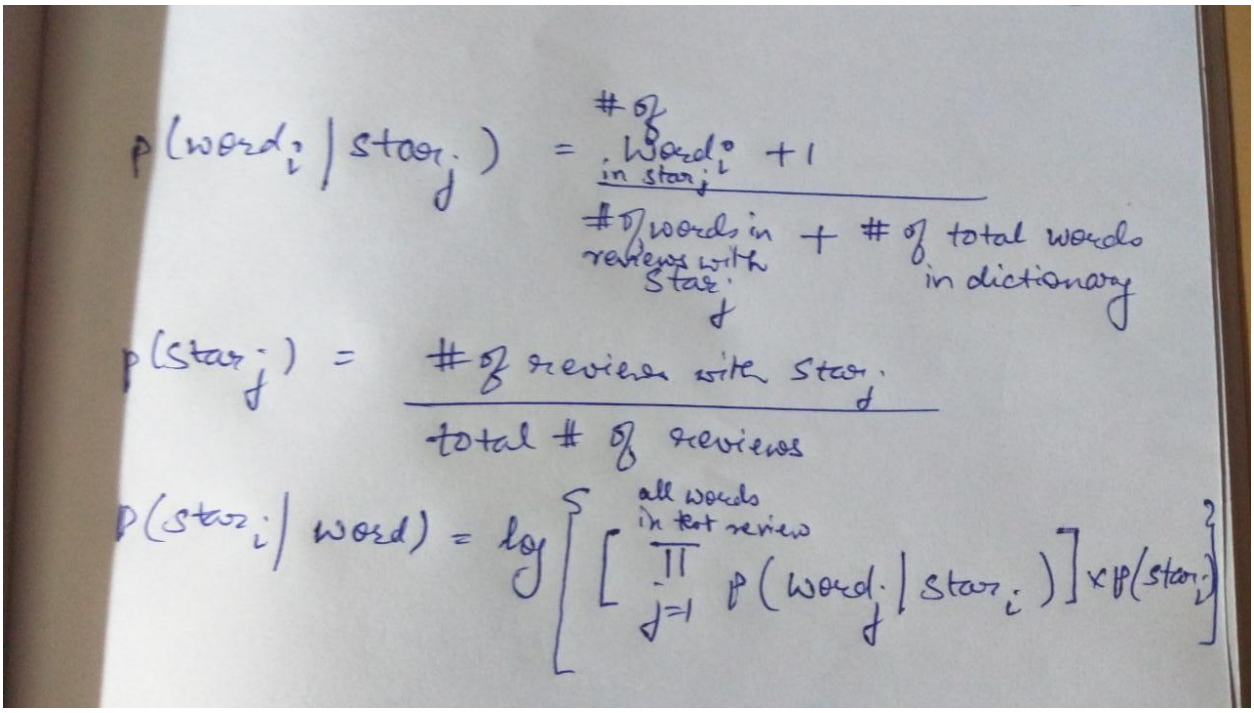
Machine Learning (COL774)

Assignment #2

Tanu Kanvar
(2018EEY7537)

Q1->Part (a)



Train Data Accuracy = 69.64096082801119 %
Test Data Accuracy = 61.7530923286319 %
Time taken to train < 1min

Q1->Part (b)

Random Prediction Accuracy = 14.235929343843013 %
Majority Prediction Accuracy = 43.9895900327555 %
Improvement of algorithm:
Improvement over Random = 47.52%
Improvement over Majority = 17.76%

Q1->Part (c)

The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct prediction

Confusion Matrix

Predicted	0	1	2	3	4	All
Actual						
0	**15850**	1430	947	1226	716	20169
1	**3720**	1291	2284	2843	700	10838
2	1767	474	2477	**8288**	1525	14531
3	1085	112	541	**17997**	9623	29358
4	2309	26	94	11433	**44960**	58822
All	24731	3333	6343	41787	57524	133718

1 and 2 are misinterpreted because 0 and 3 respectively have almost same reviews.
As can be seen in the matrix, for i star, i+1 and i-1 are the most interpreted for i.
As can be seen the probability of a review being 0 is pretty high even in 4 star review. Reason – Number of 0 star reviews are very high.

Q1->Part(d)

Stemming
Test Data Accuracy = 60.852689989380636 %

Since stemming transforms words into their basic forms, I expected the accuracy to increase.
However the accuracy reduced as compared to that in Part(a). The reason is due to the reduced amount of information the classification cannot perform as well as in the case of other methods.

Q1->Part (e)

I used the combination of following features in this part

1) Bigrams - This method converts statements “to be or not to be” to “to be”, “be or”, “or not”, “not to”, “to be” i.e. the vocabulary is combination of 2 words and not single words
2) TD-IDF - One of the most widely used techniques to process textual data is TF-IDF. Full form Term Frequency — Inverse Data Frequency. Calculates frequency of each word in the document. Words most rarely used have the highest weight in calculating star. E.g. ‘the’ ‘a’ occur most but are most insignificant. Words like ‘excellent’ occur least and are most significant.
3) N-grams – Pair of words. Single word vocabulary (1-gram), 2 word vocabulary (bi-gram) and so on

I trained my model on unstemmed data with following 2 sets of features

1) Bigrams and TD-IDF
2) N-grams and TD_IDF
3) Bigrams
4) TD-IDF

Bigrams outperformed all. However since I had to choose 2 features to be used along with stemmed model, the next best pair was bigrams and TD-IDF

Time taken = 2 min

I received accuracy as follows:

1) Unstemmed n-gram (3 pair words) + TD-IDF
   Test Data Accuracy = 60.67844269283118 % (Decreased)
2) Unstemmed bigram + TD-IDF
   Test Data Accuracy = 62.45980346699771 % (Increased)
3) Unstemmed Bigrams
   Test Data Accuracy = 63.74010978327526 % (Increased more)
4) Unstemmed TD-IDF
   Test Data Accuracy = 61.430772222139126 % (Decreased)

Reason for increase in case of bigrams – Words make more sense in pairs. E.g. “not sad” means “happy”. But in 1-gram case “not” will be separate and “sad” will be separate. So accuracy will decrease as the star will be misinterpreted for this case.

N-grams – Since the vocabulary decreased so the information for classification decreased and so the accuracy decreased

Stemmed + Bigram + TD-IDF
Test Data Accuracy = 62.35435767809868 % (Increased as compared to Part (a) and Part(d))

Q1->Part (f)

F1-score also known as balanced F-score or F-measure

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.
Recall is the ratio of correctly predicted positive observations to all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label?

Recall = TP/TP+FN

F1 'micro' - aggregate the contributions of all classes to compute the average metric.
F1 'macro' - compute the metric independently for each class and then take the average (hence treating all classes equally)

In a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance (i.e you may have many more examples of one class than of other classes).

For this type of setup, accuracy and micro behave in the same manner, however macro behaves differently as explained above. Here accuracy will be better for this case as the data is unbalanced across classes.

<u>Unstemmed n-gram</u>
F1 Score for each class = [0.66657604 0.30333817 0.3671483 0.50573949 0.75286189]
macro-F1 Score = 0.5191327791203184
micro-F1 Score = 0.6067844269283118

Unstemmed bi-gram
F1 Score for each class = [0.7639106 0.05782841 0.1007701 0.42422291 0.78739025]
macro-F1 Score = 0.4268244565513126
micro-F1 Score = 0.6245980346699771

Stemmed bigram + tf-idf
F1 Score for each class = [0.75749192 0.08975106 0.13383322 0.43987514 0.78531173]
macro-F1 Score = 0.44125261419186146
micro-F1 Score = 0.6235435767809868

Q1->Part (g)

Unstemmed + Bigram + tf-idf
Test Data Accuracy = 85.27498167785937 %
F1 Score for each class = [0.87787841 0.7218028 0.74911408 0.78163169 0.92076857]
macro-F1 Score = 0.8102391095475368
micro-F1 Score = 0.8527498167785937

Stemmed + Bigram + tf-idf
Test Data Accuracy = 82.20284479277285 %
F1 Score for each class = [0.85310324 0.68265226 0.70839574 0.7405245 0.89842925]
macro-F1 Score = 0.7766209976205832
micro-F1 Score = 0.8220284479277286

I am creating vocabulary during training and keeping the entire vocabulary as is. So training is taking around 9-10 hours. Testing is taking around 30 mins.

Q2->1

Test and Training data set – Digits 7 and 8

Q2->1->Part(a)

$$\begin{aligned} & \min \frac{1}{2} x^T P x + q^T x \\ s.t. \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

With API

```
cvxopt.solvers.qp(P, q[, G, h[, A, b[, solver[, initvals]]]])
```

Recall that the dual problem is expressed as:

$$\max_{\alpha} \sum_{i}^{m} \alpha_i - \frac{1}{2} \sum_{i,j}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)} x^{(j)} >$$

Let $\mathbf{H}$ be a matrix such that $H_{i,j} = y^{(i)} y^{(j)} < x^{(i)} x^{(j)} >$, then the optimization becomes:

$$\begin{aligned} & \max_{\alpha} \sum_{i}^{m} \alpha_i - \frac{1}{2} \alpha^T \mathbf{H} \alpha \\ s.t. \ & \alpha_i \geq 0 \\ & \sum_{i}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

We convert the sums into vector form and multiply both the objective and the constraint by -1 which turns this into a minimization problem and reverses the inequality

$$\begin{aligned} & \min_{\alpha} \frac{1}{2} \alpha^T \mathbf{H} \alpha - 1^T \alpha \\ s.t. \quad & -\alpha_i \leq 0 \\ s.t. \quad & y^T \alpha = 0 \end{aligned}$$

- $P := H$ a matrix of size $m \times m$
- $q := -\vec{1}$ a vector of size $m \times 1$
- $G := -diag[1]$ a diagonal matrix of -1s of size $m \times m$
- $h := \vec{0}$ a vector of zeros of size $m \times 1$
- $A := y$ the label vector of size $m \times 1$
- $b := 0$ a scalar

Calculating w and b

with kernel K *(positive semi-definite by Mercer's theorem)*

$$w = \sum_{i=1}^{m} \alpha_i y_i \phi(x_i)$$

$$b = y_i - \sum_{j=1}^{m} \alpha_j y_j \left\langle \phi(x_j), \phi(x_i) \right\rangle$$

$$= y_i - \sum_{j=1}^{m} \alpha_j y_j K(x_i, x_j)$$

$$b = \frac{1}{N_s} \sum_{s \in S} (y_s - \sum_{m \in S} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s)$$

Prediction

To classify the new test point x, we use *sign(f(x)) where*

$$f(x) = w.\phi(x) + b = \sum_{i=1}^{m} \alpha_i y_i \left\langle \phi(x_i), \phi(x) \right\rangle + b$$

$$= \sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b$$

f(x) >=0 => y = 1
f(x) <= 0 => y = -1

Results – CVXOPT Linear Kernel

Time Taken – 2 minutes
Test Data
Accurate Predictions = 1971
Total Test Set = 2002
Accuracy = 98.45154845154845%
Total number of support vectors = 159

w – can be found in file "BinaryClassificationLinearKernelCvxoptW.txt"
b - can be found in file "BinaryClassificationLinearKernelCvxoptB.txt"
support vectors – can be found in file "BinaryClassificationLinearKernelSV.txt"

Q2->1->Part(b)

CVXOPT Gaussian Kernel
Time Taken – 5 minutes
Test Data

Accurate Predictions = 1988
Total = 2002
Accuracy = 99.3006993006993%
Total number of support vectors = 1362

I have used threads in order to calculate kernel matrix. The use of threads has decreased the running time.

w – w cannot be calculated in Gaussian Kernel as it needs Phi(x) and we don't have that
b - can be found in file "BinaryClassificationGaussianKernelCvxoptB.txt"
support vectors – can be found in file "BinaryClassificationGaussianKernelSV.txt"

The accuracy of gaussian kernel has increased as compared to that of linear kernel because gaussian fits data better. However it might lead to overfitting, but in this case since the digits have similarities in them, the data doesn't seem to be linearly separable. Therefore gaussian kernel doesn't seem to overfit. E.g. number 1 and 9 will have similarity. So they might not be linearly separable.

Q2->1->Part(c)

Linear Kernel Libsvm
Time taken – 1min
Test Data
Total number of support vectors = 159
Accuracy = 98.4515% (1971/2002)

Same accuracy as Part (a) Linear Kernel
The number of support vectors are same

Gaussian Kernel Libsvm
Time Taken – 1min
Test Data
Total number of support vectors = 1323
Accuracy = 99.2507% (1987/2002)

Similar accuracy as Part (b) Gaussian Kernel
The number of support vectors are less and libsvm
Reason – Cvxopt uses kernels and dual formula to solve. However libsvm uses SMO algorithm to solve the same. SMO is much more efficient. It uses convergence method to find lagrange multiplier values.

Q2->2

Testing and Training Data Set – All digits

Q2->2->(a)

I used threads and semaphores in order to increase the run time

Testing Data

Time Taken – 3 hrs (This includes training and testing time)
Time Taken – 1 hr (If all the kernels, support vectors, and other information are prestored. So time is taken only in testing. Since during testing, kernel is calculated with test data, that is taking time)

Since my system is a bit slow (when compared time of other students on all other assignment questions), I think the training time will decrease on a larger core system.

Accurate = 9723
total = 10000
Accuracy = 97.23%

Training Data

Time Taken – 5 hrs (This includes training and testing time)

Accurate = 19984
Total = 20000
Accuracy = 99.92%

Q2->2->(b)

Testing Data

Time Taken – 10 mins
Accuracy = 97.23% (9723/10000)

Training Data

Time Taken = 15 mins
Accuracy = 99.92% (19984/20000)

The time taken for libsvm is negligible as compared to that of cvxopt.
The accuracy is same for both training and testing data

Q2->2->(c)

Confusion Matrix

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	**969**	0	1	0	0	3	4	1	2	0	980
1	0	**1122**	3	2	0	2	2	0	3	1	1135

2	4	0	**1000**	4	2	0	1	6	15	0	1032
3	0	0	8	**984**	0	4	0	6	5	3	1010
4	0	0	4	0	**962**	0	6	0	2	8	982
5	2	0	3	6	1	**866**	7	1	5	1	892
6	6	3	0	0	4	4	**939**	0	2	0	958
7	1	4	19	2	4	0	0	**987**	2	9	1028
8	4	0	3	10	3	5	1	3	**942**	3	974
9	5	4	3	8	13	3	0	9	12	**952**	1009
All	991	1133	1044	1016	989	887	960	1013	990	977	10000

Diagonals represent correct predictions. As we can see in this matrix, most of them are correct predictions.
Digits that have some features common among them are misinterpreted. For example, 2 is very similar to 8 and so those are the most misinterpreted ones in case of digit 2.
But since 2 is quite different from digit 5, there are no misinterpretations for 5 in case of digit 2.
Similar explanation can be given to all the other digits.

2 is most misinterpreted as 8 (16 counts)
7 is most misinterpreted as 1 (19 counts)

Q2->2->(d)

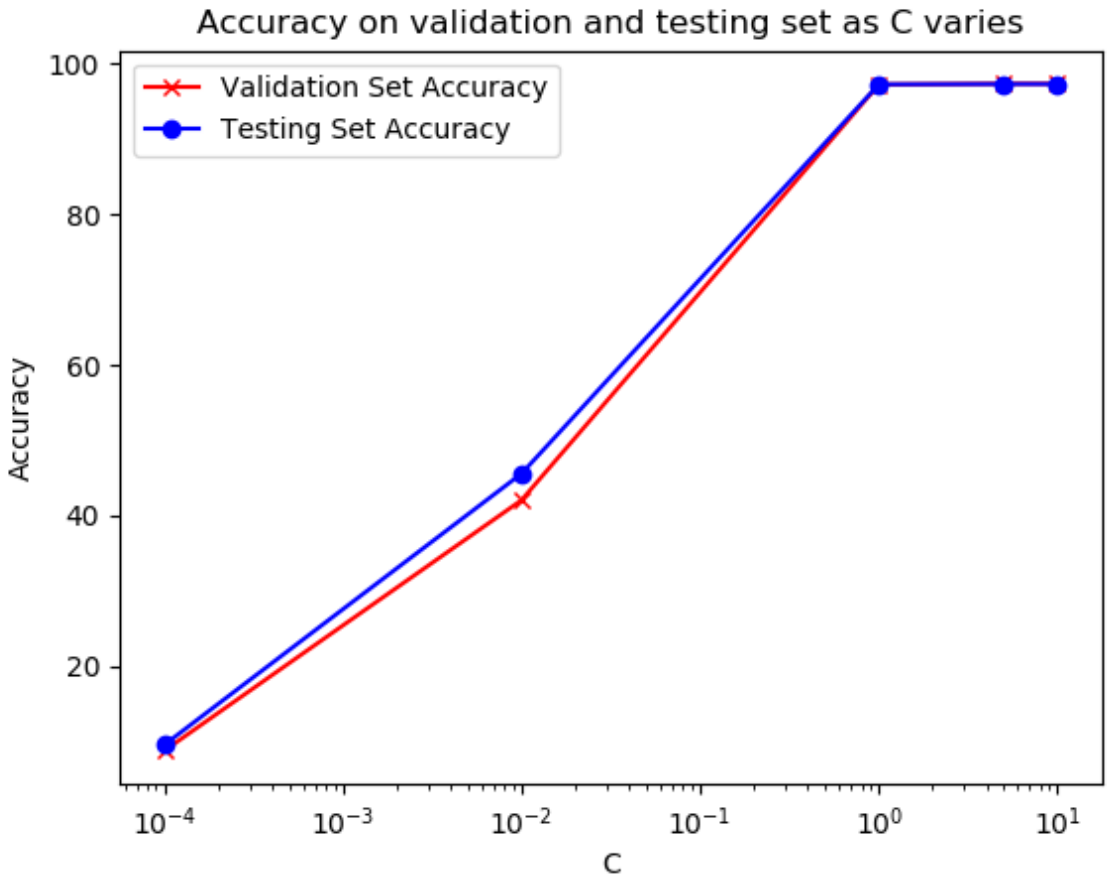
Training for 0.0001
Validation Accuracy = 8.8% (176/2000)
Testing Accuracy = 9.58% (958/10000)

Training for 0.01
Validation Accuracy = 42.05% (841/2000)
Testing Accuracy = 45.57% (4557/10000)

Training for 1
Validation Accuracy = 97.2% (1944/2000)
Testing Accuracy = 97.11% (9711/10000)

Training for 5
Validation Accuracy = 97.3% (1946/2000)
Testing Accuracy = 97.26% (9726/10000)

Training for 10
Validation Accuracy = 97.3% (1946/2000)
Testing Accuracy = 97.26% (9726/10000)



C = 5 gives the best results. This value gives the best results for both validation and testing set out of all the 5 C values. After this the percent accuracy remains same.
C is the value that controls how much weight we should give to error data points. So higher the weight, more accuracy is achieved.
After C > 5, the accuracy is same in this case because after this value of C, the margin couldn't shrink anymore to fit the error cases. However with higher value of C, it might have caused overfitting. But since we tested on Validation set and accuracy is increasing, so we can say it is not the case of overfitting.

Too high C = overfitting possible in the training set.
Too low C = overgeneralisation / underfitting possible.

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. C Parameter is used for controlling the outliers — low C implies we are allowing more outliers, high C implies we are allowing fewer outliers