Introduction

Google's self-driving cars and robots get a lot of press, but the company's real future is in machine learning, the technology that enables computers to get smarter and more personal.

– Eric Schmidt (Google Chairman)

We are probably living in the most defining period of human history. The period when computing moved from large mainframes to PCs to cloud. But what makes it defining is not what has happened, but what is coming our way in years to come.

What makes this period exciting for some one like me is the democratization of the tools and techniques, which followed the boost in computing. Today, as a data scientist, I can build data crunching machines with complex algorithms for a few dollors per hour. But, reaching here wasn't easy! I had my dark days and nights.

Broadly, there are 3 types of Machine Learning Algorithms..

1. Supervised Learning

How it works: This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.

2. Unsupervised Learning

How it works: In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.

3. Reinforcement Learning:

How it works: Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process

List of Common Machine Learning Algorithms

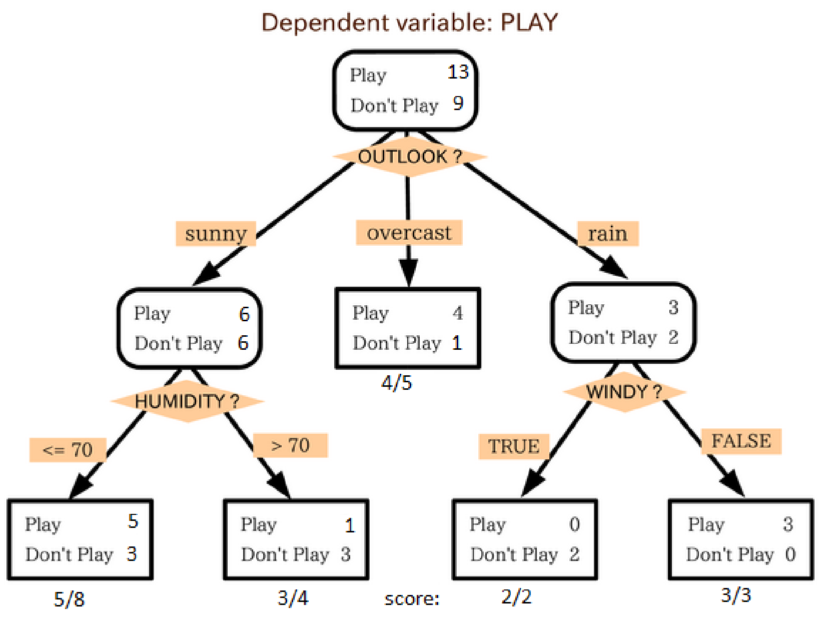
Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem:

1. Linear Regression
2. Logistic Regression
3. Decision Tree
4. SVM
5. Naive Bayes
6. kNN
7. K-Means
8. Random Forest
9. Dimensionality Reduction Algorithms
10. Gradient Boosting algorithms
    1. GBM
    2. XGBoost
    3. LightGBM
    4. CatBoost

CLASSIFICATION ALGORITHMS

3. Decision Tree

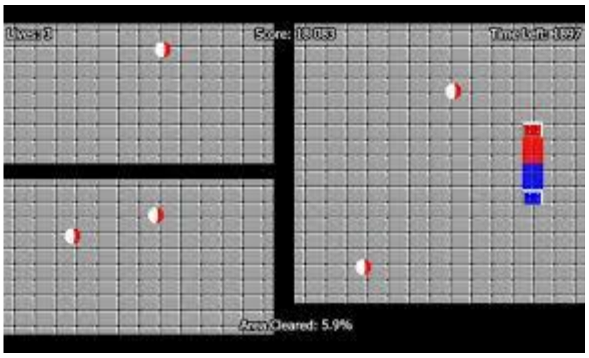
This is one of my favorite algorithm and I use it quite frequently. It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible. For more details, you can read: Decision Tree Simplified.



source: statsexchange

In the image above, you can see that population is classified into four different groups based on multiple attributes to identify 'if they will play or not'. To split the population into different heterogeneous groups, it uses various techniques like Gini, Information Gain, Chi-square, entropy.

The best way to understand how decision tree works, is to play Jezzball – a classic game from Microsoft (image below). Essentially, you have a room with moving walls and you need to create walls such that maximum area gets cleared off with out the balls.



So, every time you split the room with a wall, you are trying to create 2 different populations with in the same room. Decision trees work in very similar fashion by dividing a population in as different groups as possible.

More: Simplified Version of Decision Tree Algorithms

Python Code

```
#Import Library
```

```
#Import other necessary libraries like pandas, numpy...

from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p
redictor) of test_dataset

# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here y
ou can change the algorithm as gini or entropy (information gain) by default it is
gini

# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

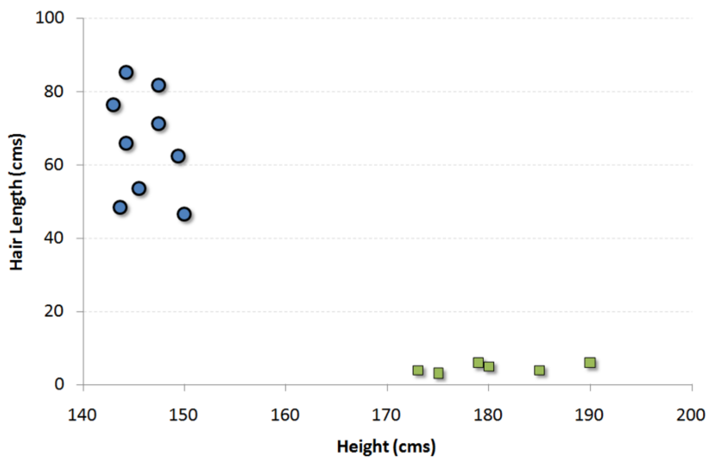
```
library(rpart)

x <- cbind(x_train,y_train)

# grow tree
```

```
fit <- rpart(y_train ~ ., data = x,method="class")

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

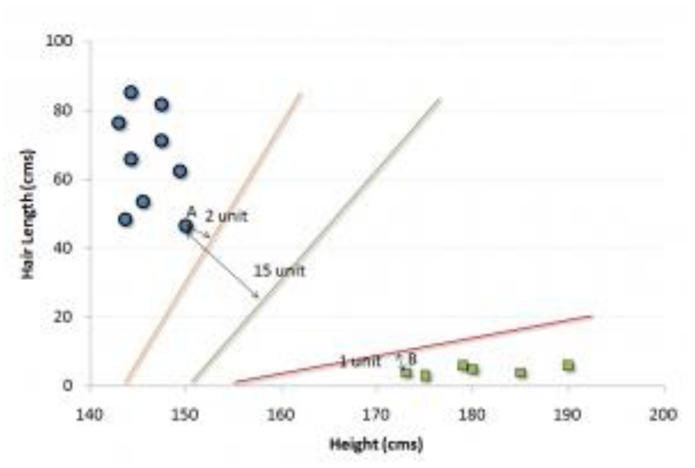
4. SVM (Support Vector Machine)

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)



Now, we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.



In the example shown above, the line which splits the data into two differently classified groups is the *black* line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

More: Simplified Version of Support Vector Machine

Think of this algorithm as playing JezzBall in n-dimensional space. The tweaks in the game are:

- You can draw lines / planes at any angles (rather than just horizontal or vertical as in classic game)
- The objective of the game is to segregate balls of different colors in different rooms.
- And the balls are not moving.

Python Code

```
#Import Library


from sklearn import svm


#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p

redictor) of test_dataset


# Create SVM classification object
```

```
model = svm.svc() # there is various option associated with it, this is simple for
classification. You can refer link, for mo# re detail.

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

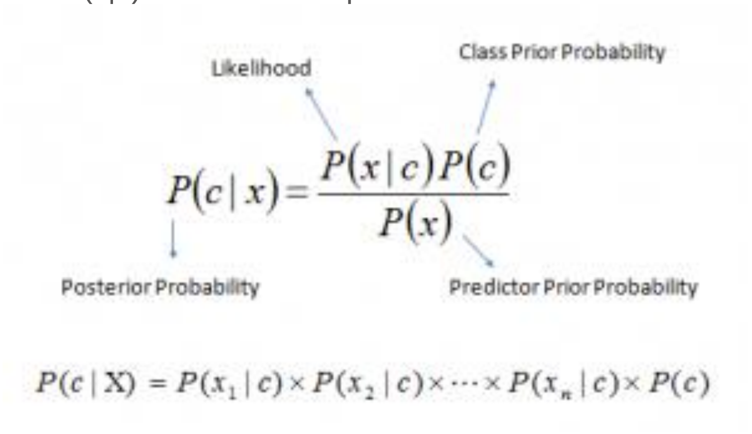
```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model

fit <-svm(y_train ~ ., data = x)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

5. Naive Bayes

It is a classification technique based on Bayes’ theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

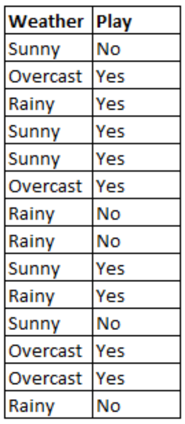
Here,

- $P(c|x)$ is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

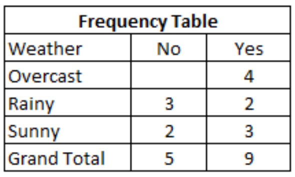
Example: Let’s understand it using an example. Below I have a training data set of weather and corresponding target variable ‘Play’. Now, we need to classify whether players will play or not based on weather condition. Let’s follow the below steps to perform it.

Step 1: Convert the data set to frequency table

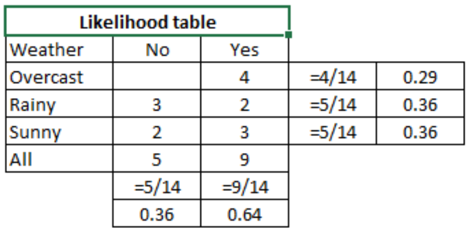
Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.



Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No



Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9



Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will pay if weather is sunny, is this statement is correct?

We can solve it using above discussed method, so P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Python Code

```
#Import Library


from sklearn.naive_bayes import GaussianNB


#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p

redictor) of test_dataset


# Create SVM classification object model = GaussianNB() # there is other distribut

ion for multinomial classes like Bernoulli Naive Bayes, Refer link
```

```
# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model

fit <-naiveBayes(y_train ~ ., data = x)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

8. Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we’ve collection of decision trees (so known as “Forest”). To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.

2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

For more details on this algorithm, comparing with decision tree and tuning model parameters, I would suggest you to read these articles:

1. Introduction to Random forest – Simplified
2. Comparing a CART model to Random Forest (Part 1)
3. Comparing a Random Forest to a CART model (Part 2)
4. Tuning the parameters of your Random Forest model

Python

```
#Import Library

from sklearn.ensemble import RandomForestClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p
redictor) of test_dataset

# Create Random Forest object

model= RandomForestClassifier()

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

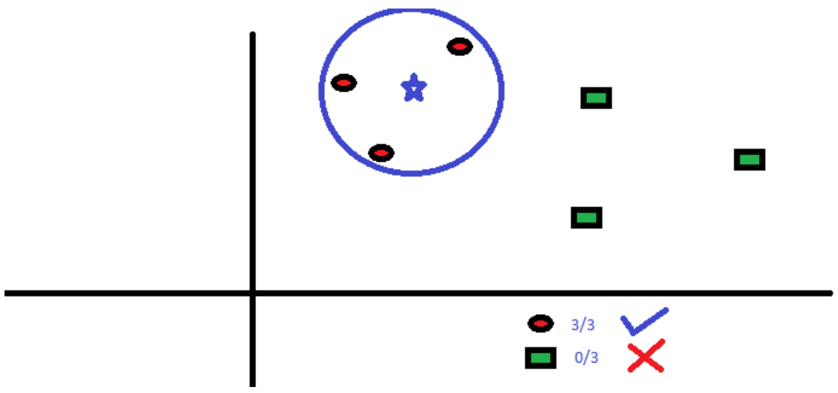
```
library(randomForest)


x <- cbind(x_train,y_train)


# Fitting model

fit <- randomForest(Species ~ ., x,ntree=500)


summary(fit)


#Predict Output


predicted= predict(fit,x_test)
```

6. kNN (k- Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.

More: Introduction to k-nearest neighbors : Simplified.



KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

Things to consider before selecting kNN:

- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for kNN like outlier, noise removal

Python Code

```
#Import Library

from sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p
redictor) of test_dataset

# Create KNeighbors classifier object model

KNeighborsClassifier(n_neighbors=6) # default value for n_neighbors is 5

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(knn)
```

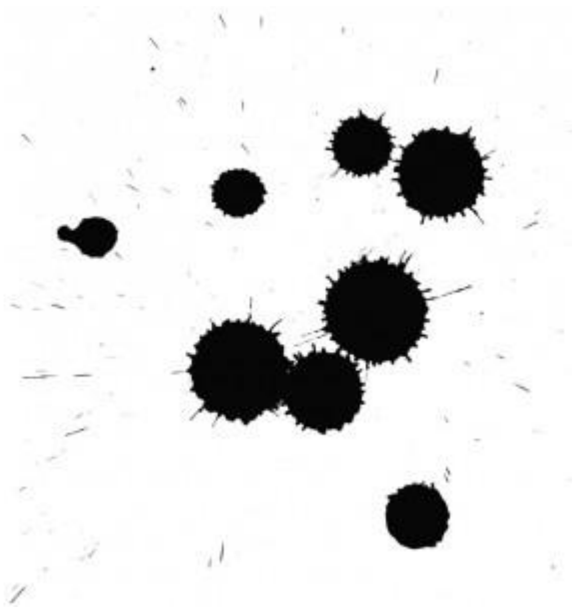
```
x <- cbind(x_train,y_train)

# Fitting model

fit <-knn(y_train ~ ., data = x,k=5)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

CLUSTERING

7. K-Means

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Remember figuring out shapes from ink blots? k means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!



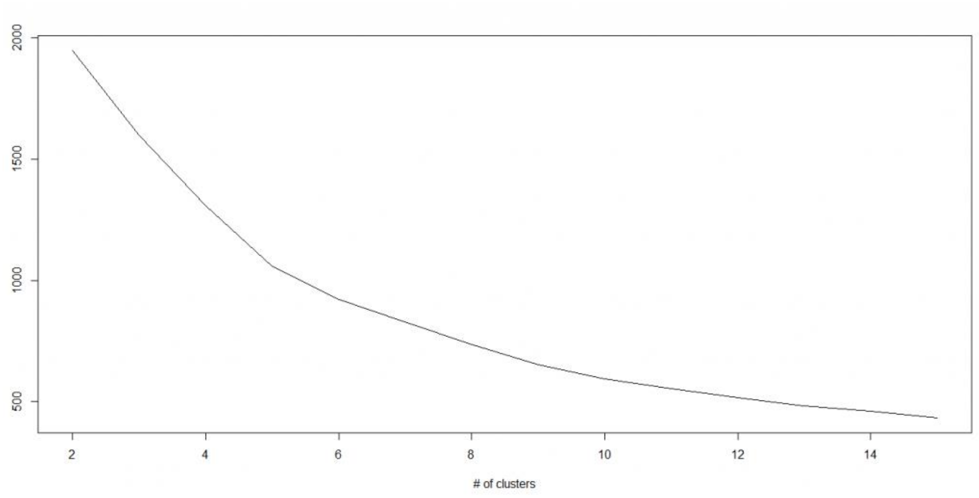
How K-means forms cluster:

1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.
3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

How to determine value of K:

In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.



<u>Python Code</u>

```
#Import Library

from sklearn.cluster import KMeans


#Assumed you have, X (attributes) for training data set and x_test(attributes) of

test_dataset
```

```
# Create KNeighbors classifier object model


k_means = KMeans(n_clusters=3, random_state=0)


# Train the model using the training sets and check score


model.fit(X)


#Predict Output


predicted= model.predict(x_test)
```

R Code

```
library(cluster)


fit <- kmeans(X, 3) # 5 cluster solution
```

Regression

1. Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

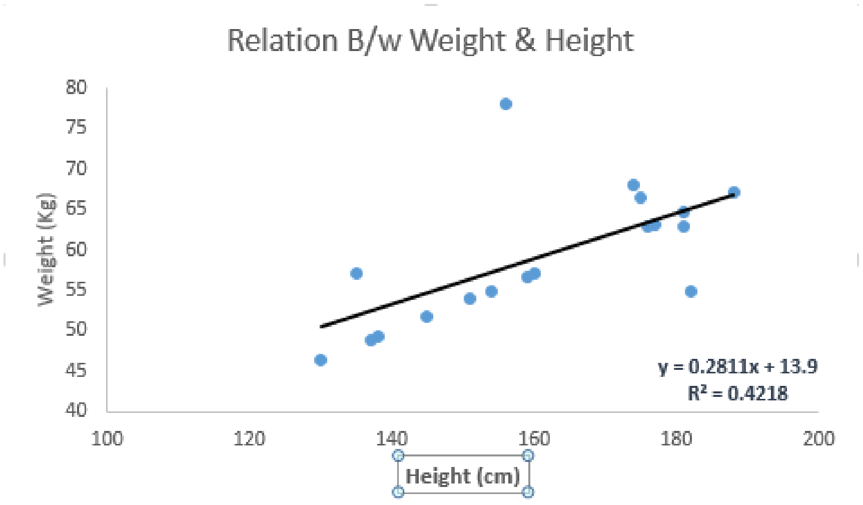
The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation **y=0.2811x+13.9**. Now using this equation, we can find the weight, knowing the height of a person.



Linear Regression is of mainly two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables. While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.

Python Code

```
#Import Library


#Import other necessary libraries like pandas, numpy...


from sklearn import linear_model


#Load Train and Test datasets
```

```
#Identify feature and response variable(s) and values must be numeric and numpy ar
rays

x_train=input_variables_values_training_datasets

y_train=target_variables_values_training_datasets

x_test=input_variables_values_test_datasets

# Create linear regression object

linear = linear_model.LinearRegression()

# Train the model using the training sets and check score

linear.fit(x_train, y_train)

linear.score(x_train, y_train)

#Equation coefficient and Intercept

print('Coefficient: \n', linear.coef_)

print('Intercept: \n', linear.intercept_)

#Predict Output

predicted= linear.predict(x_test)
```

R Code

```
#Load Train and Test datasets
```

```
#Identify feature and response variable(s) and values must be numeric and numpy ar
rays

x_train <- input_variables_values_training_datasets

y_train <- target_variables_values_training_datasets

x_test <- input_variables_values_test_datasets

x <- cbind(x_train,y_train)

# Train the model using the training sets and check score

linear <- lm(y_train ~ ., data = x)

summary(linear)

#Predict Output

predicted= predict(linear,x_test)
```

2. Logistic Regression

Don't get confused by its name! It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

Again, let us try and understand this through a simple example.

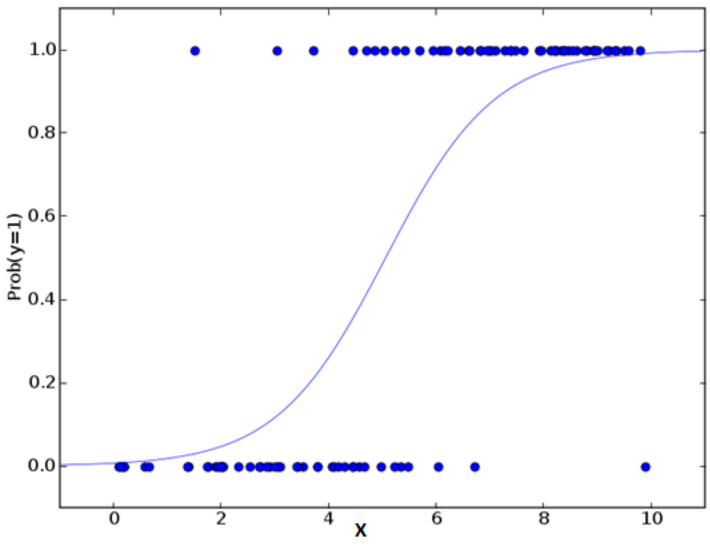
Let's say your friend gives you a puzzle to solve. There are only 2 outcome scenarios – either you solve it or you don't. Now imagine, that you are being given wide range of puzzles / quizzes in an attempt to understand which subjects you are good at. The outcome to this study would be something like this – if you are given a trignometry based tenth grade problem, you are 70% likely to solve it. On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you.

Coming to the math, the log odds of the outcome is modeled as a linear combination of the predictor variables.

```
odds= p/ (1-p) = probability of event occurrence / probability of not event occurr
ence

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. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression).

Now, you may ask, why take a log? For the sake of simplicity, let’s just say that this is one of the best mathematical way to replicate a step function. I can go in more details, but that will beat the purpose of this article.



Python Code

```
#Import Library

from sklearn.linear_model import LogisticRegression


#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p
redictor) of test_dataset
```

```
# Create logistic regression object

model = LogisticRegression()

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Equation coefficient and Intercept

print('Coefficient: \n', model.coef_)

print('Intercept: \n', model.intercept_)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
x <- cbind(x_train,y_train)

# Train the model using the training sets and check score

logistic <- glm(y_train ~ ., data = x,family='binomial')

summary(logistic)

#Predict Output
```

```
predicted= predict(logistic,x_test)
```

Furthermore..

There are many different steps that could be tried in order to improve the model:

- including interaction terms
- removing features
- regularization techniques
- using a non-linear model

How to avoid Over-fitting using Regularization?

Business Situation:

In the world of analytics, where we try to fit a curve to every pattern, Over-fitting is one of the biggest concerns. However, in general models are equipped enough to avoid over-fitting, but in general there is a manual intervention required to make sure the model does not consume more than enough attributes.

Let's consider an example here, we have 10 students in a classroom. We intend to train a model based on their past score to predict their future score. There are 5 females and 5 males in the class. The average score of females is 60 whereas that of males is 80. The overall average of the class is 70.

Now, there are several ways to make the prediction:

- Predict the score as 70 for the entire class.
- Predict score of males = 80 and females = 60. This a simplistic model which might give a better estimate than the first one.
- Now let's try to overkill the problem. We can use the roll number of students to make a prediction and say that every student will exactly score same marks as last time. Now, this is unlikely to be true and we have reached such granular level that we can go seriously wrong.

The first case here is called under fit, the second being an optimum fit and last being an over-fit.

Have a look at the following graphs,

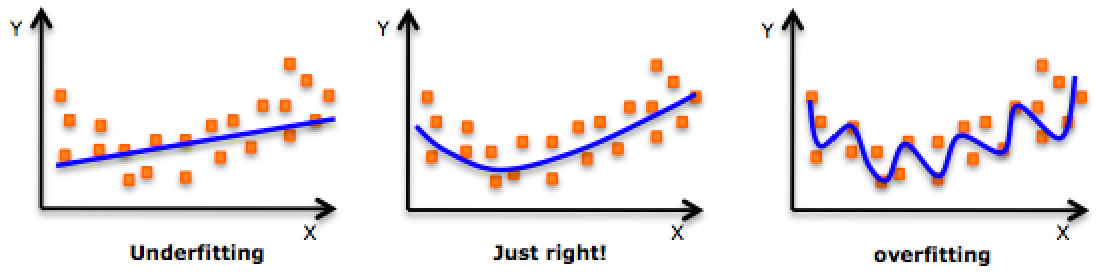


Image source: pingax.com

The trend in above graphs looks like a quadratic trend over independent variable X. A higher degree polynomial might have a very high accuracy on the train population but is expected to fail badly on test dataset. We will briefly touch up on various techniques we use to avoid over-fitting. And then focus on a special technique called **Regularization**.

Methods to avoid Over-fitting:

Following are the commonly used methodologies :

1. **Cross-Validation** : Cross Validation in its simplest form is a one round validation, where we leave one sample as in-time validation and rest for training the model. But for keeping lower variance a higher fold cross validation is preferred.
2. **Early Stopping** : Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit.
3. **Pruning** : Pruning is used extensively while building CART models. It simply removes the nodes which add little predictive power for the problem in hand.
4. **Regularization** : This is the technique we are going to discuss in more details. Simply put, it introduces a cost term for bringing in more features with the objective function. Hence, it tries to push the coefficients for many variables to zero and hence reduce cost term.

Regularization basics

A simple linear regression is an equation to estimate y, given a bunch of x. The equation looks something as follows :

```
y = a1x1 + a2x2  + a3x3 + a4x4 .......
```

In the above equation, a1, a2, a3 … are the coefficients and x1, x2, x3 .. are the independent variables. Given a data containing x and y, we estimate a1, a2 , a3 …based on an objective function. For a linear regression the objective function is as follows :

$$\min_{f} |Y_i - f(X_i)|^2$$

Now, this optimization might simply overfit the equation if x1 , x2 , x3 (independent variables) are too many in numbers. Hence we introduce a new penalty term in our objective function to find the estimates of co-efficient. Following is the modification we make to the equation :

$$\min_{f \in H} \sum_{i=1}^{n} |Y_i - f(X_i)|^2 + \lambda \|f\|_H^2$$

The new term in the equation is the sum of squares of the coefficients (except the bias term) multiplied by the parameter lambda. Lambda = 0 is a super over-fit scenario and Lambda = Infinity brings down the problem to just single mean estimation. Optimizing Lambda is the task we need to solve looking at the trade-off between the prediction accuracy of training sample and prediction accuracy of the hold out sample.

Understanding Regularization Mathematically

There are multiple ways to find the coefficients for a linear regression model. One of the widely used method is gradient descent. Gradient descent is an iterative method which takes some initial guess on coefficients and then tries to converge such that the objective function is minimized. Hence we work with partial derivatives on the coefficients. Without getting into much details of the derivation, here I will put down the final iteration equation :

$$\theta_j := \theta_j - \alpha \quad \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Here, theta are the estimates of the coefficients. Alpha is the learning parameter which will guide our estimates to convergence. Now let’s bring in our cost terms. After taking the derivative of coefficient square, it reduces down to a linear term. Following is the final iteration equation you get after embedding the penalty/cost term.

$$\theta_j := \theta_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Now if you look carefully to the equation, the starting point of every theta iteration is slightly lesser than the previous value of theta. This is the only difference between the normal gradient descent and the gradient descent regularized. This tries to find converged value of theta which is as low as possible.

A comprehensive beginners guide for Linear, Ridge and Lasso Regression

Introduction

I was talking to one of my friends who happens to be an operations manager at one of the Supermarket chains in India. Over our discussion, we started talking about the amount of preparation the store chain needs to do before the Indian festive season (Diwali) kicks in.

He told me how critical it is for them to estimate / predict which product will sell like hot cakes and which would not prior to the purchase. A bad decision can leave your customers to look for offers and products in the competitor stores. The challenge does not finish there – you need to estimate the sales of products across a range of different categories for stores in varied locations and with consumers having different consumption techniques.

While my friend was describing the challenge, the data scientist in me started smiling! Why? I just figured out a potential topic for my next article. In today's article, I will tell you everything you need to know about regression models and how they can be used to solve prediction problems like the one mentioned above.

A small exercise to get your mind racing

Take a moment to list down all those factors you can think, on which the sales of a store will be dependent on. For each factor create an hypothesis about why and how that factor would influence the sales of various products. For example – I expect the sales of products to depend on the location of the store, because the local residents in each area would have different lifestyle. The amount of bread a store will sell in Ahmedabad would be a fraction of similar store in Mumbai.

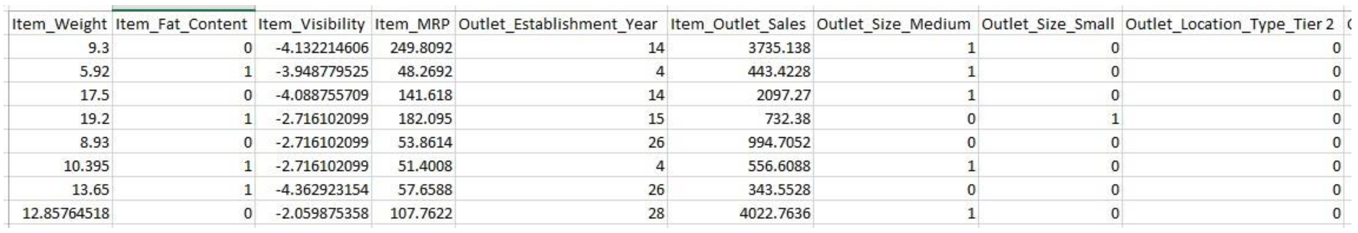
Similarly list down all possible factors you can think of.

Location of your shop, availability of the products, size of the shop, offers on the product, advertising done by a product, placement in the store could be some features on which your sales would depend on.

How many factors were you able to think of? If it is less than 15, give it more time and think again! A seasoned data scientist working on this problem would possibly think of tens and hundreds of such factors.

With that thought in mind, I am providing you with one such data set – The Big Mart Sales. In the data set, we have product wise Sales for Multiple outlets of a chain.

Let’s us take a snapshot of the dataset:



Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales	Outlet_Size_Medium	Outlet_Size_Small	Outlet_Location_Type_Tier 2
9.3	0	-4.132214606	249.8092	14	3735.138	1	0	0
5.92	1	-3.948779525	48.2692	4	443.4228	1	0	0
17.5	0	-4.088755709	141.618	14	2097.27	1	0	0
19.2	1	-2.716102099	182.095	15	732.38	0	1	0
8.93	0	-2.716102099	53.8614	26	994.7052	0	0	0
10.395	1	-2.716102099	51.4008	4	556.6088	1	0	0
13.65	1	-4.362923154	57.6588	26	343.5528	0	0	0
12.85764518	0	-2.059875358	107.7622	28	4022.7636	1	0	0

In the dataset, we can see characteristics of the sold item (fat content, visibility, type, price) and some characteristics of the outlet (year of establishment, size, location, type) and the number of the items sold for that particular item. Let’s see if we can predict sales using these features.

Table of Contents

1. Simple models for Prediction
2. Linear Regression
3. The Line of Best Fit
4. Gradient Descent
5. Using Linear Regression for prediction
6. Evaluate your Model – R square and Adjusted R squared
7. Using all the features for prediction
8. Polynomial Regression
9. Bias and Variance
10. Regularization
11. Ridge Regression
12. Lasso Regression
13. ElasticNet Regression
14. Types of Regularization Techniques [Optional]

1. Simple models for Prediction

Let us start with making predictions using a few simple ways to start with. If I were to ask you, what could be the simplest way to predict the sales of an item, what would you say?

Model 1 – Mean sales:

Even without any knowledge of machine learning, you can say that if you have to predict sales for an item – it would be the average over last few days . / months / weeks.

It is a good thought to start, but it also raises a question – how good is that model?

Turns out that there are various ways in which we can evaluate how good is our model. The most common way is Mean Squared Error. Let us understand how to measure it.

Prediction error

To evaluate how good is a model, let us understand the impact of wrong predictions. If we predict sales to be higher than what they might be, the store will spend a lot of money making unnecessary arrangement which would lead to excess inventory. On the other side if I predict it too low, I will lose out on sales opportunity.

So, the simplest way of calculating error will be, to calculate the difference in the predicted and actual values. However, if we simply add them, they might cancel out, so we square these errors before adding. We also divide them by the number of data points to calculate a mean error since it should not be dependent on number of data points.

$$\frac{(e_1^2 + e_2^2 + \ldots + e_n^2)}{n}$$

[each error squared and divided by number of data points]

This is known as the mean squared error.

Here e1, e2 …. , en are the difference between the actual and the predicted values.

So, in our first model what would be the mean squared error? On predicting the mean for all the data points, we get a mean squared error = 29,11,799. Looks like huge error. May be its not so cool to simply predict the average value.

Let’s see if we can think of something to reduce the error.

Model 2 – Average Sales by Location:

We know that location plays a vital role in the sales of an item. For example, let us say, sales of car would be much higher in Delhi than its sales in Varanasi. Therefore let us use the data of the column ‘Outlet_Location_Type’.

So basically, let us calculate the average sales for each location type and predict accordingly.

On predicting the same, we get mse = 28,75,386, which is less than our previous case. So we can notice that by using a characteristic[location], we have reduced the error.

Now, what if there are multiple features on which the sales would depend on. How would we predict sales using this information? Linear regression comes to our rescue.

2. Linear Regression

Linear regression is the simplest and most widely used statistical technique for predictive modeling. It basically gives us an equation, where we have our features as independent variables, on which our target variable [sales in our case] is dependent upon.

So what does the equation look like? Linear regression equation looks like this:

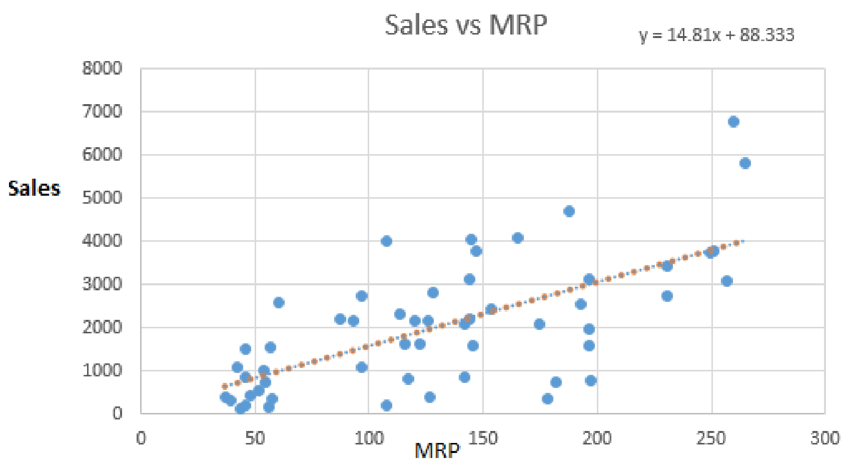
$$Y = \theta_1 X_1 + \theta_2 X_2 + \dots \theta_n X_n$$

Here, we have Y as our dependent variable (Sales), X’s are the independent variables and all thetas are the coefficients. Coefficients are basically the weights assigned to the features, based on their importance. For example, if we believe that sales of an item would have higher dependency upon the type of location as compared to size of store, it means that sales in a tier 1 city would be more even if it is a smaller outlet than a tier 3 city in a bigger outlet. Therefore, coefficient of location type would be more than that of store size.

So, firstly let us try to understand linear regression with only one feature, i.e., only one independent variable. Therefore our equation becomes,

$$Y = \theta_1 * X + \theta_0$$

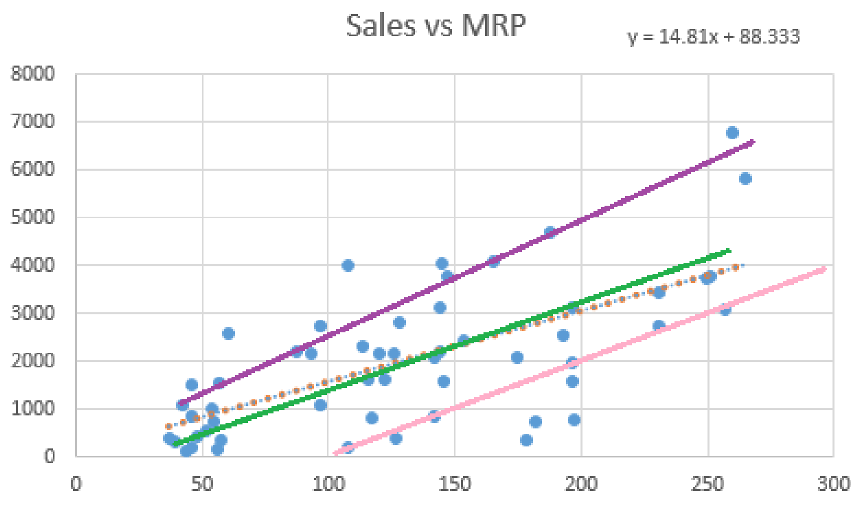
This equation is called a simple linear regression equation, which represents a straight line, where ‘Θ0’ is the intercept, ‘Θ_1’ is the slope of the line. Take a look at the plot below between sales and MRP.



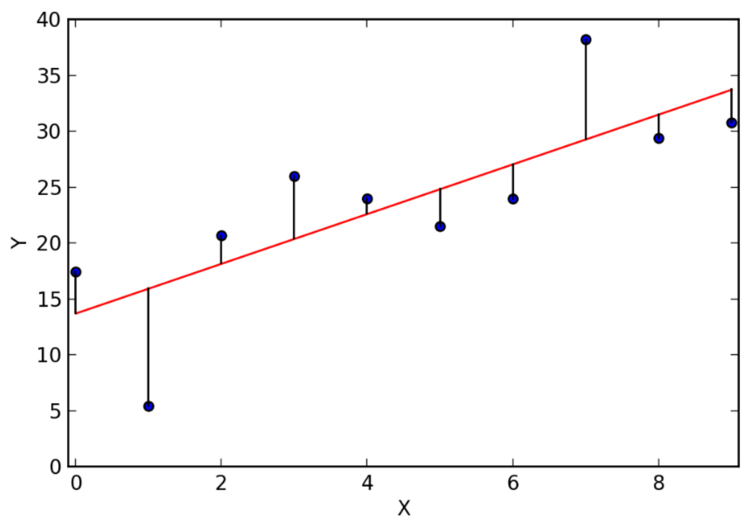
Surprisingly, we can see that sales of a product increases with increase in its MRP. Therefore the dotted red line represents our regression line or the line of best fit. But one question that arises is how you would find out this line?

3. The Line of Best Fit

As you can see below there can be so many lines which can be used to estimate Sales according to their MRP. So how would you choose the best fit line or the regression line?



The main purpose of the best fit line is that our predicted values should be closer to our actual or the observed values, because there is no point in predicting values which are far away from the real values. In other words, we tend to minimize the difference between the values predicted by us and the observed values, and which is actually termed as error. Graphical representation of error is as shown below. These errors are also called as residuals. The residuals are indicated by the vertical lines showing the difference between the predicted and actual value.



Okay, now we know that our main objective is to find out the error and minimize it. But before that, let's think of how to deal with the first part, that is, to calculate the error. We already know that error is the difference between the value predicted by us and the observed value. Let's just consider three ways through which we can calculate error:

- **Sum of residuals (∑(Y – h(X)))** – it might result in cancelling out of positive and negative errors.
- **Sum of the absolute value of residuals (∑|Y-h(X)|)** – absolute value would prevent cancellation of errors
- **Sum of square of residuals (∑ (Y-h(X))^2)** – it's the method mostly used in practice since here we penalize higher error value much more as compared to a smaller one, so that there is a significant difference between making big errors and small errors, which makes it easy to differentiate and select the best fit line.

Therefore, sum of squares of these residuals is denoted by:

$$SS_{residuals} = \sum_{i=1}^{m} (h(x) - y)^2$$

where, h(x) is the value predicted by us, h(x) =Θ_1*x +Θ_0 , y is the actual values and m is the number of rows in the training set.

The cost Function

So let's say, you increased the size of a particular shop, where you predicted that the sales would be higher. But despite increasing the size, the sales in that shop did not increase that much. So the cost applied in increasing the size of the shop, gave you negative results.

So, we need to minimize these costs. Therefore we introduce a cost function, which is basically used to define and measure the error of the model.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

If you look at this equation carefully, it is just similar to sum of squared errors, with just a factor of 1/2m is multiplied in order to ease mathematics.

So in order to improve our prediction, we need to minimize the cost function. For this purpose we use the gradient descent algorithm. So let us understand how it works.

4. Gradient Descent

Let us consider an example, we need to find the minimum value of this equation,

Y= 5x + 4x^2. In mathematics, we simple take the derivative of this equation with respect to x, simply equate it to zero. This gives us the point where this equation is minimum. Therefore substituting that value can give us the minimum value of that equation.

Gradient descent works in a similar manner. It iteratively updates Θ, to find a point where the cost function would be minimum. If you wish to study gradient descent in depth, I would highly recommend going through this article.

5. Using Linear Regression for Prediction

Now let us consider using Linear Regression to predict Sales for our big mart sales problem.

Model 3 – Enter Linear Regression:

From the previous case, we know that by using the right features would improve our accuracy. So now let us use two features, MRP and the store establishment year to estimate sales.

Now, let us built a linear regression model in python considering only these two features.

```
# importing basic libraries

import numpy as np

import pandas as pd
```

```
from pandas import Series, DataFrame

from sklearn.model_selection import train_test_split

import test and train file

train = pd.read_csv('Train.csv')

test = pd.read_csv('test.csv')

# importing linear regressionfrom sklearn

from sklearn.linear_model import LinearRegression

lreg = LinearRegression()

splitting into training and cv for cross validation

X = train.loc[:,['Outlet_Establishment_Year','Item_MRP']]

x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

training the model

lreg.fit(x_train,y_train)

predicting on cv

pred = lreg.predict(x_cv)

calculating mse

mse = np.mean((pred - y_cv)**2)
```

In this case, we got mse = 19,10,586.53, which is much smaller than our model 2. Therefore predicting with the help of two features is much more accurate.

Let us take a look at the coefficients of this linear regression model.

```
# calculating coefficients

coeff = DataFrame(x_train.columns)

coeff['Coefficient Estimate'] = Series(lreg.coef_)

coeff
```

	0	Coefficient Estimate
0	Outlet_Establishment_Year	-11.892070
1	Item_MRP	15.484779

Therefore, we can see that MRP has a high coefficient, meaning items having higher prices have better sales.

6. Evaluating your Model – R square and adjusted R-square

How accurate do you think the model is? Do we have any evaluation metric, so that we can check this? Actually we have a quantity, known as R-Square.

R-Square: It determines how much of the total variation in Y (dependent variable) is explained by the variation in X (independent variable). Mathematically, it can be written as:

$$R - Square = 1 - \frac{\sum(Y_{actual} - Y_{predicted})^2}{\sum(Y_{actual} - Y_{mean})^2}$$

The value of R-square is always between 0 and 1, where 0 means that the model does not model explain any variability in the target variable (Y) and 1 meaning it explains full variability in the target variable.

Now let us check the r-square for the above model.

```
lreg.score(x_cv,y_cv)
```

```
0.3287
```

In this case, R^2 is 32%, meaning, only 32% of variance in sales is explained by year of establishment and MRP. In other words, if you know year of establishment and the MRP, you'll have 32% information to make an accurate prediction about its sales.

Now what would happen if I introduce one more feature in my model, will my model predict values more closely to its actual value? Will the value of R-Square increase?

Let us consider another case.

Model 4 – Linear regression with more variables

We learnt, by using two variables rather than one, we improved the ability to make accurate predictions about the item sales.

So, let us introduce another feature 'weight' in case 3. Now let's build a regression model with these three features.

```
X = train.loc[:,['Outlet_Establishment_Year','Item_MRP','Item_Weight']]

splitting into training and cv for cross validation

x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

## training the model

lreg.fit(x_train,y_train)
```

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

It produces an error, because item weights column have some missing values. So let us impute it with the mean of other non-null entries.

```
train['Item_Weight'].fillna((train['Item_Weight'].mean()), inplace=True)
```

Let us try to run the model again.

```
training the model lreg.fit(x_train,y_train)

## splitting into training and cv for cross validation

x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales)

## training the model lreg.fit(x_train,y_train)

predicting on cv pred = lreg.predict(x_cv)

calculating mse

mse = np.mean((pred - y_cv)**2)

mse

1853431.59

## calculating coefficients

coeff = DataFrame(x_train.columns)

coeff['Coefficient Estimate'] = Series(lreg.coef_)
```

	0	Coefficient Estimate
0	Outlet_Establishment_Year	-10.933946
1	Item_MRP	15.650872
2	Item_Weight	1.203274

`calculating r-square`

`lreg.score(x_cv,y_cv)` `0.32942`

Therefore we can see that the mse is further reduced. There is an increase in the value R-square, does it mean that the addition of item weight is useful for our model?

Adjusted R-square

The only drawback of R^2 is that if new predictors (X) are added to our model, R^2 only increases or remains constant but it never decreases. We can not judge that by increasing complexity of our model, are we making it more accurate?

That is why, we use “Adjusted R-Square”.

The Adjusted R-Square is the modified form of R-Square that has been adjusted for the number of predictors in the model. It incorporates model’s degree of freedom. The adjusted R-Square only increases if the new term improves the model accuracy.

$$R^2 \text{ adjusted} = 1 - \frac{(1-R^2)(N-1)}{N-p-1}$$

where

R^2 = Sample R square

p = Number of predictors

N = total sample size

7. Using all the features for prediction

Now let us built a model containing all the features. While building the regression models, I have only used continuous features. This is because we need to treat categorical variables differently before they can used in linear regression model. There are different techniques to treat them, here I have used one hot encoding(convert each class of a categorical variable as a feature). Other than that I have also imputed the missing values for outlet size.

Data pre-processing steps for regression model

```
# imputing missing values

train['Item_Visibility'] =
train['Item_Visibility'].replace(0,np.mean(train['Item_Visibility']))

train['Outlet_Establishment_Year'] = 2013 - train['Outlet_Establishment_Year']

train['Outlet_Size'].fillna('Small',inplace=True)

# creating dummy variables to convert categorical into numeric values

mylist = list(train1.select_dtypes(include=['object']).columns)

dummies = pd.get_dummies(train[mylist], prefix= mylist)

train.drop(mylist, axis=1, inplace = True)

X = pd.concat([train,dummies], axis =1 )
```

Building the model

```
import numpy as np

import pandas as pd

from pandas import Series, DataFrame

import matplotlib.pyplot as plt

%matplotlib inline

train = pd.read_csv('training.csv')

test = pd.read_csv('testing.csv')

# importing linear regression

from sklearn from sklearn.linear_model import LinearRegression

lreg = LinearRegression()

# for cross validation

from sklearn.model_selection import train_test_split

X = train.drop('Item_Outlet_Sales',1)
```

```
x_train, x_cv, y_train, y_cv = train_test_split(X,train.Item_Outlet_Sales,
test_size =0.3)

# training a linear regression model on train

lreg.fit(x_train,y_train)

# predicting on cv

pred_cv = lreg.predict(x_cv)

# calculating mse

mse = np.mean((pred_cv - y_cv)**2)

mse

1348171.96

# evaluation using r-square

lreg.score(x_cv,y_cv)

0.54831541460870059
```

Clearly, we can see that there is a great improvement in both mse and R-square, which means that our model now is able to predict much closer values to the actual values.

Selecting the right features for your model

When we have a high dimensional data set, it would be highly inefficient to use all the variables since some of them might be imparting redundant information. We would need to select the right set of variables which give us an accurate model as well as are able to explain the dependent variable well. There are multiple ways to select the right set of variables for the model. First among them would be the business understanding and domain knowledge. For instance while predicting sales we know that marketing efforts should impact positively towards sales and is an important feature in your model. We should also take care that the variables we're selecting should not be correlated among themselves.

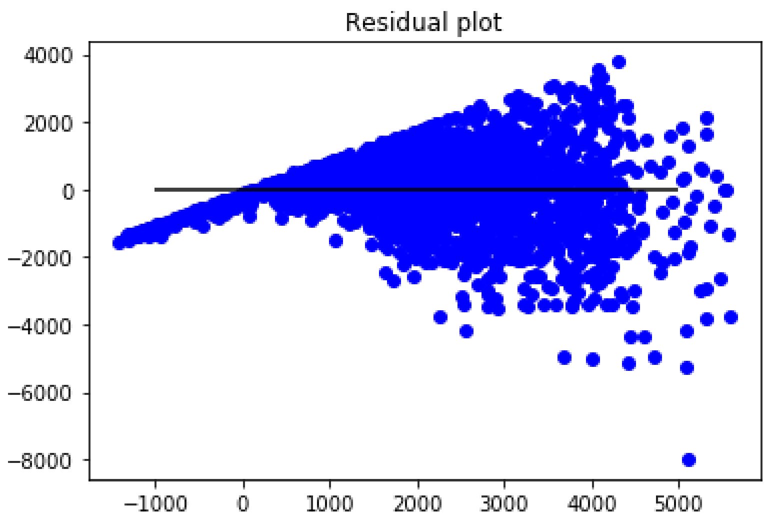
Instead of manually selecting the variables, we can automate this process by using forward or backward selection. Forward selection starts with most significant predictor in the model and adds variable for each step. Backward elimination starts with all predictors in the model and removes the least significant variable for each step. Selecting criteria can be set to any statistical measure like R-square, t-stat etc.

Interpretation of Regression Plots

Take a look at the residual vs fitted values plot.

```
residual plot

x_plot = plt.scatter(pred_cv, (pred_cv - y_cv), c='b')

plt.hlines(y=0, xmin= -1000, xmax=5000)

plt.title('Residual plot')
```

```
<matplotlib.text.Text at 0x6e915433c8>
```



We can see a funnel like shape in the plot. This shape indicates **Heteroskedasticity**. The presence of non-constant variance in the error terms results in heteroskedasticity. We can clearly see that the variance of error terms(residuals) is not constant. Generally, non-constant variance arises in presence of outliers or extreme leverage values. These values get too much weight, thereby disproportionately influencing the model’s performance. When this phenomenon occurs, the confidence interval for out of sample prediction tends to be unrealistically wide or narrow.

We can easily check this by looking at residual vs fitted values plot. If heteroskedasticity exists, the plot would exhibit a funnel shape pattern as shown above. This indicates signs of non linearity in the data which has not been captured by the model. I would highly recommend going through this article for a detailed understanding of assumptions and interpretation of regression plots.

In order to capture this non-linear effects, we have another type of regression known as polynomial regression. So let us now understand it.

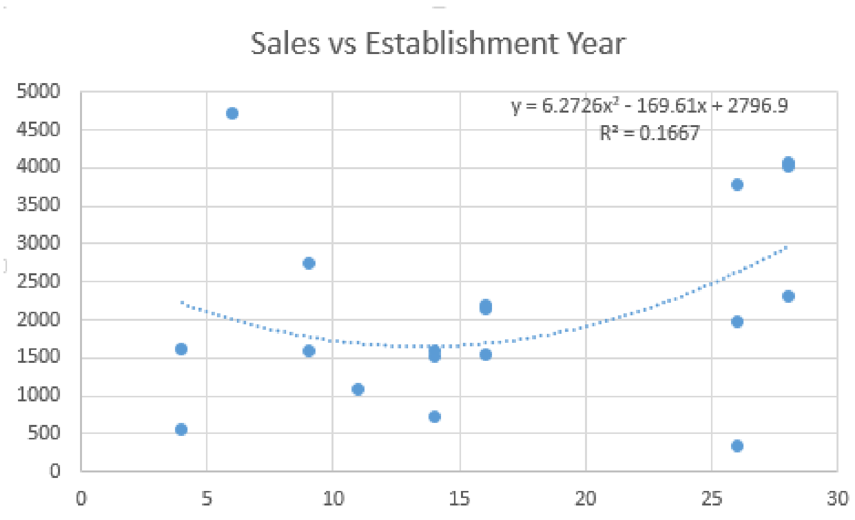
8. Polynomial Regression

Polynomial regression is another form of regression in which the maximum power of the independent variable is more than 1. In this regression technique, the best fit line is not a straight line instead it is in the form of a curve.

Quadratic regression, or regression with second order polynomial, is given by the following equation:

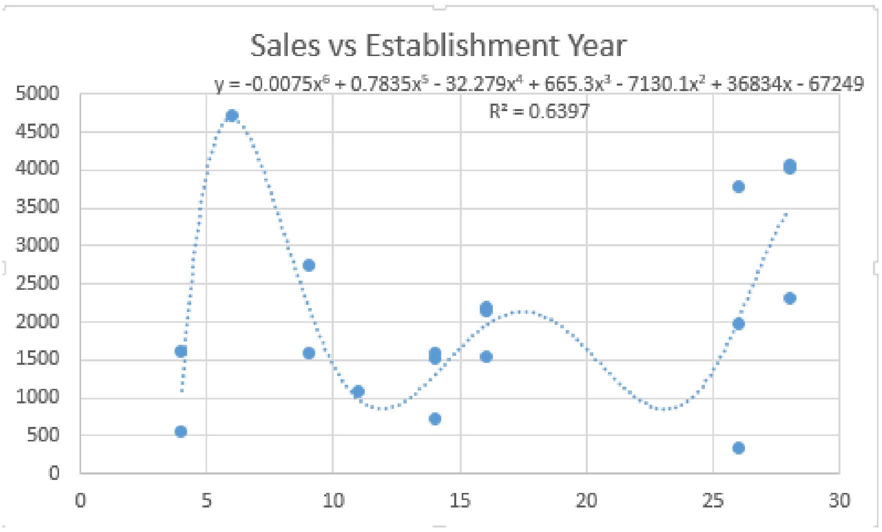
$Y = \Theta_1 + \Theta_2 * x + \Theta_3 * x^2$

Now take a look at the plot given below.

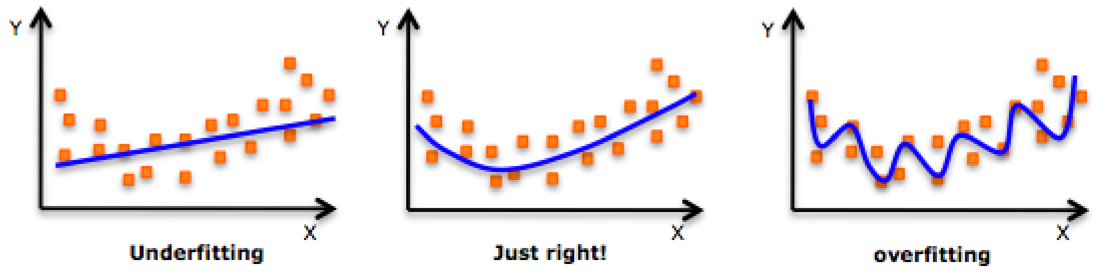


Clearly the quadratic equation fits the data better than simple linear equation. In this case, what do you think will the R-square value of quadratic regression greater than simple linear regression? Definitely yes, because quadratic regression fits the data better than linear regression. While quadratic and cubic polynomials are common, but you can also add higher degree polynomials.

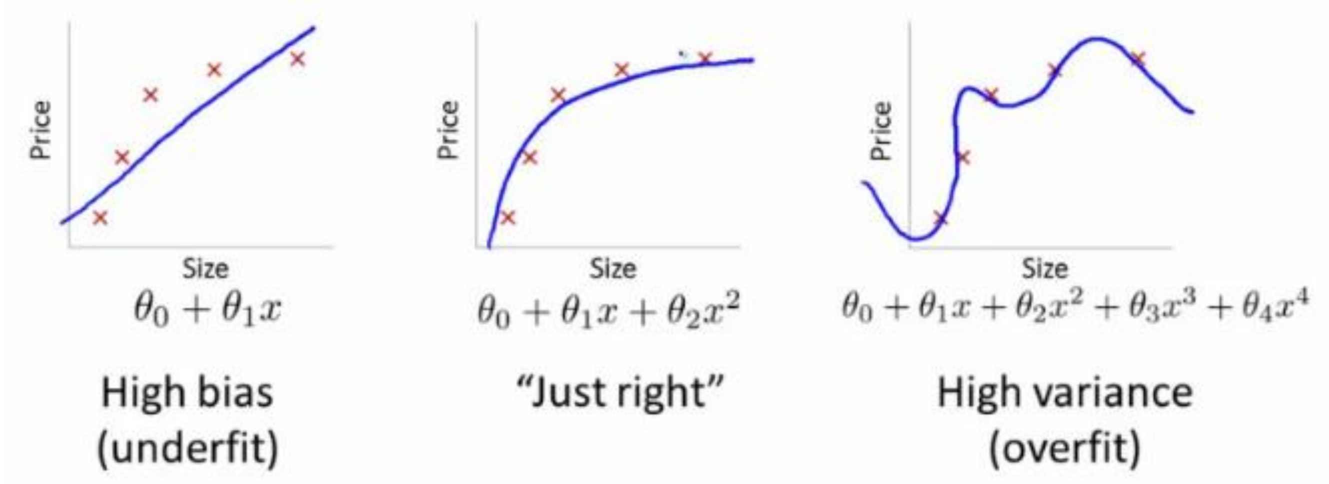
Below figure shows the behavior of a polynomial equation of degree 6.



So do you think it's always better to use higher order polynomials to fit the data set. Sadly, no. Basically, we have created a model that fits our training data well but fails to estimate the real relationship among variables beyond the training set. Therefore our model performs poorly on the test data. This problem is called as **over-fitting**. We also say that the model has high variance and low bias.



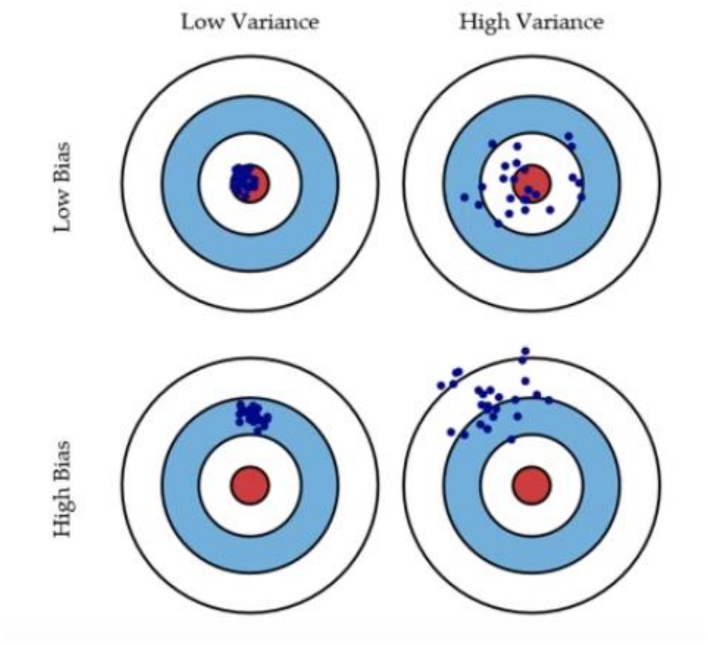
Similarly, we have another problem called **underfitting**, it occurs when our model neither fits the training data nor generalizes on the new data.



Our model is underfit when we have high bias and low variance.

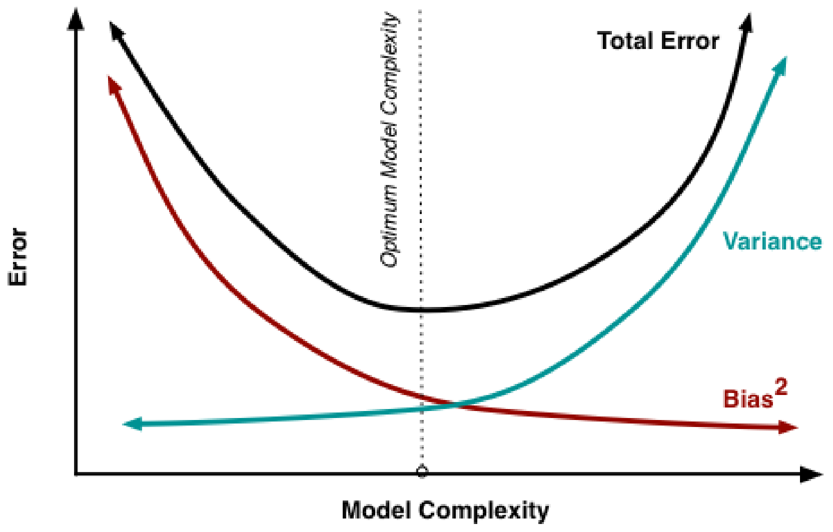
9. Bias and Variance in regression models

What does that bias and variance actually mean? Let us understand this by an example of archery targets.



Let’s say we have model which is very accurate, therefore the error of our model will be low, meaning a low bias and low variance as shown in first figure. All the data points fit within the bulls-eye. Similarly we can say that if the variance increases, the spread of our data point increases which results in less accurate prediction. And as the bias increases the error between our predicted value and the observed values increases.

Now how this bias and variance is balanced to have a perfect model? Take a look at the image below and try to understand.



As we add more and more parameters to our model, its complexity increases, which results in increasing variance and decreasing bias, i.e., overfitting. So we need to find out one optimum point in our model where the decrease in bias is equal to increase in variance. In practice, there is no analytical way to find this point. So how to deal with high variance or high bias?

To overcome underfitting or high bias, we can basically add new parameters to our model so that the model complexity increases, and thus reducing high bias.

Now, how can we overcome Overfitting for a regression model?

Basically there are two methods to overcome overfitting,

- Reduce the model complexity
- Regularization

Here we would be discussing about Regularization in detail and how to use it to make your model more generalized.

10. Regularization

You have your model ready, you have predicted your output. So why do you need to study regularization? Is it necessary?

Suppose you have taken part in a competition, and in that problem you need to predict a continuous variable. So you applied linear regression and predicted your output. Voila! You are on the leaderboard. But wait what you see is still there are many people above you on the leaderboard. But you did everything right then how is it possible?

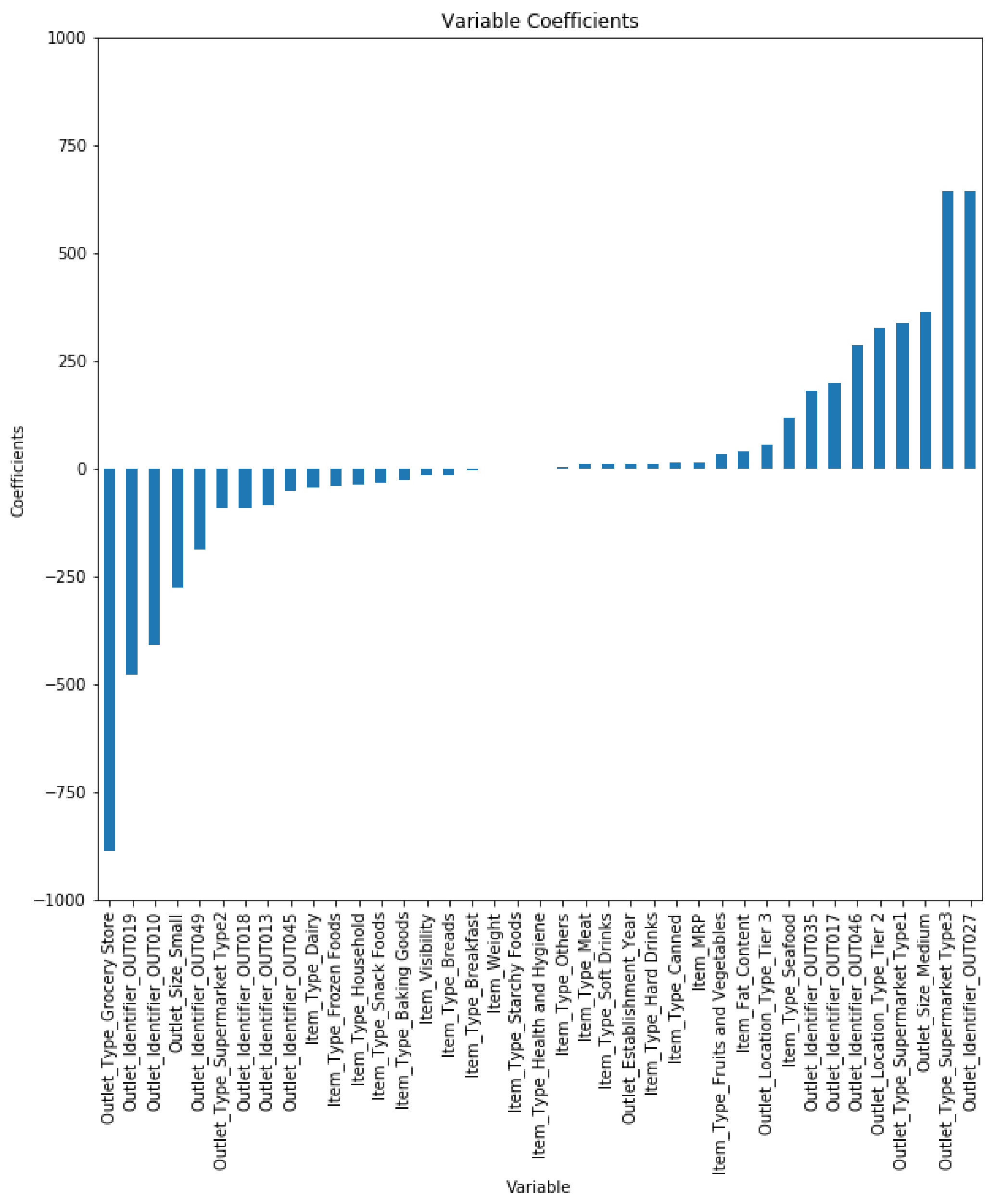
"*Everything should be made simple as possible, but not simpler – Albert Einstein*"

What we did was simpler, everybody else did that, now let us look at making it simple. That is why, we will try to optimize our code with the help of regularization.

In regularization, what we do is normally we keep the same number of features, but reduce the magnitude of the coefficients $_j$. How does reducing the coefficients will help us?

Let us take a look at the coefficients of feature in our above regression model.

```
checking the magnitude of coefficients

predictors = x_train.columns

coef = Series(lreg.coef_,predictors).sort_values()

coef.plot(kind='bar', title='Modal Coefficients')
```



We can see that coefficients of Outlet_Identifier_OUT027 and Outlet_Type_Supermarket_Type3(last 2) is much higher as compared to rest of the coefficients. Therefore the total sales of an item would be more driven by these two features.

How can we reduce the magnitude of coefficients in our model? For this purpose, we have different types of regression techniques which uses regularization to overcome this problem. So let us discuss them.

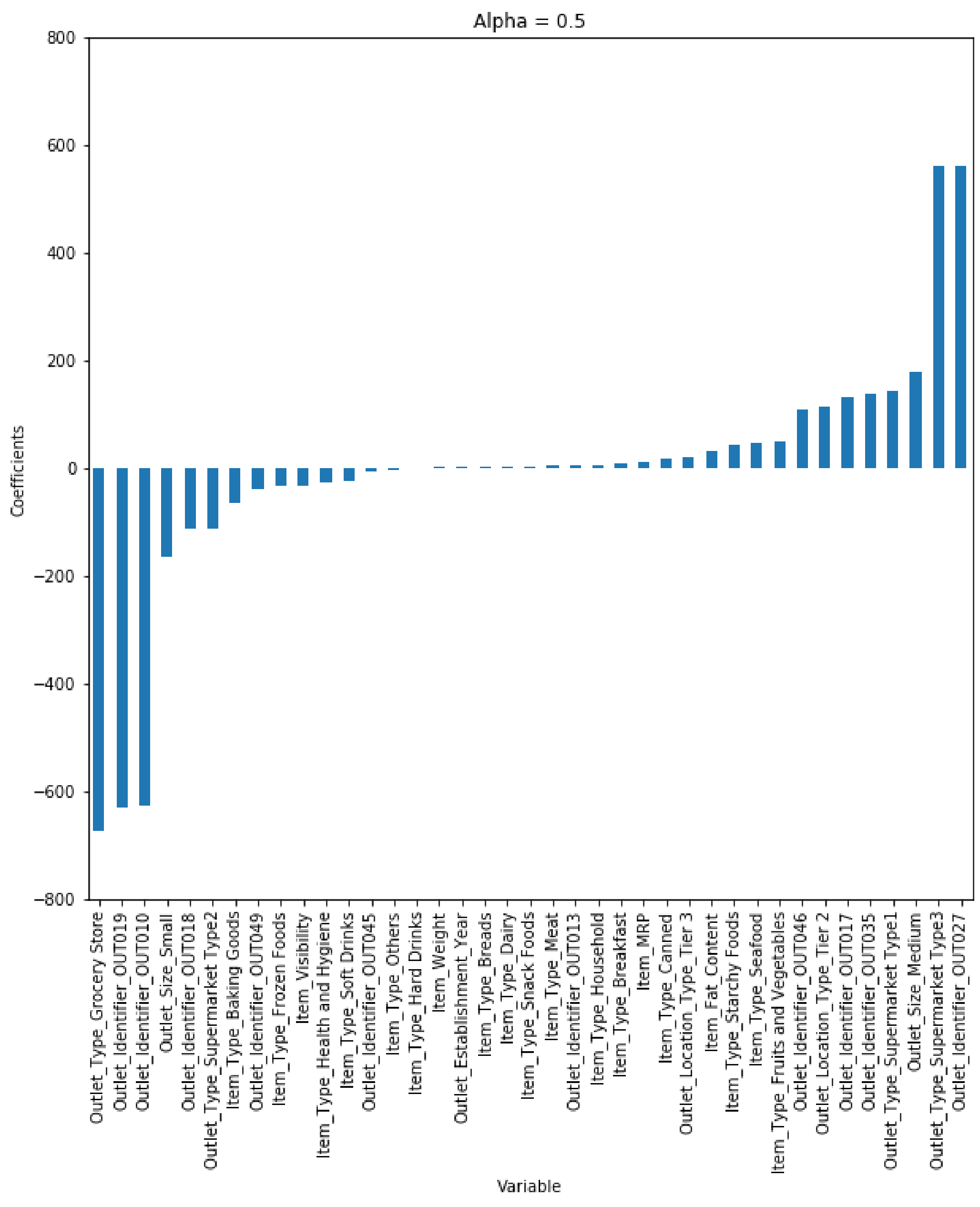
11. Ridge Regression

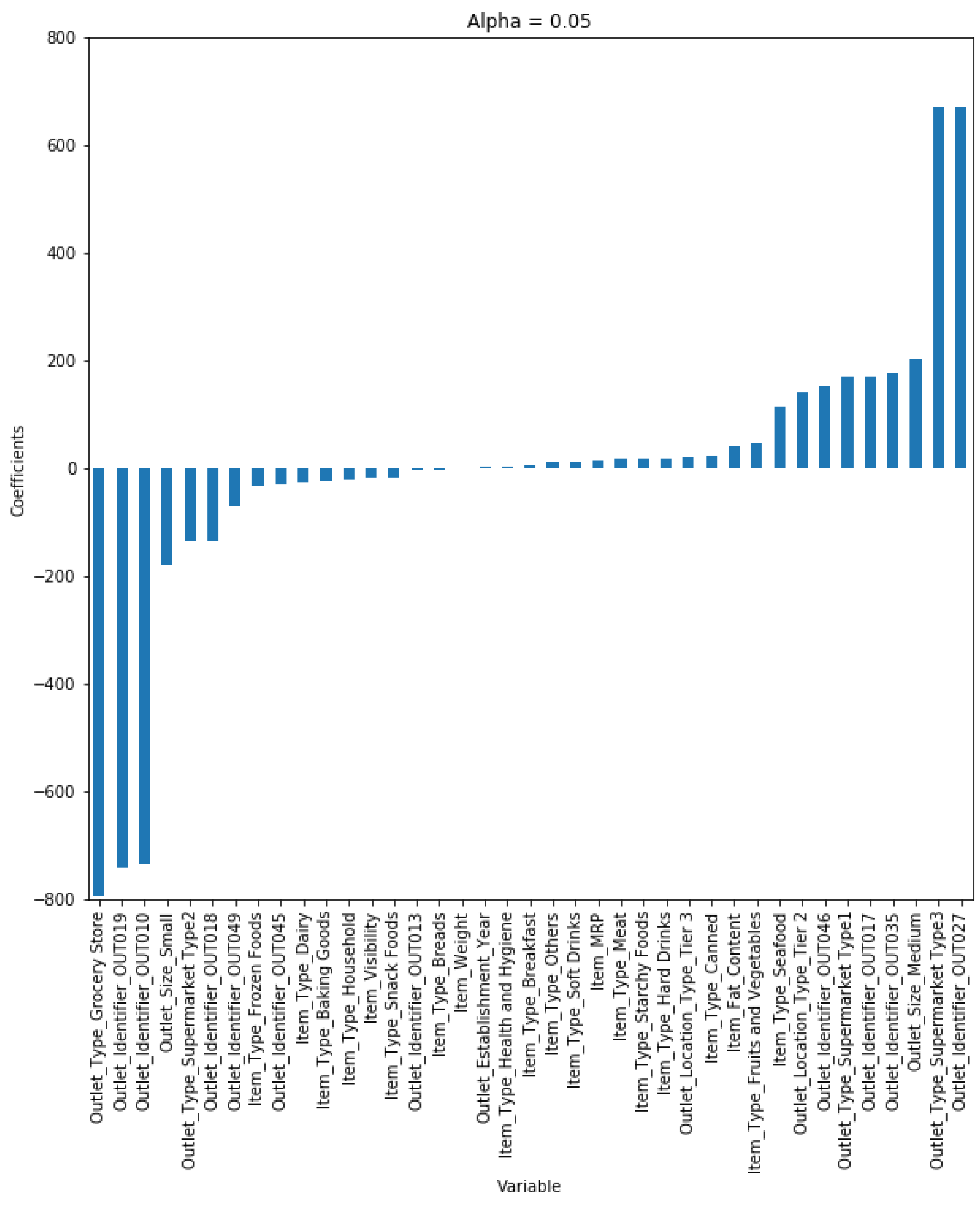
Let us first implement it on our above problem and check our results that whether it performs better than our linear regression model.

```
from sklearn.linear_model import Ridge

## training the model

ridgeReg = Ridge(alpha=0.05, normalize=True)

ridgeReg.fit(x_train,y_train)

pred = ridgeReg.predict(x_cv)

calculating mse

mse = np.mean((pred_cv - y_cv)**2)

mse 1348171.96 ## calculating score ridgeReg.score(x_cv,y_cv) 0.5691
```

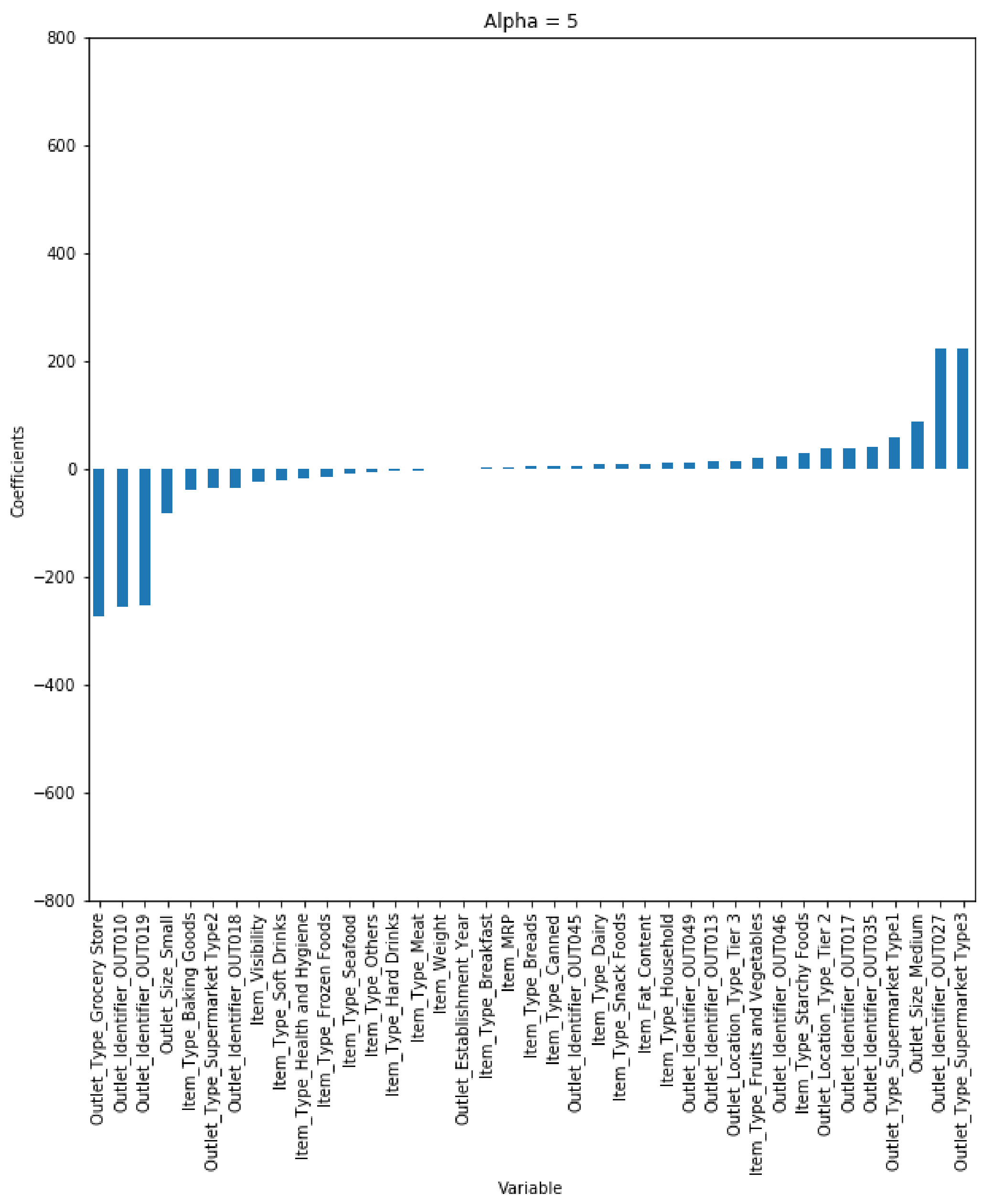
So, we can see that there is a slight improvement in our model because the value of the R-Square has been increased. Note that value of alpha, which is hyperparameter of Ridge, which means that they are not automatically learned by the model instead they have to be set manually.

Here we have consider alpha = 0.05. But let us consider different values of alpha and plot the coefficients for each case.

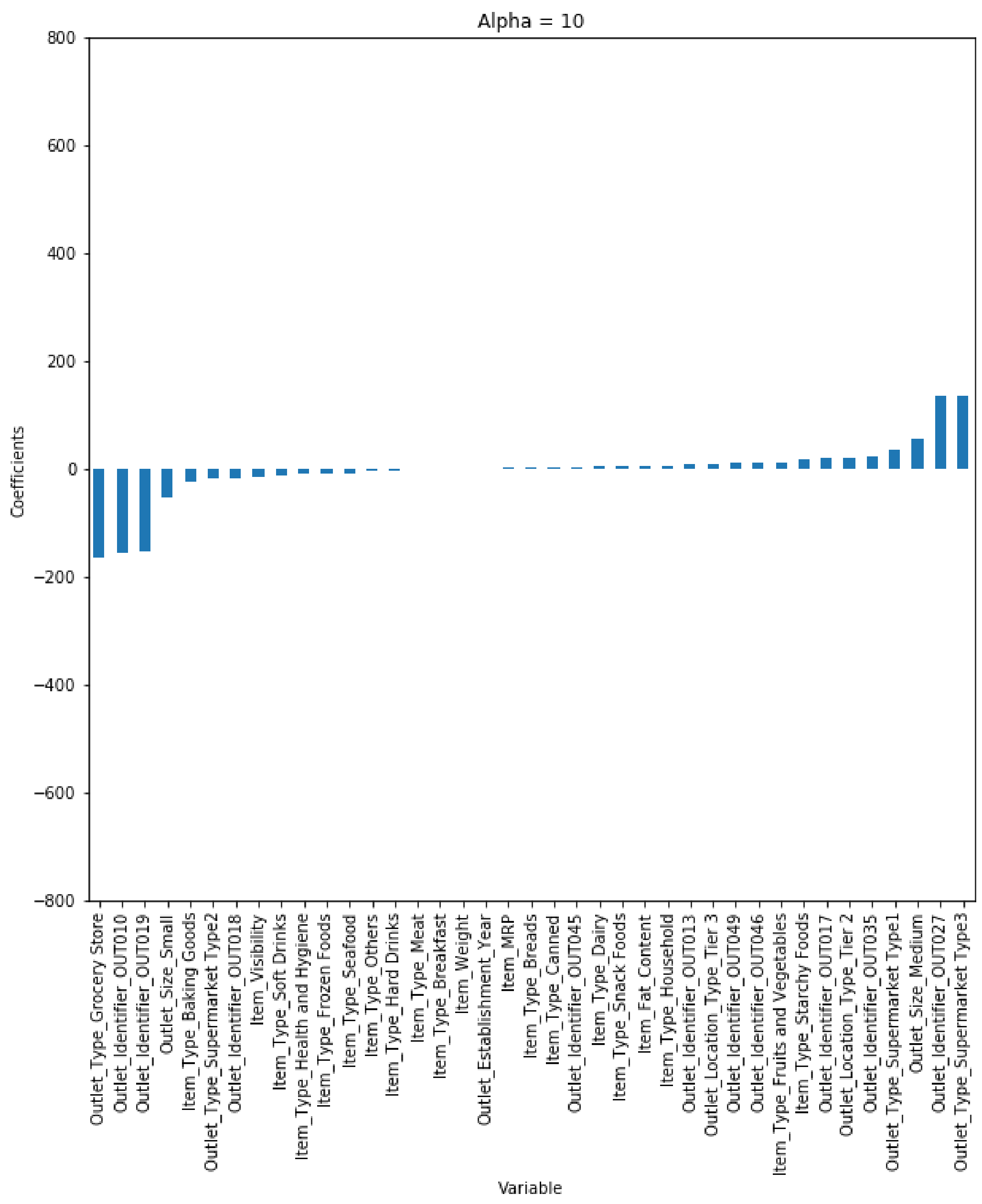

Alpha = 0.5
Coefficients
Variable
800
600
400
200
0
−200
−400
−600
−800
Outlet_Type_Grocery Store
Outlet_Identifier_OUT019
Outlet_Identifier_OUT010
Outlet_Size_Small
Outlet_Identifier_OUT018
Outlet_Type_Supermarket Type2
Item_Type_Baking Goods
Outlet_Identifier_OUT049
Item_Type_Frozen Foods
Item_Visibility
Item_Type_Health and Hygiene
Item_Type_Soft Drinks
Outlet_Identifier_OUT045
Item_Type_Others
Item_Type_Hard Drinks
Item_Weight
Outlet_Establishment_Year
Item_Type_Breads
Item_Type_Dairy
Item_Type_Snack Foods
Item_Type_Meat
Outlet_Identifier_OUT013
Item_Type_Household
Item_Type_Breakfast
Item_MRP
Item_Type_Canned
Outlet_Location_Type_Tier 3
Item_Fat_Content
Item_Type_Starchy Foods
Item_Type_Seafood
Item_Type_Fruits and Vegetables
Outlet_Identifier_OUT046
Outlet_Location_Type_Tier 2
Outlet_Identifier_OUT017
Outlet_Identifier_OUT035
Outlet_Type_Supermarket Type1
Outlet_Size_Medium
Outlet_Type_Supermarket Type3
Outlet_Identifier_OUT027



Alpha = 0.05
800
600
400
200
0
−200
−400
−600
−800
Coefficients
Variable
Outlet_Type_Grocery Store
Outlet_Identifier_OUT019
Outlet_Identifier_OUT010
Outlet_Size_Small
Outlet_Type_Supermarket Type2
Outlet_Identifier_OUT018
Outlet_Identifier_OUT049
Item_Type_Frozen Foods
Outlet_Identifier_OUT045
Item_Type_Dairy
Item_Type_Baking Goods
Item_Type_Household
Item_Visibility
Item_Type_Snack Foods
Outlet_Identifier_OUT013
Item_Type_Breads
Item_Weight
Outlet_Establishment_Year
Item_Type_Health and Hygiene
Item_Type_Breakfast
Item_Type_Others
Item_Type_Soft Drinks
Item_MRP
Item_Type_Meat
Item_Type_Starchy Foods
Item_Type_Hard Drinks
Outlet_Location_Type_Tier 3
Item_Type_Canned
Item_Fat_Content
Item_Type_Fruits and Vegetables
Item_Type_Seafood
Outlet_Location_Type_Tier 2
Outlet_Identifier_OUT046
Outlet_Type_Supermarket Type1
Outlet_Identifier_OUT017
Outlet_Identifier_OUT035
Outlet_Size_Medium
Outlet_Type_Supermarket Type3
Outlet_Identifier_OUT027



Alpha = 5
Coefficients
Variable
800
600
400
200
0
−200
−400
−600
−800
Outlet_Type_Grocery Store
Outlet_Identifier_OUT010
Outlet_Identifier_OUT019
Outlet_Size_Small
Item_Type_Baking Goods
Outlet_Type_Supermarket Type2
Outlet_Identifier_OUT018
Item_Visibility
Item_Type_Soft Drinks
Item_Type_Health and Hygiene
Item_Type_Frozen Foods
Item_Type_Seafood
Item_Type_Others
Item_Type_Hard Drinks
Item_Type_Meat
Item_Weight
Outlet_Establishment_Year
Item_Type_Breakfast
Item_MRP
Item_Type_Breads
Item_Type_Canned
Outlet_Identifier_OUT045
Item_Type_Dairy
Item_Type_Snack Foods
Item_Fat_Content
Item_Type_Household
Outlet_Identifier_OUT049
Outlet_Identifier_OUT013
Outlet_Location_Type_Tier 3
Item_Type_Fruits and Vegetables
Outlet_Identifier_OUT046
Item_Type_Starchy Foods
Outlet_Location_Type_Tier 2
Outlet_Identifier_OUT017
Outlet_Identifier_OUT035
Outlet_Type_Supermarket Type1
Outlet_Size_Medium
Outlet_Identifier_OUT027
Outlet_Type_Supermarket Type3



You can see that, as we increase the value of alpha, the magnitude of the coefficients decreases, where the values reaches to zero but not absolute zero.

But if you calculate R-square for each alpha, we will see that the value of R-square will be maximum at alpha=0.05. So we have to choose it wisely by iterating it through a range of values and using the one which gives us lowest error.

So, now you have an idea how to implement it but let us take a look at the mathematics side also. Till now our idea was to basically minimize the cost function, such that values predicted are much closer to the desired result.

Now take a look back again at the cost function for ridge regression.

$$\min\left(||Y - X(\theta)||_2^2 + \lambda||\theta||_2^2\right)$$

Here if you notice, we come across an extra term, which is known as the penalty term. λ given here, is actually denoted by alpha parameter in the ridge function. So by changing the values of alpha, we are basically controlling the penalty term. Higher the values of alpha, bigger is the penalty and therefore the magnitude of coefficients are reduced.

Important Points:

- It shrinks the parameters, therefore it is mostly used to prevent multicollinearity.
- It reduces the model complexity by coefficient shrinkage.
- It uses L2 regularization technique. (which I will discussed later in this article)

Now let us consider another type of regression technique which also makes use of regularization.

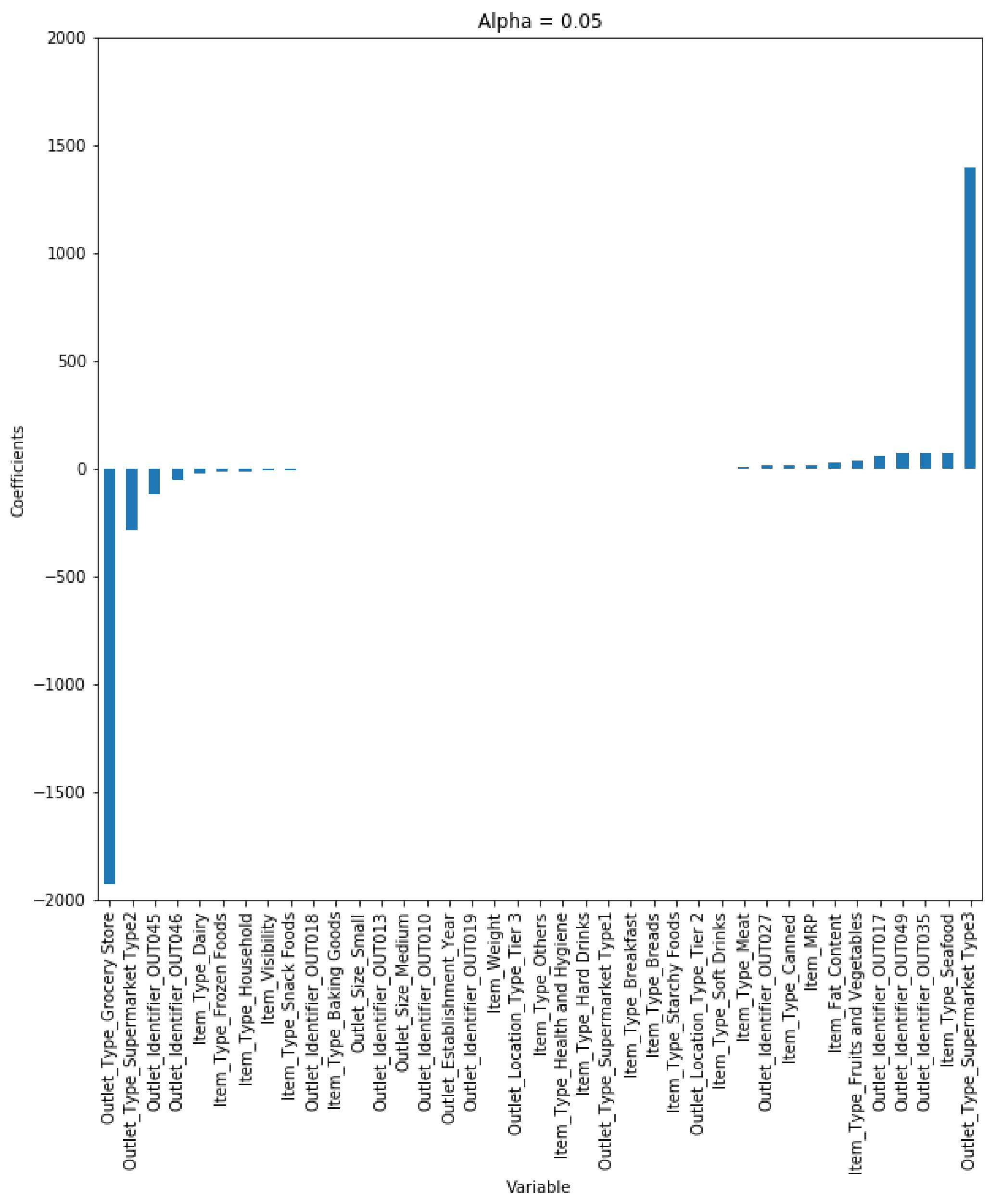
12. Lasso regression

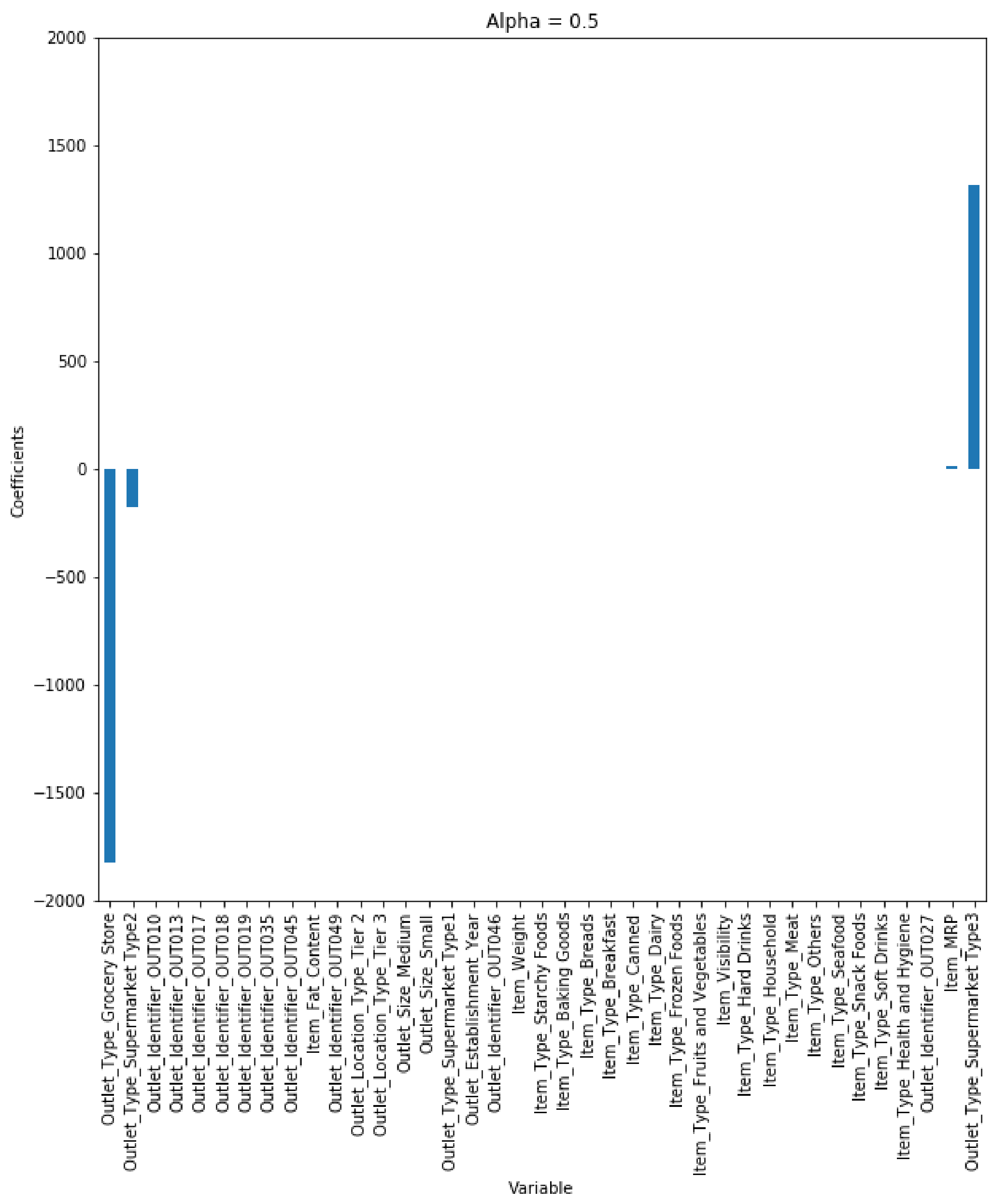
LASSO (Least Absolute Shrinkage Selector Operator), is quite similar to ridge, but lets understand the difference them by implementing it in our big mart problem.

```
from sklearn.linear_model import Lasso

lassoReg = Lasso(alpha=0.3, normalize=True)

lassoReg.fit(x_train,y_train)

pred = lassoReg.predict(x_cv)

# calculating mse

mse = np.mean((pred_cv - y_cv)**2)

mse

1346205.82

lassoReg.score(x_cv,y_cv)

0.5720
```

As we can see that, both the mse and the value of R-square for our model has been increased. Therefore, lasso model is predicting better than both linear and ridge.

Again lets change the value of alpha and see how does it affect the coefficients.


Alpha = 0.05
Coefficients
Variable
2000
1500
1000
500
0
−500
−1000
−1500
−2000
Outlet_Type_Grocery Store
Outlet_Type_Supermarket Type2
Outlet_Identifier_OUT045
Outlet_Identifier_OUT046
Item_Type_Dairy
Item_Type_Frozen Foods
Item_Type_Household
Item_Visibility
Item_Type_Snack Foods
Outlet_Identifier_OUT018
Item_Type_Baking Goods
Outlet_Size_Small
Outlet_Identifier_OUT013
Outlet_Size_Medium
Outlet_Identifier_OUT010
Outlet_Establishment_Year
Outlet_Identifier_OUT019
Item_Weight
Outlet_Location_Type_Tier 3
Item_Type_Others
Item_Type_Health and Hygiene
Item_Type_Hard Drinks
Outlet_Type_Supermarket Type1
Item_Type_Breakfast
Item_Type_Breads
Item_Type_Starchy Foods
Outlet_Location_Type_Tier 2
Item_Type_Soft Drinks
Item_Type_Meat
Outlet_Identifier_OUT027
Item_Type_Canned
Item_MRP
Item_Fat_Content
Item_Type_Fruits and Vegetables
Outlet_Identifier_OUT017
Outlet_Identifier_OUT049
Outlet_Identifier_OUT035
Item_Type_Seafood
Outlet_Type_Supermarket Type3



So, we can see that even at small values of alpha, the magnitude of coefficients have reduced a lot. By looking at the plots, can you figure a difference between ridge and lasso?

We can see that as we increased the value of alpha, coefficients were approaching towards zero, but if you see in case of lasso, even at smaller alpha's, our coefficients are reducing to absolute zeroes. Therefore, lasso selects the only some feature while reduces the coefficients of others to zero. This property is known as feature selection and which is absent in case of ridge.

Mathematics behind lasso regression is quiet similar to that of ridge only difference being instead of adding squares of theta, we will add absolute value of Θ.

$$\min\left(||Y - X\theta||_2^2 + \lambda||\theta||_1\right)$$

Here too, λ is the hypermeter, whose value is equal to the alpha in the Lasso function.

Important Points:

- It uses L1 regularization technique (will be discussed later in this article)
- It is generally used when we have more number of features, because it automatically does feature selection.

Now that you have a basic understanding of ridge and lasso regression, let's think of an example where we have a large dataset, lets say it has 10,000 features. And we know that some of the independent features are correlated with other independent features. Then think, which regression would you use, Rigde or Lasso?

Let's discuss it one by one. If we apply ridge regression to it, it will retain all of the features but will shrink the coefficients. But the problem is that model will still remain complex as there are 10,000 features, thus may lead to poor model performance.

Instead of ridge what if we apply lasso regression to this problem. The main problem with lasso regression is when we have correlated variables, it retains only one variable and sets other correlated variables to zero. That will possibly lead to some loss of information resulting in lower accuracy in our model.

Then what is the solution for this problem? Actually we have another type of regression, known as elastic net regression, which is basically a hybrid of ridge and lasso regression. So let's try to understand it.

13. Elastic Net Regression

Before going into the theory part, let us implement this too in big mart sales problem. Will it perform better than ridge and lasso? Let's check!

```
from sklearn.linear_model import ElasticNet

ENreg = ElasticNet(alpha=1, l1_ratio=0.5, normalize=False)

ENreg.fit(x_train,y_train)

pred_cv = ENreg.predict(x_cv)
```

```
#calculating mse

mse = np.mean((pred_cv - y_cv)**2)

mse 1773750.73

ENreg.score(x_cv,y_cv)

0.4504
```

So we get the value of R-Square, which is very less than both ridge and lasso. Can you think why? The reason behind this downfall is basically we didn’t have a large set of features. Elastic regression generally works well when we have a big dataset.

Note, here we had two parameters alpha and l1_ratio. First let’s discuss, what happens in elastic net, and how it is different from ridge and lasso.

Elastic net is basically a combination of both L1 and L2 regularization. So if you know elastic net, you can implement both Ridge and Lasso by tuning the parameters. So it uses both L1 and L2 penality term, therefore its equation look like as follows:

$$\min\left(\left||Y - X\theta|\right|_2^2 + \lambda_1||\theta||_1 + \lambda_2||\theta||_2^2\right)$$

So how do we adjust the lambdas in order to control the L1 and L2 penalty term? Let us understand by an example. You are trying to catch a fish from a pond. And you only have a net, then what would you do? Will you randomly throw your net? No, you will actually wait until you see one fish swimming around, then you would throw the net in that direction to basically collect the entire group of fishes. Therefore even if they are correlated, we still want to look at their entire group.

Elastic regression works in a similar way. Let’ say, we have a bunch of correlated independent variables in a dataset, then elastic net will simply form a group consisting of these correlated variables. Now if any one of the variable of this group is a strong predictor (meaning having a strong relationship with dependent variable), then we will include the entire group in the model building, because omitting other variables (like what we did in lasso) might result in losing some information in terms of interpretation ability, leading to a poor model performance.

So, if you look at the code above, we need to define alpha and l1_ratio while defining the model. Alpha and l1_ratio are the parameters which you can set accordingly if you wish to control the L1 and L2 penalty separately. Actually, we have

Alpha = a + b and l1_ratio = a / (a+b)

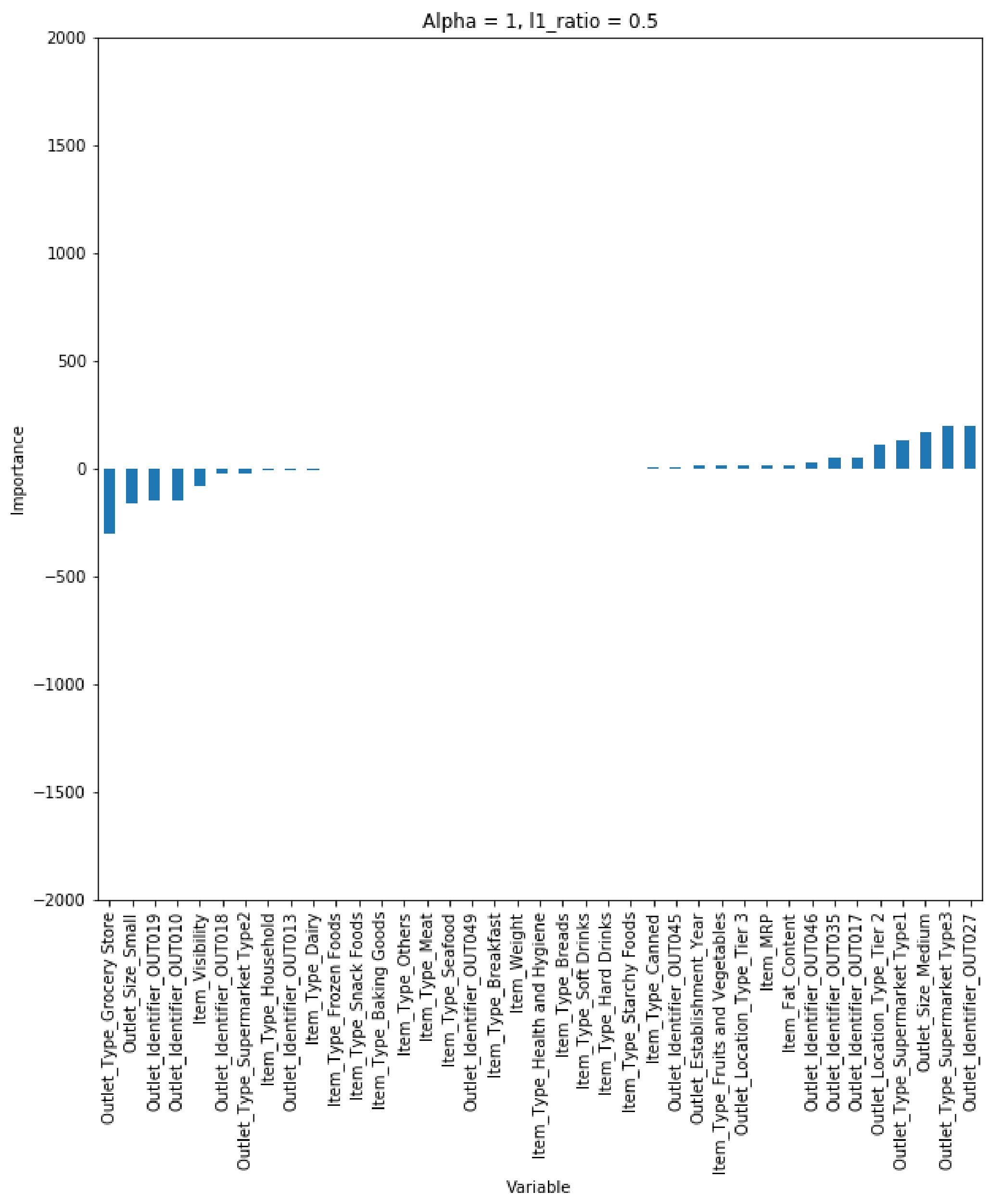
where, a and b weights assigned to L1 and L2 term respectively. So when we change the values of alpha and l1_ratio, a and b are set aaccordingly such that they control trade off between L1 and L2 as:

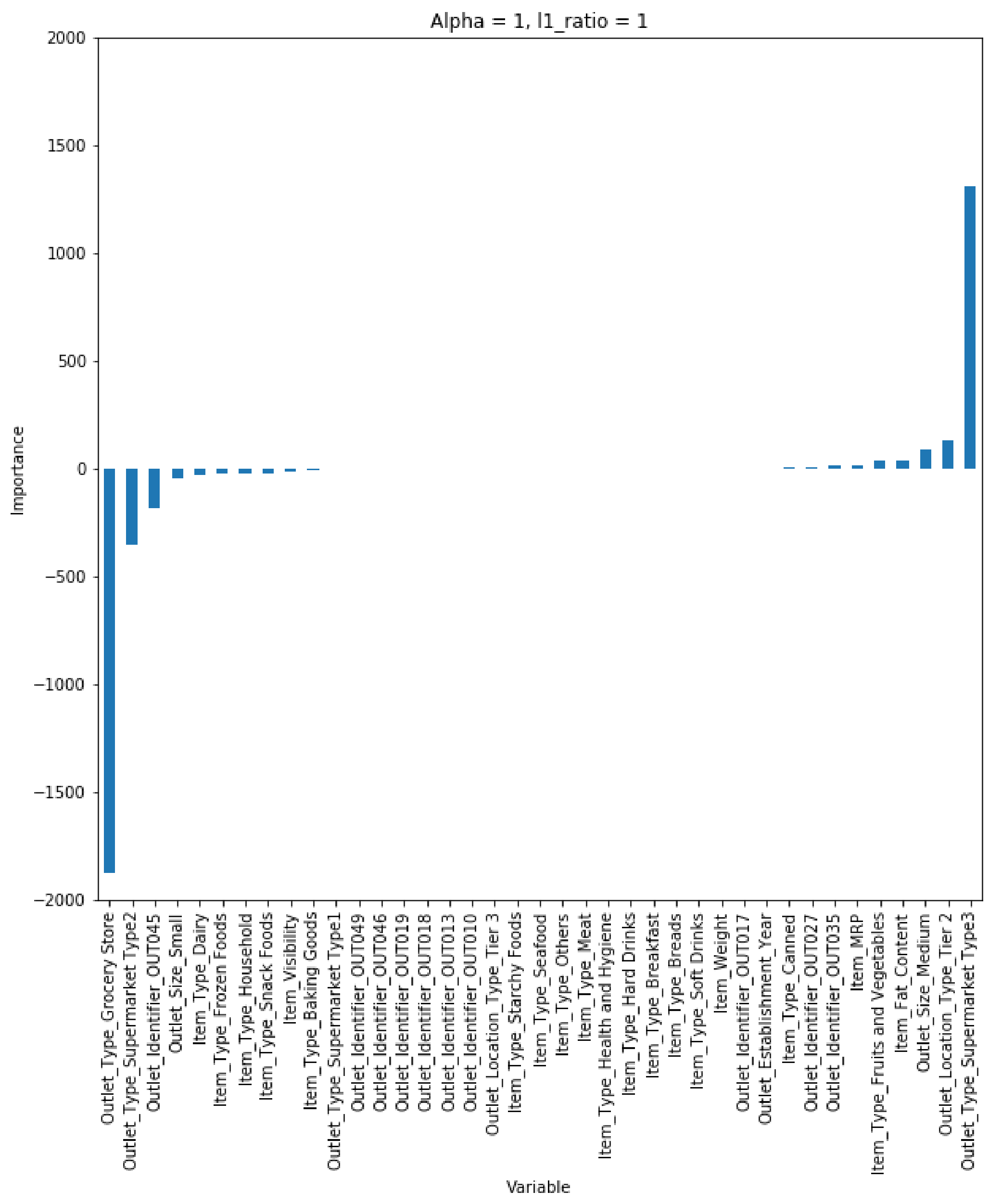
a * (L1 term) + b* (L2 term)

Let alpha (or a+b) = 1, and now consider the following cases:

- If l1_ratio =1, therefore if we look at the formula of l1_ratio, we can see that l1_ratio can only be equal to 1 if a=1, which implies b=0. Therefore, it will be a lasso penalty.
- Similarly if l1_ratio = 0, implies a=0. Then the penalty will be a ridge penalty.
- For l1_ratio between 0 and 1, the penalty is the combination of ridge and lasso.

So let us adjust alpha and l1_ratio, and try to understand from the plots of coefficient given below.


Alpha = 1, l1_ratio = 0.5
Importance
Variable
2000
1500
1000
500
0
−500
−1000
−1500
−2000
Outlet_Type_Grocery Store
Outlet_Size_Small
Outlet_Identifier_OUT019
Outlet_Identifier_OUT010
Item_Visibility
Outlet_Identifier_OUT018
Outlet_Type_Supermarket Type2
Item_Type_Household
Outlet_Identifier_OUT013
Item_Type_Dairy
Item_Type_Frozen Foods
Item_Type_Snack Foods
Item_Type_Baking Goods
Item_Type_Others
Item_Type_Meat
Item_Type_Seafood
Outlet_Identifier_OUT049
Item_Type_Breakfast
Item_Weight
Item_Type_Health and Hygiene
Item_Type_Breads
Item_Type_Soft Drinks
Item_Type_Hard Drinks
Item_Type_Starchy Foods
Item_Type_Canned
Outlet_Identifier_OUT045
Outlet_Establishment_Year
Item_Type_Fruits and Vegetables
Outlet_Location_Type_Tier 3
Item_MRP
Item_Fat_Content
Outlet_Identifier_OUT046
Outlet_Identifier_OUT035
Outlet_Identifier_OUT017
Outlet_Location_Type_Tier 2
Outlet_Type_Supermarket Type1
Outlet_Size_Medium
Outlet_Type_Supermarket Type3
Outlet_Identifier_OUT027



Now, you have basic understanding about ridge, lasso and elasticnet regression. But during this, we came across two terms L1 and L2, which are basically two types of regularization. To sum up basically lasso and ridge are the direct application of L1 and L2 regularization respectively.

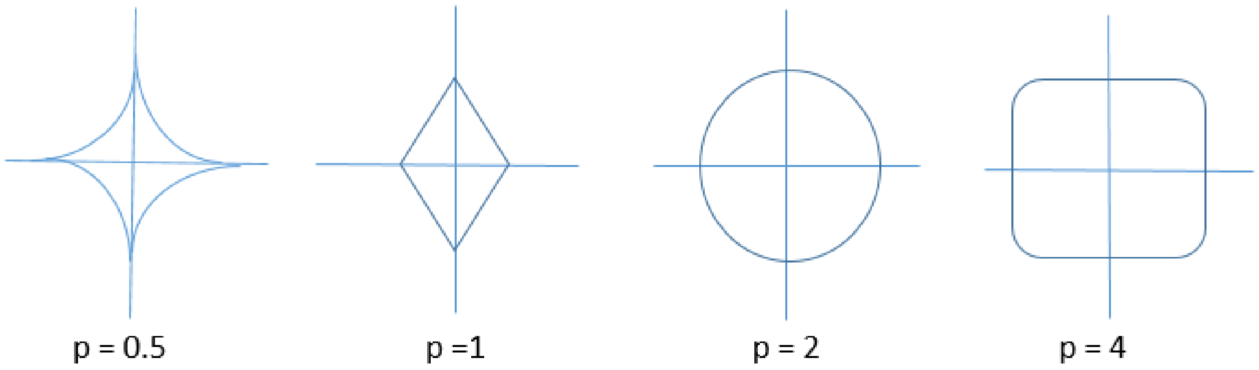
But if you still want to know, below I have explained the concept behind them, which is OPTIONAL.

14. Types of Regularization Techniques [Optional]

Let's recall, both in ridge and lasso we added a penalty term, but the term was different in both cases. In ridge, we used the squares of theta while in lasso we used absolute value of theta. So why these two only, can't there be other possibilities?

Actually, there are different possible choices of regularization with different choices of order of the parameter in the regularization term, which is denoted by $\sum_i |\theta_i|^p$. This is more generally known as L_p regularizer.

Let us try to visualize some by plotting them. For making visualization easy, let us plot them in 2D space. For that we suppose that we just have two parameters. Now, let's say if p=1, we have term as $\sum_i |\theta_i|^p = |\theta_1| + |\theta_2|$. Can't we plot this equation of line? Similarly plot for different values of p are given below.

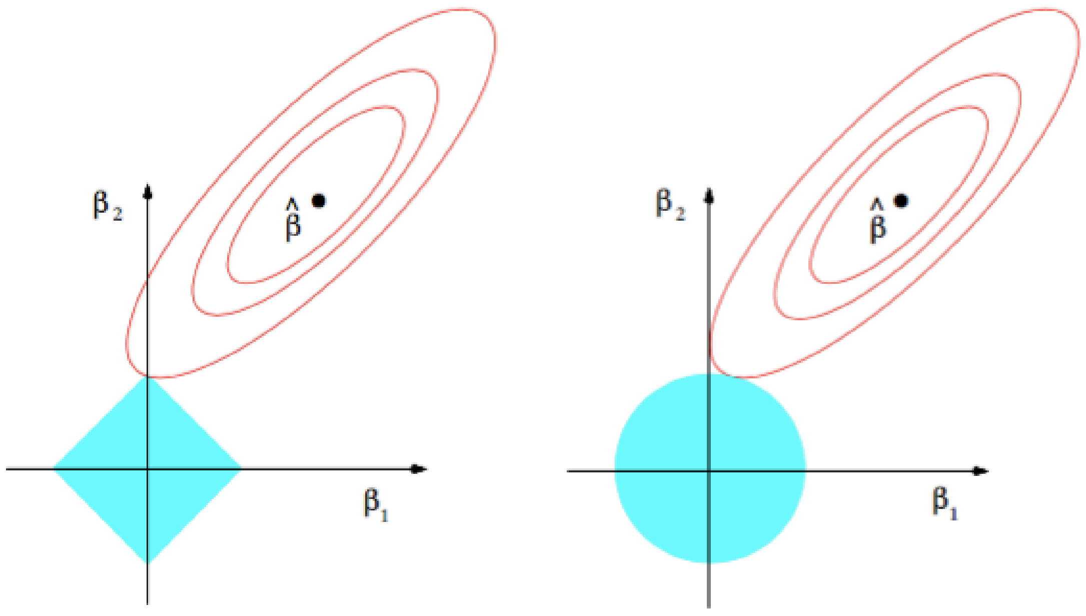


In the above plots, axis denote the parameters($\Theta_{1 \text{ and}}$ Θ2). Let us examine them one by one.

For p=0.5, we can only get large values of one parameter only if other parameter is too small. For p=1, we get sum of absolute values where the increase in one parameter Θ is exactly offset by the decrease in other. For p =2, we get a circle and for larger p values, it approaches a round square shape.

The two most commonly used regularization are in which we have p=1 and p=2, more commonly known as L1 and L2 regularization.

Look at the figure given below carefully. The blue shape refers the regularization term and other shape present refers to our least square error (or data term).



The first figure is for L1 and the second one is for L2 regularization. The black point denotes that the least square error is minimized at that point and as we can see that it increases quadratically as we move from it and the regularization term is minimized at the origin where all the parameters are zero .

Now the question is that at what point will our cost function be minimum? The answer will be, since they are quadratically increasing, the sum of both the terms will be minimized at the point where they first intersect.

Take a look at the L2 regularization curve. Since the shape formed by L2 regularizer is a circle, it increases quadratically as we move away from it. The L2 optimum(which is basically the intersection point) can fall on the axis lines only when the minimum MSE (mean square error or the black point in the figure) is also exactly on the axis. But in case of L1, the L1 optimum can be on the axis line because its contour is sharp and therefore there are high chances of interaction point to fall on axis. Therefore it is possible to intersect on the axis line, even when minimum MSE is not on the axis. If the intersection point falls on the axes it is known as sparse.

Therefore L1 offers some level of sparsity which makes our model more efficient to store and compute and it can also help in checking importance of feature, since the features that are not important can be exactly set to zero.

----

9. Dimensionality Reduction Algorithms

In the last 4-5 years, there has been an exponential increase in data capturing at every possible stages. Corporates/ Government Agencies/ Research organisations are not only coming with new sources but also they are capturing data in great detail.

For example: E-commerce companies are capturing more details about customer like their demographics, web crawling history, what they like or dislike, purchase history, feedback and many others to give them personalized attention more than your nearest grocery shopkeeper.

As a data scientist, the data we are offered also consist of many features, this sounds good for building good robust model but there is a challenge. How’d you identify highly significant variable(s) out 1000 or 2000? In such cases, dimensionality reduction algorithm helps us along with various other algorithms like Decision Tree, Random Forest, PCA, Factor Analysis, Identify based on correlation matrix, missing value ratio and others.

To know more about this algorithms, you can read “Beginners Guide To Learn Dimension Reduction Techniques“.

Python Code

```
#Import Library


from sklearn import decomposition


#Assumed you have training and test data set as train and test


# Create PCA obeject pca= decomposition.PCA(n_components=k) #default value of k =m

in(n_sample, n_features)


# For Factor analysis


#fa= decomposition.FactorAnalysis()


# Reduced the dimension of training dataset using PCA


train_reduced = pca.fit_transform(train)


#Reduced the dimension of test dataset


test_reduced = pca.transform(test)
```

```
#For more detail on this, please refer  this link.
```

R Code

```
library(stats)


pca <- princomp(train, cor = TRUE)


train_reduced  <- predict(pca,train)


test_reduced  <- predict(pca,test)
```

10. Gradient Boosting Algorithms

10.1. GBM

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to a build strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix.

More: Know about Boosting algorithms in detail

Python Code

```
#Import Library

from sklearn.ensemble import GradientBoostingClassifier


#Assumed you have, X (predictor) and Y (target) for training data set and x_test(p

redictor) of test_dataset


# Create Gradient Boosting Classifier object
```

```
model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1
, random_state=0)

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

R Code

```
library(caret)

x <- cbind(x_train,y_train)

# Fitting model
fitControl <- trainControl( method = "repeatedcv", number = 4, repeats = 4)

fit <- train(y ~ ., data = x, method = "gbm", trControl = fitControl,verbose = FAL
SE)

predicted= predict(fit,x_test,type= "prob")[,2]
```

GradientBoostingClassifier and Random Forest are two different boosting tree classifier and often people ask about the difference between these two algorithms.

10.2. XGBoost

Another classic gradient boosting algorithm that’s known to be the decisive choice between winning and losing in some Kaggle competitions.

The XGBoost has an immensely high predictive power which makes it the best choice for accuracy in events as it possesses both linear model and the tree learning algorithm, making the algorithm almost 10x faster than existing gradient booster techniques.

The support includes various objective functions, including regression, classification and ranking.

One of the most interesting things about the XGBoost is that it is also called a regularized boosting technique. This helps to reduce overfit modelling and has a massive support for a range of languages such as Scala, Java, R, Python, Julia and C++.

Supports distributed and widespread training on many machines that encompass GCE, AWS, Azure and Yarn clusters. XGBoost can also be integrated with Spark, Flink and other cloud dataflow systems with a built in cross validation at each iteration of the boosting process.

To learn more about XGBoost and parameter tuning, visit https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/.

Python Code:

```
from xgboost import XGBClassifier


from sklearn.model_selection import train_test_split


from sklearn.metrics import accuracy_score


X = dataset[:,0:10]


Y = dataset[:,10:]


seed = 1




X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_s

tate=seed)
```

```
model = XGBClassifier()



model.fit(X_train, y_train)



#Make predictions for test data

y_pred = model.predict(X_test)
```

R Code:

```
require(caret)



x <- cbind(x_train,y_train)



# Fitting model



TrainControl <- trainControl( method = "repeatedcv", number = 10, repeats = 4)
```

```
model<- train(y ~ ., data = x, method = "xgbLinear", trControl = TrainControl,verb

ose = FALSE)




OR




model<- train(y ~ ., data = x, method = "xgbTree", trControl = TrainControl,verbos

e = FALSE)




predicted <- predict(model, x_test)
```

10.3. LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

The framework is a fast and high-performance gradient boosting one based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. It was developed under the Distributed Machine Learning Toolkit Project of Microsoft.

Since the LightGBM is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

Also, it is surprisingly very fast, hence the word ‘Light’.

Refer to the article to know more about LightGBM: https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

Python Code:

```
data = np.random.rand(500, 10) # 500 entities, each contains 10 features


label = np.random.randint(2, size=500) # binary target




train_data = lgb.Dataset(data, label=label)


test_data = train_data.create_valid('test.svm')




param = {'num_leaves':31, 'num_trees':100, 'objective':'binary'}


param['metric'] = 'auc'




num_round = 10


bst = lgb.train(param, train_data, num_round, valid_sets=[test_data])




bst.save_model('model.txt')
```

```
# 7 entities, each contains 10 features

data = np.random.rand(7, 10)

ypred = bst.predict(data)
```

R Code:

```
library(RLightGBM)

data(example.binary)

#Parameters



num_iterations <- 100

config <- list(objective = "binary",  metric="binary_logloss,auc", learning_rate = 0.1, num_leaves = 63, tree_learner = "serial", feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, min_data_in_leaf = 50, min_sum_hessian_in_leaf = 5.0)



#Create data handle and booster

handle.data <- lgbm.data.create(x)


```

```
lgbm.data.setField(handle.data, "label", y)



handle.booster <- lgbm.booster.create(handle.data, lapply(config, as.character))



#Train for num_iterations iterations and eval every 5 steps



lgbm.booster.train(handle.booster, num_iterations, 5)



#Predict

pred <- lgbm.booster.predict(handle.booster, x.test)



#Test accuracy

sum(y.test == (y.pred > 0.5)) / length(y.test)



#Save model (can be loaded again via lgbm.booster.load(filename))

lgbm.booster.save(handle.booster, filename = "/tmp/model.txt")
```

If you’re familiar with the Caret package in R, this is another way of implementing the LightGBM.

```
require(caret)


require(RLightGBM)


data(iris)




model <-caretModel.LGBM()




fit <- train(Species ~ ., data = iris, method=model, verbosity = 0)


print(fit)


y.pred <- predict(fit, iris[,1:4])




library(Matrix)


model.sparse <- caretModel.LGBM.sparse()




#Generate a sparse matrix


mat <- Matrix(as.matrix(iris[,1:4]), sparse = T)
```

```
fit <- train(data.frame(idx = 1:nrow(iris)), iris$Species, method = model.sparse,

matrix = mat, verbosity = 0)


print(fit)
```

10.4. Catboost

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google's TensorFlow and Apple's Core ML.

The best part about CatBoost is that it does not require extensive data training like other ML models, and can work on a variety of data formats; not undermining how robust it can be.

Make sure you handle missing data well before you proceed with the implementation.

Catboost can automatically deal with categorical variables without showing the type conversion error, which helps you to focus on tuning your model better rather than sorting out trivial errors.

Learn more about Catboost from this article: https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/

Python Code:

```
import pandas as pd


import numpy as np




from catboost import CatBoostRegressor
```

```
#Read training and testing files

train = pd.read_csv("train.csv")

test = pd.read_csv("test.csv")



#Imputing missing values for both train and test

train.fillna(-999, inplace=True)

test.fillna(-999,inplace=True)



#Creating a training set for modeling and validation set to check model performanc
e

X = train.drop(['Item_Outlet_Sales'], axis=1)

y = train.Item_Outlet_Sales



from sklearn.model_selection import train_test_split



X_train, X_validation, y_train, y_validation = train_test_split(X, y, train_size=0
.7, random_state=1234)
```

```
categorical_features_indices = np.where(X.dtypes != np.float)[0]



#importing library and building model

from catboost import CatBoostRegressormodel=CatBoostRegressor(iterations=50, depth
=3, learning_rate=0.1, loss_function='RMSE')



model.fit(X_train, y_train,cat_features=categorical_features_indices,eval_set=(X_v
alidation, y_validation),plot=True)



submission = pd.DataFrame()



submission['Item_Identifier'] = test['Item_Identifier']

submission['Outlet_Identifier'] = test['Outlet_Identifier']

submission['Item_Outlet_Sales'] = model.predict(test)
```

R Code:

```
set.seed(1)
```

```
require(titanic)



require(caret)



require(catboost)



tt <- titanic::titanic_train[complete.cases(titanic::titanic_train),]



data <- as.data.frame(as.matrix(tt), stringsAsFactors = TRUE)



drop_columns = c("PassengerId", "Survived", "Name", "Ticket", "Cabin")



x <- data[,!(names(data) %in% drop_columns)]y <- data[,c("Survived")]



fit_control <- trainControl(method = "cv", number = 4,classProbs = TRUE)
```

```
grid <- expand.grid(depth = c(4, 6, 8),learning_rate = 0.1,iterations = 100, l2_le
af_reg = 1e-3,            rsm = 0.95, border_count = 64)



report <- train(x, as.factor(make.names(y)),method = catboost.caret,verbose = TRUE
, preProc = NULL,tuneGrid = grid, trControl = fit_control)



print(report)



importance <- varImp(report, scale = FALSE)



print(importance)
```