Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS7CS4/CSU44061 - Machine Learning
Assignment 4:

Student Name:
Hugh Jordan

Student Number:
16321743

Assignment 4

(i) Noisy Dataset

(a) The downloaded data (# id:7--7--7-1) was visualised using Matplotlib's scatter plot. This is shown below in Figure 2. It is evident from Figure 2 that the measured data is very noisy, and no model may be able to capture the important relationship.

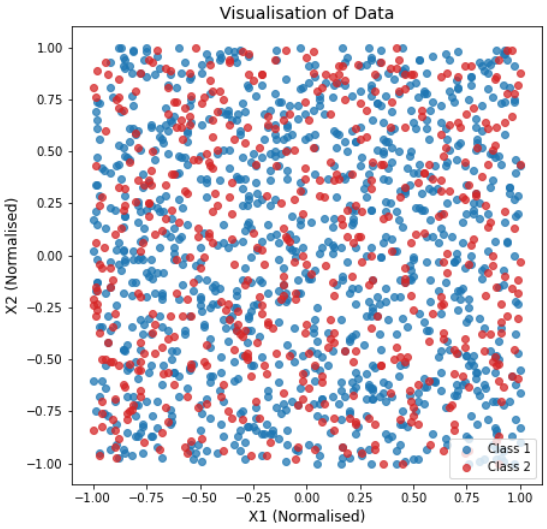

Figure 2. Visualisation of First Dataset

The two features in the dataset were augmented with polynomial features, and a Logistic Regression classifier was then trained. Furthermore, an L2 penalty was added to the cost function. 5-Fold cross-validation was then used to select the optimal degree of the polynomial features q. The segment of code used to implement this is shown in Figure 3.1. It is evident from Figure 3.1 that the program loops through a range of q values and calculates the cross-entropy loss and the accuracy of the classifier. The program calculates these metrics using K-Folds cross-validator. This class provides the train/test indices for the training and test sets. The program then calculates the two metrics this k-Fold loop.

```
kf = KFold(n_splits = 5)
loss = []; loss_std = []
acc = []; acc_std = []
q_range = [1, 2, 3, 4, 5, 6]
for q in q_range:
    Xpoly_lr = PolynomialFeatures(q).fit_transform(X)
    logreg = LogisticRegression(C=1, penalty='l2', solver='lbfgs')
    curr_loss=[]; curr_acc=[]
    for train, test in kf.split(Xpoly_lr):
        logreg.fit(Xpoly_lr[train], y[train].ravel())
        y_pred = logreg.predict(Xpoly_lr[test])
        curr_loss.append(log_loss(y[test], y_pred))
        curr_acc.append(accuracy_score(y[test], y_pred))
    loss.append(np.array(curr_loss).mean())
    loss_std.append(np.array(curr_loss).std())
    acc.append(np.array(curr_acc).mean())
    acc_std.append(np.array(curr_acc).std())
```

Figure 3.1. 5-Fold Cross-Validation Implementation

This is shown in Figure 3.2. It is evident from Figure 3.2. that q has almost no impact on the performance of the model. The performance of the model fails to improve as the value of q increases, and there is a large variance in each of the data points. This suggests that the data measured failed to capture the important relationship or that the data measured is too noisy. Therefore, 1 was chosen as the optimal value. This is because the lowest value of q that maximises the performance of the classifier should be chosen to reduce overfitting.

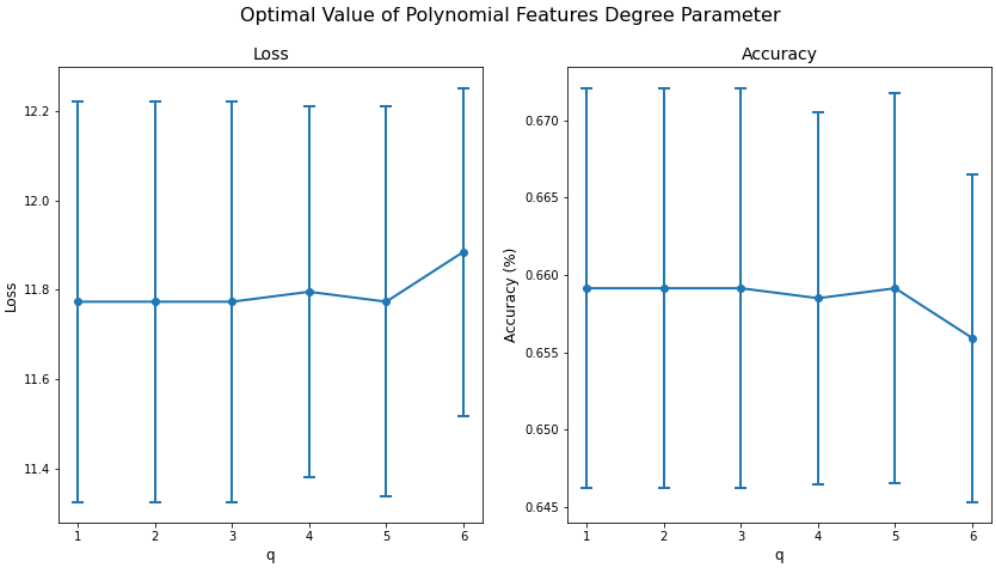

Figure 3.2 Optimal Value of Polynomial Features Degree Parameter

5-Fold cross-validation was then used to select the weight C given to the penalty in the cost function. This is shown in Figure 4. It is evident from Figure 4 that C has no impact on the performance of the model. The performance of the model fails to improve as the value of C increases, and there is a large variance in each of the data points. This suggests again that the data measured fails to capture the important relationship or that the measured data is too noisy. Therefore, 0 was chosen as the optimal value. This is because the lowest value of C that maximises the performance of the classifier should be chosen to reduce overfitting.

Finally, the trained logistic regression classifier was used to predict the target values in the validation dataset. The validation dataset comprises 20% of the measured data. These predictions are shown in Figure 5. It is evident from Figure 5 that the classifier is unable to capture the important relationship.

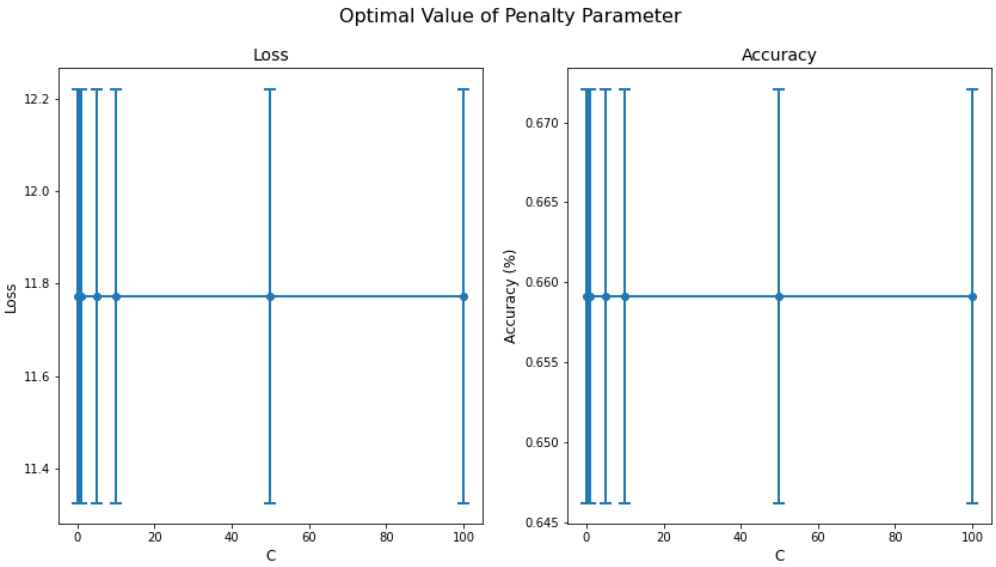

Figure 4. Optimal Value of Penalty Parameter

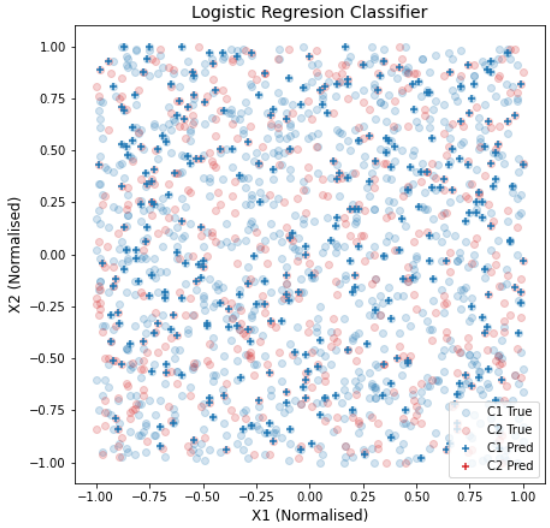

Figure 5. Trained Logistic Regression Classifier

(b) A kNN classifier was trained using the two input features. 5-Fold cross-validation was then used to select the optimal value of k, where k is the number of neighbours used to classify a new data point. This is shown in Figure 6. It is evident from Figure 6 that the performance of the model improves as the number of neighbours increases. However, the performance of the model is still very poor. This again suggests that the data measured failed to capture the important relationship or that the data measured is too noisy. Therefore, 21 was chosen as the optimal value. This is because any increase in the value of k after this point increases the performance of the model by a negligible amount. However, 21 is a very high value for k and using a value as large as 21 is computationally expensive. It is recommended that a lower value of k typically be chosen.

5-Fold cross-validation was then used to deduce whether it is worth augmenting the features with polynomial features for a kNN classifier. This is shown in Figure 3. It is evident from Figure 3 that the value of q has almost no impact on the performance of the model. Furthermore, the performance of the model fails to improve as the value of q increases, and there is a large variance for each of the data points. This suggests that the data measured failed to capture the important relationship or that the data measured is too noisy. Therefore, it is not worth augmenting the features with polynomial features for *this* kNN classifier. However, further investigation is required to deduce whether it is worth augmenting the features with polynomial features for all kNN classifiers. Otherwise, this logic could be said for Logistic Regression in (a). This will be investigated further in (ii).

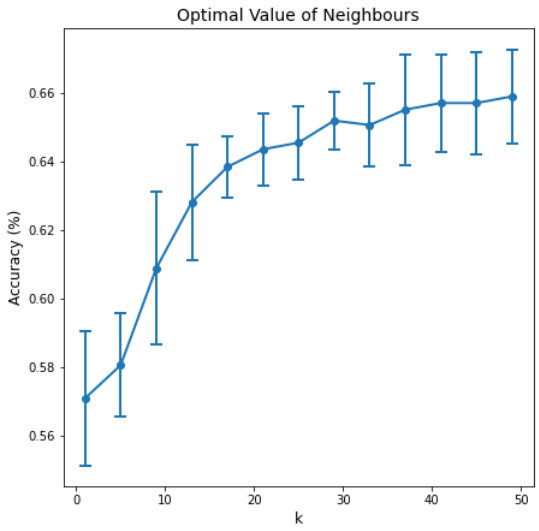

Figure 6. Optimal Value of Nearest Neighbours

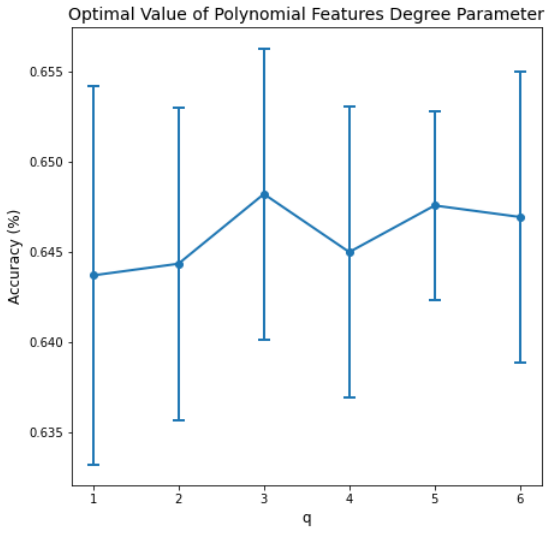

Figure 7. Optimal Value of Polynomial Features Degree Parameter

Finally, the trained kNN classifier was used to predict the target values in the validation dataset. These predictions are shown in Figure 8. It is evident from Figure 8 that the kNN classifier is unable to capture the important relationship.

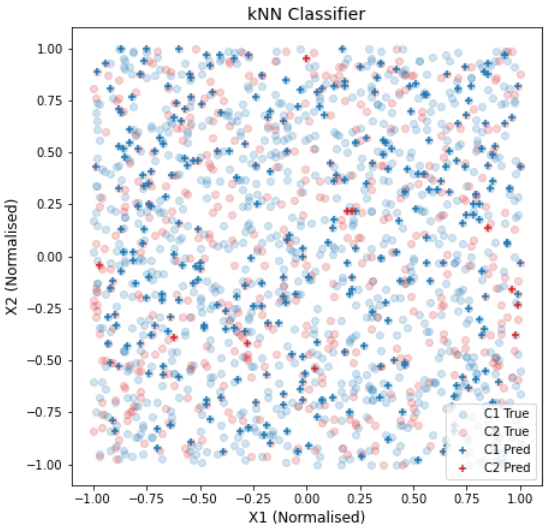

Figure 8. Trained kNN Classifier

(c) The confusion matrices for the Logistic Regression and kNN classifiers are both shown in Table 1 and 2, respectively. Additionally, the confusion matrix for a baseline classifier that always predicts the most common class is shown in Table 3. This baseline classifier was created using the Dummy Classifier. This classifier was designed to be used as a comparison and not as a real classifier. It is evident from Table 1 – 3 that all three classifiers perform similarly. This suggests again that the data measured failed to capture the important relationship or that the data measured is too noisy.

	Predicted Positive	**Predicted Negative**
True Positive	193	0
True Negative	118	0

Table 1. Logistic Regression Confusion Matrix

	Predicted Positive	**Predicted Negative**
True Positive	184	9
True Negative	113	5

Table 2. kNN Confusion Matrix

	Predicted Positive	**Predicted Negative**
True Positive	193	0
True Negative	118	0

Table 3. Baseline Confusion Matrix

(d) The ROC curves for the three classifiers were then calculated. The segment of code used to generate this is shown in Figure 9.1. In Figure 9.1, the program creates the ROC curve by calculating y_score. y_score is a probability estimate that a new data point is of the positive class. It uses Logistic Regressions' decision function and kNN's predict_proba function to calculate y_score. It then generates the ROC Curve using the function roc_curve. These curves are shown in Figure 9.2. It is evident from Figure 9.2 that all classifiers perform similarly, which again suggests that the data measured failed to capture the important relationship or that the data measured is too noisy.

```
rs = ShuffleSplit(n_splits=1, test_size=0.2)
train, test = next(rs.split(X, y))
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('ROC Curve', fontsize=14)
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
logreg.fit(Xpoly_lr[train], y[train].ravel())
y_scores = logreg.decision_function(Xpoly_lr[test])
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr)
knn.fit(Xpoly_knn[train], y[train].ravel())
y_scores = knn.predict_proba(Xpoly_knn[test])[:, 1]
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr)
Xpoly = PolynomialFeatures(1).fit_transform(X)
dummy.fit(Xpoly[train], y[train].ravel())
y_scores = dummy.predict_proba(Xpoly[test])[:, 1]
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr, '--')
ax.legend(['Logistic Regression', 'kNN', 'Baseline'], loc='lower right')
plt.savefig('roc_curve')
plt.show()
```

Figure 9.1. ROC Curve Implementation

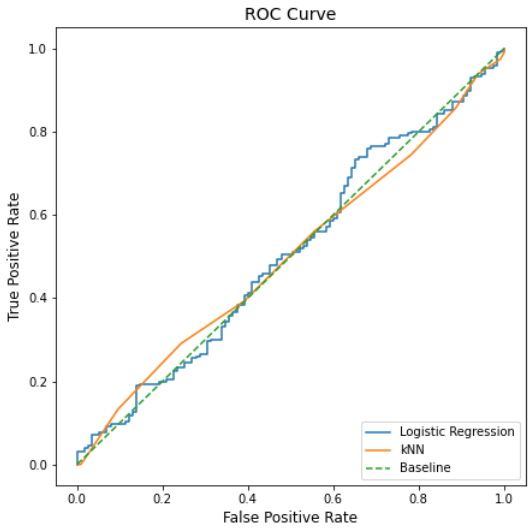

Figure 9.2. ROC Curve

(e) The performance of the three classifiers was evaluated using the data from (c) and (d). The segment of code used to implement this is shown in Figure 9.3. This program loops through each classifier and generates a confusion matrix for this classifier. It then extracts each element of the matrix and calculates the metrics shown in Table 4. It is evident from Table 4 that the Logistic Regression and kNN

classifier perform very similarly to the baseline classifier. The kNN classifiers perform slightly worse than the other two models when evaluated using the accuracy, precision, true positive rate (TPR) or F1 Score as the metric. However, the kNN classifier performs slightly better than the other two when using false positive rate (FPR) as the metric. Therefore, the logistic regression model performs slightly better than the kNN model because one metric alone is not an indication of performance. Despite this, I would not recommend either of these classifiers be used as they both were unable to capture the relationship between the training data and the target values. This becomes apparent by the fact that neither model performed better than a baseline model that simply predicted the most common class.

```
print('Metrics:')
print('%-20s %-12s %-12s %-7s %-7s %-12s' % ('Classifier', 'Accuracy(%)',
'Precision(%)', 'TPR(%)', 'FPR(%)', 'F1 Score(%)'))
for itr, model in enumerate(models):
    y_pred = model.predict(Xpolys[itr][test])
    tn, fp, fn, tp = confusion_matrix(y[test], y_pred).ravel()
    acc = accuracy_score(y[test], y_pred)
    pre = precision_score(y[test], y_pred)
    tpr = tp/(tp+fn)
    fpr = fp/(tn+fp)
    f1_sc = f1_score(y[test], y_pred)
    print('%-20s %-12.2f %-12.2f %-7.2f %-7.2f %-13.2f' % (labels[itr], acc, pre,
tpr, fpr, f1_sc))
```

Figure 9.3. Metric Table Implementation

Classifier	Accuracy	Precision	TPR	FPR	F1 Score
Logistic Regression	0.62%	0.62%	1.00%	1.00%	0.77%
kNN	0.61%	0.62%	0.95%	0.96%	0.75%
Baseline	0.62%	0.62%	1.00%	1.00%	0.77%

Table 4. Evaluation and Comparison of Classifiers

(ii) Non-Noisy Dataset

(a) The second dataset was visualised using a scatter plot as well. This is shown in Figure 10. It is evident from Figure 10 that the measured data is not very noisy and that the measured data captured the important relationship. The data is more than likely quadratically separable.

The two features in the dataset were augmented with polynomial features again, and a Logistic Regression classifier was then trained. 5-Fold cross-validation was then used to select the optimal value of q. This is shown in Figure 11. It is evident from Figure 11 that the performance of the model improves as the value of q increases until it reaches the point where q = 2. After that point, the performance of the model remains constant. Therefore, 2 was chosen as the optimal value. This is because the lowest value of q that maximises the performance of the classifier should be chosen to reduce overfitting.

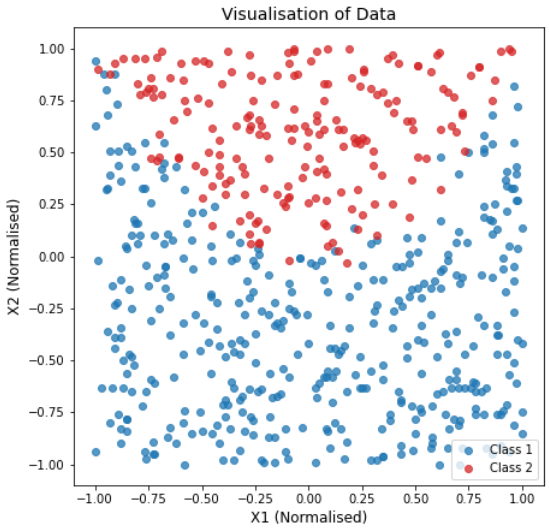

Figure 10. Visualisation of Second Dataset

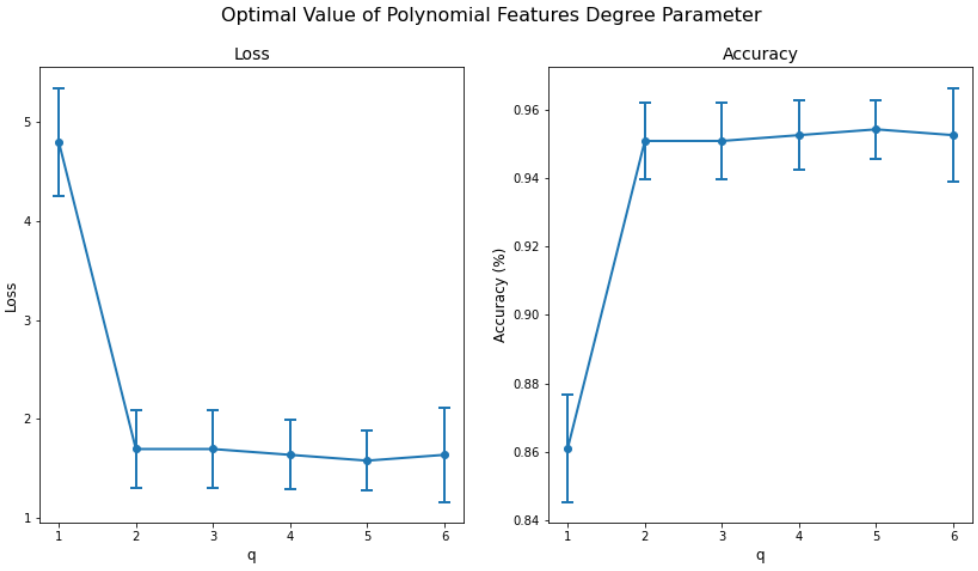

Figure 11. Optimal Value of Polynomial Features Degree Parameter

5-Fold cross-validation was then used to select C. This is shown in Figure 12. It is evident from Figure 12 that the performance of the model improves as the value of C increases until it reaches the point where C = 5. After that point, the performance of the model plateaus. Therefore, a value of 5 was chosen as the optimal value. This is because the lowest value of C that maximises the performance of the classifier should be chosen to reduce overfitting.

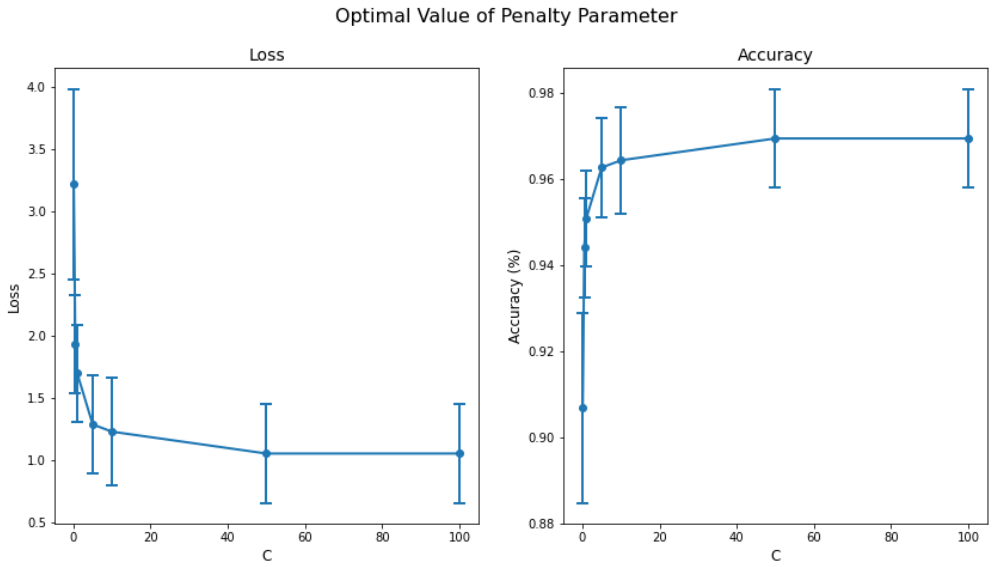

Figure 12. Optimal Value of Penalty Parameter

Finally, the trained logistic regression classifier was used to predict the target values in the validation dataset. These predictions are shown in Figure 13. It is evident from Figure 13 that the classifier was able to capture the relationship between the training data and the target values.

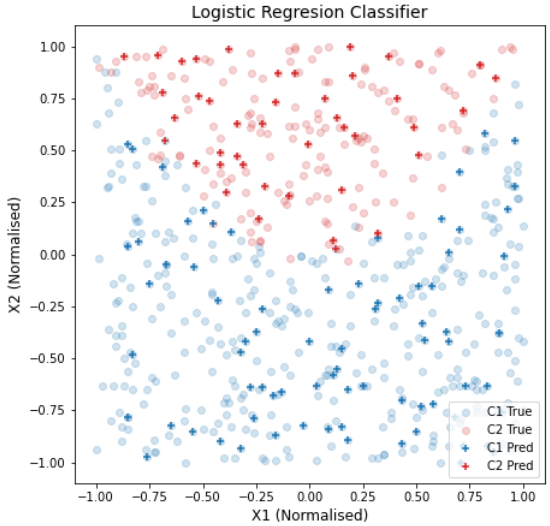

Figure 13. Trained Logistic Regression Classifier

(b) A kNN classifier was trained using the two input features. 5-Fold cross-validation was then used to select the optimal value of k. This is shown in Figure 14. It is evident from Figure 14 that the performance of the model is maximised when k = 1 or 29. Therefore, the value 1 was chosen as the optimal value. This is because 1 is smaller than 29, which means it is less computationally expensive. However, 1 is a small value for k and using a value as small as 1 means that noise will have a more significant influence on the predictions. This is not the case for this dataset;

however, because this data point has a lower variance than every other data point in Figure 14.

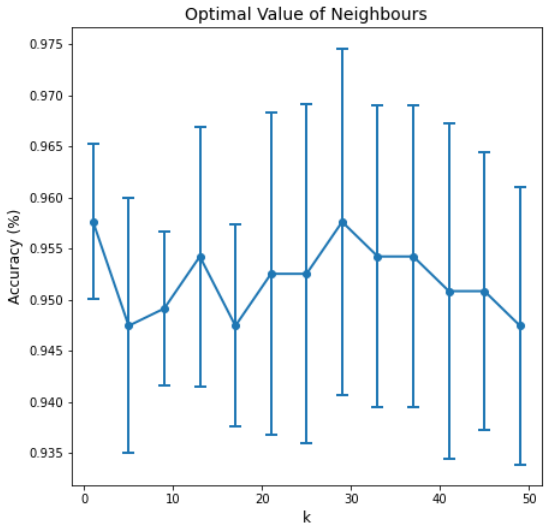

Figure 14. Optimal Value of Nearest Neighbours

5-Fold cross-validation was then used to deduce whether it is worth augmenting the features with polynomial features for a kNN classifier. This is shown in Figure 15. It is evident from Figure 15 that the performance of the model remains almost constant as the value of q increase until it reaches the point where q = 4. After this point, the performance of the model begins to decline. Therefore, it is not worth augmenting the features with polynomial features for kNN classifiers.

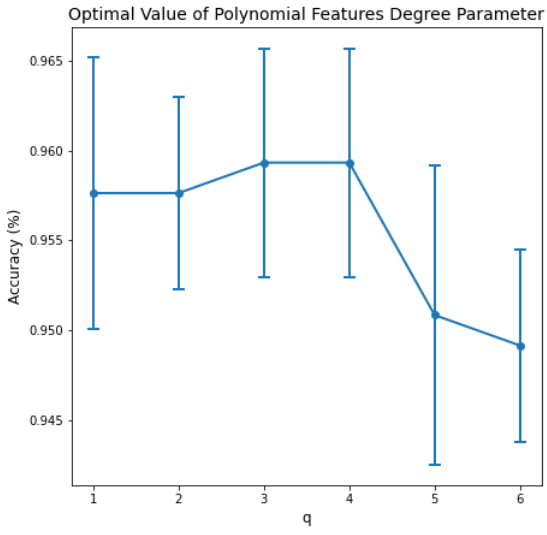

Figure 15. Optimal Value of Polynomial Features Degree Parameter

Finally, the trained kNN classifier was used to predict the target values in the validation dataset. These predictions are shown in Figure 16. It is evident from Figure 16 that the classifier was able to capture the relationship between the training data and the target values.

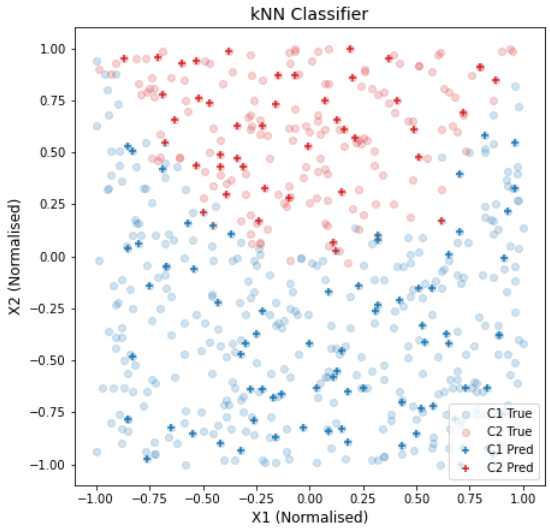

Figure 16. Trained kNN Classifier

(c) The confusion matrices for the Logistic Regression and kNN classifiers are both shown in Table 5 and 6, respectively. Additionally, the confusion matrix for a baseline classifier that always predicts the most common class is shown in Table 3. It is evident from Table 5 – 7 that both the Logistic Regression and kNN classifiers performed significantly better than the baseline model. This was because they were both able to capture the relationship between the training data and the target values.

	Predicted Positive	**Predicted Negative**
True Positive	82	2
True Negative	3	31

Table 5. Logistic Regression Confusion Matrix

	Predicted Positive	**Predicted Negative**
True Positive	79	5
True Negative	1	33

Table 6. kNN Confusion Matrix

	Predicted Positive	**Predicted Negative**
True Positive	84	0
True Negative	34	0

Table 7. Baseline Confusion Matrix

(d) The ROC curves for the three classifiers were then calculated. This is shown in Figure 17. It is evident from Figure 17 that both the Logistic Regression and kNN classifier performed significantly better than the baseline model again. However, the Logistic Regression model performed a little better than the kNN model.

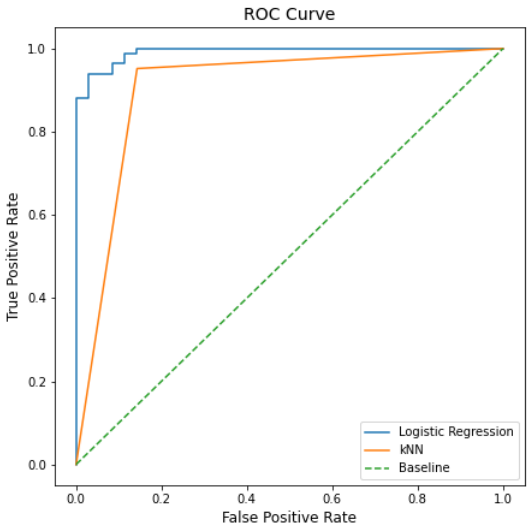

Figure 17. ROC Curve

(e) The performance of the three classifiers was evaluated using the data from (c) and (d). This is shown in Table 8. It is evident from Table 8 that the Logistic Regression and kNN classifier performed significantly better than the baseline classifier. Additionally, the Logistic Regression classifiers perform slightly better than the kNN classifier when evaluated using the accuracy, precision, true positive rate (TPR) or F1 Score as the metric. However, the kNN performs slightly better than the logistic regression classifier when evaluated using the false positive rate as the metric. In contrast to this, the baseline classifier performed better than the other two classifiers when evaluated using the false positive rate as the metric. However, it is recommended that at least two metrics are used when evaluating a classifier because only one metric alone can be misleading, as demonstrated in Table 8. I would recommend using the Logistic Regression classifier because this classifier was able to capture the relationship between the training data and the target values slightly better than the kNN classifier.

Classifier	Accuracy	Precision	TPR	FPR	F1 Score
Logistic Regression	0.96%	0.96%	0.98%	0.09%	0.97%
kNN	0.95%	0.99%	0.94%	0.03%	0.96%
Baseline	0.71%	0.71%	1.00%	1.00%	0.83%

Table 4. Evaluation and Comparison of Classifiers

Appendix:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, accuracy_score, precision_score, f1_score,
confusion_matrix, roc_curve
from sklearn.model_selection import ShuffleSplit
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

def split(X, y, cond):
   indices_true = [i for i in range(len(y)) if y[i] > cond]
   indices_false = [i for i in range(len(y)) if y[i] < cond]
   return [X[indices_true, :], X[indices_false, :]]

df = pd.read_csv('week4-2.csv',comment='#')
X1=df.iloc[:, 0]
X2=df.iloc[:, 1]
X=np.column_stack((X1, X2))
y=df.iloc[:, 2]
y = np.array(df.iloc[:, 2])
y = np.reshape(y, (-1, 1))

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Visualisation of Data', fontsize=14)
ax.set_xlabel('X1 (Normalised)', fontsize=12)
ax.set_ylabel('X2 (Normalised)', fontsize=12)
class1, class2 = split(X, y, 0)
ax.scatter(class1[:, 0], class1[:, 1], marker='o', c='C0', alpha=0.75)
ax.scatter(class2[:, 0], class2[:, 1], marker='o', c='C3', alpha=0.75)
ax.legend(['Class 1', 'Class 2'], loc='lower right')
plt.savefig('data_visualisation')
plt.show()

kf = KFold(n_splits = 5)
loss = []; loss_std = []
acc = []; acc_std = []
q_range = [1, 2, 3, 4, 5, 6]
for q in q_range:
   Xpoly_lr = PolynomialFeatures(q).fit_transform(X)
   logreg = LogisticRegression(C=1, penalty='l2', solver='lbfgs')
```

```
    curr_loss=[]; curr_acc=[]
    for train, test in kf.split(Xpoly_lr):
        logreg.fit(Xpoly_lr[train], y[train].ravel())
        y_pred = logreg.predict(Xpoly_lr[test])
        curr_loss.append(log_loss(y[test], y_pred))
        curr_acc.append(accuracy_score(y[test], y_pred))
    loss.append(np.array(curr_loss).mean())
    loss_std.append(np.array(curr_loss).std())
    acc.append(np.array(curr_acc).mean())
    acc_std.append(np.array(curr_acc).std())

fig = plt.figure(figsize=(14, 7))
fig.suptitle('Optimal Value of Polynomial Features Degree Parameter',
fontsize=16)
ax = fig.add_subplot(1, 2, 1)
ax.set_title('Loss', fontsize=14)
ax.set_xlabel('q', fontsize=12)
ax.set_ylabel('Loss', fontsize=12)
ax.errorbar(q_range, loss, yerr=loss_std, linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(q_range, loss, marker='o')
ax = fig.add_subplot(1, 2, 2)
ax.set_title('Accuracy', fontsize=14)
ax.set_xlabel('q', fontsize=12)
ax.set_ylabel('Accuracy (%)', fontsize=12)
ax.errorbar(q_range, acc, yerr=acc_std, linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(q_range, acc, marker='o')
plt.savefig('lr_q_cross_validation')
plt.show()

q_lr = q_range[np.argmax(np.diff(acc) < 0.0025)]
print('Optimal Value: %d\n' % (q_lr))
Xpoly_lr = PolynomialFeatures(q_lr).fit_transform(X)

kf = KFold(n_splits=5)
loss = []; loss_std = []
acc = []; acc_std = []
penalties = [0.1, 0.5, 1, 5, 10, 50, 100]
for penalty in penalties:
    logreg = LogisticRegression(C=penalty, penalty='l2', solver='lbfgs')
    curr_loss = []; curr_acc = []
    for train, test in kf.split(X):
        logreg.fit(Xpoly_lr[train], y[train].ravel())
        y_pred = logreg.predict(Xpoly_lr[test])
        curr_loss.append(log_loss(y[test], y_pred))
        curr_acc.append(accuracy_score(y[test], y_pred))
```

```
        loss.append(np.array(curr_loss).mean())
        loss_std.append(np.array(curr_loss).std())
        acc.append(np.array(curr_acc).mean())
        acc_std.append(np.array(curr_acc).std())

    fig = plt.figure(figsize=(14, 7))
    fig.suptitle('Optimal Value of Penalty Parameter', fontsize=16)
    ax = fig.add_subplot(1, 2, 1)
    ax.set_title('Loss', fontsize=14)
    ax.set_xlabel('C', fontsize=12)
    ax.set_ylabel('Loss', fontsize=12)
    ax.errorbar(penalties, loss, yerr=loss_std, linewidth=2.0, capsize=5.0,
    elinewidth=2.0, markeredgewidth=2.0)
    ax.scatter(penalties, loss, marker='o')
    ax = fig.add_subplot(1, 2, 2)
    ax.set_title('Accuracy', fontsize=14)
    ax.set_xlabel('C', fontsize=12)
    ax.set_ylabel('Accuracy (%)', fontsize=12)
    ax.errorbar(penalties, acc, yerr=acc_std, linewidth=2.0, capsize=5.0,
    elinewidth=2.0, markeredgewidth=2.0)
    ax.scatter(penalties, acc, marker='o')
    plt.savefig('lr_C_cross_validation')
    plt.show()

    penalty = penalties[np.argmax(np.diff(acc) < 0.0025)]
    print('Optimal Value: %d\n' % (penalty))
    logreg = LogisticRegression(C=penalty, penalty='l2', solver='lbfgs')

    fig = plt.figure(figsize=(7,7))
    ax = fig.add_subplot(1, 1, 1)
    ax.set_title('Logistic Regresion Classifier', fontsize=14)
    ax.set_xlabel('X1 (Normalised)', fontsize=12)
    ax.set_ylabel('X2 (Normalised)', fontsize=12)
    class1, class2 = split(X, y, 0)
    ax.scatter(class1[:, 0], class1[:, 1], marker='o', c='C0', alpha=0.2)
    ax.scatter(class2[:, 0], class2[:, 1], marker='o', c='C3', alpha=0.2)
    class1, class2 = split(X[test], y_pred, 0)
    ax.scatter(class1[:, 0], class1[:, 1], marker='+', c='C0', alpha=1.0)
    ax.scatter(class2[:, 0], class2[:, 1], marker='+', c='C3', alpha=1.0)
    ax.legend(['C1 True', 'C2 True', 'C1 Pred', 'C2 Pred'], loc='lower right')
    plt.savefig('lr_classifier')
    plt.show()



    q_knn = 1
    Xpoly_knn = PolynomialFeatures(q_knn).fit_transform(X)
```

```
kf = KFold(n_splits=5)
acc = []; acc_std = []
k_range = [1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49]
for k in k_range:
   knn = KNeighborsClassifier(n_neighbors=k, weights='uniform')
   curr_acc = []
   for train, test in kf.split(X):
      knn.fit(Xpoly_knn[train], y[train].ravel())
      y_pred = knn.predict(Xpoly_knn[test])
      curr_acc.append(accuracy_score(y[test], y_pred))
   acc.append(np.array(curr_acc).mean())
   acc_std.append(np.array(curr_acc).std())

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Optimal Value of Neighbours', fontsize=14)
ax.set_xlabel('k', fontsize=12)
ax.set_ylabel('Accuracy (%)', fontsize=12)
ax.errorbar(k_range, acc, yerr=acc_std, linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(k_range, acc, marker='o')
plt.savefig('kNN_k_cross_validation')
plt.show()

k = k_range[np.argmax(np.diff(acc) < 0.0025)]
print('Optimal Value: %d\n' % (k))
knn = KNeighborsClassifier(n_neighbors=k, weights='uniform')

kf = KFold(n_splits = 5)
acc = []; acc_std = []
q_range = [1, 2, 3, 4, 5, 6]
for q in q_range:
   Xpoly_knn = PolynomialFeatures(q).fit_transform(X)
   curr_acc=[]
   for train, test in kf.split(Xpoly_knn):
      knn.fit(Xpoly_knn[train], y[train].ravel())
      y_pred = knn.predict(Xpoly_knn[test])
      curr_acc.append(accuracy_score(y[test], y_pred))
   loss.append(np.array(curr_loss).mean())
   loss_std.append(np.array(curr_loss).std())
   acc.append(np.array(curr_acc).mean())
   acc_std.append(np.array(curr_acc).std())

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
```

```
ax.set_title('Optimal Value of Polynomial Features Degree Parameter',
fontsize=14)
ax.set_xlabel('q', fontsize=12)
ax.set_ylabel('Accuracy (%)', fontsize=12)
ax.errorbar(q_range, acc, yerr=acc_std, linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(q_range, acc, marker='o')
plt.savefig('kNN_q_cross_validation')
plt.show()

q_knn = q_range[np.argmax(np.diff(acc) < 0.0025)]
print('Optimal Value: %d\n' % (q_knn))
Xpoly_knn = PolynomialFeatures(q_knn).fit_transform(X)

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('kNN Classifier', fontsize=14)
ax.set_xlabel('X1 (Normalised)', fontsize=12)
ax.set_ylabel('X2 (Normalised)', fontsize=12)
class1, class2 = split(X, y, 0)
ax.scatter(class1[:, 0], class1[:, 1], marker='o', c='C0', alpha=0.2)
ax.scatter(class2[:, 0], class2[:, 1], marker='o', c='C3', alpha=0.2)
class1, class2 = split(X[test], y_pred, 0)
ax.scatter(class1[:, 0], class1[:, 1], marker='+', c='C0', alpha=1.0)
ax.scatter(class2[:, 0], class2[:, 1], marker='+', c='C3', alpha=1.0)
ax.legend(['C1 True', 'C2 True', 'C1 Pred', 'C2 Pred'], loc='lower right')
plt.savefig('kNN_classifier')
plt.show()



dummy = DummyClassifier(strategy='most_frequent')

rs = ShuffleSplit(n_splits=1, test_size=0.2)
train, test = next(rs.split(X, y))
print('Confusion Matrixes:')
print('%-20s %-10s %-10s %-10s %-10s' % ('Classifier', 'TP', 'FN', 'FP', 'TN'))
labels = ['Logistic Regression', 'kNN', 'Dummy']
models = [logreg, knn, dummy]
Xpolys = [Xpoly_lr, Xpoly_knn, PolynomialFeatures(1).fit_transform(X)]
for itr, model in enumerate(models):
   model.fit(Xpolys[itr][train], y[train].ravel())
   y_pred = model.predict(Xpolys[itr][test])
   tn, fp, fn, tp = confusion_matrix(y[test], y_pred).ravel()
   print('%-20s %-10d %-10d %-10d %-10d' % (labels[itr], tp, fn, fp, tn))
print()
```

```
print('Metrics:')
print('%-20s %-12s %-12s %-7s %-7s %-12s' % ('Classifier', 'Accuracy(%)',
'Precision(%)', 'TPR(%)', 'FPR(%)', 'F1 Score(%)'))
for itr, model in enumerate(models):
   y_pred = model.predict(Xpolys[itr][test])
   tn, fp, fn, tp = confusion_matrix(y[test], y_pred).ravel()
   acc = accuracy_score(y[test], y_pred)
   pre = precision_score(y[test], y_pred)
   tpr = tp/(tp+fn)
   fpr = fp/(tn+fp)
   f1_sc = f1_score(y[test], y_pred)
   print('%-20s %-12.2f %-12.2f %-7.2f %-7.2f %-13.2f' % (labels[itr], acc, pre, tpr,
fpr, f1_sc))



rs = ShuffleSplit(n_splits=1, test_size=0.2)
train, test = next(rs.split(X, y))
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('ROC Curve', fontsize=14)
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
logreg.fit(Xpoly_lr[train], y[train].ravel())
y_scores = logreg.decision_function(Xpoly_lr[test])
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr)
knn.fit(Xpoly_knn[train], y[train].ravel())
y_scores = knn.predict_proba(Xpoly_knn[test])[:, 1]
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr)
Xpoly = PolynomialFeatures(1).fit_transform(X)
dummy.fit(Xpoly[train], y[train].ravel())
y_scores = dummy.predict_proba(Xpoly[test])[:, 1]
fpr, tpr, _ = roc_curve(y[test], y_scores)
ax.plot(fpr, tpr, '--')
ax.legend(['Logistic Regression', 'kNN', 'Baseline'], loc='lower right')
plt.savefig('roc_curve')
plt.show()
```