Machine Learning Week 4 Assignment - id:8-8-8-1

1. **First Dataset**
   I started by inspecting the datasets to see if there are any relevant trends or patterns I should be aware of going into analysisng it. I discovered that the data points of the first dataset do not visibly follow any pattern, with the scatter of $y = -1$ and $y = 1$ being seemingly random. I also discovered that for both datasets, the mean y value is 0.35 - there are twice as many $y = 1$ points as there are $y = -1$. Because of this imbalance, accuracy might not be the best method of measuring performance. I then chose the F1 score as a performance measure due to its ability to capture both precision and recall. I then set on to perform 5-fold cross-validation to determine the maximum degree for the polynomial features and the C value for the logistic regression with L2 penalty.
   (a) Figure 1 shows the results of this process, with the different lines representing the maximum degree of the polynomial features considered, and the resulting F1 score plotted on the y-axis for different values of C on the x-axis. Keeping in mind that this is a plot of the F1 score, higher values are preferred. However, not only does the F1 score never improve for higher degrees or C, but it instead seems to have the opposite effect. The error bars also do not change. I will thus pick the simplest model, with a maximum degree of 1, and I will keep C = 1 as the default.

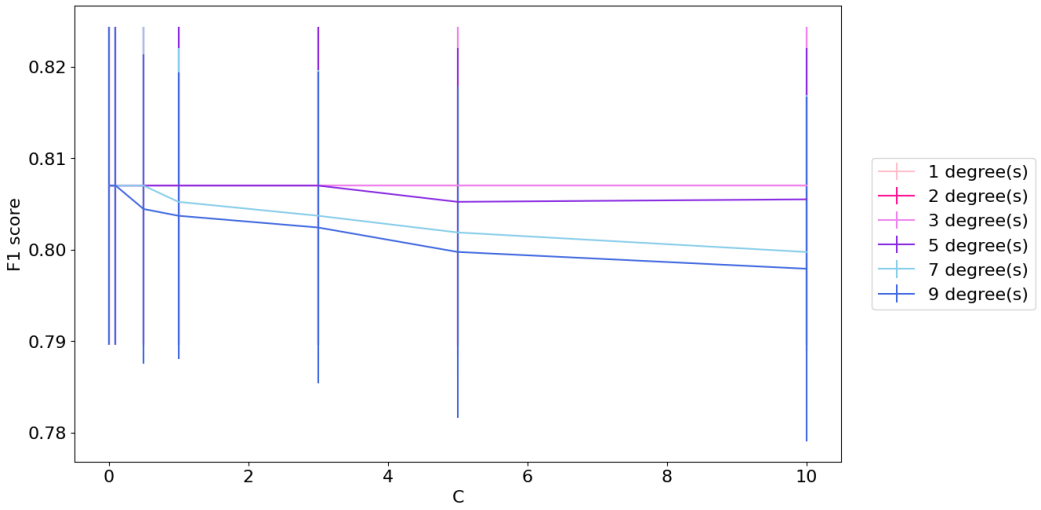

Figure 1: 5-Fold cross-validation measuring average F1 score for different combinations of C and maximum polynomial degree

Figure 2 shows the LR predicted values plotted against the true values. It can be seen the the LR classifier predicts all values $\hat{y} = 1$. The random-looking data point distribution is also visible.

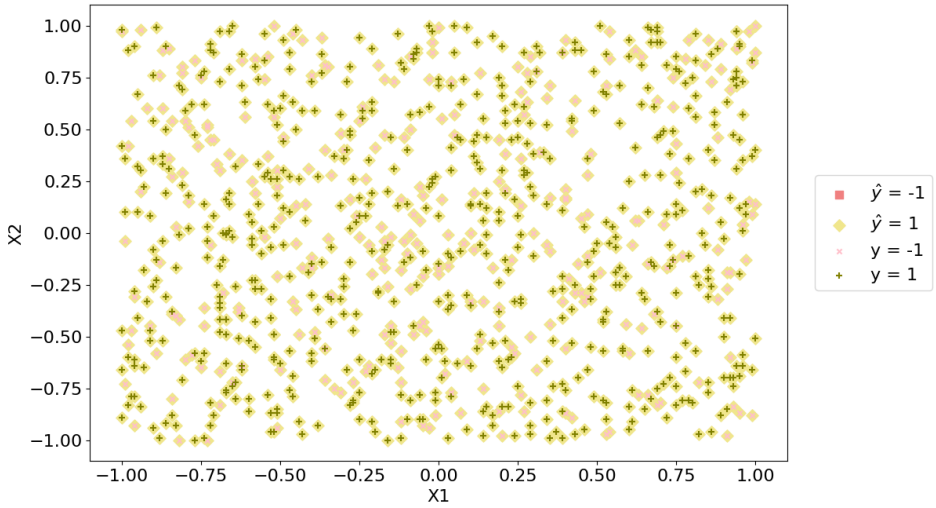

Figure 2: Scatter plot of true values y vs predictions $\hat{y}$ for the LR classifier

(b) Figure 3 shows the results of the cross-validation to find the best number K of clusters for the KNN classifier. This time, a lower K will make the model more sensitive to variation in the training data, and K has to be sufficiently big to avoid overfitting but sufficiently small to avoid underfitting. The best value should be in the elbow of the curve as that is the value for which a sufficiently specific model is achieved, without underfitting the data. In this case, a value between 5 and 11 seems appropriate. I will pick $K = 11$ because it manages to achieve good results without overfitting the training data to the point of poor test performance and it has a very small error bar.

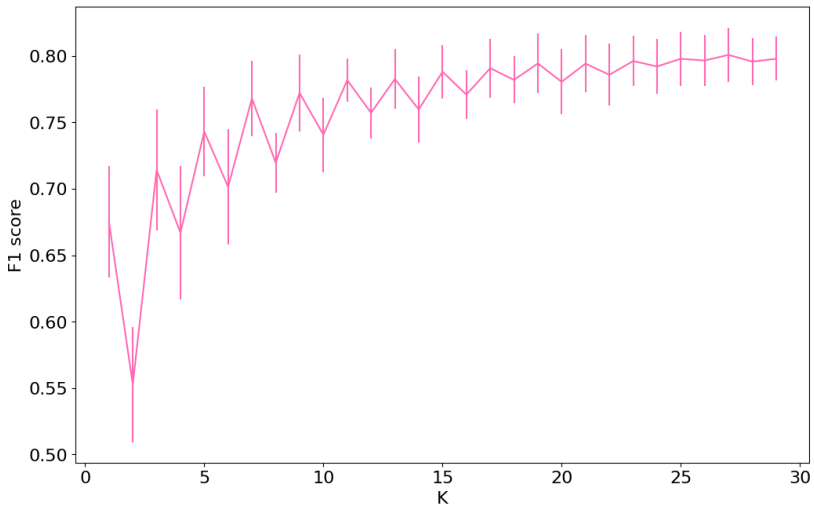

Figure 3: 5-Fold cross-validation measuring average F1 score for different K

Figure 4 shows the KNN predicted values plotted against the true values. It can be seen the the KNN classifier performs better than the LR, making some $\hat{y} = -1$ predictions, many of which accrate.

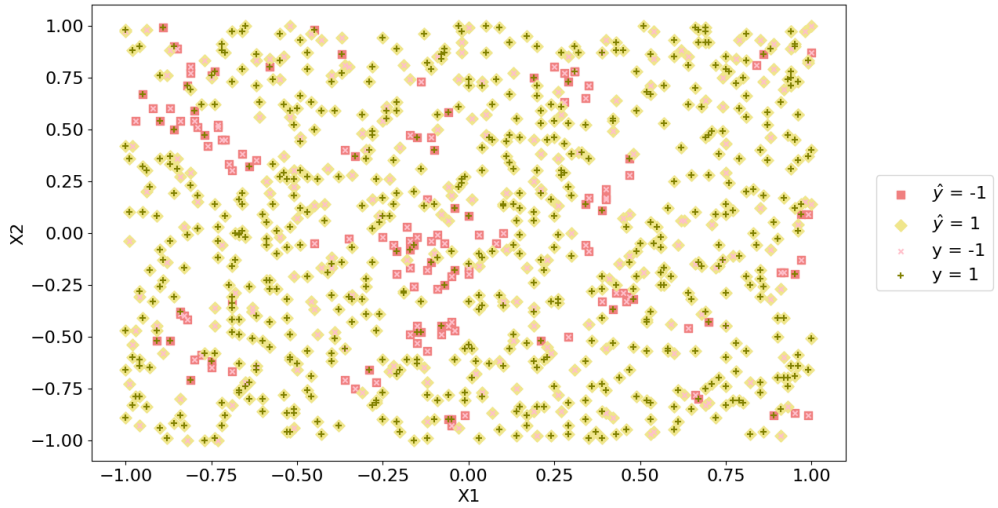

Figure 4: Scatter plot of true values y vs predictions $\hat{y}$ for the KNN classifier

(c) Figure 5 shows the confusion matrices of the three models, Logistic Regression with L2 penalty and C=1, K Nearest Neighbours with K = 11, and the baseline of always predicting the most common class. These figures are more closely examined at (e).

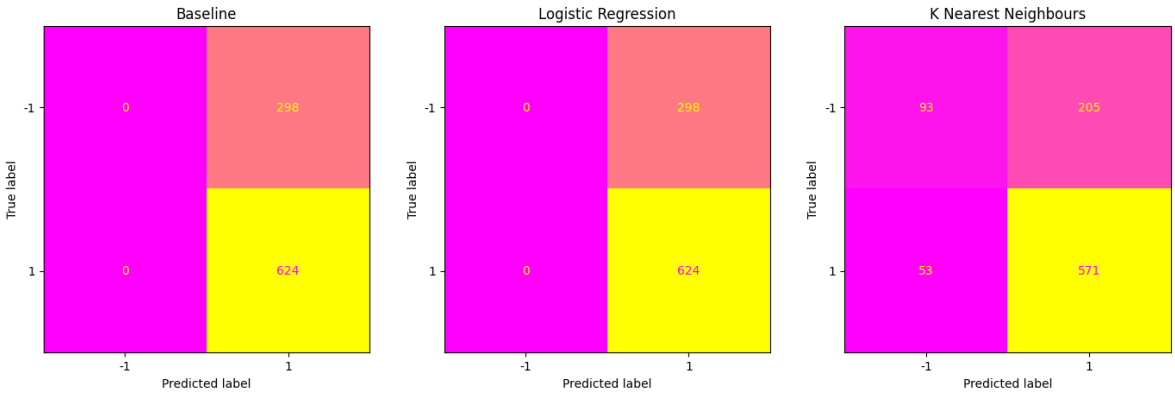

Figure 5: Confusion matrices for the LR and KNN classifiers vs a baseline

(d) Figure 6 shows the ROC curve for the LR and KNN classifiers as well as the baseline classifier shown as the $x = y$ line.

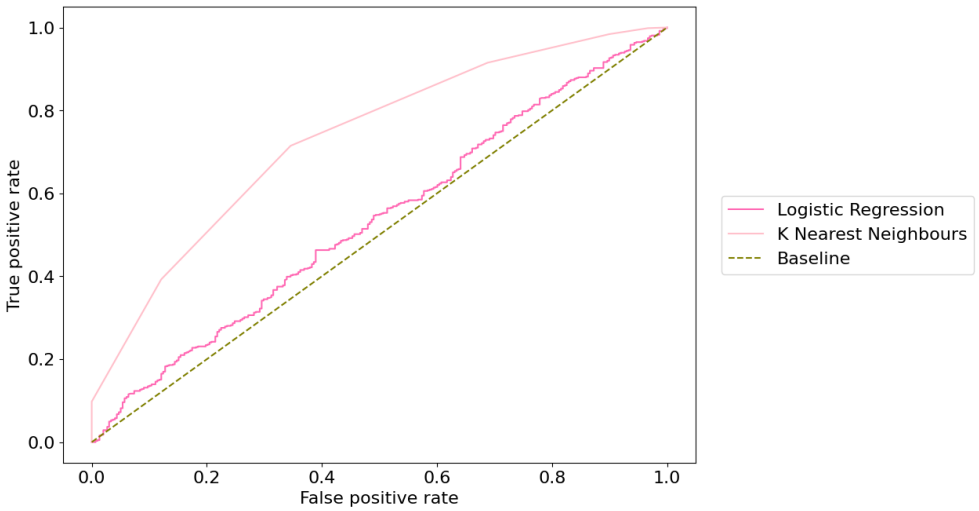

Figure 6: ROC curves of the LR and KNN classifiers vs a baseline

(e) It appears that the Logistic Regression always predicts $y = 1$, just like the baseline, while KNN makes almost 150 predictions of $\hat{y} = -1$ which are accurate in 60% of the cases. A completely random prediction for $\hat{y} = -1$ would be accurate in an expected 30% of cases, which shows that the KNN does indeed identify some meaningful patterns in the data. Thus, KNN achieves much better precision on the $y = -1$ points. KNN does have lower precision when predicting $y = 1$ points, but it does have slightly better recall, with a higher percentage of its $\hat{y} = 1$ predictions being correct. It appears as if KNN takes more 'risks' by making some $\hat{y} = -1$ predictions, but it ends up paying off - achieving better accuracy and overall matching the data more closely than the LR (and the baseline). This can also be seen by inspecting the ROC curves, with LR being basically as flat as the baseline, and KNN being much more arched (although still far from the upper left corner). Taking all this into consideration, I believe the KNN classifier is performs better than LR on this dataset, and I would recommend it over the LR.

2. **Second Dataset**

(a) Figue 7 shows the average F1 score for the 5-fold cross-validation with varying C and makimum degree of the polynomial features. It appears that that any degree ≤ 2 manages to capture the data quite well. As such, I will choose $d = 2$ as the maximum polynomial degree to capture the shape of the data and avoid overfitting. FUrthermore, there is an increase in the F1 score as C increases, with $C = 0.5$ being right in the elbow of this curve.

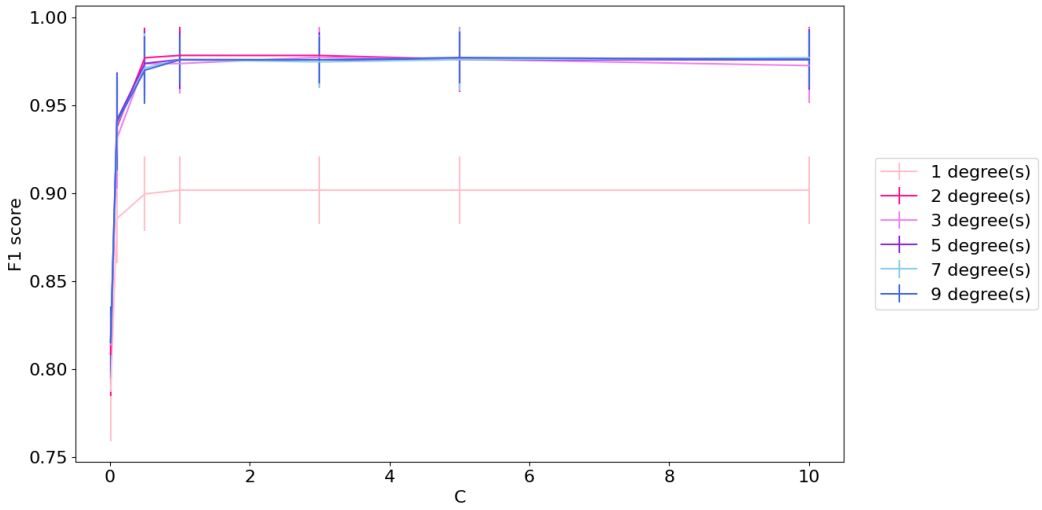

Figure 7: 5-Fold cross-validation measuring average F1 score for different combinations of C and maximum polynomial degree

Figure 8 shows the LR predicted values plotted against the true values for the second dataset. It can be seen the the LR achieves a very good accuracy, managing to capture the quadratic shape of the data points' distribution very well.

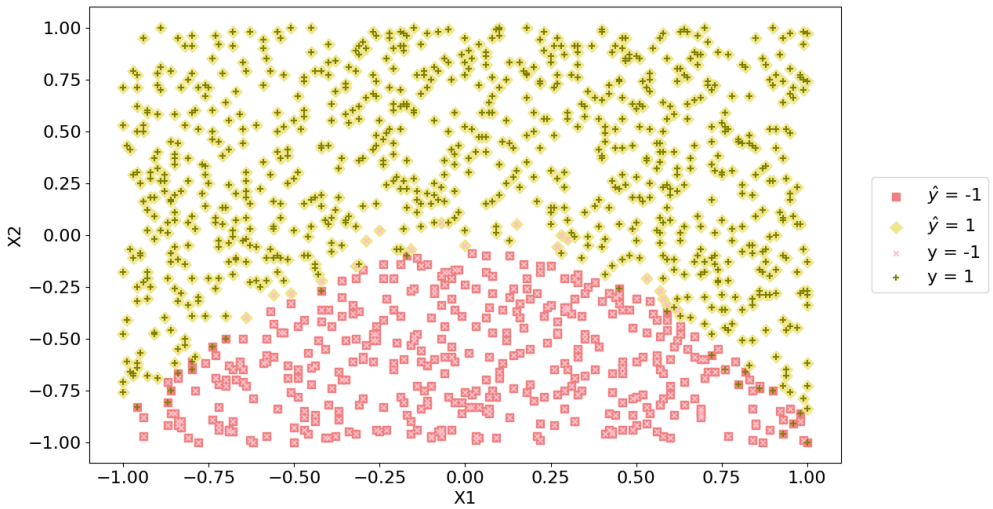

Figure 8: Scatter plot of true values y vs predictions $\hat{y}$ for the LR classifier

(b) Figure 9 shows that the F1 score does not significantly increase for any choice of K. I will choose $K = 15$ for the KNN classifier as it achieves a slightly higher mean F1 score. K has to be sufficiently big to avoid overfitting. A very small K would likely fit this dataset perfectly, but it might

not perform as well on a new test set. This is why the F1 score for the 5-fold cross validation is not at 100%. 5-Fold cross validation splits the set repeatedly into training and validation sets, and the performance is measured based on the performance achieved on the validation set. Thus, when the very small K ends up overfitting the training data, this translates to poorer performance when testing.

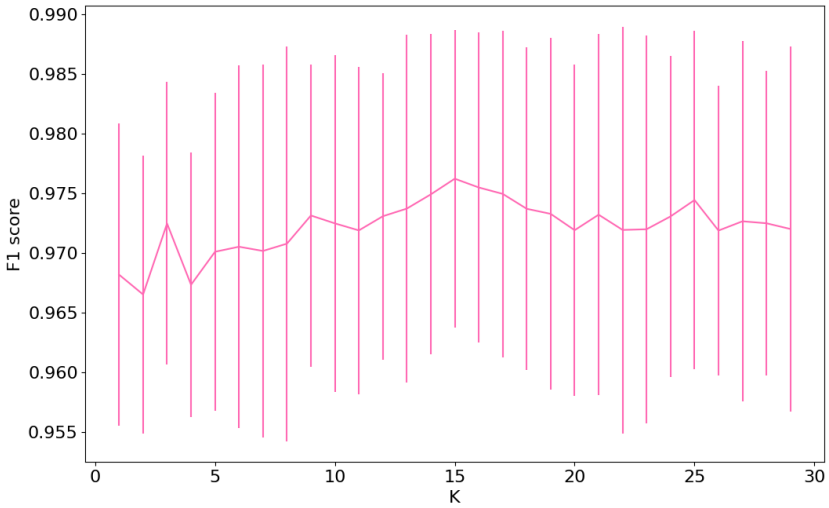

Figure 9: 5-Fold cross-validation measuring average F1 score for different K

Figure 8 shows the KNN predicted values plotted against the true values for the second dataset. The KNN also manages to capture the shape of the data very well, similar to the LR model.

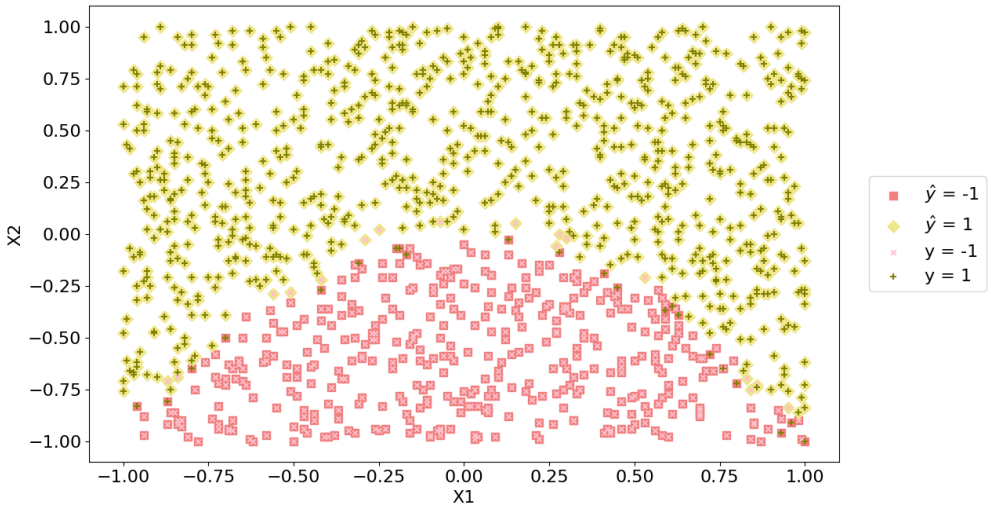

Figure 10: Scatter plot of true values y vs predictions $\hat{y}$ for the KNN classifier

(c) Figure 11 shows the confusion matrices for the Logistic Regression classifier with $C = 0.5$ and added polynomial features up to degree $d = 2$, the KNN classifier with $K = 15$, and the baseline model which always predicts $\hat{y} = 1$

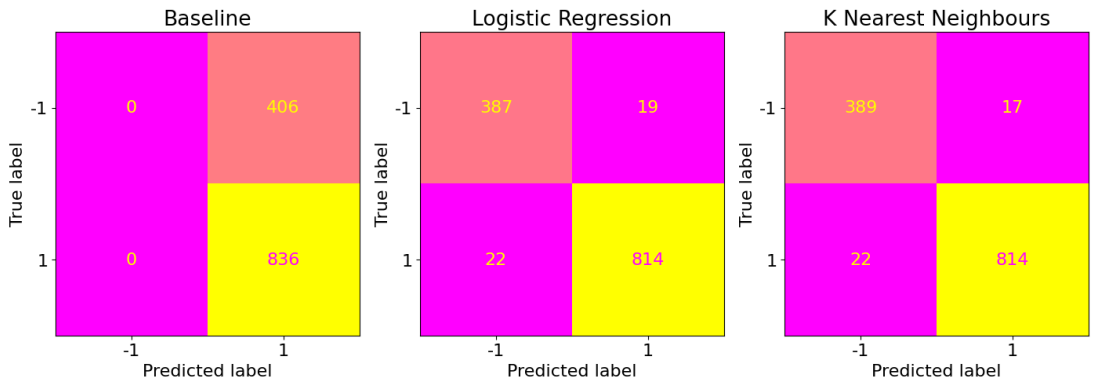

Figure 11: Confusion matrices for the LR and KNN classifiers vs a baseline

(d) Figure 12 shows the ROC curves for the LR and KNN classiers.

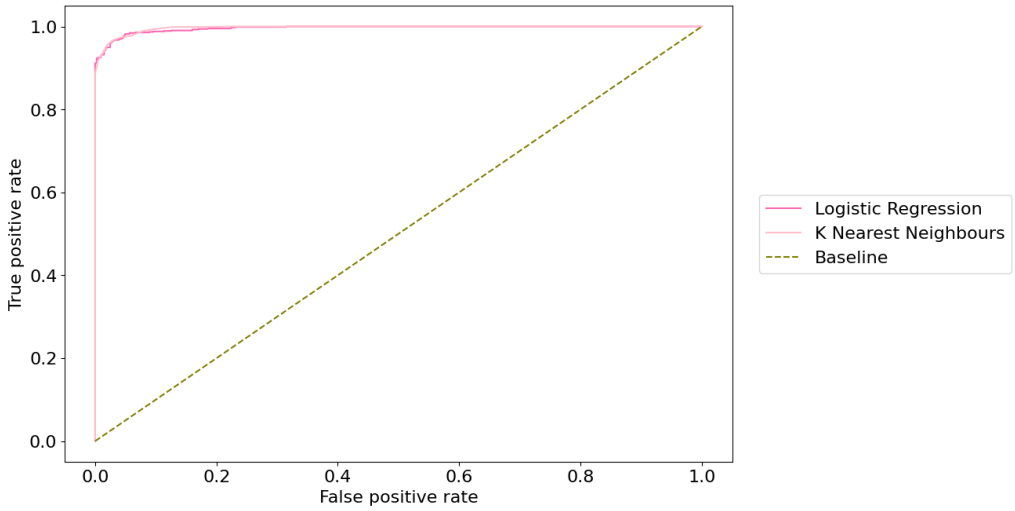

Figure 12: ROC curves of the LR and KNN classifiers vs a baseline

(e) Both the LR and KNN models achieve very good results this time. They both achieve 97% accuracy, which translates to their ROC curve being near perfect and getting very close to the top left corner. The LR classifier outperforms KNN by only two predictions. Ultimately, both of these classifiers performed very well on this dataset. That being said, this dataset was quite small, with just over 1200 observations. On a very large dataset with the same general features and showing the same patter, KNN can be slow and costly. LR with quadratic features, however, will have better scalability as its decision boundary technique will perform just as well on a larger dataset. Thus, I would reccommend LR.

Appendix - Code

```
#id:8-8-8-1
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from sklearn import metrics
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.linear_model import LogisticRegression, Lasso, Ridge
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error,\
    accuracy_score, f1_score, recall_score, precision_score, plot_confusion_matrix
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import roc_curve


def lrCrossValidation(X, y, cRange, dRange, colours):
    fig = plt.figure()
    plt.rcParams['font.size'] = '16'
    plt.xlabel('C')
    plt.ylabel('F1 score')
    for i, d in enumerate(list(dRange)):
        mean_accuracy = []
        std_accuracy = []
        for c in cRange:
            poly = PolynomialFeatures(degree=d)
            XPoly = poly.fit_transform(X)
            clf = LogisticRegression(penalty='l2', C=c, max_iter=1000)
            temp = []
            kf = KFold(n_splits=10)
            for train, test in kf.split(XPoly):
                clf.fit(XPoly[train], y[train])
                yPred = clf.predict(XPoly[test])
                temp.append(precision_score(y[test], yPred))
            mean_accuracy.append(np.array(temp).mean())
            std_accuracy.append(np.array(temp).std())
        plt.errorbar(cRange, mean_accuracy, yerr=std_accuracy, color=colours[i],
                     label=(str(d) + ' degree(s)'))
        plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()


def logisticRegression(X, y, C, d):
    poly = PolynomialFeatures(degree=d)
    XPoly = poly.fit_transform(X)
    clf = LogisticRegression(penalty='l2', C=C, max_iter=1000)
    clf.fit(XPoly, y)

    plt.rcParams['font.size'] = '16'

    yPred = clf.predict(XPoly)
    print('F-measure', f1_score(y, yPred, average='weighted'))
    print('Accuracy', accuracy_score(y, yPred))
    plt.scatter(X1[yPred == -1], X2[yPred == -1], marker='s', color='lightcoral', s=60)
    plt.scatter(X1[yPred == 1], X2[yPred == 1], marker='D', color='khaki', s=60)
    plt.scatter(X1[y == -1], X2[y == -1], marker='x', color='pink', s=20)
    plt.scatter(X1[y == 1], X2[y == 1], marker='+', color='olive', s=30)
```

```
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.legend(["$\^y$ = -1", "$\^y$ = 1", "y = -1", "y = 1"],
               bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()
    return [yPred, clf.decision_function(XPoly)]


def knnCrossValidation(X, y, kRange):
    fig = plt.figure()
    plt.rcParams['font.size'] = '16'
    plt.xlabel('K')
    plt.ylabel('F1 score')
    mean_accuracy = []
    std_accuracy = []
    for k in kRange:
        clf = KNeighborsClassifier(n_neighbors=k)
        temp = []
        kf = KFold(n_splits=10)
        for train, test in kf.split(X):
            clf.fit(X[train], y[train])
            yPred = clf.predict(X[test])
            temp.append(f1_score(y[test], yPred))
        mean_accuracy.append(np.array(temp).mean())
        std_accuracy.append(np.array(temp).std())
    plt.errorbar(kRange, mean_accuracy, yerr=std_accuracy, color='hotpink')
    plt.show()


def kNearestNeighbours(X, y, k):
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X, y)
    yPred = clf.predict(X)
    plt.rcParams['font.size'] = '16'
    print('F-measure', f1_score(y, yPred, average='weighted'))
    print('Accuracy', accuracy_score(y, yPred))
    plt.scatter(X1[yPred == -1], X2[yPred == -1], marker='s', color='lightcoral', s=60)
    plt.scatter(X1[yPred == 1], X2[yPred == 1], marker='D', color='khaki', s=60)
    plt.scatter(X1[y == -1], X2[y == -1], marker='x', color='pink', s=20)
    plt.scatter(X1[y == 1], X2[y == 1], marker='+', color='olive', s=30)
    plt.xlabel("X1")
    plt.ylabel("X2")
    plt.legend(["$\^y$ = -1", "$\^y$ = 1", "y = -1", "y = 1"],
               bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()
    return [yPred, clf.predict_proba(X)[:, 1]]


def confusionMatrix(y, yB, yLR, yKNN):
    print(accuracy_score(y, yB))
    print(accuracy_score(y, yLR))
    print(accuracy_score(y, yKNN))
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
    baseline = metrics.confusion_matrix(y, yB)
    lr = metrics.confusion_matrix(y, yLR)
    knn = metrics.confusion_matrix(y, yKNN)
    metrics.ConfusionMatrixDisplay(confusion_matrix=baseline, display_labels=[-1, 1])\
        .plot(cmap='spring',ax=ax1,colorbar=False)
```

```
    metrics.ConfusionMatrixDisplay(confusion_matrix=lr, display_labels=[-1, 1])\
        .plot(cmap='spring', ax=ax2,colorbar=False)
    metrics.ConfusionMatrixDisplay(confusion_matrix=knn, display_labels=[-1, 1])\
        .plot(cmap='spring', ax=ax3,colorbar=False)
    ax1.set_title('Baseline')
    ax2.set_title('Logistic Regression')
    ax3.set_title('K Nearest Neighbours')
    plt.show()

def roc(y, LR, KNN):
    plt.rcParams['font.size'] = '16'
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    LRfpr, LRtpr, _ = roc_curve(y, LR)
    KNNfpr, KNNtpr, _ = roc_curve(y, KNN)
    plt.plot(LRfpr, LRtpr, color='hotpink', label='Logistic Regression')
    plt.plot(KNNfpr, KNNtpr, color='pink', label='K Nearest Neighbours')

    plt.plot([0, 1], [0, 1], color='olive', linestyle='--', label='Baseline')
    plt.legend(bbox_to_anchor=(1.04, 0.5), loc="center left", borderaxespad=0)
    plt.show()


df = pd.read_csv("data2.csv", header=None)
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = df.iloc[:, 2]

colours = ['pink', 'deeppink', 'violet', 'blueviolet', 'skyblue', 'royalblue']
cRange = [0.01, 0.1, 0.5, 1, 3, 5, 10]
dRange = [1, 2, 3, 5, 7, 9]
kRange = range(1, 30)
C, d, k = 1, 1, 7 #I chose 1, 1, 7 for dataset 1 and 0.5, 2, 15 for dataset 2

''' Uncomment the desired method. logisticRegression and kNearestNeighbours
 will display the prediction scatter plots and are used by other methods.
 confusionMatrix and roc will display both scatter plots as well'''

#lrCrossValidation(X,y,cRange,dRange,colours)
#knnCrossValidation(X,y,kRange)
#logisticRegression(X,y,C,d)
#kNearestNeighbours(X, y, k)
#confusionMatrix(y, pd.DataFrame([1] * len(y)).iloc[:, 0],
                logisticRegression(X, y, C, d)[0],kNearestNeighbours(X, y, k)[0])
#roc(y,logisticRegression(X,y,C,d)[1],kNearestNeighbours(X, y, k)[1])
```