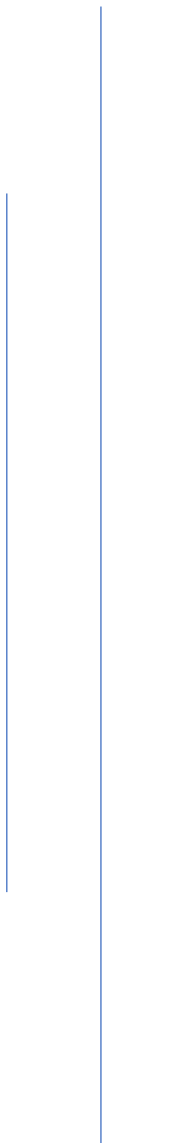


Project 2 :

Classifiers



Saphal Karki

Rajesh Upadhayaya

Karan Aryal

Naive Bayes Classifier:

It is one of the algorithms for supervised learning in machine learning. Naive Bayes algorithm is a classification technique based on Bayes theorem which predicts predictors with an assumption of independence. In simple definition, these assumptions made are independent where a present of one feature does not affect the other. Therefore, given the name naive. It is a popular classification technique which works with large data set, making it simple and easy to build.

Here, in this project we are given a large dataset of 20 newsgroup. The main objective of the project is to classify the given data of approximately 20,000 newsgroups accurately into 20 different newsgroups. In the topic below, quick description of how the program works is demonstrated with some snippets of accuracy obtained when we ran the testing data to test our algorithm.

Description of how the program works:

First when running the program, the model is first trained using the supplied training data, and then it is used to properly forecast newsgroups using testing data. First and foremost, the data from the dataset is imported into the data frame. We also construct a count table using the provided data, which is a compressed version of the table. From the supplied dataset, we construct training and testing data frames. Following that, the value of random variable X_i at the i position is tallied to predict the document's Y value. The dataset is then transformed into a sparse matrix. Method `mle` and `map_function` calculates the formula stated in the documentation and accurately predicts the y value for the newsgroups present for a particular newsletter.

Snippet of the code:

```
import math
import pandas as pd
from scipy.sparse import csr_matrix, lil_matrix

# Here we are importing data in csv format.
# countTable is compressed table form of the given raw data
data = pd.read_csv('countTable.csv')
data_test = pd.read_csv('testing.csv')

# Creating dataframe for training and testing data
df = pd.DataFrame(data=data)
df_test = pd.DataFrame(data=data_test)

# Here we drop ID for training data and count the number of  $X_i$  in  $Y_k$ 
df.drop('1',axis=1,inplace=True)
df2 = df.copy()
df2.drop('count',axis=1,inplace=True)
df['totalcount']=df2.sum(axis=1)

# Here we dropped IDs to create csr matrix
df_test.drop('12001',axis=1,inplace=True)
df_test.drop('14',axis=1,inplace=True)

# Converting into csr sparse matrix
df_matrix=csr_matrix(df.values)
df_test_matrix=csr_matrix(df_test.values)

# We take the dot product of the testing sparse matrix and MAP matrix
# It is not the final prediction.
predict = df_test_matrix.dot(transposed_matrix).toarray()

# This array holds all the predicted class of the testing data.
result = []

# Here we add each value of the MLE to the initial prediction.
# The highest probability gives the classification of the class.
for i in range(len(predict)):
    args = []
    for j in range(len(predict[i])):
        args.append(PY[j]+predict[i][j])
    result.append(args.index(max(args))+1)

# Creating CSV to submit in Kaggle
ID = []
for x in range(12002, 18775):
    ID.append(x)
df3 = pd.DataFrame(data=ID, columns=['id'])
df3['class']=result
df3.to_csv('prediction.csv',index=False)
```

Logistic Regression Classifier:

Logistic regression is an algorithm used for categorical data which has dependent variable. This algorithm is one of the simplest and commonly used machine learning algorithms which shines for binary classification problems. Here, the value of Y is modeled using a function that gives output between 0 and 1 for all values of X. After training of the model with the help of training data, evaluation of the performance is noted based on testing data. For the effective knowledge of performance, we use confusion matrix. Confusion matrix is simply defined as a table that is used to describe the performance for the classification model on a set of test data for which the values are already known. In this assignment, we are given a huge dataset of news article. The primary objective of this project is to train the newsgroup for a certain article and then accurately predict the test data. Brief description of the logic in the code is described below with some snippets of the python code.

Description of how the program works:

First and foremost, we train the model with the training data which was provided from the project description and later the uses testing data to accurately predict newsgroup. Here we used scipy library from python specifically called softmax to calculate the higher values of e in a fast and efficient way. Methods: `y_matrix_encode`: calculates the dataframe and creates a matrix for all values in the given classifier, `lastPartofFormula`: formulae method for calculating the ending, `updateW`: returns the updated value of W and classifier perform calculations and then prints out the predicted value in a csv format.

Snippet of the code:

```
def lastPartofFormula(X, Y, W, lambda):
    const1 = X @ W
    # taking softmax
    softmax_var= softmax(const1.toarray(),axis=1)
    X_T=X.transpose()
    # formula to update w
    #X_T is the transpose of X matrix for multiplaciton.
    # \
    #mue= \
    # delat is Y coming from Y_matrix_encode function()

    last_term = (X_T @ (Y-softmax_var)) - lambda * W
    return last_term
def updateW(X, Y, atea=0.01, mu=0.001):
    # initializing the W to be of row= 61188 and colum= 20.
    W= lil_matrix(np.zeros((61188,20)))
    # creating a matrix of Y whose shape is equal to row 61188 and column is 20 for multiplication.
    encoded_class_matrix=y_matrix_encode(Y_df)
    # creating a while loop in order to minize the error.
    for z in range(0,1000):
        # updating the value of W by multipling it with eta
        W += atea * lastPartofFormula(X, encoded_class_matrix, W, mu)
    # returning the vlaue of W from this function.
    return W

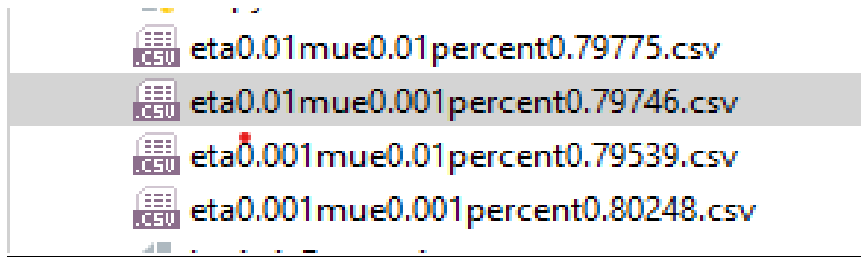
# method to predict the result value classifier.
def classifier(feature):
    const2 = feature @ W
    softmax_var2=softmax(const2.toarray(),axis=1)
    # result data class array
    class_data=(np.argmax(softmax_var2, axis=1))+1
    # id class value
    ID = []
    for x in range(12002, 18775):
        ID.append(x)
    df3 = pd.DataFrame(data=ID, columns=['id'])
    df3['class']=class_data
    # saving the result class of finalResult.csv file
    df3.to_csv("eta0.01mue0.001percent0.79746.csv",index=False)
    return(df3)

# reading the training.csv file using pandas.read_csv
X_train=pd.read_csv('training.csv')
# creating a dataframe
X_df = pd.DataFrame(data=X_train)
# Dropping of the id colum
X_df.drop('1',axis=1,inplace=True)
# training vlaue assigning to a Y_df variable
Y_df =pd.DataFrame(X_df['14'])
# creating a sparse matrix of type lil_matrix
Y=lil_matrix(Y_df.values)
# reshaping Y matrix for multiplication.
Y.reshape(11999,1)
# dropping the prediction colum from the feature as it is assign to another matrix
X_df.drop('14',axis=1,inplace=True)
# creating the sparse matrix of row 1199 and colum 61188.
X=lil_matrix(X_df.values)
# reading the testing file.
data2=pd.read_csv('testing.csv')
# converting trainign file to a dataframe
df_testing= pd.DataFrame(data2)
# dropping index colum from dataframe.
df_testing.drop('12001',axis=1,inplace=True)
# converting into a sparse matrix of type lil_matrix as it doesnot store 0.
x_preicate_matrix=lil_matrix(df_testing.values)
# Updated value of W from our model as it is assign to zero vlaues from above method.
W = updateW(X, Y)
# classifying the classes using the classifier function.
result=classifier(x_preicate_matrix)

print("printing the resulting class of 6673 rows as numpy array")
print(result)
```

Accuracy Obtained:

From different values of eta and lambda.



eta0.01mue0.01percent0.79775.csv
eta0.01mue0.001percent0.79746.csv
eta0.001mue0.01percent0.79539.csv
eta0.001mue0.001percent0.80248.csv

Answer to Question 1 to 7:

Question 1: In your answer sheet, explain in a sentence or two why it would be difficult to accurately estimate the parameters of this model on a reasonable set of documents (e.g., 1000 documents, each 1000 words long, where each word comes from a 50,000-word vocabulary)?

Answer:

If the document provided is a reasonable i.e., of only 1000 documents and 1000 long. The size of the document would be very small. Which makes it hard to keep track of the proper calculations that are made in the individual X_i classes. Being very small, the size of each probability would result into smaller and smaller values, which makes the value approximately close to 0. Thus, this makes it very difficult to predict any type of newsgroup.

Question 2: Re-train your naïve bayes classifier for values of B between 0.00001 and 1. Report the accuracy over the test set for each value of B . Create a plot with values of B on the x axis and accuracy on the y axis. Use a logarithmic scale for the x axis (in Matlab, `semilogx` command). Explain in a few sentences why the accuracy drops for both small and large value of B .

Answer: In the formula we can see that, $\alpha = 1 + \beta$, also conditional probability is inversely proportional to $(1/(\alpha - 1))$. If β increases, then the conditional probability decreases whereas if β decreases then the conditional probability increases. Thus, increase or decrease in

the value of beta, results into the change in conditional probability of a given word. Hence, the accuracy is also affected by this measure.

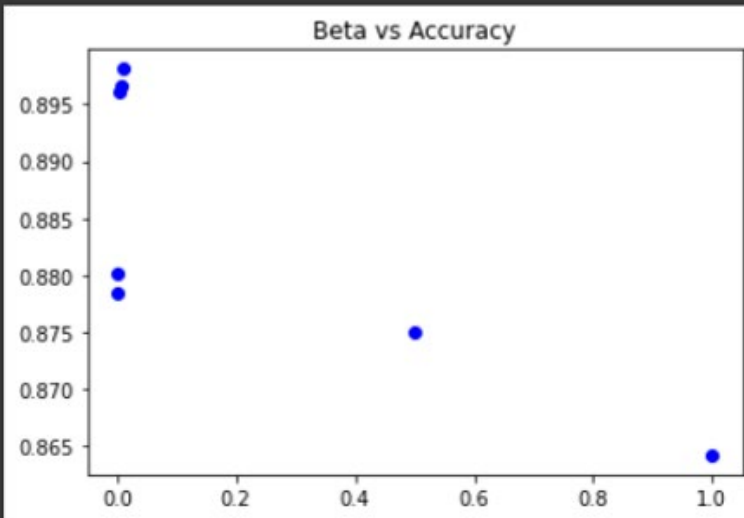
Plot and Accuracy Obtained:

```
import numpy as np
from matplotlib import pyplot as plt

# x is the value of beta
x = np.array([1, 0.5, 0.01, 0.008, 0.005, 0.00001, 0.00001634307])
# y is the accuracy of the beta
y = np.array([0.86418, 0.87502, 0.89813, 0.89666, 0.89607, 0.87835, 0.88012])

plt.plot(x, y, 'ob')

plt.title("Beta vs Accuracy")
plt.show()
```

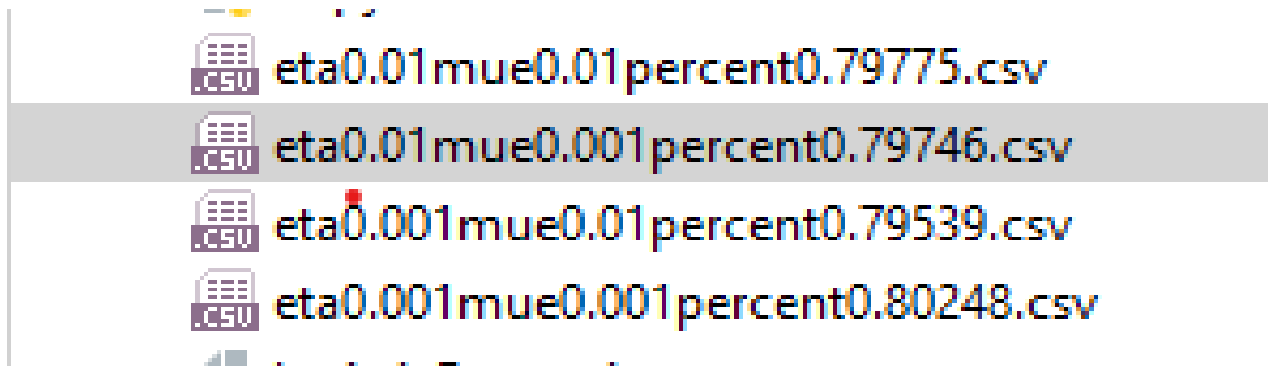


Question 3: Re-train your logistic Regression classifier for values of n starting from 0.01 to 0.001, $\lambda = 0.01$ to 0.001 and vary your stopping criterium from number of total iterations (e.g. 10,000) or $= 0.00001$ and report the accuracy over the test set for each value (this is more effective if you plot your parameter value vs accuracy). Explain in a few sentences your observations with respect to accuracies and sweet spots.

Answer: To obtain very accurate model, what we found was if the iterations were done number of time ranging from (1000 to 10,000) then the accuracy would get more better and better. But

with only 1000 iteration, the computation was so gpu heavy, it took a lot of time just to complete 1000 iterations. So, in our project we were only able to do it twice. Also, whenever the value of eta was low the accuracy was much better. Its also the same case with lamba, the lesser the value of lambda, the better the accuracy. Below chart shows the value of each eta, lamda (mue) and accuracy relationship.

Chart showing the accuracy, lambda, mue relationship:












Question 4: In your answer sheet, report your overall testing accuracy (Number of correctly classified documents in the test set over the total number of test documents), and print out the confusion matrix. (The matrix C, where C_{ij} is the number of times a document with ground truth category j was classified as category i).















Answer:

Team name: rajesh295

The accuracy obtained from logistic regression classifier which is 0.802 :

#	Team	Members	Score	Entries	Last	Code
1	The Prediction	 	0.87983	1	1d	
2	Susmita Garai + Andres Qua n	 	0.87894	1	2d	
3	Ken Plackowski	 	0.86064	34	2h	
4	DataHallucinations		0.85208	37	10h	
5	team_e		0.85178	3	4d	
6	rajesh295		0.80248	5	1s	

The accuracy obtained from naïve bayes classifier which is 0.898:

1	Charlie Foxtrot	 	0.89873	34	4d
2	RuntimeTerror	 	0.89873	10	4d
3	A Team	 	0.89843	4	12d
4	frelen	 	0.89843	7	4d
5	GeolnTheLoop	 	0.89843	27	1d
6	Susmita Garai + Andres Quan	 	0.89843	7	1h
7	Pizza Time		0.89843	2	4h
8	rajesh295		0.89813	13	2d

Confusion matrix of the above classifier:

```
C:\Users\rages\PycharmProjects\UpdatedProject20401t\venv\Scripts\python.exe C:/Users/rages/Desktop/ConfusionMatrix.py
```

```
confusion matrix
```

```
[[481    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0  
   0    2]  
  
[ 0 611    4    4    2    1    1    0    0    0    0    1    0    0    0    0    0    0  
   0    0]  
  
[ 0  1610    7    2    2    0    0    0    0    0    0    0    0    0    0    0    0    0  
   0    0]  
  
[ 0  1 1636    0    1    3    0    0    0    0    0    0    1    0    0    0    0    0  
   0    0]  
  
[ 1  0  0 1598    0    0    0    0    0    0    0    0    2    0    0    0    0    0  
   0    0]  
  
[ 0  2  0  2 1624    0    0    0    0    0    0    0    0    0    0    0    1    0  
   0    0]      .  
  
[ 0  1  0  3  2  0608    1    0    0    1    0    1    0    0    0    0    0    0  
   1    0]  
  
[ 1  0  1  0  0  0 2605    2    1    0    0    1    0    0    0    0    0    0  
   0    1]  
  
[ 0  0  0  0  0  0  0 1648    0    0    0    0    0    0    0    0    0    0  
   0    0]  
  
[ 0  0  0  0  1  0  0  0 1623    3    0    0    0    0    0    0    0    0  
   0    0]  
  
[ 1  0  0  0  0  0  0  0  0 2641    1    1    0    0    0    0    0    0  
   0    0]  
  
[ 0  1  0  0  1  0  0  0  0  0 0635    1    0    0    0    0    0    0  
   1    0]  
  
[ 0  0  0  0  3  2  1  0  0  0  0  0  0  0 0620    0    0    0    0  
   0    0]
```

[illegible]

Question 5: Are there any newsgroups that the algorithm(s) confuse more often than others? Why do you think this is?

Answer:

Yes, there are newsgroups which both the algorithms got confused often. Since, the dataset is very large, it is utmost difficult to exactly identify the exact newsgroup which the algorithm got completely confused and predicted wrong. Then main problem why this is happening as we figured out must be because of the following reasons:

1. A lot of newsletters contained 0 value words,
2. Column values missing and tons of outliers
3. Large dataset, large computation time which created difficult to track which values were right and which values were wrong.

Thus, because of the above reasons both algorithms got confused often for a certain news group.

Question 6: Propose a method for ranking the words in the dataset based on how much the classifier 'relies on' them when performing its classification. Your metric should give high scores to those words that appear frequently in one or a few of the newsgroups but not in all of them. Words that are used frequently in general English (the, of, in, at, etc) should have low scores. Words that appear only extremely rarely in the datasets should have low scores.

Answer:

As we know classification depends upon the conditional probability of a given word.

i.e., $Y(\text{new})$ directly proportional to $P(X_i | Y_k)$ [from the given formula]

Also, conditional probability depends on the no. of sequence of the word,

i.e., $P(X_i | Y_k)$ directly proportional to (count of X_i in Y_k)

Hence, we propose one more dictionary with most of the common words like (to, the, in, at, etc)

Now, we take a highest value from the conditional probability of each word and compare if the word is in our common word dictionary, if it is in the dictionary then we simply ignore the words otherwise the word is one of our determining factors in the model.

Question 7: Implement your method and print out the 100 words with the highest measure.

Answer:

For this implementation, we do have a python code implementing the above method. Also below is the screenshot of the 100 words with the highest measure.

Code snippet:

```
# for confusion matrix
import pandas as pd
from sklearn.metrics import confusion_matrix

# already saved the data from the algorithm NB.
df = pd.DataFrame(pd.read_csv('confusionMatrix_data.csv'))
# converting to numpy array
y_true = df['actual'].to_numpy()
y_pred = df['predicted'].to_numpy()
# printing the result using the confusion matrix of sklearn library
print("confusion matrix")
print(confusion_matrix(y_true, y_pred))

df2= pd.DataFrame(pd.read_csv('countTable.csv'))
txt=pd.DataFrame(pd.read_csv('vocabulary.txt'))
df2.drop('1',axis=1,inplace=True)
df2.drop('count',axis=1,inplace=True)
df3= pd.DataFrame(df2.sum(axis=0))
df3.drop(index='14',axis=0,inplace=True)
df3.drop(index='0',axis=0,inplace=True)
df3['vocabulary']=txt.to_numpy()
maxWord=[]
for x in df3[0].nlargest(100):
    maxWord.append(df3.loc[df3[0]==x]['vocabulary'][0])

print("Printing the 100 words appeared most frequently in the given datasets")
print(maxWord)
```

Print of words:

```
..
Printing the 100 words appeared most frequently in the given datasets
['addresses', 'the', 'to', 'of', 'and', 'in', 'is', 'that', 'it', 'you', 'for', 'this', 'on', 'be', 'not', 'have', 'are', 'with', 'as', 'or', 'if', 'but', 'they', 'was', 'edu', 'from', 'can', 'by', 'at',
'then', 'does', 'time', 'use', 'these', 'should', 'could', 'may', 'new', 'good', 'am', 'because', 'well', 'even', 'now', 'very', 'see', 'see',
```