Practical Machine Learning – Bayesian Classification

Objectives

Naive Bayes classifiers are a remarkably successful ML algorithm for learning to classify text documents. The initial objective of this assignment is to provide an implementation of **a Multinomial Naive Bayes** learning algorithm in Python for document classification.

The dataset you will be using (data.zip), which you can find on Canvas under Unit3, consists of movie reviews from IMDB (Internet Movie Database).

The data consists of three folders (train, test and trainSmall). Within each folder there are two subfolders. One called ***neg***, which contains movie reviews classified as negative and the other called ***pos*** which contains movie reviews classified as positive.

The objective of your multinomial Naïve Bayes algorithm is to:

(i) Build a model by using the negative and positive movie reviews contained in the *train* folder.
(ii) It should then take in the movie reviews in the folder *test* and classify them.
(iii) You should compare the predictions of your model with the true class label and calculate the accuracy of the model for positive and negative movie reviews..

The dataset contains 25,000 movie reviews in the *train* folder, split evenly between positive and negative.

If you find that the processing time for building your model is prohibitive then you can use the *trainSmall* folder (instead of the *train* folder) to train your model. The *trainSmall* folder contains just 6000 movie reviews split evenly between positive and negative. If possible, I recommend you use the larger *train* folder.

The folder *test*, which you will used to test your model, contains 4000 movie reviews, split evening between positive and negative.

Remember Naïve Bayes will treat each word as a single feature. This would give you as many features as there are words in your vocabulary. You should use a “bag of words” (Multinomial model) approach. The Multinomial model places emphasis on the frequency of occurrence of a word within all documents of a class (see Week 3 lecture slides for more details and a worked example).

I recommend that you complete this assignment using standard Python (no need for inclusion of Pandas and NumPy).

Stage 1 – File Parsing and Vocabulary Composition

You should initially create a data structure to store all unique words in a vocabulary. A **set** data structure in Python is useful for this purpose. You can keep iteratively adding lists of words to the set and it will only retain unique words.

Also, the following code is provided to get you started with iterating through the files in a specific directory.

```
import os
path = 'data\\train\\pos\\'

listing = os.listdir(path)

for file in listing:

    print ('Current file : ',  file)

    # open current file for reading
    f = open(path+file, "r", encoding="utf8")

    # read function will read all contents of a file into a single string
    # split function splits a string into a list of strings
    # using white space as a delimiter
    allWords = f.read().split()

    # close the file
    f.close()
```

Your next step is to record the frequency with which words occur in both the positive and negative movie reviews. I suggest that you create a dictionary to store the frequency of words in the negative reviews and another dictionary to store the frequency of words in the positive reviews.

The keys to each dictionary should correspond to all words in the vocabulary and the values should specify how often they occur for that class. For example, if the word “brilliant” occurs 55 times in the positive movie reviews then the key value pair in your positive dictionary should be <”brilliant” : 55>. You need to record the frequency of all the words for each class (positive and negative).

It can be useful when initially creating the positive or negative dictionary to use the values from the set (which contains all your unique words) to initialize all the keys for the dictionary. See example code below:

```
vocab = set(["this", "is", "an", "example"])

# creates a dictionary, which is initialized
# so that each key is a value from the set vocab
negDict = dict.fromkeys(vocab, 0)

print (negDict)
# prints the following
# {'is': 0, 'example': 0, 'an': 0, 'this': 0}
```

Stage 2 – Calculating Word Probability Calculations

Once you have populated your positive and negative dictionary with the frequency of each word, you must then work out the conditional probabilities for all words (for each class). In other words, for each word *w* you should work out the *P(w|positive)* and *P(w|negative).* These are the required probabilistic components of the Naïve Bayes model. I suggest that you create a dictionary for the positive conditional probabilities and another for the negative conditional probabilities.

Remember you are building a multinomial model therefore you should be using the following formula for calculating the conditional probabilities.

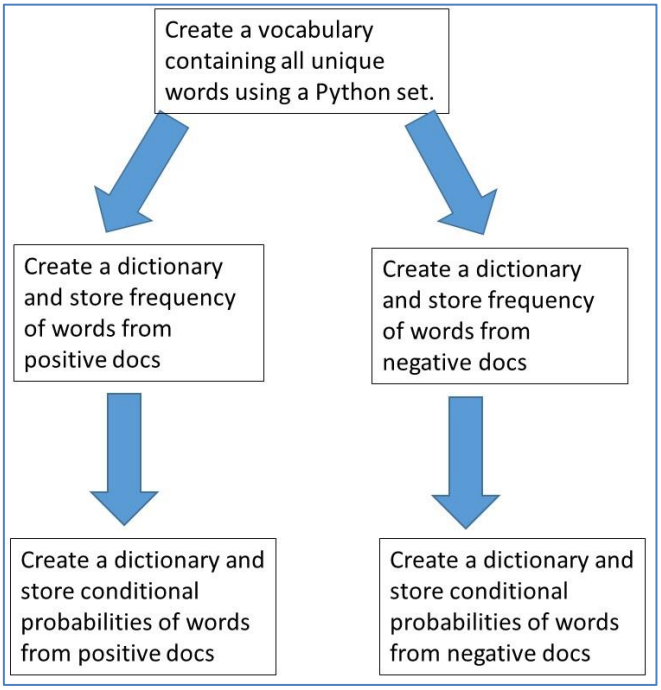
$$P(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$$

count(w, c) is the number of occurrences of the word w in all documents of class c.

count(c) The total number of words in all documents of class c (including duplicates).

|V| The number of words in the vocabulary

The general workflow of the program is depicted below.



Stage 3 – Classifying Unseen Documents and Basic Evaluation

The final section of your code will take as input a new document (a movie review that has not been used for training the algorithm) and classify the document as a positive or negative review. You will need to read all words from the document and determine the probability of that document being a positive review and negative review using the multinomial model.

Remember to use the log in your probability calculations. See lecture notes for more details on probability classification.

You should run all documents from the *test* folder through your algorithm and determine the level of accuracy (the percentage of documents correctly classified) for both the positive and negative movie reviews in the test folder.

Preprocessing

Next you can investigate the impact of pre-processing the dataset.

Initially you can implement basic techniques such as lower-casing all words and removing punctuation. You should see that even these basic techniques can significantly reduce the size of your vocabulary.

You can also explore common method for pre-processing and improving accuracy such as stopword removal, stemming, n-grams etc.

NLTK, Pythons natural language toolkit (http://nltk.org/) is a useful resource.

The regular expression library in Python may also prove useful in performing pre-processing techniques (https://docs.python.org/3.7/library/re.html). This provides capabilities for extracting whole words and removing punctuationYou can find a tutorial on regular expression at https://developers.google.com/edu/python/regular-expressions .