# Question (i)

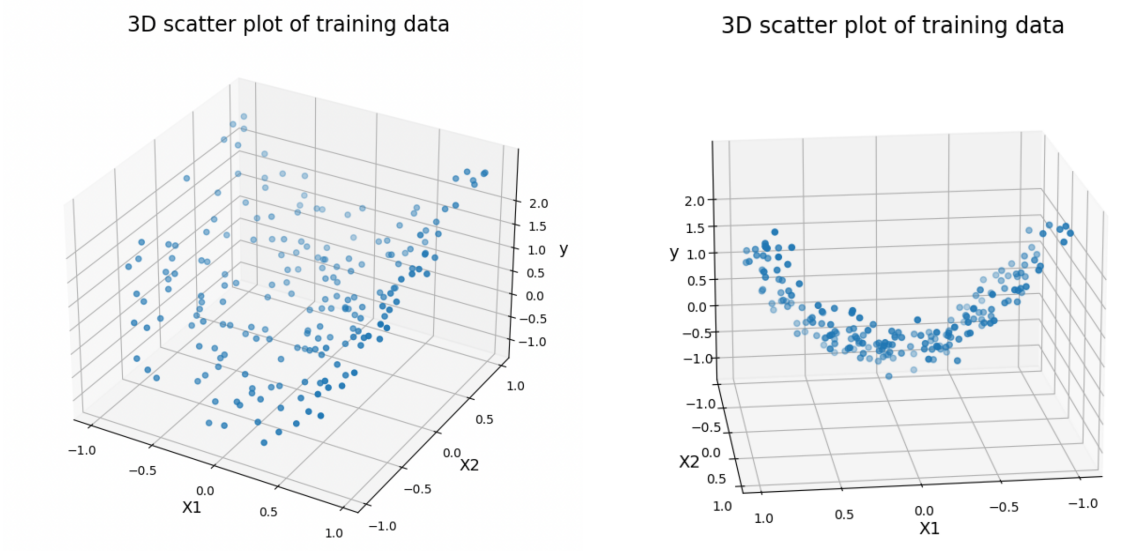
My downloaded dataset for this assignment is identified by # id:9-18-9

## Part (a)

I visualized the downloaded data on a 3D scatter plot with feature 1 on the x-axis, feature 2 on the y-axis, and the target on the z-axis. I read in the data using code from 1.1 in the appendix and used 1.2 in the appendix to plot the 3D scatter plot.

```
    X1, X2, X, y, y_train = read_data()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(X1.to_numpy(), X2.to_numpy(), y)
```

We can see from figure 1 (b) below that the training data lies on a curve, and it seems to look quadratic since it has a parabolic shape.




(a) Normal angle

(b) Angle showing quadratic behavior

Figure 1: A 3D scatter plot visualizing the training data with feature X1 on the x-axis, feature X2 on the y-axis, and the target y on the z-axis

## Part (b)

Using code from 1.3 in the appendix, I added extra polynomial features equal to all combinations of powers of the two features up to power 5 using the PolynomialFeatures function in sklearn. I then trained the Lasso regression models with these polynomial features for values of C such as 1, 10 and 1000. I also calculated the mean squared error (MSE) for each trained model, including a baseline model to compare with which always predicts the mean of the training set.

```
Xpoly = PolynomialFeatures(5)
    XpolyFittedAndTransformed = Xpoly.fit_transform(X)
    compareWithABaseline =
    DummyRegressor(strategy="mean").fit(XpolyFittedAndTransformed, y)
    print("MSE using dummy regressor =", mean_squared_error(y,
    compareWithABaseline.predict(XpolyFittedAndTransformed)))
```

```
for C in [1, 10, 1000]:
    model = Lasso(alpha=(1 / (2 * C))).fit(XpolyFittedAndTransformed,
    y)
    theta = np.append(model.intercept_, model.coef_)
```

We can get the feature names using the get_feature_names_out function:
$[1, X_1, X_2, (X_1)^2, (X_1) * (X_2), (X_2)^2, (X_1)^3, (X_1)^2 * (X_2), (X_1) * (X_2)^2, (X_2)^3, (X_1)^4,$
$(X_1)^3 * (X_2), (X_1)^2 * (X_2)^2, (X_1) * (X_2)^3, (X_2)^4, (X_1)^5, (X_1)^4 * (X_2), (X_1)^3 * (X_2)^2,$
$(X_1)^2 * (X_2)^3, (X_1) * (X_2)^4, (X_2)^5]$
The parameters of the trained models with different values of C are as follows (with what features they correspond to):

| | C=1 | C=10 | C=1000 |
|---|---|---|---|
| $\theta_0$ (intercept) | 0.6957975 | 0.1777648 | -0.03083267 |
| $\theta_1$ [corresponds to *bias* (i.e. 1)] | 0 | 0 | 0 |
| $\theta_2$ [corresponds to $X_1$] | 0 | 0 | 0.0312615 |
| $\theta_3$ [corresponds to $X_2$] | 0 | 0.82572033 | 0.94948294 |
| $\theta_4$ [corresponds to $(X_1)^2$] | 0 | 1.4739457 | 2.039377 |
| $\theta_5$ [corresponds to $(X_1) * (X_2)$] | 0 | 0 | -0 |
| $\theta_6$ [corresponds to $(X_2)^2$] | 0 | 0 | 0.06142604 |
| $\theta_7$ [corresponds to $(X_1)^3$] | 0 | 0 | 0.01912658 |
| $\theta_8$ [corresponds to $(X_1)^2 * (X_2)$] | 0 | 0 | 0 |
| $\theta_9$ [corresponds to $(X_1) * (X_2)^2$] | 0 | 0 | -0.04515518 |
| $\theta_{10}$ [corresponds to $(X_2)^3$] | 0 | 0 | 0.05385162 |
| $\theta_{11}$ [corresponds to $(X_1)^4$] | 0 | 0 | 0 |
| $\theta_{12}$ [corresponds to $(X_1)^3 * (X_2)$] | 0 | 0 | -0 |
| $\theta_{13}$ [corresponds to $(X_1)^2 * (X_2)^2$] | 0 | 0 | 0 |
| $\theta_{14}$ [corresponds to $(X_1) * (X_2)^3$] | 0 | 0 | 0.04774425 |
| $\theta_{15}$ [corresponds to $(X_2)^4$] | 0 | 0 | -0.01735093 |
| $\theta_{16}$ [corresponds to $(X_1)^5$] | 0 | 0 | 0 |
| $\theta_{17}$ [corresponds to $(X_1)^4 * (X_2)$] | 0 | 0 | 0.04684794 |
| $\theta_{18}$ [corresponds to $(X_1)^3 * (X_2)^2$] | 0 | 0 | -0 |
| $\theta_{19}$ [corresponds to $(X_1)^2 * (X_2)^3$] | 0 | 0 | -0.04626425 |
| $\theta_{20}$ [corresponds to $(X_1) * (X_2)^4$] | 0 | 0 | -0 |
| $\theta_{21}$ [corresponds to $(X_2)^5$] | 0 | 0 | 0.00137878 |
| $MSE$ (rounded to 6 decimal places) | 0.703895 | 0.076507 | 0.038444 |

$$MSE_{DummyRegressor} = 0.703895$$

Lasso regression uses a L1 penalty which encourages sparsity of the solution meaning that it tries to make $\theta$ have mainly 0 values. When C is big (i.e. C=1000), the L1 penalty is small meaning that we have very little 0 values as seen above. We can also see that since few values in $\theta$ are 0, the model will rely on many features such as $(X_1)^2$ which follows a quadratic shape that is appropriate for our training data that lies on a curve. However since the model depends on many features and fits very accurately with the training data, it could lead to overfitting meaning it could perform inaccurately for new data. This can be proven by the fact that $MSE_{C=1000}$ is very small (0.038444).

When C is small (i.e. C=10), the L1 penalty is bigger meaning there are more 0 values in $\theta$, with only $\theta_3$ and $\theta_4$ being non-zero values excluding the intercept $\theta_0$. When C is really

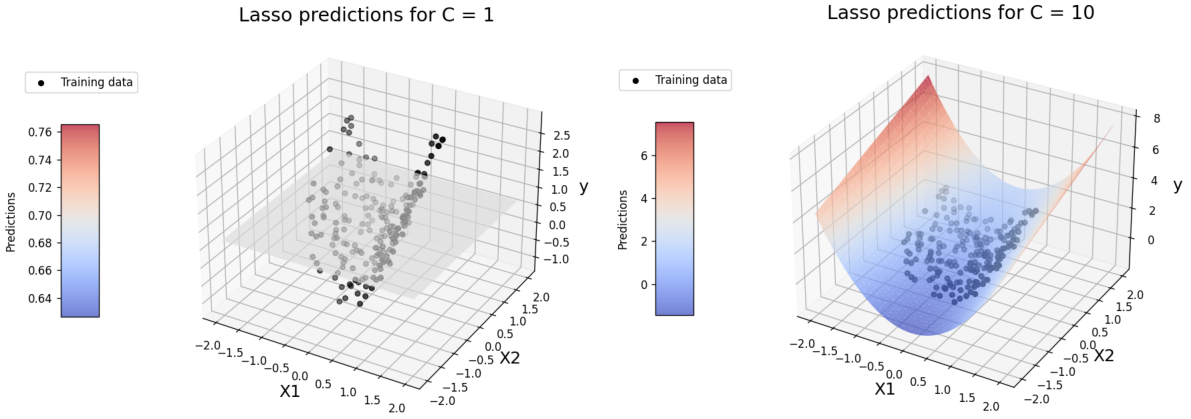
small (i.e. C=1), the L1 penalty is very big meaning that we have many 0 values in $\theta$ as seen above. Since the L1 penalty is big, all of the elements of $\theta$ are penalized heavily and this will not be accurate for our curved training data from (i)(a). This point is proven by the fact that $MSE_{C=1}$ (0.703895) $\approx MSE_{DummyRegressor}$ (0.703895).

## Part (c)

I used the models from (i)(b) and generated predictions for the target variable on a grid of feature values that extends beyond the range of values in the dataset. The feature values in the dataset range from -1 to 1, so I chose a range from -2 to 2 for the grid of predictions. I chose this so that the training data can still be clearly seen in the plot. I did this with code from 1.4 in the appendix, and plotted the predictions on a 3D data plot using the plot_trisurf function.

```
ax.scatter(X1.to_numpy(), X2.to_numpy(), y, color="black",
label="Training data")
plot_surface = ax.plot_trisurf(X1FromXTest, X2FromXTest,
model.predict(Xtest), cmap=cm.coolwarm, alpha=0.7)
cbar = fig.colorbar(plot_surface, shrink=0.5, aspect=5, location="left")
```

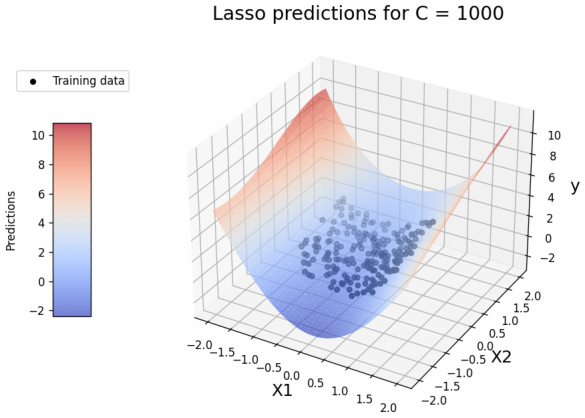
I plotted the 3d plots using the same values of C as in (i)(b) which were 1, 10 and 1000 as follows:




(a) Lasso predictions with C=1




(b) Lasso predictions with C=10




(c) Lasso predictions with C=1000

Figure 2: 3D plots with lasso predictions for different values of C with the training data

Referring to the plots above, we can see that when C is low (i.e. C=1), the prediction surface is fairly flat and too simple since we can see in figure 2 (a) that it shows signs of underfitting. When C is bigger (i.e. C=1000), the prediction surface becomes very complex and doesn't match our training data in the red parts of the curve along the edges. We can see in figure 2 (c) that this shows signs of overfitting. Our explanations from (i)(b) are solidified from these plots as we can see that C=10 is the best value that is appropriate for the shape of our training data that lies on a curve as it makes the prediction surface follow a curved quadratic shape.

## Part (d)

Underfitting is when our model has inaccurate predictions because the model is too simple since it depends on too few features. When we use C=1, we can see from the table in (i)(b) that the parameters are almost 0 except the intercept $\theta_0$. This leads to the flat prediction surface in figure 2 (a) from (i)(c). Since our training data has a curved quadratic shape, this flat prediction surface will not suffice and therefore shows underfitting.

Overfitting is when our model performs poorly against new data since it takes too many factors into account including minor variations (which can be called "noise"). When we use C=1000, we can see in the table (i)(b) that the parameters are almost all non-zero, and this leads to a complex prediction surface as seen in figure 2 (c). The shape of the prediction surface along the edges of the red part of the curve differs from the curved quadratic shape of the training data and therefore shows overfitting.

Hyperparameter C can be used to manage to trade off between underfitting and overfitting data since we can use C to make our model ignore particular features if we need them to by decreasing C to avoid overfitting, and we can also make our model take more features into account if we need them to by increasing C to avoid underfitting. Therefore we can change the value of C to suit our model and get the right fit for our model.

## Part (e)

I repeated (i)(b) and (i)(c) using the Ridge Regression model with code from 1.5 in the appendix. This time, I used different values for C such as 0.00001, 0.1 and 1. I chose these values so that I could see how changing C changes the parameters from being close to zero to all of them being non-zero.

```
model = Ridge(alpha=1 / (2 * C)).fit(XpolyFittedAndTransformed, y)
theta = np.append(model.intercept_, model.coef_)
```

The feature names are the same from (i)(b), however the parameters of the trained models are different with the different values of C:

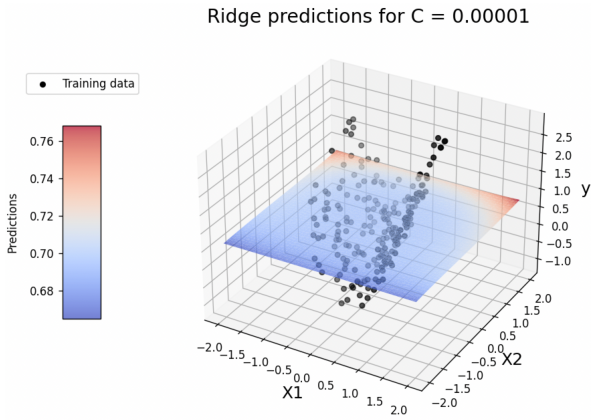
| | C=0.00001 | C=0.1 | C=1 |
|---|---|---|---|
| $\theta_0$ (intercept) | 0.6952409 | 0.15284132 | 0.02589047 |
| $\theta_1$ [corresponds to *bias* (i.e. 1)] | 0 | 0 | 0 |
| $\theta_2$ [corresponds to $X_1$] | 0.00009691 | 0.02987326 | 0.01762356 |
| $\theta_3$ [corresponds to $X_2$] | 0.0012057 | 0.70247459 | 0.86485395 |
| $\theta_4$ [corresponds to $(X_1)^2$] | 0.00070615 | 1.03625724 | 1.56452688 |
| $\theta_5$ [corresponds to $(X_1) * (X_2)$] | 0.00008466 | 0.00491339 | -0.03732236 |
| $\theta_6$ [corresponds to $(X_2)^2$] | 0.00013399 | 0.0075098 | 0.0931881 |
| $\theta_7$ [corresponds to $(X_1)^3$] | 0.00007315 | 0.02923256 | 0.05105396 |
| $\theta_8$ [corresponds to $(X_1)^2 * (X_2)$] | 0.00047145 | 0.11534977 | 0.12451327 |
| $\theta_9$ [corresponds to $(X_1) * (X_2)^2$] | 0.00004701 | -0.03239274 | -0.01288176 |
| $\theta_{10}$ [corresponds to $(X_2)^3$] | 0.00074751 | 0.23304689 | 0.20182663 |
| $\theta_{11}$ [corresponds to $(X_1)^4$] | 0.00060832 | 0.71758198 | 0.46681609 |
| $\theta_{12}$ [corresponds to $(X_1)^3 * (X_2)$] | 0.0000612 | 0.00770816 | 0.01383756 |
| $\theta_{13}$ [corresponds to $(X_1)^2 * (X_2)^2$] | 0.00028762 | 0.23423759 | 0.16011192 |
| $\theta_{14}$ [corresponds to $(X_1) * (X_2)^3$] | 0.00008437 | 0.03212909 | 0.09835117 |
| $\theta_{15}$ [corresponds to $(X_2)^4$] | 0.00012088 | -0.02886263 | -0.10891807 |
| $\theta_{16}$ [corresponds to $(X_1)^5$] | 0.00004644 | 0.01191896 | 0.01167576 |
| $\theta_{17}$ [corresponds to $(X_1)^4 * (X_2)$] | 0.00030498 | 0.04885767 | 0.06435231 |
| $\theta_{18}$ [corresponds to $(X_1)^3 * (X_2)^2$] | 0.00004009 | -0.03645269 | -0.07057372 |
| $\theta_{19}$ [corresponds to $(X_1)^2 * (X_2)^3$] | 0.00031386 | -0.01338245 | -0.31858558 |
| $\theta_{20}$ [corresponds to $(X_1) * (X_2)^4$] | 0.00005416 | -0.02948576 | -0.01198597 |
| $\theta_{21}$ [corresponds to $(X_2)^5$] | 0.0005341 | 0.07594459 | -0.01550373 |
| $MSE$ (rounded to 6 decimal places) | 0.702015 | 0.053114 | 0.039115 |

$$MSE_{DummyRegressor} = 0.703895$$

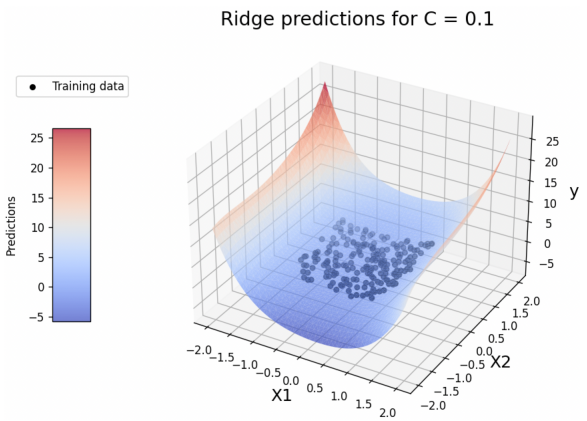
Ridge regression uses a L2 penalty which encourages very small parameters. When C is big (i.e. C $\geq$ 0.1), the L2 penalty is small meaning that we have big parameter values. This leads to a more complex model which likely has a lower MSE, meaning it will take into account "noise" and is likely to be overfitted. This is proven by the low MSE's such as 0.053114 and 0.039115.

When C is small (i.e. C $\leq$ 0.00001), the L2 penalty is big meaning that we have very small parameter values that are close to zero. This leads to the prediction surface being flat and is similar to the dummy regressor that is used as a baseline predictor. This is proven by the fact that $MSE_{C=0.00001}$ (0.702015) $\approx$ $MSE_{DummyRegressor}$ (0.703895). This is not appropriate for our training data that lies on a curve and has a quadratic shape.

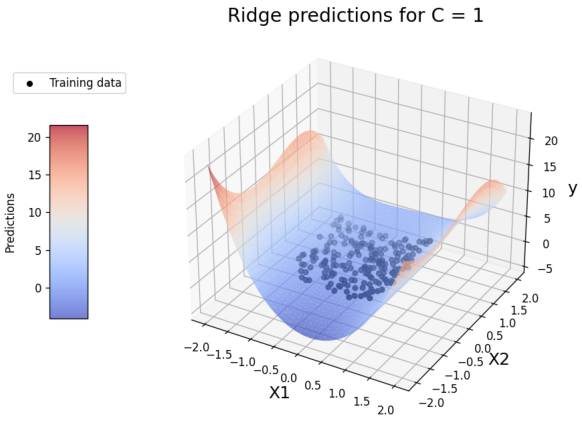
I then used the ridge regression models and generated predictions for the target variable on a grid of feature values that extends beyond the range of values in the dataset. I once again used a range of -2 to 2. I did this using code from 1.5 in the appendix, and plotted the predictions the same way as (i)(c), however I used values for C from above such as 0.00001, 0.1 and 1.




(a) Ridge predictions with C=0.00001




(b) Ridge predictions with C=0.1




(c) Ridge predictions with C=1

Figure 3: 3D plots with ridge predictions for different values of C with the training data

Referring to the plots above, we can see that when C is low (i.e. C=0.0001), the prediction surface is fairly flat and too simple as seen in figure 3 (a) that shows signs of underfitting. When C is bigger (i.e. C=1), the prediction surface becomes very complex and doesn't match our training data as seen in figure 3 (c) that shows signs of underfitting. Our explanations previously are solidified from these plots as we can see that C=0.1 is the best value that is appropriate for the shape of our training data that lies on a curve since it makes the prediction surface follow a curved quadratic shape.

Comparing the impact on the model parameters by changing C for Ridge and Lasso Regression, we can see that by changing C we are only able to increase and decrease the parameter values and therefore not cancel them out completely using Ridge regression. Using Lasso regression, we can change C to completely cancel particular parameters since they can be set to 0. This is much less common in Ridge regression. Therefore using Lasso regression we can easily change the behaviour of the prediction surface by changing C, however using Ridge regression it is difficult to find a value for C that is appropriate for how the training data is shaped.

# Question (ii)

## Part (a)

I plotted the mean and standard deviation of the prediction error vs C using 5-fold cross validation with code from 1.6 in the appendix. From (i)(c) we found that C=1 gives an underfitted model and C=1000 gives an overfitted model while C=10 was the best. Therefore we can use values for C such as 1, 10, 20, 50 and 100 to plot to see which is best from the range of 1 to 100.

```
range_of_values_for_C = [1, 10, 20, 50, 100]
kf = KFold(n_splits=5)
for C in range_of_values_for_C:
    TrainingTemp = []
    TestingTemp = []
    model = Lasso(alpha=1/(2*C))
    for train, test in kf.split(Xpolynomial):
        model.fit(Xpolynomial[train], y[train])
        ypred_test = model.predict(Xpolynomial[test])
        ypred_train = model.predict(Xpolynomial[train])
```

This resulted in the following plot:

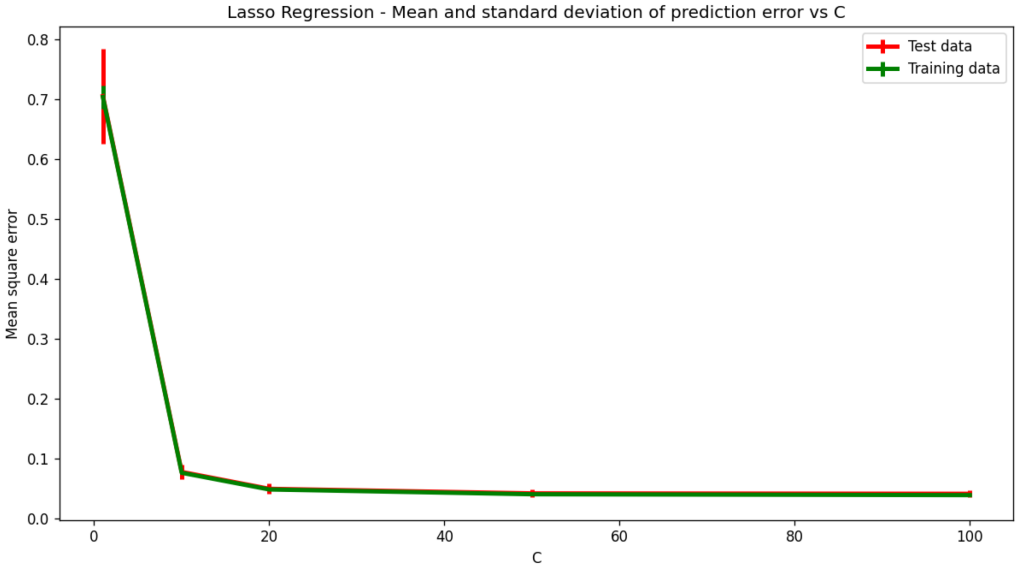



Figure 4: A plot of the mean and standard deviation of the prediction error vs C using lasso regression

## Part (b)

Based on the cross validation data, the value of C that I would recommend would be C=50. The reasoning for this is that the mean squared error is lowest when $C \geq 50$ as seen in figure 4. By choosing C as 50 rather than something lower, we can avoid underfitting. However we also need to worry about overfitting, so we need to choose the model that does not take into account too many factors (i.e. noise) meaning it has to be simple. We can do this by choosing the smallest value of C that we can from the range of $C \geq 50$, which is 50.

## Part (c)

I used 5 fold cross validation to plot the mean and standard deviation of the prediction error vs C using the Ridge model with code from 1.7 in the appendix. From (i)(e) we found that C=0.00001 gives an underfitted model and C=1 gives an overfitted model while C=0.1 was the best. Therefore we can use values for C such as 0.00001, 0.0001, 0.005, 0.01, 0.05 and 0.1 to plot to see which is the best from the range 0.00001-0.1. This resulted in the following plot:

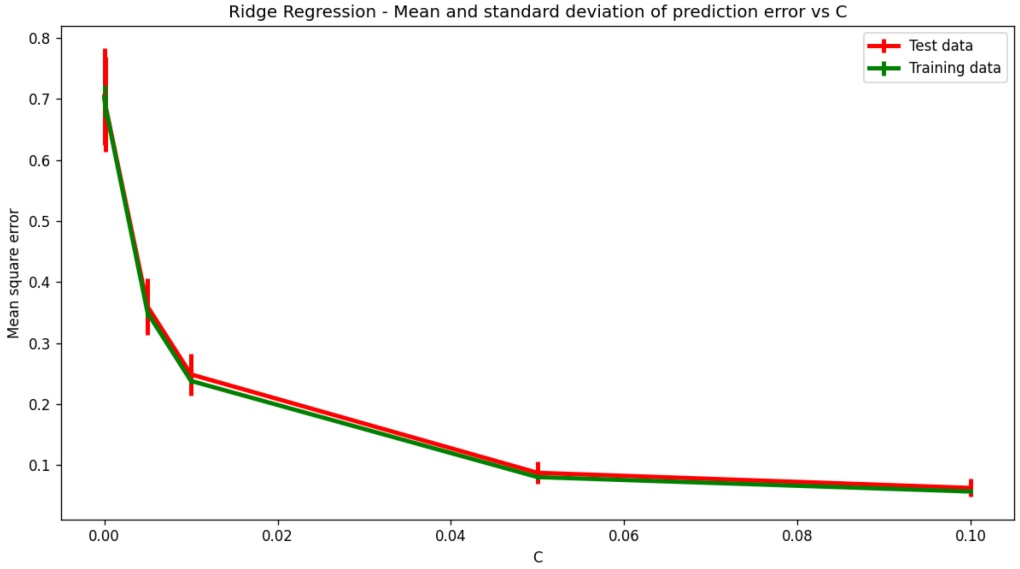



Figure 5: A plot of the mean and standard deviation of the prediction error vs C using ridge regression

Based on the cross validation data, the value of C that I would recommend would be C=0.05. The reasoning for this is that the mean squared error is lowest when C $\geq$ 0.05 as seen in figure 5. By not choosing anything lower than 50, we are avoiding underfitting but we also need to not overfit our model. Therefore, we need to make our model as simple as possible, while not taking into account as much "noise". We can do this by simply choosing the lowest value for C while C $\geq$ 0.05, which is 0.05.

# 1 Appendix

## 1.1 Helper function to read data from CSV

```
import numpy as np
import pandas as pd

def read_data():
    df = pd.read_csv("week3.csv")
    X1 = df.iloc[:, 0]
    X2 = df.iloc[:, 1]
    X = np.column_stack((X1, X2))
    y = df.iloc[:, 2]
    ytrain = np.sign(y)
    return X1, X2, X, y, ytrain
```

## 1.2 Question (i) (a)

```
import matplotlib.pyplot as plt
import matplotlib
from helper_functions import read_data
import numpy as np
matplotlib.use('TkAgg')

def part_a():
    X1, X2, X, y, y_train = read_data()
    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    ax.scatter(X1, X2, y)
    ax.set_title("3D scatter plot of training data", fontsize=17)
    ax.set_xlabel("X1", fontsize=13)
    ax.set_ylabel("X2", fontsize=13)
    ax.set_zlabel("y", fontsize=13)
    ax.set_xticks(np.arange(-1, 1.01, 0.5))
    ax.set_yticks(np.arange(-1, 1.01, 0.5))
    ax.set_zticks(np.arange(-1, 2.01, 0.5))
    plt.show()
```

## 1.3 Question (i) (b)

```
from sklearn.dummy import DummyRegressor
from helper_functions import read_data
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

def part_b():
    np.set_printoptions(suppress=True)
    X1, X2, X, y, y_train = read_data()
    Xpoly = PolynomialFeatures(5)
    XpolyFittedAndTransformed = Xpoly.fit_transform(X)
    compareWithABaseline =
    DummyRegressor(strategy="mean").fit(XpolyFittedAndTransformed, y)
    print("MSE using dummy regressor =", mean_squared_error(y,
    compareWithABaseline.predict(XpolyFittedAndTransformed)))
    for C in [1, 10, 1000]:
        model = Lasso(alpha=(1 / (2 * C))).fit(XpolyFittedAndTransformed,
        y)
        theta = np.append(model.intercept_, model.coef_)
        print("C = ", C)
        print("MSE = ", mean_squared_error(y,
        model.predict(XpolyFittedAndTransformed)))
        print(Xpoly.get_feature_names_out(["X1", "X2"]))
        print("θ = ", theta)
```

## 1.4 Question (i) (c)

```
import matplotlib.pyplot as plt
import matplotlib
from helper_functions import read_data
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import Lasso
from matplotlib import cm

def part_c():
    Xtest = []
    grid = np.linspace(-2, 2)
    for i in grid:
        for j in grid:
            Xtest.append([i, j])
    Xtest = PolynomialFeatures(5).fit_transform(np.array(Xtest))
    X1FromXTest = Xtest[:, 1]
    X2FromXTest = Xtest[:, 2]
    X1, X2, X, y, y_train = read_data()
    Xpolynomial = PolynomialFeatures(5).fit_transform(X)
    for C in [1, 10, 1000]:
        model = Lasso(alpha=1 / (2 * C)).fit(Xpolynomial, y)
        fig = plt.figure(dpi=120)
        ax = fig.add_subplot(111, projection="3d")
        ax.scatter(X1.to_numpy(), X2.to_numpy(), y, color="black",
        label="Training data")
        plot_surface = ax.plot_trisurf(X1FromXTest, X2FromXTest,
        model.predict(Xtest), cmap=cm.coolwarm, alpha=0.7)
        cbar = fig.colorbar(plot_surface, shrink=0.5, aspect=5,
        location="left")
        cbar.ax.get_yaxis().labelpad = 10
        cbar.ax.set_ylabel('Predictions')
        ax.legend(bbox_to_anchor=(-0.4, 0.9), loc='upper left')
        ax.set_xlabel("X1", fontsize=15)
        ax.set_ylabel("X2", fontsize=15)
        ax.set_zlabel("y", fontsize=15)
        ax.set_title("Lasso predictions for C = " + str(C), fontsize=17)
        plt.show()
```

## 1.5 Question (i) (e)

```
import matplotlib.pyplot as plt
import matplotlib
from sklearn.dummy import DummyRegressor
from helper_functions import read_data
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from matplotlib import cm

matplotlib.use('TkAgg')
def part_e():
    Xtest = []
    grid = np.linspace(-2, 2)
    for i in grid:
        for j in grid:
            Xtest.append([i, j])
    Xtest = PolynomialFeatures(5).fit_transform(np.array(Xtest))
    X1FromXTest = Xtest[:, 1]
    X2FromXTest = Xtest[:, 2]
    np.set_printoptions(suppress=True)
```

```
    X1, X2, X, y, y_train = read_data()
    Xpoly = PolynomialFeatures(5)
    XpolyFittedAndTransformed = Xpoly.fit_transform(X)
    baseline =
    DummyRegressor(strategy="mean").fit(XpolyFittedAndTransformed, y)
    print("MSE using dummy regressor =", mean_squared_error(y,
    baseline.predict(XpolyFittedAndTransformed)))
    for C in [0.00001, 0.1, 1]:
        model = Ridge(alpha=1 / (2 * C)).fit(XpolyFittedAndTransformed, y)
        theta = np.append(model.intercept_, model.coef_)
        print("C =", np.format_float_positional(C, trim='-'))
        print("MSE = ", mean_squared_error(y,
        model.predict(XpolyFittedAndTransformed)))
        print(Xpoly.get_feature_names_out(["X1", "X2"]))
        print("θ =", theta)
        fig = plt.figure(dpi=120)
        ax = fig.add_subplot(111, projection="3d")
        ax.scatter(X1.to_numpy(), X2.to_numpy(), y, color="black",
        label="Training data")
        plot_surface = ax.plot_trisurf(X1FromXTest, X2FromXTest,
        model.predict(Xtest), cmap=cm.coolwarm, alpha=0.7)
        cbar = fig.colorbar(plot_surface, shrink=0.5, aspect=5,
        location="left")
        cbar.ax.get_yaxis().labelpad = 10
        cbar.ax.set_ylabel('Predictions')
        ax.legend(bbox_to_anchor=(-0.4, 0.9), loc='upper left')
        ax.set_xlabel("X1", fontsize=15)
        ax.set_ylabel("X2", fontsize=15)
        ax.set_zlabel("y", fontsize=15)
        ax.set_title("Ridge predictions for C = " +
        np.format_float_positional(C, trim='-'), fontsize=17)
        plt.show()
```

## 1.6 Question (ii) (a)

```
from helper_functions import read_data
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import KFold
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
matplotlib.use('TkAgg')

def part_a():
    X1, X2, X, y, y_train = read_data()
    Xpolynomial = PolynomialFeatures(5).fit_transform(X)
    plt.figure(dpi=120)
    TrainingDataStandardError = []
    TrainingDataMeanError = []
    TestingDataStandardError = []
    TestingDataMeanError = []
    range_of_values_for_C = [1, 10, 20, 50, 100]
    kf = KFold(n_splits=5)
    for C in range_of_values_for_C:
```

```
        TrainingTemp = []
        TestingTemp = []
        model = Lasso(alpha=1 / (2 * C))
        for train, test in kf.split(Xpolynomial):
            model.fit(Xpolynomial[train], y[train])
            ypred_test = model.predict(Xpolynomial[test])
            ypred_train = model.predict(Xpolynomial[train])
            TestingTemp.append(mean_squared_error(y[test], ypred_test))
            TrainingTemp.append(mean_squared_error(y[train], ypred_train))
        TrainingDataStandardError.append(np.array(TrainingTemp).std())
        TrainingDataMeanError.append(np.array(TrainingTemp).mean())
        TestingDataStandardError.append(np.array(TestingTemp).std())
        TestingDataMeanError.append(np.array(TestingTemp).mean())
    plt.title("Lasso Regression - Mean and standard deviation of
    prediction error vs C")
    plt.ylabel("Mean square error")
    plt.xlabel("C")
    plt.errorbar(range_of_values_for_C, TestingDataMeanError,
    yerr=TestingDataStandardError, linewidth=3, c="red",
                 label="Test data")
    plt.errorbar(range_of_values_for_C, TrainingDataMeanError,
    yerr=TrainingDataStandardError, linewidth=3, c="green",
                 label="Training data")
    plt.legend()
    plt.show()
```

## 1.7 Question (ii) (c)

```
from helper_functions import read_data
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
matplotlib.use('TkAgg')
def part_c():
    X1, X2, X, y, y_train = read_data()
    Xpolynomial = PolynomialFeatures(5).fit_transform(X)
    plt.figure(dpi=120)
    TrainingDataStandardError = []
    TrainingDataMeanError = []
    TestingDataStandardError = []
    TestingDataMeanError = []
    range_of_values_for_C = [0.00001, 0.0001, 0.005, 0.01, 0.05, 0.1]
    kf = KFold(n_splits=5)
    for C in range_of_values_for_C:
        TrainingTemp = []
        TestingTemp = []
        model = Ridge(alpha=1 / (2 * C))
        for train, test in kf.split(Xpolynomial):
            model.fit(Xpolynomial[train], y[train])
            ypred_test = model.predict(Xpolynomial[test])
            ypred_train = model.predict(Xpolynomial[train])
            TestingTemp.append(mean_squared_error(y[test], ypred_test))
```

```
            TrainingTemp.append(mean_squared_error(y[train], ypred_train))
        TrainingDataStandardError.append(np.array(TrainingTemp).std())
        TrainingDataMeanError.append(np.array(TrainingTemp).mean())
        TestingDataStandardError.append(np.array(TestingTemp).std())
        TestingDataMeanError.append(np.array(TestingTemp).mean())
    plt.title("Ridge Regression - Mean and standard deviation of
    prediction error vs C")
    plt.ylabel("Mean square error")
    plt.xlabel("C")
    plt.errorbar(range_of_values_for_C, TestingDataMeanError,
    yerr=TestingDataStandardError, linewidth=3, c="red",
                label="Test data")
    plt.errorbar(range_of_values_for_C, TrainingDataMeanError,
    yerr=TrainingDataStandardError, linewidth=3, c="green",
                label="Training data")
    plt.legend()
    plt.show()
```