|  |
| --- |
| What is Regression Analysis?  Regression analysis is a form of predictive modelling technique which investigates the relationship between a **dependent**(target) and **independent variable (s)** (predictor). This technique is used for forecasting, time series modelling and finding the [causal effect relationship](https://www.analyticsvidhya.com/blog/2015/06/establish-causality-events/) between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.  Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.  [Regression_Line](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Regression_Line.png)    Why do we use Regression Analysis?  As mentioned above, regression analysis estimates the relationship between two or more variables. Let’s understand this with an easy example:  Let’s say, you want to estimate growth in sales of a company based on current economic conditions. You have the recent company data which indicates that the growth in sales is around two and a half times the growth in the economy. Using this insight, we can predict future sales of the company based on current & past information.  There are multiple benefits of using regression analysis. They are as follows:   1. It indicates the **significant relationships** between dependent variable and independent variable. 2. It indicates the **strength of impact** of multiple independent variables on a dependent variable.   Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.    How many types of regression techniques do we have?  There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics (number of independent variables, type of dependent variables and shape of regression line). We’ll discuss them in detail in the following sections.  [Regression_Type](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Regression_Type.png)  For the creative ones, you can even cook up new regressions, if you feel the need to use a combination of the parameters above, which people haven’t used before. But before you start that, let us understand the most commonly used regressions:    1. Linear Regression  It is one of the most widely known modeling technique. Linear regression is usually among the first few topics which people pick while learning predictive modeling. In this technique, the dependent variable is continuous, independent variable(s) can be [continuous or discrete](https://en.wikipedia.org/wiki/Continuous_and_discrete_variables), and nature of regression line is linear.  Linear Regression establishes a relationship between **dependent variable (Y)** and one or more **independent variables (X)** using a **best fit straight line** (also known as regression line).  It is represented by an equation **Y=a+b\*X + e**, where a is intercept, b is slope of the line and e is error term. This equation can be used to predict the value of target variable based on given predictor variable(s).  [Linear_Regression](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Linear_Regression1.png)  The difference between simple linear regression and multiple linear regression is that, multiple linear regression has (>1) independent variables, whereas simple linear regression has only 1 independent variable.  Now, the question is “How do we obtain best fit line?”.  How to obtain best fit line (Value of a and b)?  This task can be easily accomplished by Least Square Method. It is the most common method used for fitting a regression line. It calculates the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line. Because the deviations are first squared, when added, there is no cancelling out between positive and negative values.  [Least_Square](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Least_Square.png)  [reg_error](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/reg_error.gif)  We can evaluate the model performance using the metric **R-square**. To know more details about these metrics, you can read: Model Performance metrics [Part 1](https://www.analyticsvidhya.com/blog/2015/01/model-performance-metrics-classification/), [Part 2](https://www.analyticsvidhya.com/blog/2015/01/model-perform-part-2/) .  Important Points:   * There must be **linear relationship** between independent and dependent variables * Multiple regression suffers from **multicollinearity, autocorrelation, heteroskedasticity**. * Linear Regression is very sensitive to **Outliers**. It can terribly affect the regression line and eventually the forecasted values. * Multicollinearity can increase the variance of the coefficient estimates and make the estimates very sensitive to minor changes in the model. The result is that the coefficient estimates are unstable * In case of multiple independent variables, we can go with **forward selection**, **backward elimination** and **step wise approach** for selection of most significant independent variables.     2. Logistic Regression  Logistic regression is used to find the probability of event=Success and event=Failure. We should use logistic regression when the dependent variable is binary (0/ 1, True/ False, Yes/ No) in nature. Here the value of Y ranges from 0 to 1 and it can represented by following equation.  odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence  ln(odds) = ln(p/(1-p))  logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk  Above, p is the probability of presence of the characteristic of interest. A question that you should ask here is “why have we used log in the equation?”.  Since we are working here with a binomial distribution (dependent variable), we need to choose a link function which is best suited for this distribution. And, it is [**logit**](https://en.wikipedia.org/wiki/Logistic_function) function. In the equation above, the parameters are chosen to maximize the likelihood of observing the sample values rather than minimizing the sum of squared errors (like in ordinary regression).  [Logistic_Regression](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Logistic_Regression.png)  Important Points:   * It is widely used for **classification problems** * Logistic regression doesn’t require linear relationship between dependent and independent variables.  It can handle various types of relationships because it applies a non-linear log transformation to the predicted odds ratio * To avoid over fitting and under fitting, we should include all significant variables. A good approach to ensure this practice is to use a step wise method to estimate the logistic regression * It requires **large sample sizes** because maximum likelihood estimates are less powerful at low sample sizes than ordinary least square * The independent variables should not be correlated with each other i.e. **no multi collinearity**.  However, we have the options to include interaction effects of categorical variables in the analysis and in the model. * If the values of dependent variable is ordinal, then it is called as **Ordinal logistic regression** * If dependent variable is multi class then it is known as **Multinomial Logistic regression**.     3. Polynomial Regression  A regression equation is a polynomial regression equation if the power of independent variable is more than 1. The equation below represents a polynomial equation:  y=a+b\*x^2  In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.  [Polynomial](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Polynomial.png)  Important Points:   * While there might be a temptation to fit a higher degree polynomial to get lower error, this can result in over-fitting. Always plot the relationships to see the fit and focus on making sure that the curve fits the nature of the problem. Here is an example of how plotting can help:   [underfitting-overfitting](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/underfitting-overfitting.png)   * Especially look out for curve towards the ends and see whether those shapes and trends make sense. Higher polynomials can end up producing wierd results on extrapolation.     4. Stepwise Regression  This form of regression is used when we deal with multiple independent variables. In this technique, the selection of independent variables is done with the help of an automatic process, which involves *no* human intervention.  This feat is achieved by observing statistical values like R-square, t-stats and AIC metric to discern significant variables. Stepwise regression basically fits the regression model by adding/dropping co-variates one at a time based on a specified criterion. Some of the most commonly used Stepwise regression methods are listed below:   * Standard stepwise regression does two things. It adds and removes predictors as needed for each step. * Forward selection starts with most significant predictor in the model and adds variable for each step. * Backward elimination starts with all predictors in the model and removes the least significant variable for each step.   The aim of this modeling technique is to maximize the prediction power with minimum number of predictor variables. It is one of the method to handle[higher dimensionality](https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/) of data set.    5. Ridge Regression  Ridge Regression is a technique used when the data suffers from multicollinearity ( independent variables are highly correlated). In multicollinearity, even though the least squares estimates (OLS) are unbiased, their variances are large which deviates the observed value far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.  Above, we saw the equation for linear regression. Remember? It can be represented as:  y=a+ b\*x  This equation also has an error term. The complete equation becomes:  y=a+b\*x+e (error term),  [error term is the value needed to correct for a prediction error between the observed and predicted value]  => y=a+y= a+ b1x1+ b2x2+....+e, for multiple independent variables.  In a linear equation, prediction errors can be decomposed into two sub components. First is due to the **biased** and second is due to the **variance**. Prediction error can occur due to any one of these two or both components. Here, we’ll discuss about the error caused due to variance.  Ridge regression solves the multicollinearity problem through [shrinkage parameter](https://en.wikipedia.org/wiki/Shrinkage_estimator) λ (lambda). Look at the equation below.  [Ridge](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Ridge2.png)  In this equation, we have two components. First one is least square term and other one is lambda of the summation of β2 (beta- square) where β is the coefficient. This is added to least square term in order to shrink the parameter to have a very low variance.  Important Points:   * The assumptions of this regression is same as least squared regression except normality is not to be assumed * It shrinks the value of coefficients but doesn’t reache zero, which suggests no feature selection feature * This is a regularization method and uses [l2 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)).     6. Lasso Regression  Similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition, it is capable of reducing the variability and improving the accuracy of linear regression models.  Look at the equation below:  [Lasso](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Lasso.png)  Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. Larger the penalty applied, further the estimates get shrunk towards absolute zero. This results to variable selection out of given n variables.  Important Points:   * The assumptions of this regression is same as least squared regression except normality is not to be assumed * It shrinks coefficients to zero (exactly zero), which certainly helps in feature selection * This is a regularization method and uses [l1 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)) * If group of predictors are highly correlated, lasso picks only one of them and shrinks the others to zero     7. ElasticNet Regression  ElasticNet is hybrid of Lasso and Ridge Regression techniques. It is trained with L1 and L2 prior as regularizer. Elastic-net is useful when there are multiple features which are correlated. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.  [Elastic_Net](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Elastic_Net.png)  A practical advantage of trading-off between Lasso and Ridge is that, it allows Elastic-Net to inherit some of Ridge’s stability under rotation.  Important Points:   * It encourages group effect in case of highly correlated variables * There are no limitations on the number of selected variables * It can suffer with double shrinkage   Beyond these 7 most commonly used regression techniques, you can also look at other models like [Bayesian](https://en.wikipedia.org/wiki/Bayesian_linear_regression), [Ecological](https://en.wikipedia.org/wiki/Ecological_regression) and [Robust regression](https://en.wikipedia.org/wiki/Robust_regression).    How to select the right regression model?  Within multiple types of regression models, it is important to choose the best suited technique based on type of independent and dependent variables, dimensionality in the data and other essential characteristics of the data. Below are the key factors that you should practice to select the right regression model:   1. Data exploration is an inevitable part of building predictive model. It should be you first step before selecting the right model like identify the relationship and impact of variables 2. To compare the goodness of fit for different models, we can analyse different metrics like statistical significance of parameters, R-square, Adjusted r-square, AIC, BIC and error term. Another one is the [Mallow’s Cp](http://support.minitab.com/en-us/minitab/17/topic-library/modeling-statistics/regression-and-correlation/goodness-of-fit-statistics/what-is-mallows-cp/) criterion. This essentially checks for possible bias in your model, by comparing the model with all possible submodels (or a careful selection of them). 3. Cross-validation is the best way to evaluate models used for prediction. Here you divide your data set into two group (train and validate). A simple mean squared difference between the observed and predicted values give you a measure for the prediction accuracy. 4. If your data set has multiple confounding variables, you should not choose automatic model selection method because you do not want to put these in a model at the same time. 5. It’ll also depend on your objective. It can occur that a less powerful model is easy to implement as compared to a highly statistically significant model. 6. Regression regularization methods(Lasso, Ridge and ElasticNet) works well in case of high dimensionality and multicollinearity among the variables in the data set.   ---  Ridge and Lasso in Python: 1. Brief Overview Ridge and Lasso regression are powerful techniques generally used for creating parsimonious models in presence of a ‘large’ number of features. Here ‘large’ can typically mean either of two things:   1. Large enough to enhance the tendency of a model to overfit (as low as 10 variables might cause overfitting) 2. Large enough to cause computational challenges. With modern systems, this situation might arise in case of millions or billions of features   Though Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially. If you’ve heard of them before, you must know that they work by penalizing the magnitude of coefficients of features along with minimizing the error between predicted and actual observations. These are called ‘regularization’ techniques. The key difference is in how they assign penalty to the coefficients:   1. **Ridge Regression:**    * Performs L2 regularization, i.e. adds penalty equivalent to **square of the magnitude** of coefficients    * Minimization objective = LS Obj + α \* (sum of square of coefficients) 2. **Lasso Regression:**    * Performs L1 regularization, i.e. adds penalty equivalent to **absolute value of the magnitude** of coefficients    * Minimization objective = LS Obj + α \* (sum of absolute value of coefficients)   Note that here ‘LS Obj’ refers to ‘least squares objective’, i.e. the linear regression objective without regularization.  If terms like ‘penalty’ and ‘regularization’ seem very unfamiliar to you, don’t worry we’ll talk about these in more detail through the course of this article. Before digging further into how they work, lets try to get some intuition into why penalizing the magnitude of coefficients should work in the first place.   2. Why Penalize the Magnitude of Coefficients? Lets try to understand the impact of model complexity on the magnitude of coefficients. As an example, I have simulated a **sine curve** (between 60° and 300°) and added some random noise using the following code:  #Importing libraries. The same will be used throughout the article.  import numpy as np  import pandas as pd  import random  import matplotlib.pyplot as plt  %matplotlib inline  from matplotlib.pylab import rcParams  rcParams['figure.figsize'] = 12, 10  #Define input array with angles from 60deg to 300deg converted to radians  x = np.array([i\*np.pi/180 for i in range(60,300,4)])  np.random.seed(10) #Setting seed for reproducability  y = np.sin(x) + np.random.normal(0,0.15,len(x))  data = pd.DataFrame(np.column\_stack([x,y]),columns=['x','y'])  plt.plot(data['x'],data['y'],'.')  The input-output looks like: [1.sine curve](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/1.sine-curve.png)  This resembles a sine curve but not exactly because of the noise. We’ll use this as an example to test different scenarios in this article. Lets try to estimate the sine function using **polynomial regression** with powers of x form 1 to 15. Lets add a column for each power upto 15 in our dataframe. This can be accomplished using the following code:  for i in range(2,16): #power of 1 is already there  colname = 'x\_%d'%i #new var will be x\_power  data[colname] = data['x']\*\*i  print data.head()  The dataframe looks like: [1.2 15 powers](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/1.2-15-powers.png)  Now that we have all the 15 powers, lets make 15 different linear regression models with each model containing variables with powers of x from 1 to the particular model number. For example, the feature set of model 8 will be – {x, x\_2, x\_3, … ,x\_8}.  First, we’ll define a generic function which takes in the required maximum power of x as an input and returns a list containing – [ model RSS, intercept, coef\_x, coef\_x2, … upto entered power ]. Here RSS refers to ‘Residual Sum of Squares’ which is nothing but the sum of square of errors between the predicted and actual values in the training data set. The python code defining the function is:  #Import Linear Regression model from scikit-learn.  from sklearn.linear\_model import LinearRegression  def linear\_regression(data, power, models\_to\_plot):  #initialize predictors:  predictors=['x']  if power>=2:  predictors.extend(['x\_%d'%i for i in range(2,power+1)])    #Fit the model  linreg = LinearRegression(normalize=True)  linreg.fit(data[predictors],data['y'])  y\_pred = linreg.predict(data[predictors])    #Check if a plot is to be made for the entered power  if power in models\_to\_plot:  plt.subplot(models\_to\_plot[power])  plt.tight\_layout()  plt.plot(data['x'],y\_pred)  plt.plot(data['x'],data['y'],'.')  plt.title('Plot for power: %d'%power)    #Return the result in pre-defined format  rss = sum((y\_pred-data['y'])\*\*2)  ret = [rss]  ret.extend([linreg.intercept\_])  ret.extend(linreg.coef\_)  return ret  Note that this function will not plot the model fit for all the powers but will return the RSS and coefficients for all the models.  Now, we can make all 15 models and compare the results. For ease of analysis, we’ll store all the results in a Pandas dataframe and plot 6 models to get an idea of the trend. Consider the following code:  #Initialize a dataframe to store the results:  col = ['rss','intercept'] + ['coef\_x\_%d'%i for i in range(1,16)]  ind = ['model\_pow\_%d'%i for i in range(1,16)]  coef\_matrix\_simple = pd.DataFrame(index=ind, columns=col)  #Define the powers for which a plot is required:  models\_to\_plot = {1:231,3:232,6:233,9:234,12:235,15:236}  #Iterate through all powers and assimilate results  for i in range(1,16):  coef\_matrix\_simple.iloc[i-1,0:i+2] = linear\_regression(data, power=i, models\_to\_plot=models\_to\_plot)    We would expect the models with increasing complexity to better fit the data and result in lower RSS values. This can be verified by looking at the plots generated for 6 models:    [2. lin reg o:p](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/2.-lin-reg-op.png)  This clearly aligns with our initial understanding. As the model complexity increases, the models tends to fit even smaller deviations in the training data set. Though this leads to overfitting, lets keep this issue aside for some time and come to our main objective, i.e. the impact on the magnitude of coefficients. This can be analysed by looking at the data frame created above.  Python Code:  #Set the display format to be scientific for ease of analysis  pd.options.display.float\_format = '{:,.2g}'.format  coef\_matrix\_simple  The output looks like: [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/3-linear_output_modIFIED-1024x486.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/3-linear_output_modIFIED.png)  It is clearly evident that the **size of coefficients increase exponentially with increase in model complexity**. I hope this gives some intuition into why putting a constraint on the magnitude of coefficients can be a good idea to reduce model complexity.  Lets try to understand this even better.  What does a large coefficient signify? It means that we’re putting a lot of emphasis on that feature, i.e. the particular feature is a good predictor for the outcome. When it becomes too large, the algorithm starts modelling intricate relations to estimate the output and ends up overfitting to the particular training data.  I hope the concept is clear. I’ll be happy to discuss further in comments if needed. Now, lets understand ridge and lasso regression in detail and see how well they work for the same problem.   3. Ridge Regression As mentioned before, ridge regression performs ‘**L2 regularization**‘, i.e. it adds a factor of sum of squares of coefficients in the optimization objective. Thus, ridge regression optimizes the following: Objective = RSS + α \* (sum of square of coefficients) Here, α (alpha) is the parameter which balances the amount of emphasis given to minimizing RSS vs minimizing sum of square of coefficients. α can take various values:   1. **α = 0:**    * The objective becomes same as simple linear regression.    * We’ll get the same coefficients as simple linear regression. 2. **α = ∞:**    * The coefficients will be zero. Why? Because of infinite weightage on square of coefficients, anything less than zero will make the objective infinite. 3. **0 < α < ∞:**    * The magnitude of α will decide the weightage given to different parts of objective.    * The coefficients will be somewhere between 0 and ones for simple linear regression.   I hope this gives some sense on how α would impact the magnitude of coefficients. One thing is for sure that any non-zero value would give values less than that of simple linear regression. By how much? We’ll find out soon. Leaving the mathematical details for later, lets see ridge regression in action on the same problem as above.  First, lets define a generic function for ridge regression similar to the one defined for simple linear regression. The Python code is:  from sklearn.linear\_model import Ridge  def ridge\_regression(data, predictors, alpha, models\_to\_plot={}):  #Fit the model  ridgereg = **Ridge(alpha=alpha,normalize=True)**  ridgereg.fit(data[predictors],data['y'])  y\_pred = ridgereg.predict(data[predictors])    #Check if a plot is to be made for the entered alpha  if alpha in models\_to\_plot:  plt.subplot(models\_to\_plot[alpha])  plt.tight\_layout()  plt.plot(data['x'],y\_pred)  plt.plot(data['x'],data['y'],'.')  plt.title('Plot for alpha: %.3g'%alpha)    #Return the result in pre-defined format  rss = sum((y\_pred-data['y'])\*\*2)  ret = [rss]  ret.extend([ridgereg.intercept\_])  ret.extend(ridgereg.coef\_)  return ret  Note the ‘Ridge’ function used here. It takes ‘alpha’ as a parameter on initialization. Also, keep in mind that normalizing the inputs is generally a good idea in every type of regression and should be used in case of ridge regression as well.  Now, lets analyze the result of Ridge regression for 10 different values of α ranging from 1e-15 to 20. These values have been chosen so that we can easily analyze the trend with change in values of α. These would however differ from case to case.  Note that each of these 10 models will contain all the 15 variables and only the value of alpha would differ. This is different from the simple linear regression case where each model had a subset of features.  Python Code:  #Initialize predictors to be set of 15 powers of x  predictors=['x']  predictors.extend(['x\_%d'%i for i in range(2,16)])  #Set the different values of alpha to be tested  alpha\_ridge = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20]  #Initialize the dataframe for storing coefficients.  col = ['rss','intercept'] + ['coef\_x\_%d'%i for i in range(1,16)]  ind = ['alpha\_%.2g'%alpha\_ridge[i] for i in range(0,10)]  coef\_matrix\_ridge = pd.DataFrame(index=ind, columns=col)  models\_to\_plot = {1e-15:231, 1e-10:232, 1e-4:233, 1e-3:234, 1e-2:235, 5:236}  for i in range(10):  coef\_matrix\_ridge.iloc[i,] = ridge\_regression(data, predictors, alpha\_ridge[i], models\_to\_plot)  This would generate the following plot: [4. ridge output](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/4.-ridge-output.png)  Here we can clearly observe that **as the value of alpha increases, the model complexity reduces**. Though higher values of alpha reduce overfitting, significantly high values can cause underfitting as well (eg. alpha = 5). Thus alpha should be chosen wisely. A widely accepted technique is cross-validation, i.e. the value of alpha is iterated over a range of values and the one giving higher cross-validation score is chosen.  Lets have a look at the value of coefficients in the above models:  Python Code:  #Set the display format to be scientific for ease of analysis  pd.options.display.float\_format = '{:,.2g}'.format  coef\_matrix\_ridge  The table looks like: [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/5.-ridge-table_modified-1024x334.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/5.-ridge-table_modified.png)  This straight away gives us the following inferences:   1. The RSS increases with increase in alpha, this model complexity reduces 2. An alpha as small as 1e-15 gives us significant reduction in magnitude of coefficients. How? Compare the coefficients in the first row of this table to the last row of simple linear regression table. 3. High alpha values can lead to significant underfitting. Note the rapid increase in RSS for values of alpha greater than 1 4. Though the coefficients are **very very small**, they are **NOT zero**.   The first 3 are very intuitive. But #4 is also a crucial observation. Let’s reconfirm the same by determining the number of zeros in each row of the coefficients data set:  Python Code:  coef\_matrix\_ridge.apply(lambda x: sum(x.values==0),axis=1)  Output: [6. ridge zeros](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/6.-ridge-zeros.png)  This confirms that all the 15 coefficients are greater than zero in magnitude (can be +ve or -ve). Remember this observation and have a look again until its clear. This will play an important role in later while comparing ridge with lasso regression.   4. Lasso Regression LASSO stands for Least Absolute Shrinkage and Selection Operator. I know it doesn’t give much of an idea but there are 2 key words here – ‘absolute‘ and ‘selection‘.  Lets consider the former first and worry about the latter later.  Lasso regression performs **L1 regularization**, i.e. it adds a factor of sum of absolute value of coefficients in the optimization objective. Thus, lasso regression optimizes the following: Objective = RSS + α \* (sum of absolute value of coefficients) Here, α (alpha) works similar to that of ridge and provides a trade-off between balancing RSS and magnitude of coefficients. Like that of ridge, α can take various values. Lets iterate it here briefly:   1. α = 0: Same coefficients as simple linear regression 2. α = ∞: All coefficients zero (same logic as before) 3. 0 < α < ∞: coefficients between 0 and that of simple linear regression   Yes its appearing to be very similar to Ridge till now. But just hang on with me and you’ll know the difference by the time we finish. Like before, lets run lasso regression on the same problem as above. First we’ll define a generic function:  from sklearn.linear\_model import Lasso  def lasso\_regression(data, predictors, alpha, models\_to\_plot={}):  #Fit the model  lassoreg = **Lasso(alpha=alpha,normalize=True, max\_iter=1e5)**  lassoreg.fit(data[predictors],data['y'])  y\_pred = lassoreg.predict(data[predictors])    #Check if a plot is to be made for the entered alpha  if alpha in models\_to\_plot:  plt.subplot(models\_to\_plot[alpha])  plt.tight\_layout()  plt.plot(data['x'],y\_pred)  plt.plot(data['x'],data['y'],'.')  plt.title('Plot for alpha: %.3g'%alpha)    #Return the result in pre-defined format  rss = sum((y\_pred-data['y'])\*\*2)  ret = [rss]  ret.extend([lassoreg.intercept\_])  ret.extend(lassoreg.coef\_)  return ret  Notice the additional parameters defined in Lasso function – ‘max\_iter‘. This is the maximum number of iterations for which we want the model to run if it doesn’t converge before. This exists for Ridge as as well but setting this to a higher than default value was required in this case. Why? I’ll come to this in next section, just keep it in the back of the envelope.  Lets check the output for 10 different values of alpha using the following code:  #Initialize predictors to all 15 powers of x  predictors=['x']  predictors.extend(['x\_%d'%i for i in range(2,16)])  #Define the alpha values to test  alpha\_lasso = [1e-15, 1e-10, 1e-8, 1e-5,1e-4, 1e-3,1e-2, 1, 5, 10]  #Initialize the dataframe to store coefficients  col = ['rss','intercept'] + ['coef\_x\_%d'%i for i in range(1,16)]  ind = ['alpha\_%.2g'%alpha\_lasso[i] for i in range(0,10)]  coef\_matrix\_lasso = pd.DataFrame(index=ind, columns=col)  #Define the models to plot  models\_to\_plot = {1e-10:231, 1e-5:232,1e-4:233, 1e-3:234, 1e-2:235, 1:236}  #Iterate over the 10 alpha values:  for i in range(10):  coef\_matrix\_lasso.iloc[i,] = lasso\_regression(data, predictors, alpha\_lasso[i], models\_to\_plot)  This gives us the following plots: [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/7.-lasso-output1-1024x833.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/7.-lasso-output1.png)  This again tells us that the model complexity decreases with increase in the values of alpha. But notice the straight line at alpha=1. Appears a bit strange to me. Let’s explore this further by looking at the coefficients:  [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/8.-lasso-table_modified-1024x334.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/8.-lasso-table_modified.png)  Apart from the expected inference of higher RSS for higher alphas, we can see the following:   1. For the same values of alpha, the coefficients of lasso regression are much smaller as compared to that of ridge regression (compare row 1 of the 2 tables). 2. For the same alpha, lasso has higher RSS (poorer fit) as compared to ridge regression 3. Many of the coefficients are zero even for very small values of alpha   Inferences #1,2 might not generalize always but will hold for many cases. The real difference from ridge is coming out in the last inference. Lets check the number of coefficients which are zero in each model using following code:  coef\_matrix\_lasso.apply(lambda x: sum(x.values==0),axis=1)  Output: [9. lasso zeros](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/9.-lasso-zeros.png)  We can observe that **even for a small value of alpha, a significant number of coefficients are zero**. This also explains the horizontal line fit for alpha=1 in the lasso plots, its just a baseline model! This phenomenon of most of the coefficients being zero is called ‘**sparsity**‘. Although lasso performs feature selection, this level of sparsity is achieved in special cases only which we’ll discuss towards the end.  This has some really interesting implications on the use cases of lasso regression as compared to that of ridge regression. But before coming to the final comparison, lets take a **bird’s eye view** of the mathematics behind why coefficients are zero in case of lasso but not ridge.   5. Sneak Peak into Statistics (Optional) I personally love statistics but many of you might not. That’s why I have specifically marked this section as ‘**OPTIONAL**‘. If you feel you can handle the algorithms without going into the maths behind them, I totally respect the decision and you can feel free to skip this section.  But I personally feel that getting some elementary understanding of how the thing works can be helpful in the long run.  As promised, I’ll keep it to a bird’s eye view. If you wish to get into the details, I recommend taking a good statistics textbook. One of my favorites is the [Elements of Statistical Learning](http://statweb.stanford.edu/~tibs/ElemStatLearn/). The best part about this is that it has been made available for free by the authors.  Let’s start by reviewing the basic structure of data in a regression problem.  [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/fig31-1024x838.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/fig31.png)  In this infographic, you can see there are 4 data elements:   1. **X**: the matrix of input features (nrow: N, ncol: M+1) 2. **Y**: the actual outcome variable (length:N) 3. **Yhat**: these are predicted values of Y (length:N) 4. **W**: the weights or the coefficients (length: M+1)   Here, N is the total number of data points available and M is the total number of features. X has M+1 columns because of M features and 1 intercept.  The predicted outcome for any data point i is:  [eq1](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq1.png)  It is simply the weighted sum of each data point with coefficients as the weights. This prediction is achieved by finding the optimum value of weights based on certain criteria, which depends on the type of regression algorithm being used. Lets consider all 3 cases:   **1. Simple Linear Regression** The objective function (also called as the cost) to be minimized is just the RSS (Residual Sum of Squares), i.e. the sum of squared errors of the predicted outcome as compared to the actual outcome. This can be depicted mathematically as:  [eq2](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq2.png)  In order to minimize this cost, we generally use a ‘gradient descent’ algorithm. I’ll not go into the details right now but you can refer this. The overall algorithm works as:  1. initialize weights (say w=0)  2. iterate till not converged  2.1 iterate over all features (j=0,1...M)  2.1.1 determine the gradient  2.1.2 update the jth weight by subtracting learning rate times the gradient  w(t+1) = w(t) - learning rate \* gradient  Here the important step is #2.1.1 where we compute the gradient. Gradient is nothing but a partial differential of the cost with respect to a particular weight (denoted as wj). The gradient for the jthweight will be:    [eq3_updated](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq3_updated.png)  This is formed from 2 parts:   1. 2\*{..} : This is formed because we’ve differentiated the square of the term in {..} 2. -wj : This is the differentiation of the part in {..} wrt wj. Since its a summation, all other would become 0 and only wj would remain.   Step #2.1.2 involves updating the weights using the gradient. This update step for simple linear regression looks like:  [eq4](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq4-1.png)    I hope you are able to follow along. Note the +ve sign in the RHS is formed after multiplication of 2 -ve signs. I would like to explain point #2 of the gradient descent algorithm mentioned above ‘**iterate till not converged**‘. Here convergence refers to attaining the optimum solution within pre-defined limit.  It is checked using the value of gradient. If the gradient is small enough that means we are very close to optimum and further iterations won’t have a substantial impact on coefficients. The lower-limit on gradient can be changed using the ‘**tol**‘ parameter.  Lets consider the case of ridge regression now.   **2. Ridge Regression** The objective function (also called the cost) to be minimized is the RSS plus the sum of square of the magnitude of weights. This can be depicted mathematically as:  [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq5-1-1024x206.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq5-1.png)  In this case, the gradient would be:  [eq6](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq6-1.png)    Again in the regularization part of gradient, only wj remains and all other would become zero. The corresponding update rule is:  [eq7](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq7-1.png)    Here we can see that second part of the RHS is same as that of simple linear regression. Thus, ridge regression is equivalent to reducing the weight by a factor of (1-2λη) first and then applying the same update rule as simple linear regression. I hope this gives some intuition into why the coefficients get reduced to small numbers but never become zero.  Note that the criteria for convergence in this case remains similar to simple linear regression, i.e. checking the value of gradients. Lets discuss Lasso regression now.   **3. Lasso Regression** The objective function (also called the cost) to be minimized is the RSS plus the sum of absolute value of the magnitude of weights. This can be depicted mathematically as:  [https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq8-1-1024x198.png](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq8-1.png)  In this case, the gradient is not defined as the absolute function is not differentiable at x=0. This can be illustrated as:  [fig1](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/fig1.png)  We can see that the parts on the left and right side of 0 are straight lines with defined derivates but the function can’t be differentiated at x=0. In this case, we have to use a different technique called as **coordinate descent** which is based on the concept of sub-gradients. One of the coordinate descent follows the following algorithms (this is also the default in sklearn):  1. initialize weights (say w=0)  2. iterate till not converged  2.1 iterate over all features (j=0,1...M)  2.1.1 update the jth weight with a value which minimizes the cost  #2.1.1 might look too generalized. But I’m intentionally leaving the details and jumping to the update rule:  [eq9](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/eq9.png)  Here **g(w-j)** represents (but not exactly) the difference between actual outcome and the predicted outcome considering **all EXCEPT the jth variable**. If this value is small, it means that the algorithm is able to predict the outcome fairly well even without the jth variable and thus it can be removed from the equation by setting a zero coefficient. This gives us some intuition into why the coefficients become zero in case of lasso regression.  In coordinate descent, checking convergence is another issue. Since gradients are not defined, we need an alternate method. Many alternatives exist but the simplest one is to **check the step size of the algorithm**. We can check the maximum difference in weights in any particular cycle over all feature weights (#2.1 of algo above).  If this is lower than ‘tol’ specified, algo will stop. The convergence is not as fast as gradient descent and we might have to set the ‘max\_iter’ parameter if a warning appears saying that the algo stopped before convergence. This is why I specified this parameter in the Lasso generic function.  Lets summarize our understanding by comparing the coefficients in all the three cases using the following visual, which shows how the ridge and lasso coefficients behave in comparison to the simple linear regression case.  [fig2](https://www.analyticsvidhya.com/wp-content/uploads/2016/01/fig2.png)  Apologies for the lack of visual appeal. But I think it is good enough to re-inforce the following facts:   1. The ridge coefficients are a reduced factor of the simple linear regression coefficients and thus never attain zero values but very small values 2. The lasso coefficients become zero in a certain range and are reduced by a constant factor, which explains there low magnitude in comparison to ridge.   Before going further, one important issue in case of both ridge and lasso regression is **intercept handling**. Generally, regularizing the intercept is not a good idea and it should be left out of regularization. This requires slight changes in the implementation, which I’ll leave for you to explore.  Now, lets come to the concluding part where we compare the Ridge and Lasso techniques and see where these can be used.   6. Conclusion Now that we have a fair idea of how ridge and lasso regression work, lets try to consolidate our understanding by comparing them and try to appreciate their specific use cases. I will also compare them with some alternate approaches. Lets analyze these under three buckets: **1. Key Difference**  * **Ridge:** It includes all (or none) of the features in the model. Thus, the major advantage of ridge regression is coefficient shrinkage and reducing model complexity. * **Lasso:** Along with shrinking coefficients, lasso performs feature selection as well. (Remember the ‘selection‘ in the lasso full-form?) As we observed earlier, some of the coefficients become exactly zero, which is equivalent to the particular feature being excluded from the model.   Traditionally, techniques like **stepwise regression** were used to perform feature selection and make parsimonious models. But with advancements in Machine Learning, ridge and lasso regression provide very good alternatives as they give much**better output**, require **fewer tuning parameters** and can be **automated** to a large extend. |