COL774
Assignment 2
Manupriya Gupta
2018CS10355

QUESTION 1

PART 1 A

To run :
./run.sh 1 a col774_yelp_data/train.json col774_yelp_data/test.json output1a.txt

I copied the functions given in utils.py in the main file do_question1a.py
I read the files lines by line from train.json and test.json to get training data & labels and test data & labels.

The class Naive Bayes has a method to fit the training data and corresponding label classes. As a first step, all means, variances and then priors are calculated in turn for each class. Then for making predictions, class-conditional probability is added to the prior to obtain the posterior. The class with highest posterior probability is predicted for the given sample. No built-in functions are used except for numpy array utilities.

On test set :
Accuracy = 59.21 %
Time taken = 221.5 seconds

On training set :
Accuracy = 67.53 %
Time taken = 783.4 seconds

PART 1 B

To run :
./run.sh 1 b col774_yelp_data/train.json col774_yelp_data/test.json output1b.txt

Accuracy from Naive Bayes : 59.21 %
Accuracy from random guessing : 20.03 %
Accuracy from majority prediction : 43.88 %

Naive Bayes algo used the same as in PART 1 A.

For random guessing, I generated random numbers using random.randint(0,n) where n is the number of unique classes and each class is enumerated. Then each sample in test set is assigned this random class.
For max probability, I calculate the class with max frequency in the training data and the same class is assigned to all samples in the test set.

So, Naive Bayes definitely outperforms random guessing and majority prediction methods to predict the class of the data. It takes more time but accuracy is much better. However, if a large chunk of data, say about 75% belongs to a particular class, majority prediction might fair better in terms of time complexity with a bit less accuracy than Naive Bayes.

PART 1 C

To run :
./run.sh 1 c col774_yelp_data/train.json col774_yelp_data/test.json output1c.txt

The confusion matrix is nxn dimensional where n is the number of classes

Pred/True	1	2	3	4	5
1	14777	2963	1395	1083	2900
2	3234	2844	1397	528	233
3	1163	3213	4590	1923	411
4	588	1395	6164	18477	14643
5	407	423	985	7347	40635

Class with highest diagonal entry = 5
This class has the best prediction of the samples. We notice that the number of points in training data belonging to this class is also highest among all classes.
We can observe that more the number of samples of a particular class in the training set, higher the diagonal entry and hence better is the prediction. So we can say that the Naive Bayes method is not free from overfitting of data

PART 1 D

To run :
./run.sh 1 d col774_yelp_data/train.json col774_yelp_data/test.json output1d.txt

Instead of using text.split(), I used getStemmedDocuments() for stemming purpose.

New accuracy = 67.98 %
Time taken = Approx 11 mins

There is a slight increase in accuracy here but considerable increase in time. Stemming procedure adds to time consumed but it is essential.

PART 1 E

Alternative Feature 1 :
Store the number of occurrences of each word in the document instead of just 0/1 indicating its presence
Accuracy = 72.13 %
Time taken = Approx 13 mins
[A slight increase in accuracy noticed]

Alternative Feature 2 :
Use of Bigrams
Accuracy = 68.04 %
Time taken = Approx 12 mins
[A slight increase in accuracy]

So, from both features we notice a slight increase in accuracy but that comes at the cost of increased time of execution.

QUESTION 2

0	1	2	3	4	5	6	7	8	9
T-shirt	Trouser	Pull-Over	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Boot

PART 2 A (i)

To run :
./run.sh 2 a 1 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv output2a1.txt

I have done 5 vs 6 binary classification.
The training and test data present in csv files, hence data is read using csv reader. Then using training data and labels, the parameters P, q, G, h, A and b are calculated using the cvxopt package. The way P is calculated, is based on the linear kernel i.e. kernel(X[i],X[j]) = X[i]'X[j].

Accuracy = 99.6 %
Time taken = 210.8 seconds
[I chose support vectors with alpha > 1e-5 to avoid excessive number of support vectors]

Number of support vectors = 1125

PART 2A (ii)

To run :
./run.sh 2 a 2 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv output2a2.txt

Everything is the same as part 2 A (i) except that for calculating P, gaussian kernel is used. Also, since we cannot calculate W explicitly, predictions are directly made using the Gaussian matrix.

Accuracy = 100 %
Time taken = 419.6 seconds
[I chose support vectors with alpha > 1e-5 to avoid excessive number of support vectors]

Number of support vectors = 668

In the gaussian kernel, accuracy improves from 99.6% to 100%, the number of support vectors is also reduced but the time of execution becomes almost double. But overall performance of the gaussian kernel is better.

PART 2B (i)

I used one vs one classification technique and trained kC2 classifiers using the gaussian kernel for cvxopt module. Then, for each sample in test data, that class was chosen which has max number of +1 predictions for that particular sample.

For test set :
./run.sh 2 b 1 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv output2b1.txt

Accuracy = 85.06 %
Time taken = 6 hours approx

For validation set :
./run.sh 2 b 1 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/val.csv output2b1.txt

Accuracy = 85.04 %
Time taken = 6 hours approx

PART 2B (ii)

Test set accuracy = 88.08 %
Total time taken for test set (22500 points) = 479.5836498737335 seconds

Validation set accuracy = 87.92 %
Total time taken for validation set (22500 points) = 500.8104224205017 seconds

To run on the test set :
./run.sh 2 b 2 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv q2_a/output2a.txt

To run on validation set :
../run.sh 2 b 2 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/val.csv q2_a/output2a.txt

Here the accuracy improves compared to part 2 B (i) and time taken is also less. So overall scikit fairs better than the cvxopt solver.

PART 2B (iii)

To run on test set :
./run.sh 2 b 3 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv q2_a/output2a.txt

To run on validation set :
./run.sh 2 b 3 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/val.csv q2_a/output2a.txt

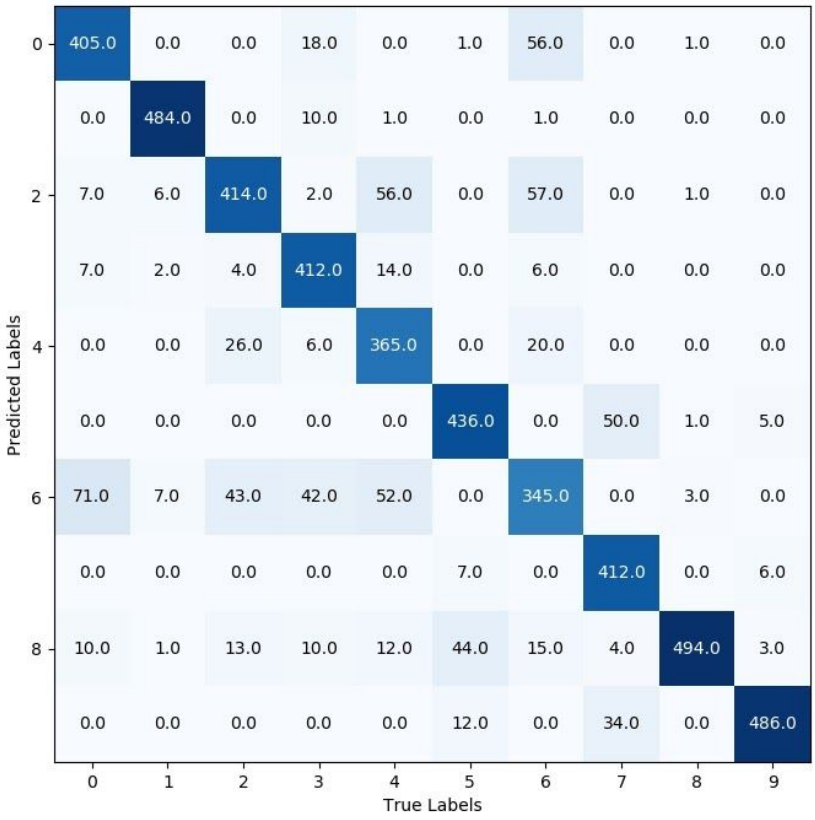
Since, the accuracy by scikit module is less compared to cvxopt module, more misclassification is seen in the confusion matrix of the former.

For test.csv :
Accuracy from scikit = 88.08 % (time taken = 479.6 seconds)
Accuracy from cvxopt = 85.06 % (time taken = approx 6 hours)

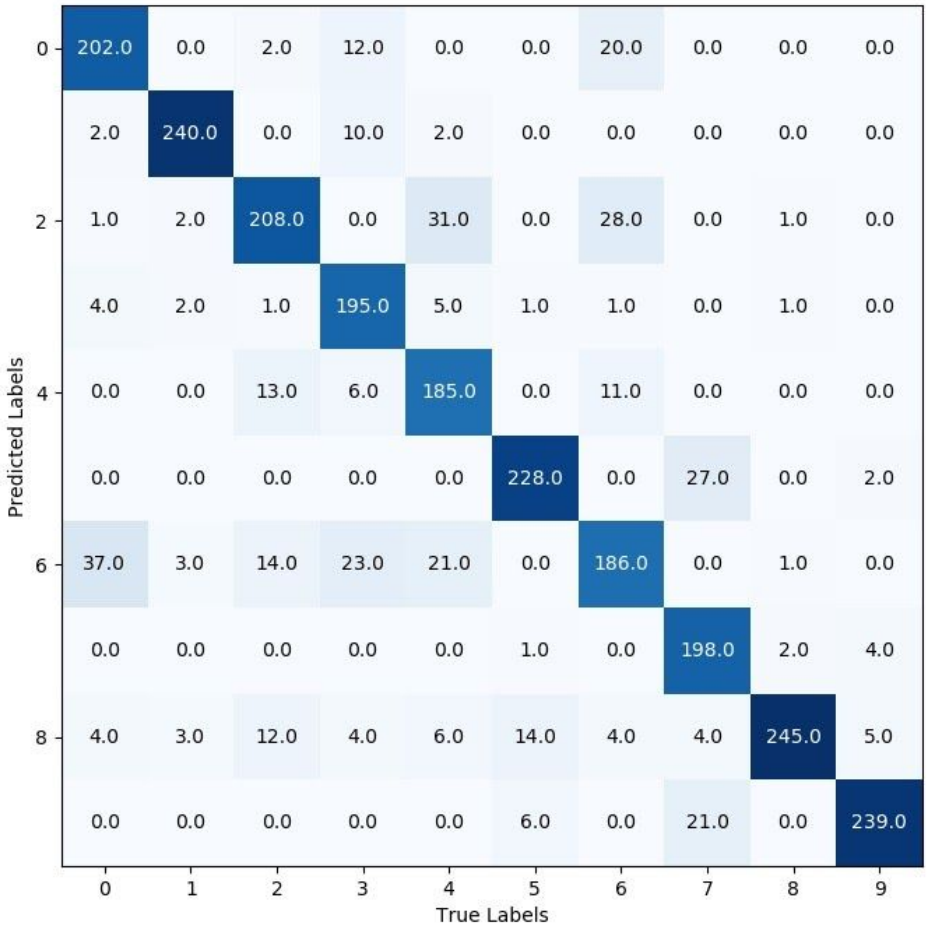
For val.csv :
Accuracy from scikit = 87.92 %
Accuracy from cvxopt = 85.04 % (time taken = approx 6 hours)

The confusion matrices:

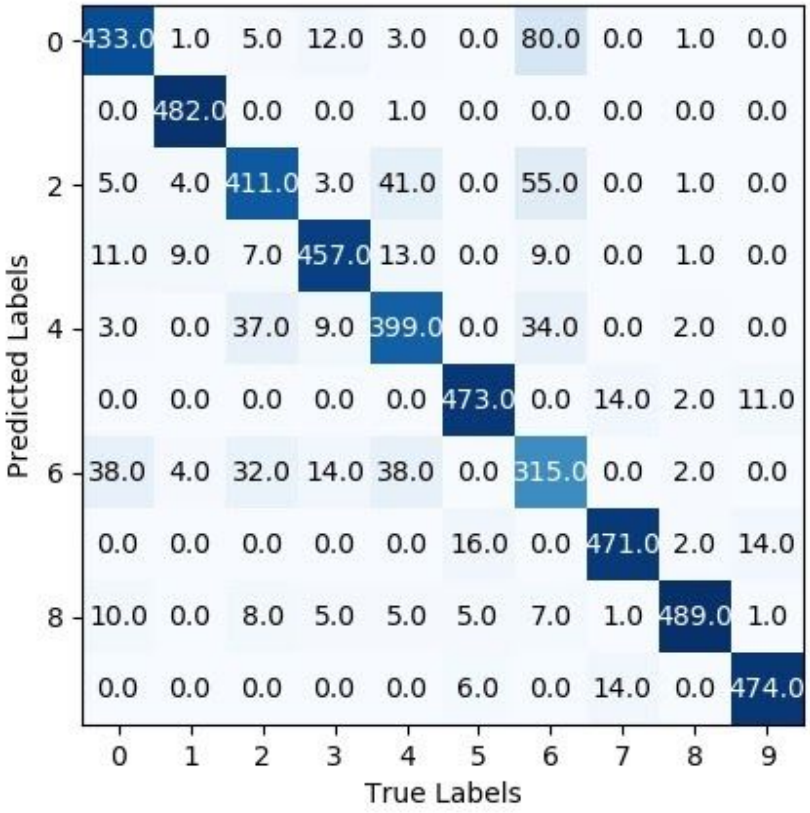
For cvxopt test.csv :



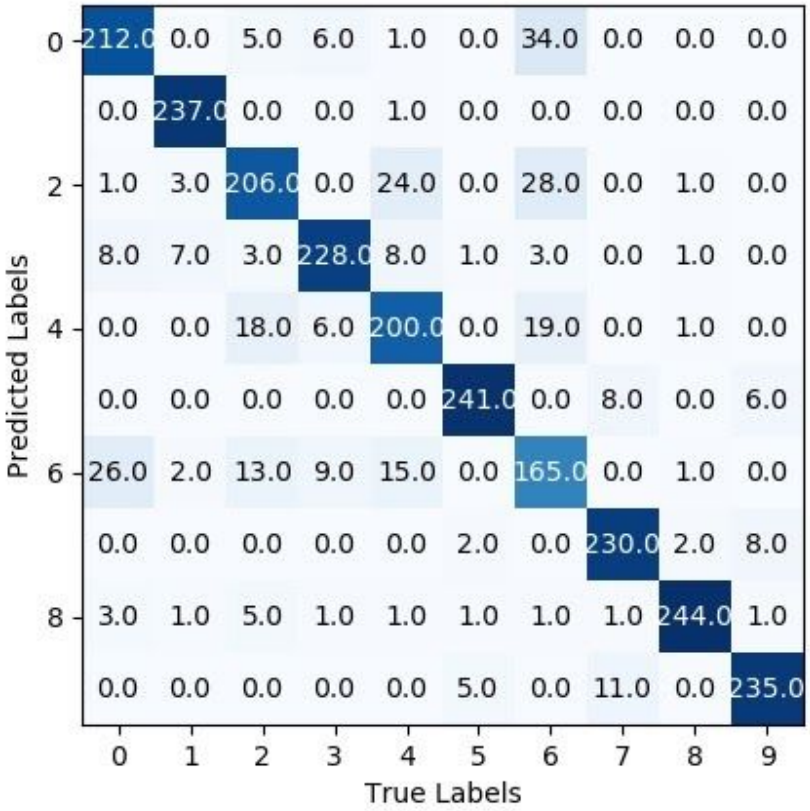
For cvxopt val.csv :



For scikit test.csv :



For scikit val.csv :



The samples which get misclassified the most are the ones which are similar in real life. Label 6 is predicted as Label 0 or 1 quite often because a shirt (6) can look quite similar to a t-shirt(0) or pull-over(2)

PART 2B (iv)

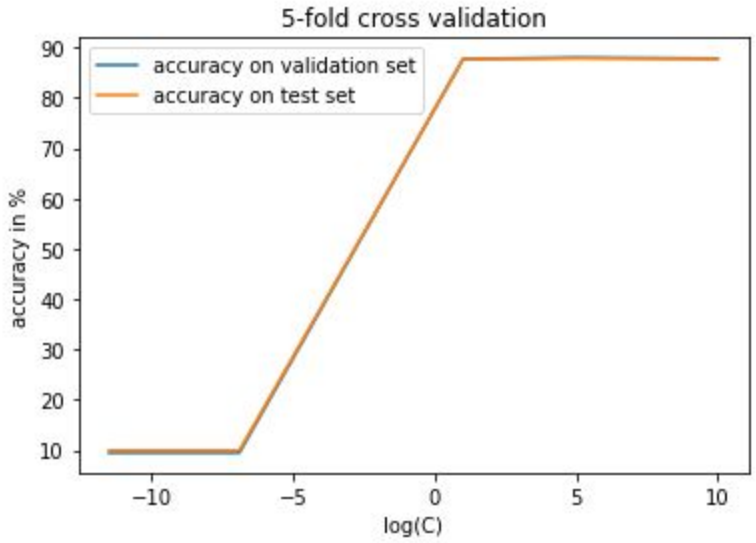
To run :

./run.sh 2 b 4 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv q2_a/output2a.txt

Total time taken was 5.5 hours on intel i7 8th gen processor

Execution logs :

```
manupriya@manupriya-Vostro-3583:~/Desktop/col774/a2$ chmod +x run.sh
manupriya@manupriya-Vostro-3583:~/Desktop/col774/a2$ ./run.sh 2 b 4 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv q2_a/output2a.txt
test : fmnist_data/fashion_mnist/test.csv
test data done
using c = 1e-05
iter : 0
iter : 1
iter : 2
iter : 3
iter : 4
for iteration : 0
0.09475555555555555
0.1
using c = 0.001
iter : 0
iter : 1
iter : 2
iter : 3
iter : 4
for iteration : 1
0.09475555555555555
0.1
using c = 1
iter : 0
iter : 1
iter : 2
iter : 3
iter : 4
for iteration : 2
0.8770222222222221
0.87684
using c = 5
iter : 0
iter : 1
iter : 2
iter : 3
iter : 4
for iteration : 3
0.8799555555555555
0.8782
using c = 10
iter : 0
iter : 1
iter : 2
iter : 3
iter : 4
for iteration : 4
0.8782222222222222
0.8772
best val set : 5
best test set : 5
total time taken = 19022.072350502014
manupriya@manupriya-Vostro-3583:~/Desktop/col774/a2$
```



[Accuracy vs C plot]
[This plot has not been saved in the execution nor is it shown because of the guidelines given on piazza]

We observe from the plot that for a given value of C, validation and test set accuracies are almost the same. There is a steep increase in accuracy as C approaches 1 and it after that it is almost the same for all values of C > 1.

Best cross-validation accuracy is obtained for C = 5, accuracy being 88%
Best accuracy of 87.82 % is obtained on the test set for C = 5
Hence, we get max accuracy for both validation as well as test set for value of C = 5

QUESTION 3

PART 3A

To run :
./run.sh 3 a col774_yelp_data/train.json col774_yelp_data/test.json output3a.txt

The features I used were the presence of a particular word in a document. Stemming and lemmatization was performed while parsing the data.

Time taken by Naive Bayes using sklearn = 1102.1249425411224 seconds
Accuracy = 60.45 %

Time taken by SVM using LibLinear = 4738.5337352752686 seconds
Accuracy = 64.23 %

Overall, the algorithm which is better in my opinion is SVM using Liblinear because it takes more time but is much more accurate.

PART 3B

To run :
./run.sh 3 b col774_yelp_data/train.json col774_yelp_data/test.json output3b.txt

Features are the same as in part 3 A.

For SVM using Liblinear:
Accuracy = 66.23 %
Time taken = 1738.5337352752686 seconds
Best value of C = 0.1 (for max accuracy)

Using SGD :
Accuracy = 68.61 %
Time taken = 134.56 seconds
Max iters = 1000
Tolerance = 1e-3
Learning rate = 0.01 (it is kept constant since optimal mode increases time of execution)

Other configurations of the hyper parameters (increasing max iterations to 10000 or reducing learning rate) significantly increase the time with little improvement in accuracy.

SGD has a lower running time and better accuracy, so it fairs better than SVM using liblinear.

FINAL AUTOGRADING

1. ./run.sh 1 col774_yelp_data/train.json col774_yelp_data/test.json output1.txt
2. ./run.sh 2 fmnist_data/fashion_mnist/train.csv fmnist_data/fashion_mnist/test.csv output2.txt
3. ./run.sh 3 col774_yelp_data/train.json col774_yelp_data/test.json output3.txt