HengLiEnShaun_task1

August 26, 2021

[1]:
```
import pandas as pd
import numpy as np
import scipy as sp
from collections import defaultdict
import re

# Reading the dataset
dfw = pd.read_csv('wordsList', sep=' ', header=None)
dfw.columns = ['wordList']

dfc = pd.read_csv('classList', sep=" ", header=None)
dfc.columns = ['classList']

# add classList column to dfw
dfw['classList'] = dfc['classList']
print(dfw.shape[0])

# To print the first 5 rows of wordsList
print(dfw.head())

# to print the first 5 rows of classList
print(dfc.head())
```

```
72
                                              wordList  classList
0  codeine,15mg,for,203,visa,only,codeine,methylm…          1
1  peter,with,jose,out,town,you,want,meet,once,wh…          0
2  hydrocodone,vicodin,brand,watson,vicodin,750,1…          1
3  yay,you,both,doing,fine,working,mba,design,str…          0
4  you,have,everything,gain,incredib1e,gains,leng…          1
   classList
0          1
1          0
2          1
3          0
4          1
```

[2]:
```
print(dfw.shape)
print(dfw.classList.value_counts())
```

```
(72, 2)
1    37
0    35
Name: classList, dtype: int64
```

[3]:
```
#Stratified sampling as requested from the question
import random
def train_test_split(X, y, test_size):

    if isinstance(test_size, float):
        test_size = round(test_size * len(X))

    indices = X.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)

    test_X = X.loc[test_indices]
    train_X = X.drop(test_indices)

    if isinstance(test_size, float):
        test_size = round(test_size * len(y))

    indices = y.index.tolist()
    test_indices = random.sample(population=indices, k=test_size)

    test_y = y.loc[test_indices]
    train_y = y.drop(test_indices)

    return train_X, test_X, train_y, test_y

X = dfw.wordList
y = dfc.classList
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.08)
print("Training no of rows X: "+str(len(X_train)))
print("Testing no of rows X: "+str(len(X_test)))
print("Training no of rows y: "+str(len(y_train)))
print("Testing no of rows y: "+str(len(y_test)))
```

```
Training no of rows X: 66
Testing no of rows X: 6
Training no of rows y: 66
Testing no of rows y: 6
```

[4]:
```
def preprocess_string(str_arg):
    #cleaned_str=re.sub('[^a-z\s]+',' ',str_arg,flags=re.IGNORECASE)
    #every char except alphabets is replaced
```

```
    #cleaned_str=re.sub('(\s+)',' ',cleaned_str)
    #multiple spaces are replaced by single space
    cleaned_str=str_arg.lower()
    #converting the cleaned string to lower case

    return cleaned_str # returning the preprocessed string
```

[5]:
```
class NaiveBayes:

    def __init__(self,unique_classes):

        self.classes=unique_classes
        # Constructor is simply passed with unique number of classes of the␣
↪training set


    def addToBow(self,example,dict_index):

        if isinstance(example,np.ndarray): example=example[0]

        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.shape[0])])

        #only convert to numpy arrays if initially not passed as numpy arrays -␣
↪else its a useless recomputation

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

        #constructing BoW for each category
        for cat_index,cat in enumerate(self.classes):

            all_cat_examples=self.examples[self.labels==cat]
            #filter all examples of category == cat

            #get examples preprocessed
```

```
            cleaned_examples=[preprocess_string(cat_example) for cat_example in␣
↪all_cat_examples]

            cleaned_examples=pd.DataFrame(data=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.
↪shape[0])

            #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(list(self.
↪bow_dicts[cat_index].values())))+1 # |v| is remaining to be added

            #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 -V- of entire training set

        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_index,cat in enumerate(self.classes)])

        self.cats_info=[(self.
↪bow_dicts[cat_index],prob_classes[cat_index],denoms[cat_index]) for␣
↪cat_index,cat in enumerate(self.classes)]
        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 each class
```

```
        #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 of each test word
                #This loop computes : for each word w [ count(w|c)+1 ] / [␣
↪count(c) + |V| + 1 ]

                #get total count of this test token from it's respective␣
↪training dict to get numerator value
                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 need posterior probility
        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_index][1])

        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 for both classes
```

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

        return np.array(predictions)
```

[6]:
```
nb=NaiveBayes(np.unique(y_train))
nb
```

[6]:
```
<__main__.NaiveBayes at 0x1fa9028ff40>
```

[7]:
```
nb.train(X_train,y_train)
```

[8]:
```
pclasses=nb.test(X_test) #get predcitions for test set

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

print ("Test Set Examples: ",y_test.shape[0])
print ("Test Set Accuracy: ",test_acc*100,"%")
```

```
Test Set Examples:  6
Test Set Accuracy:  66.66666666666666 %
```

[ ]: