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.

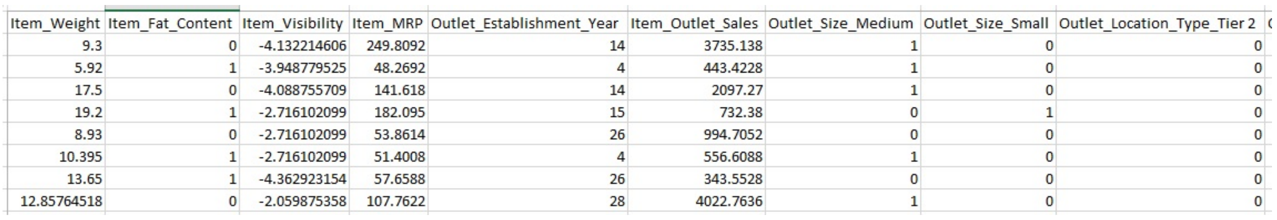
Similarly list down all possible factors you can think of.

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

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.

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

[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:

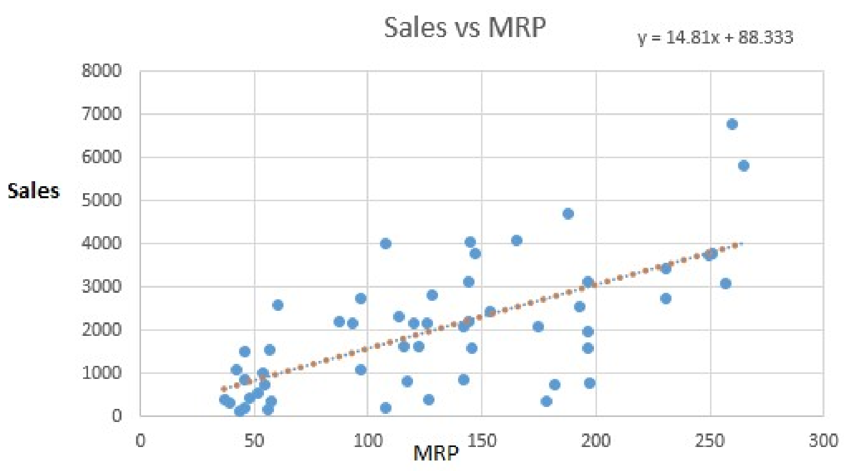
$$Y = \theta_1 X_1 + \theta_2 X_2 + \ldots \theta_n X_n$$

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,

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

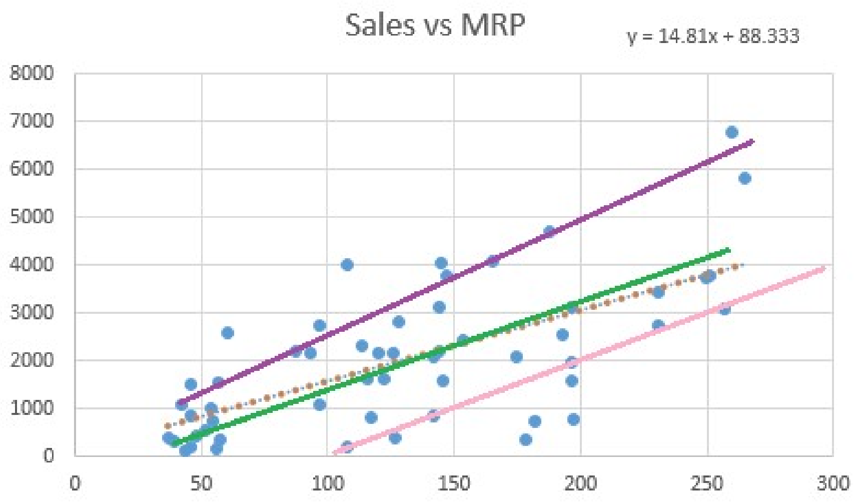
This equation is called a simple linear regression equation, which represents a straight line, where '$\Theta 0$' is the intercept, '$\Theta 1$' is the slope of the line. Take a look at the plot below between sales and MRP.



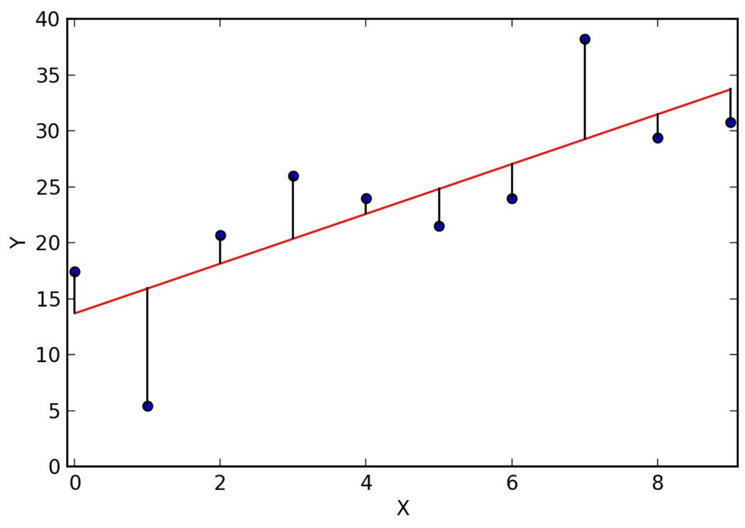
Surprisingly, we can see that sales of a product increases with increase in its MRP. Therefore the dotted red line represents our regression line or the line of best fit. But one question that arises is how you would find out this line?

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 (Σ(Y – h(X)))** – it might result in cancelling out of positive and negative errors.
- **Sum of the absolute value of residuals (Σ|Y-h(X)|)** – absolute value would prevent cancellation of errors
- **Sum of square of residuals (Σ (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) =ϴ1*x +ϴ0 , 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.

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

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.

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.

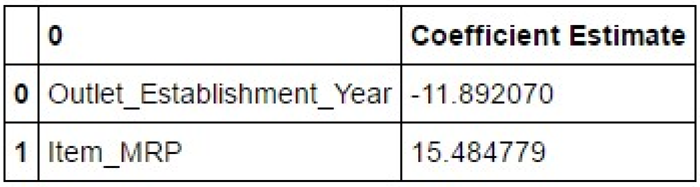
```
# importing basic libraries

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:

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

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’).

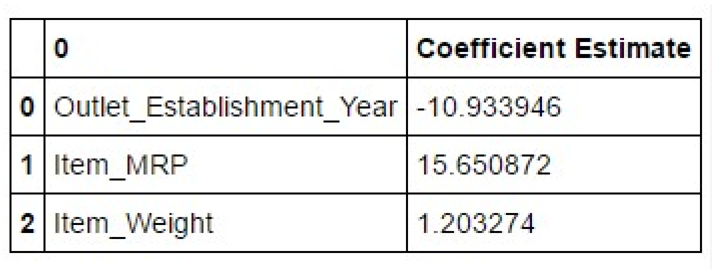
It produces an error, because item weights column have some missing values. So let us impute it with the mean of other non-null entries.

```
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 R2 is that if new predictors (X) are added to our model, R2 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

R2 = 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.

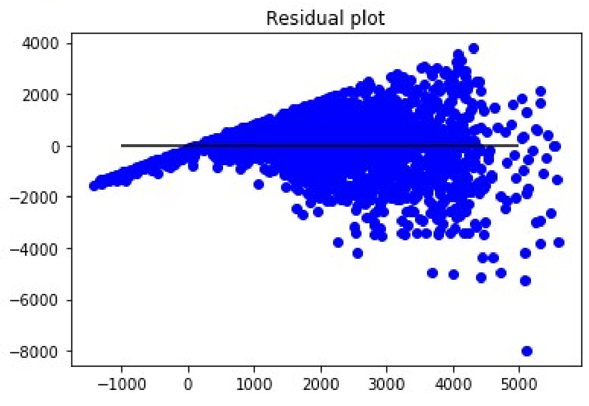
Instead of manually selecting the variables, we can automate this process by using forward or backward selection. Forward selection starts with most significant predictor in the model and adds variable for each step. Backward elimination starts with all predictors in the model and removes the least significant 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.

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

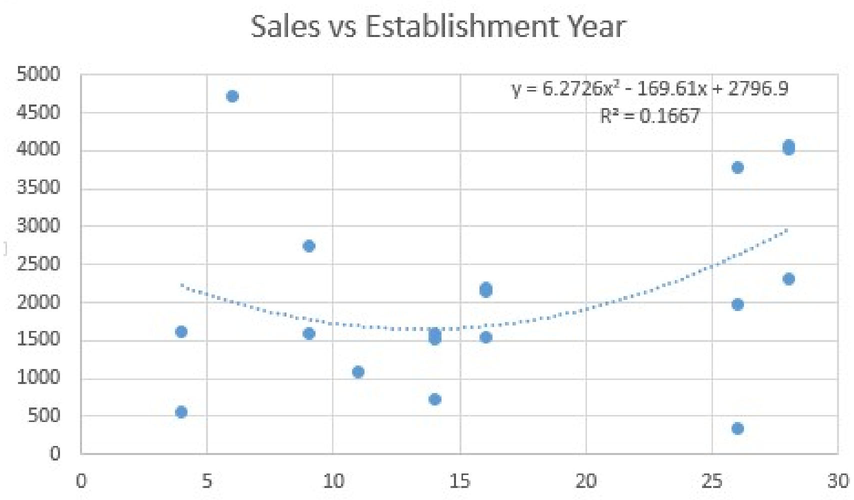
8. Polynomial Regression

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:

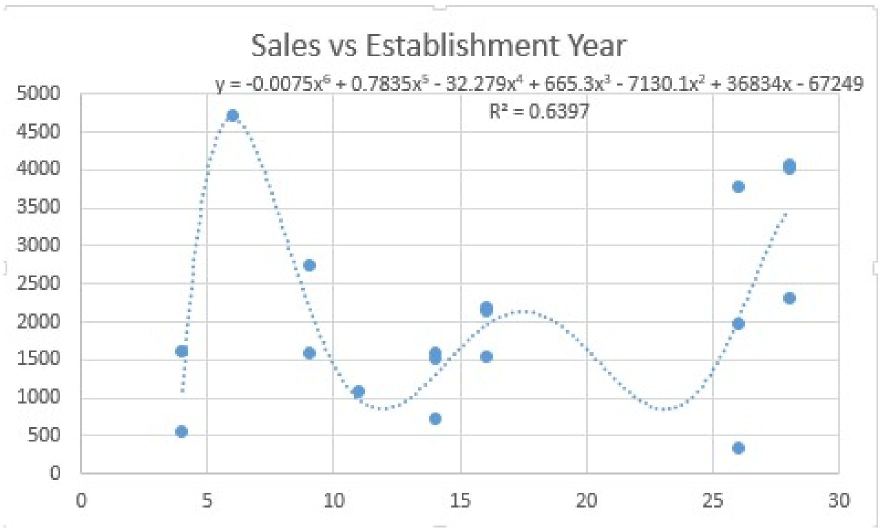
Y =Θ1 +Θ2*x +Θ3*x2

Now take a look at the plot given below.

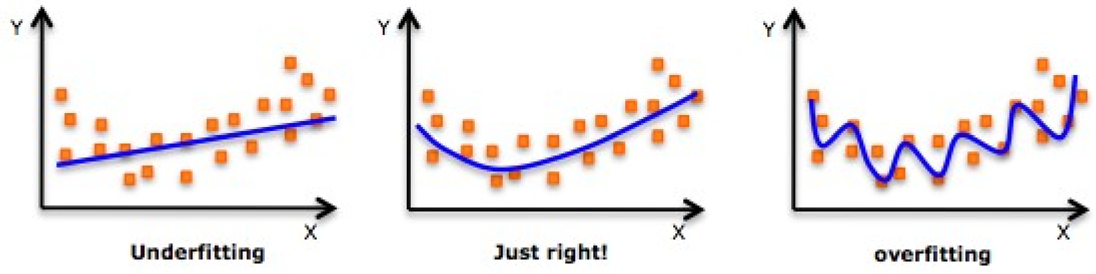


Clearly the quadratic equation fits the data better than simple linear equation. In this case, what do you think will the R-square value of quadratic regression greater than simple linear regression? Definitely yes, because quadratic regression fits the data better than linear regression. While quadratic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

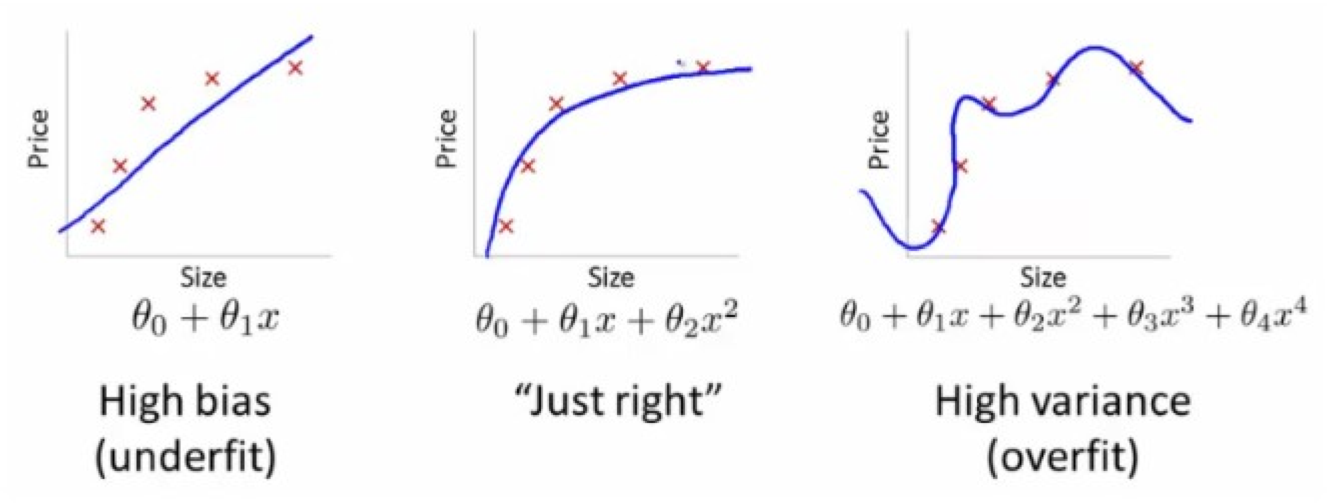
Below figure shows the behavior of a polynomial equation of degree 6.



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.



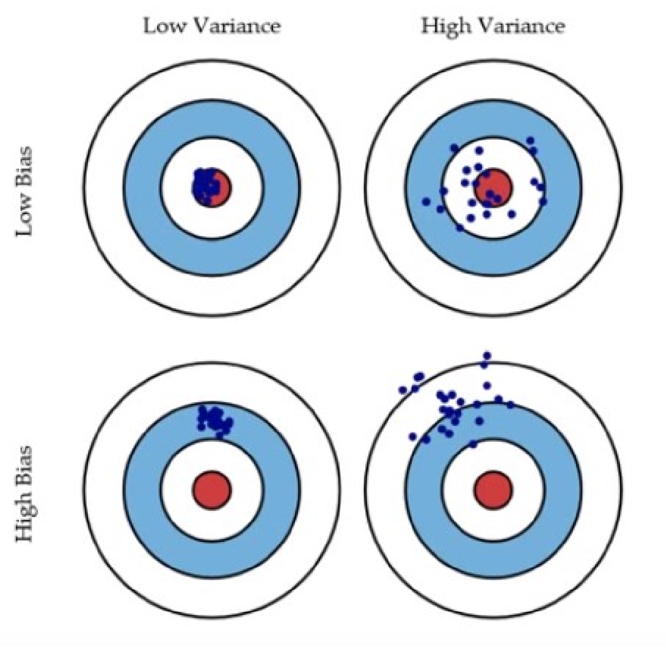
Similarly, we have another problem called **underfitting**, it occurs when our model neither fits the training data nor generalizes on the new data.



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

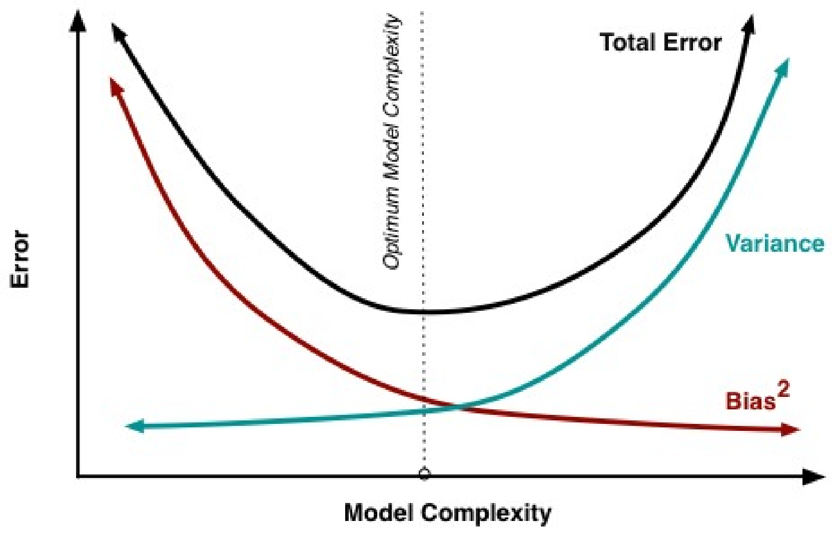
9. Bias and Variance in regression models

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?

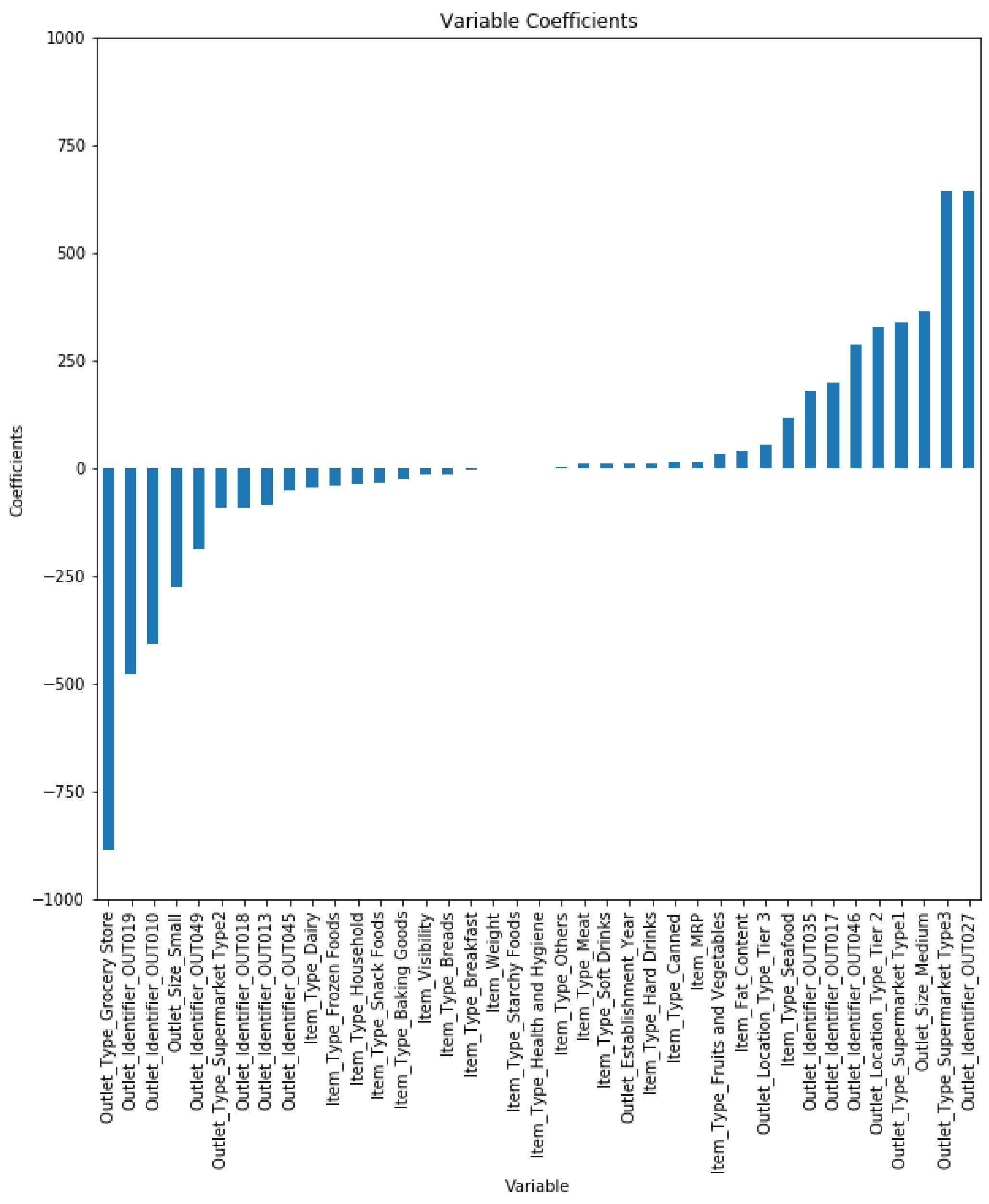
"Everything should be made simple as possible, but not simpler – Albert Einstein"

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.

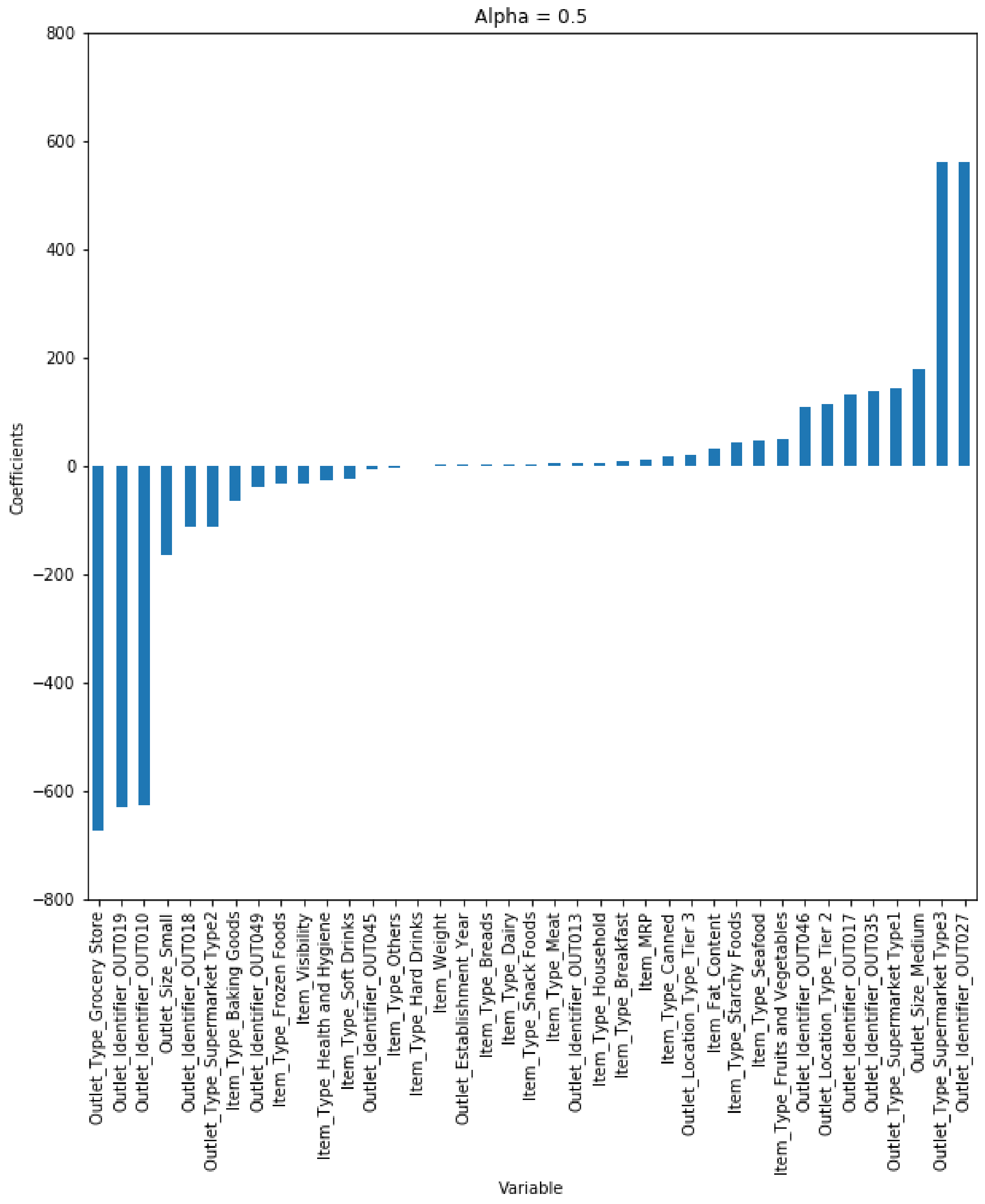
11. Ridge Regression

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

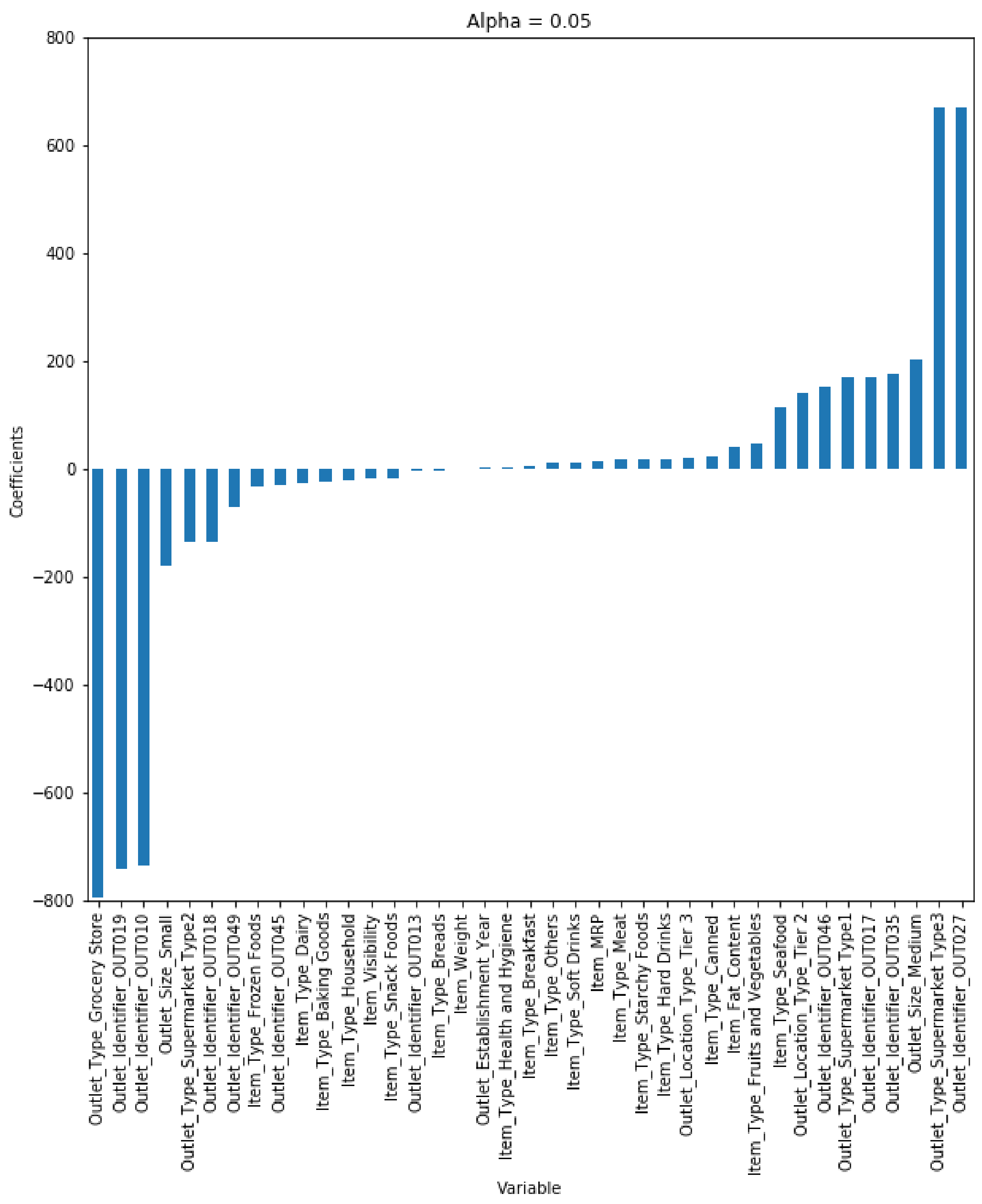
```
from sklearn.linear_model  import Ridge

## 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.

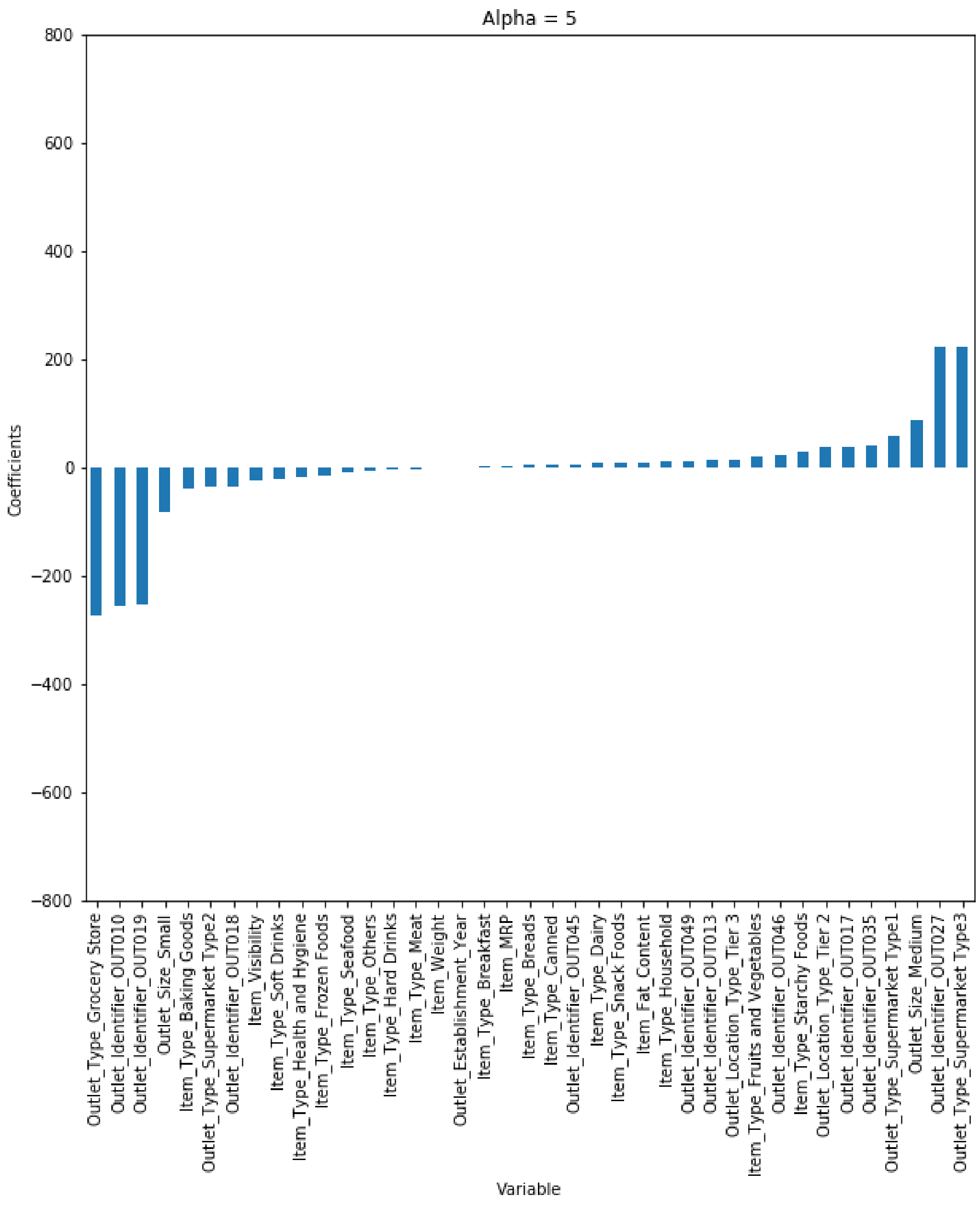
Here we have consider alpha = 0.05. But let us consider different values of alpha and plot the coefficients for each case.



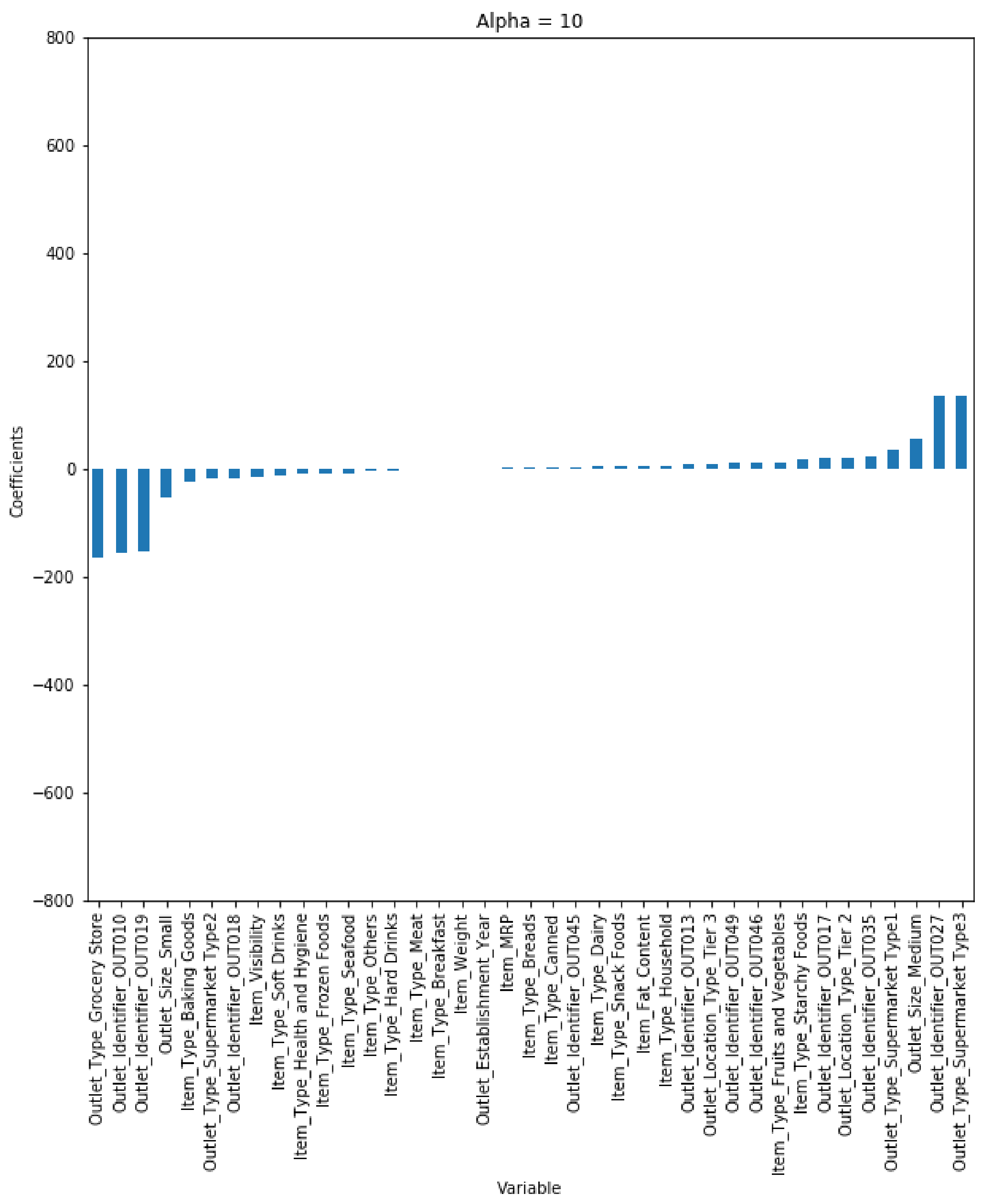
Alpha = 0.5
Coefficients
Variable
800
600
400
200
0
−200
−400
−600
−800
Outlet_Type_Grocery Store
Outlet_Identifier_OUT019
Outlet_Identifier_OUT010
Outlet_Size_Small
Outlet_Identifier_OUT018
Outlet_Type_Supermarket Type2
Item_Type_Baking Goods
Outlet_Identifier_OUT049
Item_Type_Frozen Foods
Item_Visibility
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
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_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 will be discussed later in this article)

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

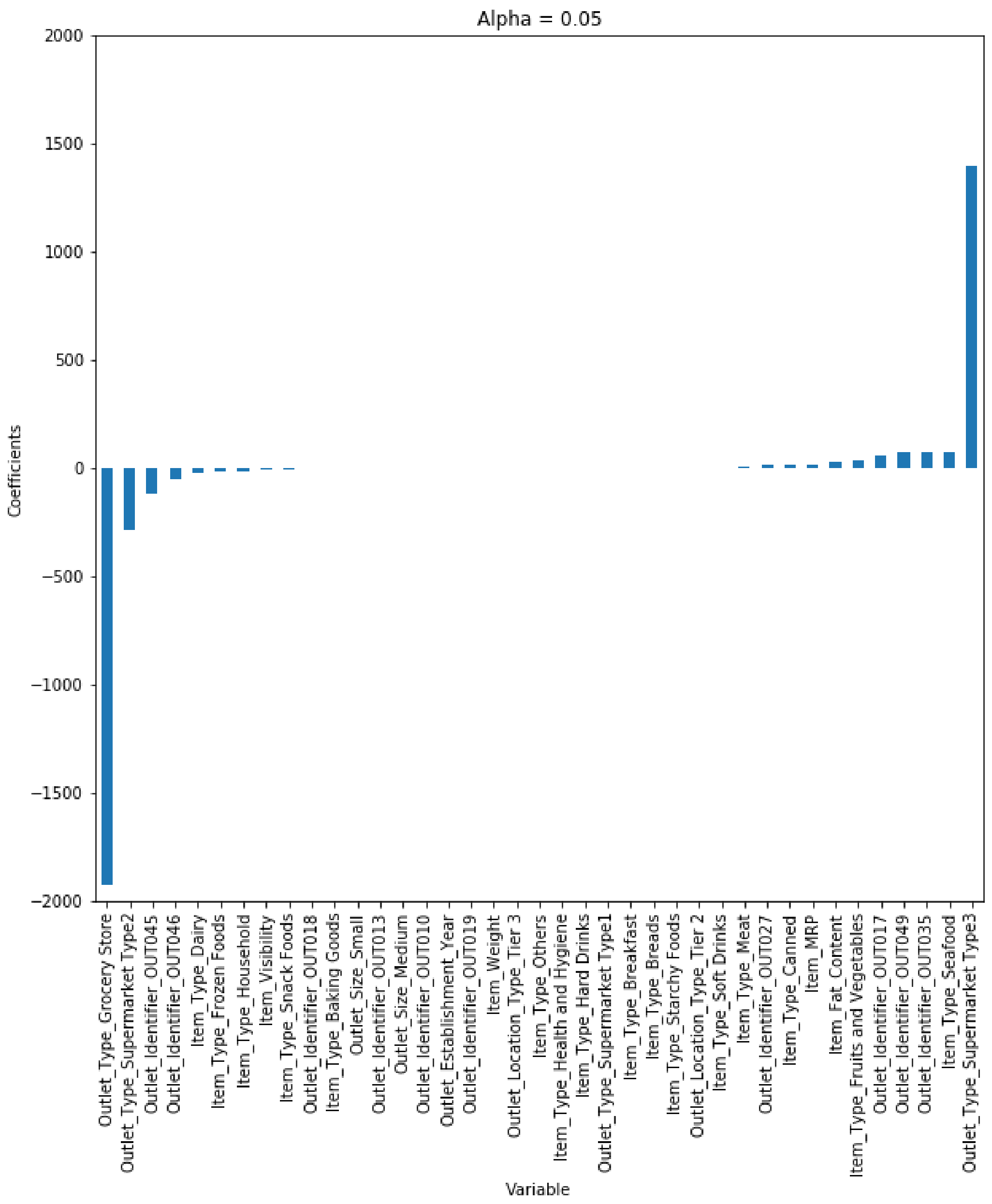
12. Lasso regression

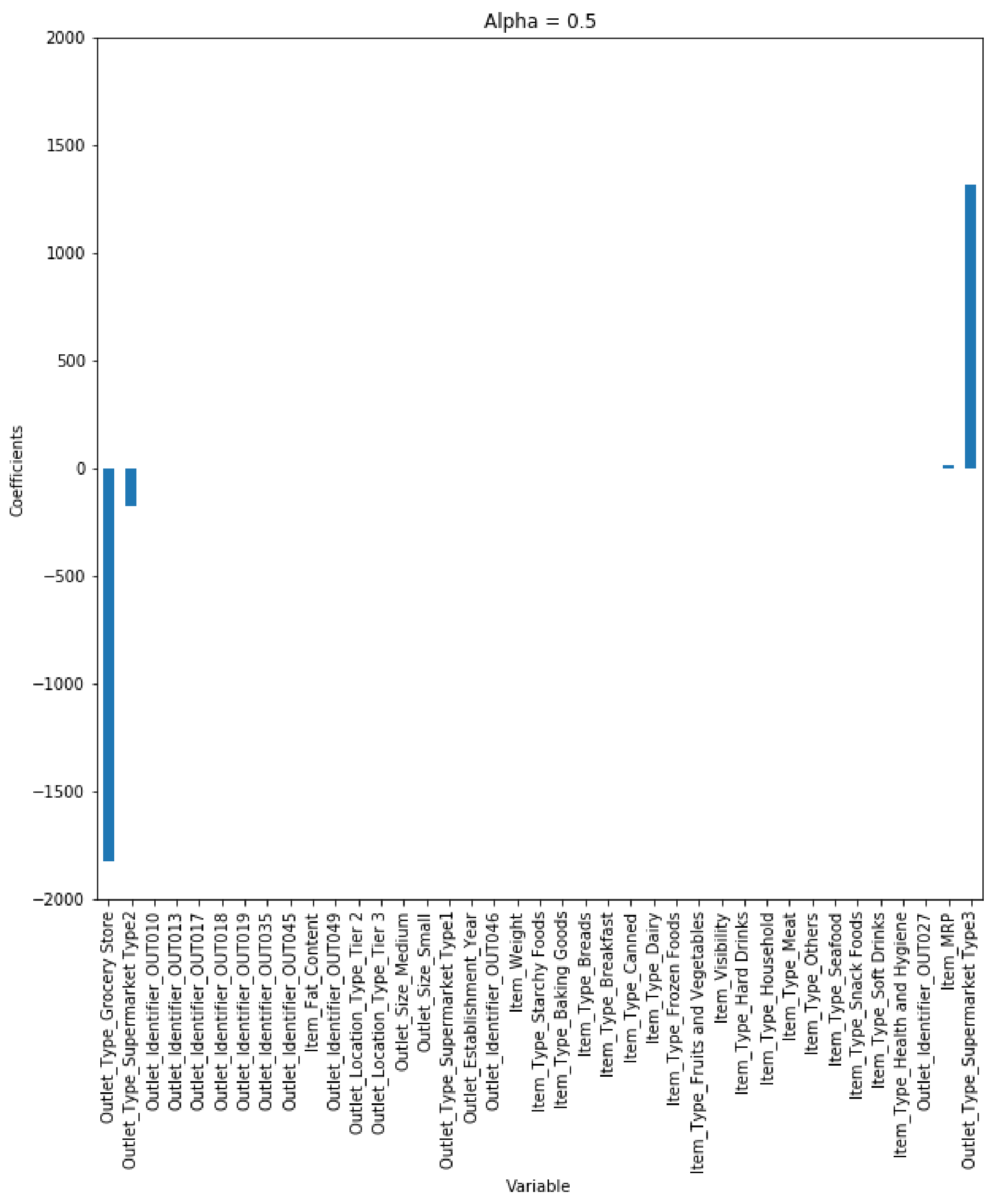
LASSO (Least Absolute Shrinkage Selector Operator), is quite similar to ridge, but lets understand the difference them by implementing it in our big mart problem.

```
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.





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.

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

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:

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

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)

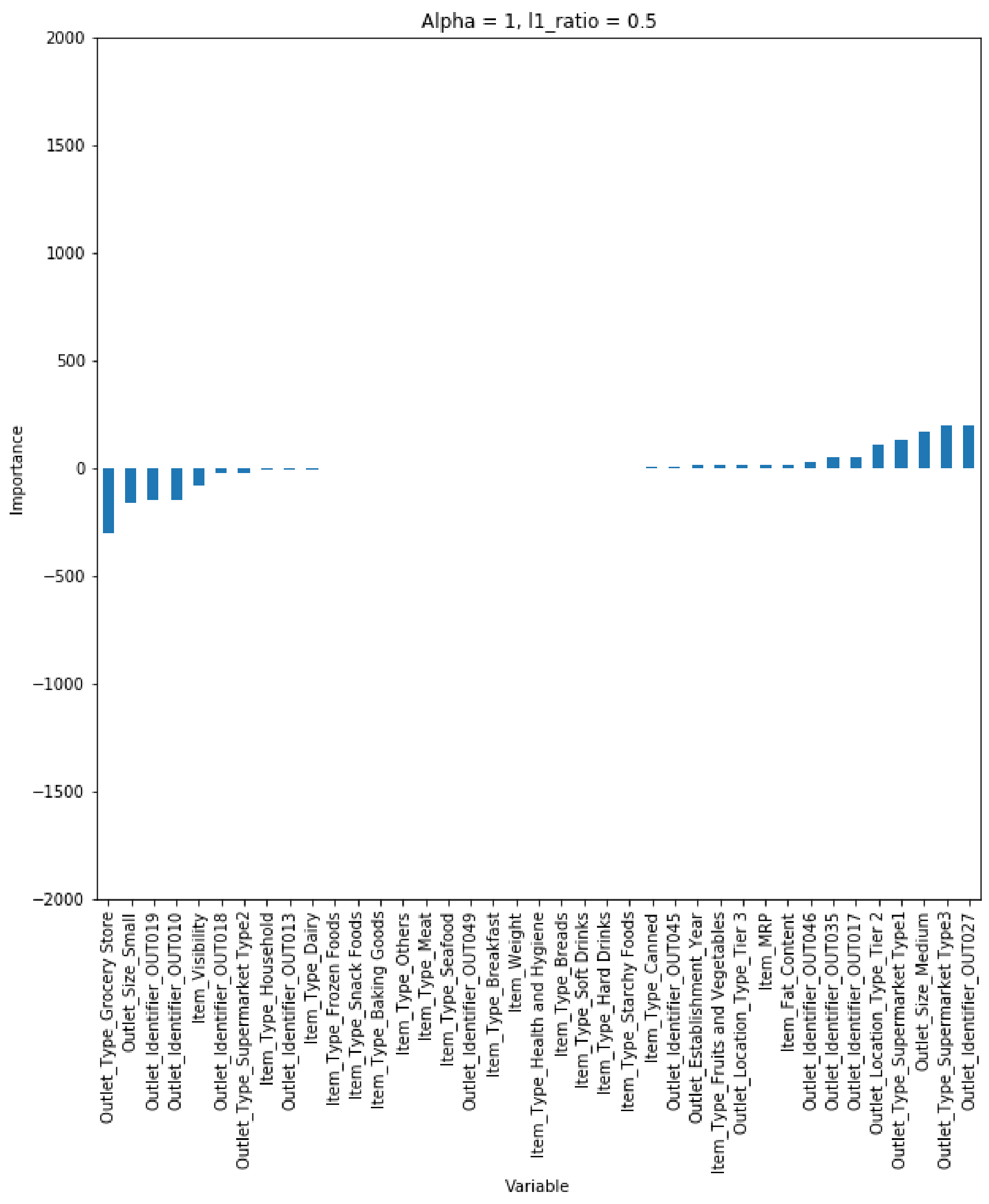
where, a and b weights assigned to L1 and L2 term respectively. So when we change the values of alpha and l1_ratio, a and b are set aaccordingly such that they control trade off between L1 and L2 as:

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

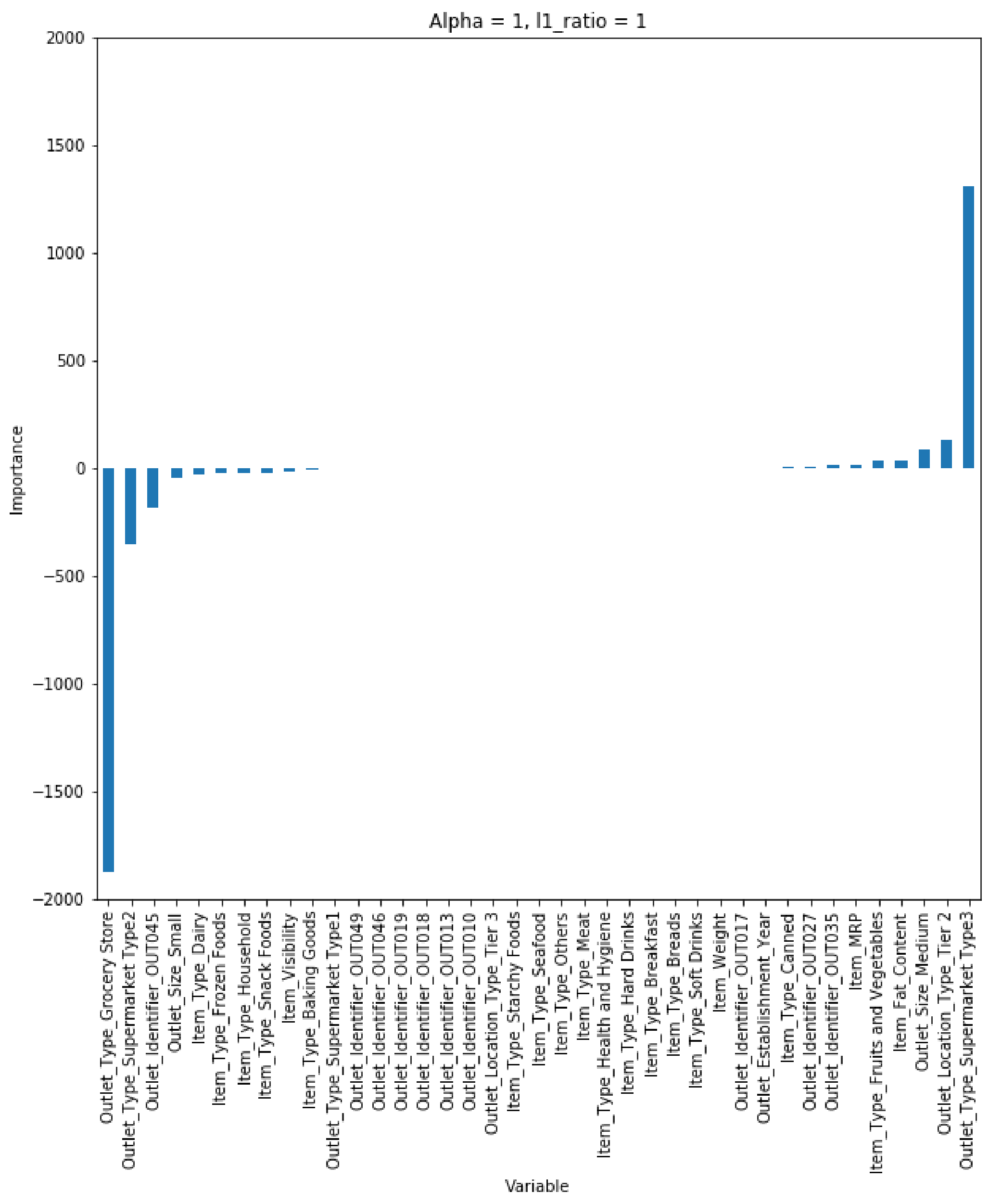
Let alpha (or a+b) = 1, and now consider the following cases:

- 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.

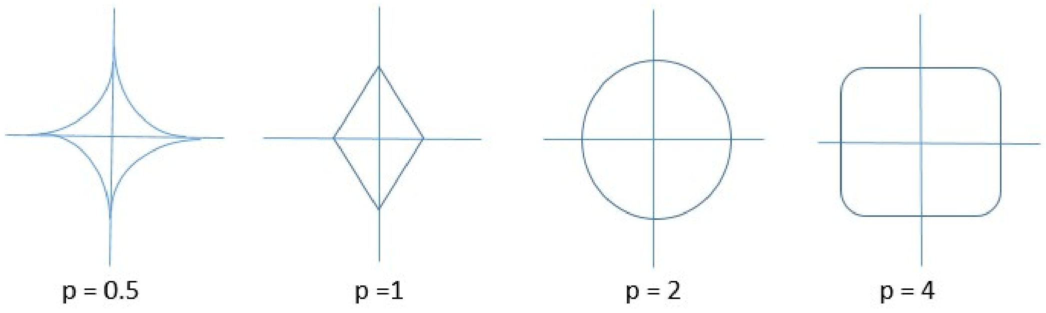
But if you still want to know, below I have explained the concept behind them, which is OPTIONAL.

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 Lp 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.

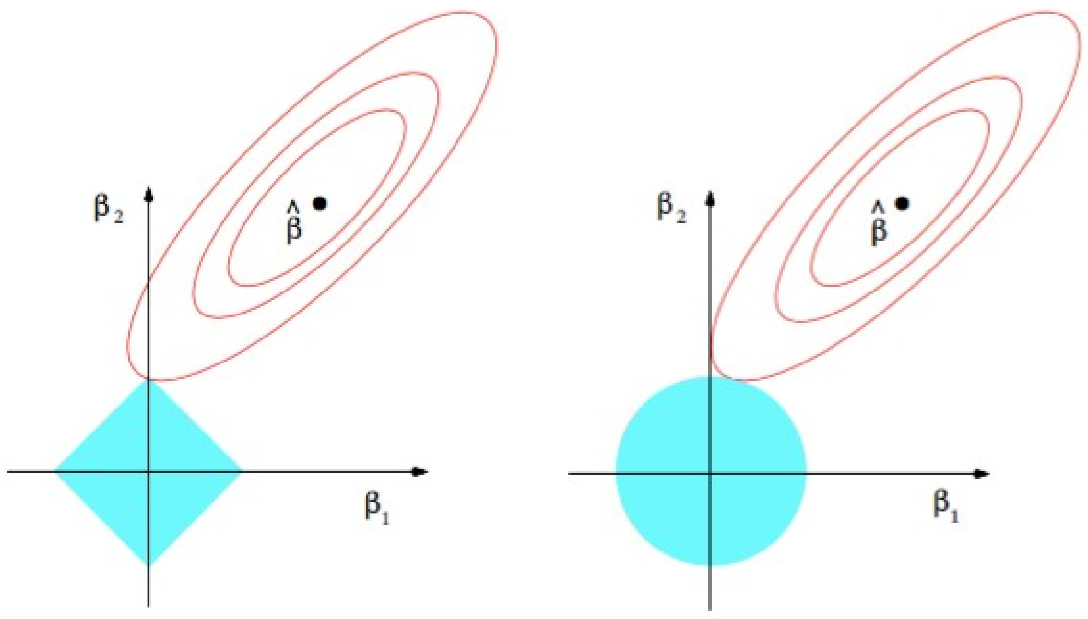


In the above plots, axis denote the parameters(Θ1 and Θ2). Let us examine them one by one.

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.