MINORS Lab 4: Naïve Bayes

Varun Kamath

2019110023

19/11/2022

1. Upload csv files and import libraries

```
from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd
import io
from sklearn import preprocessing
from sklearn.naive_bayes import MultinomialNB

import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.style.use('ggplot')
```

```
Mounted at /content/drive
```

```
df = pd.read_csv('/content/drive/MyDrive/data/amazon_alexa.tsv', sep='\t', low_memory=False)
df.shape
```

```
(3150, 5)
```

2. Analyse the data by finding out data types of each attribute and their

```
[4] df.info()
    print(df['feedback'].value_counts())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   rating            3150 non-null   int64 
 1   date              3150 non-null   object
 2   variation         3150 non-null   object
 3   verified_reviews  3150 non-null   object
 4   feedback          3150 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 123.2+ KB
1    2893
0     257
Name: feedback, dtype: int64
```

3. Pre-process each review by removing spaces and non alphabetical data. Lower the case of all the words present in the review for simplicity

```
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=cleaned_str.lower() #converting the cleaned string to lower case

    return cleaned_str
```

4. Define function for classifying reviews into binary classes.

```
class NaiveBayes:

    def __init__(self,unique_classes):
        self.classes=unique_classes


    def addToBow(self,example,dict_index):
        if isinstance(example,np.ndarray): example=example[0]
        for token_word in example.split():
            self.bow_dicts[dict_index][token_word]+=1
```

5. Create a training function which will train the Naive Bayes Model i.e compute a BoW for each category/class.

```
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])])
    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)

    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_cat_examples]
        cleaned_examples=pd.DataFrame(data=cleaned_examples)
        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):

        prob_classes[cat_index]=np.sum(self.labels==cat)/float(self.labels.shape[0])

        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

        all_words+=self.bow_dicts[cat_index].keys()

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

    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)
```

6. Create a function that estimates posterior probability of the given test example and a function to determine probability of each test example against all classes and predicts the label against which the class probability is maximum

```
def getExampleProb(self,test_example):
    likelihood_prob=np.zeros(self.classes.shape[0])

    for cat_index,cat in enumerate(self.classes):

        for test_token in test_example.split():

            test_token_counts=self.cats_info[cat_index][0].get(test_token,0)+1

            test_token_prob=test_token_counts/float(self.cats_info[cat_index][2])

            likelihood_prob[cat_index]+=np.log(test_token_prob)

    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=[]
    for example in test_set:
        cleaned_example=preprocess_string(example)
        post_prob=self.getExampleProb(cleaned_example)
        predictions.append(self.classes[np.argmax(post_prob)])
    return np.array(predictions)
```

7. Divide the attributes into dependent and independent variables. Using sklearn library, split the model into 75% train and 25% test data and calculate accuracy

```
[7] y_train=df['feedback'].values
    x_train=df['verified_reviews'].values
    print ("Unique Classes: ",np.unique(y_train))
    print ("Total Number of Training Examples: ",x_train.shape)
```

```
Unique Classes:  [0 1]
Total Number of Training Examples:  (3150,)
```

```
[12] from sklearn.model_selection import train_test_split
     from collections import defaultdict
     import re
     train_data,test_data,train_labels,test_labels=train_test_split(x_train,y_train,shuffle=True,test_size=0.25,random_state=42,stratify=y_train)
     classes=np.unique(train_labels)
```

```
nb=NaiveBayes(classes)
print ("Training Examples: ",train_data.shape)
nb.train(train_data,train_labels)
pclasses=nb.test(test_data)
test_acc=np.sum(pclasses==test_labels)/float(test_labels.shape[0])
print ("Test Set Examples: ",test_labels.shape[0])
print ("Test Set Accuracy: ",test_acc)
```

```
Training Examples:  (2362,)
Test Set Examples:  788
Test Set Accuracy:  0.934010152284264
```

8. Calculate confusion matrix to see the false positive and false negative (wrong) predictions

```
[17] from sklearn.metrics import confusion_matrix

     confusion_matrix(pclasses, test_labels)

     array([[ 22,  10],
            [ 42, 714]])
```

Conclusion:

- Successfully trained a model to predict whether feedback provided is positive or negative using Naïve bayes.
- This is an example of Bernoulli Naïve Bayes since the feedback is being classified into 2 classes