***Abstract****—For the given NetNews Text Articles, our aim is to classify the articles into relevant article groups using Bayesian learning. We have 500 articles from each group to perform supervised learning to each group. After making the model learn how to classify, it is given ten thousand test files to classify the data into given groups. This model uses Naïve Bayes Classification which analyze the text of each article and defines the class based on the prior probability of each word appearing in the class.*

**Introduction**

The data set is available on the following URL “<http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html>”. The data set has total twenty thousand articles classified in twenty classes. This model uses five hundred articles from each class for learning and remaining five hundred as testing. This model performs text analyzation on the documents and classify them their respective class using Naïve Bayes classification

**What is Naïve Bayes Classification?**

*Naive Bayes classifiers* are a family of simple [probabilistic classifiers](https://en.wikipedia.org/wiki/Probabilistic_classifier) based on applying [Bayes' theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem) with strong (naive) [independence](https://en.wikipedia.org/wiki/Statistical_independence) assumptions between the features.[1-wikipedia]. It uses simplified Bayes law of probability for classification.

The probability of document d being in class c is given by[2]:



where **P(tk|c)** is the conditional probability of term t­ occurring in a document of class **c** We interpret **P(tk|c)** as a measure of how much evidence **tk** contributes that **c** is the correct class. P(c) is the prior probability of a document occurring in class **c**. If a document's terms do not provide clear evidence for one class versus another, we choose the one that has a higher prior probability. [3 Stanford]

Advantages of Bayes Classification:

Naïve Bayes Classification is simple to use and implement. It is competitive to more sophisticated classification Models as SVM in case of accuracy and perfection.

**Naïve Classification on this Data**

The model tunes the data by compressing the text files and the sentences are divided into words, where each word is called as token. The data is then refined for any stop words like (the, a, and etc.). Model finds, Prior probability of class, Vocabulary and the probability of each token in a class in the Train data and this data is stored for testing. While testing a document is scanned for each token in the vocabulary. The probability of each class is found with the help of prior probability of class and how much a word contribute to the class.

**Accuracy of our Model**

The accuracy for this model is determines by the Total number of Test Documents correctly classified class divided by total number of Test Documents

**Accuracy**= Total Number of correctly Classified Test Documents/ Total Number of Test Documents

**Results and Conclusions**

Total Documents=20000

Total Classes= 20

Total Train Documents=10000

Total Test Documents= 10000

Number of Documents correctly classified=195

Accuracy for the model= 1.95%

**Conclusion:** For Naïve Bayes Classification to work properly Text should be properly cleared of stop words and tokens should be most appropriate

**Limitation**

1. Naïve Bayes Classifier assumes there is not connection between different attributes of a class. Hence it should not be used where the attributes are highly dependent on each other
2. The model presented here believes every attribute is location independent. However sometimes attributes at different location have different face value.

Code:

*"""""Code By Prachi Goel   
Student ID-1001234789"""***import** time  
**import** os,codecs,string  
**from** nltk.tokenize **import** word\_tokenize  
**from** nltk.corpus **import** stopwords  
**import** numpy **as** np  
**import** math  
  
dir1=**"C:\\Users\\prach\\Desktop\\New folder\\UTA Ms\\Fall'17\\Machine Learning\\Project\_2\\Twenty\_newsgroups"**Train\_Class=[]  
Test\_Class=[]  
Tokenized\_Train\_data=[]  
Tokenized\_Test\_data=[]  
file\_class\_train=**""**Class\_Tokens=[]  
Vocabulary=[]  
Class\_of\_test\_Document=[]  
Test\_dict={}  
**''' Train Class's Compressed Data list ,Test Document list, remove end words  
for each Train class find token probability,Each class probabilty, Total number of token in each class, total number of token in vocabulary'''  
  
def** Splitting\_Data\_Into\_Test\_Train(dir1,Train\_Class,Test\_Class,file\_class\_train):  
 print(**"Please wait Data is being split into Trianing and Test"**)  
 **for** folder **in** os.listdir(dir1):  
  
 iteration=len(Test\_Class)  
 count =0  
 **for** file **in** os.listdir(dir1+**"\\"**+folder):  
 **with** codecs.open(dir1 + **"\\"** + folder + **"\\"** + file, **'r'**,encoding=**'utf-8'**,errors=**'ignore'**) **as** readingf:  
 **if** count<=500:  
 file\_class\_train += readingf.read()  
 count+=1  
 **if** count>500:  
 file\_test=readingf.read()  
 Test\_Class.append(file\_test)  
 count += 1  
  
 print (**"Number of documents in test data "**,folder,**" are: "**, len(Test\_Class)-iteration)  
 Train\_Class.append(file\_class\_train)  
 print(**"All The files for Class "**, len(Train\_Class), **"merged"**)  
 print(**"Data ready for Training"**)  
 **return** Train\_Class,Test\_Class  
  
  
**def** Tokenization(Data,Tokenized\_Data,Vocabulary):  
 punc = string.punctuation  
 stopWords = set(stopwords.words(**'english'**))  
 tokens = word\_tokenize(Data)  
 words=[w.lower() **for** w **in** tokens]  
 wordsFiltered = []  
 **for** word **in** words:  
 **if** word **not in** (stopWords) **and** word **not in** (punc) **and** word.isdigit()==**False and** len(word)>1:  
 **for** w **in** word:  
 **if** w **not in** (punc):  
 wordsFiltered.append(word)  
 Vocabulary.append(word)  
 print( **"class"**,len(Tokenized\_Data)+1,**"extracted"**)  
 Tokenized\_Data.append(wordsFiltered)  
 **return** Tokenized\_Data,Vocabulary  
  
**def** Token\_Probability(Tokenized\_Data,Class\_Tokens,Vocabulary\_Count):  
 Different\_tokens= list(set(Tokenized\_Data))  
 dict={}  
 total\_words=0  
 **for** i **in** Different\_tokens:  
 count=Tokenized\_Data.count(i)  
 total\_words+=count  
 dict[i]=(count+1)/(Vocabulary\_Count+total\_words)  
 (Class\_Tokens.append(dict))  
 print(len(Class\_Tokens))  
 **return** Class\_Tokens  
  
**def** Training\_Data(Train\_Class,Tokenized\_Train\_Data,Class\_Tokens,Vocabulary):  
 Number\_of\_Classes = 20  
 Number\_of\_Total\_Articles = 20000  
 Number\_of\_Articles\_Each\_Class = 500  
 Prior\_Probability\_of\_Class = Number\_of\_Articles\_Each\_Class / Number\_of\_Total\_Articles  
 **for** i **in** range (len(Train\_Class)):  
 Tokenized\_Data=Tokenization(Train\_Class[i],Tokenized\_Train\_data,Vocabulary)  
 Vocabulary\_Count=(len(list(set(Tokenized\_Data[1]))))  
 Vocabulary=list(set(Tokenized\_Data[1]))  
 **for** i **in** range (len(Tokenized\_Data[0])):  
 print(**"Please wait finding the probability of token in class: "**,i)  
 Class\_Tokens=Token\_Probability(Tokenized\_Data[0][i],Class\_Tokens,Vocabulary\_Count)  
 **return** Class\_Tokens,Vocabulary,Prior\_Probability\_of\_Class  
  
**def** Testing\_class(Test\_Class):  
 **for** i **in** range(len(Test\_Class)):  
 **if** i<500:  
 Class=0  
 **return** Class  
 **elif** 500<=i<1000:  
 Class=1  
 **return** Class  
 **elif** 1000<=i<1500:  
 Class=2  
 **return** Class  
 **elif** 1500<=i<2000:  
 Class=3  
 **return** Class  
 **elif** 2000<=i<2500:  
 Class=4  
 **return** Class  
 **elif** 2500<=i<3000:  
 Class=5  
 **return** Class  
 **elif** 3000<=i<3500:  
 Class=6  
 **return** Class  
 **elif** 3500<=i<4000:  
 Class=7  
 **return** Class  
 **elif** 4000<=i<4500:  
 Class=8  
 **return** Class  
 **elif** 4500<=i<5000:  
 Class=9  
 **return** Class  
 **elif** 5000<=i<5500:  
 Class=10  
 **return** Class  
 **elif** 5500<=i<6000:  
 Class=11  
 **return** Class  
 **elif** 6000<=i<6500:  
 Class=12  
 **elif** 6500<i<7000:  
 Class=13  
 **return** Class  
 **elif** 7000<=7500:  
 Class=14  
 **return** Class  
 **elif** 7500<=i<8000:  
 Class=15  
 **return** Class  
 **elif** 8000<=i<8500:  
 Class=16  
 **return** Class  
 **elif** 8500<=i<9000:  
 Class=17  
 **return** Class  
 **elif** 9000<=i<9500:  
 Class=18  
 **return** Class  
 **elif** 9500<=i<10000:  
 Class=19  
 **return** Class  
  
**def** Testing\_Data(Class\_Tokens,Vocabulary,Prior\_Probability\_of\_Class,Test\_Class,Class\_of\_test\_Document):  
 **for** i **in** range(Test\_Class):  
 Score=0  
 Tokenized\_Data = Tokenization(Test\_Class[i],Tokenized\_Test\_data, Vocabulary)  
 **for** text **in** Tokenized\_Data:  
 Probability\_of\_Class\_given\_document=[]  
 Class\_Score=0  
 **for** c **in** range(0,20):  
 Class\_Score=math.log(Prior\_Probability\_of\_Class)  
 **for** token **in** Tokenized\_Data:  
 **if** token **in** Vocabulary:  
 Class\_Score=math.log(Class\_Tokens[c][token])  
 Probability\_of\_Class\_given\_document.append(Class\_Score)  
 Class\_of\_test\_Document.append(np.argmax(np.array(Probability\_of\_Class\_given\_document)))  
 **if** Testing\_class(i)==Class\_of\_test\_Document:  
 Score+=1  
 print(Class\_of\_test\_Document)  
 Accuracy=Score/(10000)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 Stime=time.time()  
 Dataset=Splitting\_Data\_Into\_Test\_Train(dir1,Train\_Class,Test\_Class,file\_class\_train)  
 Data\_after\_Training=Training\_Data(Dataset[0],Tokenized\_Train\_data,Class\_Tokens,Vocabulary)  
 Testing\_Data(Data\_after\_Training[0],Data\_after\_Training[1],Data\_after\_Training[2],Dataset[2],Class\_of\_test\_Document)  
 Stop\_time=time.time()  
 Total\_time=Stop\_time-Stime  
 print (Total\_time)