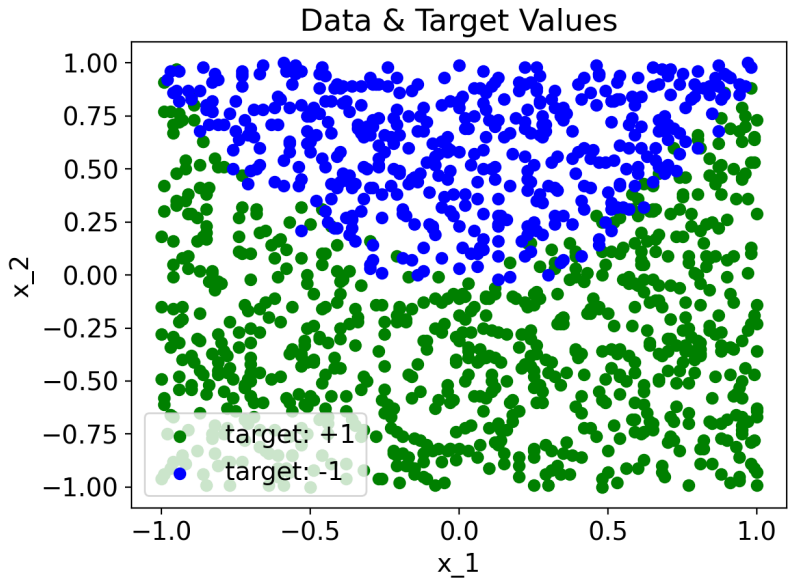
Cian Higgins 05 November 2020

Dataset 1: `# id:8-8--8-0`
Dataset 2: `# id:8-16-8-0`

(i) Dataset 1

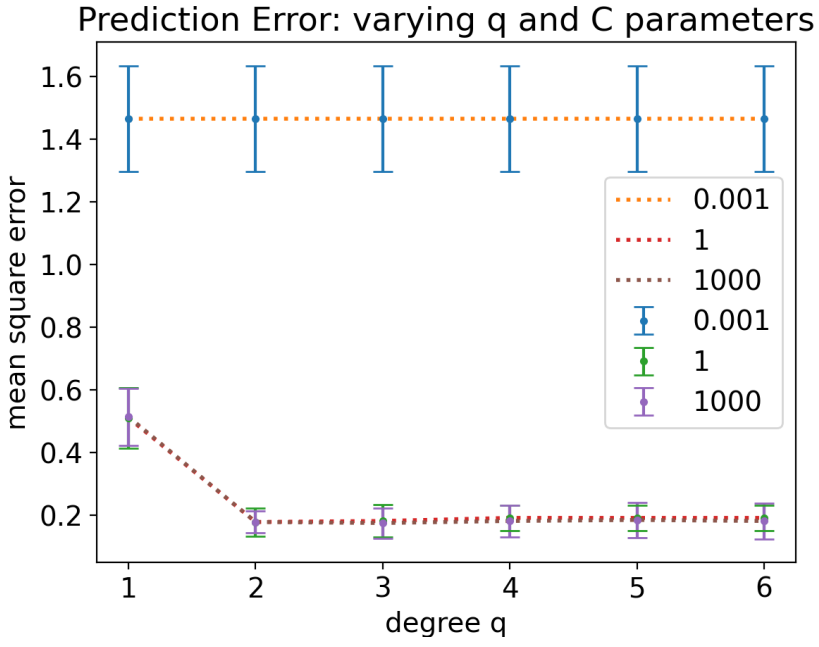
(a) Logistic Regression

The training data was plotted and coloured based on target value. The data shows clearly a curved decision boundary:

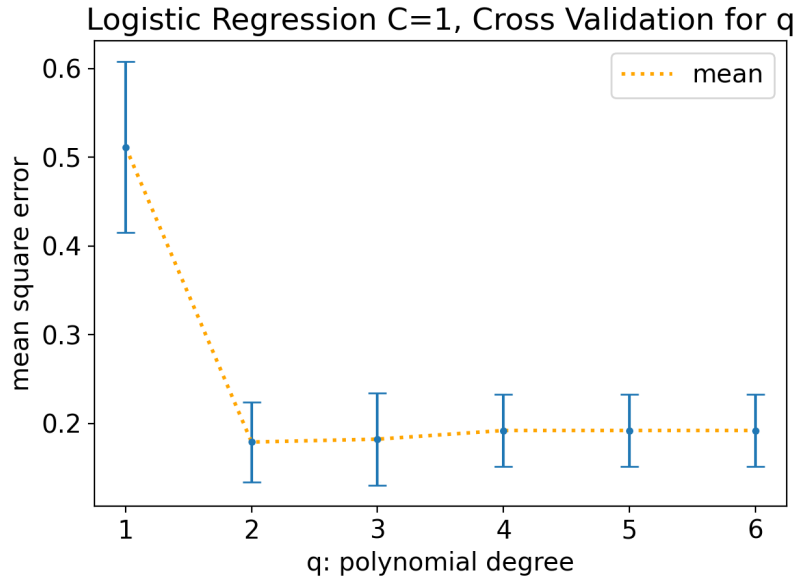


A wide range of C values was selected to get a rough overview of what C to use in the selection of the polynomial order q. The range used: [0.001, 1, 1000]. 5-fold cross validation was used to evaluate the performance of each of the values.

This error bar plot shows that a very small C value does not perform very well. Its prediction error is consistently high. The prediction error of C=1 is significantly lower, with a smaller variance. The mean error for C=1000 is very similar. However, to avoid possible overfitting, one should try to use the 'simplest' model, for this reason the value **C=1** was chosen.



Using the value C=1, 5-fold cross validation was carried out to determine the best order of polynomial to use.

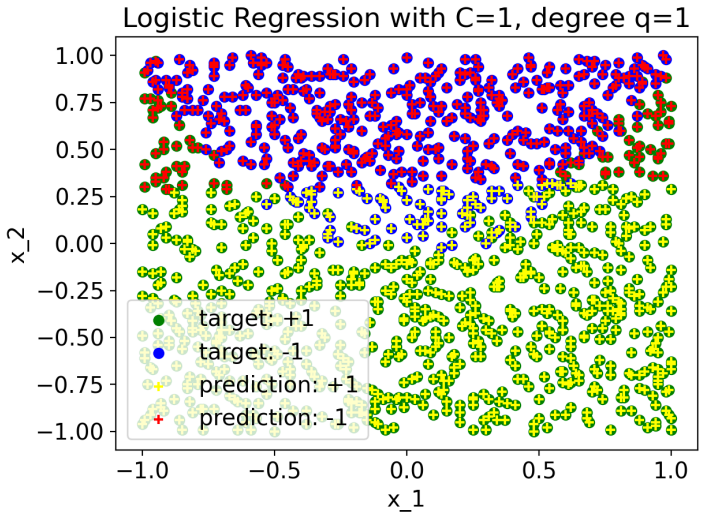


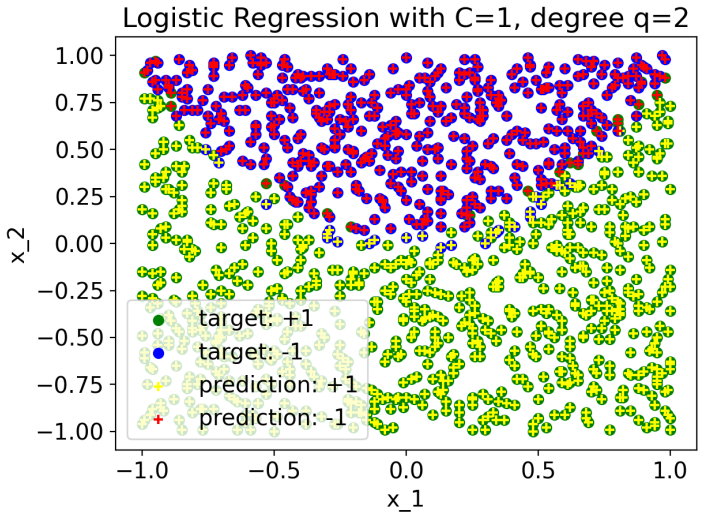
When q=1 the prediction error is high and variable.

For **q=2**, the error is relatively low, and the variance is small.

As the degree increases, the means start to rise slowly, but variance stays fairly constant.

Although q=4, q=5, etc. also seem good, once again the simpler model is being chosen to steer clear of overfitting.





The two plots above show exactly why prediction error improves so much when polynomial features are used. When q=2 the curved shape of the data can be modelled.

Function used for cross validation:

```
def k_fold_cross_val(k, model, input):
  k_fold = KFold(n_splits=k)
  sq_errs = []
  for train, test in k_fold.split(input):
      curr_model = model.fit(input[train], y[train])
      pred = curr_model.predict(input[test])
      sq_err = mean_squared_error(y[test], pred)
      sq_errs.append(sq_err)
  mean = np.mean(sq_errs)
  std = np.std(sq_errs)
  return mean, std
```

Code for determining best q value:

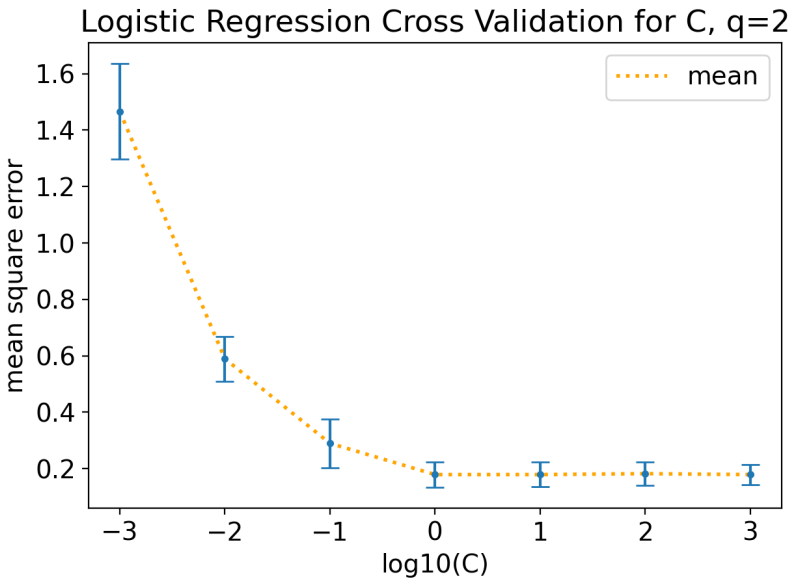
```
q_means = []
q_std_devs = []
degrees = [1, 2, 3, 4, 5, 6]
for deg in degrees:
  polyX = make_poly_features(X, deg)
  l2_log_reg = LogisticRegression(penalty='l2', C=1)
  l2_log_reg.fit(polyX, y)
  results = k_fold_cross_val(5, l2_log_reg, polyX)
  q_means.append(results[0])
  q_std_devs.append(results[1])
```

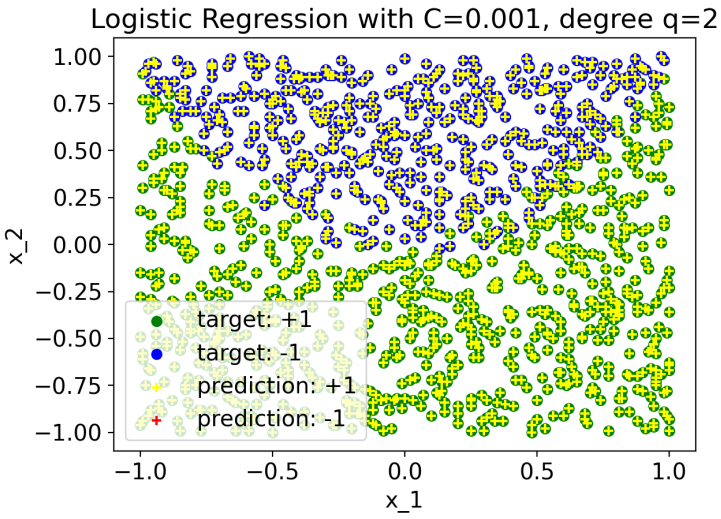
```
def make_poly_features(x, q):
  poly = PolynomialFeatures(degree=q)
  new_x = poly.fit_transform(x)
  return new_x
```

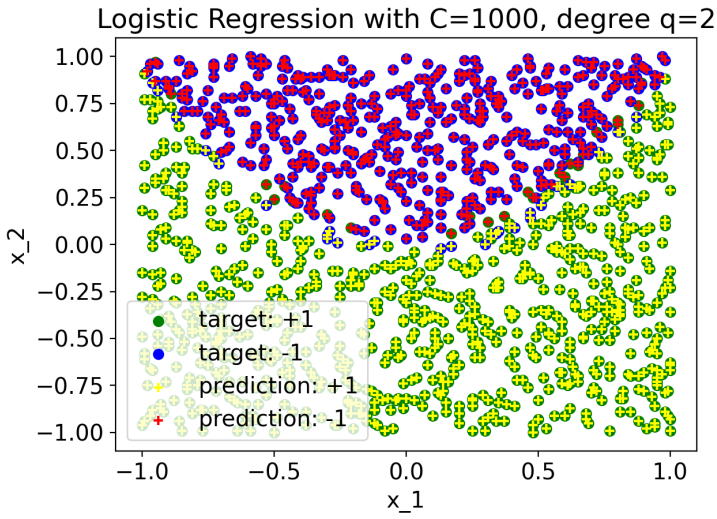
Taking the order of polynomial as q=2, cross validation was once again used to determine the best C value, this time around taking a more detailed range of values. Range used: [0.001, 0.01, 0.1, 1, 10, 100, 1000].

The values 0.001, 0.01 and 0.1 are ruled out. They each have high prediction errors and large variances.

All C values ≥ 1 yielded good (very similar) results. To avoid any potential overfitting to the training data, the lowest value from the range is once again being chosen. **C=1** produces good metrics and model that will generalise well.







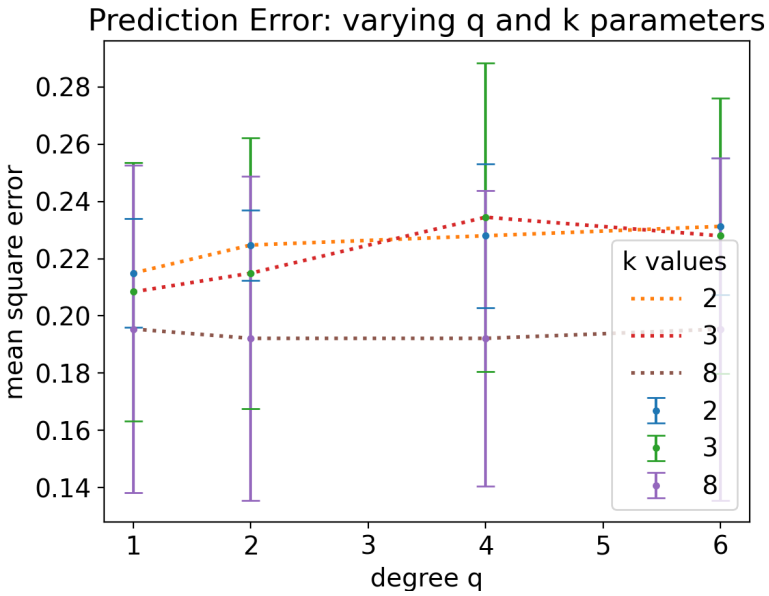
Two plots above show an underfit when C=0.001, every data point is predicted as +1. When C=1000, the prediction is very good – but it is an overfit and will not generalise well. C=1 was chosen. A plot for that can be seen on the previous page!

Code for C value tests:

```
C_means = []
C_std_devs = []
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
polyX = make_poly_features(X, 2)
for C_value in C_values:
  l2_log_reg = LogisticRegression(penalty='l2', C=C_value)
  l2_log_reg.fit(polyX, y)
  results = k_fold_cross_val(5, l2_log_reg, polyX)
  C_means.append(results[0])
  C_std_devs.append(results[1])
```

(b) k-Nearest Neighbours Classifier

A kNN classifier was trained on the dataset. Cross validation (5-fold) was used to measure the performance of the models. A quick cross validation was done on both the original two-feature dataset and on the data augmented with polynomial features.



This cross-validation plot suggests that polynomial features do not have a positive effect on the models.

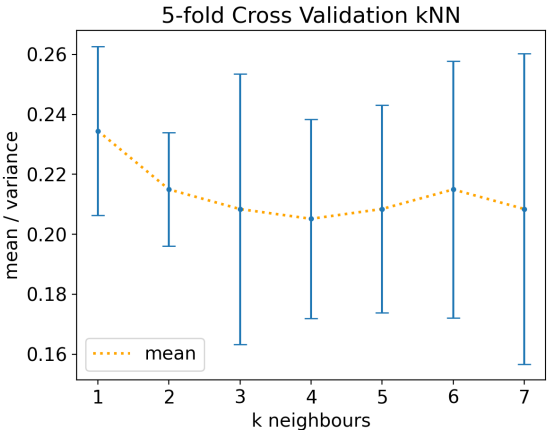
Results obtained from the original dataset (when q=1) are good. Beyond that the prediction errors increase. It is also possible that higher q values will lead to overfitting and this is to be avoided.

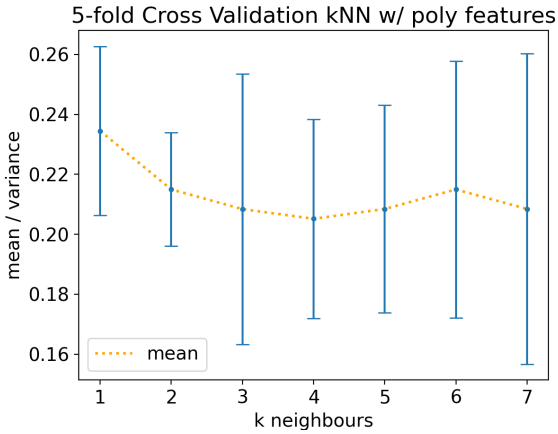
Due to the larger variance and the increasing error, **using polynomial features is <u>not</u> worth it in this case.**

Code for k value tests:

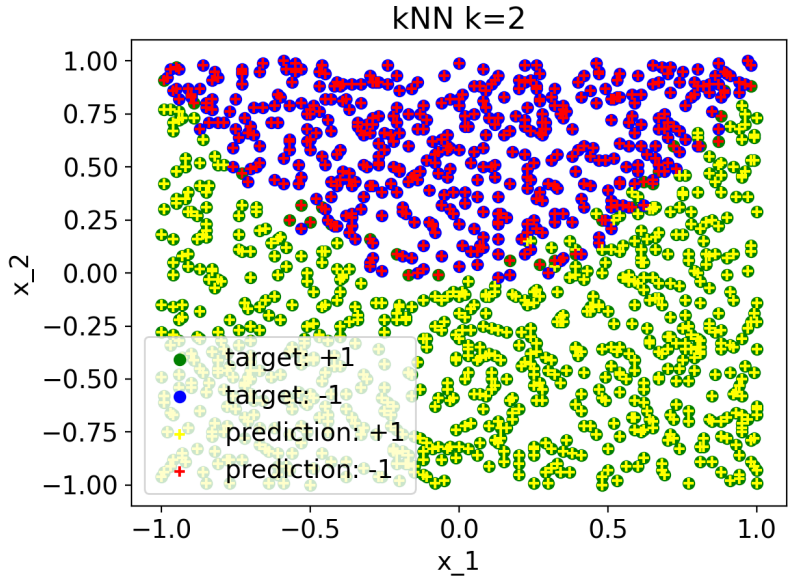
```
knn_means = []
knn_std_devs = []

k_vals = [1, 2, 3, 4, 5, 6, 7]
for k in k_vals:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(X, y)
  pred = knn.predict(X)
  print(f'kNN, k={k}, accuracy={knn.score(X, y)}')
  results = k_fold_cross_val(5, knn, X)
  knn_means.append(results[0])
  knn_std_devs.append(results[1])
```

This code was used again but with a polynomial X input, results on next page:





The above plots further prove that using polynomial features does not have a worthwhile effect on the dataset. The results stay almost exactly the same. It is also clear that the most reliable model has parameter **k=2**. When k is 4, 5, or 6 there is greater variance, suggesting that higher k values cause noise to be realised differently in each test set. Such a model will not generalise well and would perform badly on new data – it is therefore being avoided.



Target/predicted values plot of **kNN, k=2, original features (not polynomial).**

(c) Confusion Matrices

Confusion matrices were calculated for the chosen models based on the hyperparameter selections justified above. Key for the confusion matrices in the table below:

```
[ True Negatives  | False Positives ] --> [ TN| FP]
[ False Negatives | True Positives  ] --> [ FN| TP]
```

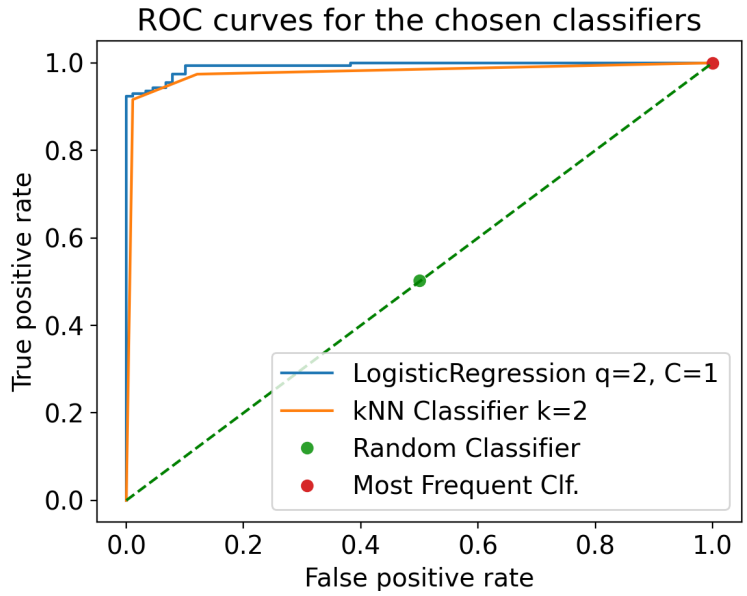
Logistic Regression **C=1, q=2**	*K-Nearest Neighbours* **k=2**	*Most Frequent Class* **baseline 1**	*Random Classifier* **baseline 2**
[426| 24] [25|753]	[438| 12] [19|753]	[0|450] [0|778]	[202|248] [381|397]

The most frequent classifier always predicts positive. There are no negative predictions. The classifiers appear to be functioning better than the baseline classifiers. We will now visualise their true positive and false positive rates with ROC curves.

Code: `confusion_matrix(y, y_pred)`

(d) ROC Curves

The chosen classifiers were plotted as ROC curves, along with points for the two baseline classifiers:



AUC (area under curve) of the classifiers:

Logistic Regression 0.995

k-Neighbours (kNN) 0.963

(e) Performance Evaluation and Comparison

Using the metric shown in parts (c) and (d) a number of conclusions can be drawn. Neither of the two classifiers is significantly better than the other. However it is very clear that they both completely outperform the most frequent and random baseline classifiers. This proves that the classifiers are very useful for this dataset – the models are finding patterns/structure in the data and effectively fitting to it – the data is not simply noise.

The ROC curve of a random classifier (provided the sample size is big enough) should lie along the 45 degree dashed green line. This corresponds to low accuracy i.e. a rate of 50:50 or simple chance.

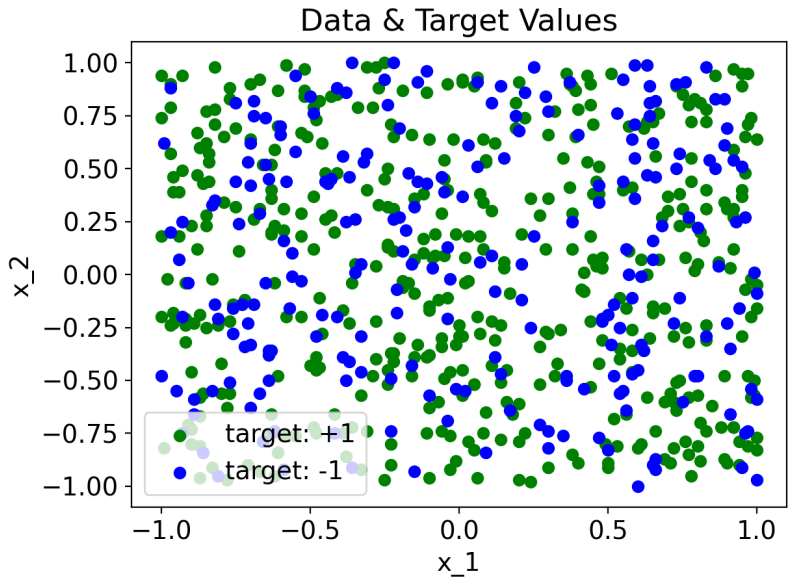
The ROC curve of the kNN and logistic regression classifiers perform much better – they reach high into the top left corner. The further into the top left and away from the diagonal the better the accuracy. The AUC can be used to summarise the models' performances. The kNN classifier has an AUC of around 3 per cent less than the logistic regression. The AUC is a general measure of prediction accuracy.

Both classifiers have similar accuracy and prediction error means/variances. They are both good choices. The logistic regression does have slightly better metrics. It also uses polynomial features which capture the quadratic shape of the data. It will also scale better than the kNN classifier (kNN's required computing power increases quickly – distance from each point to every other is calculated). Therefore, if the final use case will be on a big dataset I would recommend logistic regression.

(ii) Dataset 2

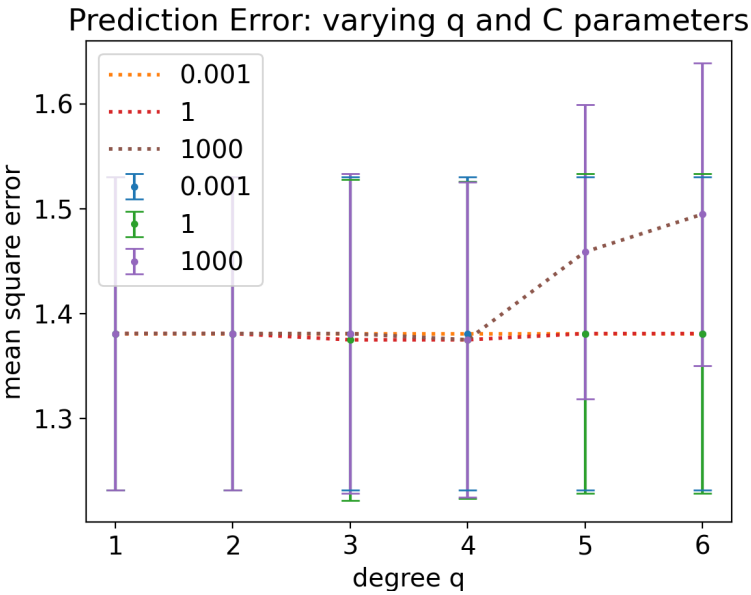
Note: excluding minor changes, the same code was used for dataset 2 as was used for dataset 1. Please refer to the previous parts for code snippets.

(a) Logistic Regression

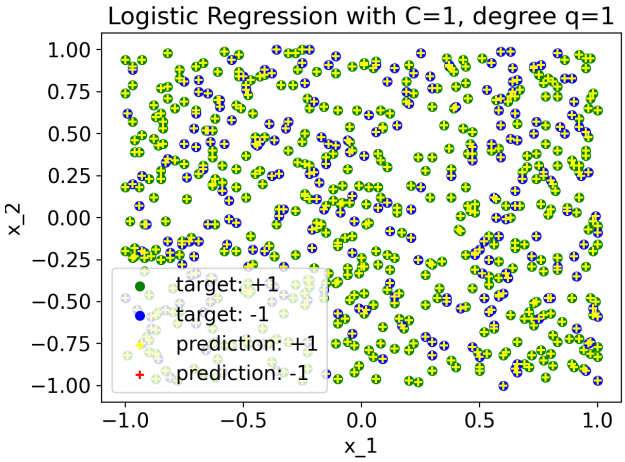


Once again, this dataset was plotted and colour coded by target value. In this dataset it is not immediately obvious if there is an underlying structure or pattern to the data. It is possible that it is just noise.

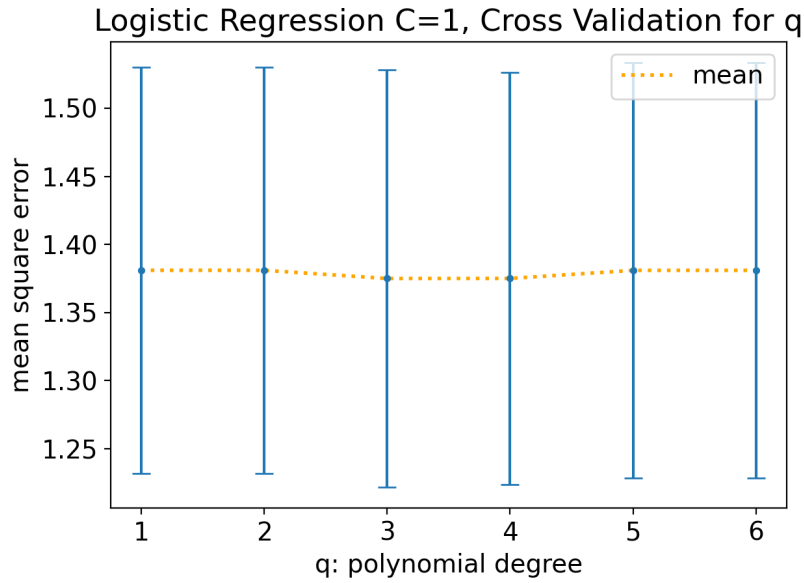
A wide range of C values was used to quickly get an overview of the data to choose a C for the following test to choose the order q of polynomial. The 5-fold cross validation error plot:



This time around the choice for C is not very clear. All C values with all degrees of q show relatively high error and variance. From the plot C=0.001 or C=1 should be chosen. C=1000 becomes erratic as q increases, thus is ruled out. We will choose **C=1** to keep the model simple and avoid possible underfitting with C=0.001.



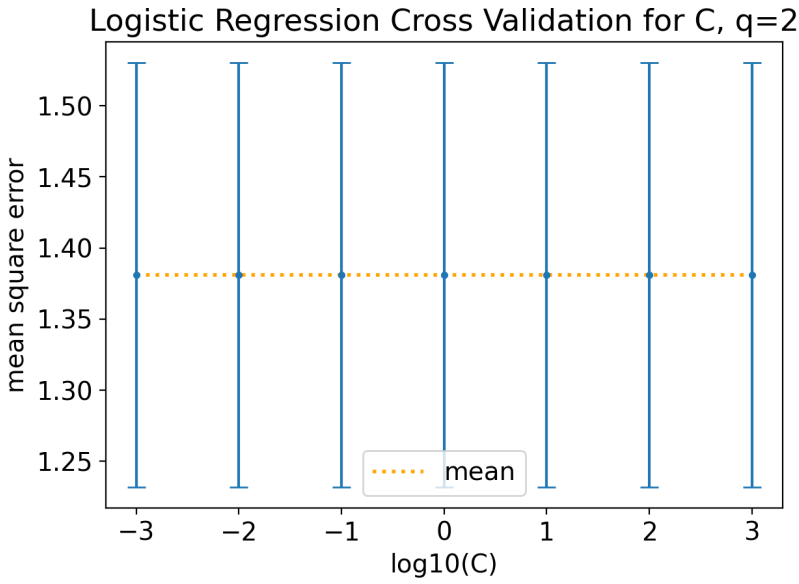
This plot (left) shows that the logistic regression classifier is predicting every data point as +1. This is the most frequent target value. The classifier is not finding any patterns or structure in the data. Every single plot for logistic regression looked the same – for all values q={1, 2, 3, 4, 5, 6} and for all values C={0.001, 0.01, 0.1, 1, 10, 100, 1000}.



Next, we use C=1 to attempt to find the best value for q, the order of the polynomial features. Once again, there is no clear choice. All values perform roughly the same. This suggests that the data has a structure which can be approximated with polynomial features. It could also point to the data simply being noise.

Despite this, we will choose **q=2**, as it performs almost identically to q=1 (no polynomial features) but it has a number of extra features which could help the model in some way, although currently it appears to not be the case.

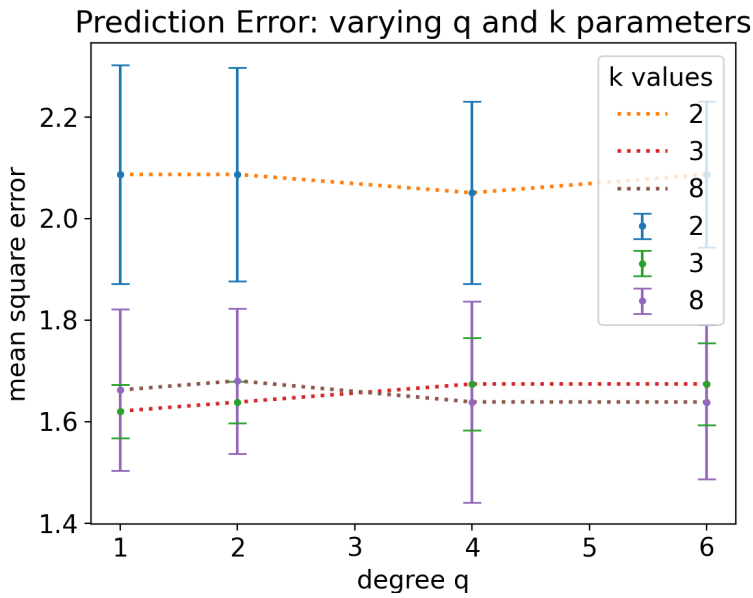
In the cross validation for C, using q=2, there is no clear choice for C. All values perform the exact same. They all have large variance and high prediction error. The models are performing no better than baseline models, see parts (c) and (d). Any of the C values could be chosen, but **C=1** will be chosen, a value in the middle of the range. This is to be sure there's no risk of under- or overfitting, despite the data likely only being noise.



(b) k-Nearest Neighbours Classifier

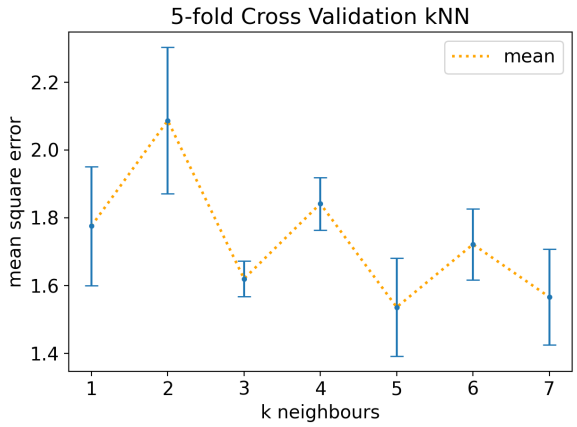
The second dataset was also used to train a kNN classifier. Cross validation was used to select k and to determine whether it is worth it or not to use polynomial features as an input.

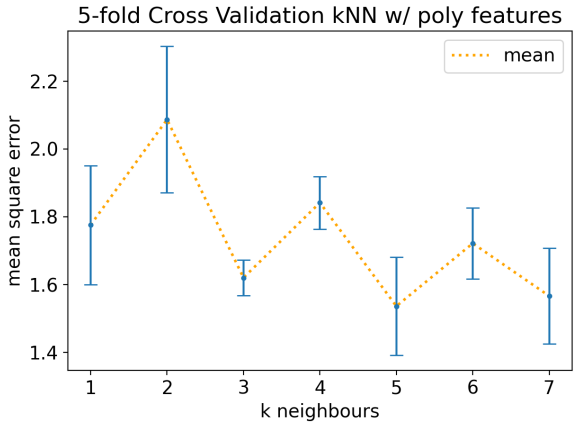
The following plot shows the mean and variance of the prediction error for kNN classifiers with variable q (order of polynomial) and k-nearest parameters.



Regardless of polynomial order, a value of k=2 has much lower performance when compared to the other k values. When k=2 the variance is also large. This should be avoided.

K=3 and k=8 have similar mean prediction errors, k=3 has a slightly better error and when k=8 the variance is much higher. There does not appear to be any reason to use polynomial features here. Despite this, we will choose the best performing polynomial value, q=2, and compare models of order 2 with models without polynomial features, see below:

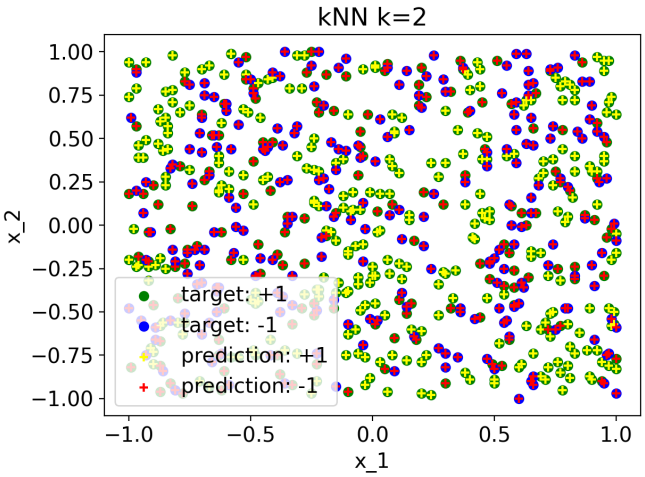


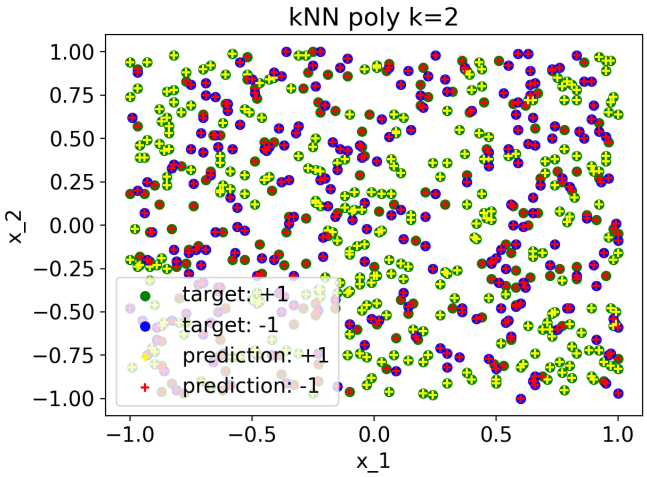


These plots further prove that **using polynomial features is <u>not</u> worth it** for this dataset. The results obtained are the same with or without.

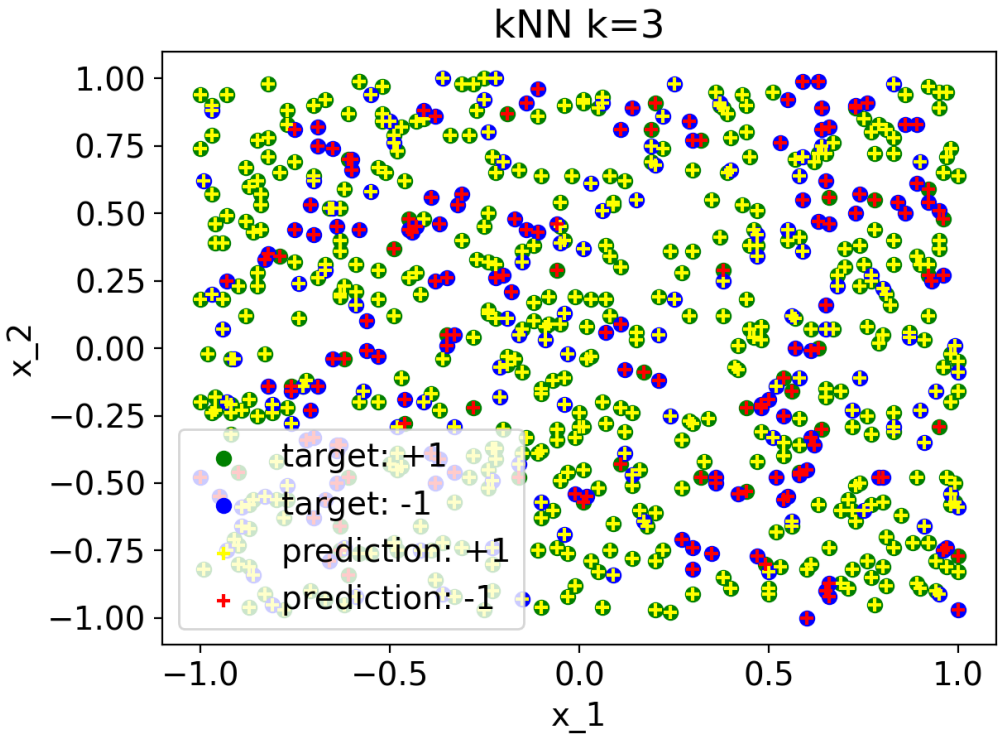
Values k=1 and k=1 display large variances and high error means. They will not be chosen. K=3 has the smallest variance and a respectable mean, it is a good candidate. When k=5, and similarly when k=7, the highest point of the variance is no higher than that of k=3. These could be chosen, but given that the variance is so large, it is suspected that in some of the train-test splits the models overfitted to the training data. In turn, the models performed badly on the test data. This overfitting should be avoided, because such a model will not generalise well. For these reasons, **k=3 is the best choice.**

Plots for kNN, variable k values, with/without polynomial features:





The two above plots are very similar. Augmenting to make polynomial features is not worth it. There is no improvement in prediction errors or variance (see error bar plots on previous page). For every kNN plot, the plot of that model with polynomial features was the same.



This scatter plot shows the **chosen model (k=3).** You can see the kNN model is trying to find clusters in the data. It does to some extent, but there is also a number of incorrect predictions.

(c) Confusion Matrices

Confusion matrices were calculated for the chosen models based on the hyperparameter selections justified above. Key for the confusion matrices in the table below:

```
[ True Negatives  | False Positives ] --> [ TN| FP]
[ False Negatives | True Positives  ] --> [ FN| TP]
```

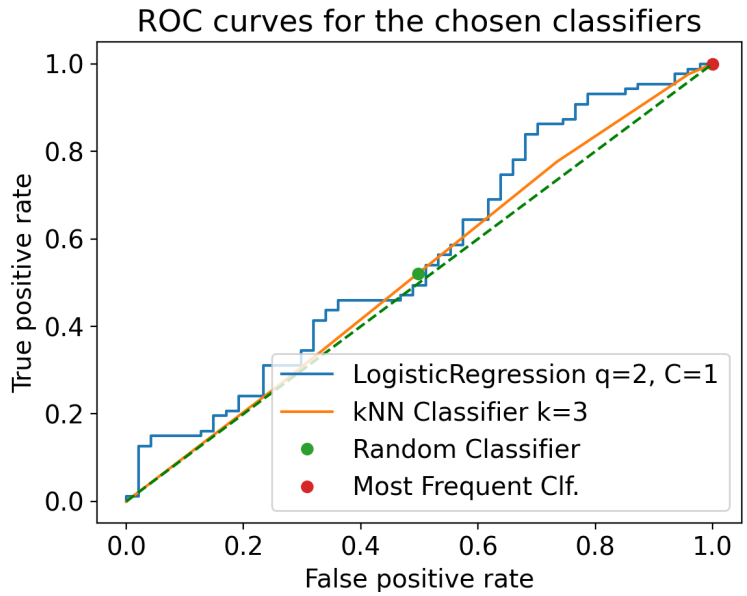
Logistic Regression **C=1, q=2**	*K-Nearest Neighbours* **k=3**	*Most Frequent Class* **baseline 1**	*Random Classifier* **baseline 2**
[0|231] [0|438]	[125|106] [41|397]	[0|231] [0|438]	[126|105] [228|210]

Based on these confusion matrices we can see that the logistic regression classifier is behaving no differently to the most frequent classifier. Logistic regression is definitely not suitable for this dataset.

The kNN classifier is also not performing well. This is even more clear with the ROC curve in part (d). The kNN classifier has a similar number of true negatives and false positives when compared with the random classifier.

(d) ROC Curves

The classifiers' ROC curves were plotted. The baseline classifiers were also plotted as points.



AUC (area under curve) of the classifiers:

Logistic Regression 0.559

k-Neighbours (kNN) 0.519

The AUC for both classifiers is very close to that of a random classifier (0.5). These models are not performing well.

(e) Performance Evaluation and Comparison

Neither of the two classifiers is significantly better than the other. The logistic regression and kNN classifiers perform similarly to the most frequent and random baseline classifiers. This shows that the classifiers are not modelling the data effectively and not making good predictions. The models are not finding patterns/structure in the data, despite best efforts to tune model hyperparameters. This suggests that the second dataset is little more than noise.

The ROC curve of a random classifier (provided the sample size is big enough) should lie along the 45 degree dashed green line. This corresponds to low accuracy i.e. a rate of 50:50 or simple chance. As mentioned, the classifiers are not performing much better than this.

The ROC curves of the kNN and logistic regression classifiers perform poorly – they do not reach into the top left corner. The closer the curves are to the diagonal green line, the worse the performance. Curves closer to the diagonal have poor accuracies. The AUC can be used to summarise the models' performances. Both the classifiers have poor AUC, and thus poor performance.

As a result of the poor performances, I would recommend neither classifiers for this dataset. The baseline classifiers performed better – this is a very bad sign. This dataset is likely just noise. These models are not suitable for noisy training data.

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.model_selection import train_test_split


# read in data (choose dataset by commenting)
# if changing dataset also change line 184/185!
# df = pd.read_csv("week4_1.csv", comment='#', header=None)
df = pd.read_csv("week4_2.csv", comment='#', header=None)
print(df.head())
X1 = df.iloc[:, 0]
X2 = df.iloc[:, 1]
X = np.column_stack((X1, X2))
y = np.array(df.iloc[:, 2])


def do_2d_plot(x, pred, title):
  plt.rc('font', size=14)
  plt.scatter(x[:, 0][y==1], x[:, 1][y==1], color='green', marker='o',
label='target: +1')
  plt.scatter(x[:, 0][y==-1], x[:, 1][y==-1], color='blue', marker='o',
label='target: -1')
  if (y-pred).any():
      plt.scatter(x[:, 0][pred==1], x[:, 1][pred==1], color='yellow',
marker='+', label='prediction: +1')
      plt.scatter(x[:, 0][pred==-1], x[:, 1][pred==-1], color='red', marker='+',
label='prediction: -1')
  plt.title(title)
  plt.xlabel('x_1')
  plt.ylabel('x_2')
  plt.legend(loc='lower left')
  plt.tight_layout()
  plt.show()


def do_error_plot(x, means, yerr, title, x_label):
  plt.errorbar(x, means, yerr=yerr, fmt='.', capsize=5)
  plt.plot(x, means, linestyle=':', label='mean', linewidth=2, color='orange')
  plt.legend()
  plt.title(title)
  plt.xlabel(x_label)
  plt.ylabel('mean square error')
  plt.tight_layout()
  plt.show()
```

```
def make_poly_features(x, degree):
  poly = PolynomialFeatures(degree=degree)
  new_x = poly.fit_transform(x)
  return new_x


def k_fold_cross_val(k, model, input):
  k_fold = KFold(n_splits=k)
  sq_errs = []
  for train, test in k_fold.split(input):
      curr_model = model.fit(input[train], y[train])
      pred = curr_model.predict(input[test])
      sq_err = mean_squared_error(y[test], pred)
      sq_errs.append(sq_err)
  mean = np.mean(sq_errs)
  std = np.std(sq_errs)
  return mean, std


do_2d_plot(X, y, 'Data & Target Values')

""" (i)(a) i. polynomial features and logistic regression with L2 penalty """
cc = [0.001, 1, 1000]
qq = [1, 2, 3, 4, 5, 6]
for c in cc:
  mm = []
  sd = []
  for q in qq:
      poly = make_poly_features(X, q)
      log_clf = LogisticRegression(penalty='l2', C=c, max_iter=100000)
      log_clf.fit(poly, y)
      results = k_fold_cross_val(5, log_clf, poly)
      mm.append(results[0])
      sd.append(results[1])
  plt.errorbar(qq, mm, yerr=sd, fmt='.', capsize=5, label=c)
  plt.plot(qq, mm, linestyle=':', label=c, linewidth=2)
plt.ylabel('mean square error')
plt.title('Prediction Error: varying q and C parameters')
plt.xlabel('degree q')
plt.legend()
plt.show()

kk = [2, 3, 8]
qq = [1, 2, 4, 6]
for k in kk:
  mm = []
  sd = []
  for q in qq:
      poly = make_poly_features(X, q)
      log_clf = KNeighborsClassifier(n_neighbors=k)
      log_clf.fit(poly, y)
      results = k_fold_cross_val(5, log_clf, poly)
      mm.append(results[0])
      sd.append(results[1])
  plt.errorbar(qq, mm, yerr=sd, fmt='.', capsize=5, label=k)
  plt.plot(qq, mm, linestyle=':', label=k, linewidth=2)
plt.ylabel('mean square error')
plt.title('Prediction Error: varying q and k parameters')
```

```
plt.xlabel('degree q')
plt.legend(title='k values')
plt.show()

q_means = []
q_std_devs = []
degrees = [1, 2, 3, 4, 5, 6]
for deg in degrees:
  polyX = make_poly_features(X, deg)
  l2_log_reg = LogisticRegression(penalty='l2', C=1)
  l2_log_reg.fit(polyX, y)
  pred = l2_log_reg.predict(polyX)
  print(f'C=1, degree={deg}, accuracy={l2_log_reg.score(polyX, y)}')
  results = k_fold_cross_val(5, l2_log_reg, polyX)
  q_means.append(results[0])
  q_std_devs.append(results[1])
  do_2d_plot(X, pred, f'Logistic Regression with C=1, degree q={deg}')
do_error_plot(degrees, q_means, q_std_devs, 'Logistic Regression C=1, Cross
Validation for q', 'q: polynomial degree')

""" (i)(a) ii. C parameter and logistic regression with L2 penalty """
C_means = []
C_std_devs = []
C_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
polyX = make_poly_features(X, 2)
for C_value in C_values:
  l2_log_reg = LogisticRegression(penalty='l2', C=C_value)
  l2_log_reg.fit(polyX, y)
  print(f'C={C_value}, degree=2, accuracy={l2_log_reg.score(polyX, y)}')
  results = k_fold_cross_val(5, l2_log_reg, polyX)
  C_means.append(results[0])
  C_std_devs.append(results[1])
  y_pred = l2_log_reg.predict(polyX)
  do_2d_plot(X, y_pred, f'Logistic Regression with C={C_value}, degree q=2')
do_error_plot(np.log10(C_values), C_means, C_std_devs, 'Logistic Regression
Cross Validation for C, q=2', 'log10(C)')

""" (i)(b) kNN classifier """
knn_means = []
knn_std_devs = []
knn_poly_means = []
knn_poly_std_devs = []

k_vals = [1, 2, 3, 4, 5, 6, 7]
for k in k_vals:
  knn = KNeighborsClassifier(n_neighbors=k)
  knn.fit(X, y)
  pred = knn.predict(X)
  print(f'kNN, k={k}, accuracy={knn.score(X, y)}')
  results = k_fold_cross_val(5, knn, X)
  knn_means.append(results[0])
  knn_std_devs.append(results[1])
  do_2d_plot(X, pred, f'kNN k={k}')

  knn.fit(polyX, y)
  knn_poly_pred = knn.predict(polyX)
  print(f'kNN-poly, k={k}, accuracy={knn.score(polyX, y)}')
  poly_results = k_fold_cross_val(5, knn, polyX)
```

```
  knn_poly_means.append(results[0])
  knn_poly_std_devs.append(results[1])
  do_2d_plot(X, knn_poly_pred, f'kNN poly k={k}')

do_error_plot(k_vals, knn_means, knn_std_devs, '5-fold Cross Validation kNN', 'k
neighbours')
do_error_plot(k_vals, knn_poly_means, knn_poly_std_devs, '5-fold Cross
Validation kNN w/ poly features', 'k neighbours')


""" (i)(c) confusion matrices """
models = []
""" final logistic regression classifier, q=2, C=1 """
log_reg_clf = LogisticRegression(C=1, penalty='l2')
log_reg_clf.fit(polyX, y)
log_reg_pred = log_reg_clf.predict(polyX)
print('log_reg\n', confusion_matrix(y, log_reg_pred))  # tn, fp, tp, fn
models.append(log_reg_clf)

""" final knn classifier, k=2, NOT poly """
# knn_clf = KNeighborsClassifier(n_neighbors=2)  # dataset1
knn_clf = KNeighborsClassifier(n_neighbors=3)  # dataset2
knn_clf.fit(X, y)
knn_pred = knn_clf.predict(X)
print('knn\n', confusion_matrix(y, knn_pred))
models.append(knn_clf)

""" most frequent dummy classifier """
most_freq_clf = DummyClassifier(strategy='most_frequent')
most_freq_clf.fit(polyX, y)
most_freq_pred = most_freq_clf.predict(polyX)
most_freq_conf_mat = confusion_matrix(y, most_freq_pred)
print('most_freq\n', most_freq_conf_mat)
models.append(most_freq_clf)

""" random classifier """
rand_clf = DummyClassifier(strategy='uniform')
rand_clf.fit(polyX, y)
rand_pred = rand_clf.predict(polyX)
rand_conf_mat = confusion_matrix(y, rand_pred)
print('rand\n', rand_conf_mat)
models.append(rand_conf_mat)

""" (i)(d) ROC curves """
Xtrain, Xtest, ytrain, ytest = train_test_split(polyX, y, test_size=0.2)
# polynomial features for LogReg
for model in models[:2]:
  model.fit(Xtrain, ytrain)
  scores = model.predict_proba(Xtest)
  fpr, tpr, _ = roc_curve(ytest, scores[:, 1])
  print(auc(fpr, tpr))
  model_name = type(model).__name__
  if model_name == 'LogisticRegression':
      model_name = 'LogisticRegression q=2, C=1'
  else:
      model_name = 'kNN Classifier k=3'
  plt.plot(fpr, tpr, label=model_name)
  Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)  # switch
```

```
to normal features for kNN iteration

""" plotting points of the baseline clfs: rand and most_freq """
rand_fpr = rand_conf_mat[0][1] / (rand_conf_mat[0][1] + rand_conf_mat[0][0])
# FP / (FP + TN)
rand_tpr = rand_conf_mat[1][1] / (rand_conf_mat[1][1] + rand_conf_mat[1][0])
# TP / (TP + FN)
most_freq_fpr = most_freq_conf_mat[0][1] / (most_freq_conf_mat[0][1] +
most_freq_conf_mat[0][0])  # FP / (FP + TN)
most_freq_tpr = most_freq_conf_mat[1][1] / (most_freq_conf_mat[1][1] +
most_freq_conf_mat[1][0])  # TP / (TP + FN)

plt.plot(rand_fpr, rand_tpr, label='Random Classifier', marker='o',
linestyle='None')
plt.plot(most_freq_fpr, most_freq_tpr, label='Most Frequent Clf.', marker='o',
linestyle='None')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.title('ROC curves for the chosen classifiers')
plt.legend()
plt.show()
```