Name: Adit Devendra Patil Id:1001757372

Classifying Newsgroup Data using Naïve Bayes

Introduction:

In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models. But they could be coupled with Kernel density estimation and achieve higher accuracy levels.

Data:

The dataset contains 20,000 newsgroup messages drawn from the 20 newsgroups. The dataset contains 1000 documents from each of the 20 newsgroups.

Section of the problem:

We have to use Multinomial Naïve Bayes to classify the dataset. Multinomial Naïve bayes classification algorithm is being used to classify the dataset by dividing the dataset into training and testing set into 50%.

Method:

1. Data Preprocessing: Data Preprocessing is that step in which the data gets transformed, or Encoded, to bring it to such a state that now the machine can easily parse it. Firstly, I have created a list of stop words and symbols and remove them from our data as won't be of much importance when we apply our algorithm. Secondly, we will be filtering out the digits as they won't be of importance being not a word.

```
def clean(dataDict):
    # Creating list of stop words
    stop_words = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
    symbols = ['<','>','?','.','"',')','(','|','-','#','*','+','\'','&','^','`','~','\t','$','%',"'",'!','/','\\',
    stop_words = stop_words + symbols
    vocab={}
# Creating a dictionary of words and their frequency
    for i in range(len(dataDict)):
        for doc in dataDict[folders_list[i]]:
            for word in doc.split():
                if word.lower() not in stop_words and len(word.lower()) >= 5:
                    if word.lower() not in vocab:
                        vocab[word.lower()]=1
                    else:
                        vocab[word.lower()]+=1
    return vocab

vocab = clean(dataDict)
```

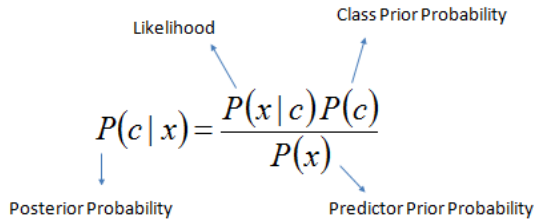
2. Training the model: In this step, we would start training our model. We would be creating the vocabulary dictionary. We would filter the words from each file based on their frequency.

```
def sort_word_frequency(vocab):
    words_frequency=sorted(vocab.items(), key=lambda kv: kv[1], reverse=True)
    return words_frequency

words_frequency = sort_word_frequency(vocab)
#Building the feature list from vocab which contains highest freqency words

# Choosing top 1500 vocab words as features
dictionary_words = {}
feature_list=[]
i=0
for key in words_frequency:
    if i ==1500:
        break
    dictionary_words[key[0]] = i
    feature_list.append(key[0])
    i+=1
```

Also, we would be creating a feature list which will contain the top featured words in all the documents. I have chosen top 1500 words as feature word list. The more words you take in feature list, the more accuracy will be.
Here, we can see first 30 words of feature word list:

```
on.py
Top 30 freqency words are:
[('subject:', 20486), ('from:', 20417), ('date:', 20137), ('newsgroups:', 20081)
, ('message-id:', 20050), ('lines:', 20042), ('path:', 20029), ('organization:',
 19246), ('would', 15512), ('writes:', 14040), ('references:', 12548), ('article
', 12108), ('sender:', 10907), ('nntp-posting-host:', 8598), ('people', 8415), (
'university', 8203), ('think', 7205), ('xref:', 6054), ('cantaloupe.srv.cs.cmu.e
du', 6050), ('could', 5905), ('distribution:', 4406), ('first', 4114), ('anyone'
, 3976), ('world', 3602), ('really', 3507), ('since', 3418), ('right', 3326), ('
believe', 3308), ('still', 3290), ("max›'ax›'ax›'ax›'ax›'ax›'ax›'ax›'ax›'ax›'ax›
'ax›'ax›'ax›'ax›'", 3289)]
```

3. Testing the model: After training is finished, model starts to test the testing set and predicts the best class by doing probability calculations.
Our model is based on Multinomial Naive Bayes algorithm to classify textual data and predict class of the document. Naive Bayes algorithm works on bayes theorem.
We select the best class for the text in the text classification.
We calculate the probability of each term belonging to the class in the naive Baye s algorithm.
We also
measure each class 's prior probabilities and multiply this probability with the pro bability previously calculated. We continue the same calculations for each class and assign the class having the highest score to the particular category.
Understanding the algorithm.
The famous formula of Naïve bayes:



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

P(c) : Prior probability of a class. It can be calculated as below:
P(c) = Nc/N
Nc : no of documents in a class

N : total no of documents

The probability of term belonging to a particular class can be calculated as:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'},$$

Tct : no of occurrences of term t in training documents in class c.
And denominator denotes total of no of terms of in a class and no of terms in a vocabulary(B').

Smoothing in probabilities: The frequency-based probability might introduce zeros when multiplying the probabilities, leading to a failure in preserving the information contributed by the non-zero probabilities. Therefore, a smoothing approach, for example, the Laplace smoothing, must be adopted to counter this problem.
Probabilities lie between range 0 to 1 and often are in decimal format. Hence, while taking multiplication, product will be a very small number as multiplication of two decimal numbers always result smaller number (For example, 0.01 * 0.5 = 0.005). To deal with this problem, we apply normalization by taking logarithmic

probabilities.

```
def cal_probability(x,dictionary,class_present):
    result=np.log(dictionary[class_present]["total_count"])-np.log(dictionary["total_data"])
    for i in range(len(feature_list)):
        word_count_present=dictionary[class_present][feature_list[i]]+1
        total_word_count=dictionary[class_present]["total_count"]+len(feature_list)
        current_word_probability=np.log(word_count_present)-np.log(total_word_count)
        for j in range(int(x[i])): # if the frequency of word in test data point is zero then we wont consider it.
            result+=current_word_probability
    return result


def prediction(X_test,dictionary):
    Y_prediction_res=[]
    num = 0
    for x in X_test:
        best_class=-1000
        best_prob=-1000
        F=True
        classes=dictionary.keys()
        for class_present in classes:
            if class_present=="total_data":
                continue
            class_present_probability=cal_probability(x,dictionary,class_present)
            if(F==True or class_present_probability>best_prob):
                best_class=class_present
                best_prob=class_present_probability
            F=False
        Y_prediction_res.append(best_class)
    return Y_prediction_res
```

We do this calculation for no of classes and we will assign best class to the document by picking the maximum score.

Results: We are evaluating the result based on two metrics to show the performance of the model: accuracy score and MSE (mean square error). Below is the snapshot from the output:

	precision	recall	f1-score	support
alt.atheism	0.82	0.70	0.76	355
comp.graphics	0.77	0.78	0.77	286
comp.os.ms-windows.misc	0.82	0.87	0.84	275
comp.sys.ibm.pc.hardware	0.86	0.89	0.87	332
comp.sys.mac.hardware	0.89	0.85	0.87	318
comp.windows.x	0.82	0.90	0.85	285
misc.forsale	0.88	0.68	0.77	381
rec.autos	0.88	0.79	0.83	314
rec.motorcycles	0.90	0.83	0.86	334
rec.sport.baseball	0.98	0.92	0.95	335
rec.sport.hockey	0.98	0.96	0.97	294
sci.crypt	0.87	0.92	0.89	306
sci.electronics	0.89	0.83	0.85	327
sci.med	0.83	0.88	0.86	279
sci.space	0.86	0.89	0.87	275
soc.religion.christian	1.00	0.97	0.98	302
talk.politics.guns	0.81	0.68	0.74	352
talk.politics.mideast	0.78	0.95	0.86	235
talk.politics.misc	0.47	0.74	0.57	178
talk.religion.misc	0.49	0.60	0.54	236
accuracy			0.83	5999
macro avg	0.83	0.83	0.83	5999
weighted avg	0.84	0.83	0.83	5999

Additional Experiment:

Initially there are many words in the feature list which do not help us in classification as they are very common.

In my experiment, I have choose some words which had top frequency and appeared many time.

With that respect I have added those words into stop words and fittered the data accordingly.

```
    stop_words = stop_words + symbols
    # Common words throughout all docs play no part in classification ,so removing them
    stop_words+=['subject:','from:', 'date:', 'newsgroups:', 'message-id:', 'lines:', 'path:', 'organization:',
                 'would', 'writes:', 'references:', 'article', 'sender:', 'nntp-posting-host:', 'people',
                 'university', 'think', 'xref:', 'cantaloupe.srv.cs.cmu.edu', 'could', 'distribution:', 'first',
                 'anyone','world', 'really', 'since', 'right', 'believe', 'still',
                 "max>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'ax>'"]
    vocab={}
# Creating a dictionary of words and their frequency
```

By doing that, I came to the conclusion that it gives little bit more accuracy.

Name: Adit Devendra Patil Id:1001757372

	precision	recall	f1-score	support
alt.atheism	0.82	0.70	0.76	350
comp.graphics	0.82	0.78	0.80	318
comp.os.ms-windows.misc	0.84	0.85	0.84	310
comp.sys.ibm.pc.hardware	0.86	0.82	0.84	308
comp.sys.mac.hardware	0.90	0.91	0.90	310
comp.windows.x	0.86	0.88	0.87	305
misc.forsale	0.84	0.82	0.83	322
rec.autos	0.88	0.84	0.86	301
rec.motorcycles	0.88	0.84	0.86	309
rec.sport.baseball	0.97	0.94	0.96	347
rec.sport.hockey	0.97	0.93	0.95	298
sci.crypt	0.87	0.91	0.89	292
sci.electronics	0.86	0.83	0.84	306
sci.med	0.82	0.85	0.84	281
sci.space	0.87	0.92	0.89	272
soc.religion.christian	0.99	0.92	0.95	304
talk.politics.guns	0.81	0.72	0.76	329
talk.politics.mideast	0.85	0.92	0.88	287
talk.politics.misc	0.59	0.68	0.63	250
talk.religion.misc	0.46	0.68	0.55	200
accuracy			0.84	5999
macro avg	0.84	0.84	0.84	5999
weighted avg	0.85	0.84	0.84	5999

Name: Adit Devendra Patil Id:1001757372

References:

https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67

https://www.youtube.com/watch?v=CPqOCI0ahss&t=1s
https://www.youtube.com/watch?v=0kPRaYSgbIM
https://medium.com/@saitejaponugoti/stop-words-in-nlp-5b248dadad47

https://www.deep-teaching.org/notebooks/natural-language-processing/text-classification/exercise-naive-bayes