**Machine Learning – Week 3 Assignment**

The data I am using has the following first line: # id:12-12-12

1. The scatter plot generated through running my code can be seen in figure 1. The training data looks like it lies on a curve rather than a plane, if change our perspective of the plot to look like figure 2. Feature 2 and Y can be seen to correlate positively in a linear fashion. As feature 2 rises we can also see that the output rises.

As feature 1 approaches 0, the output (Y) also appears to descend. Feature 1 seems to have a quadratic curve in relation to the output. My code can be found below.

**Figure 1 and 2:**

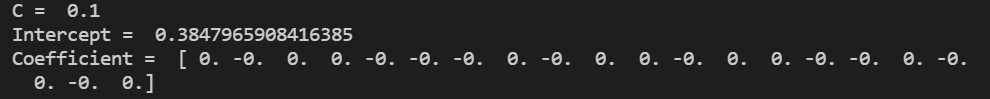
Chart, scatter chart

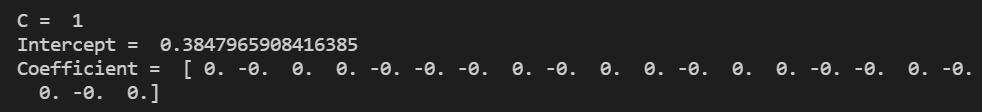
Description automatically generatedChart, scatter chart

Description automatically generated

1. Passing the parameter ‘degree = 5’ to the PolynomialFeatures class and fitting it to our 2 features allowed me to create extra features.

We can use the alpha parameter of sklearn’s Lasso class to specify how much of a regularization penalty we want to add to the loss function. This parameter multiplies the L1 term. This means if we set alpha to 0, it is the same as least squares regression. I used values of 0.1, 1, 10, 100, 1000, 4000 and 5000 for C and the results can be seen below.





A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

We can see that as the value of C increases from 0, the intercept decreases up until when C = 100, where it begins to increase slowly once again. For lower values of C, less of the polynomial features in the coefficient are affected. As C is increased, more and more of these values begin to change from 0 as expected with L1 regularisation. This form of regularisation can help with feature selection as some of the less important features are shrunk to 0.

1. For each of the models I chose in part b), I generated the predictions using the intercept and coefficient features. I chose the range -5 to 5 and plotted the predictions and training data on a 3D plot. The training data can be seen in green and the predictions in blue.

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated

Chart, surface chart

Description automatically generatedChart, line chart

Description automatically generated

Chart

Description automatically generatedChart

Description automatically generated

Chart

Description automatically generated Chart, surface chart

Description automatically generated

Chart, radar chart

Description automatically generated Chart, surface chart

Description automatically generated

Chart

Description automatically generated Chart, surface chart

Description automatically generated

For lower values of C we can see underfitting occurring as the relationship between input and output is not accurately displayed. This is expected as the coefficients from part b) are all 0 for C = 0.1 and C = 0. The graphs for C = 0.1 and C = 1 are identical as the model fails to capture the behaviour of the data.

For higher values of C we can begin to see the model fit the data better. For C = 10 and C = 100 a curve which fits well can be seen. When the model leaves the training data we do not want it to diverge too far as overfitting can begin to occur. This can be seen for all of the graphs with values of C above 100. Once the model leaves the training data for C = 1000, C = 4000 and C = 5000, the curve diverges inaccurately.

To conclude it seems that a value for C between 10 and 100 would fit most appropriately. With this knowledge I will test this for C = 40. The graph generated can be found below and it seems to fit the training data and not skew too far after passing it. This shows it would be a good value for C.

Chart, line chart

Description automatically generated

1. Underfitting and Overfitting are the causes of poor performance within machine learning. Overfitting is when a model learns about the training data and the noise in the training data so that it has a negative impact on the models performance when it comes to interpreting new data. Any noise or outliers within the training data is learned by the model and taken into account for future predictions, however this noise should be ignored in most cases. This can be avoided by limiting the amount of detail the model learns.

Underfitting is easy to detect as the model will perform poorly with the training data. As I stated before the relationship between the input and output is modelled inaccurately, leading to errors. This can occur with a model with not enough features or if it has been over-regularized.

Regularization is a technique used to shrink the coefficients towards 0. When noise is encountered, if the estimated coefficients are close to 0 they can generalize new data better. In lasso regression, a weight is added to the penalty which can be seen here: Text

Description automatically generated with medium confidence. This weight is C, as we have been discussing in part a), b) and c). We can see here as the value or ‘cost’, C, grows, its effect lessens. If we take the model I used for example and look at part b). When C = 10, there are 2 features trending away from 0. Then when C = 1000, there are 6 features trending away from 0. As we increase C, it leads to coefficients trending further away from 0, which can lead to overfitting. This overfitting can be seen clearly in the graphs generated for values of C above 100 in part c). In order to find the best value for C, you must test lots of values to see which best fits your model.

1. i)

We can use the alpha parameter of sklearn’s Ridge class to specify how much of a regularization penalty we want to add to the loss function. This parameter multiplies the L2 term. Once again, I used values of 0.1, 1, 10, 100, 1000, 4000 and 5000 for C and the results can be seen below.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

We can see that as the value of C increases from 1, the intercept begins to decrease.

Ridge regression is clearly different to lasso as all of the coefficient features are used and trending away from 0. For lower values of C, less of the polynomial features in the coefficient are affected for lasso regression. L2 regularization add the value of the squared weights compared to L1 regularization which adds the absolute value of the weights. Ridge regression never sets the coefficients to absolute zero which lasso does. This is an evident difference.

ii) For each of the models chosen above, I again generated the predictions using the intercept and coefficient features. I chose the range -5 to 5 and plotted the predictions and training data on a 3D plot. The training data can be seen in green and the predictions in blue.

Chart, surface chart

Description automatically generatedChart, radar chart

Description automatically generated

Chart, surface chart

Description automatically generatedChart, radar chart

Description automatically generated

Chart, radar chart

Description automatically generatedChart, surface chart

Description automatically generated

Chart, radar chart

Description automatically generatedChart, surface chart

Description automatically generated

Chart, radar chart, surface chart

Description automatically generated Chart, surface chart

Description automatically generated

Chart, radar chart

Description automatically generatedChart, surface chart

Description automatically generated

For lower values of C we can see a significantly less amount of underfitting occurring compared to the Lasso model. As stated before, ridge regression never sets the coefficients to absolute zero which lasso does. This makes it harder for us to determine whether the graphs are underfitting or overfitting. We can take an educated guess however based on how closely the model fits the training data.

Using this basis we can estimate that for the values of C greater than 10 the model is overfitting. This is because the model trends diverges rapidly after it exits the training data. The graphs for all values above C = 10 all look similar in this way. They all diverge abruptly away from the training data towards infinity. This gives the impression that a lower value for C would be optimum. The model for C = 0.1 seems the best fit but it is our smallest value for C. Testing C with smaller values will be useful. The graphs for C = 0.01 and C = 0.001 can be found below.

These graphs seem to fit the training data slightly better. It can be safe to assume that for values of C below 0.01, the model begins to become underfitting.   
The optimum value for C based on the graphs in my opinion should be C = 0.1.

Chart, surface chart

Description automatically generated Chart, surface chart

Description automatically generated

Chart, surface chart

Description automatically generated Chart, surface chart

Description automatically generated

It was decided that a value for C around 40 was optimum with Lasso regression, whereas with ridge regression we decided around C = 0.1. The large difference is attributed to the difference in the regularisation occurring using both methods. (L1 and L2). For low values of C, the Ridge model seems to fit better as it uses all features either way, whereas the Lasso model does not use many. Overfitting begins to occur when we use higher values of C with the Ridge model as each feature will have a more profound effect. This works similarly for the Lasso model but more features are used with higher values of C.

I used the same function calls for creating the predicted values and drawing the graph as in part c), but added a new function to create a ridge model. The code can be found below.

ii)

1. The graph I generated can be found below. I chose to increase C by a factor of 5 as recommended. This is so I can scan across a large range quickly. The values I chose were [0.1, 0.5, 1, 5, 10, 50, 100]. Overfitting seems to occur for C values over 40, and underfitting begins to occur for C values under 10 or so.

Chart

Description automatically generated

1. I would choose a value between 10 and 40. For any values less, the penalty is too strong and we can begin to see underfitting occurring as we see the mean square error increase. However, to avoid overfitting we want to choose the smallest possible value of C. This means my final decision would be to use a value of 10 for C.

The code used for part a) and b) can be found below.

1. i)

The graph I generated can be found below. I chose to increase C by a factor of 5 again as recommended. This is so I can scan across a large range quickly. The values I chose were [0.1, 0.5, 1, 5, 10, 50, 100]. Overfitting seems to occur for C values over 1, and underfitting begins to occur for C values under 0.5 or so.

Chart

Description automatically generated

Chart, line chart

Description automatically generated

ii)

I would choose a value between 0.5 and 1. For any values less, the penalty seems slightly too strong but not terrible. We can begin to see slight underfitting occurring as we see the mean square error increase. To avoid overfitting we want to choose the smallest possible value of C. This means my final decision would be to use a value of 0.5 for C.

The code used for the Ridge model for part a) and b) can be found below.

**Appendix:**

import numpy as np # id:12-12-12

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

df = pd.read\_csv("week3.csv")

X1 = df.iloc[:,0]

X2 = df.iloc[:,1]

X = np.column\_stack((X1,X2))

y = df.iloc[:,2]

def LReg(polyX, y, X):

for C in [0.1, 1,10, 100, 1000, 4000, 5000]:

model = linear\_model.Lasso(alpha=(1/(2\*C)))

model.fit(polyX, y)

print("C = ", C)

print("Intercept = ", model.intercept\_)

print("Coefficient = ", model.coef\_)

print("\n")

Xtest1, Xtest2, predictions = generatePredictions(model.intercept\_, model.coef\_)

title = "C = " + str(C)

displayPredictedGraph(Xtest1, Xtest2, predictions, X, y, title)

def RReg(polyX, y, X):

for C in [0.1, 1, 10, 100, 1000, 4000, 5000]:

model = linear\_model.Ridge(alpha=1/(2\*C))

model.fit(polyX, y)

print("C =", C)

print("Intercept = ", model.intercept\_)

print("Coefficient = ", model.coef\_)

print("\n")

title = "C = " + str(C)

Xtest1, Xtest2, predictions = generatePredictions(model.intercept\_, model.coef\_)

displayPredictedGraph(Xtest1, Xtest2, predictions, X, y, title)

def generatePredictions(intercept\_, coef\_):

Xtest1 = []

Xtest2 = []

Ytest = []

grid = np.linspace(-5, 5)

poly = PolynomialFeatures(5)

for i in grid:

YPredictions = []

i1 = []

j1 = []

for j in grid:

result = poly.fit\_transform(np.column\_stack((i, j)))

YPredictions.append(sum(coef\_ \* result[0, :]) + intercept\_)

i1.append(i)

j1.append(j)

Ytest.append(YPredictions)

Xtest1.append(i1)

Xtest2.append(j1)

Xtest1 = np.array(Xtest1)

Xtest2 = np.array(Xtest2)

Ytest = np.array(Ytest)

return Xtest1, Xtest2, Ytest

def displayPredictedGraph(Xtest1, Xtest2, predictedY, X, y, title):

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], y, c = 'g')

ax.plot\_surface(predictedY, Xtest1, Xtest2, antialiased=False, alpha=0.2)

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

ax.set\_zlabel("y")

plt.title(title)

plt.legend(["Training Data"])

plt.show()

def crossVal5FoldLReg(X):

mean\_error=[]; std\_error=[]

Ci\_range = [0.1, 0.5, 1, 5, 10, 50, 100]

for c in Ci\_range:

model = linear\_model.Lasso(alpha=1/(2\*c))

temp = []

kf = KFold(n\_splits=5)

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

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

ypred = model.predict(X[test])

temp.append(mean\_squared\_error(y[test],ypred))

mean\_error.append(np.array(temp).mean())

std\_error.append(np.array(temp).std())

print("Mean Error = ", mean\_error)

print("\n")

print("Standard Deviation Error = ", std\_error)

print("\n")

plt.title("5-fold cross-validation, Lasso Mean + Standard Deviation Error Vs C")

plt.plot(Ci\_range, mean\_error)

plt.errorbar(Ci\_range, mean\_error, yerr=std\_error, fmt ='ro', label="Standard Deviation")

plt.xlabel("Ci")

plt.ylabel("Mean square error")

plt.legend()

plt.show()

def crossVal5FoldRReg(X):

mean\_error=[]; std\_error=[]

Ci\_range = [0.1, 0.5, 1, 5, 10, 50, 100]

for c in Ci\_range:

model = linear\_model.Ridge(alpha=1/(2\*c))

temp = []

kf = KFold(n\_splits=5)

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

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

ypred = model.predict(X[test])

temp.append(mean\_squared\_error(y[test],ypred))

mean\_error.append(np.array(temp).mean())

std\_error.append(np.array(temp).std())

print("Mean Error = ", mean\_error)

print("\n")

print("Standard Deviation Error = ", std\_error)

print("\n")

plt.title("5-fold cross-validation, Ridge Mean + Standard Deviation Error Vs C")

plt.plot(Ci\_range, mean\_error)

plt.errorbar(Ci\_range, mean\_error, yerr=std\_error, fmt ='ro', label="Standard Deviation")

plt.xlabel("Ci")

plt.xlim((0,50))

plt.ylabel("Mean square error")

plt.legend()

plt.show()

def scatterPlot(features, output):

fig = plt.figure()

ax = fig.add\_subplot(111 , projection = '3d')

ax.scatter(features[:,0], features[:,1], output)

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend(["Training data"])

plt.show()

poly = PolynomialFeatures(degree=5)

result = poly.fit\_transform(X)

scatterPlot(X, y)

LReg(result, y, X)

RReg(result, y, X)

crossVal5FoldLReg(X)

crossVal5FoldRReg(X)