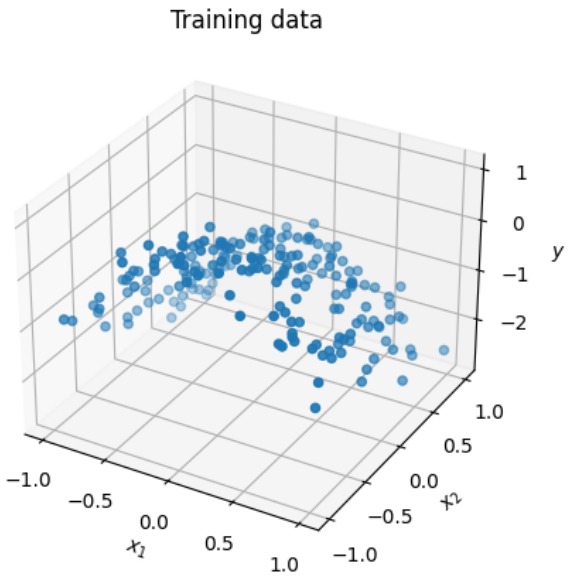
Assignment 3: Regularisation

Ciarán Donegan

24 October 2020

Dataset: # id:11–22–11

(i) (a)



The training data lies on a curved surface, not a plane. The saddle shape to the data is apparent when observing the training data with the interactive plot output.

The training data already occupies the range from -1 to $+1$, so it will not be necessary to normalise the data.

(b) Adding polynomial features up to degree 5 will result in 21 features. These 21 features along with the bias (θ_0) result in a hypothesis function ($h_\theta(x)$) of 22 learnable parameters in total, as summarised below.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + ... + \theta_{21} x_2^5$$

The cost function for linear regression with lasso regularisation is:

$$J(\theta) = \frac{1}{m} \sum^{m} (h_\theta(x - y)^2) + \frac{1}{C} \sum^{n} |\theta|$$

The value of C determines the strength of the regularisation applied i.e. the degree to which the parameters are penalised. Small C results in strong regularisation and large C results in weak regularisation.

Below are the results with different C values.

For $C = 1$

$$\theta_0 \approx -0.724$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

```
[ 0.  0.  -0.  -0.  0.  -0.  0.  -0.  0.  -0.  -0.  0.  -0.  0.  -0.  0.  -0.  0.
-0.  0.  -0.]
```

For $C = 10$

$$\theta_0 \approx -0.168$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

```
[ 0.  0.  -0.84885405 -1.47889254 0.  -0.  0.  -0.  0.  -0.  -0.  0.  -0.  0.  -0.
0.  -0.  0.  -0.  0.  -0.  ]
```

For $C = 1000$

$$\theta_0 \approx 0.038$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

```
[ 0.  -0.03423076 -1.05099462 -2.06052328 -0.  -0.06540851 0.05422588 -0.01586918
-0.  0.  0.12772312 0.  -0.  -0.  0.  0.03001919 0.  0.01486492 0.  -0.02599885 0.16446393]
```

From these learned parameters, we can see that when $C = 1$, and the strength of regularisation is large, all of the learned parameters are equal to zero. This is because the small value of C applies a huge penalty to the parameters and so they are set to zero during the gradient descent process.

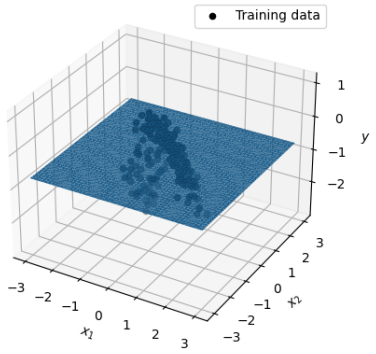
When $C = 1000$, the strength of the regularisation is much lower, and so the resulting parameter matrix is much less sparse (i.e fewer zero values).

When $C = 10$, the parameter matrix is very sparse, with only two non-zero values. The non-zero values correspond to θ_3 and θ_4 which are the weights on features x_1^2 and $x_1 x_2$ respectively. It makes sense that these features have been learned to be very important. Looking at the plot of the training data in (i)(a), it is clear that there is some sort of quadratic pattern in the x_1 feature. There also seems to be some interaction between the x_1 and x_2 features i.e. when x_1 is close to zero and x_2 is increasingly negative, the output y increases. So it makes sense that the parameter applied to the $x_1 x_2$ feature is non-zero.

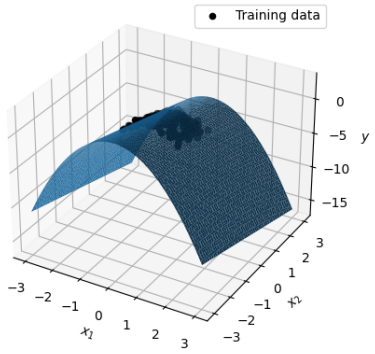
It is also important to note that Lasso Regression which penalises the magnitude of the features ($|\theta|$) results in sparsity in the learned parameters. The parameters are set to zero rather than just a small value.

(c) As shown in figure 1, when $C = 1$ the model is simply a plane at $y = -0.724$. This is obviously not capturing the true distribution of the data, an so too much regularisation has been applied.

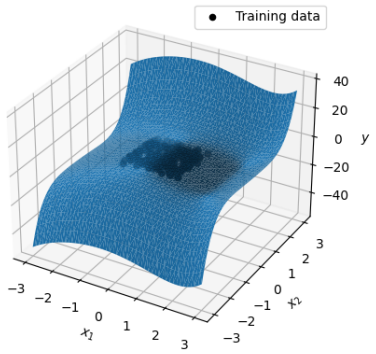
When $C = 1000$, the model is a very complex looking curve which includes features up the the 5th degree. From the training data, there is no way you can infer this sort of curve which rises and falls



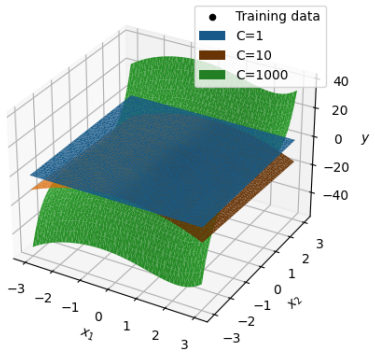
(a) $C = 1$



(b) $C = 100$



(c) $C = 1000$



(d) All models overlayed

Figure 1: Lasso regression models with varying degrees of regularisation.

sharply as x_2 moves further from zero. This model is too complex, and not enough regularisation has been applied.

The $C = 100$ model seems to be the most appropriate. It captures the quadratic nature in x_1 and it also gives increasing y values as x_2 becomes more negative.

(d) Under-fitting occurs when the model is so simple that if fails to capture the behaviour of the data. This is also refereed to as high bias.

Over-fitting, on the other hand, is when the model is too complex that it starts to fit the noise in the training data instead of just the general trend. This is also called high variance.

Setting C to a small number leads to under-fitting. This is apparent from the fact that all the learned parameter have been set to zero. i.e. the learned model is simply a straight plane. Also note how it fails to generalise to the unseen test data.

Setting C too large leads to over-fitting. The resulting model is far too complex and captures the noise in the data. Note how on the unseen test data it perform very poorly because the model predicts extreme values when it uses higher order features.

It would seem that when $C = 10$, the best trade-off between over-fitting and under-fitting is made.

(e) Ridge regression.

Below are the results with different C values.

For $C = 1$

$$\theta_0 = -0.010723189538216493$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

```
 [ 0.00000000e+00 -4.12590819e-02 -9.97700698e-01 -1.66607963e+00 -6.07333067e-04
-8.64498996e-02 5.21110647e-02 -1.73657560e-01 -2.50398540e-02 -4.58702441e-02 -2.41710217e-01
9.00150839e-03 -1.07340628e-01 -5.99336082e-04 6.52912018e-02 1.48772697e-02 8.77975138e-02
1.28897236e-01 1.03743313e-01 -8.11547688e-02 1.43199584e-01]
```

For $C = 10$

$$\theta_0 = 0.049807719988011256$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

```
 [ 0.00000000e+00 -6.61148113e-02 -1.03438300e+00 -2.11071764e+00 -3.85552463e-03
-1.15545710e-01 1.33760967e-01 -2.48359259e-01 5.14175962e-02 5.88972113e-02 1.83325625e-01
8.24616844e-03 8.20766256e-03 1.70778255e-03 5.30630999e-02 -3.86600020e-02 1.92636745e-01
1.56766640e-01 1.07059765e-01 -1.95163124e-01 8.88792972e-02]
```

For $C = 1000$

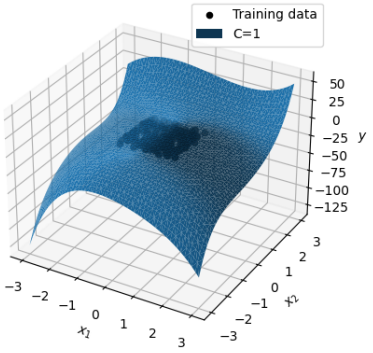
$$\theta_0 = 0.06140908328237982$$

$$[\theta_1, \theta_2, ..., \theta_{21}] =$$

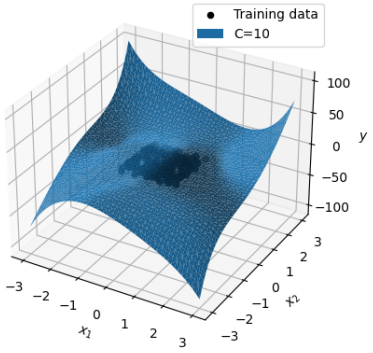
```
[ 0.  -0.08231236 -1.04348254 -2.19969515 -0.00544819 -0.12046659 0.17590654 -0.27436511
0.09763334 0.10328414 0.27022868 0.01001051 0.03308067 0.00256433 0.04918796 -0.06445922
0.22417328 0.14283294 0.11005574 -0.23726828 0.05396836]
```

A similar pattern emerges with the parameters becoming smaller as the value of C decreases. However, there is a major difference between lasso and ridge regression in that lasso regression will encourage parameters to be zero (i.e encourage sparsity) whereas ridge regression will only set the parameters to small values close to zero.

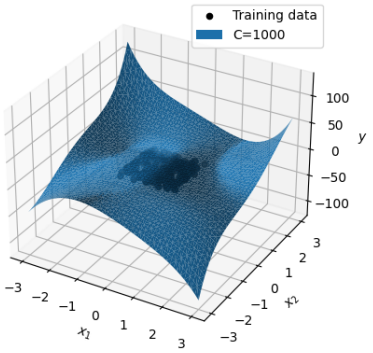
Figure 2 shows a similar pattern to the lasso models, where insreasing values of C results in over-fitting and more complex models. However, in this case, even the model with the most regularisation applied, $C = 1$, still looks too complex. The model has sharp curves that rise and fall sharply on unseen data. It seems that more regularisation is needed (i.e. a lower C value) to get a better bias-variance tradeoff with ridge regression.



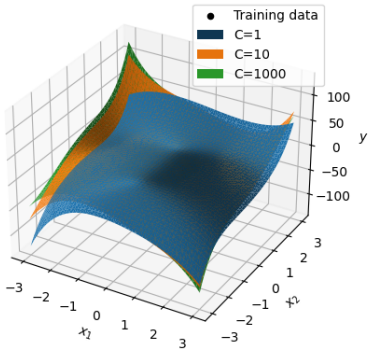
(a) $C = 1$



(b) $C = 100$

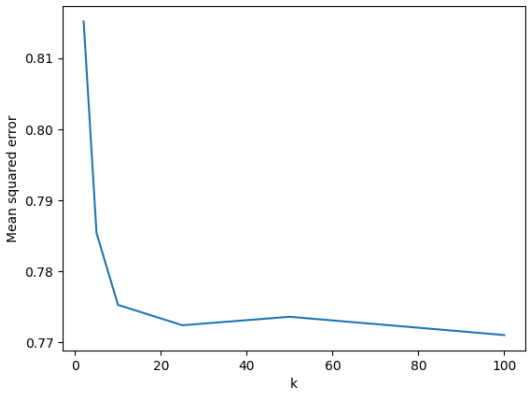


(c) $C = 1000$

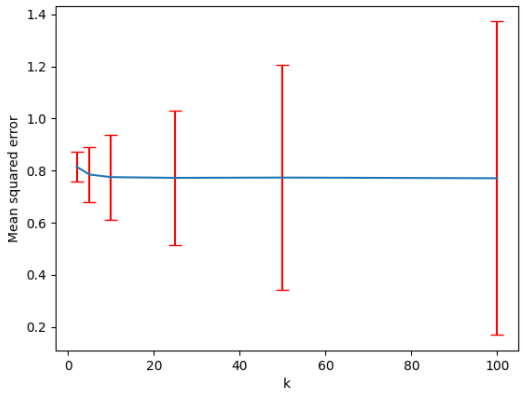


(d) All models overlayed

Figure 2: Ridge regression models with varying degrees of regularisation.

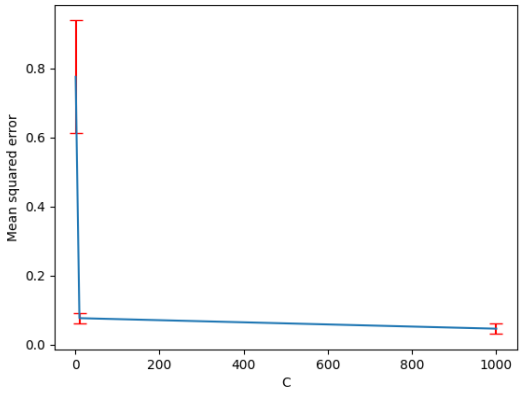


(a) Mean square error vs k

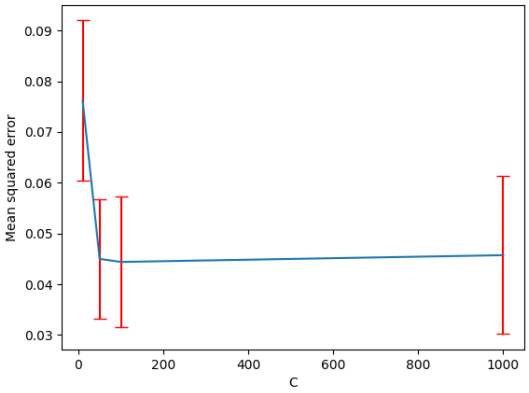


(b) Mean square error vs k (including error bars)

Figure 3: Analysis of the effect of k in k-folds cross validation.



(a) MSE vs C for $C = \{1, 10, 1000\}$



(b) MSE vs C for $C = \{10, 50, 100, 1000\}$

Figure 4: Analysis of the effect of C in lasso regression (using 10-fold cross validation).

(ii) (a) As figure 3 shows, the mean square error (MSE) is mostly decreasing as the value of k, the number of folds in k-folds, increases. However, when we show the standard deviation with error bars, an interesting narrative emerges. While, the MSE is lower for $k = 100$ than for $k = 5$, the standard deviation between the MSE in each fold is by much greater. This is because as k increases, the size of the test and training sets in each fold decreases. Therefore, you will get a much larger variance between folds dues to the smaller number of data points in each size.

In summary, larger k values results in fewer data points in each fold, thereby leading to high variance.

I would recommend using $k = 10$ because it has a low MSE and at the same time is not too large which would increases computation time i.e. $k = 100$ requires 100 separate models to be trained.

(b) As figure 4 shows, I first examined the MSE vs. C for $C = \{1, 10, 1000\}$, but since the MSE for $C = 1$ was so great, it skewed the graph and made interpreting it difficult. So I then changed to $C = \{10, 50, 100, 1000\}$ which allowed me to more clearly see the effect of different C values.

(c) For lasso regression, I would set $C = 100$. This value of C produces a low MSE when compared with other values.

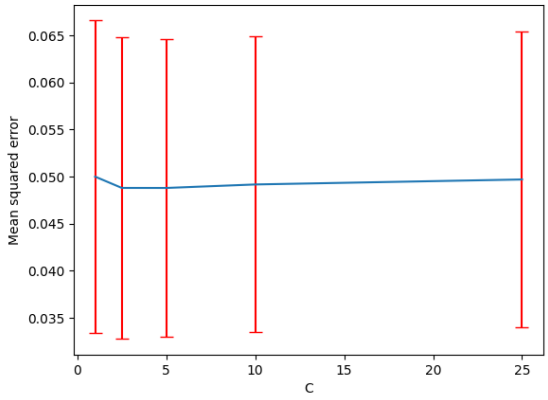


Figure 5: Effect of C in ridge regression for $C = \{1, 2.5, 5, 10, 25\}$. (Using 10-fold cross validation)

(d) From figure 5, I would select $C = 2.5$ for ridge regression. This value of C seems to minimise the MSE, and still have error bars comparable to that of other C values.

A Source code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error


def read_csv(filename):
    df = pd.read_csv(filename, comment="#", header=None)
    X1 = np.array(df.iloc[:,0])
    X2 = np.array(df.iloc[:,1])
    X = np.column_stack((X1, X2))
    y = np.array(df.iloc[:,2])
    return X, y


def plot_3d_scatter(x1, x2, y, title):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(x1, x2, y)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('$y$')
    plt.title(title)
    plt.savefig((title.lower()).replace(" ", "_") + ".png")


def train_lasso_model(X, y, C):
    alpha = 1 / (2*C)
    clf = linear_model.Lasso(alpha=alpha)
    clf.fit(X,y)
    # print("Coefficients\n", clf.coef_)
    # print("Intercept\n", clf.intercept_)
    return clf


def train_ridge_model(X, y, C):
    alpha = 1 / (2*C)
    clf = linear_model.Ridge(alpha=alpha)
    clf.fit(X,y)
    print("Coefficients\n", clf.coef_)
    print("Intercept\n", clf.intercept_)
    return clf


def get_test_features(min_val, max_val):
    X_test = []
    grid = np.linspace(min_val, max_val)
    for i in grid:
```

```
        for j in grid:
            X_test.append([i,j])
    return np.array(X_test)


def use_kfold(X, y, C, k, regression_type):
    valid_types = ["lasso", "ridge"]
    if regression_type not in valid_types:
        raise ValueError("Invalid reggression type")

    kf = KFold(n_splits=k)
    mean_error = []

    for train, test in kf.split(X):
        if regression_type == "lasso":
            model =  train_lasso_model(X[train], y[train], C)
        else:
            model = train_ridge_model(X[train], y[train], C)
        y_pred = model.predict(X[test])
        mean_error.append(mean_squared_error(y[test], y_pred))

    mse = np.array(mean_error).mean()
    std = np.array(mean_error).std()

    print("MSE = ", mse)
    print("STD = ", std)
    return mse, std


if __name__ == "__main__":

    X, y = read_csv("week3.csv")
    # plot training data
    plot_3d_scatter(X[:,0], X[:,1], y, title= "Training data")

    # add polynomial features up to degree 5
    poly = PolynomialFeatures(5)
    X_poly = poly.fit_transform(X)

    # train and test models with different regularisation (C) values
    # plot the predictions on a surface plot
    X_test = get_test_features(-3, 3)
    X_test = poly.fit_transform(X_test)

    # Plot the resilts of model predictions along with training data
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(X[:,0], X[:,1], y, c="k", label="Training data")
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_zlabel('$y$')
    c_list = [1, 10, 1000]
    for c in c_list:
        clf = train_ridge_model(X_poly, y, c)
```

```
        print(X_test.shape)
        y_pred = clf.predict(X_test)
        print(y_pred.shape)
        print(X_test[:,0].shape)
        trisurf = ax.plot_trisurf(X_test[:,1], X_test[:,2], y_pred, label="C="+str(c))
        trisurf._facecolors2d=trisurf._facecolors3d
        trisurf._edgecolors2d=trisurf._edgecolors3d
plt.legend()
plt.savefig("ridge_regression_models.png")


# Perfrom k-fold cross validation
k_values = [2, 5, 10, 25, 50, 100]
mse_list = []
std_list = []
for k in k_values:
    mse, std = use_kfold(X_poly, y, C=1, k=k, regression_type="lasso")
    mse_list.append(mse)
    std_list.append(std)

plt.figure()
plt.errorbar(k_values, mse_list, yerr=std_list, ecolor="r", capsize=5)
plt.xlabel("k")
plt.ylabel("Mean squared error")
plt.savefig("lasso_kfold_errorbars.png")


# Plot MSE vs C using kfolds
mse_list = []
std_list = []
c_list = [1, 2.5, 5, 10, 25]
for c in c_list:
    mse, std = use_kfold(X_poly, y, C=c, k=10, regression_type="ridge")
    mse_list.append(mse)
    std_list.append(std)

plt.figure()
plt.errorbar(c_list, mse_list, yerr=std_list, ecolor="r", capsize=5)
plt.xlabel("C")
plt.ylabel("Mean squared error")
plt.savefig("ridge_mse_vs_C.png")
```