Table of Contents

[Other Machine Learning Algorithms for Regression and Classification 1](#_Toc187418157)

[Reshaping Data 1](#_Toc187418158)

[Linear Regression with Scaling 3](#_Toc187418159)

[Gradient Descent 5](#_Toc187418160)

[Ridge Regression 6](#_Toc187418161)

[Lasso Regression 8](#_Toc187418162)

[Elastic Net Regression 9](#_Toc187418163)

[Equation Comparison 11](#_Toc187418164)

[Support Vector Machines 12](#_Toc187418165)

[Summary 15](#_Toc187418166)

# Other Machine Learning Algorithms for Regression and Classification

<https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/>

Since the start of the course, we have discussed several machine learning algorithm alternatives which perform linear regression and classification. During this section we will extend the stochastic gradient descent algorithm and also will briefly cover support vector machines.

## Reshaping Data

Many algorithms will require a transformation to adjust how X and y values are arranged. The need to transform, scale and then revert back to original formats is magnified when working with machine learning algorithms. The need to switch data formats can be frustrating but the problem will become less troubling with practice and experience. A common error to watch for is:

**Reshape your data either using array.reshape(-1, 1)**

Example 1: Resolving the Reshape Error

This example demonstrates the reshape error. Initially the data is presented as:

642 5

679 5

473 5

390 8

1096 6

When trying to scale the data, the following error displays:

**Reshape your data either using array.reshape(-1, 1)**

The code needs to be transformed into a set of vertical arrays.

[[5]

[5]

[5]

...

[6]

[6]

[5]]

Here is the code which leads to the error while scaling:

|  |
| --- |
| import pandas as pd  import numpy as np  from sklearn.metrics import mean\_squared\_error  from sklearn.model\_selection import train\_test\_split  import statsmodels.api as sm  PATH = "/Users/pm/Desktop/DayDocs/data/"  CSV\_DATA = "winequality.csv"  dataset = pd.read\_csv(PATH + CSV\_DATA)  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Include only statistically significant columns.  X = dataset[['volatile acidity', 'chlorides', 'total sulfur dioxide',  'pH', 'sulphates','alcohol']]  y = dataset['quality']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  # Stochastic gradient descent models are sensitive to differences  # in scale so a MinMax is usually used.  from sklearn.preprocessing import MinMaxScaler  scalerX = MinMaxScaler()  scalerX.fit(X\_train)  print(y\_train)  # Build scaler for y.  scalerY = MinMaxScaler()  scalerY.fit(y\_train) |

To fix the error we use:

y\_train = np.array(y\_train).reshape(-1, 1)

To transform the y values to the following format of vertical arrays:

Modified y\_train

[[5]

[5]

[5]

...

[6]

[6]

[5]]

To build this example, replace the code that performs the scaling in Example 1 with the following:

|  |
| --- |
| import numpy as np  # Create array of y-values and reshape.  y\_train = np.array(y\_train).reshape(-1, 1)  print("\nModified y\_train")  print(y\_train)  # Build scaler for y.  scalerY = MinMaxScaler()  scalerY.fit(y\_train)  y\_scaled = scalerY.transform(y\_train)  print("\nScaled y")  print(y\_scaled) |

## Linear Regression with Scaling

Example 2: Linear Regression with Scaling

Here is linear regression with scaling. The RMSE is 0.620698340123314.

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn                 import metrics  import statsmodels.api       as sm  import numpy as np  PATH     = "/Users/pm/Desktop/DayDocs/data/"  CSV\_DATA = "winequality.csv"  dataset  = pd.read\_csv(PATH + CSV\_DATA)  X = dataset[['volatile acidity', 'chlorides', 'total sulfur dioxide', 'sulphates',               'alcohol']]  # Adding an intercept \*\*\* This is requried \*\*\*. Don't forget this step.  # The intercept centers the error residuals around zero  # which helps to avoid over-fitting.  X\_withConst = sm.add\_constant(X)  y = dataset['quality'].values  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  import statsmodels.api       as sm  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Include only statistically significant columns.  X = dataset[['volatile acidity', 'chlorides', 'total sulfur dioxide',               'pH', 'sulphates','alcohol']]  X = sm.add\_constant(X)  y = dataset['quality']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)  # Stochastic gradient descent models are sensitive to scaling.  # Fit X scaler and transform X\_train.  from sklearn.preprocessing import StandardScaler  scalerX = StandardScaler()  X\_train\_scaled = scalerX.fit\_transform(X\_train)  # Build y scaler and transform y\_train.  scalerY = StandardScaler()  y\_train\_scaled = scalerY.fit\_transform(np.array(y\_train).reshape(-1,1))  # Scale test data.  X\_test\_scaled = scalerX.transform(X\_test)  def performLinearRegression(X\_train, X\_test, y\_train, y\_test, scalerY):      model = sm.OLS(y\_train, X\_train).fit()      scaledPredictions = model.predict(X\_test) # make the predictions by the model      predictions = scalerY.inverse\_transform(np.array(scaledPredictions).reshape(-1,1))      print(model.summary())      print('Root Mean Squared Error:',            np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))      return predictions  predictions = performLinearRegression(X\_train\_scaled, X\_test\_scaled,                                        y\_train\_scaled, y\_test, scalerY) |

## Gradient Descent

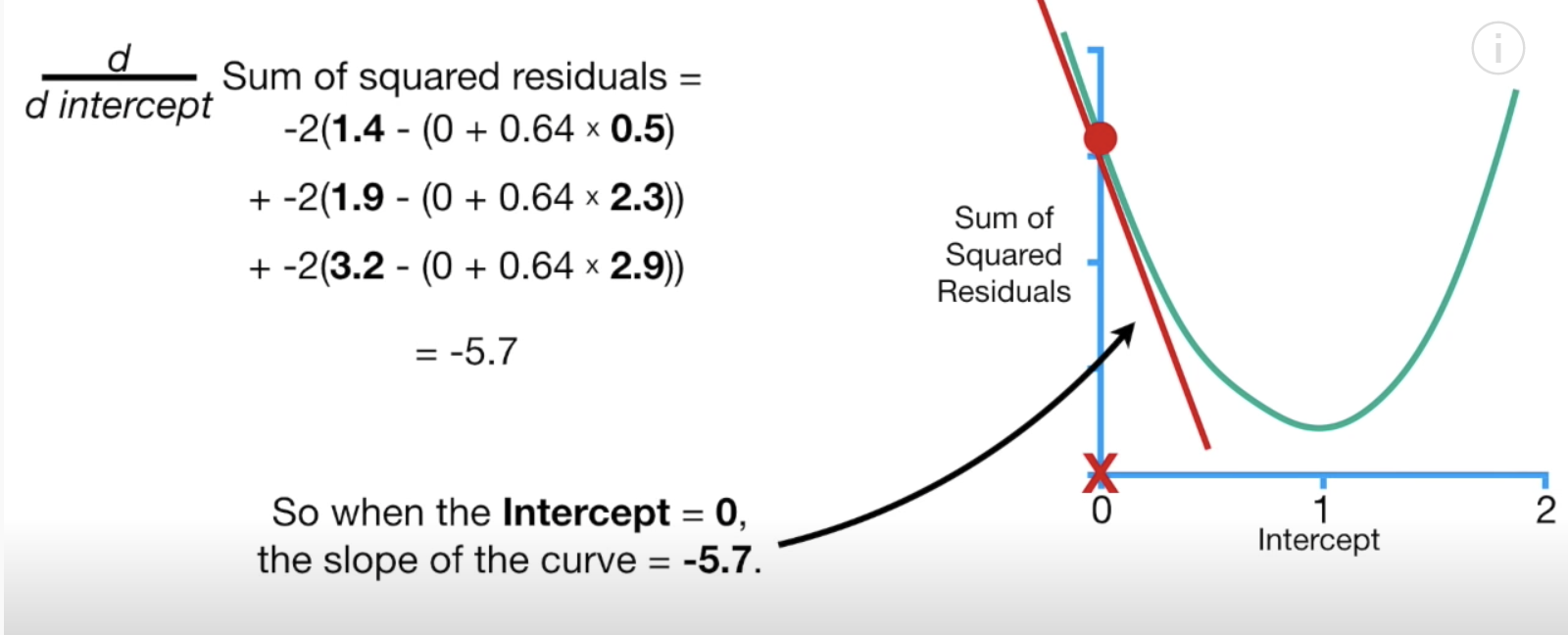
On the first day of COMP4948 we also discussed stochastic gradient descent. In case you need a review, you may want to watch Josh Stormer’s video where he explains how gradient descent works at <https://youtu.be/sDv4f4s2SB8>.

Gradient descent finds the minimum of the loss function which in turn identifies the best fit line. The example in the video uses an ordinary least squares regression. The loss function uses the equation to quantify the deviation between expected and actual values.

Gradient descent involves taking the derivative of the loss function to find the lowest point of the function’s curve. Derivatives with respect to different predictor variables are called gradients.

Given that we are only making adjustments for different intercepts, Figure 1 shows how the gradient descent algorithm could calculate the slope of the loss function when the intercept is 0. The slope of -5.7 is not optimal but it is a start. Ideally, we want to find a slope that is as close to zero as possible somewhere around where the intercept equals 1.

Figure : Following Gradients to the Minimum



Example : Stochastic Gradient Descent

This example shows how to build a predictive model using stochastic gradient descent.

To build this example, add this code to the end of Example 2.

The root mean squared error from SGD is 0.68442053944026. The RMSE in this case is higher than the RMSE value of 0.6259157889490549 that was generated with the ordinary least-squares regression.

|  |
| --- |
| from sklearn.linear\_model import SGDRegressor  def performSGD(X\_train, X\_test, y\_train, y\_test, scalerY):  sgd = SGDRegressor()  sgd.fit(X\_train, y\_train)  print("\n\*\*\*SGD=")  predictionsUnscaled = sgd.predict(X\_test)  predictions = scalerY.inverse\_transform(  np.array(predictionsUnscaled).reshape(-1,1))  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test,  predictions)))  predictions = performSGD(X\_train\_scaled, X\_test\_scaled,  y\_train\_scaled, y\_test, scalerY) |

## Ridge Regression

On the first day of COMP4948 we also discussed ridge regression. Josh Stormer’s video explains ridge regression nicely in detail in case you would like to review it

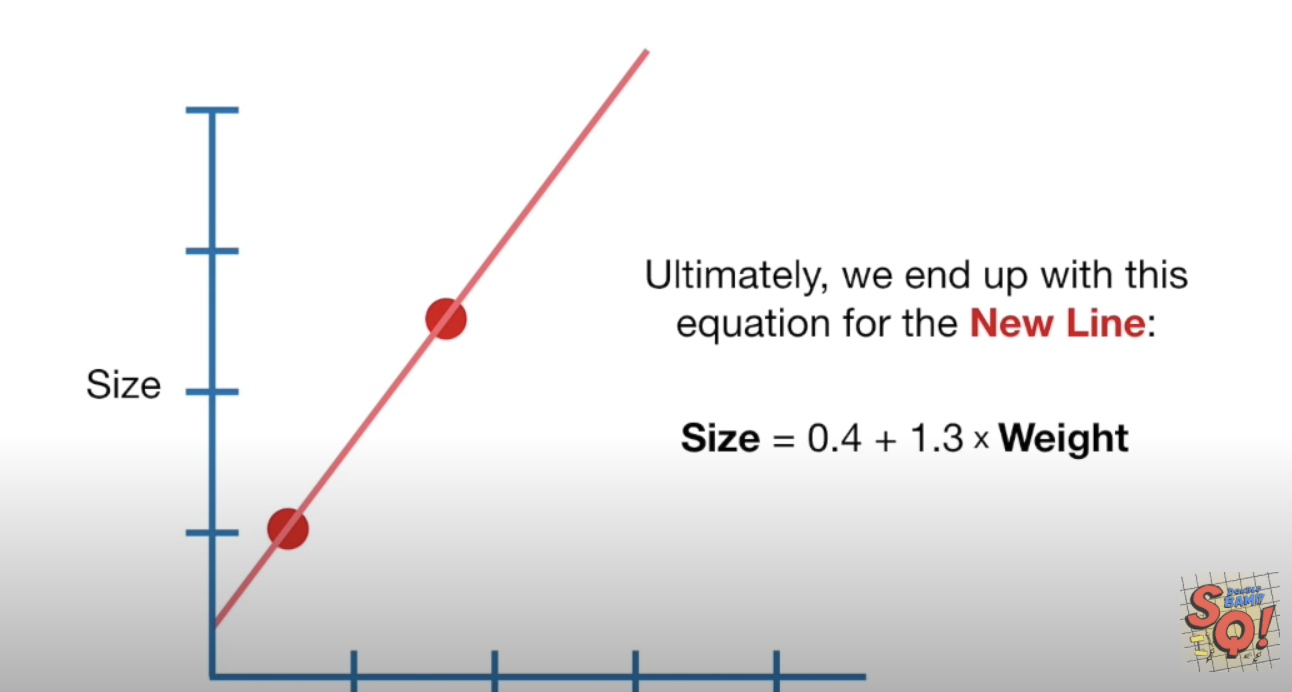
<https://www.youtube.com/watch?v=Q81RR3yKn30>

**Note:** Josh Stormer’s videos refers to as the parameter which sets the strength of L1 and L2 penalties. (We will talk about the L1 and L2 penalties shortly). However, sklearn uses since sklearn already refers to as the learning rate. These notes will also use to refer to the strength of the L1 and L2 penalties. The conflicting references to the penalty strength are confusing so I will try my best to make it as clear as I can.

As Josh explains, the original ordinary least squares best-fit line in Figure 2 is a perfect fit for the training data. The model equation in this case is:

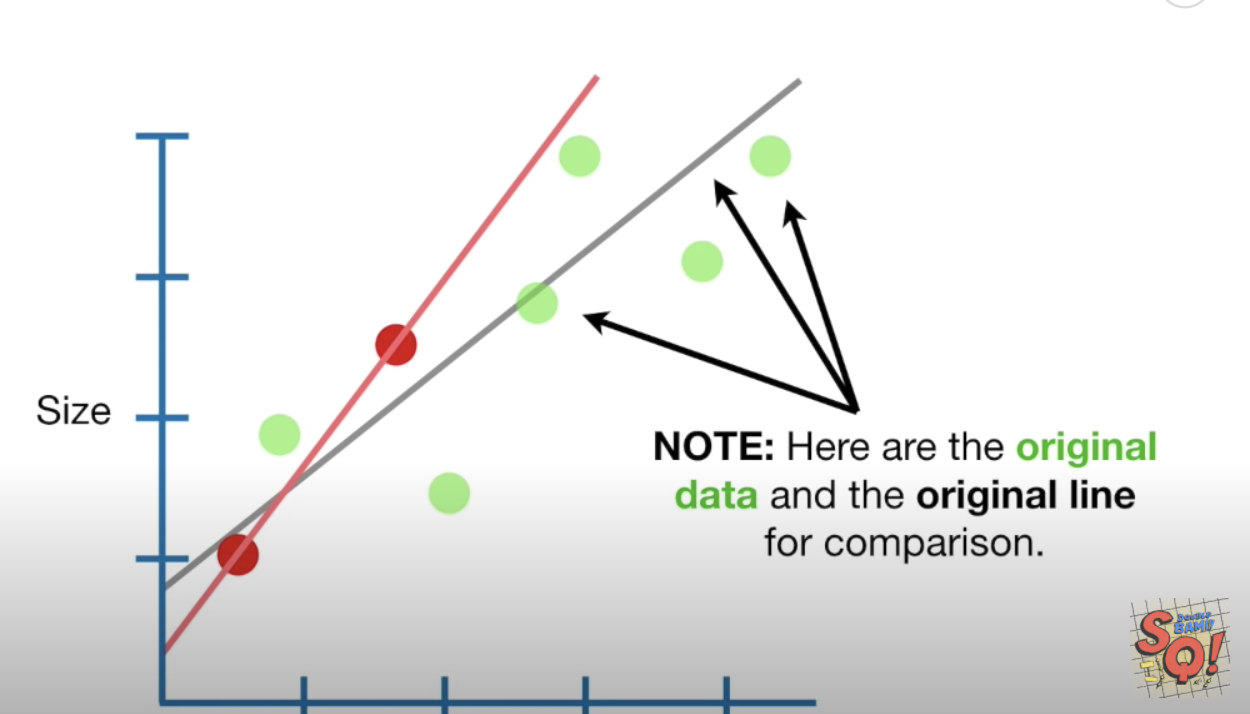
intercept + slope

Figure : Fitting a best-fit line to training data without a penalty



However, if we were given new data which is represented by the green dots in Figure 3, it is obvious that the original model does not fit the new data well.

Figure : Poor fit with new data



Ridge regression adds an **L2** penalty which equals to the best fit line equation.

determines how strong the penalty is. The new model equation becomes;

intercept + slope +

Ridge regression then includes the penalty when calculating gradients to minimize the loss function. The loss function in this case is;

=

When the minimum point of the loss function is found, it is used to identify the best fit line.

Example : Ridge Regression

This example uses ridge regression to model the best fit line. To build this example add this code to the end of Example 3. When alpha=0.17, the RMSE of 0 0.623445 for ridge regression is slightly better than 0.6259157889490531 for ordinary least squares regression.

The coefficients in this case become:

Intercept: 3.349750965754926

'volatile acidity', 'chlorides', 'total sulfur dioxide', 'sulphates', 'alcohol'

[-1.09042919 -1.5465499 -0.00208941 0.79016659 0.24841788]

To build this program, add this code to the end of Example 3.

|  |
| --- |
| from sklearn.linear\_model import Ridge  def ridge\_regression(X\_train, X\_test, y\_train, y\_test, alpha, scalerY):  # Fit the model  ridgereg = Ridge(alpha=alpha)  ridgereg.fit(X\_train, y\_train)  y\_pred\_scaled = ridgereg.predict(X\_test)  # predictions = scalerY.inverse\_transform(y\_pred.reshape(-1,1))  print("\n\*\*\*Ridge Regression Coefficients \*\* alpha=" + str(alpha))  print(ridgereg.intercept\_)  print(ridgereg.coef\_)  y\_pred = scalerY.inverse\_transform(np.array(y\_pred\_scaled).reshape(-1,1))  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  test\_size=0.2, random\_state=0)  alphaValues = [0, 0.16, 0.17, 0.18]  for i in range(0, len(alphaValues)):  ridge\_regression(X\_train\_scaled, X\_test\_scaled, y\_train\_scaled,  y\_test, alphaValues[i], scalerY) |

Exercise (1 mark)

What is the RMSE value of ridge regression when alpha = 0?

|  |
| --- |
|  |

Exercise (1 mark)

How does the RMSE from OLS compare with the RMSE of ridge regression when alpha = 0?

|  |
| --- |
| This is OLS Summary below. The RMSE is exactly the same in comparison to the above Ridge regression RMSE vs the below OLS regression RMSE |

Exercise (1 mark)

In addition to giving the same result as OLS, when alpha equals 0 ridge regression also provides a result that is identical to which other algorithm?

|  |
| --- |
| It also gives the same as Linear Regression |

## Lasso Regression

A problem with ridge regression is that it cannot fully eliminate any of insignificant predictor variables unless we remove them manually. Lasso regression addresses this problem by adding an **L1 penalty** of so insignificant predictor variables vanish.

intercept + slope +

To understand lasso regression in detail, see Josh Stormer’s videos at

<https://www.youtube.com/watch?v=NGf0voTMlcs> (Lasso regression)

<https://www.youtube.com/watch?v=Xm2C_gTAl8c> (Comparing lasso and ridge regression)

Example : Lasso

To implement lasso regression, add the code below Example 4. Lasso does very poorly. For this case, lasso set all predictor coefficients to zero so the intercept of 5.6466 is the only predictive feature in the entire model. It turns out that 5.6465 is approximately the average wine quality rating.

|  |
| --- |
| \*\*\*Lasso Regression Coefficients \*\* alpha=0.1  3.0833025347842193  [ 0. -0. -0. -0.00268159 0. 0.25827142]  Root Mean Squared Error: 0.6606896871528144  \*\*\*Lasso Regression Coefficients \*\* alpha=0.5  5.836986684298963  [ 0. -0. -0. -0.00404099 0. 0. ]  Root Mean Squared Error: 0.7450216331718217  \*\*\*Lasso Regression Coefficients \*\* alpha=1  5.8160845378755255  [ 0. -0. -0. -0.00359734 0. 0. ]  Root Mean Squared Error: 0.7455442007369433 |

Here is the code:

|  |
| --- |
| from sklearn.linear\_model import Lasso  def performLassorRegression(X\_train, X\_test, y\_train, y\_test, alpha,  scalerY):  lassoreg = Lasso(alpha=alpha)  lassoreg.fit(X\_train, y\_train)  y\_pred\_scaled = lassoreg.predict(X\_test)  y\_pred = scalerY.inverse\_transform(np.array(y\_pred\_scaled).reshape(-1,1))  print("\n\*\*\*Lasso Regression Coefficients \*\* alpha=" + str(alpha))  print(lassoreg.intercept\_)  print(lassoreg.coef\_)  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))  alphaValues = [0, 0.1, 0.5, 1]  for i in range(0, len(alphaValues)):  performLassorRegression(X\_train\_scaled, X\_test\_scaled, y\_train\_scaled,  y\_test, alphaValues[i], scalerY) |

Lasso did poorly in this case but we will see shortly how it can make a difference as part of the elastic net algorithm.

## Elastic Net Regression

Elastic Net is useful when there are many predictor variables and we do not know which ones are actually significant for making a prediction. Elastic Net regression combines the L1 penalty from lasso regression and the L2 penalty from ridge regression.

For a reference consider reading:

<https://machinelearningmastery.com/elastic-net-regression-in-python/>

You might watch Josh Stormer’s video on elastic net for a better understanding at;

<https://www.youtube.com/watch?v=1dKRdX9bfIo>

The equation for generating the best fit line with elastic net regression is;

intercept + slope +

As mentioned at the start of this document, the conflicting variable references for the strength of L1 and L2 penalties is confusing. The sklearn package also provides an additional l1\_ratio parameter for their ElasticNet regressor to set the strength level of the L1 penalty. This extra parameter makes the terminology even more confusing so rather than overthinking it – I recommend just grid searching for the *alpha* and *l1\_ratio* parameters to find the best results.

Example : ElasticNet Regression

This example demonstrates how to implement elastic net regression. When running the code, given all potential predictor variables, elastic net chose to only implement the L2 penalty. We know this because the l1ratio is set to 0 where the RMSE is at its lowest value of 0.61973. It is possible though that the model is overfit. A way to test this further would involve cross fold validation.

\*\*\*ElasticNet Regression Coefficients \*\* alpha=0.01 l1ratio=0

2.9628181790919284

[ 0.0248009 -0.90311047 -0.00991496 0.00671778 -0.26896603 0.00376948

-0.00286842 -0.00340709 -0.18190618 0.55111427 0.31511091]

Root Mean Squared Error: 0.619732479961025

If we are concerned about overfitting, we can assign more strength to the L1 penalty for lasso regression. When an L1 penalty is used you will notice that some of the insignificant predictor variables are eliminated. The RMSE values from the elastic net regression with the L1 and L2 penalties still out-perform all OLS, SGD, ridge and lasso models.

\*\*\*ElasticNet Regression Coefficients \*\* alpha=0.01 l1ratio=0.25

2.5743182663196325

[ 0.03011923 -0.9144444 -0. 0.00502741 -0.05385138 0.00359964

-0.00275648 -0. -0.06977843 0.50768012 0.31363392]

Root Mean Squared Error: 0.6214618411645152

\*\*\*ElasticNet Regression Coefficients \*\* alpha=0.01 l1ratio=0.5

2.357912146791541

[ 0.03387944 -0.92442306 0. 0.00376742 -0. 0.00348747

-0.00267609 -0. -0. 0.47166676 0.31168149]

Root Mean Squared Error: 0.6225075422100831

Here is the code that implements elastic net regression. Add this code to the end of Example 6.

|  |
| --- |
| from sklearn.linear\_model import ElasticNet  bestRMSE = 100000.03  def performElasticNetRegression(X\_train, X\_test, y\_train, y\_test, alpha, l1ratio, bestRMSE,  bestAlpha, bestL1Ratio, scalerY):  model = ElasticNet(alpha=alpha, l1\_ratio=l1ratio)  # fit model  model.fit(X\_train, y\_train)  y\_pred\_scaled = model.predict(X\_test)  y\_pred = scalerY.inverse\_transform(np.array(y\_pred\_scaled).reshape(-1,1))  print("\n\*\*\*ElasticNet Regression Coefficients \*\* alpha=" + str(alpha)  + " l1ratio=" + str(l1ratio))  rmse = np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))  print(model.intercept\_)  print(model.coef\_)  try:  if(rmse < bestRMSE):  bestRMSE = rmse  bestAlpha = alpha  bestL1Ratio = l1ratio  print('Root Mean Squared Error:', rmse)  except:  print("rmse =" + str(rmse))  return bestRMSE, bestAlpha, bestL1Ratio  alphaValues = [0, 0.00001, 0.0001, 0.001, 0.01, 0.18]  l1ratioValues = [0, 0.25, 0.5, 0.75, 1]  bestAlpha = 0  bestL1Ratio = 0  for i in range(0, len(alphaValues)):  for j in range(0, len(l1ratioValues)):  bestRMSE, bestAlpha, bestL1Ratio = performElasticNetRegression(  X\_train\_scaled, X\_test\_scaled, y\_train\_scaled,  y\_test,  alphaValues[i], l1ratioValues[j], bestRMSE,  bestAlpha, bestL1Ratio, scalerY)  print("Best RMSE " + str(bestRMSE) + " Best alpha: " + str(bestAlpha)  + " " + "Best l1 ratio: " + str(bestL1Ratio)) |

## Equation Comparison

With all of the methods discussed we are using gradients to find the minimum loss with the minimum least squared error equation. To implement each of the machine learning algorithms, we search for the minimum loss by calculating the gradient of the loss function.

=

The prediction equation changes with the algorithm. The prediction equation includes an L2 penalty with ridge regression. The prediction equation includes an L1 penalty for lasso regression. The prediction equation includes both L1 and L2 penalties for elastic net regression.

Table : Regression Comparison

|  |  |  |
| --- | --- | --- |
|  | Prediction equation as |  |
| Gradient Descent | intercept+slope | Simple |
| Stochastic Gradient Descent | intercept+slope  (Training set is randomly selected) | Simple |
| Ridge Regression | intercept+slope + | Tends to do better when most variables are relevant. |
| Lasso Regression | intercept+slope + | Gets rid of useless variables. |
| Elastic Net Regression  (Is a hybrid of ridge and lasso) | intercept+slope++ | Good for many variables when don’t know which ones are relevant. |

## Support Vector Machines

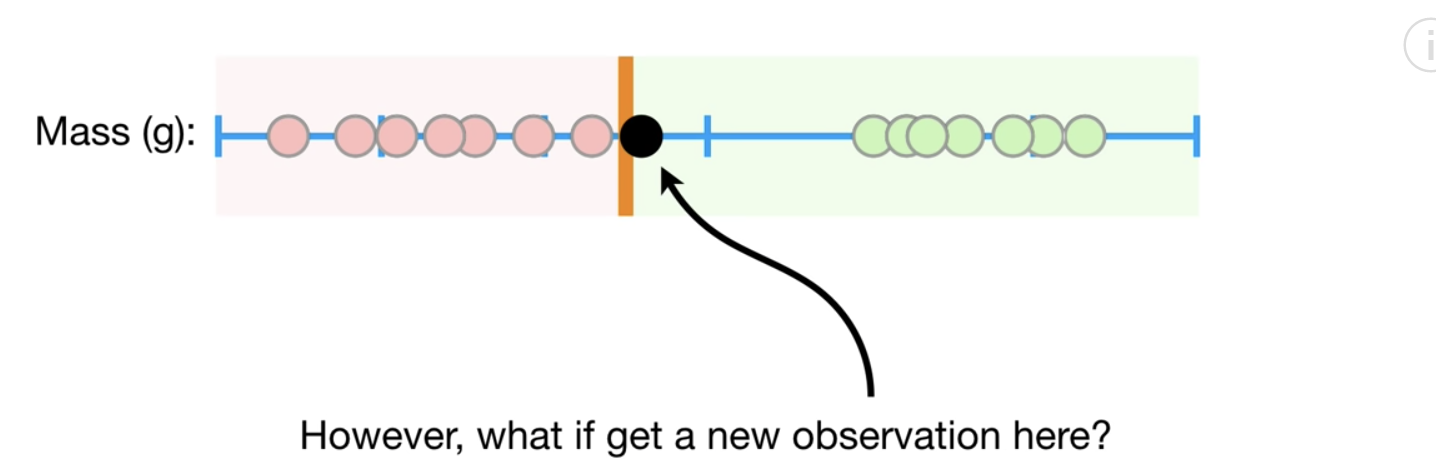
Support vector machines offer another way to apply machine learning for conducting linear regression and classification. Once again Josh Stormer has an excellent video to explain how support vector machines work.

<https://www.youtube.com/watch?v=efR1C6CvhmE>

Support vector machines (SVMs) are supervised algorithms for both classification and regression. Generally, an SVM algorithm tries to find the widest margin between classes to distinguish one from the other.

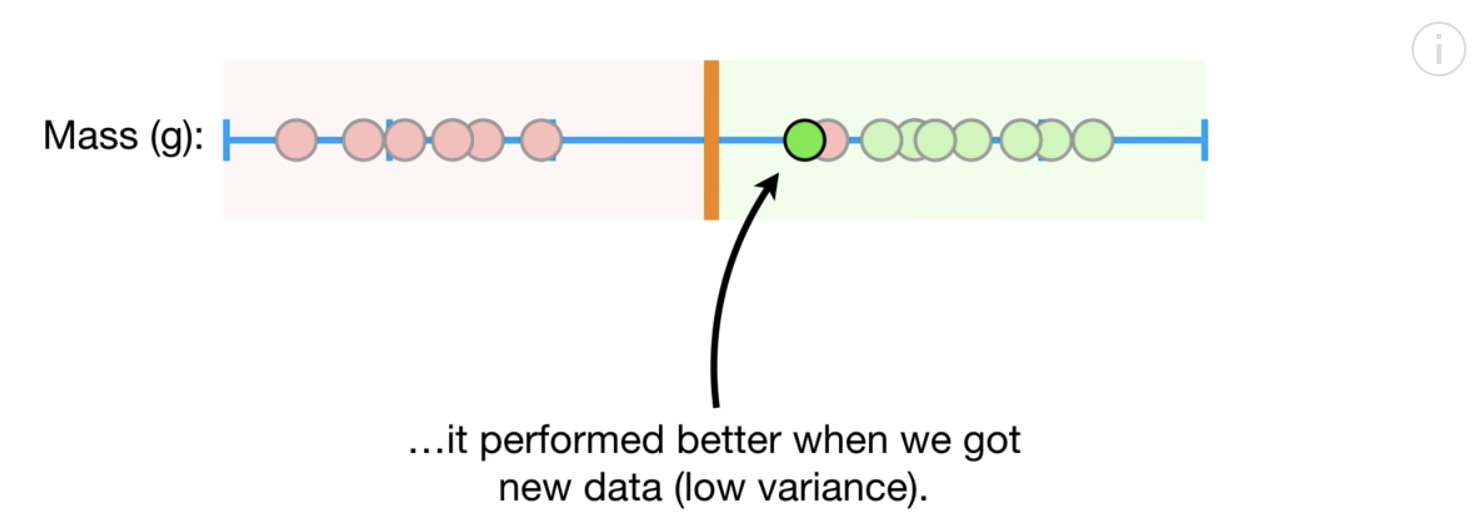
Josh Stormer presents an example of an SVM for classifying mice as either not obese or obese. Figure 4 illustrates how we can set a threshold to separate non-obese mice from obese mice.

Figure : Classifying Non-Obese Mice (Left) with Obese Mice (Right)



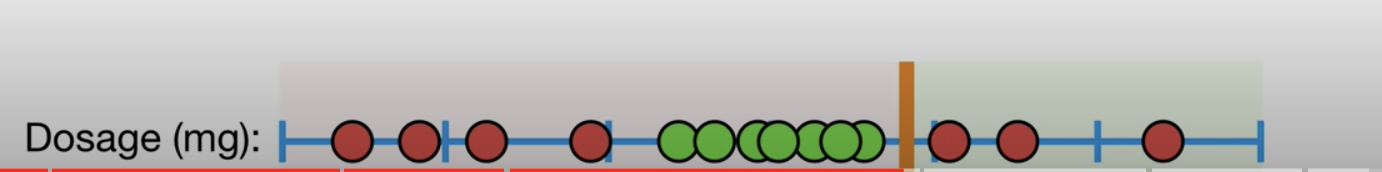
In order to optimize most predictions, we may place set threshold that allows some samples to be misclassified. Obviously, there is a trade-off but the benefit of high accuracy may be worth the occasional misclassification.

Figure : Allowing for Mis-classification



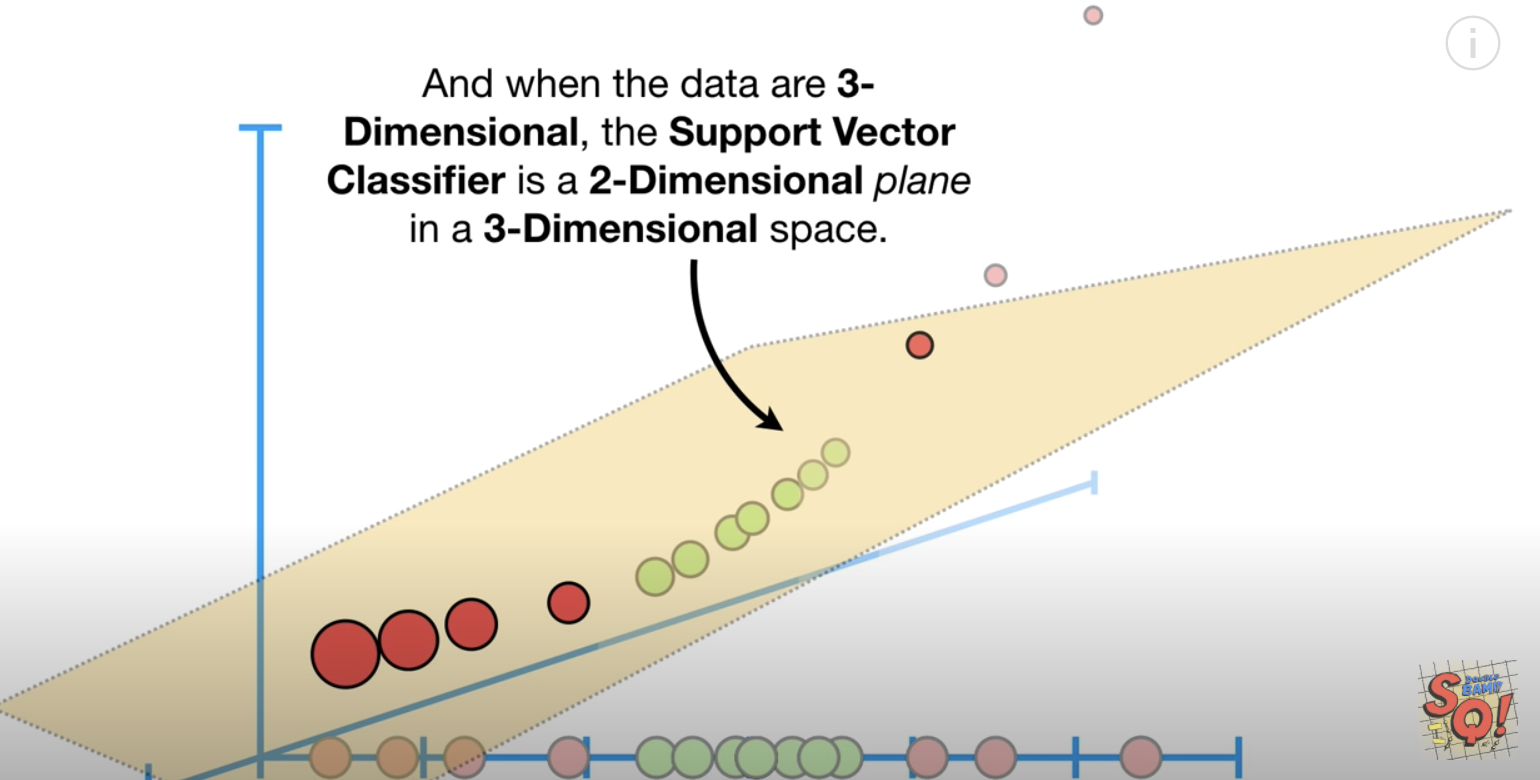
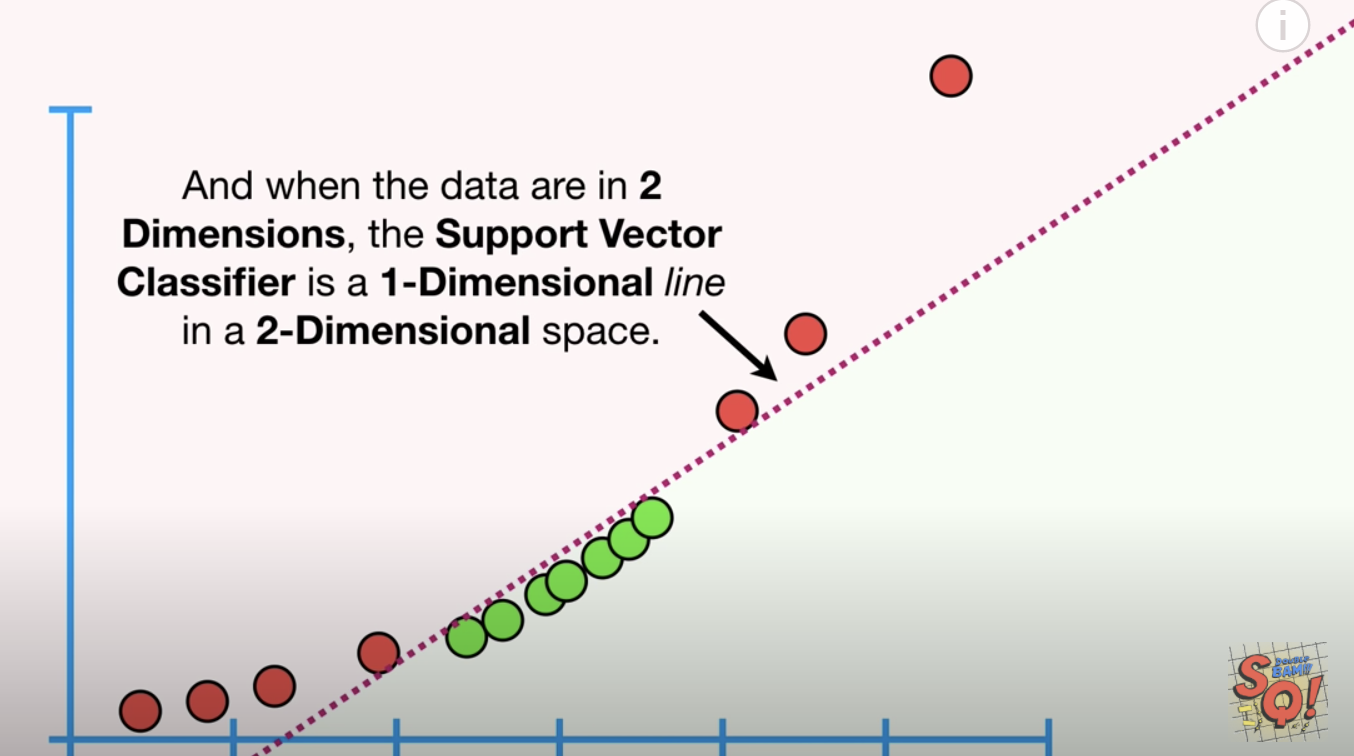
Josh presents another scenario in Figure 6 for vaccine dosage. The samples on the left do not receive enough of the vaccine to be cured. The samples on the right receive too much of the dosage to be cured. The samples in the middle receive the right amount of the dosage. A threshold in the same dimension does not help with classification.

Figure : Vaccine Dosage



Support vector classifiers allow us to create lines or planes in higher dimensions to enable better separation between classes (refer to Figure 7).

Figure : Enabling Classification with Higher Dimensions



We could Kernel could be sigmoid, linear, rbf (gaussian), poly with degree.

clf = svm.SVC(kernel='linear')

clf = svm.SVC(kernel='rbf')

clf = svm.SVC(kernel='sigmoid')

svclassifier = SVC(kernel='poly', degree=8)

Example : Basic Logistic Regression

In this example we will start the series by first perform basic logistic regression with the diabetes data set. Simple logistic regression can detect diabetes with accuracy score of 80%.

|  |
| --- |
| Accuracy: 0.8070866141732284  Confusion Matrix  Predicted 0 1  Actual  0 156 16  1 33 49 |

Here is the code:

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  PATH = "/Users/pm/Desktop/DayDocs/data/"  from sklearn.linear\_model import LogisticRegression  from sklearn import metrics  import numpy as np  # load the dataset  df = pd.read\_csv(PATH + 'diabetes.csv', sep=',')  # split into input (X) and output (y) variables  X = df[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI',  'DiabetesPedigreeFunction', 'Age']]  y = df[['Outcome']]  # Split into train and test data sets.  X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.33)  # Perform logistic regression.  logisticModel = LogisticRegression(fit\_intercept=True, random\_state = 0,  solver='liblinear')  logisticModel.fit(X\_train,y\_train)  y\_pred=logisticModel.predict(X\_test)  # Show model coefficients and intercept.  print("\nModel Coefficients: ")  print("\nIntercept: ")  print(logisticModel.intercept\_)  print(logisticModel.coef\_)  # Show confusion matrix and accuracy scores.  confusion\_matrix = pd.crosstab(np.array(y\_test['Outcome']), y\_pred,  rownames=['Actual'],  colnames=['Predicted'])  print('\nAccuracy: ',metrics.accuracy\_score(y\_test, y\_pred))  print("\nConfusion Matrix")  print(confusion\_matrix) |

Example : Support Vector Machines with a Linear Kernel

This example implements an SVM model with a linear kernel setting. The original logistic regression equation generates reasonable accuracy.

Accuracy: 0.7716535433070866

Confusion Matrix

Predicted 0 1

Actual

0 145 26

1 32 51

The linear kernel appears to be slightly better in this case. Running the code multiple times though obtains varying results.

Accuracy: 0.7755905511811023

To build this example, add this code to the end of Example 7.

|  |
| --- |
| # Import svm package  from sklearn import svm  # Create a svm Classifier using one of the following options:  # linear, polynomial, and radial  clf = svm.SVC(kernel='linear')  # Train the model using the training set.  clf.fit(X\_train, y\_train)  # Evaluate the model.  y\_pred = clf.predict(X\_test)  from sklearn import metrics  print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred)) |

Exercise (1 mark)

Try running the code in Example 8 with the ‘rbf’ option. How do the results compare with logistic regression?

|  |
| --- |
| Using RBF in this The accuracy of the RBF is lower in comparison to the logistic |

Exercise (1 mark)

Try running the code in Example 8 with the ‘sigmoid’ option. How do the results compare with logistic regression?

|  |
| --- |
| The accuracy dropped to even lower than RBF |

## Summary

The content that has been covered in this document shows several other ways to perform linear regression and classification. In addition to OLS and logistic regression, we have added gradient descent, ridge regression, lasso regression, elastic net regression and support vector machines to our tool kit. To know which algorithm performs best always cross validate and compare.

Exercise (2 marks)

Run the following code and show the model summary and RMSE. Indicate if any of the predictor variables are insignificant in a 5% confidence interval. It is fine if the intercept is not significant.

|  |
| --- |
| import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn import metrics  import statsmodels.api as sm  import numpy as np  PATH = "/Users/pm/Desktop/DayDocs/data/"  CSV\_DATA = "petrol\_consumption.csv"  dataset = pd.read\_csv(PATH + CSV\_DATA)  # Petrol\_Consumption  X = dataset[['Petrol\_tax','Average\_income', 'Population\_Driver\_licence(%)']]  # Adding an intercept \*\*\* This is requried \*\*\*. Don't forget this step.  # The intercept centers the error residuals around zero  # which helps to avoid over-fitting.  X\_withConst = sm.add\_constant(X)  y = dataset['Petrol\_Consumption'].values  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_withConst, y,  test\_size=0.2, random\_state=0)  def performLinearRegression(X\_train, X\_test, y\_train, y\_test):  model = sm.OLS(y\_train, X\_train).fit()  predictions = model.predict(X\_test) # make the predictions by the model  print(model.summary())  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test, predictions)))  return predictions  predictions = performLinearRegression(X\_train, X\_test, y\_train, y\_test) |

Show the model summary here:

|  |
| --- |
|  |

Show the RMSE here:

|  |
| --- |
|  |

Are any predictor variables insignificant? If so which ones are insignificant?

|  |
| --- |
| Based on the OLS summary since non are higher than 0.05 that means none of the predictors are insignificant |

Exercise (2 marks)

Build a linear regression model for petrol consumption using stochastic gradient descent. Big hint: add in the code from Example 3. How does the RMSE from SGD compare with the RMSE when using OLS? Explain why this may have happened.

|  |
| --- |
| Since SGD is sensitive to scaling we scaled the train and test data and used those values in the OLS and SGD regressions    The SGD gave a much lower rmse.  The possibility of it being lower is due to the fact that the set of data it grabbed could have all resulted in a lower RMSE giving us a lower RMSE in comparison to OLS |

Exercise (3 marks)

Using petrol consumption data from the previous exercise, add in the code to perform **ridge** regression. This is easy. Please see the example code for ridge regression. Make sure none of the SGD or scaling code exists in this file so you can get a clear view of how ridge regression compares with OLS. Also, remove the parameter random\_state=0 to prevent seeding so you can have truly random data. Grid search optimal values for alpha. Compare RMSE’s for your OLS and ridge regression models here:

|  |
| --- |
|  |
| Here can we see that the RMSE on avg with Ridge regression is much higher when we get rid of the scaling. It also has massive jumps as the alpha increases |

Exercise (3 marks)

Start with the code for OLS again from the previous exercise. Add in the code to perform **lasso** regression. This is easy. Please see the example code for lasso regression. Make sure none of the SGD or scaling code exists in this file so you can get a clear view of how ridge regression compares with OLS. Also, remove the parameter random\_state=0 to prevent seeding so you can have truly random data. Grid search optimal values for alpha. Compare RMSE’s for your OLS and your best performing lasso regression models here:

|  |
| --- |
| Here you can see that our best model is the alpho 0 its exactly the same maybe slightly higher near the last 4 digits. The RMSe is much higher but doesn’t jump as much especially with its alpha values also being higher. |

Exercise (6 marks)

Start with the code from the previous exercise. Add in the following code to perform **elastic net** regression:

|  |
| --- |
| from sklearn.linear\_model import ElasticNet  bestRMSE = 100000.03  def performElasticNetRegression(X\_train, X\_test, y\_train, y\_test, alpha, l1ratio, bestRMSE,  bestAlpha, bestL1Ratio):  model = ElasticNet(alpha=alpha, l1\_ratio=l1ratio)  # fit model  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test)  print("\n\*\*\*ElasticNet Regression Coefficients \*\* alpha=" + str(alpha)  + " l1ratio=" + str(l1ratio))  rmse = np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred))  print(model.intercept\_)  print(model.coef\_)  try:  if(rmse < bestRMSE):  bestRMSE = rmse  bestAlpha = alpha  bestL1Ratio = l1ratio  print('Root Mean Squared Error:', rmse)  except:  print("rmse =" + str(rmse))  return bestRMSE, bestAlpha, bestL1Ratio  alphaValues = [0, 0.00001, 0.0001, 0.001, 0.01, 0.18]  l1ratioValues = [0, 0.25, 0.5, 0.75, 1]  bestAlpha = 0  bestL1Ratio = 0  for i in range(0, len(alphaValues)):  for j in range(0, len(l1ratioValues)):  bestRMSE, bestAlpha, bestL1Ratio = performElasticNetRegression(  X\_train, X\_test, y\_train, y\_test,  alphaValues[i], l1ratioValues[j], bestRMSE,  bestAlpha, bestL1Ratio)  print("Best RMSE " + str(bestRMSE) + " Best alpha: " + str(bestAlpha)  + " " + "Best l1 ratio: " + str(bestL1Ratio)) |

Compare RMSE’s for your OLS and your best performing elastic net regression model here:

|  |
| --- |
| OLS RMSE value  Elastic net regression Best performing RMSE value  The best performing regression is of Alpha 0. The last |

Show a screenshot of your best elastic net model settings here:

|  |
| --- |
| The alpha is 0 and the l1ration is 0 |

What are the model coefficients of your best model? What does this imply?

|  |
| --- |
| The best are here in the screenshot. This indicates that the coeeficients here have minimal RSE when performing a regression. |

Exercise (5 marks)

Create a table. Show RMSE values, alpha values and where applicable l1\_ratio values for your best OLS, ridge, lasso and elastic net models. Show your table here:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Regressoin name | RMSE value | Alpha value | L1 ration | | OLS |  | 0 | None | | Ridge |  | 0 | None | | Lasso |  | 0 | None | | Elastic Net |  | 0 | 0 | |

Exercise (2 marks)

Match the following terms with a definition. Use each term only once to generate the optimal matches.

Gradient descent, Elastic net regression, Lasso regression, Ridge Regression, Support Vector Machines

1. Ridge regression\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Is good for data sets with many attributes and significant predictor variables are unknown.

1. Lasso Regression\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Is good for eliminating insignificant predictor variables.

1. Gradient\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Adds a penalty to eliminate bias.

1. SVM \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Uses higher dimensions for better separation between classes.

Exercise (3 marks)

Indicate if each of the following statements is either true or false.

\_\_true\_\_\_\_ Lasso regression adds an L1 penalty so insignificant variable coefficients effectively

vanish.

\_\_true\_\_\_\_ Ridge regression uses an L2 penalty to help minimize the loss function.

\_false\_\_\_\_\_ Support vector machines combine L1 and L2 penalties.

\_\_true \_\_\_\_ Elastic net regression combines L1 and L2 penalties.

\_\_true\_\_\_\_ The L1 penalty is

\_\_false\_\_\_\_ The L1 penalty is