Machine Learning Week 4 Assignment

Dataset1: # id:2-2--2-1 Dataset2: # id:2-4--2-1

i.

a.

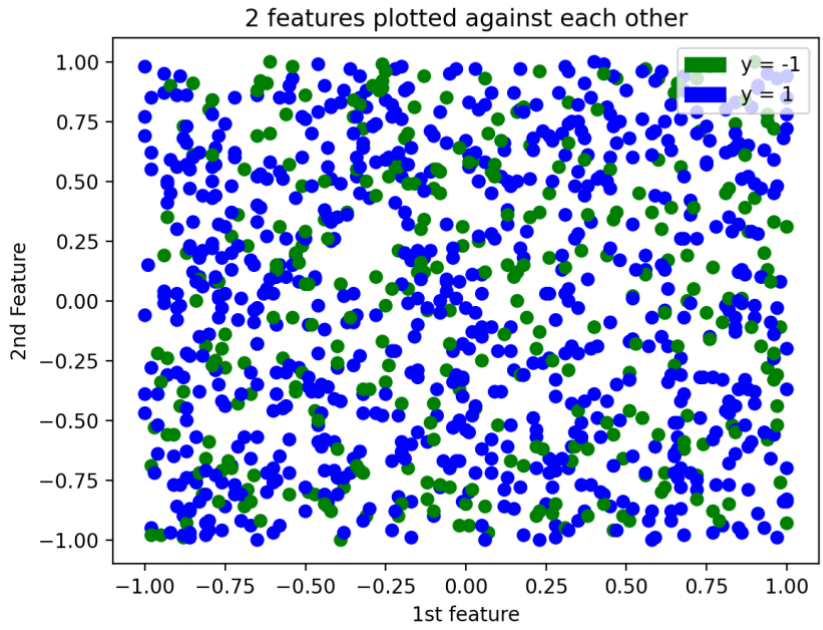

Figure 1

Figure 1 shows the two features plotted against each other for the first set of data. As can be seen from the graph there is no apparent decision boundary between the two classes. The data appears random. I expect these features to be of little use in creating the models.

i.

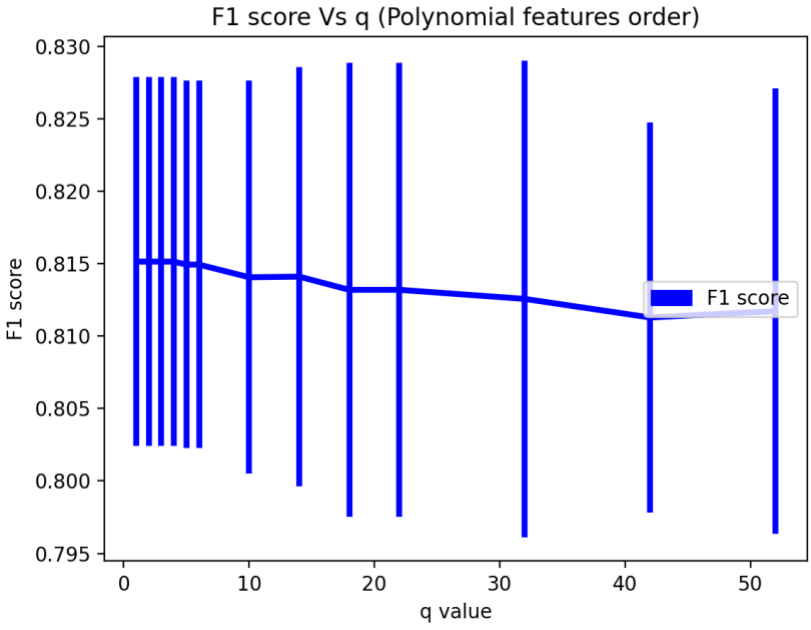

Figure 2

Figure 2 shows the F1 score of the model plotted against the q (order of the polynomial features). Firstly, MSE isn't suited to classifying so the F1 score metric was chosen because it measures the effect of both false positives and false negatives, and it enables the performance of the model to be evaluated with a single metric. Accuracy wouldn't have been a viable option as it suffers from biases in the data, which F1 score doesn't.
It's clear that the optimum q value for the model is 1 since the F1 score decreases with increasing q. As the performance of the classifier is poor, it's a good idea to minimise to complexity of the features and thus the model by picking a small of a q value as possible which is what was done.

ii.

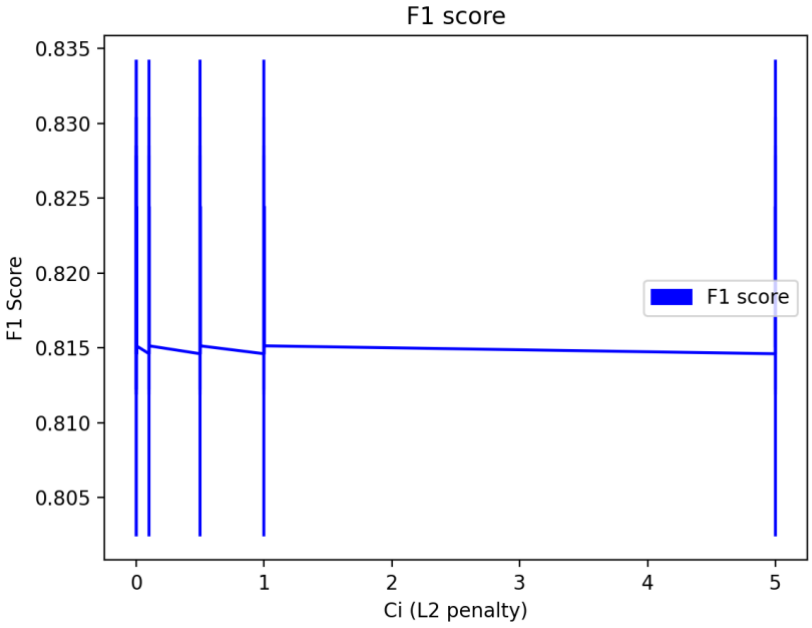

Figure 3

Figure 3 shows the F1 score for the model with the L2 penalty parameter C varied. As can be seen the f1 score remains relatively constant at approximately .81. For this reason a low C value of .0000005 was chosen in order to keep the model complexity as low as possible, as the performance was not as desired.

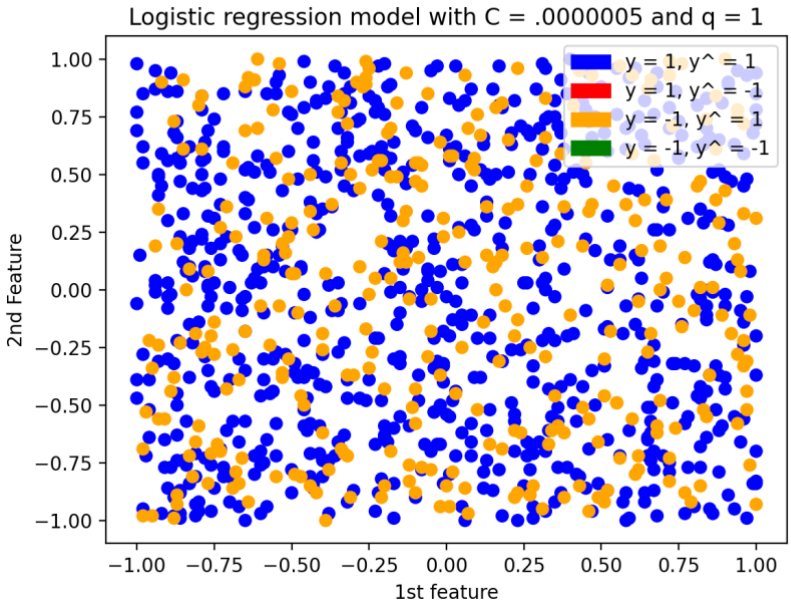

Figure 4

Figure 4 shows the LR model with the tuned hyperparameters of C=1 and q = 2. The blue and green dots are the true positives and trues negatives respectively. The red and orange dots represent the false negatives and false positives respectively. The model predicts only positive values. The plot contains just true positives and false positives. This is equivalent to baseline model predicting the majority class but with the extra complications necessary for LR model. This is not good but was expected due to the lack of decision boundary in the training data.

b.

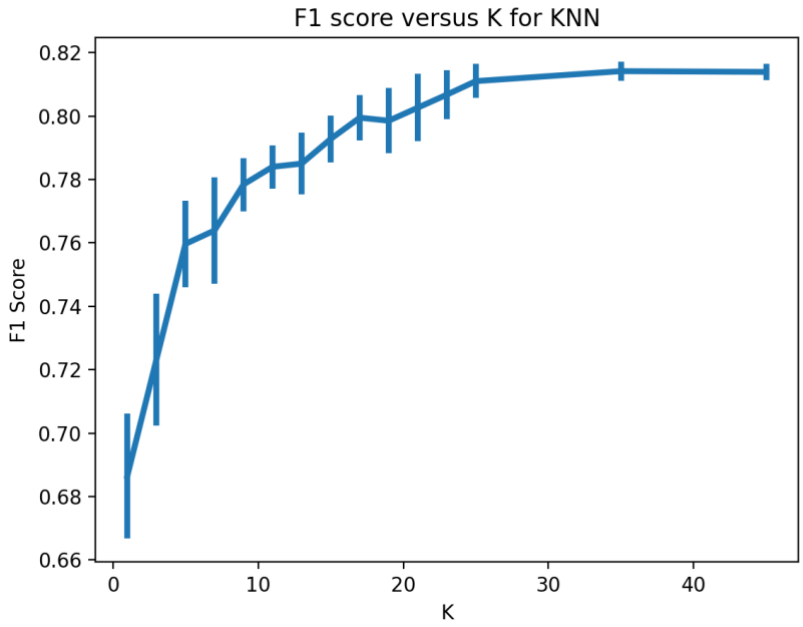

Figure 5

Figure 5 shows the graph of the F1 score of a KNN model versus the K parameter. It rises rapidly initially before it plateaus to an F1 score of roughly .81 at K = 24. A value of 24 was chosen for K to achieve the maximum performance.

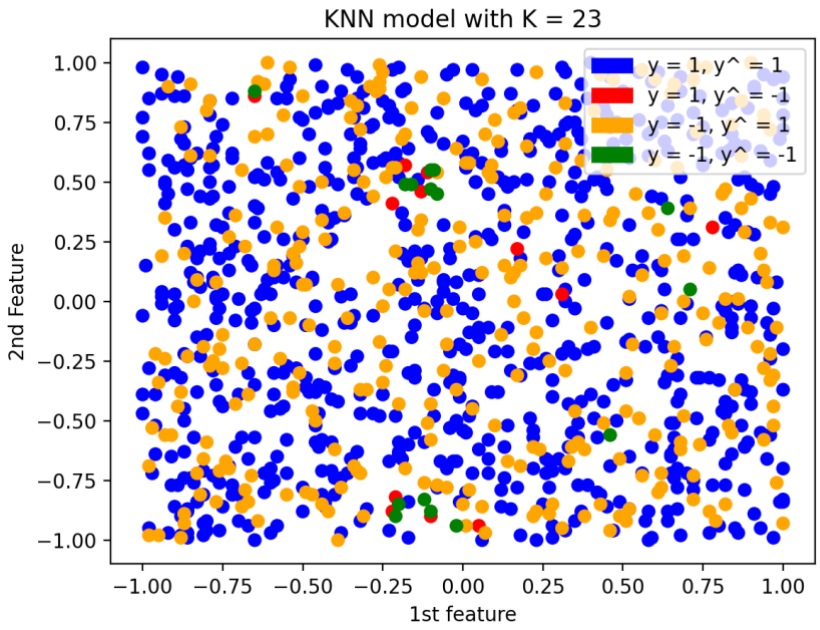

Figure 6

Figure 6 shows the KNN model with K = 23. The blue and green dots are the true positives and trues negatives respectively. The red and orange dots represent the false negatives and false positives respectively. The KNN model seems to perform slightly better than the logistic regression as there a few true negative values. The performance of the model is poor as there is a lot of false positives, which again was expected with the training data.

c.

Confusion matrix (LR):

0	329
0	726

Confusion matrix (KNN):

9	320
7	719

Confusion matrix (Baseline):

0	329
0	726

The confusion matrices for LR, KNN and the baseline model (Majority class) are shown above. It's clear that all three models perform poorly with high numbers of false positives. The LR and baseline predictor perform the same. This shows that there is no point in using the LR model as its more complex than the baseline model which just predicts the majority class which in this case is the positive class. The KNN model performs slightly better with less false positives but more false negatives. KNN has more true cases, thus it performs better.

d.

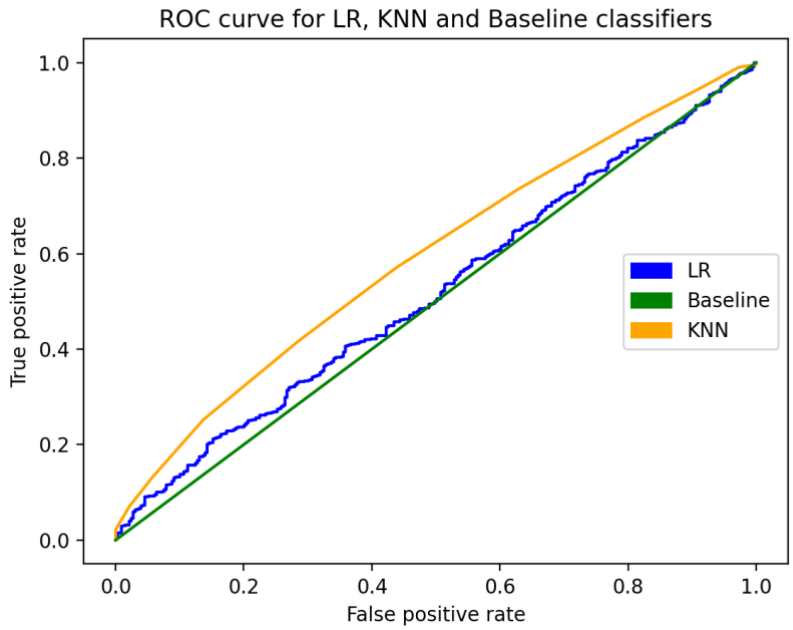

Figure 7

Figure 7 shows the ROC curves for LR, KNN and baseline model. As seen from the confusion matrices the baseline and the LR model perform equally which is supported by there ROC curves being nearly identical. As the decision threshold rises the number of true positives rises proportionally with the false positives. The KNN model perform better with its ROC curve slightly closer to ideal. It is still very far away from the ideal. The ROC curves confirm the slightly better performance of KNN versus LR and the baseline.

e.

It's clear that the training data was not suitable to create well performing models. The LR model performed the same as the baseline, thus I would recommend the baseline model over the LR model due to it being more simplistic. The KNN model performed slightly better than the LR and baseline and for this reason I would recommend this model for the optimum performance.

a.

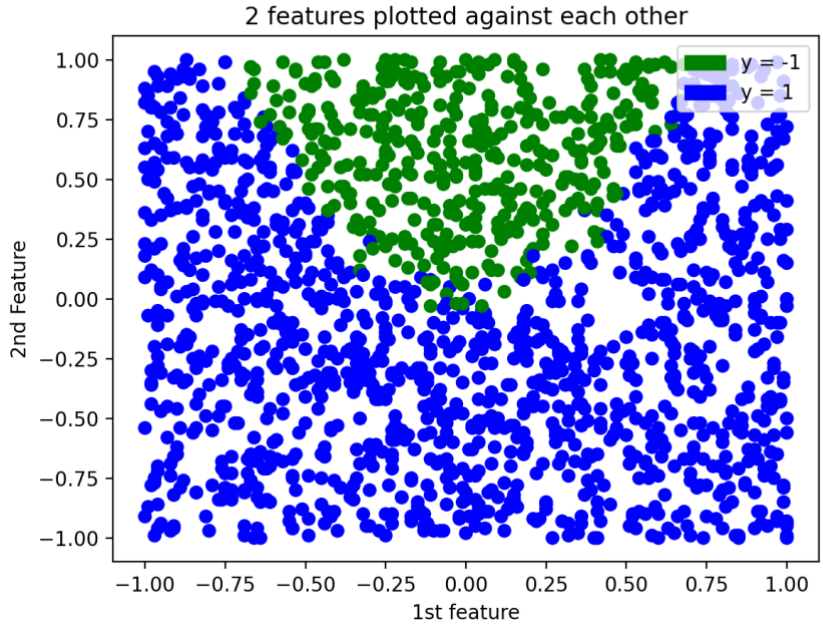

Figure 1

Figure 1 shows feature one plotted against feature 2. It's clear from the graph that the decision boundary is shaped in the form of a positive quadratic. This is in contrast to part I and it should yield well performing models. I anticipate the optimum order for the system features to be 2. The decision boundary is very distinct which would enable for a high accuracy model.

i.

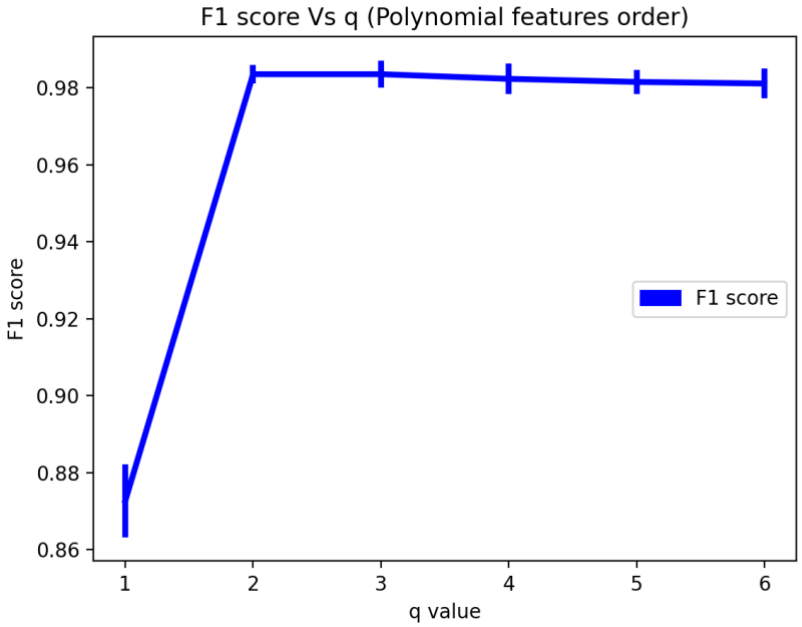

Figure 2

Figure 2 shows the cross validation performed for various q values. As in part i, F1 score is used for evaluation. As can be seen when the q value is 1, the f1 score is approximately .87. The model is suffering from underfitting as the model order is below 2 and the training data

decision boundary appears to have the shape of a quadratic. When the q value is raised to 2, the F1 score increases dramatically to approximately .98. The higher order features can model the data more accurately. Any further increases in q doesn't return a greater f1 score. For this reason, the optimum value of q for this model and data would be 2, since anything bigger than 2 will lead to overfitting and a more complex model with more features than are needed.

ii.

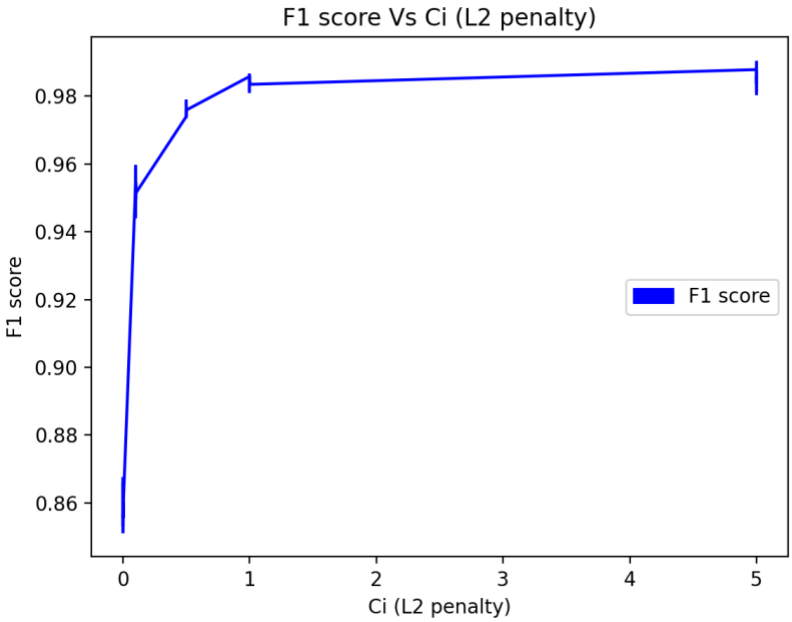

Figure 3

Figure 4 shows the graph for F1 score versus C. It can be seen from C ~ 0, the F1 score increases dramatically. The F1 score reaches its max of .98 at approximately C=1. Any further increase in C beyond 1 doesn't yield a great increase in the F1 score. For this reason, the optimum value of C = 1, to keep the weight as large as possible and thus reduce the parameter sizing to reduce the complexity of the model.

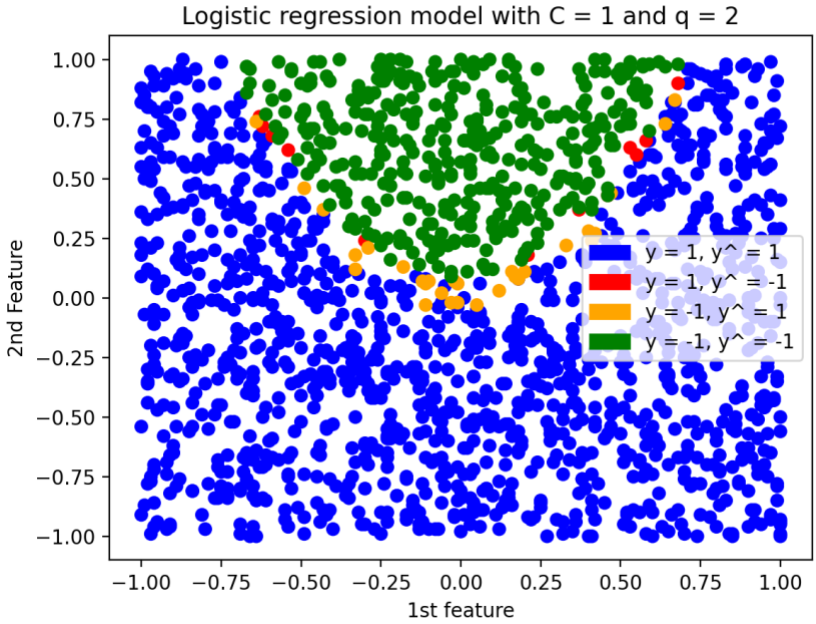

Figure 4

Figure 5 shows the logistic regression model with the tuned hyperparameters of C=1 and q = 2. The blue and green dots are the true positives and trues negatives respectively. The red and orange dots represent the false negatives and false positives respectively. It can be seen there are very few orange and red dots, showing that the logistic regression model performs well. This is backed up by an F1 score of .97.

b.

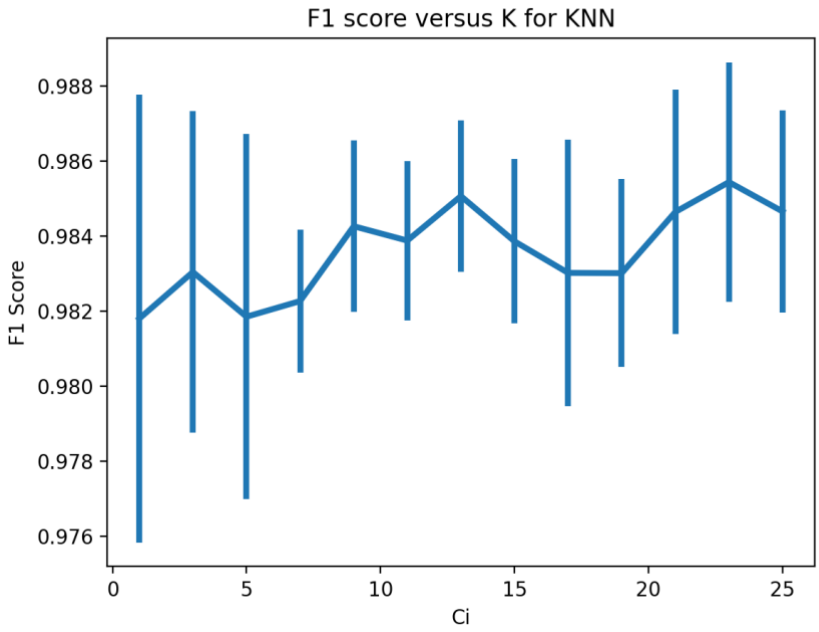

Figure 5

Figure 3 shows the F1 score of the KNN model versus K. The F1 score fluctuates with increasing K. The F1 score for K = 13 and K = 23, were very similar. A larger K increase the KNN model complexity. I decided on a K value of 23 as it provided the highest F1 score, and the model loaded quickly. This would be something g to take into consideration for larger training sets.

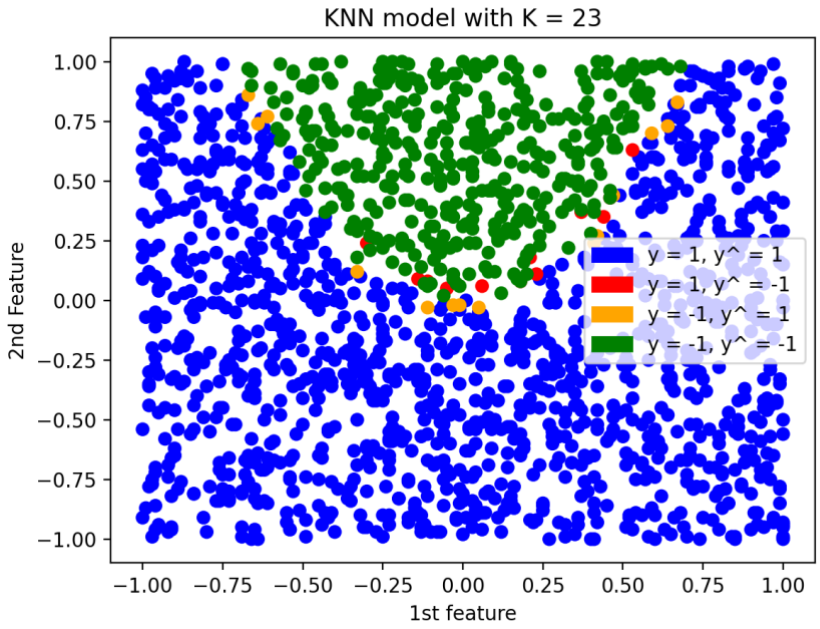

Figure 6

The KNN classifications with the tuned parameter K = 23 on the original data are shown in figure 7. The blue and green dots are the true positives and trues negatives respectively. The red and orange dots represent the false negatives and false positives respectively. It seems from visual inspection that the KNN classifier performs slightly better than the logistic regression classifier. This is confirmed by a higher F1 score of .98 versus .96 for LR. It must be noted that KNN was able to achieve this without the higher order polynomial features which the LR model used. This will be beneficial to the performance and efficiency of the model.

c.

Confusion matrix (LR):

37	29
12	1223

Confusion matrix (KNN):

393	14
12	1223

Confusion matrix (Baseline):

0	407
0	1235

The confusion matrices above confirms that both models perform well and that KNN performs slightly better than LR. The KNN and LR have the same number of false positives and true negatives, but the KNN has a superior number of true positives and lower number of false negatives. The confusion matrix for the baseline highlights the simplicity of the model as it only predicts negative, since it the more common. This results in a high number

(407) of false negatives and no true positives. It clear that both the LR and KNN model are far superior to the baseline classifier, justifying their extra complexity.

d.

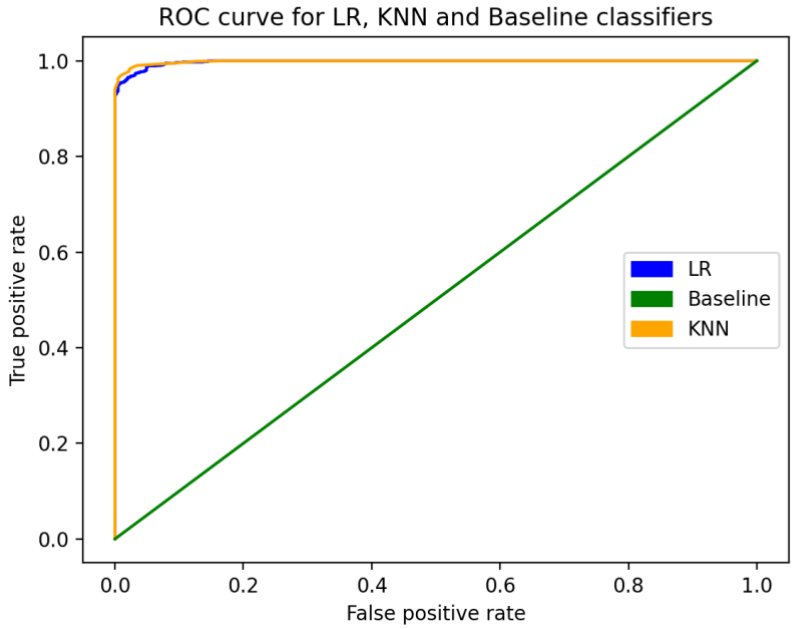

Figure 7

Figure 8 shows the ROC curves for the LR, KNN and baseline classifier. The KNN and LR are nearly identical and are near the ideal shape of a unit step function. This further backs up the good performance seen from the confusion matrices and F1 scores. The KNN model is nearer to the ideal of a unit step function than the logistic regression, further backing up the slightly superior performance of KNN. The baseline predictor ROC curve is a straight-line through the origin, showing that the true positives increase proportionally with the false positives, which is not desirable. This is what was expected for the baseline.

e.

From the confusion matrices and ROC curves, it's clear that the KNN and LR models perform very well with very high numbers of true positives and true negatives while very low numbers of false negatives and false positives. The KNN model performs slightly better with a ROC curve nearer the optimum with slightly higher true positive and thus lower false negatives. This is impressive as it used the two original features rather than the higher order polynomial features used by the LR model. The baseline predictor is far worse than the model, which was expected since it takes very little resources to run. Its main function was to evaluate the KNN and LR, to be able to justify the extra complexity for the LR and KNN models. The KNN model achieves a slightly higher score but in general KNN is slower than LR particularly the more data that is used. On this dataset I noticed no difference in computation time for the models, so I would recommend KNN to achieve maximum F1 score.

It clear to see the effect of training data has on the model when you compare the results of part I and part ii, when the same model was used with different parameters. The difference in performance was huge.

Appendix

```
# i, a, ii: C range for Logistic regression (cross val)
Xpoly = PolynomialFeatures(2).fit_transform(X)  # Chose optimum q value
mean_error = []; std_error = []; mean_error_dum = []; std_error_dum = []
Ci_range = [0.000000005, 0.000005, .001, 0.1, 0.5, 1, 5]
ci_graph = []
for Ci in Ci_range:
    LR = LogisticRegression(penalty='l2', C=Ci)
    dummy_regr = DummyRegressor(strategy="mean")
    temp=[]; temp_dum = []

    kf = KFold(n_splits=5)
    for train, test in kf.split(Xpoly):
        LR.fit(Xpoly[train], y[train])
        ypred_LR = LR.predict(Xpoly[test])

        temp.append(mean_squared_error(y[test],ypred_LR))
        mean_error.append(np.array(temp).mean())
        std_error.append(np.array(temp).std())

        ci_graph.append(Ci)


plt.figure(2)
plt.title("F1 score")
plt.errorbar(ci_graph,mean_error,yerr=std_error, c='blue')
plt.xlabel('Ci (L2 penalty)')
plt.ylabel('Mean square error')
f1_leg = mpatches.Patch(color='blue', label="F1 score")# legend for f1 score
plt.legend(handles=[f1_leg], loc="center right")  # Plot legend
plt.show()

```

Figure 8

Figure 3 shows the code used to generate the cross validation for the C value for the polynomial features. Firstly, the polynomial features are defined. Then the arrays for storing the F1 score, and standard deviation are created. Then the range of C penalty is defined. The model is then created and the number of splits of the data is defined. Cross validation is then performed and the F1 score for the various c values are stored along with the standard deviation. The F1 score are then plotted against the various C values. This was repeated for the K value for KNN and q value for the polynomial features for LR.

```
model_knn = KNeighborsClassifier(n_neighbors=23).fit(X, y)  # Optimised knn model
y_pred_knn = model_knn.predict(X)  # fitting knn model
df["E"] = y_pred_knn  # Storing knn predictions in csv file
print("F1 score: ",f1_score(y, y_pred_knn, average="macro")) #F1 score

plt.figure(5)  # Figure 1 for part A
plt.title("KNN predictions plotted against actual values")# Title
plt.xlabel("1st feature")  # X label
plt.ylabel("2nd Feature")  # Y label
y_neg_patch = mpatches.Patch(color='blue', label="y = 1, y^ = 1") #legend
y_pos_patch = mpatches.Patch(color='red', label="y = 1, y^ = -1") #legend
y_neg1_patch = mpatches.Patch(color='orange', label="y = -1, y^ = 1") #legend
y_pos1_patch = mpatches.Patch(color='green', label="y = -1, y^ = -1") #legend
plt.legend(handles=[y_neg_patch, y_pos_patch, y_neg1_patch, y_pos1_patch], loc="center right") #Plotting legend
colours = np.where((df['E'] == 1) & (df['C'] == 1), 'blue', '#00000000')  #Color for positive correct prediction
colours1 = np.where((df['E'] == -1) & (df['C'] == 1), 'red', '#00000000') #Color for false negative
colours2 = np.where((df['E'] == 1) & (df['C'] == -1), 'orange', '#00000000') #Color for false negative
colours3 = np.where((df['E'] == -1) & (df['C'] == -1), 'green', '#00000000') #Color for negative correct prediction
plt.scatter(c1, c2, c=colours) #scatter for positve correct prediction
plt.scatter(c1, c2, c=colours1) #scatter for false negative
plt.scatter(c1, c2, c=colours2) #scatter for false positve
plt.scatter(c1, c2, c=colours3) #scatter for negative correct prediction
plt.show()
```

Figure 9

Figure 9 shows how to KNN model was created and plotted. In lines 141-142, the model was created and fitted the training data. The graph was then created with a title, labels, legend and 4 scatters for the true positives, true negatives, false positives and false negatives. The graphs were then plotted.

```
165     conf_knn = confusion_matrix(y, y_pred_knn)  #Confusion matrix for knn
166     conf_log = confusion_matrix(y, y_pred_log)  # Confusion matrix for knn
167     print("Confusion matrix(KNN)", conf_knn)  #Printing confusion matrix
168     print("Confusion matrix(LR)", conf_log)  #Printing confusion matrix
169     model_bl = DummyRegressor(strategy = 'median')  # Baseline model
170     model_bl.fit(Xpoly, y) #Fitting baseline model
171     y_pred_bl = model_bl.predict(Xpoly)# Baseline model predictions
172     conf_bl = confusion_matrix(y, y_pred_bl)  # Confusion matrix for baseline model
173     print("Confusion matrix(BL)", conf_bl)  #Print baseline confusion matrix
```

Figure 10

Figure 10 shows how the confusion matrices for the LR and KNN were got and printed. The baseline classifier was created and fitted to the training data and its confusion matrix was printed out.

```
175     plt.figure(6)
176     fpr, tpr, _ = roc_curve(y,model_log.decision_function(Xpoly))  # Defining false positives and true positives
177     plt.title("ROC curve for LR, KNN and Baseline classifiers")# Title
178     plt.plot(fpr,tpr, c = 'blue')  # Plot roc curve
179     plt.xlabel('False positive rate')  # X label
180     plt.ylabel('True positive rate')  # Y label
181
182     # calculate the fpr and tpr for all thresholds of the classification
183     probs = model_knn.predict_proba(X)
184     preds = probs[:,1]
185     fpr, tpr, threshold = metrics.roc_curve(y, preds)
186     roc_auc = metrics.auc(fpr, tpr)
187     plt.plot(fpr,tpr, c = 'orange')  # Plot roc curve
188
189     plt.xlabel('False positive rate')# X label
190     plt.ylabel('True positive rate')# Y label
191     y_neg_patch = mpatches.Patch(color = 'blue', label = "LR") #legend
192     y_pos_patch = mpatches.Patch(color = 'green', label = "Baseline") #legend
193     y_neg1_patch = mpatches.Patch(color = 'orange', label = "KNN") #legend
194     plt.legend(handles=[y_neg_patch, y_pos_patch, y_neg1_patch], loc="center right") #Plotting legend
195
196
197     fpr, tpr, thresholds = roc_curve(y, y_pred_bl)
198     plt.plot(fpr,tpr, c = 'green')  # Plot roc curve
199     plt.xlabel('False positive rate')  # X label
200     plt.ylabel('True positive rate')  # Y label
201     plt.show()
```

Figure 11

Figure 11 shows how the ROC curves for the three model were created. Firstly, the true positives and false positives for the LR model were defined and plotted. This was repeated for KNN and baseline predictor.