Nuthalapati Jahnavi

2208-CSE-6363-001-MACHINE LEARNING
INSTRUCTOR : DAJIANG ZHU

Bayesian Learning for classifying news text articles
Project 2 – Report

Note : The path given is not relative i.e it won't be same for every system. So please kindly change the path accordingly in-order to view the result correctly. I have displayed the screenshots of the correct result after providing that path.

SYNOPSIS

Given is the dataset of 20_newsgroups in which the task is to perform naïve bayes algorithm for text classification. We know there has been a humongous exponential growth of data as compared to past. It's not that easy to access or find for the data which they need from the available sources as there are millions of available categories to choose from. Therefore, in-order to make it effective, text classification will be used in-order to classify the data into various categories. And this will also be one the challenging task as we have various factors such as accuracy/preciseness, classification etc. to be satisfied in-order to get a correct result from classification.

So in this task, Naïve Bayes is proven is to be on of the most effective algorithm in text classification area. Basically, a naïve bayes classifier is a probabilistic machine learning model that will be used for task classification in which every feature couple that will be classified is independent of each other. (Source : https://www.geeksforgeeks.org/naive-bayes-classifiers/ | https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c). It tries to estimate the likelihood of a particular data falling into one particular category by using probabilistic models which is possible by using a group of words for retrieving the information where the order isn't not so important.

As per, https://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html we know that the given default code is based on Rainbow software package which has enough code for classifying text documents to implement naïve bayes algorithm. This given code is used as a base or support for creating other classification models.
Important steps involved are :

Document Reading and building a model : In this step the files are organized in such a way that they're placed in separate folders with a particular name labels, which has all the relevant training data and also test data which will be revised in later parts of the classification. Then a

model is built in such a way that, rainbow is called with -i option, with one folder name for each class.
For example rainbow -d ~/model –I ~/20_newsgroups/talk.religion.* where all the sub-categories related to religion will be further opened.

Next step is **Options Tokenizing** where certain filler words will be eliminated such as “is”, “a”, “the”, etc. In this, as the name suggests that character’s list into tokens. The list of tokenization, working of Rainbow and it’s implementation is mentioned here : http://www.cs.cmu.edu/~mccallum/bow/rainbow/ .

Next is the **Document Classification** phase in which from the given training documents will be able to find the classifier parameters based on which a group of testing set will be produced as the output. This process will involve certain number of trials in which it will produce a unique output. So the test-train split will be will be in the ratio 40 to 60.

Choosing Method of Classification
A special option is present --method in which default and various classifications will be present which makes the process simpler in which the Naïve Bayes is the default one along with other methods. In which the default keywords will be knn, prind, naivebayes, etc.
As our project is based on Naïve Bayes, we have some of the methods as – (Pic Courtesy : http://www.cs.cmu.edu/~mccallum/bow/rainbow/)

`--smoothing-method=METHOD`	Set the method for smoothing word probabilities to avoid zeros; `METHOD` may be one of: `goodturing, laplace, mestimate, wittenbell`. The default is `laplace`, which is a uniform Dirichlet prior with alpha=2.
`--event-model=EVENTNAME`	Set what objects will be considered the `events' of the probabilistic model. `EVENTNAME` can be one of: `word` (i.e. multinomial, unigram), `document` (i.e. multi-variate Bernoulli, bit vector), or `document-then-word` (i.e. document-length-normalized multinomial). For more details on these methods, see *A Comparison of Event Models for Naive Bayes Text Classification*. The default is `word`.
`--uniform-class-priors`	When classifying and calculating mutual information, use equal prior probabilities on classes, instead of using the distribution determined from the training data.

Code Explanation/Implementation

- First download the data from the given link.
- It gets imported to do the text classification using naïve bayes to calculate the accuracy on how the data is classified properly by using train-split data.
- **Specify the path of dataset in which the folders are present :** The dataset path will be provided in. order to do the naïve bayes process on the given data.
- An array is created in order to store the files in separate folders which specifies the **file length.**
- The data is **divided into test and train** and initialized after which the **filler/blockwords will be eliminated** in-order to find the accuracy properly. Post data-splitting **training data is cleaned** i.e the fillers will be removed (symbols, punctuations, etc.)

```
[ ]
    list_pathname = []
    for fo in range(len(Folder)):
        for fi in Files[fo]:
            list_pathname.append(join(folder_path, join(Folder[fo], fi)))

            #here in the block of code helps in divind or splitting the dataset into training dataset and test dataset
```

"blockwords = ['the','a','we','was', etc.] → This removes the fillers

- Then **preprocessing** will be done on the given data in-order to remove the punctuations and other unnecessary fillers

```
def tokenize_line(line):
    words = line[0:len(line)-1].strip().split(" ")
    words = preprocess(words)
    words = remove_blockwords(words)

    return words
```

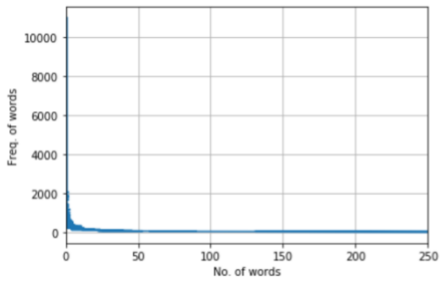
-

```
table = str.maketrans('', '', '\t')
words = [word.translate(table) for word in words]
```

(here words is a keyword in tokenization process)

- **Words appearance (frequency)** is taken into account in every file. After this is done, a total of this is calculated and stored.

```
import matplotlib.pyplot as plt
y = f_o_w
x = n_o_w
plt.xlim(0,250)
plt.xlabel("No. of words")
plt.ylabel("Freq. of words")
plt.plot(x, y)
plt.grid()
plt.show()
```



- In the same way as training-data, the **test-data** is also cleaned in a similar way.
- Then **tokenizing** will be done i.e the list of sentences will be broken down into certain set of tokens as "`document_words.append(tokenize_line(line))`".

```
#tokenizing is done here

def tokenize(path):

    f = open(path, 'r')
    text_lines = f.readlines()


    text_lines = delete_metadata(text_lines)


    document_words = []
    #stores the words that has been generated so far


    for line in text_lines:
        document_words.append(tokenize_line(line)) #tokenizing process

    return document_words
```

- A **list of unique words** will be taken from the function collected
- Then these are sorted/placed in such a way according to the count/number of times in which they occur in the document.
- Now the total words will be used in featuring and then **a dictionary** will be created that consists of words that frequently occurs where a higher dimensional array will be created in order to store these words.

```
dictionary = {}
document_num = 1
Add code cell
⌘/Ctrl+M B
ument_words in words_list:
#print(document_words)
    np_document_words = np.asarray(document_words)
    w, c = np.unique(np_document_words, return_counts=True)
    dictionary[document_num] = {}
    for i in range(len(w)):
        dictionary[document_num][w[i]] = c[i]
    document_num = document_num + 1
```

+ Code + Text

- Now a function “log_probability” will be used in-order to calculate the probability using the naïve-bayes method. **Predict()** will be used in predicting the class of the data.
- This finally gives the **accuracy** of the given data by using the function

```
def fit(X_train, Y_train):
    word_result = {}
    classes, counts = np.unique(Y_train, return_counts=True)

    for i in range(len(classes)):
        curr_class = classes[i]

        word_result["TOTAL_DATA"] = len(Y_train)
        word_result[curr_class] = {}

        X_train_curr = X_train[Y_train == curr_class]

        number_of_features = n

        for j in range(number_of_features):
            word_result[curr_class][features[j]] = X_train_curr[:,j].sum()

        word_result[curr_class]["TOTAL_COUNT"] = counts[i]

    return word_result
```

```
def log_probablity(dictionary_train, x, curr_class):
    result = np.log(dictionary_train[curr_class]["TOTAL_COUNT"]) - np.log(dictionary_train["TOTAL_DATA"])
    number_of_words = len(x)
    for j in range(number_of_words):
        if(x[j] in dictionary_train[curr_class].keys()):
            xj = x[j]
            count_curr_class_xj = dictionary_train[curr_class][xj] + 1
            count_curr_class = dictionary_train[curr_class]["TOTAL_COUNT"] + len(dictionary_train[curr_class].keys())
            curr_xj_prob = np.log(count_curr_class_xj) - np.log(count_curr_class)
            result = result + curr_xj_prob
        else:
            continue

    return result
```

- Here in the screenshot below, "`details such as precision, recall, f1-score and support is printed.`"

```
print(classification_report(Y_train, Y_prediction_train))
#details such as precision, recall, f1-score and support is printed
```

	precision	recall	f1-score	support
alt.atheism	0.71	0.85	0.77	767
comp.graphics	0.67	0.77	0.72	747
comp.os.ms-windows.misc	0.81	0.78	0.80	751
comp.sys.ibm.pc.hardware	0.77	0.81	0.79	760
comp.sys.mac.hardware	0.80	0.87	0.83	764
comp.windows.x	0.88	0.79	0.83	760
misc.forsale	0.86	0.84	0.85	739
rec.autos	0.88	0.88	0.88	731
rec.motorcycles	0.85	0.94	0.89	716
rec.sport.baseball	0.94	0.94	0.94	752
rec.sport.hockey	0.91	0.95	0.93	769
sci.crypt	0.92	0.89	0.90	767
sci.electronics	0.84	0.78	0.81	756
sci.med	0.94	0.88	0.90	744
sci.space	0.93	0.88	0.90	754
soc.religion.christian	0.84	0.89	0.86	745
talk.politics.guns	0.74	0.88	0.80	751
talk.politics.mideast	0.91	0.85	0.88	719
talk.politics.misc	0.74	0.71	0.72	741
talk.religion.misc	0.73	0.45	0.56	764
accuracy			0.83	14997
macro avg	0.83	0.83	0.83	14997
weighted avg	0.83	0.83	0.83	14997

```
def predict(dictionary_train, X_test):
    Y_pred = []
    for x in X_test:
        y_predicted = predictSinglePoint(dictionary_train, x)
        Y_pred.append(y_predicted)
```

```
X_test = []

for key in test_dictionary.keys():
    X_test.append(list(test_dictionary[key].keys()))
```

```
final_prediction = predict(train_dictionary, X_test)
```

```
final_prediction = np.asarray(final_prediction)
```

```
accuracy_score(Y_test, final_prediction)
```

```
0.5632
```

```
print(classification_report(Y_test, final_prediction))
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.65      0.64      0.64       233
           comp.graphics       0.51      0.57      0.54       253
 comp.os.ms-windows.misc       0.85      0.26      0.40       249
comp.sys.ibm.pc.hardware       0.63      0.57      0.60       240
   comp.sys.mac.hardware       0.92      0.37      0.53       236
          comp.windows.x       0.52      0.80      0.63       240
            misc.forsale       0.83      0.30      0.44       261
               rec.autos       0.76      0.35      0.48       269
         rec.motorcycles       0.98      0.31      0.47       284
      rec.sport.baseball       0.98      0.61      0.75       248
        rec.sport.hockey       0.86      0.84      0.85       231
               sci.crypt       0.53      0.85      0.65       233
         sci.electronics       0.77      0.32      0.46       244
                 sci.med       0.89      0.61      0.73       256
               sci.space       0.81      0.63      0.71       246
  soc.religion.christian       0.62      0.88      0.73       252
      talk.politics.guns       0.75      0.42      0.54       249
   talk.politics.mideast       0.51      0.93      0.66       281
      talk.politics.misc       0.19      0.83      0.31       259
      talk.religion.misc       0.51      0.20      0.29       236

                accuracy                           0.56      5000
               macro avg       0.70      0.56      0.57      5000
            weighted avg       0.70      0.56      0.57      5000
```

Important Methods to be noted

- **folder_path :** The path of the dataset is given
- **`list_pathname.append(): Divide or splits the data into training and testing.`**

- **def preprocess(words) :** The words are pre-processed in-order to remove the punctuations/fillers.
- **Def tokenize_line(line):** A particular line of sentences will be broken into words.
- **Def delete_metadata(lines):** `metadata is removed`
- `def tokenize(path): It tokenizes the words that has been generated so far from the given set.`
- **np.unique() :** The unique words that have been extracted from the file documents.
- `sorted(zip(counts, words): Sorts the words based on the occurance.`
- `dictionary = {} : creates a dictionary which consists of each doc's vocabulary.`
- `dictionary.keys(): It provides the indexing for the words generated so far.`
- `def fit(X_train, Y_train): creates a training dictionary consisting of the word occurance.`
- `def log_probablity(dictionary_train, x, curr_class): The log probability (naïve-bayes) of the document of test data.`
- `def predictSinglePoint(dictionary_train, x): The class of the document is predicted for a file`
- `accuracy_score(Y_test, final_prediction): The accuracy is calculated by considering the predicted data from the above processes.`
- Certain screenshots, has been provided with respect to the methods, all the detailed methods are present in the code.
- Note : Every time you execute the precison and other details will be sometimes changed as the the words considered everytime will be different.

References

Several resources have been referenced. I have mentioned few of them.

- http://www.cs.cmu.edu/~mccallum/bow/rainbow/
- https://www.geeksforgeeks.org/naive-bayes-classifiers/
- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://github.com/mohsen-imani/Text-Classification/blob/master/Naive_Bayes.py
- https://github.com/poojasrinivasan/TextClassification
- https://www.tutorialspoint.com/python/os_listdir.htm