CIS4930 Assignment 3

Venkata Vadrevu (FSU id:vv18d)

Due: 20 November 2020

1 Introduction

This assignment requires us to use NaiveBayes method to classify a data set. I have used python to implement the NaiveBayes classification method. Naive-Bayes method is a probabilistic method for classification. Naive Bayes method classifies instance by choosing the class which has the highest probability.

2 Implementation Details

Why did I use a class?
In the implementation I have created a "NaiveBayes" class. In this class we have all functionalities such as train and predict. Using a class is beneficial because it is easy to implement and encapsulation makes it easy to share data and to implement. We can use a class as it helps us in other methods such as Ensemble to classify datasets.

What data Structures did I use?
I have used dictionaries extensively to solve the problem. Dictionaries are the preferred as the time taken to access any item is $O(1)$. The dictionaries are "classCount", "labelDict", "classDict".

classCount: This dictionary maps the classLabel with the number of times it appears in the training file. This is useful as we can get the count of a class in $O(1)$ time. This provides a part of information for calculating probability.

labelDict: This is a dictionary of dictionaries. It maps each Label to a dictionary that maps a value to it's count. This is a bit complicated to understand so I have provided an example. Let us say that the test file is as follows ...

+1 1:2 2:2 3:2
-1 1:1 2:1 3:1
+1 1:1 2:2 3:2

Now that we have this training file, we can see that the labels are 1, 2, and 3. Label '1' has takes the value 2, 1, and 1. Label '2' takes the value 2, 1, 2 and label '3' takes the value 2, 1, 2. So the labelDict[1] would return a dictionary {2:1, 1:2} because label 1 takes the value 2 one time, and the value 1 two times. Similarly labelDict[2] will return a dictionary {2: 2, 1:1} because it takes the value 2 two times and value 1 one time. Similarly labelDict[3] will return a dictionary {2:2, 1:1} because it takes the value 2 two times, and value 1 one time.

classDict: This is a dictionary that maps the classLabel to a labelDict. This dictionary makes the a labelDict for each classLabel.

Apart from dictionaries I have used one set called "labels" this stores all the labels. I have used a set because we can avoid duplicates and the search time for an element is $O(1)$

What are the methods?
The methods are:
1. def __init__(self):
2. def train(self, filename):
3. def testFile(self, filename):
4. def predict(self, instance):
5. def main(Trainfile, Testfile):

For creating a NaiveBayes model we need to read the file and train the model, we need to test a file by predicting each instance in the test files. I have created these functions to handle these tasks. Let us explore the functionalities of each functions.

1. **def__init__(self):**
This function just initiates an object. This is the constructor. We just instantiate the data structures in class.

2. **def train(self, filename):**
This function is reads the file three times. This is a pass algorithm, after these passes we populate all the dictionaries. Populating the dictionaries constitutes as training the model because we can use this information to predict and classify new instances.

3. **def testFile(self, filename):**
This function reads the file and calls the predict function to predict an instance. This function prints the False positives, False negatives, True positives, and True negatives.

4. **def predict(self, instance):**

This function predict takes an instance and returns the class label. This function chooses the class label by calculating the probability of each class and chooses the maximum one from them. In this implementation we use log to prevent underflow problems.

5. **def main(Trainfile, Testfile):**
This function creates a NaiveBayes model Object and trains and tests it.

3 Output

The program outputs 8 numbers. Each line has four numbers. The 8 integers are (first line) true positive in training, false negative in training, false positive in training, and true negative in training, and (second line) true positive in test, false negative in test, false positive in test, and true negative in test.

We have to test the file in two datasets bc and led, the output for both of them is given below.

Output for bc-train and bc-test
30 26 19 105
16 13 14 63

Time taken is: 0.0235 seconds

Output for led-train and led-test
399 239 92 1357
207 144 41 742

Time taken is: 0.0810 seconds

4 Thoughts and Reflection

This implementation of Naive Bayes takes linear time to train. The prediction of a single instance takes $O(1)$ time. So it is quick. But as Naive Bayes algorithm assumes that the labels are independent of other labels, it is not that accurate. The accuracy could be improved by ensemble methods or using a better prediction method. This implementation also does not perform smoothing. In this implementation I just assume the probability to be 0 when we are unable to find an instance with a similar value. Using smoothing can also help improve the methods accuracy.