Q2

April 15, 2020

a) The category of each text article must depend on the meaning of its content. Explain why Naive-Bayes assumption is not too unrealistic for text categorization problem.

Ans. Bag of Words concepts(text categorization problems) treat each word individually and the order in which the words occur does not matter. Using the naive-bayes assumption of conditional independence, a lot of the dependence among features can be explained away by the underlying class.

b)Train the model based on the training data by MLE. For each class k among 4 different classes, you should learn the parameter (j|y)=k, which is the conditional probability p(x(j) |y = k). You should also learn the parameter (y=k), which is the prior probability of each class p(y = k). Report the confusion matrix and training accuracy by predicting the class labels of the training set by your trained Bernoulli Naive-Bayes model.

Ans.

[1]:
```
import os
import re
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
import matplotlib.pylab as plt
import scipy.sparse as sparse
import math
import sklearn.metrics as metrics
```

[2]:
```
os.chdir('C:\\Users\\dhana\\Courses\\MSBA\\Spring_Sem\\IDS575\\Assignment3')
```

[3]:
```
def data(file):
    datafile = open(file, 'r')
    dataLines = datafile.readlines()
    strip = [x.rstrip("\n") for x in dataLines]
    dataTokens = [re.split(":| ",x) for x in strip]
    Y = [int(x[0]) for x in dataTokens]
    Xlist = [np.reshape(np.array(x[1:len(x)],dtype=np.int32),(-1,2)) for x in␣
 ↪dataTokens] #word,count array
    Xlist = [np.insert(Xlist[x],0,x,axis=1) for x in␣
 ↪range(len(Xlist))]#appending instance number
    Xdata = np.concatenate(tuple(Xlist),axis=0) #concatenating all the instance␣
 ↪arrays to one array
```

```
    X = csr_matrix((Xdata[:,2], (Xdata[:,0], Xdata[:,1])))#sparse matrix
    Bernoulli_X = csr_matrix((np.ones((Xdata.shape[0],),dtype=np.int32),
↪(Xdata[:,0], Xdata[:,1])))# bernoulli csr_matrix (word in doc implies 1 else
↪0)
    return([np.array(Y),X,Bernoulli_X,Xdata])
```

[4]:
```
train = data("articles.train")
test = data("articles.test")
```

[5]:
```
Ytrain = train[0]-1
Ytest = test[0]-1
Xtrain = train[1][:,1:]
Xtest = test[1][:,1:]
Berno_Xtrain = train[2][:,1:]
Berno_Xtest = test[2][:,1:]
```

[6]:
```
def count_unique(array):
    unique, count = np.unique(np.asarray(array), return_counts=True)
    return(dict(zip(unique, count)))
```

[7]:
```
def split(cdata):
    cdata1 = cdata[:1000,:]
    cdata2 = cdata[1000:2000,:]
    cdata3 = cdata[2000:3000,:]
    cdata4 = cdata[-1000:,:]
    return([cdata1,cdata2,cdata3,cdata4])
```

[8]:
```
splitM = split(Berno_Xtrain)
frequencyM = split(Xtrain)
```

[9]:
```
def BernoulliNB(csr,X):
    prior=[]
    for i in range(4):
        prior.append(csr[i].sum(axis=0))

    if prior[0].shape[1]<X.shape[1]:
        for i in range(4):
            prior[i]=np.hstack((prior[i],np.zeros((1,X.shape[1]-prior[i].
↪shape[1]))))

    cprob=[]
    for i in range(4):
        cprob.append(X.multiply(np.log((prior[i])/(1000))).tocsr())

    return(np.hstack([cprob[0].sum(axis=1),cprob[1].sum(axis=1),cprob[2].
↪sum(axis=1),cprob[3].sum(axis=1)]))
```

[10]:
```
metrics.confusion_matrix(Ytrain,np.
↪argmax(BernoulliNB(splitM,Berno_Xtrain),axis=1))/10
```

```
C:\Users\gitap\.conda\envs\bas575\lib\site-packages\ipykernel_launcher.py:12:
RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```

[10]:
```
array([[ 99.9,   0. ,   0. ,   0.1],
       [  0.1,  99.2,   0.3,   0.4],
       [  0.2,   0.2,  99.5,   0.1],
       [  0. ,   0. ,   0. , 100. ]])
```

[11]:
```
metrics.confusion_matrix(Ytest,np.
↪argmax(BernoulliNB(splitM,Berno_Xtest),axis=1))/6
```

```
C:\Users\gitap\.conda\envs\bas575\lib\site-packages\ipykernel_launcher.py:12:
RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```

[11]:
```
array([[100.        ,   0.        ,   0.        ,   0.        ],
       [ 94.33333333,   5.33333333,   0.        ,   0.33333333],
       [ 92.        ,   0.        ,   7.83333333,   0.16666667],
       [ 94.5       ,   0.        ,   0.16666667,   5.33333333]])
```

[12]:
```
def MultinomialNB(csr,X):
    prior=[]
    for i in range(4):
        prior.append(csr[i].sum(axis=0))

    if prior[0].shape[1]<X.shape[1]:
        for i in range(4):
            prior[i]=np.hstack((prior[i],np.zeros((1,X.shape[1]-prior[i].
↪shape[1]))))

    cprob=[]
    for i in range(4):
        cprob.append(X.multiply(np.log(prior[i]/(prior[i].sum(axis=1)))).
↪tocsr())

    return(np.hstack([cprob[0].sum(axis=1),cprob[1].sum(axis=1),cprob[2].
↪sum(axis=1),cprob[3].sum(axis=1)]))
```

[13]:
```
def MultinomialNB_laplace(csr,X):
    prior=[]
    for i in range(4):
        prior.append(csr[i].sum(axis=0))
```

```
    if prior[0].shape[1]<X.shape[1]:
        for i in range(4):
            prior[i]=np.hstack((prior[i],np.zeros((1,X.shape[1]-prior[i].
↪shape[1]))))

    cprob=[]
    for i in range(4):
        cprob.append(X.multiply(np.log(prior[i]+1/(prior[i].
↪sum(axis=1)+51949))).tocsr())

    return(np.hstack([cprob[0].sum(axis=1),cprob[1].sum(axis=1),cprob[2].
↪sum(axis=1),cprob[3].sum(axis=1)]))
```

[14]:
```
metrics.confusion_matrix(Ytrain,np.
↪argmax(MultinomialNB(frequencyM,Xtrain),axis=1))/10
```

```
C:\Users\gitap\.conda\envs\bas575\lib\site-packages\ipykernel_launcher.py:12:
RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```

[14]:
```
array([[99.9,  0.1,  0. ,  0. ],
       [ 0.1, 99.8,  0.1,  0. ],
       [ 0.2,  0.4, 99.4,  0. ],
       [ 0. ,  0.1,  0. , 99.9]])
```

[15]:
```
metrics.confusion_matrix(Ytest,np.
↪argmax(MultinomialNB(frequencyM,Xtest),axis=1))/6
```

```
C:\Users\gitap\.conda\envs\bas575\lib\site-packages\ipykernel_launcher.py:12:
RuntimeWarning: divide by zero encountered in log
  if sys.path[0] == '':
```

[15]:
```
array([[100.        ,   0.        ,   0.        ,   0.        ],
       [ 94.33333333,   5.5       ,   0.16666667,   0.        ],
       [ 92.        ,   0.16666667,   7.83333333,   0.        ],
       [ 94.66666667,   0.        ,   0.16666667,   5.16666667]])
```

c) Learn the model parameters again by performing Laplace smoothing (in Lecture Notes #09a). Report the new confusion matrix and training accuracy when predicting on training data. Report another confusion matrix and test accuracy when predicting on test data. (Note: You cannot report test statistics without Laplace smoothing because there are unseen words in the test data as we experienced at Problem 5 in Homework 2)

Ans.

[16]:
```
def BernoulliNB_laplace(csr,X):
    prior=[]
    for i in range(4):
```

```
        prior.append(csr[i].sum(axis=0))

    if prior[0].shape[1]<X.shape[1]:
        for i in range(4):
            prior[i]=np.hstack((prior[i],np.zeros((1,X.shape[1]-prior[i].
↪shape[1]))))

    cprob=[]
    for i in range(4):
        cprob.append(X.multiply(np.log((prior[i]+1)/(1002))).tocsr())

    return(np.hstack([cprob[0].sum(axis=1),cprob[1].sum(axis=1),cprob[2].
↪sum(axis=1),cprob[3].sum(axis=1)]))
```

[17]:
```
metrics.confusion_matrix(Ytrain,np.
↪argmax(BernoulliNB_laplace(splitM,Berno_Xtrain),axis=1))/10
```

[17]:
```
array([[ 95.1,   0.1,   0. ,   4.8],
       [  0.3,  83.7,   0.1,  15.9],
       [  0.4,   0.1,  94.3,   5.2],
       [  0. ,   0. ,   0. , 100. ]])
```

[18]:
```
metrics.confusion_matrix(Ytest,np.
↪argmax(BernoulliNB_laplace(splitM,Berno_Xtest),axis=1))/6
```

[18]:
```
array([[77.33333333,  0.33333333,  0.16666667, 22.16666667],
       [ 0.33333333, 63.        ,  0.16666667, 36.5       ],
       [ 0.33333333,  0.        , 76.66666667, 23.        ],
       [ 0.        ,  0.16666667,  0.5       , 99.33333333]])
```

d) Report part (c) with multinomial Naive-Bayes model. Report correspondingly to part (c).

Ans.

[19]:
```
def MultinomailNB_laplace(csr,X):
    prior=[]
    for i in range(4):
        prior.append(csr[i].sum(axis=0))

    if prior[0].shape[1]<X.shape[1]:
        for i in range(4):
            prior[i]=np.hstack((prior[i],np.zeros((1,X.shape[1]-prior[i].
↪shape[1]))))

    cprob=[]
    for i in range(4):
        cprob.append(X.multiply(np.log(prior[i]+1/(prior[i].
↪sum(axis=1)+51949))).tocsr())
```

```
    return(np.hstack([cprob[0].sum(axis=1),cprob[1].sum(axis=1),cprob[2].
↪sum(axis=1),cprob[3].sum(axis=1)]))
```

[20]:
```
metrics.confusion_matrix(Ytrain,np.
↪argmax(MultinomialNB_laplace(frequencyM,Xtrain),axis=1))/10
```

[20]:
```
array([[ 99.7,   0. ,   0. ,   0.3],
       [  0.1,  96.7,   0.2,   3. ],
       [  0.3,   0.1,  99.2,   0.4],
       [  0. ,   0. ,   0. , 100. ]])
```

[21]:
```
metrics.confusion_matrix(Ytest,np.
↪argmax(MultinomialNB_laplace(frequencyM,Xtest),axis=1))/6
```

[21]:
```
array([[87.66666667,  0.33333333,  1.33333333, 10.66666667],
       [ 0.5       , 82.66666667,  0.        , 16.83333333],
       [ 0.83333333,  0.16666667, 91.        ,  8.        ],
       [ 0.33333333,  0.66666667,  0.33333333, 98.66666667]])
```

e) Compare and contrast the results from part (c) and (d). Justify why one works better than the other in our dataset. Explain, more in general, the weakness of NaıveBayes models by comparing Bernoulli event model and multinomial event model. (Hint: Think about what happen if the same word occurs multiple times in an article)

Ans. The multinomial laplace