In [123]:

```
import pandas as pd
import numpy as np
from collections import defaultdict
import re
```

In [124]:

```
def preprocess_string(str_arg):
    cleaned_str=re.sub('[^a-z\s]+',' ',str_arg,flags=re.IGNORECASE) #every char except
    cleaned_str=re.sub('(\s+)',' ',cleaned_str) #multiple spaces are replaced by singl
    cleaned_str=cleaned_str.lower() #converting the cleaned string to lower case

    return cleaned_str # returning the preprocessed string
```

In [125]:

```
class NaiveBayes:

    def __init__(self,unique_classes):

        self.classes=unique_classes # Constructor is sinply passed with unique number


    def addToBow(self,example,dict_index):

        #print("Ex 1: ",example)

        if isinstance(example,np.ndarray):
            example=example[0]
            #print("is instance executed")

        #print("Ex 2: ",example)
        #print("dict indx:",dict_index)

        for token_word in example.split(): #for every word in preprocessed example
            self.bow_dicts[dict_index][token_word]+=1 #increment in its count


    def train(self,dataset,labels):
        self.examples=dataset
        self.labels=labels
        self.bow_dicts=np.array([defaultdict(lambda:0) for index in range(self.classes
        print("Init Bow Dict",self.bow_dicts)

        if not isinstance(self.examples,np.ndarray): self.examples=np.array(self.examp
        if not isinstance(self.labels,np.ndarray): self.labels=np.array(self.labels)

        #constructing BoW for each category
        #print(self.labels==0)
        for cat_index,cat in enumerate(self.classes):
            all_cat_examples=self.examples[self.labels==cat]


            cleaned_examples=[preprocess_string(cat_example) for cat_example in all_ca
            #print("Cleaned Ex 1: ",cleaned_examples)
            #print("Cleaned Ex 1 type: ",type(cleaned_examples))

            cleaned_examples=pd.DataFrame(data=cleaned_examples)
            #print("Cleaned Ex 2: ",cleaned_examples)
            #print("Cleaned Ex 2 type: ",type(cleaned_examples))

            #now costruct BoW of this particular category
            np.apply_along_axis(self.addToBow,1,cleaned_examples,cat_index)

            prob_classes=np.empty(self.classes.shape[0])
            all_words=[]
            cat_word_counts=np.empty(self.classes.shape[0])

            for cat_index,cat in enumerate(self.classes):
                #Calculating prior probability p(c) for each class
                prob_classes[cat_index]=np.sum(self.labels==cat)/float(self.labels.sha

                #Calculating total counts of all the words of each class
                count=list(self.bow_dicts[cat_index].values())
                cat_word_counts[cat_index]=np.sum(np.array(count))+1 # |v| is remainin

                #get all words of this category
                all_words+=self.bow_dicts[cat_index].keys()

            #combine all words of every category & make them unique to get vocabulary

            self.vocab=np.unique(np.array(all_words))
            self.vocab_length=self.vocab.shape[0]

            #computing denominator value
            denoms=np.array([cat_word_counts[cat_index]+self.vocab_length+1 for cat_in



            '''
            Now that we have everything precomputed as well, its better to organize ev
            rather than to have a separate list for every thing.

            Every element of self.cats_info has a tuple of values
            Each tuple has a dict at index 0, prior probability at index 1, denominato
            '''

            self.cats_info=[(self.bow_dicts[cat_index],prob_classes[cat_index],denoms[
            self.cats_info=np.array(self.cats_info)


    def getExampleProb(self,test_example):
        likelihood_prob=np.zeros(self.classes.shape[0]) #to store probability w.r.t ea

        #finding probability w.r.t each class of the given test example
        for cat_index,cat in enumerate(self.classes):

            for test_token in test_example.split(): #split the test example and get p

                #get total count of this test token from it's respective training dict
                test_token_counts=self.cats_info[cat_index][0].get(test_token,0)+1

                #now get likelihood of this test_token word
                test_token_prob=test_token_counts/float(self.cats_info[cat_index][2])

                #remember why taking log? To prevent underflow!
                likelihood_prob[cat_index]+=np.log(test_token_prob)

        # we have likelihood estimate of the given example against every class but we
        post_prob=np.empty(self.classes.shape[0])
        for cat_index,cat in enumerate(self.classes):
            post_prob[cat_index]=likelihood_prob[cat_index]+np.log(self.cats_info[cat_

        return post_prob


    def test(self,test_set):

        predictions=[] #to store prediction of each test example
        for example in test_set:

            #preprocess the test example the same way we did for training set exampels
            cleaned_example=preprocess_string(example)

            #simply get the posterior probability of every example
            post_prob=self.getExampleProb(cleaned_example) #get prob of this example f

            #simply pick the max value and map against self.classes!
            predictions.append(self.classes[np.argmax(post_prob)])

        return np.array(predictions)
    def print_data(self):
        print("Bow Dict",self.bow_dicts)
        print("Outer Bow type",type(self.bow_dicts))
        print("Inner Bow type",type(self.bow_dicts[0]))
        print("Bow Dict Shape",self.bow_dicts.shape)
        print("Bow Dict indx:0 ",self.bow_dicts[0])
        print("Bow Dict indx:1 ",self.bow_dicts[1])
```

In [126]:

```
import numpy as np
#x = ["This was an awesome movie",
#     "Great Movie! I liked it a lot.",
#     "Happy ending! Awesome acting by the hero",
#      "Loved it! Truly great",
#     "bad. not upto mark",
#      "could have been better",
#     "surely a disappointing movie"]

#y = [1,1,1,1,0,0,0]
```

In [91]:

```
#y_labels = np.unique(y)
```

In [92]:

```
#nb = NaiveBayes(y_labels)
```

In [93]:

```
#nb.train(x,y)
```

```
Init Bow Dict [defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8D1F0>, {})
 defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8D3A0>, {})]
[False False False False  True  True  True]
Cleaned Ex 1:  ['bad not upto mark', 'could have been better', 'surely a disappointing movie']
Cleaned Ex 1 type:  <class 'list'>
Cleaned Ex 2:                                 0
0             bad not upto mark
1        could have been better
2  surely a disappointing movie
Cleaned Ex 2 type:  <class 'pandas.core.frame.DataFrame'>
Ex 1:  ['bad not upto mark']
is instance executed
Ex 2:  bad not upto mark
dict indx: 0
Ex 1:  ['could have been better']
is instance executed
Ex 2:  could have been better
dict indx: 0
Ex 1:  ['surely a disappointing movie']
is instance executed
Ex 2:  surely a disappointing movie
dict indx: 0
Cleaned Ex 1:  ['this was an awesome movie', 'great movie i liked it a lot ', 'happy ending awesome acting by
reat']
Cleaned Ex 1 type:  <class 'list'>
Cleaned Ex 2:                                            0
0                 this was an awesome movie
1             great movie i liked it a lot
2  happy ending awesome acting by the hero
3                      loved it truly great
Cleaned Ex 2 type:  <class 'pandas.core.frame.DataFrame'>
Ex 1:  ['this was an awesome movie']
is instance executed
Ex 2:  this was an awesome movie
dict indx: 1
Ex 1:  ['great movie i liked it a lot ']
is instance executed
Ex 2:  great movie i liked it a lot
dict indx: 1
Ex 1:  ['happy ending awesome acting by the hero']
is instance executed
Ex 2:  happy ending awesome acting by the hero
dict indx: 1
Ex 1:  ['loved it truly great']
is instance executed
Ex 2:  loved it truly great
dict indx: 1
```

In [88]:

```
#nb.print_data()
```

```
Bow Dict [defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8DAF0>, {'bad': 
k': 1, 'could': 1, 'have': 1, 'been': 1, 'better': 1, 'surely': 1, 'a': 1, 'disappointing': 1, 'movie': 1})
 defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8D310>, {'this': 1, 'was'
'movie': 2, 'great': 2, 'i': 1, 'liked': 1, 'it': 2, 'a': 1, 'lot': 1, 'happy': 1, 'ending': 1, 'acting': 1, '
'loved': 1, 'truly': 1})]
Outer Bow type <class 'numpy.ndarray'>
Inner Bow type <class 'collections.defaultdict'>
Bow Dict Shape (2,)
Bow Dict indx:0  defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8DAF0>, {
1, 'mark': 1, 'could': 1, 'have': 1, 'been': 1, 'better': 1, 'surely': 1, 'a': 1, 'disappointing': 1, 'movie':
Bow Dict indx:1  defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F595F8D310>, {
1, 'awesome': 2, 'movie': 2, 'great': 2, 'i': 1, 'liked': 1, 'it': 2, 'a': 1, 'lot': 1, 'happy': 1, 'ending': 
e': 1, 'hero': 1, 'loved': 1, 'truly': 1})
```

In [110]:

```
#a = {"b":2,"c":3}
#d = defaultdict(lambda:"Not Present")
#d["a"]=1
#d["b"]=2
#l1 = list(d.values())
#print(l1)
#l2 = list(d)
#print(l2)
#print(type(d))
```

```
[1, 2]
['a', 'b']
```

In [115]:

```
from sklearn.datasets import fetch_20newsgroups
```

In [116]:

```
categories=['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
newsgroups_train=fetch_20newsgroups(subset='train',categories=categories)
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (https://ndownloader.figshare.com/files/5975967) (14 MB
```

In [117]:

```
train_data=newsgroups_train.data #getting all trainign examples
train_labels=newsgroups_train.target #getting training labels
```

In [127]:

```
print ("Total Number of Training Examples: ",len(train_data)) # Outputs -> Total Numbe
print ("Total Number of Training Labels: ",len(train_labels)) # Outputs -> #Total Numb
```

```
Total Number of Training Examples:  2257
Total Number of Training Labels:  2257
```

In [128]:

```
nb=NaiveBayes(np.unique(train_labels)) #instantiate a NB class object
print ("---------------- Training In Progress --------------------")
```

```
---------------- Training In Progress --------------------
```

In [129]:

```
nb.train(train_data,train_labels) #start tarining by calling the train function
print ('----------------- Training Completed ---------------------')
```

```
Init Bow Dict [defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F59D840430>, {})
 defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F59D840D30>, {})
 defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F59D840EE0>, {})
 defaultdict(<function NaiveBayes.train.<locals>.<listcomp>.<lambda> at 0x000001F59D840670>, {})]
----------------- Training Completed ---------------------
```

In [130]:

```
newsgroups_test=fetch_20newsgroups(subset='test',categories=categories) #loading test
test_data=newsgroups_test.data #get test set examples
test_labels=newsgroups_test.target #get test set labels

print ("Number of Test Examples: ",len(test_data)) # Output : Number of Test Examples:
print ("Number of Test Labels: ",len(test_labels)) # Output : Number of Test Labels:
```

```
Number of Test Examples:  1502
Number of Test Labels:  1502
```

In [131]:

```
pclasses=nb.test(test_data) #get predcitions for test set

#check how many predcitions actually match original test labels
test_acc=np.sum(pclasses==test_labels)/float(test_labels.shape[0])

print ("Test Set Examples: ",test_labels.shape[0]) # Outputs : Test Set Examples:  150
print ("Test Set Accuracy: ",test_acc*100,"%") # Outputs : Test Set Accuracy:  93.8748
```

```
Test Set Examples:  1502
Test Set Accuracy:  93.87483355525966 %
```

In [ ]: