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

CS7CS4/CSU44061 - Machine Learning
Assignment 3:

Student Name:
Hugh Jordan

Student Number:
16321743

Assignment 3

(i) Lasso & Ridge Regression Models

(a) The downloaded data (# id:21--42-21) was plotted as a 3D scatter plot using Matplotlib's scatter plot. The segment of code used to read in the data is shown in Figure 1. This segment reads in the two input features (X1 & X2) as well as the target value (y). The data was then plotted using Matplotlib's scatter plot.

```
14 df = pd.read_csv("week3.csv",comment='#')
15 X1=df.iloc[:, 0]
16 X2=df.iloc[:, 1]
17 X=np.column_stack((X1, X2))
18 y=df.iloc[:, 2]
19 y = np.array(df.iloc[:, 2])
20 y = np.reshape(y, (-1,1))
```

Figure 1. Read in the Data

The scatter plot is shown below in Figure 2. It is evident from Figure 2 that the data lies on a curve and not a plane. This suggests that the model will require some combination of powers of the two features provided to capture the relationship between the training data and the target values.

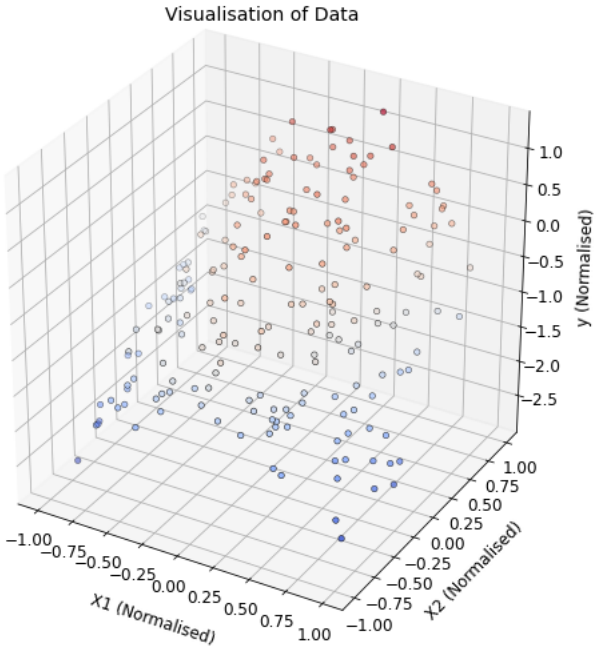

Figure 2. Visualisation of Data

(b) In addition to the two features in the data file, each combination of powers of these two features up to the power of 5 were added as extra features using Sklearn's PolynomialFeatures. A Lasso regression model was then trained using these polynomial features for a large range of values of C. The segment of code used to perform this is shown in Figure 3.

```
q = 5
Xpoly = PolynomialFeatures(q).fit_transform(X)

lassos = []
penalties = [1, 10, 1000]
for itr, penalty in enumerate(penalties):
    model = Lasso(alpha=1/(2*penalty))
    model.fit(Xpoly, y)
    lassos.append(model)
```

Figure 3. Train Lasso Regression Model

The parameter values for each of these models are shown below in Table 1. It is evident from Table 1 that when C is equal to 1, all the parameter values are equal to 0. However, as C increases, the weight of each of the parameter values also increases. This is because the Lasso regression model uses L1 regularisation. L1 regularisation adds the sum of the absolute value of the parameter values as a penalty to the loss function. This type of regularisation has the desirable effect of shrinking less important parameter values to 0. This process can reduce the complexity of the model by removing some features altogether, thus reducing overfitting in the model.

C	θ_0	θ_{X1}	θ_{X2}	θ_{X1^5}	θ_{X2^5}
1	-0.6688106	0	0	0	0
10	-0.23122063	0	0.82098556	-1.31367627	0
1000	-0.01812302	-0.00356471	0.91978043	-1.97844553	0.1567035

Table 1. Lasso Parameter values for range of penalty values

(c) The three models from (b) were used to generate predictions for the target variable. These predictions were generated using a grid of the feature values. The segment of code used to generate this grid is shown in Figure 4. This segment divides the two input parameters into 30 segments. A 1D array was then used to store each set of these values. This is because PolynomialFeatures takes a 1D array as an input instead of a 2D array. However, Matplotlib's plot_surface function takes a 2D array as an input instead of a 1D array. Consequently, the 1D array was later reshaped into a 2D array later to form the grid.

```
X_list =[]
X1_min, X1_max = X[:, 0].min() - 4, X[:, 0].max() + 4
X2_min, X2_max = X[:, 1].min() - 4, X[:, 1].max() + 4
X1_list = np.linspace(X1_min, X1_max, 30)
X2_list = np.linspace(X2_min, X2_max, 30)
for i in X1_list:
    for j in X2_list:
        X_list.append([i ,j])
X_list = np.array(X_list)
Xpoly_list = PolynomialFeatures(q).fit_transform(X_list)
```

Figure 4. Grid of Feature Values

The model's predictions and are shown in Figure 5. It is evident from Figure 5 that the complexity of the model increases as the value of C increases. This increase in the complexity of the model increases the accuracy of the model's predictions; that is until the model is fully able to capture the relationship

between the training data and the target values. After this point, the accuracy of the model's predictions begins to decrease.

It is evident from Figure 5 that when C is equal to 1, the model underfits the data. This is evident because the generated predictions lie on a flat plane. However, it was stated in (a) that the underlying data lies on a curve. Therefore, the weight of the penalty parameter is too large, and the model underfits the data as a result. However, when C is equal to 10, the model accurately fits the underlying data. This is evident in Figure 5 because the model is able to fully capture the relationship between the training data and the target values.

Conversely, when C is equal to 1000, the model overfits the underlying data. This is evident in Figure 5 from the model's poor ability to generalise. This becomes most apparent when the model predicts that y = 556.98 when X1=5, X2=5 and C = 1000. This prediction is in stark contrast to the model's prediction that y = 3.84 when X1=5, X2=5 and C = 10. This is because as C increases, the weight of the parameter values also increases. Therefore, the complexity of the model increases and the model is more likely to suffer from overfitting.

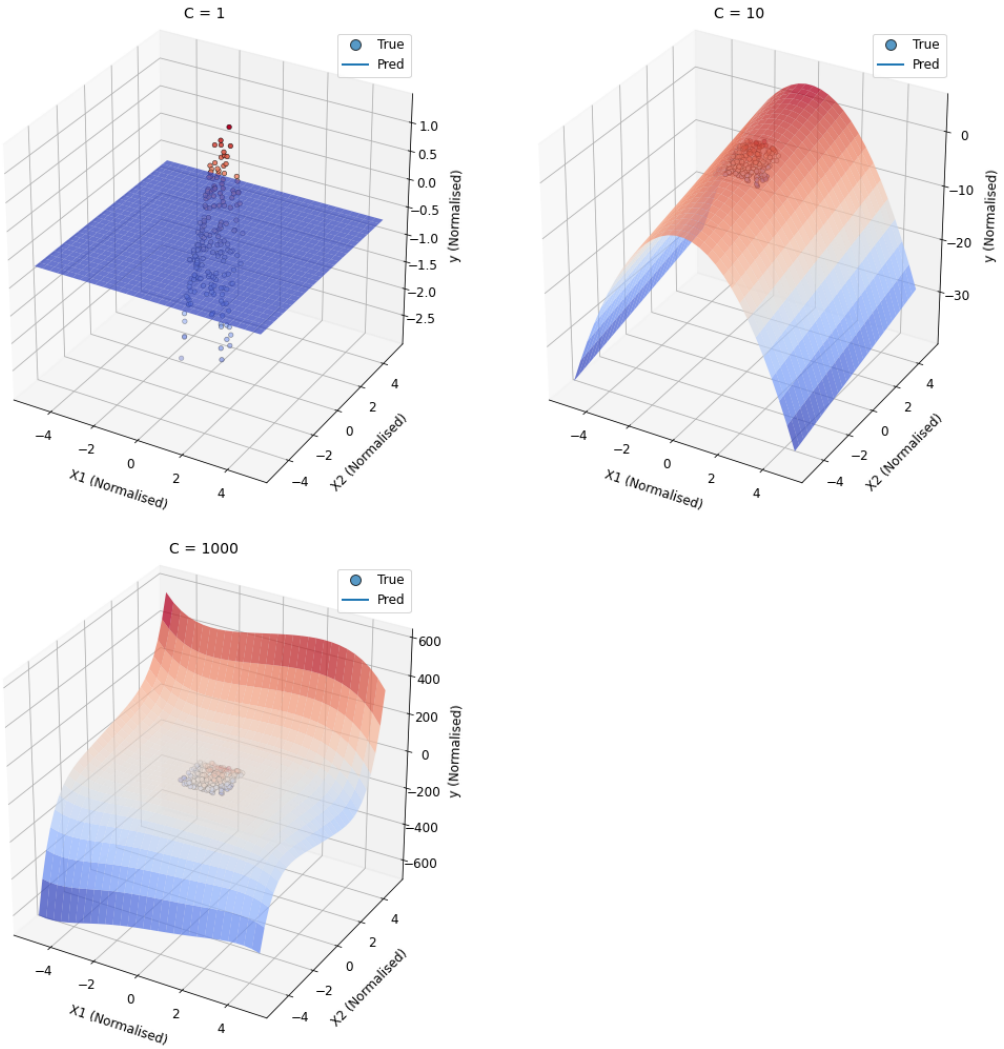

Figure 5. Lasso Classifier with a range of penalty values

(d) Underfitting occurs when the model is too simple and is unable to capture the relationship between the training data and the target values. Underfitting can occur as a result of using polynomial features with q = 1 when the relationship is quadratic. However, overfitting occurs when the model is too complex and starts to fit the "noise" in the training data, negatively impacting the performance of the model. Overfitting can occur as a result of using polynomial features with q = 5 when the relationship is quadratic.

The penalty weight parameter C can be used to strike the right balance between overfitting and underfitting. As the degree of the input features increases, the model becomes more complex and tries to fit each of the data points as closely as possible. However, the model starts fitting the "noise" in the data instead. The penalty weight parameter C can be used to simplify the model. As the value of C decreases, the parameter weight values decrease. This simplifies the model and reduces overfitting. However, this process penalises all of the parameter weight values. Therefore, if the value of C is too small, the weight of all of the parameter values will become too small, and the model will underfit the data. This is because the model is too simple and cannot capture the relationship between the training data and the target values.

(e) A Ridge regression model was trained using the same polynomial features as the Lasso regression model for a large range of values of C. The parameter values for each of these models are shown below in Table 2. It is evident from Table 2 that when C is equal to 1, all the parameter values are small. However, as C increases, the weight of each of the parameter values also increases. This is because the Ridge regression model uses L2 regularisation. L2 regularisation adds the sum of the square of the parameter values as a penalty to the loss function. This type of regularisation forces weights to be small and reduces overfitting. However, it does not force them to be zero. Therefore, it does not perform feature selection like L1 regularisation.

C	$\boldsymbol{\theta_o}$	$\boldsymbol{\theta_{X1}}$	$\boldsymbol{\theta_{X2}}$	$\boldsymbol{\theta_{X1^5}}$	$\boldsymbol{\theta_{X2^5}}$
1	-0.07686546	-0.02279312	0.86319876	-1.58505668	0.06062172
10	-0.01362624	-0.04652767	0.97742408	-1.98470516	0.24604138
1000	-0.00201892	-0.05476305	1.04933051	-2.0609864	0.39830094

Table 2. Ridge Parameter values for range of penalty values

The three models were then used to generate predictions for the target variable. These predictions are shown in Figure 6. It is evident from Figure 6 that the complexity of the model increases as the value of C increases. Similar to Lasso regression, this increase in the complexity of the model increases the accuracy of the model's predictions; that is until the model is able to fully capture the relationship between the training data and the target values. After this point, the accuracy of the model's predictions begins to decrease.

It is evident from Figure 6 that when C is equal to 1, the model fits the underlying data. This is evident because the model is fully able to capture the relationship

between the training data and the target values. However, the model appears to overfit the data. This is evident from the model's poor ability to generalise. This poor generalisation is most apparent when the model predicts that y =217.86 when X1=5, X2=5 and C=1. This prediction is very similar to the Lasso regression model's prediction that y = 556.98 when X1=5, X2=5 and C = 1000. This is because as C increases, the weight of the parameter values also increases. Therefore, the complexity of the model increases and the model is more likely to suffer from overfitting. This is further supported when C is equal to 10 or 1000.

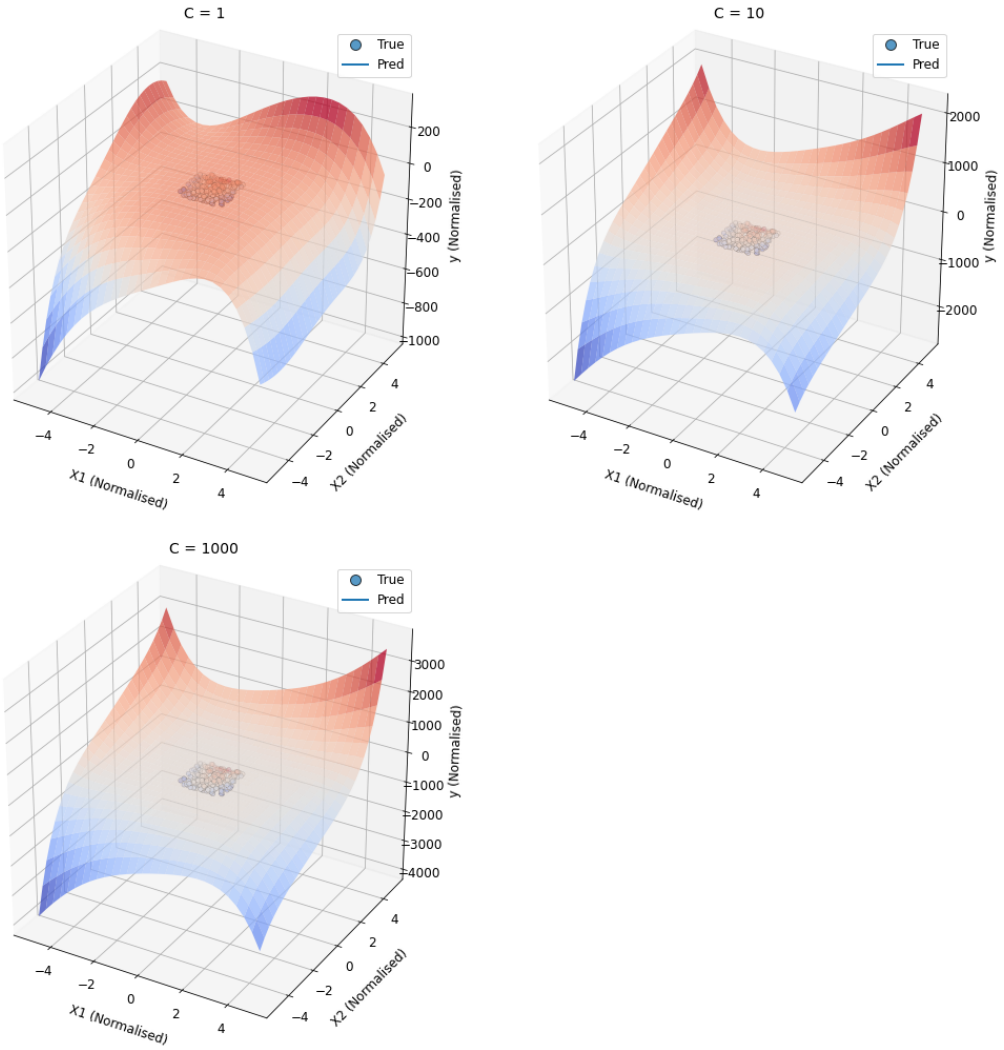

Figure 6. Ridge Classifier with a range of penalty values

(ii) K-Fold Cross-Validation

(a) The mean and variance of the model's predicted error were calculated for a range of k-fold values, but the mean and standard deviation are both displayed in Figure 7. The mean and standard deviation were plotted succinctly using Matplotlib's error-bar function. It is evident from Figure 7 that as the number of folds increases, the mean of the predicted error settles. However, the standard deviation increases as the number of folds increases. Therefore, the mean of the predicted error is more likely to have settled by choosing a larger value of k.

However, the variance of the predicted error will continue to increase. Additionally, it is computationally expensive to create multiple models and calculate the mean and variance of each of these models. Consequently, I would recommend using k=5. It is evident from Figure 7 that the mean of the predicted error has settled, and the variance is lower when compared to a larger number of folds.

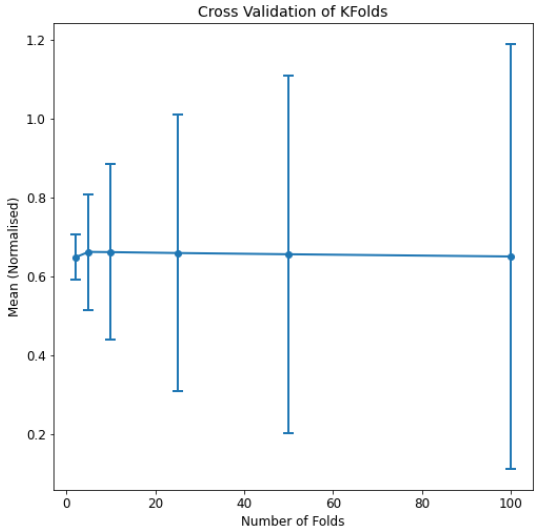

Figure 7. Cross Validation of K-Folds

(b) The mean and standard deviation of the Lasso model's predicted error were calculated for a range of values of C using 5-Fold cross-validation. This is shown in Figure 8. The range of C values chosen was [0.1, 0.5, 1, 5, 10, 50, 100]. This range was chosen because the value of C increases by a factor of 10 after every second iteration allowing the quick analysis of a large range of values.

(c) I would recommend using a value of 50 for C. It is evident from Figure 8 that the mean and standard deviation of the predicted error decrease as the value of C increases; that is, until the value of C reaches 50. After that point, any increase in the value of C only brings about a negligible decrease in the mean and standard deviation of the predicted error. Therefore, a value of 50 or 100 would both be suitable. However, a value of 50 should be chosen instead of 100 for C because the lowest value of C possible should be used to avoid overfitting.

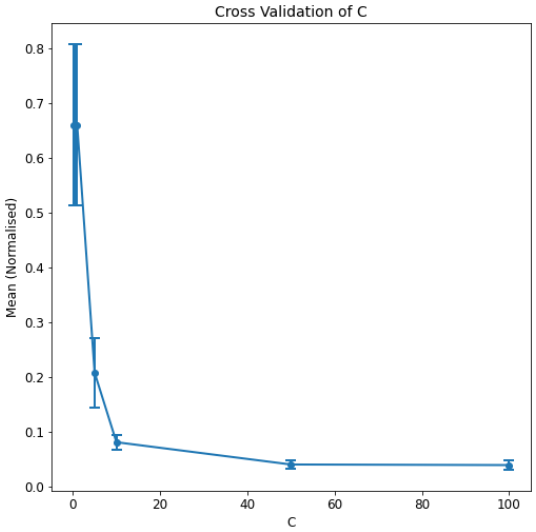

Figure 8. Cross-Validation of C using Lasso model

(d) The mean and standard deviation of the Ridge model's predicted error were calculated for a range of values of C using 5-Fold cross validation. This is shown in Figure 9. The same range of C values chosen for the lasso regression model was selected for the ridge regression model. This range was selected because the value of C increases by a factor of 10 after every second iteration allowing the quick analysis of a large range of values.

I would recommend using a value of 5 for C. It is evident from Figure 9 that the mean of the predicted error decreases as the value of C increases; that is, until the value of C reaches 5. After that point, any increase in the value of C brings about a small increase in the mean predicted error. This increase in the mean of the predicted error is a result of overfitting. Conversely, an increase in C brings about no change in the variance.

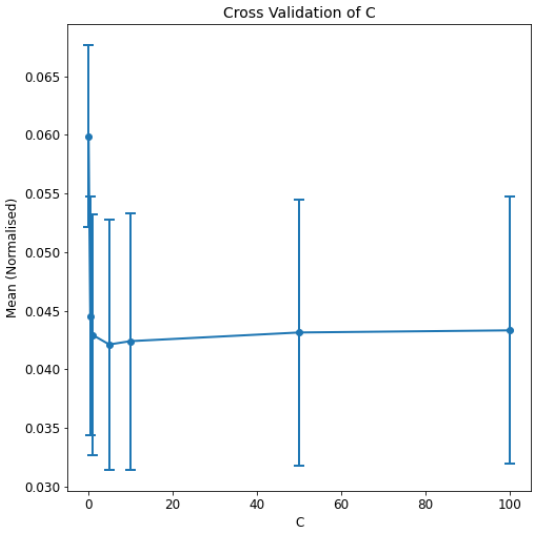

Figure 9. Cross-Validation of C using Ridge model

Finally, I would recommend using the Lasso regression model on the downloaded data instead of the Ridge regression model. This is because the Lasso regression model uses L1 regularisation, which has the desirable effect of shrinking less important parameters to 0. Consequently, it reduces the weights of higher-order parameters to 0, thus reducing the complexity of the model and therefore, overfitting.

However, the Ridge regression model uses L2 regularisation, which forces the weights to be small. This type of regularisation does not lower the weights to the value of 0. As a result, L2 regularisation is not as robust when it comes to outliers. This is because the square term inflates the predicted error of outliers and the regularisation term attempts to fix it by penalising all of the weights. The Ridge regression model is more suitable when all the input features influence the target value, and the value of the weights are all similar in size.

Appendix:

```
import numpy as np
import pandas as pd
import math
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

df = pd.read_csv("week3.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, projection='3d')
ax.set_title("Visualisation of Data", fontsize=14, pad=14)
ax.set_xlabel("X1 (Normalised)", fontsize=12, labelpad=12)
ax.set_ylabel("X2 (Normalised)", fontsize=12, labelpad=12)
ax.set_zlabel("y (Normalised)", fontsize=12, labelpad=12)
ax.scatter(X[:, 0], X[:, 1], y, c=y, edgecolor='black', linewidth=0.5, cmap='coolwarm')
plt.savefig('data_visualisation')
plt.show()

q = 5
Xpoly = PolynomialFeatures(q).fit_transform(X)

lassos = []
penalties = [1, 10, 1000]
for itr, penalty in enumerate(penalties):
   model = Lasso(alpha=1/(2*penalty))
   model.fit(Xpoly, y)
   lassos.append(model)
print('Parameter Values Table:')
print('%-8s %-21s %-20s' % ('Penalty', 'Intercept', 'Slope'))
for itr, model in enumerate(lassos):
   print('%-8s %-21a %-20a\n' % (penalties[itr], model.intercept_, model.coef_))

X_list =[]
```

```
X1_min, X1_max = X[:, 0].min() - 4, X[:, 0].max() + 4
X2_min, X2_max = X[:, 1].min() - 4, X[:, 1].max() + 4
X1_list = np.linspace(X1_min, X1_max, 30)
X2_list = np.linspace(X2_min, X2_max, 30)
for i in X1_list:
   for j in X2_list:
      X_list.append([i ,j])
X_list = np.array(X_list)
Xpoly_list = PolynomialFeatures(q).fit_transform(X_list)

fig = plt.figure(figsize=(14, 14));
fig.tight_layout(pad=4.0)
for itr, model in enumerate(lassos):
   y_pred = model.predict(Xpoly_list)
   Xpoly_grid = np.reshape(Xpoly_list, (30,30,-1))
   y_pred = np.reshape(y_pred, (30,30))
   ax = fig.add_subplot(2, 2, itr + 1, projection='3d')
   ax.set_title("C = " + str(penalties[itr]), fontsize=14, pad=18)
   ax.set_xlabel("X1 (Normalised)", fontsize=12, labelpad=12)
   ax.set_ylabel("X2 (Normalised)", fontsize=12, labelpad=12)
   ax.set_zlabel("y (Normalised)", fontsize=12, labelpad=12)
   ax.scatter(X[:, 0], X[:, 1], y, c=y, cmap='coolwarm', edgecolor='black', linewidth=0.5)
   ax.plot_surface(Xpoly_grid[:,:,1], Xpoly_grid[:,:,2], y_pred ,cmap='coolwarm',
edgecolor='none', alpha=0.75)
   legend_elements = [Line2D([0], [0], marker='o', markeredgecolor='black', linewidth=0.5,
                        label='True', linestyle = 'None', markersize=10, alpha=0.75),
                  Line2D([0],[0], linewidth=2.0, label='Pred')]
   ax.legend(handles=legend_elements, bbox_to_anchor=(0.91,0.94))
plt.savefig('lasso_classifiers')
plt.show()

ridges = []
for itr, penalty in enumerate(penalties):
   model = Ridge(alpha=1/(2*penalty))
   model.fit(Xpoly, y)
   ridges.append(model)
print('Parameter Values Table:')
print('%-8s %-21s %-20s' % ('Penalty', 'Intercept', 'Slope'))
for itr, model in enumerate(ridges):
   print('%-8s %-21a %-20a\n' % (penalties[itr], model.intercept_, model.coef_))

fig = plt.figure(figsize=(14, 14));
fig.tight_layout(pad=4.0)
for itr, model in enumerate(ridges):
   y_pred = model.predict(Xpoly_list)
   Xpoly_grid = np.reshape(Xpoly_list, (30,30,-1))
   y_pred = np.reshape(y_pred, (30,30))
```

```
    ax = fig.add_subplot(2, 2, itr + 1, projection='3d')
    ax.set_title("C = " + str(penalties[itr]), fontsize=14, pad=18)
    ax.set_xlabel("X1 (Normalised)", fontsize=12, labelpad=12)
    ax.set_ylabel("X2 (Normalised)", fontsize=12, labelpad=12)
    ax.set_zlabel("y (Normalised)", fontsize=12, labelpad=12)
    ax.scatter(X[:, 0], X[:, 1], y, c=y, cmap='coolwarm', edgecolor='black', linewidth=0.5)
    ax.plot_surface(Xpoly_grid[:,:,1], Xpoly_grid[:,:,2], y_pred ,cmap='coolwarm',
edgecolor='none', alpha=0.75)
    legend_elements = [Line2D([0], [0], marker='o', markeredgecolor='black', linewidth=0.5,
                              label='True', linestyle = 'None', markersize=10, alpha=0.75),
                       Line2D([0],[0], linewidth=2.0, label='Pred')]
    ax.legend(handles=legend_elements, bbox_to_anchor=(0.91,0.94))
plt.savefig('ridge_classifiers')
plt.show()

means = []
variances = []
n_splits = [2, 5, 10, 25, 50, 100]
for n_split in n_splits:
    mses = []
    kf = KFold(n_splits=n_split)
    for train, test in kf.split(X):
        model = Lasso(alpha=1/(2*1))
        Xpoly = PolynomialFeatures(q).fit_transform(X[train])
        model.fit(Xpoly, y[train])
        Xpoly = PolynomialFeatures(q).fit_transform(X[test])
        ypred = model.predict(Xpoly)
        mse = (mean_squared_error(y[test], ypred))
        mses.append(mse)
    mean = np.mean(mses)
    means.append(mean)
    variance = np.var(mses)
    variances.append(variance)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Cross Validation of KFolds', fontsize=14)
ax.set_xlabel('Number of Folds', fontsize=12)
ax.set_ylabel('Mean (Normalised)', fontsize=12)
ax.errorbar(n_splits, means, yerr=np.sqrt(variances), linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(n_splits, means, marker='o', linewidth=1.0)
plt.savefig('cross_val_of_lasso_kfolds')
plt.show()

means = []
variances = []
```

```
penalties = [0.1, 0.5, 1, 5, 10, 50, 100]
for penalty in penalties:
    mses = []
    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model = Lasso(alpha=1/(2*penalty))
        Xpoly = PolynomialFeatures(q).fit_transform(X[train])
        model.fit(Xpoly, y[train])
        Xpoly = PolynomialFeatures(q).fit_transform(X[test])
        ypred = model.predict(Xpoly)
        mse = (mean_squared_error(y[test], ypred))
        mses.append(mse)
    mean = np.mean(mses)
    means.append(mean)
    variance = np.var(mses)
    variances.append(variance)

fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Cross Validation of C', fontsize=14)
ax.set_xlabel('C', fontsize=12)
ax.set_ylabel('Mean (Normalised)', fontsize=12)
ax.errorbar(penalties, means, yerr=np.sqrt(variances), linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(penalties, means, marker='o')
plt.savefig('cross_val_of_lasso_penalty')
plt.show()

means = []
variances = []
for penalty in penalties:
    mses = []
    kf = KFold(n_splits=5)
    for train, test in kf.split(X):
        model = Ridge(alpha=1/(2*penalty))
        Xpoly = PolynomialFeatures(q).fit_transform(X[train])
        model.fit(Xpoly, y[train])
        Xpoly = PolynomialFeatures(q).fit_transform(X[test])
        ypred = model.predict(Xpoly)
        mse = (mean_squared_error(y[test], ypred))
        mses.append(mse)
    mean = np.mean(mses)
    means.append(mean)
    variance = np.var(mses)
    variances.append(variance)

fig = plt.figure(figsize=(7, 7))
```

```
ax = fig.add_subplot(1, 1, 1)
ax.set_title('Cross Validation of C', fontsize=14)
ax.set_xlabel('C', fontsize=12)
ax.set_ylabel('Mean (Normalised)', fontsize=12)
ax.errorbar(penalties, means, yerr=np.sqrt(variances), linewidth=2.0, capsize=5.0,
elinewidth=2.0, markeredgewidth=2.0)
ax.scatter(penalties, means, marker='o')
plt.savefig('cross_val_of_ridge_penalty')
plt.show()
```