Assignment 3

Student number: 20324412

Module: CS7CS4/CSU44061

Part i.

Part a. Data

The data was read in and the resultant data is of the form in table 1. It includes two features X_1, X_2 and a target variable y.

	X1	X2	y
0	-1.00	0.83	1
1	0.99	0.22	1
2	0.76	0.85	-1
3	0.78	0.59	1
4	0.02	0.91	-1

Table 1. Data including two features X_1, X_2 and a target variable y.

Data Visualisation.

A visualisation of the data is shown in figure 1. The data appears to lie on a curve. It is evident that the relationship between the features X and the dependent variable y is a non-linear relationship. This is good to bear in mind for future model fitting.

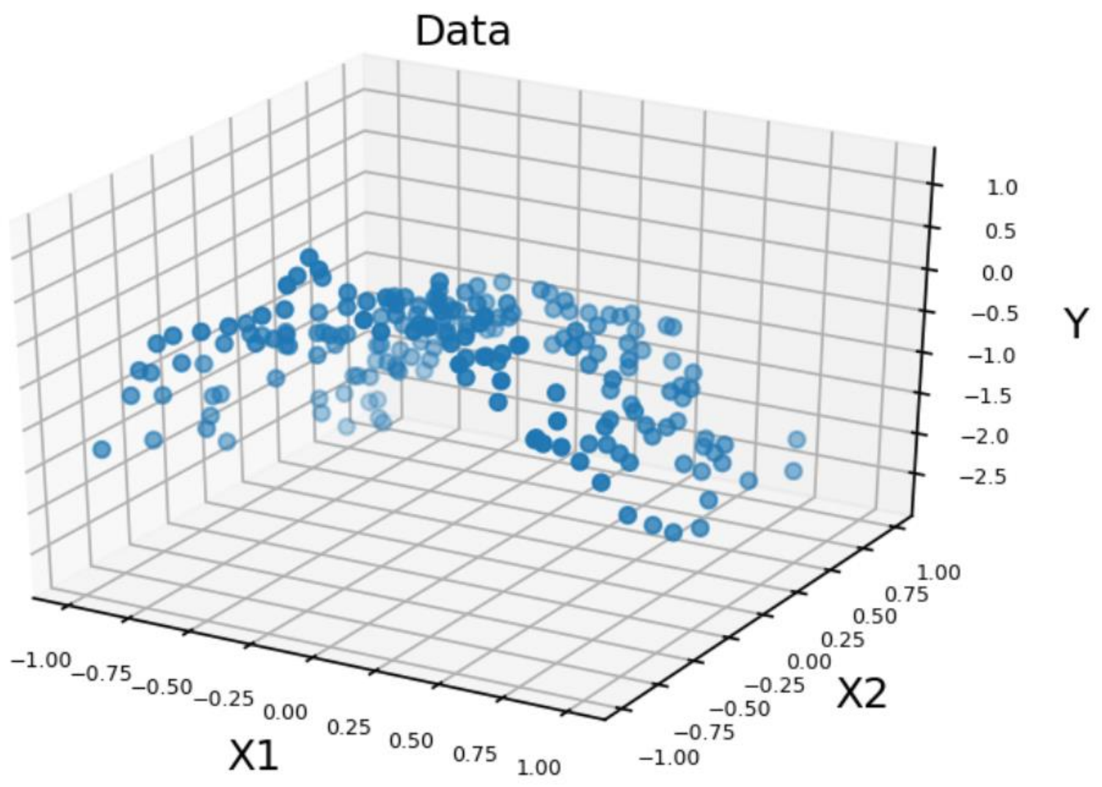


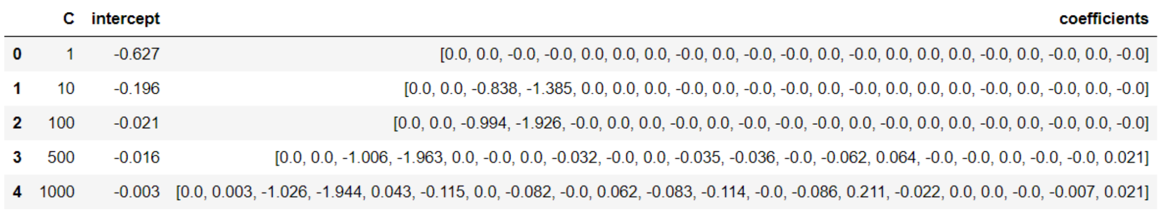
Figure 1. 3d Visualisation of the data

Part (ii) Lasso Regression

Lasso Regression models were then trained on the data. Lasso Regression is a type of regularized linear regression that includes an L1 penalty. Extra polynomial features equal to all combinations of powers of the two features up to power 5 were added to the model using the `PolynomialFeatures` function from sklearn. Lasso regression models were then trained on the data for a large range of values of C, specifically [1, 10, 100, 500, 1000]. The resultant models are shown below in table 1. The results were obtained using the function *regression_model_range_c* which returns a dataframe of the model parameters for a range of C values. This function can be used to implement either a Ridge or Lasso Regression depending on what 'model_type' is specified, in this case lasso.

```
def regression_model_range_c(X, y, degree_poly, c_test, model_type):

    '''Implement regression (lasso or ridge) for a range of values of the
penatly term C '''

    #Setup

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    df_results = []

    #Loop through c parameters and implement regression (lasso or ridge)

    for c_param in c_test:

        if model_type == 'model_lasso':

            model = Lasso(alpha=1/(2*c_param))

        elif model_type == 'model_ridge':

            model = Ridge(alpha=1/(2*c_param))

        #Fit data

        model.fit(Xpoly, y)

        #Dictionarry of values

        d = {

            'C' : c_param,

        'intercept': model.intercept_,

        'coefficients' :  np.around(model.coef_, decimals = 3),

        }

        df_results.append(d)

    df_svc_results = pd.DataFrame(df_results) #Results;

    return df_svc_results
```

This function returns the following dataframe.



	C	intercept	coefficients
0	1	-0.627	[0.0, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, -0.0, 0.0, -0.0, 0.0, -0.0]
1	10	-0.196	[0.0, 0.0, -0.838, -1.385, 0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, 0.0, -0.0, 0.0, -0.0, 0.0, -0.0]
2	100	-0.021	[0.0, 0.0, -0.994, -1.926, -0.0, 0.0, 0.0, -0.0, 0.0, -0.0, -0.0, -0.0, 0.0, -0.0, 0.0, 0.0, -0.0, 0.0, -0.0, 0.0, -0.0]
3	500	-0.016	[0.0, 0.0, -1.006, -1.963, 0.0, -0.0, 0.0, -0.032, -0.0, 0.0, -0.035, -0.036, -0.0, -0.062, 0.064, -0.0, -0.0, 0.0, -0.0, -0.0, 0.021]
4	1000	-0.003	[0.0, 0.003, -1.026, -1.944, 0.043, -0.115, 0.0, -0.082, -0.0, 0.062, -0.083, -0.114, -0.0, -0.086, 0.211, -0.022, 0.0, 0.0, -0.0, -0.007, 0.021]

Table 1. Lasso models of polynomial degree 5 (q = 5) for varying penalty term C.

C	**Intercept**	**Lasso Model**	**Lasso model parameter values**
1	-0.626	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. - 0. 0. 0. 0. 0. 0 0. 0. 0.
10	-0.626	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0. 0. -0.838 -1.385 0. 0. 0. 0. 0. 0. 0. 0. -0. 0. . 0. 0 0. 0. 0.
100	-0.021	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0. 0. -0.994 -1.926 0. 0. 0. 0. 0. 0. 0. 0. -0. 0. . 0. 0 0. 0. 0.
500	-0.155	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0. 0. -1.006 -1.963 0. 0. 0. -0.032 0. 0. -0.352 - 0.0355 0. -0.062 0.0641 0. 0. 0 0. 0. 0.0208
1000	-0.003	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0. 0.003 -1.026 -1.944 0.043 -0.115 0 -0.082 -0 0.062 - 0.083 -0.114 0 -0.086 0.211 - 0.022 0 0 0 -0.007 -0.021

Lasso Regression – discuss results

Lasso Regression is a type of regularized linear regression that includes an L1 penalty as shown in the cost function given by equation 1. The L1 penalty has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task. The hyper parameter C allows the influence of the penalty term to be controlled as shown by equation I.

$$J(\vartheta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 + \frac{1}{C} \sum_{j=1}^{n} |\theta_j| \tag{1}$$

It's value has a significant effect on the model parameters. Decreasing C gives the penalty more importance within the cost function. Thus for C = 1 the lasso model has shrunk all parameter values to zero as shown in table 1. The model is simply a constant value equal to the intercept (-0.626) for all the range of values of X1, X2. As C increases from 1 to 1000, less importance is placed on the penalty as it's relative amplitude to term one within the cost function is less.

Therefore more features are introduced into the model, i.e less coefficients have been shrunk to zero. For C = 10, 100, the model is the form of a polynomial function of degree 2 as follows;

Mode C = 10: $-0.021 - 0.838x_2 - 1.385x^2$

Model C = 100: $-0.021 - 0.838x_2 - 1.385x^2$

As C increases the lasso models become more complex and include higher order polynomial terms. For C = 500, 8 features are included within the model while for C =1000, 14 features have been added as shown in table 1.

Part c. Plot predictions

The predictions are plotted as below in Figures 1a-e. This was implemented using the function *plot_preds_range_c*. The predictions were made on test data which was generated over a grid of values. The original training data extends from [-1, 1] and so the predictions were made on an extended range from [-2, 2]. The resultant predictions are plotted as a surface against the data which is plotted as a scatter plot.

```
def plot_preds_range_c(X, y, Xtest, c_test, model_type, plot_colors):

    '''Plot predictions from lasso model for a range of C values '''

    #Get polynomial features

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    Xpoly_test = PolynomialFeatures(degree_poly).fit_transform(Xtest)

    #Loop through c parameters and implement lasso regression

    for c_param in c_test:

        if model_type == 'Lasso':

            model = Lasso(alpha=1/(2*c_param))

        elif model_type == 'Ridge':

            model = Ridge(alpha=1/(2*c_param))

        #Fit model

        model.fit(Xpoly, y)

        predictions = model.predict(Xpoly_test)

        #Plot

        fig = plt.figure()

        ax = fig.add_subplot(111, projection = '3d')

        #Plot predictions

        ax.plot_trisurf(Xtest[:,0], Xtest[:,1], predictions, color = 
plot_colors[0], alpha=.5)
```

```
        #Plot Data

        ax.scatter(X[:,0], X[:,1], y, color = plot_colors[1], label =
'Data')

        #Plot configuration

        colors = ['y', 'r']

        ax.set_xlabel('X1')

        ax.set_ylabel('X2')

        ax.set_zlabel('Y')

        ax.set_title('{}, C = {}'.format(model_type, c_param),
fontdict={'fontsize': 8.5})

        #Legend

        scatter1_proxy = matplotlib.lines.Line2D([0],[0], linestyle="none",
c=plot_colors[0], marker = 'o')

        scatter2_proxy = matplotlib.lines.Line2D([0],[0], linestyle="none",
c=plot_colors[1], marker = 'v')

        ax.legend([scatter1_proxy, scatter2_proxy], ['{}
Predictions'.format(model_type), 'Data'], numpoints = 1)

        ax.view_init(azim = 60)
```

This was implemented as follows

```
#Apply function

model_type = 'Ridge'; plot_colors = ['g', 'r']

plot_preds_range_c(X, y, Xtest, c_test[3:5], model_type, plot_colors)
```

Whereby Xtest is initiated across a grid of values ranging from x_min to x_max, using get_test_set;

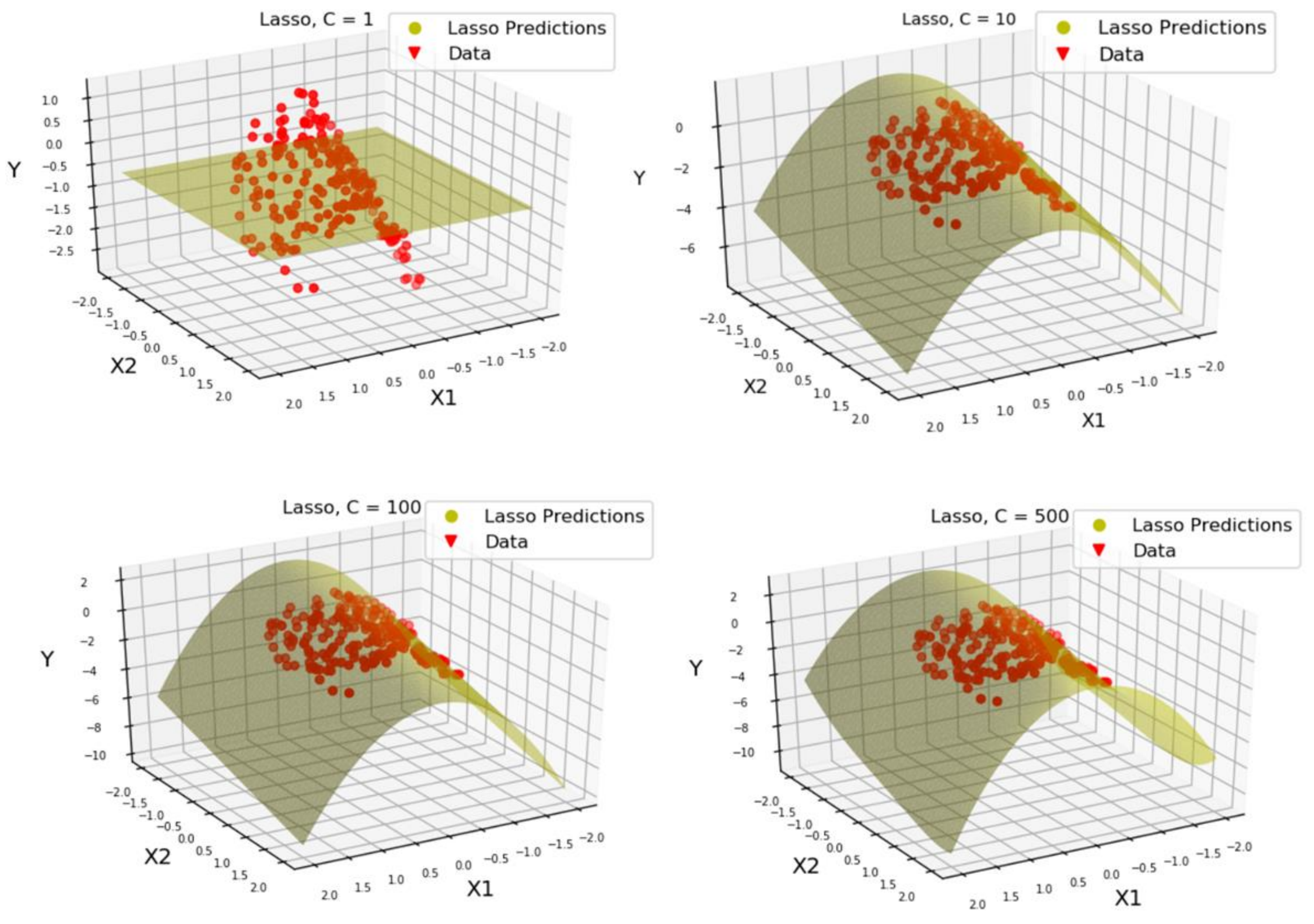
```
def get_test_set(x_min, x_max):

    '''Get test set across grid '''

    Xtest = []; grid = np.linspace(x_min, x_max)

    for i in grid:

        for j in grid:

            Xtest.append([i,j])

    Xtest = np.array(Xtest)

    return Xtest
```

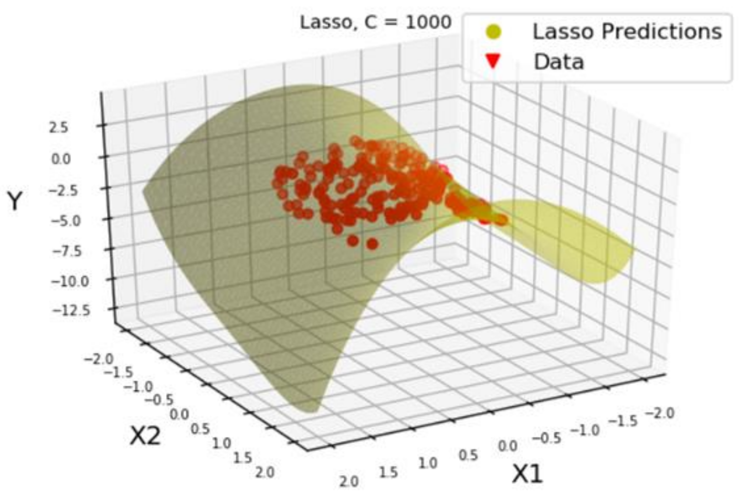
Lasso Regression - discuss plots

Considering the plots 1a-1e, it is evident that for increasing values of C the model becomes progressively more complex, arguably providing a better fit to the data. This is as expected given the parameter values obtained in part b.

For C = 1, lasso regression has shrunk all the parameter values to zero. As a result the model is simply a constant value, that of a plane, equal to the intercept (-0.626) for all the range of values of X1, X2 (Figure 1a). For larger values of C, in which the penalty term has less importance, lasso does not shrink all model coefficients and more features are included in the model. As a result the models are more complex and are able to capture the nonlinearity in the data. For C = 10, 100, the model is the form of a polynomial surface of degree 2 and so provides a better fit of the data. (Figure 1b,c). For C = 500, 1000, the model contains higher order polynomial features and so the polynomial surface is more complex (Figure 1d,e).

Figure 1a-e. Lasso Regression Predictions against Data for C = [1, 10, 100, 500,1000]





Part d. Under-fitting and Over-fitting

Under fitting refers to when a predictive model fails to capture the underlying trend of the data, usually occurring if the model is too simple. The model performs poorly on both the training data and generalizes poorly to other unseen data. Such a model often shows low variance but high bias.

Overfitting occurs when a predictive model captures the noise in the data and fits the training data too precisely. This often occurs if the model is too complex. As a result the model provides a very good fit on the training data but can perform poorly on unseen data given that it has captured the noise in the specific training set. Such a model often shows low bias but high variance.

The parameter C can be used to manage the trade-off between under fitting and overfitting of the data. It is evident from Figure 1a that for C = 1, the model is under fitting the data. The model is a constant value for all the range of values of X_1, X_2, that of a plane, equal to the intercept (-0.626). Therefore it fails to capture the non-linear relationship between the features X and y. On the other hand it appears that for C = 500, 1000, the model is overly complex and is overfitting the data. As shown in Table 1 the models contain 8 and 14 polynomial features respectively, up to degree 5, which is arguably too complex. The aim is always to create a simple a model as possible and so it could be argued that these models contain an unnecessary amount of features. The best trade off between under- and over-fitting the data is achieved for C = 10, 100. As shown in table 1 and displayed in figures 1(b)(c), the models are polynomial surfaces of degree 2. The models are able to capture the non-linear relationship between x and y, however they are not overly complex like those for C = 500, 1000. Thus a trade-off is achieved between under and over-fitting.

Part e – Ridge Regression

Ridge Regression models were then trained on the data. This was implemented in the same way as that for lasso regression. Extra polynomial features equal to all combinations of powers of the two features up to power 5 were added. Ridge regression models were then trained on this data for a large range of values of C, specifically [1, 10, 100, 500, 1000]. The resultant models are shown below in table 2. The results were again obtained using the function *regression_model_range_c* (as described above for lasso regression) in which the model type was specified as 'ridge'.

Table 2. Ridge regression models of polynomial degree 5 (q = 5) for varying penalty term C.

C	Intercept	Ridge Regression Model	Ridge model parameter values
1	-0.0569	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0.0, 0.013, -0.981, -1.538, 0.112, -0.09, 0.068, -0.198, -0.014, -0.023, -0.477, -0.218, -0.149, -0.117, 0.231, -0.157, 0.052, 0.08, 0.024, -0.084, 0.102
10	-0.0057	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0.0, -0.009, -1.031, -1.877, 0.131, -0.26, 0.233, -0.255, -0.149, 0.1, -0.149, -0.227, -0.081, -0.153, 0.419, -0.341, 0.218, 0.327, -0.143, -0.109, 0.103
100	-0.0184	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0.0, -0.012, -1.045, -1.953, 0.117, -0.297, 0.283, -0.286, -0.224, 0.155, -0.066, -0.204, -0.067, -0.153, 0.46, -0.401, 0.3, 0.434, -0.21, -0.095, 0.081
500	-0.0197	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0.0, -0.012, -1.047, -1.961, 0.115, -0.301, 0.288, -0.29, -0.234, 0.162, -0.057, -0.201, -0.065, -0.152, 0.464, -0.407, 0.311, 0.446, -0.217, -0.092, 0.078
1000	-0.0198	$1 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \cdots \theta_{19} x_1^5 + \theta_{20} x_1 x_2^4 + \theta_{21} x_2^5$	0.0, -0.012, -1.047, -1.962, 0.115, -0.301, 0.289, -0.291, -0.235, 0.163, -0.056, -0.201, -0.065, -0.152, 0.464, -0.408, 0.312, 0.448, -0.218, -0.092, 0.077

Ridge Regression – discuss results

Ridge Regression is a type of regularized linear regression that includes an L2 penalty as shown in the cost function given by equation 2. The hyper parameter C allows the influence of the penalty term to be controlled.

$$J(\vartheta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 + \frac{\theta^T\theta}{C} \qquad (2)$$

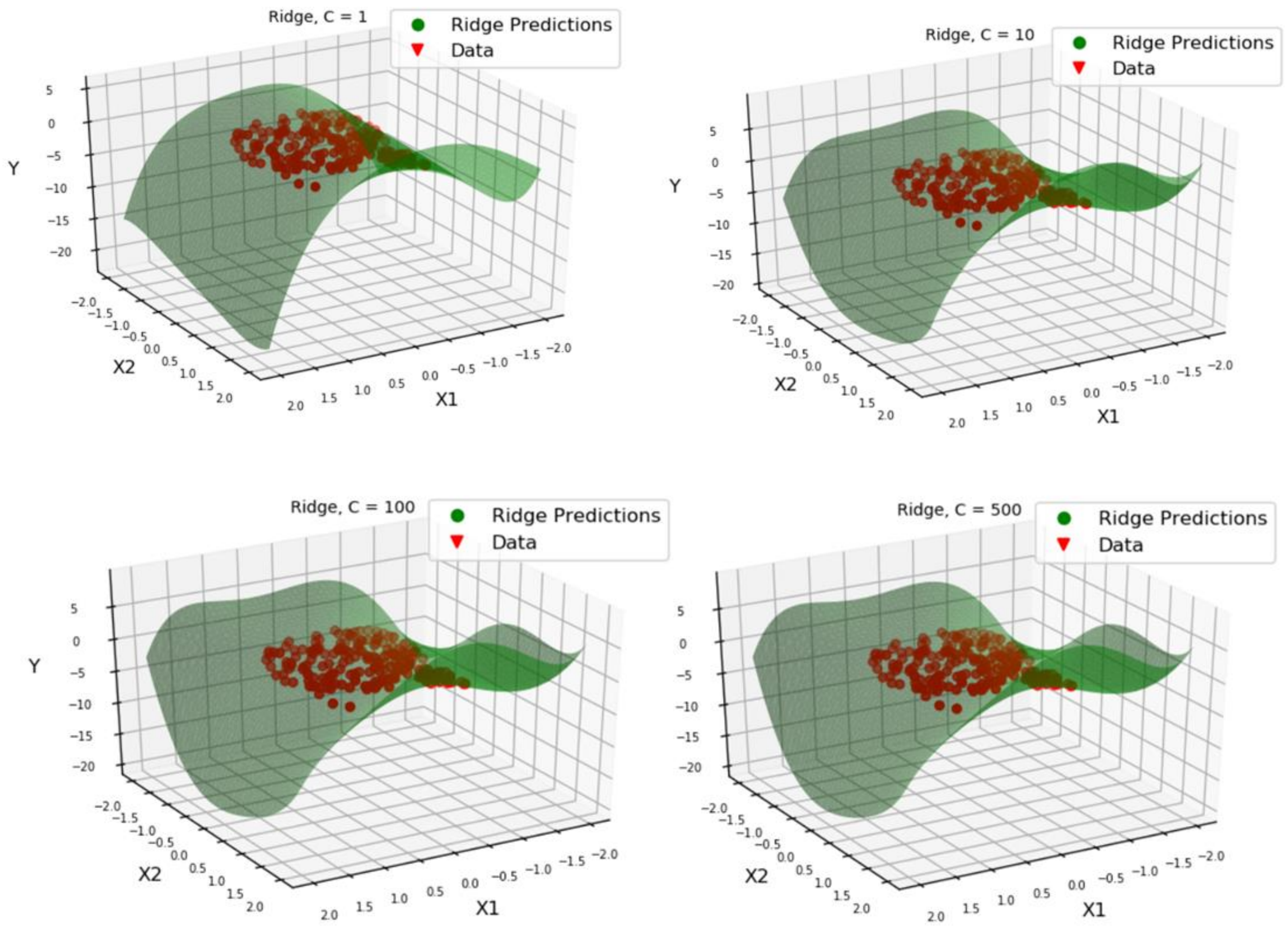
As C is increased, less importance is placed on the penalty term and the model reverts to that of a linear regression model. While decreasing C gives the penalty more importance within the cost function. However inspecting the model parameters, there is not an obvious difference between the models for C = 1 up to C = 1000, like that of Lasso. This is discussed in further detail below.

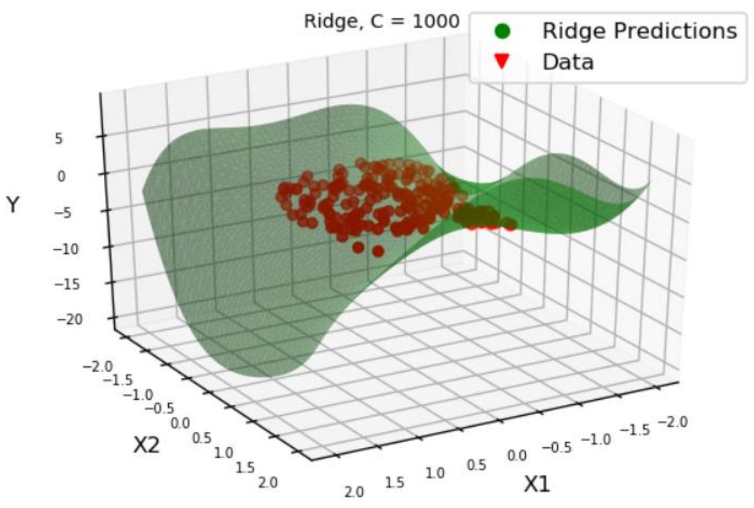
Plot predictions

The predictions are plotted as below in Figures 1a-e. This was implemented using the function *plot_preds_range_c*. The predictions were made on test data which was generated over a grid of values. The original training data extends from [-1, 1] and so the predictions were made on an extended range from [-2, 2]. The resultant predictions are plotted as a surface against the data which is plotted as a scatter plot.

Considering plots 2a-e, all models are polynomial surfaces of a certain degree, which is in contrast to Lasso. However as C increases C the model becomes progressively more complex. For example comparing C = 1 with C = 1000, the later function is significantly less smooth.

<u>Figure 2a-e. Ridge Regression Predictions against Data</u>





Comparison with Lasso

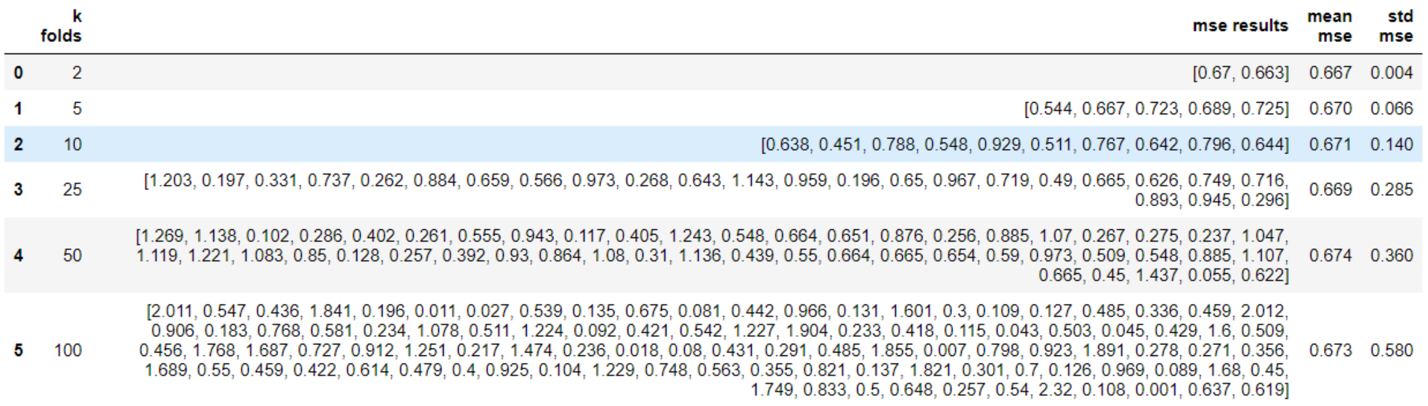
There is a significant difference between the model results. The main difference is that Lasso regression which includes L1 regularisation shrinks unimportant features to zero as seen in table 1. For small values of C, for example C = 1, where significant importance is placed on the penalty term, all model coefficients are shrunk to zero. This is contrast to Ridge Regression, where even for C = 1, the model still contains multiple polynomial features. Lasso Regression thus inherently implements feature selection whereby non important features are shrunk to zero and so the final model will contain less features. This is a significant advantage as it reduces the complexity of the final model and so overfitting is less likely to occur.

Part 2 – Cross Validation

To train and test the lasso model k-fold cross validation was implemented. It was implemented for k equal to [2, 5, 10, 25, 50, 100], while keeping C fixed in the lasso regression model. This was implemented using the function *get_df_kfold_cv* as below.

```
def get_df_kfold_cv(X, y, degree_poly, folds, C, model_type):

    '''Implement k fold cross validation for testing

    regression model (lasso or ridge) and return dataframe of results'''

    #Param setup

    mean_error=[]; std_error=[]; df_results = []

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    #Model

    if model_type == 'Lasso':

        model = Lasso(alpha=1/(2*C))

    elif model_type == 'Ridge':

        model = Ridge(alpha=1/(2*C))

    #Loop through each k fold

    for k in folds:

        mse_temp = []

        kf = KFold(n_splits = k)

        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])

            #Predict on test set

            ypred = model.predict(Xpoly[test])

            mse = mean_squared_error(y[test],ypred)

            mse_temp.append(mse)

        #Dictionarry of values - mean & variance

        d = {

            'k folds' : k,

        'mse results': np.around(mse_temp, decimals = 3),

        'mean mse' :  np.around(np.array(mse_temp).mean(), decimals = 3),

        'std mse' : np.around(np.array(mse_temp).std(), decimals = 3)
        }

        df_results.append(d)

    df_kf_results = pd.DataFrame(df_results) #Results

    return df_kf_results
```

The resultant dataframe is as below. The mean squared error (MSE) of each prediction for each fold is given as well as the mean and standard deviation for a given fold.



	k folds	mse results	mean mse	std mse
0	2	[0.67, 0.663]	0.667	0.004
1	5	[0.544, 0.667, 0.723, 0.689, 0.725]	0.670	0.066
2	10	[0.638, 0.451, 0.788, 0.548, 0.929, 0.511, 0.767, 0.642, 0.796, 0.644]	0.671	0.140
3	25	[1.203, 0.197, 0.331, 0.737, 0.262, 0.884, 0.659, 0.566, 0.973, 0.268, 0.643, 1.143, 0.959, 0.196, 0.65, 0.967, 0.719, 0.49, 0.665, 0.626, 0.749, 0.716, 0.893, 0.945, 0.296]	0.669	0.285
4	50	[1.269, 1.138, 0.102, 0.286, 0.402, 0.261, 0.555, 0.943, 0.117, 0.405, 1.243, 0.548, 0.664, 0.651, 0.876, 0.256, 0.885, 1.07, 0.267, 0.275, 0.237, 1.047, 1.119, 1.221, 1.083, 0.85, 0.128, 0.257, 0.392, 0.93, 0.864, 1.08, 0.31, 1.136, 0.439, 0.55, 0.664, 0.665, 0.654, 0.59, 0.973, 0.509, 0.548, 0.885, 1.107, 0.665, 0.45, 1.437, 0.055, 0.622]	0.674	0.360
5	100	[2.011, 0.547, 0.436, 1.841, 0.196, 0.011, 0.027, 0.539, 0.135, 0.675, 0.081, 0.442, 0.966, 0.131, 1.601, 0.3, 0.109, 0.127, 0.485, 0.336, 0.459, 2.012, 0.906, 0.183, 0.768, 0.581, 0.234, 1.078, 0.511, 1.224, 0.092, 0.421, 0.542, 1.227, 1.904, 0.233, 0.418, 0.115, 0.043, 0.503, 0.045, 0.429, 1.6, 0.509, 0.456, 1.768, 1.687, 0.727, 0.912, 1.251, 0.217, 1.474, 0.236, 0.018, 0.08, 0.431, 0.291, 0.485, 1.855, 0.007, 0.798, 0.923, 1.891, 0.278, 0.271, 0.356, 1.689, 0.55, 0.459, 0.422, 0.614, 0.479, 0.4, 0.925, 0.104, 1.229, 0.748, 0.563, 0.355, 0.821, 0.137, 1.821, 0.301, 0.7, 0.126, 0.969, 0.089, 1.68, 0.45, 1.749, 0.833, 0.5, 0.648, 0.257, 0.54, 2.32, 0.108, 0.001, 0.637, 0.619]	0.673	0.580

Table 3. Cross validation results.

The mean and variance values vs the number of folds used was plotted. The model was tested on both the train set and the test set to get an idea of model performance for increasing k. This was implemented using the following function;

```
def plot_kf_cv(df_kf):

    'Plot mean and std of kfold cross validation model results'

    #Plot

    plt.errorbar(np.array(df_kf.iloc[:,0]), np.array(df_kf.iloc[:,5]),
yerr=df_kf.iloc[:,6], color = 'orange', alpha = 0.8) #,linewidth = 2)

    plt.errorbar(np.array(df_kf.iloc[:,0]), np.array(df_kf.iloc[:,2]),
yerr=df_kf.iloc[:,3], color = 'green', alpha = 0.9) #, linewidth = 2)

    plt.xlabel('k fold')

    plt.ylabel('Mean square error')

    plt.title('K fold cross validation - lasso regression')

    plt.legend(['Test set', 'Training set'])

    plt.show()
```

Figure 3 (a-b). ***K fold cross validation results – k vs*** **MSE of lasso model**

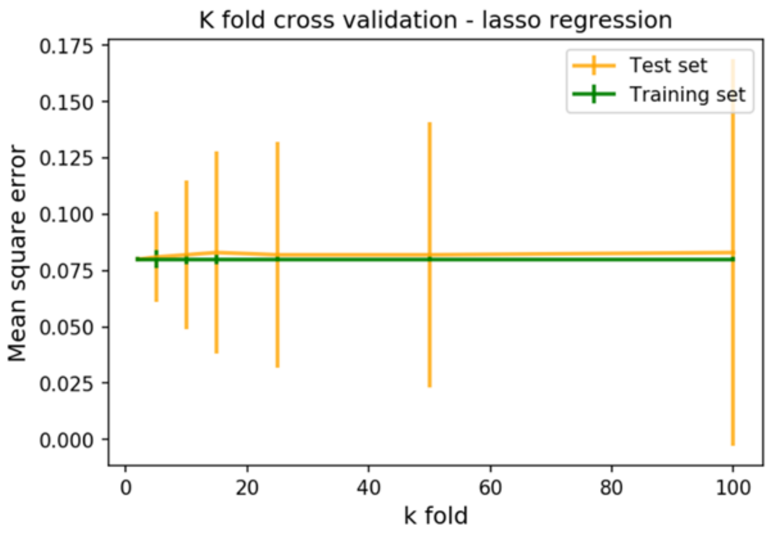


Figure 3a. C = [1, 5, 10, 50, 100, 500].

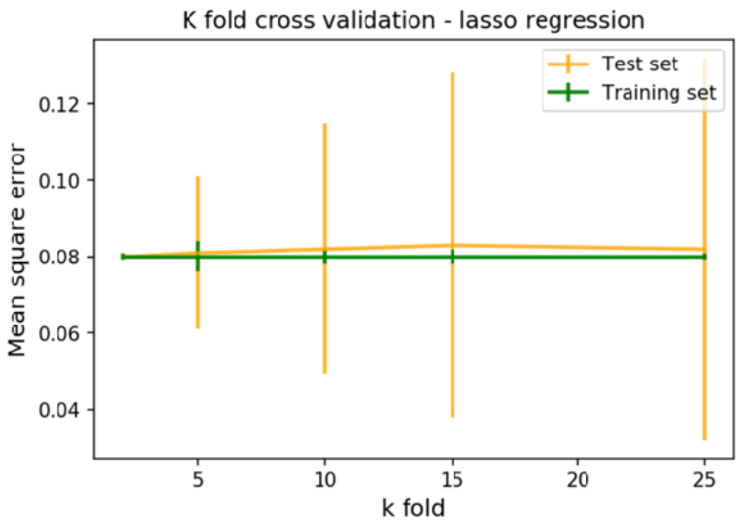


Figure 3b. C = [1, 5, 10, 20, 50]

Cross validation results - discuss

To get an idea of model performance, the mean and std of the MSE on the test and train set are shown in Figure 3. In k fold cross validation there is usually a trade-off in how the data is split. If more data is used for training then it expected that the model would be better trained, however in this case there is less data to test on. In Figure 3b it can be seen that the mean squared error (MSE) of the model increases slightly as k is increased from 2 to 15, from which point it reaches a somewhat steady level (See Figure 3b). Furthermore the standard deviation of the MSE greatly increases as the number of folds k is increased (Figure 3a). This implies that the model is performing well on some folds and poorly on other folds. This is also evident from the raw results shown in table 3, that there is high variability in model performance (mse). This makes sense in that for large k, for example k = 100, each held out fold is relatively small and so could prove difficult for the model to consistently predict well on such a small subsample. Larger values of k also necessitate additional computation and so are less efficient options. Based on these factors, a smaller value of k seems to result in the best performance, that of 2-fold or 5-fold cross validation. In the case of the former the ratio of the training to test data is 50:50 while in the case of the latter, the ratio is 80:20. Despite the excellent results for 2-fold, a training set of 50% seems relatively small so overall a 5-fold cross validation would be recommended for the fold split size.

Part b Cross Validation – Choice of C for Lasso Regression

The optimal value of the penalty term C in lasso regression was determined using 5-fold cross validation. This was implemented using the function *plot_kf_cv_c_vs_mse* as below. For each value of C, 5 fold cross validation was implemented to determine model performance, specifically the average and std of the MSE. A range of values of C was implemented. Firstly the range C = [1, 5, 10, 50, 100, 500] was tested as shown in Figure

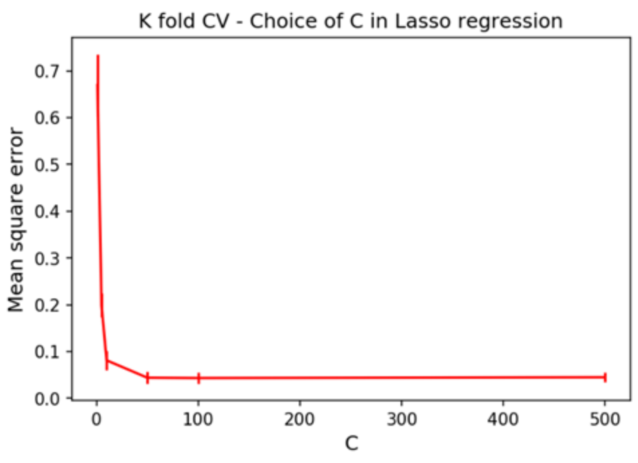
4a and subsequently C = [1, 5, 10, 20, 50] to determine exactly what value of C the greatest reduction in the MSE occurs.

```
def plot_kf_cv_c_vs_mse(X, y, degree_poly, k, c_test, model_type, 
plot_color):

    '''Implement k fold cross validation for testing

    regression model (lasso or ridge) and plot results'''

    #Param setup

    kf = KFold(n_splits = k)

    mean_error=[]; std_error=[]; df_results = []

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    #Loop through each k fold

    for c in c_test:

        mse_temp = []

        #Model

        if model_type == 'Lasso':

            model = Lasso(alpha=1/(2*c))

        elif model_type == 'Ridge':

            model = Ridge(alpha=1/(2*c))

        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])

            ypred = model.predict(Xpoly[test])

            mse = mean_squared_error(y[test],ypred)

            mse_temp.append(mse)

        #Get mean & variance

        mean_error.append(np.array(mse_temp).mean())

        std_error.append(np.array(mse_temp).std())

    #Plot

    plt.errorbar(c_test, mean_error, yerr=std_error, color = plot_color)

    plt.xlabel('C')

    plt.ylabel('Mean square error')
    plt.title('K fold CV - Choice of C in {}
regression'.format(model_type))

    plt.show()
```

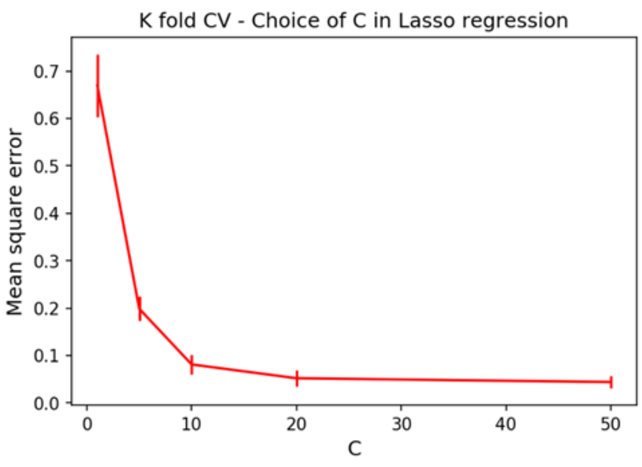
This function was implemented as follows;

```
#Apply

k = 5

model_type = 'Lasso'

c_test = [1, 5, 10, 50, 100, 500]

plot_color = 'red'

plot_kf_cv_c_vs_mse(X, y, degree_poly, k, c_test, model_type, plot_color)
```

Figure 4 (a-b). C vs MSE obtained using 5 fold Cross Validation



4a. C = [1, 5, 10, 50, 100, 500].



4b._C = [1, 5, 10, 20, 50]

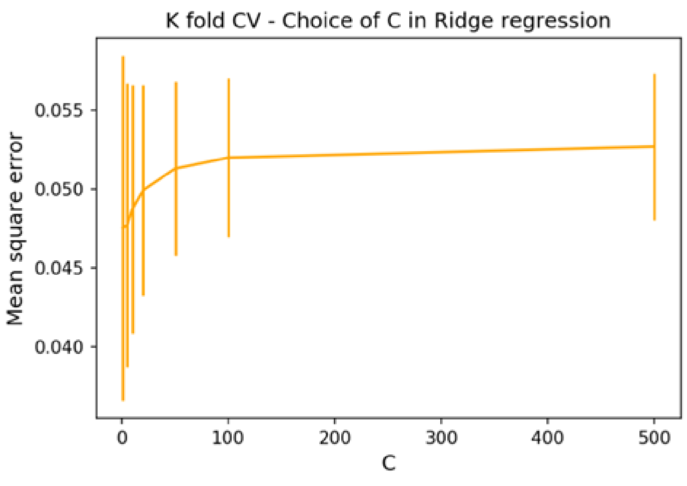
Part c - Choice of C for Lasso

The plot of C vs the MSE, obtained using 5 fold cross validation, provides a useful way of choosing C. There is a significant difference in the MSE as C is increased. For small values of C, e.g 1, the MSE is very high, approximately 0.7. It then dramatically reduces in amplitude as C is increased. For larger values of C, greater then 50, there is negligible reduction in the MSE. Thus models of increased complexity are unnecessary past this point. The 'elbow' method was deemed appropriate for choosing the optimal value of C, i.e the point of inflection on the graph, at which the MSE has decreased sufficiently. From Figure 4b, this point appears to occur for a value of C equal to 10. In line with the earlier analysis, a value of C equal to 10 would appears to provide the best trade-off between under- and over and fitting. For values of C less then 10, the MSE is quite high indicating under-fitting, while for greater values of C the MSE is very small, and perhaps the models could be overfitting the data.

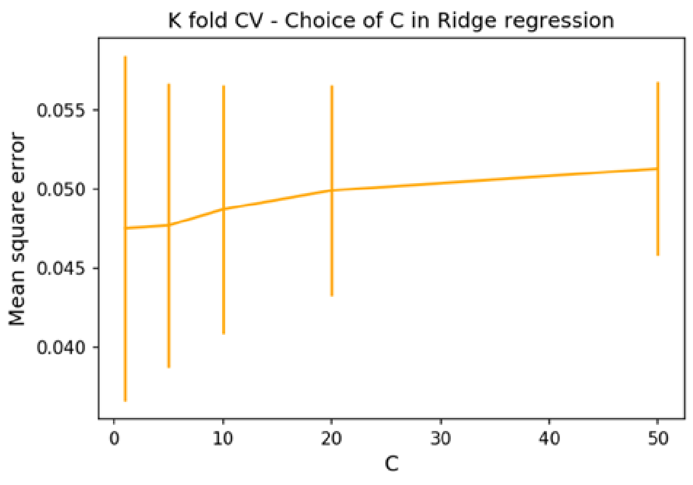
Part d Cross Validation – Choice of C for Ridge Regression

5-fold cross validation was used to determine the optimal value of the penalty term C in ridge regression. This was again implemented using the function *plot_kf_cv_c_vs_mse* specifying ridge as the model type. For each value of C, 5 fold cross validation was implemented to determine model performance, specifically the average and std of the MSE. A range of values of C was implemented. Firstly the range C = [1, 5, 10, 50, 100, 500] was tested as shown in Figure 5a and subsequently C = [1, 5, 10, 20, 50] to determine exactly what value of C the greatest reduction in the MSE occurs.

Figure 5 (a-b). C vs MSE obtained using 5 fold Cross Validation



5a. C = [1, 5, 10, 50, 100, 500].



5b._C = [1, 5, 10, 20, 50]

Choice of C

The optimal choice of C is less obvious in Ridge Regression as is the case in Lasso Regression. For all values of C the MSE is very low, however it does steadily increase as C is increased from 1 to 100 after which the increase is negligible as C approaches 500. In Figure 5b it can be seen that the MSE for C = 1 and C = 5 is at it's lowest. Furthermore the std of the latter is lower than that of the former and based on these two criteria, a value of 5 for the penalty term was deemed appropriate for Ridge Regression.

Appendix - Code

```
#Week 3 Assignment - Lasso, Ridge Regression & Cross Validation

#Imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import Lasso

from sklearn.linear_model import Ridge

from sklearn.model_selection import KFold

from mpl_toolkits.mplot3d import Axes3D

from sklearn.preprocessing import PolynomialFeatures

import matplotlib

from sklearn.metrics import mean_squared_error


#Plot params

plt.rcParams['figure.dpi'] = 150

matplotlib.rc('xtick', labelsize=10)

matplotlib.rc('ytick', labelsize=10)

pd.set_option('display.max_colwidth', -1)

MEDIUM_SIZE = 12

plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels


#*********************** Part a - Data  ********************

#***************************

#Part i Data + Visualisation

#Data

data = pd.read_csv('week3.txt',)

data.head()

data.reset_index(inplace=True)

data.columns = ['X1', 'X2', 'y']
```

```
#Extract Features
X1 = df.iloc[:,0]
X2 = df.iloc[:,1]
X = np.column_stack((X1,X2))
y= df.iloc[:,2]

#Plot data - 3d visualisation
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('Y')
ax.set_title('Data')
ax.scatter(X1, X2, y)

#************** Part b Lasso Regression Model *******************
#Get model results
def regression_model_range_c(X, y, degree_poly, c_test, model_type):
    '''Implement regression (lasso or ridge)
    for a range of values of the penatly term C '''
    #Setup
    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)
    df_results = []
    #Loop through c parameters and implement regression (lasso or ridge)
    for c_param in c_test:
        if model_type == 'Lasso':
            model = Lasso(alpha=1/(2*c_param))
        elif model_type == 'Ridge':
            model = Ridge(alpha=1/(2*c_param))
        #Fit data
        model.fit(Xpoly, y)
        #Dictionarry of values
        d = {
            'C' : c_param,
        'intercept': model.intercept_,
        'coefficients' :  np.around(model.coef_, decimals = 3),
        }
        df_results.append(d)
    df_svc_results = pd.DataFrame(df_results) #Results;
    return df_svc_results

#Apply to Lasso
degree_poly = 5
c_test = [1, 10, 100, 500, 1000]
model_type = 'Lasso'
regression_model_range_c(X, y, degree_poly, c_test, model_type)

#*********************************
#Part c Predictions
def plot_preds_range_c(X, y, Xtest, c_test, model_type, plot_colors):
    '''Plot predictions from lasso model for a range of C values '''
    #Get polynomial features
    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)
    Xpoly_test = PolynomialFeatures(degree_poly).fit_transform(Xtest)
    #Loop through c parameters and implement lasso regression
    for c_param in c_test:
        if model_type == 'Lasso':
            model = Lasso(alpha=1/(2*c_param))
        elif model_type == 'Ridge':
            model = Ridge(alpha=1/(2*c_param))

        #Fit model

        model.fit(Xpoly, y)

        predictions = model.predict(Xpoly_test)

        #Plot

        fig = plt.figure()

        ax = fig.add_subplot(111, projection = '3d')

        #Plot predictions

        ax.plot_trisurf(Xtest[:,0], Xtest[:,1], predictions, color = 
plot_colors[0], alpha=.5)

        #Plot Data

        ax.scatter(X[:,0], X[:,1], y, color = plot_colors[1], label = 
'Data')

        #Plot configuration

        colors = ['y', 'r']

        ax.set_xlabel('X1')

        ax.set_ylabel('X2')

        ax.set_zlabel('Y')

        ax.set_title('{}, C = {}'.format(model_type, c_param), 
fontdict={'fontsize': 8.5})

        #Legend

        scatter1_proxy = matplotlib.lines.Line2D([0],[0], linestyle="none", 
c=plot_colors[0], marker = 'o')

        scatter2_proxy = matplotlib.lines.Line2D([0],[0], linestyle="none", 
c=plot_colors[1], marker = 'v')

        ax.legend([scatter1_proxy, scatter2_proxy], ['{} 
Predictions'.format(model_type), 'Data'], numpoints = 1)

        ax.view_init(azim = 60)


#Apply function to Lasso

model_type = 'Lasso'

plot_colors = ['y', 'r']

c_test = [1, 10, 100, 500, 1000]
```

```
plot_preds_range_c(X, y, Xtest, c_test, model_type, plot_colors)


#******************* Part e Ridge Regression
*********************************

#Get Ridge egression results

degree_poly = 5

c_test = [1, 10, 100, 500, 1000]

model_type = 'Ridge'

regression_model_range_c(X, y, degree_poly, c_test, model_type)

#Plot Predictions

model_type = 'Lasso'

plot_colors = ['y', 'r']

plot_preds_range_c(X, y, Xtest, c_test, model_type, plot_colors)




#*********** Part 2 Cross Validation ******************************


def get_df_kfold_cv(X, y, degree_poly, folds, C, model_type):

    '''Implement k fold cross validation for testing

    regression model (lasso or ridge) and return dataframe of results'''


    #Param setup

    mean_error=[]; std_error=[]; df_results = []

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    #Model

    if model_type == 'Lasso':

        model = Lasso(alpha=1/(2*C))

    elif model_type == 'Ridge':

        model = Ridge(alpha=1/(2*C))


    #Loop through each k fold
    for k in folds:
        mse_temp = []
        kf = KFold(n_splits = k)

        for train, test in kf.split(Xpoly):
            model.fit(Xpoly[train], y[train])
            #Predict on test set
            ypred = model.predict(Xpoly[test])
            mse = mean_squared_error(y[test],ypred)
            mse_temp.append(mse)

        #Dictionarry of values - mean & variance
        d = {
            'k folds' : k,
        'mse results': np.around(mse_temp, decimals = 3),
        'mean mse' :  np.around(np.array(mse_temp).mean(), decimals = 3),
        'std mse' : np.around(np.array(mse_temp).std(), decimals = 3)
        }

        df_results.append(d)
    df_kf_results = pd.DataFrame(df_results) #Results

    return df_kf_results

#Apply to Lasso
c = 10
model_type = 'Lasso'
folds = [2, 5, 10, 15, 25, 50, 100]
df_kf_results = get_df_kfold_cv(X, y, degree_poly, folds, c, model_type)
df_kf_results
```

```
#Train and test results

def get_df_kfold_cvII(X, y, degree_poly, folds, C, model_type):

    '''Implement k fold cross validation for testing

    regression model (lasso or ridge) and return dataframe of results. Test
on train and test set'''


    #Param setup

    mean_error=[]; std_error=[]; df_results = []

    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    #Model

    if model_type == 'Lasso':

        model = Lasso(alpha=1/(2*C))

    elif model_type == 'Ridge':

        model = Ridge(alpha=1/(2*C))


    #Loop through each k fold

    for k in folds:

        mse_train_temp = []

        mse_test_temp = []

        kf = KFold(n_splits = k)


        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])


            #Predict on train set

            ypred = model.predict(Xpoly[train])

            mse = mean_squared_error(y[train],ypred)

            mse_train_temp.append(mse)


            #Predict on test set
            ypred = model.predict(Xpoly[test])

            mse = mean_squared_error(y[test],ypred)

            mse_test_temp.append(mse)


        #Dictionarry of values - mean & variance

        d = {

            'k folds' : k,

        'mse train results': np.around(mse_train_temp, decimals = 3),

        'mean mse train' :  np.around(np.array(mse_train_temp).mean(), 
decimals = 3), #np.around(model.coef_, decimals = 3),

        'std mse train' : np.around(np.array(mse_train_temp).std(), 
decimals = 3),

        'mse test': np.around(mse_test_temp, decimals = 3),

        'mean mse test' :  np.around(np.array(mse_test_temp).mean(), 
decimals = 3), #np.around(model.coef_, decimals = 3),

        'std mse test' : np.around(np.array(mse_test_temp).std(), decimals 
= 3)

        }


        df_results.append(d)


    df_kf_results = pd.DataFrame(df_results) #Results


    return df_kf_results


#Apply

model_type = 'Lasso'

folds = [2, 5, 10, 15, 25, 50, 100]

c = 10

df_kf2 = get_df_kfold_cv2(X, y, degree_poly, folds, c, model_type)


#*** Plot results ********
```

```
def plot_kf_cv(df_kf):
    'Plot mean and std of kfold cross validation model results'

    #Plot
    plt.errorbar(np.array(df_kf.iloc[:,0]), np.array(df_kf.iloc[:,5]),
yerr=df_kf.iloc[:,6], color = 'orange', alpha = 0.8) #,linewidth = 2)
    plt.errorbar(np.array(df_kf.iloc[:,0]), np.array(df_kf.iloc[:,2]),
yerr=df_kf.iloc[:,3], color = 'green', alpha = 0.9) #, linewidth = 2)
    plt.xlabel('k fold')
    plt.ylabel('Mean square error')
    plt.title('K fold cross validation - lasso regression')
    plt.legend(['Test set', 'Training set'])
    plt.show()

#Apply
plot_kf_cv(df_kf2)

#******** Part e C vs MSE using k fold Cross Val *******
def plot_kf_cv_c_vs_mse(X, y, degree_poly, k, c_test, model_type,
plot_color):
    '''Implement k fold cross validation for testing
    regression model (lasso or ridge) and plot results'''

    #Param setup
    kf = KFold(n_splits = k)
    mean_error=[]; std_error=[]; df_results = []
    Xpoly = PolynomialFeatures(degree_poly).fit_transform(X)

    #Loop through each k fold
    for c in c_test:
        mse_temp = []
        #Model
        if model_type == 'Lasso':

            model = Lasso(alpha=1/(2*c))

        elif model_type == 'Ridge':

            model = Ridge(alpha=1/(2*c))


        for train, test in kf.split(Xpoly):

            model.fit(Xpoly[train], y[train])

            ypred = model.predict(Xpoly[test])

            mse = mean_squared_error(y[test],ypred)

            mse_temp.append(mse)


        #Get mean & variance

        mean_error.append(np.array(mse_temp).mean())

        std_error.append(np.array(mse_temp).std())


    #Plot

    plt.errorbar(c_test, mean_error, yerr=std_error, color = plot_color)

    plt.xlabel('C')

    plt.ylabel('Mean square error')

    plt.title('K fold CV - Choice of C in {}
regression'.format(model_type))

    plt.show()
```

```
#Apply to Lasso

k = 5

model_type = 'Lasso'

c_test = [1, 5, 10, 50, 100, 500]

plot_color = 'red'

plot_kf_cv_c_vs_mse(X, y, degree_poly, k, c_test, model_type, plot_color)


#Apply to Ridge
```

```
k = 5

model_type = 'Ridge'

c_test = [1, 5, 10, 20, 50] #, 100, 500]

plot_color = 'orange'

plot_kf_cv_c_vs_mse(X, y, degree_poly, k, c_test, model_type, plot_color)
```