Table of Contents

[Other Machine Learning Algorithms for Regression and Classification 1](#_Toc115673512)

[Least Squares Regression 1](#_Toc115673513)

[Gradient Descent 2](#_Toc115673514)

[Ridge Regression 4](#_Toc115673515)

[Lasso Regression 6](#_Toc115673516)

[Elastic Net Regression 7](#_Toc115673517)

[Equation Comparison 9](#_Toc115673518)

[Support Vector Machines 10](#_Toc115673519)

[Summary 14](#_Toc115673520)

# Other Machine Learning Algorithms for Regression and Classification

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

At the start of the course we discussed gradient descent as a way to perform linear and classification regressions. Today we will examine machine learning techniques for performing linear and classification regression without building neural networks. Why bother? Sometimes these techniques can offer better results.

**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.

## Least Squares Regression

The examples in this section examine different ways to perform linear regression with the wine quality data set. When examining several different algorithms to perform linear regression starting with a simple OLS model is a great place to start to identify significant predictor variables and a baseline target.

Example : Ordinary Least Squares Regression

This example shows how to perform ordinary least squares regression. The code is actually the same code that we used when we first covered multiple regression in COMP3948. Insignificant variables have been removed from the training data. The root mean square error after building the model with OLS is: 0.6259157889490549. This RMSE score will be our benchmark for other models. Here is the code;

|  |
| --- |
| 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,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('fixed acidity', 'volatile acidity', 'citric acid',  'residual sugar', 'chlorides', 'free sulfur dioxide',  'total sulfur dioxide', 'density', 'pH', 'sulphates',  'alcohol', 'quality'))  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  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) |

## Gradient Descent

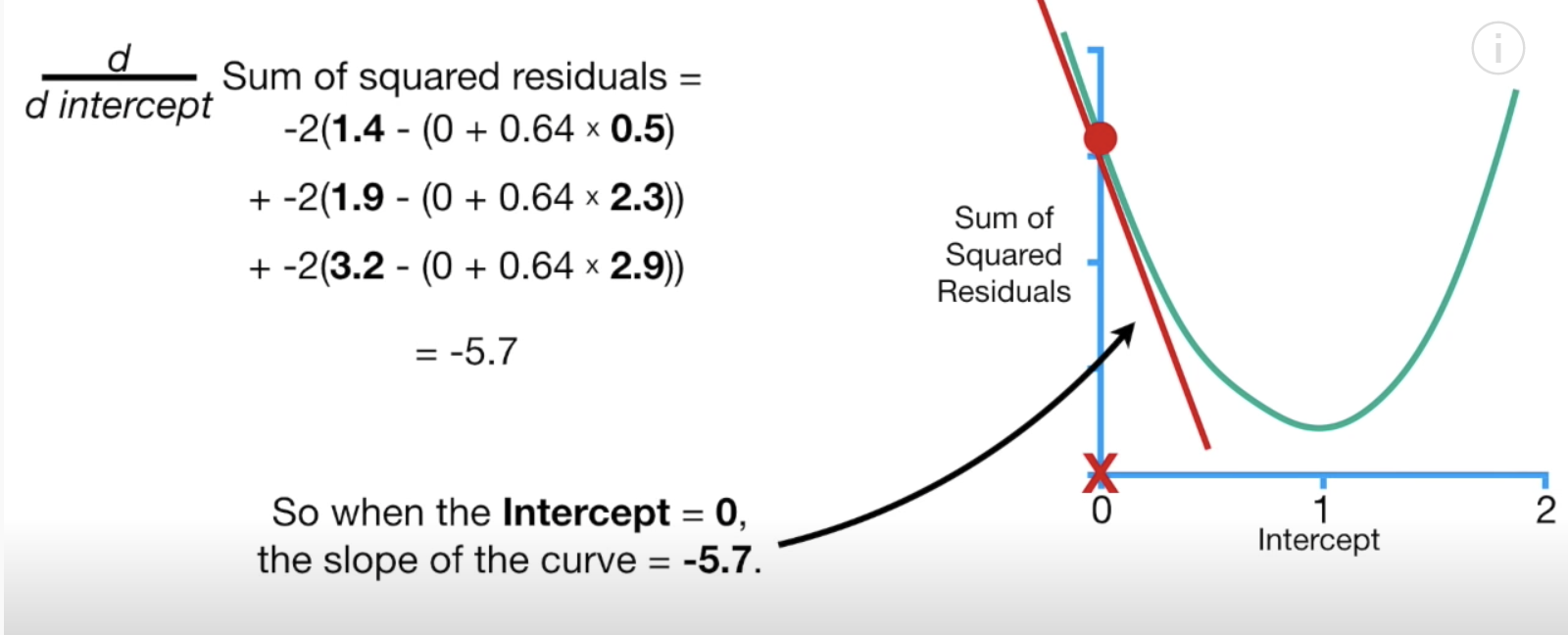
On the first day of COMP3948 we also discussed stochastic gradient descent. In case you need a review, you may want to watch Josh Stormer’s video where he explains how gradient descent works at <https://youtu.be/sDv4f4s2SB8>.

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 1.

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. Based on this finding we can say OLS performed better. The SGD model may have performed worse because it is using randomly selected data from the training set.

|  |
| --- |
| from sklearn.linear\_model import SGDRegressor  def performSGD(X\_train, X\_test, y\_train, y\_test, scalerY):  sgd = SGDRegressor(verbose=1)  sgd.fit(X\_train, y\_train)  print("\n\*\*\*SGD=")  predictions = sgd.predict(X\_test)  #print(predictions)  y\_test\_unscaled = scalerY.inverse\_transform(y\_test)  predictions\_unscaled = scalerY.inverse\_transform(predictions.reshape(-1, 1) )  #print(predictions\_unscaled)  print('Root Mean Squared Error:',  np.sqrt(metrics.mean\_squared\_error(y\_test\_unscaled,  predictions\_unscaled)))  from sklearn.preprocessing import MinMaxScaler  scalerX = MinMaxScaler()  scalerX.fit(X)  x2Scaled = scalerX.transform(X)  scalerY = MinMaxScaler()  reshapedY = y.reshape(-1,1)  scalerY.fit(reshapedY)  yScaled = scalerY.transform(reshapedY)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(x2Scaled, yScaled,  test\_size=0.2, random\_state=0)  performSGD(X\_train, X\_test, y\_train, y\_test, scalerY) |

## Ridge Regression

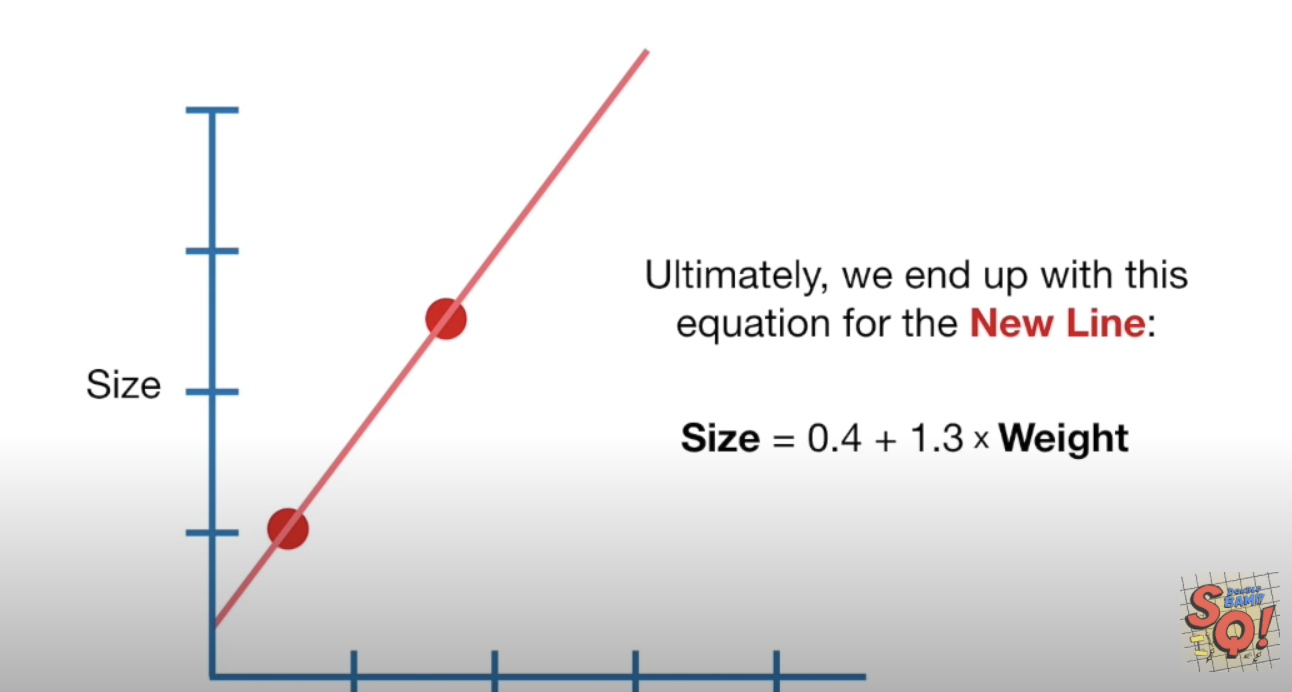
On the first day of COMP4948 we also discussed ridge regression. Josh Stormer’s video explains ridge regression nicely in detail in case you would like to review it

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

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:

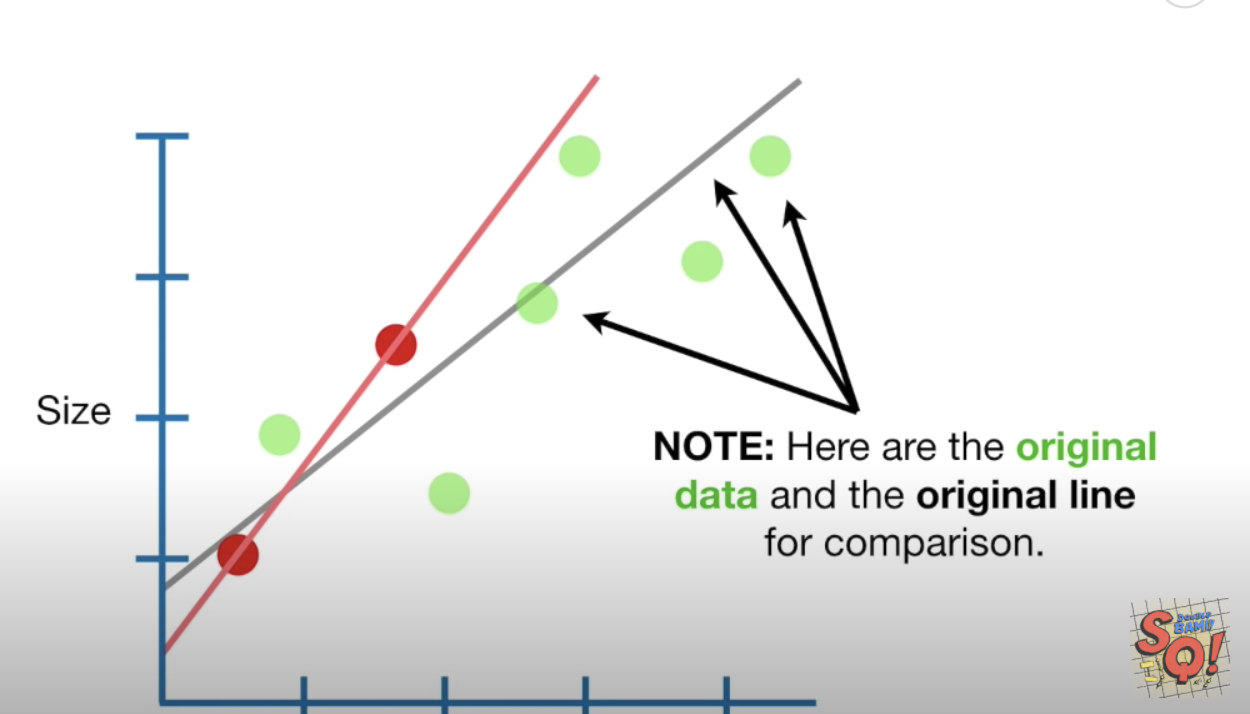
intercept + slope

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 of 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 1. 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 1.

|  |
| --- |
| from sklearn.linear\_model import Ridge  def ridge\_regression(X\_train, X\_test, y\_train, y\_test, alpha):  # Fit the model  ridgereg = Ridge(alpha=alpha, normalize=True)  ridgereg.fit(X\_train, y\_train)  y\_pred = 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\_)  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, X\_test, y\_train, y\_test,  alphaValues[i]) |

Exercise (1 mark)

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

|  |
| --- |
| Text  Description automatically generated |

Exercise (1 mark)

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

|  |
| --- |
| When used OLS the RMSE is  Root Mean Squared Error: 0.6259157889490532  \*\*\*Ridge Regression Coefficients \*\* alpha=0  Root Mean Squared Error: 0.625915788949053  They are almost the same. |

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?

|  |
| --- |
| When alpha equals 0 ridge regression has an intercept 3.01052 which is identical as the const in the OSL model summary. |

## 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 on to the end of any of the prior examples. 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=0.5  5.64659890539484  [-0. -0. -0. 0. 0.]  Root Mean Squared Error: 0.7584549718351333  \*\*\*Lasso Regression Coefficients \*\* alpha=1  5.64659890539484  [-0. -0. -0. 0. 0.]  Root Mean Squared Error: 0.7584549718351333 |

Here is the code:

|  |
| --- |
| 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 = "C:\\datasets\\" CSV\_DATA = "winequality.csv" dataset = pd.read\_csv(PATH + CSV\_DATA,  skiprows=1, # Don't include header row as part of data.  encoding = "ISO-8859-1", sep=',',  names=('fixed acidity', 'volatile acidity', 'citric acid',  'residual sugar', 'chlorides', 'free sulfur dioxide',  'total sulfur dioxide', 'density', 'pH', 'sulphates',  'alcohol', 'quality'))  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  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)   from sklearn.linear\_model import Ridge def ridge\_regression(X\_train, X\_test, y\_train, y\_test, alpha):  # Fit the model  ridgereg = Ridge(alpha=alpha)  ridgereg.fit(X\_train, y\_train)  y\_pred = 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\_)  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, X\_test, y\_train, y\_test,  alphaValues[i])  from sklearn.linear\_model import Lasso def performLassorRegression(X\_train, X\_test, y\_train, y\_test, alpha):  lassoreg = Lasso(alpha=alpha, max\_iter=15000)  lassoreg.fit(X\_train, y\_train)  y\_pred = lassoreg.predict(X\_test)  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, X\_test, y\_train, y\_test,  alphaValues[i]) |

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:

|  |
| --- |
| 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  X\_elastic = dataset[['fixed acidity', 'volatile acidity', 'citric acid',  'residual sugar', 'chlorides', 'free sulfur dioxide',  'total sulfur dioxide', 'density', 'pH', 'sulphates',  'alcohol']]  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_elastic, y,  test\_size=0.2, random\_state=0)  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)) |

## 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

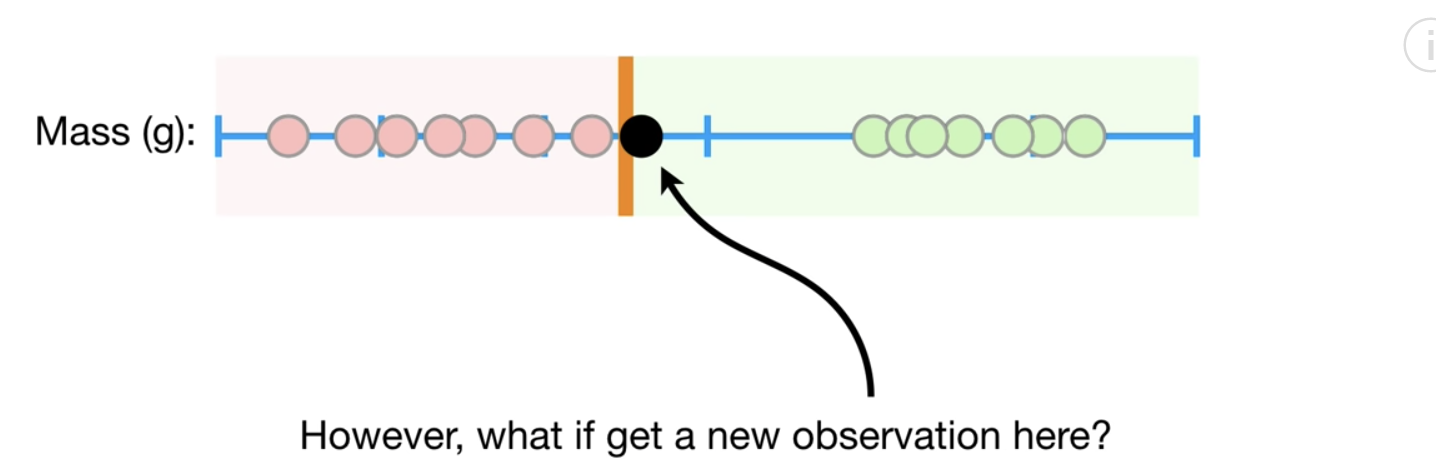
Support vector machines offer another way to apply machine learning for conducting linear regression and classification. Once again Josh Stormer has an excellent video to explain how support vector machines work.

<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.

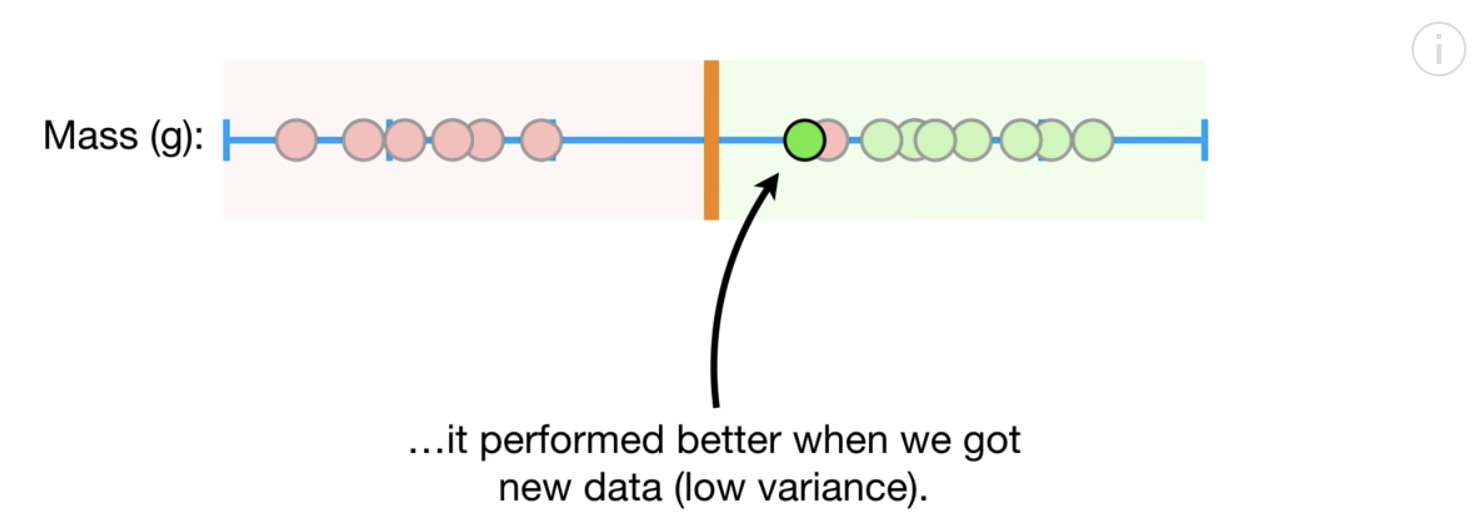
Josh Stormer presents an example of an SVM for classifying mice as either not obese or obese. Figure 4 illustrates how we can set a threshold to separate non-obese mice from obese mice.

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



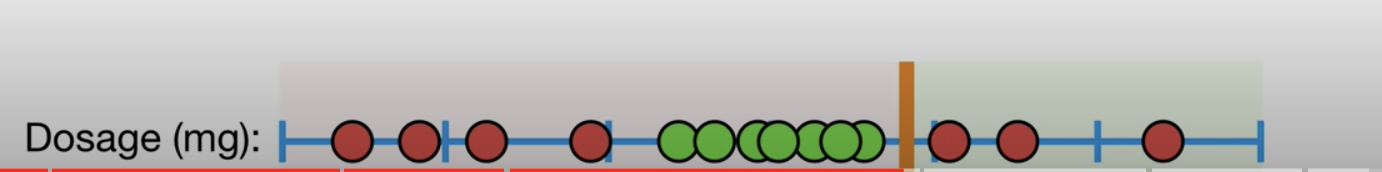
In order to optimize most predictions, we may place set threshold that allows some samples to be misclassified. Obviously, there is a trade-off but the benefit of high accuracy may be worth the occasional misclassification.

Figure : Allowing for Mis-classification



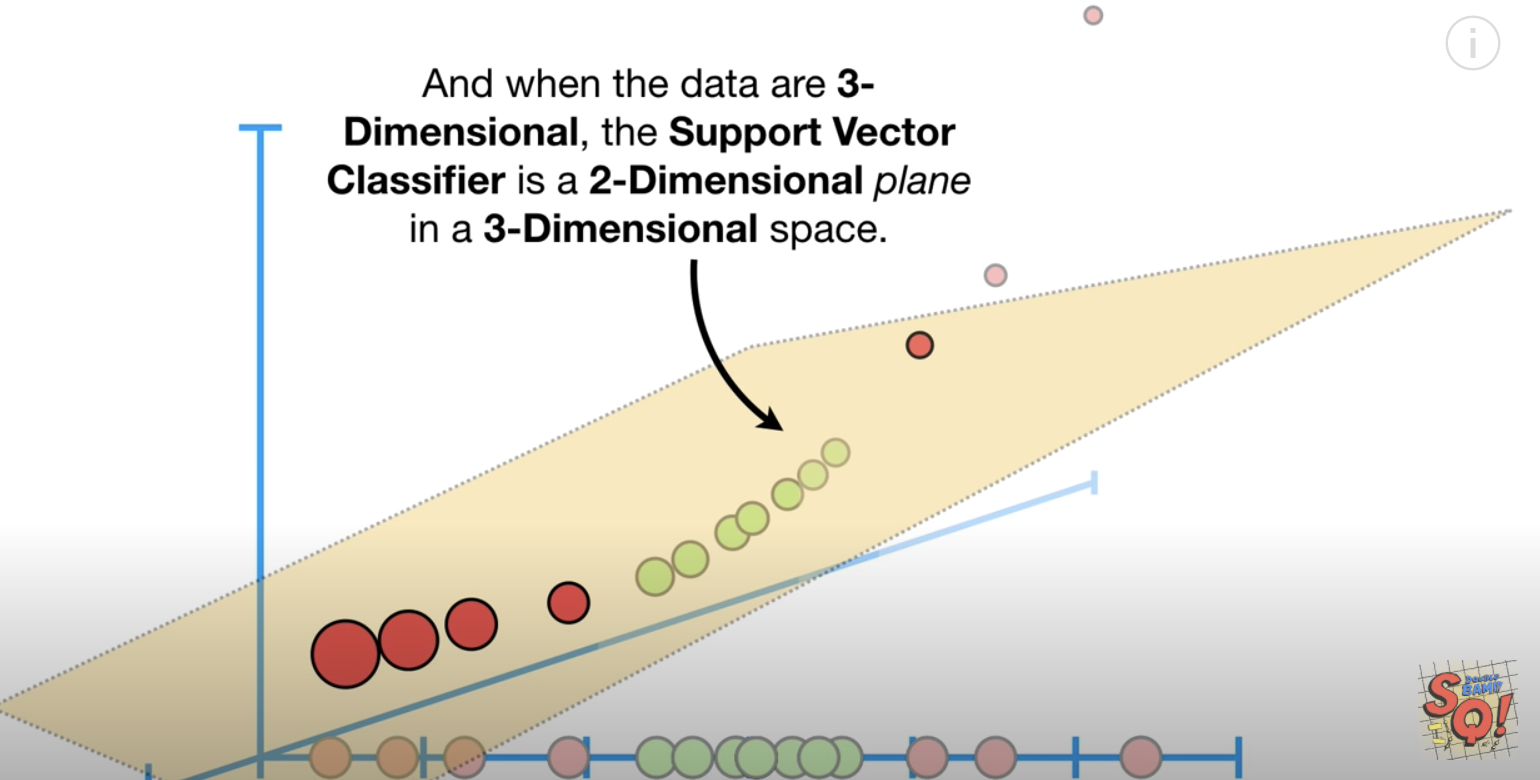
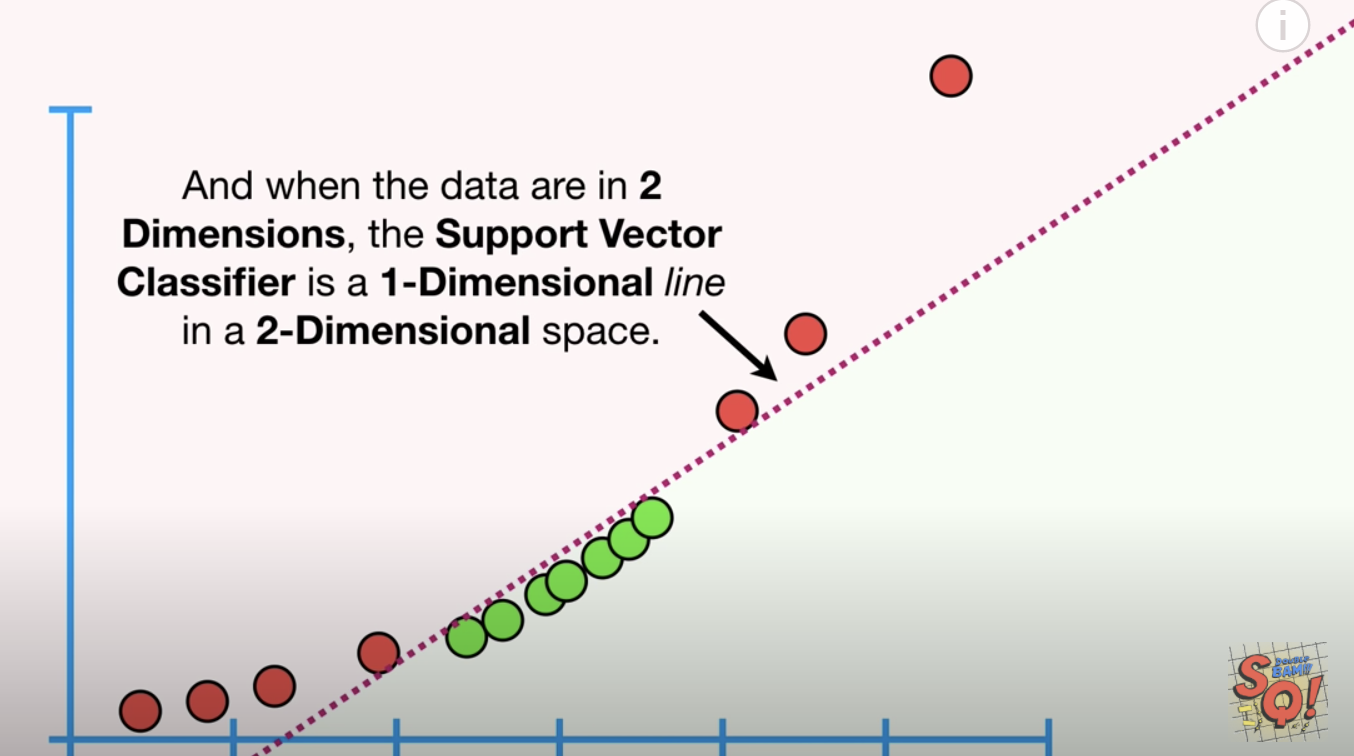
Josh presents another scenario in Figure 6 for vaccine dosage. The samples on the left do not receive enough of the vaccine to be cured. The samples on the right receive too much of the dosage to be cured. The samples in the middle receive the right amount of the dosage. A threshold in the same dimension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

|  |
| --- |
| # 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 7 with the ‘rbf’ option. How do the results compare with logistic regression?

|  |
| --- |
| With rbf option  Text  Description automatically generated  With regression option  Text  Description automatically generated  Accuracy is better when using rbf option |

Exercise (1 mark)

Try running the code in Example 7 with the ‘sigmoid’ option. How do the results compare with logistic regression?

|  |
| --- |
| Text  Description automatically generated  Using sigmoid is better accuracy than use of logistic regression but still using rbf is the best result. |

Example : Fitting an SVM classifier with a polynomial kernel

This example shows how to implement the SVM classifier with a polynomial kernel setting. The routine grid searches for the optimal polynomial order:

Accuracy: 0.6181102362204725 Num degrees: 0

Accuracy: 0.7401574803149606 Num degrees: 1

Accuracy: 0.7440944881889764 Num degrees: 2

Accuracy: 0.7440944881889764 Num degrees: 3

Accuracy: 0.7401574803149606 Num degrees: 4

Accuracy: 0.7322834645669292 Num degrees: 5

In this case, the results are quite volatile so the polynomial kernel may not offer the best fit. To build this example, add this code to the end of Example 6.

|  |
| --- |
| # Suppress the data convergence warning.  import warnings  from sklearn.exceptions import DataConversionWarning  warnings.filterwarnings(action='ignore', category=DataConversionWarning)  from sklearn import svm  def buildSVMmodel(degree):  # Create a svm Classifier using one of the following options:  # linear, polynomial, and radial  clf = svm.SVC(kernel='poly', degree=degree)  # 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  accuracy = metrics.accuracy\_score(y\_test, y\_pred)  print("Accuracy: " + str(accuracy) + " Num degrees: " + str(degree))  degrees = [0, 1,2,3,4,5]  for i in range(0, len(degrees)):  buildSVMmodel(degrees[i]) |

## Summary

The content that has been covered in this document shows several other ways to perform linear regression and classification regression. 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. In the words of Dr. Donald Wedding, line the models up and let them have a street brawl.

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:

|  |
| --- |
| Text  Description automatically generated |

Show the RMSE here:

|  |
| --- |
| Root Mean Squared Error: 62.880234412269076 |

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

|  |
| --- |
| All of variable’s p value are the smaller than 0.05 so it looks like every variable is significant. |

Exercise (2 marks)

Build a linear regression model for petrol consumption using stochastic gradient descent. Big hint: add in the code from Example 2. How does the RMSE from SGD compare with the RMSE when using OLS? Explain why this may have happened.

|  |
| --- |
| \*\*OSL =  Root Mean Squared Error: 62.880234412269076  \*\*\*SGD=  Root Mean Squared Error: 81.26539988391927  RMSE is getting worse with SGD because SGD model may have performed worse because it is using randomly selected data from the training set. |

Exercise (3 marks)

Start with the code for OLS again from Example 2. 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:

|  |
| --- |
| \*\*OSL = Root Mean Squared Error: 0.13195170801442288  \*\*\*Ridge Regression Coefficients \*\* alpha=0  288.61194481858126  [-2.93087031e+01 -7.08395205e-02 1.43056503e+03]  Root Mean Squared Error: 62.880234412276394  \*\*\*Ridge Regression Coefficients \*\* alpha=0.16  345.3234770957313  [-2.79282706e+01 -5.98484543e-02 1.23165414e+03]  Root Mean Squared Error: 60.76305901802673  \*\*\*Ridge Regression Coefficients \*\* alpha=0.17  348.14921884230057  [-2.78287536e+01 -5.92756939e-02 1.22114820e+03]  Root Mean Squared Error: 60.692091232739976  \*\*\*Ridge Regression Coefficients \*\* alpha=0.18  350.90825722563005  [-2.77284341e+01 -5.87139820e-02 1.21083003e+03]  Root Mean Squared Error: 60.62664201594916 |

Exercise (3 marks)

Start with the code for OLS again from Example 2. 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:

|  |
| --- |
| \*\*OSL =  Root Mean Squared Error: 62.880234412269076  \*\*\*Lasso Regression Coefficients \*\* alpha=1  495.1922008049145  [-3.29826305e+01 -6.77708844e-02 1.09798730e+03]  Root Mean Squared Error: 61.076267203776986 |

Exercise (6 marks)

Start with the code for OLS again from Example 2. 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)) |

Make sure none of the SGD or scaling code exists in this file so you can get a clear view of how elastic net 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 and the l1\_ratio.

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

|  |
| --- |
| OSL: Root Mean Squared Error: 0.12175984109482359  \*\*\*Lasso Regression Coefficients \*\* alpha=0  [0.39242321]  [-0.23921041 -0.25922128 0.53484621]  Root Mean Squared Error: 0.09980773903979615  \*\*\*Lasso Regression Coefficients \*\* alpha=0.1  [0.34358131]  [-0. -0. 0.]  Root Mean Squared Error: 0.2178674460128872  \*\*\*Lasso Regression Coefficients \*\* alpha=0.5  [0.34358131]  [-0. -0. 0.]  Root Mean Squared Error: 0.2178674460128872  \*\*\*Lasso Regression Coefficients \*\* alpha=1  [0.34358131]  [-0. -0. 0.] |

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

|  |
| --- |
|  |

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

|  |
| --- |
| \*\*\*ElasticNet Regression Coefficients \*\* alpha=0.001 l1ratio=1  3.032044247341152  [ 0. -1.10987582 -0.81796756 -0.00237869 0.85866456 0.26884447]  Root Mean Squared Error: 0.6219120038940735  Coefficient is 0 for the first value so it is insignificant |

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:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  | alpha | L1\_ratio | RMSE | | OLS | - | - | 0.6373767069346159 | | Lasso | 0 | - | 0.6494030840114315 | | Ridge | 0.18 | - | 0.6257606461145467 | | Elastic | 0 | 1 | 0.6219120038940735 |   \*\*OLS : Root Mean Squared Error: 0.6373767069346159  \*\*\*SGD=  Root Mean Squared Error: 0.6791320898858916  \*\*\*Lasso Regression Coefficients \*\* alpha=0  3.019785722517928  [ 0. -1.06766773 -2.07443589 -0.002506 0.992349 0.27167645]  Root Mean Squared Error: 0.6494030840114315  \*\*\*Lasso Regression Coefficients \*\* alpha=0.1  3.1464399491446766  [ 0. -0. -0. -0.00282312 0. 0.25177034]  Root Mean Squared Error: 0.7145578054104719  \*\*\*Lasso Regression Coefficients \*\* alpha=0.5  5.825606918768603  [ 0. -0. -0. -0.00399791 0. 0. ]  Root Mean Squared Error: 0.8001912339442738  \*\*\*Lasso Regression Coefficients \*\* alpha=1  5.803799861376133  [ 0. -0. -0. -0.00353126 0. 0. ]  Root Mean Squared Error: 0.8008965513038041  \*\*\*Ridge Regression Coefficients \*\* alpha=0  3.0105263238850815  [-1.21046416 -1.7655295 -0.00217522 0.89491841 0.28268935]  Root Mean Squared Error: 0.6259157889490529  \*\*\*Ridge Regression Coefficients \*\* alpha=0.16  2.996341578456658  [-1.21016667 -1.64751147 -0.00217073 0.87871982 0.28405494]  Root Mean Squared Error: 0.6257754368384649  \*\*\*Ridge Regression Coefficients \*\* alpha=0.17  2.99550614318568  [-1.21013088 -1.64064747 -0.00217047 0.87777088 0.28413488]  Root Mean Squared Error: 0.6257679740145431  \*\*\*Ridge Regression Coefficients \*\* alpha=0.18  2.9946762878436846  [-1.21009323 -1.63383932 -0.00217022 0.87682886 0.28421422]  Root Mean Squared Error: 0.6257606461145467  Rigge regression with alpha 0.18 has the best RMSE |

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. Elastic net 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. Ridge Regression\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Adds a penalty to eliminate bias.

1. Support vector machines\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Uses higher dimensions for better separation between classes.

Exercise (3 marks)

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

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

vanish.

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

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

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

\_\_\_\_T\_\_ The L1 penalty is

\_\_\_F\_\_\_ The L1 penalty is