negative file.

Negative directory, it recognised 18% of positive files and 81% of negative file.

```
neg dir, Number of files1000
Neg - Number of Pos File 187 18%
Neg - Number of Neg File 812 81%
```

To calculate the percentage : (i.e.)
(numberOfPosFile*100)/1000). The reason of multiplying

100 first is because division, the number can get too small the computer will round up to 0;

Small IMDB

```
Done Populating hashmap pos.txt
Done Populating hashmap neg.txt
pos dir, Number of files1000
Pos - Number of Pos File 988 98%
Pos - Number of Neg File 12 1%
```

Within the small data sets. Positive directory, it recognised 98% of positive files and 1% of negative file.

Negative directory, it recognised 18% of positive files and 81% of negative file.

```
neg dir, Number of files1000
Neg - Number of Pos File 818 81%
Neg - Number of Neg File 181 18%
```

Optimization

To speed up the process by having less word within the vocabulary simple word parsing is done in the BayesianSample class. First it check if the word length is not less than 2 character, if it is the word will be discarded. Then it check the word isn't start with a symbol or number, then if the word is not meaningless word such as "the, an, his, her etc". The rest that passed the parsing test will be loaded into vocabulary set. Using set to allow it only store the unique word not the duplicated word.

This help to optimize, provide to run faster during the testing phase where it's used to test on the unseen data. So that the unseen word have less word to compare to in the positive and negative words.