ML REPORT ASSIGNMENT 2

1) Naive Bayes

1a)

- Used Laplace Smoothing C = 1
- Used Logarithms to compute the probabilities to avoid underflow
- Data Preprocessing
    - Lowercase all the words
    - Split all the words
    - RegEx to remove tags and brackets *“re.sub("[^a-zA-Z0-9]", " ", s)”*

- Used bag of words :- Word appears in the document or not context doesn’t matter

Accuracy in TRAIN = 0.63086
Accuracy in TEST = 0.5996

1b) #Accuracy Majority Prediction = 0.4388
#Accuracy Random = 0.2000

My Algo Gives about **1.5** times improvement over Majority Prediction.
My Algo Gives about **3** times improvement over Random Prediction.

1c)

Actual > Prediction	1	2	3	4	5
1	14260	2762	1336	1093	3188
2	3941	3362	1751	762	350
3	1183	3349	5316	2664	669

4	436	1032	5250	17949	15316
5	349	333	878	6890	39299

Category 5 has the highest value of the diagonal entry.

1. We have correctly classified most values of this class.
2. Test samples are highly of this category.
3. We have learned this category well.

It can be observed from the confusion matrix that :-

1. We are slightly confusing 1 rated text with 2
2. We are highly confusing 2 rated text with 1 and 3
3. We are highly confusing 3 rated text with 4
4. We are confusing 4 rated text with 5
5. We are confusing 5 rated text with 4

3 rated and 2 rated texts don't have a clear majority which signifies their rating in the middle(moderate rating) and the model is also confused to rate it clearly.

#Confusion Matrix [[14260, 2762, 1336, 1093, 3188], #
#[3941, 3362, 1751, 762, 350],
#[1183, 3349, 5316, 2664, 669],
#[436, 1032, 5250, 17949, 15316],
#[349, 333, 878, 6890, 39299]]

1d)

Used Stemming and removed the stopwords Without cleaning data

Accuracy in TRAIN = 0.637
Accuracy in TEST = 0.6068

Accuray on both train and test data increased by 1 percent. Possibly due to the fact stopwords only caused confusion to the model and didn't provide much relevant info about the ratings of the text. And stemming helped in easy classifying to the model as we reduced different forms of a word to a single form.

1e) Feature Engineering

Used Stemming and removed the stopwords without beautifying data(pre processing (lower case and removing tags and numbers))

 # Accuracy in TRAIN = 0.637
 # Accuracy in TEST = 0.6068

Though words like non-starbuck , anti-starbuck , coffee/starbuck were not considered same.

No Stemming Just lowercasing and splitting words with pre-processing

Accuracy in TRAIN = 0.63086
Accuracy in TEST = 0.5996

Used Stemming and removed the stopwords after beautifying data(pre processing (lower case and removing tags and numbers))

Accuracy in TEST = 0.5991

Used Bigrams + Stemming

Test Accuracy - 0.59

Most Users are likely to use two consecutive consecutive words in their reviews. Ex “Burger King”

Used Unigrams+Bigrams + Trigrams

Test Data 47% After Training from 0.3Million Samples

Most Users are highly unlikely to use two consecutive consecutive words in their reviews. Ex A restaurant name is generally 2 words long

Even after incorporating different features I wasn’t able to obtain any significant improvement over the basic model of Naive Bayes. This might be due to the fact that the training set is not equally distributed among classes.

1f) F1 Score = 2* Precision * Recall / (Precision + Recall)
Precision = True +ve / (Ture +ve + False +ve)
Recall = True +ve / (Ture +ve + False -ve)

P1 = 0.62 R1 = 0.70 F1 = 0.65
P2 = 0.33 R2 = 0.31 F2 = 0.31
P3 = 0.40 R3 = 0.36 F3 = 0.37
P4 = 0.44 R4 = 0.61 F4 = 0.51
P5 = 0.82 R5 = 0.66 F5 = 0.73

Macro F1 = 0.51

Macro F1 measure is better than test accuracy in case of multi-class classification as it provides a better representative of our predictions by taking into account the false positives and false negatives rather than just the true positives as in case of test accuracy.

1g) Running the model I Stemming and Removal of StopWords obtained
65% test Accuracy .
And an F1 score of 0.56

Vocabulary Link :
https://drive.google.com/drive/folders/1p0zLqEjP_IO1pQIlEVTi_WiEaItSTjco?usp=sharing

2) SVM

Scaling Down of values was done from [0,255] to [0,1]
1a) Entry No last digit 0 :
TRAIN Accurray = 100 %
TEST Accuracy = 99.90 %

// Calculation of w
p2 = (zo_label.reshape(-1,1)*alphas).transpose()
w = np.dot(p2,zo_samples)

```
// Calculation of b
found is the index of alphas which is between 0 and C.
b =  zo_label[found] - np.dot(w,(zo_samples[found].reshape(-1,1)))

42 support vectors
```

1b) Entry No last digit 0 :

```
TRAIN  Accurray  = 100 %
TEST Accuracy = 99.81 %
```

Depending Upon

```
// Kernel Function 1 Called while making the kernel matrix
def gaussian_kernel(x, y, gamma = 0.05):
    return np.dot(np.subtract(x,y),np.subtract(x,y))


// Kernel Function Called while testing
def gaussian_kernel2(x, y, gamma = 0.05):
    return np.exp(-linalg.norm(x-y)**2 * gamma)

// Kernel Matrix
kernel_matrix = np.zeros((no_of_samples, no_of_samples))
for i in range(no_of_samples):
    for j in range(no_of_samples):
        print i,j
        kernel_matrix[i][j] = gaussian_kernel(zo_samples[i], zo_samples[j])

730 #SupportVectors
```

For calculation of w we would have to calculate the phi function of the gaussian kernel which is tough, so instead we directly calculate w^(T)x .

```
// Calculation of wTx
p2 = (zo_label.reshape(-1,1)*alphas).transpose()
w = np.dot(p2,zo_samples)

// Calculation of b
```

found is the index of alphas which is between 0 and C.
b = zo_label[found] - np.dot(w,(zo_samples[found].reshape(-1,1)))

Gaussian Kernel gave slightly lower accuracy than the linear kernel which might be due to the overfitting of the gaussian.

1c) Entry No last digit 0 :

Linear Kernel
TRAIN Accurray = 100 %
TEST Accuracy = 99.9 %
nsV = 53
In part a) i obtained 42 #svectors with 0.0001 as cutoff.

Gaussian Kernel
TRAIN Accurray = 100 %
TEST Accuracy = 99.8 %
nSV = 745
In part a) i obtained 730 #svectors with 0.0001 as cutoff.
Training time for LIBSVM is about 1000 times smaller than that of CVXOPT.

2a)

- Made 10C2 = 45 Classifiers using CVXOPT and stored them in an array (stored the Alphas , b , trains_samples , train_samples_label , non_zero_alphas_index)
- A list of dictionary was made to store the count of prediction of length len(test_samples).
- Using Each classifier Predicted all the test_samples and stored the prediction in the dictionary corresponding to the test_sample.
- Finally predicted the digit with the maximum count.

2b) Train Acc : 99.7 % Test Acc : 97.24 %

- Made 10C2 = 45 Classifiers using LIBSVM and stored them in an array
- A list of dictionary was made to store the count of prediction of length len(test_samples).
- Using Each classifier Predicted all the test_samples and stored the prediction in the dictionary corresponding to the test_sample.
- Finally predicted the digit with the maximum count.

2c) C = 1

Actual > Predtn /	0	1	2	3	4	5	6	7	8	9
0	969	0	4	0	0	2	6	1	4	4
1	0	1122	0	0	0	0	3	4	0	4
2	1	3	1000	8	4	3	0	19	3	3
3	0	2	4	985	0	6	0	2	10	8
4	0	0	2	0	962	1	4	4	3	13
5	3	2	0	4	0	866	4	0	5	4
6	4	2	1	0	6	7	939	0	1	0
7	1	0	6	6	0	1	0	987	3	9
8	2	3	15	5	2	5	2	2	942	12
9	0	1	0	2	8	1	0	9	3	952

Only Digits which appear alike are confused with each other by the model.
We can observe that Following are slightly confused with the other
2 with 8 , 3 with 2 , 7 with 2, 8 with 3 , 9 with 4.
An explanation to this might be their similar appearance in Handwritten cases.

2d)

	-5	-3	0	0.69	1
Validation	71.25	71.25	97	97.25	97.25
Test	71.76	71.76	97.11	97.24	97.24

We get maximum accuracy for C = 10 for both validation and test data.
The Curves are almost overlapping with each other. As both the data are seen by the model.