Regression

Agenda

In this lesson, we will cover the following concepts with the help of a business use case:

- Use Case: Regression
- Regression Algorithms
- Types of Model Evaluation Metrics
- Gradient Descent
- Types of Gradient Descents
- Use Case: Stochastic Gradient Descent (SGD)

Use Case: Regression

Note: At first, with the help of a use case, we are going to perform all the basic steps to reach the model training and predicition part.

Problem Statement:

Google Play Store team is about to launch a new feature wherein, certain apps that are promising are boosted in visibility. The boost will manifest in multiple ways including higher priority in recommendations sections ("Similar apps", "You might also like", "New and updated games"). These will also get a boost in search results visibility. This feature will help bring more attention to newer apps that have the potential.

Analysis to be done:

The problem is to identify the apps that are going to be good for Google to promote. App ratings, which are provided by the customers, are always great indicators of the goodness of the app. The problem reduces to: predict which apps will have high ratings.

Dataset

Google Play Store data (**googleplaystore.csv**)

Link: https://www.dropbox.com/sh/i06ohrau3ucfgbm/AACeYXumL56543KnDNQFlj8ma?dl=0

Data Dictionary:

Variables	Description
App	Application name
Category	Category to which the app belongs
Rating	Overall user rating of the app
Reviews	Number of user reviews for the app
Size	Size of the app
Installs	Number of user downloads/installs for the app

Variables	Description
Type	Paid or Free
Price	Price of the app
Content Rating	Age group the app is targeted at - Children / Mature 21+ / Adult
Genres	An app can belong to multiple genres (apart from its main category)For example, a musical family game will belong to Music, Game, Family genres
Last Updated	Date when the app was last updated on Play Store
Current Ver	Current version of the app available on Play Store
Android Ver	Minimum required Android version

Solution:

Import Libraries

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

Import and Check Dataset

```
inp0 = pd.read_csv("./googleplaystore.csv")

# Check first five rows
inp0.head(2)
```

Observations

The data will be displayed on the screen.

```
#Check number of columns and rows, and data types
inp0.info()
```

Check Data Types

```
#checking datatypes
inp0.dtypes
```

Finding and Treating Null Values

```
#Finding count of null values
inp0.isnull().sum(axis=0)
```

```
#Dropping the records with null ratings
#This is done because ratings is our target variable
inp0.dropna(how ='any', inplace = True)

inp0.isnull().sum(axis=0)
```

Handling the Variables

1. Clean the price column

```
#Cleaning the price column
inp0.Price.value_counts()[:5]
```

Observations

Some have dollars, some have 0

- We need to conditionally handle this.
- First, let's modify the column to take 0 if value is 0, else take the first letter onwards.

```
#Modifying the column
inp0['Price'] = inp0.Price.map(lambda x: 0 if x=='0' else
float(x[1:]))
```

The other columns with numeric data are:

1. Reviews
2. Installs
3. Size

2. Convert reviews to numeric

```
#Converting reviews to numeric
inp0.Reviews = inp0.Reviews.astype("int32")

inp0.Reviews.describe()
```

3. Handle the installs column

```
#Handling the installs column
inp0.Installs.value_counts()
```

We'll need to remove the commas and the plus signs.

Defining function for the same

```
def clean_installs(val):
    return int(val.replace(",","").replace("+",""))

inp0.Installs = inp0.Installs.map(clean_installs)
```

```
inp0.Installs.describe()
```

4. Handle the app size field

```
#Handling the app size field
def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000
        return(x)
    elif 'k' == size[-1:]:
        x = size[:-1]
        x = float(x)
        return(x)
    else:
        return None

inp0["Size"] = inp0["Size"].map(change_size)

inp0.Size.describe()

#Filling Size which had NA
inp0.Size.fillna(method = 'ffill', inplace = True)

#Checking datatypes
inp0.dtypes
```

Sanity checks

1. Average rating should be between 1 and 5, as only these values are allowed on Play Store. Drop any rows that have a value outside this range.

```
#Checking the rating
inp0.Rating.describe()
```

Observations

Min is 1 and max is 5. None of the values have rating outside the range.

1. Reviews should not be more than installs as only those who installed can review the app.

Checking if reviews are more than installs. Counting total rows like this.

```
#Checking and counting the rows
len(inp0[inp0.Reviews > inp0.Installs])

inp0[inp0.Reviews > inp0.Installs]

inp0 = inp0[inp0.Reviews <= inp0.Installs].copy()

inp0.shape
```

1. For free apps **(Type == "Free")**, the price should not be **> 0**. Drop any such rows.

```
len(inp0[(inp0.Type == "Free") & (inp0.Price>0)])
```

EDA

Box Plot: Price

```
#Are there any outliers? Think about the price of usual apps on the
Play Store.
sns.boxplot(inp0.Price)
plt.show()
```

Box Plot: Reviews

```
#Are there any apps with very high number of reviews? Do the values
seem right?
sns.boxplot(inp0.Reviews)
plt.show()
```

Checking Distribution and Skewness:

How are the ratings distributed? Is it more toward higher ratings?

Distribution of Ratings

```
#Distributing the ratings
inp0.Rating.plot.hist()
#Show plot
plt.show()
```

Histogram: Size

```
inp0['Size'].plot.hist()
#Show plot
plt.show()
```

Observations

A histogram is plotted with ratings on the x-axis and frequency on the y-axis, and the ratings are distributed.

```
#Pair plot
sns.pairplot(data=inp0)
```

Outlier Treatment:

1. Price:

From the box plot, it seems like there are some apps with very high prices. A price of $200 for an application on the Play Store is very high and suspicious. Check the records that have very high price: Is 200 a high price?

```
#Checking the records
len(inp0[inp0.Price > 200])

inp0[inp0.Price > 200]

inp0 = inp0[inp0.Price <= 200].copy()

inp0.shape
```

2. Reviews:

Very few apps have very high number of reviews. These are all star apps that don't help with the analysis and, in fact, will skew it. Drop records having more than 2 million reviews.

```
#Dropping the records with more than 2 million reviews
inp0 = inp0[inp0.Reviews <= 2000000]
inp0.shape
```

3. Installs:

There seem to be some outliers in this field too. Apps having a very high number of installs should be dropped from the analysis. Find out the different percentiles – 10, 25, 50, 70, 90, 95, 99.

Decide a threshold as the cutoff for outliers and drop records having values more than the threshold.

```
#Dropping the apps that have a very high number of installs
inp0.Installs.quantile([0.1, 0.25, 0.5, 0.70, 0.9, 0.95, 0.99])
```

Observations

Looks like there are just 1% of apps having more than 100M installs. These apps might be genuine, but will definitely skew our analysis.
We need to drop these.

```
#Dropping the apps with more than 100M installs
len(inp0[inp0.Installs >= 1000000000])

inp0 = inp0[inp0.Installs < 1000000000].copy()
inp0.shape

#Importing warnings
import warnings
warnings.filterwarnings("ignore")
```

Bi-variate Analysis:

Let's look at how the available predictors relate to the variable of interest, i.e., our target variable rating. Make scatter plots (for numeric features) and box plots (for character features) to assess the relationhips between rating and the other features.

1. Make scatter plot/join plot for Rating vs. Price

```
#What pattern do you observe? Does rating increase with price?
sns.jointplot(inp0.Price, inp0.Rating)
```

2. Make scatter plot/joinplot for Rating vs Size

```
#Are heavier apps rated better?
sns.jointplot(inp0.Size, inp0.Rating)
```

3. Make scatter plot/joinplot for Rating vs Reviews

```
# Does more review mean a better rating always?
sns.jointplot(inp0.Reviews, inp0.Rating)
```

4. Make boxplot for Rating vs Content Rating

```
#Is there any difference in the ratings? Are some types liked better?
plt.figure(figsize=[8,6])
sns.boxplot(inp0['Content_Rating'], inp0.Rating)
```

5. Make boxplot for Ratings vs. Category

```
#Which genre has the best ratings?
plt.figure(figsize=[18,6])
g = sns.boxplot(inp0.Category, inp0.Rating)
plt.xticks(rotation=90)
```

Pre-processing the Dataset

1. Make a copy of the dataset

```
#Making a copy
inp1 = inp0.copy()
```

2. Apply log transformation (np.log1p) to Reviews and Installs

Reviews and Installs have some values that are still relatively very high. Before building a linear regression model, you need to reduce the skew.

```
#Reducing the skew
inp0.Installs.describe()

inp1.Installs = inp1.Installs.apply(np.log1p)

inp1.Reviews = inp1.Reviews.apply(np.log1p)
```

3. Drop columns App, Last Updated, Current Ver, and Android Ver

These variables are not useful for our task.

```
inp1.dtypes

#Dropping the variables that are not useful for our task
inp1.drop(["App", "Last Updated", "Current Ver", "Android Ver"],
axis=1, inplace=True)
inp1.shape
```

4. Dummy Columns:

Get dummy columns for Category, Genres, and Content Rating. This needs to be done as the models do not understand categorical data, and all data should be numeric. Dummy encoding is one way to convert character fields to numeric fields. Name of the dataframe should be **inp2**.

```
inp2 = pd.get_dummies(inp1, drop_first=True)

inp2.columns

inp2.shape
```

Train-test split

Let us distribute the data into **training** and **test** datasets using the **train_test_split()** function.

```
from sklearn.model_selection import train_test_split

#?train_test_split

df_train, df_test = train_test_split(inp2, train_size = 0.7,
random_state = 100)

df_train.shape, df_test.shape
```

Let us separate the dataframes into **X_train, y_train, X_test, y_test.**

```
y_train = df_train.pop("Rating")
X_train = df_train

X_train.head(1)

y_test = df_test.pop("Rating")
X_test = df_test

X_test.head(1)
```

Regression Algorithms:

Note: Let us take a look at the theory part before moving on to the training and prediction.

Types of Regression Algorithms:

- Linear regression
- Multiple linear regression
- Polynomial regression
- Ridge regression
- Lasso regression
- ElasticNet regression

When to use regression?

If target variable is a continuous numeric variable (100–2000), then use a regression algorithm.

Example: Predict the price of a house given its sq. area, location, no of bedrooms, etc.

A simple regression algorithm is given below

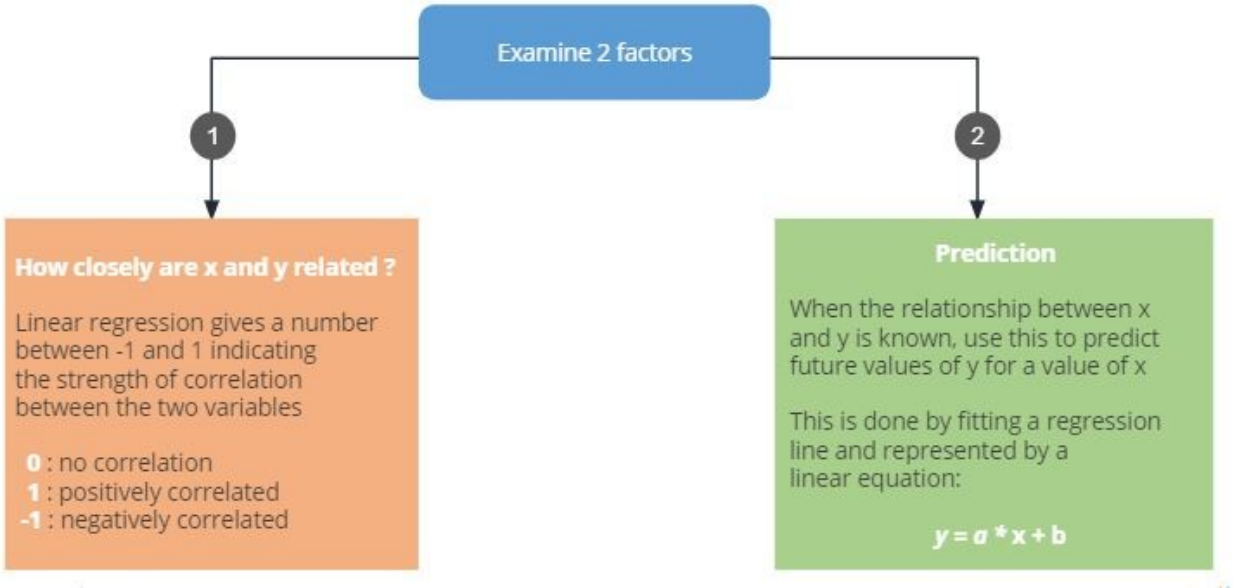
$$y = w * x + b$$

This shows relationship between price (y) and sq. area (x) where price is a number from a defined range.

Note: Let us take a look at the basics of linear regression and then move on to the model builiding part where we are going to use all the concepts that we saw in previous sessions.

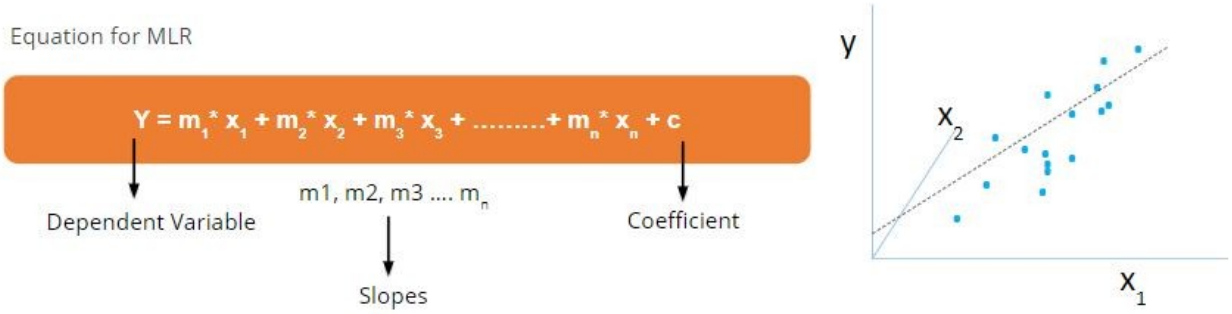
1. Linear Regression:

Linear Regression is a statistical model used to predict the relationship between independent and dependent variables denoted by x and y respectively.



2. Muliple Linear Regression:

Multiple linear regression is a statistical technique used to predict the outcome of a response variable through several explanatory variables and model the relationships between them.

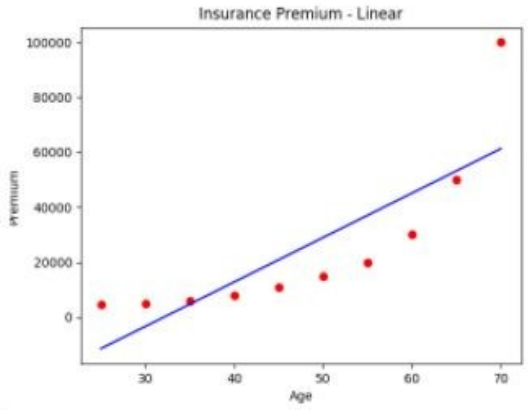


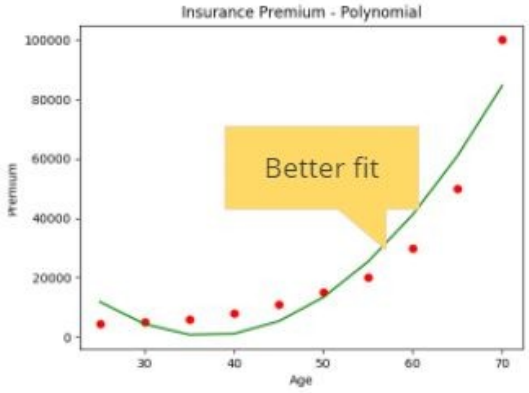
3. Polynomial Regression:

Polynomial regression is applied when data is not formed in a straight line. It is used to fit a linear model to non-linear data by creating new features from powers of non-linear features.

Example: Quadratic features

$$x_2' = x_2^2$$
$$y = w_1 x_1 + w_2 x_2^2 + \text{residual error}$$
$$= w_1 x_1 + w_2 x_2' + \text{residual error}$$





Types of Model Evaluation Metrics:

Assumption

Let us consider the following:

y_i – the observed value

$\acute{y}$ – the mean value of a sample

$\hat{y}_i$ – the value estimated by the regression line

Sum of Squares Total (SST)

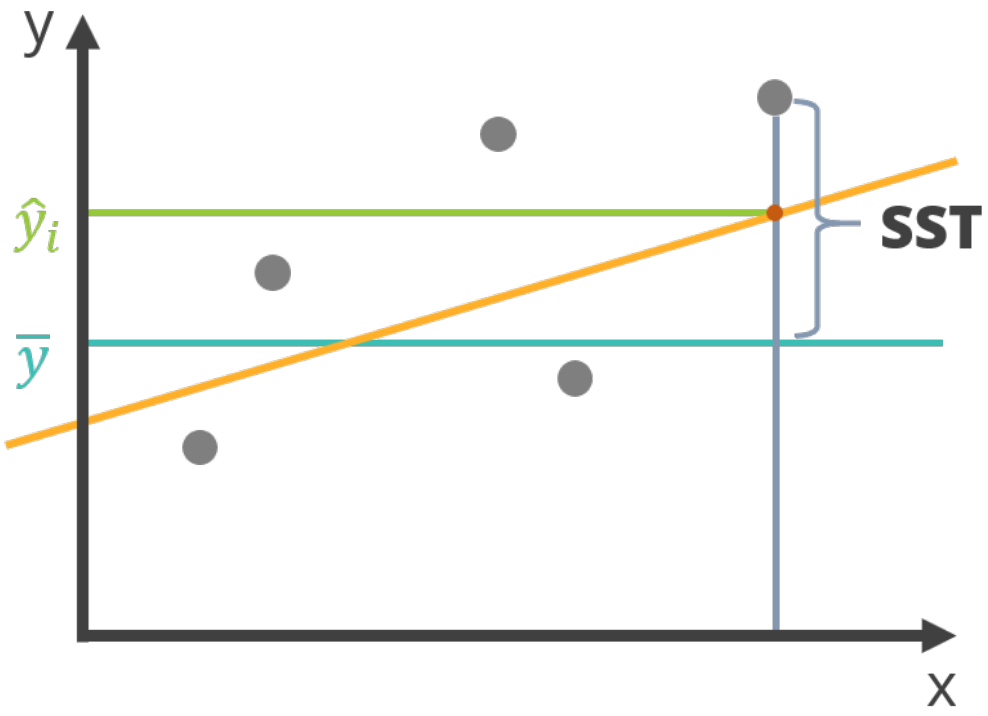
The squared variations between the measured dependent variable and its mean are referred to as the **Sum of Squares Total (SST)** or **Total Sum of Squares (TSS)**.

It's similar to the variation of descriptive statistics in that it's the dispersion of measured variables around the mean.

It is a measure of the dataset's overall variability.

$$SST = SSR + SSE = \sum_{i=1}^{n} (\hat{y}_i - \acute{y}_i)^2 + \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Sum of Squares Total (SST)



$$\sum_{i=1}^{n} (y_i - \bar{y})^2$$

Sum of Squares due to Regression (SSR)

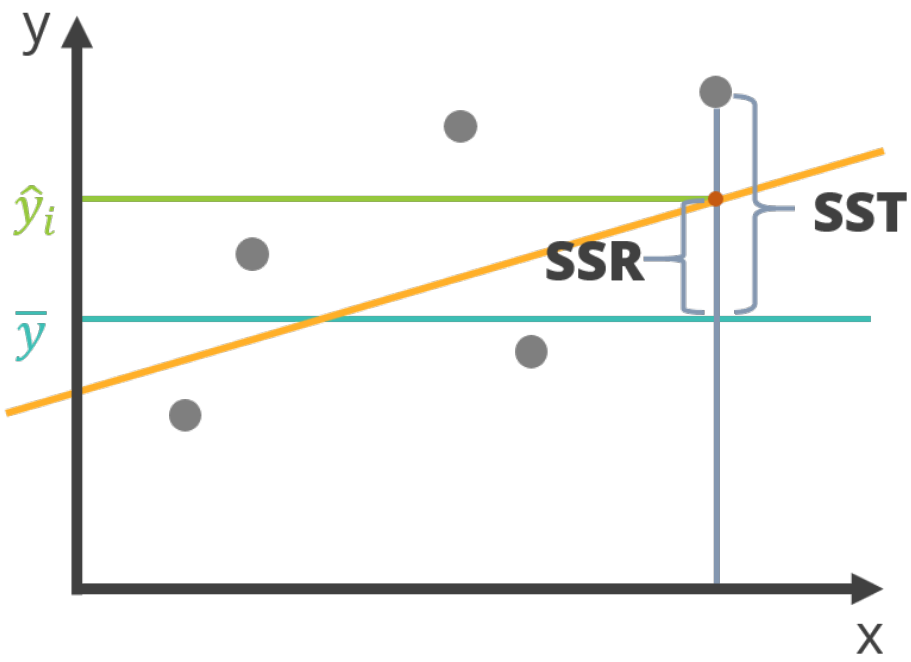
The difference between the predicted value and the dependent variable's mean are referred to as the **Sum of Squares due to Regression (SSR)** or **Explained Sum of Squares (ESS)**.

It can be considered as a metric for describing how well our line fits the data.

If the SSR (or ESS) is equal to the SST (or TSS), the regression model is flawless and captures all observed variability.

$$SSR = \sum_{i=1}^{n} \left(y_i - \acute{y}\right)^2$$

Sum of Squares Regression (SSR)



Measures the explained variability by your line

$$\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$$

Sum of Squares Error (SSE)

The difference between the observed and predicted values are referred to as the **Sum of Squares Error (SSE)** or **Residual Sum of Squares (RSS)**, where `residual` stands for `remaining` or `unexplained`.

This error must be reduced since the smaller it is, the better the regression's estimation power.

$$SSE = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$$

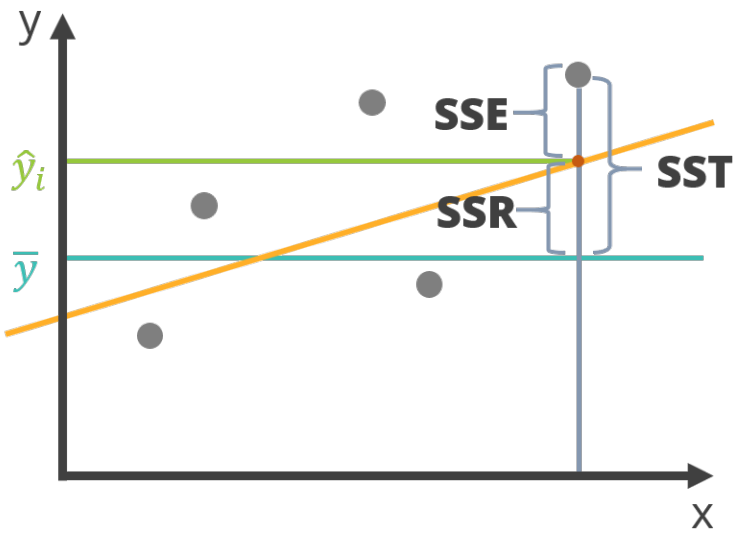
$$Or$$

$$SSE = \sum_{i=1}^{n} e_i^2$$

$$where,$$

$$e_i = \left(y_i - \hat{y}_i \right)$$

Sum of Squares Error (SSE)



Measures the unexplained variability by the regression

$$\sum_{i=1}^{n} e_i^2$$

Relation Among SST, SSR, and SSE

Since certain people use these abbreviations in various ways, it can be very confusing.

We use one of two sets of notations for these abbreviations: SST, SSR, and SSE or TSS, ESS, and RSS.

These equations are related in the following ways:

$$SST = SSR + SSE$$

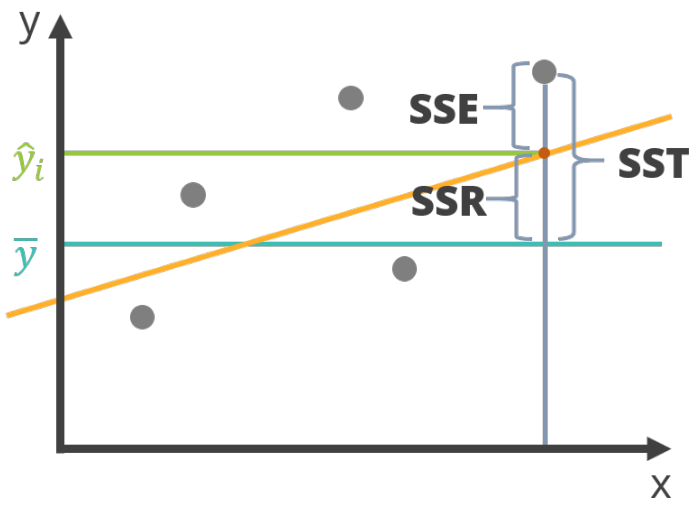
$$Or$$

$$TSS = ESS + RSS$$

This is because the overall variability of the dataset is equivalent to the variability described by the regression line and the unknown variability (also known as error).

For a constant total variability, a lower error would result in a better regression. A higher error, on the other hand, would result in a weaker regression. This should always be remembered regardless of the notation set used.

Connection?



Total variability= Explained variability + Unexplained variability

$$\sum_{i=1}^{n}(y_i - \bar{y})^2 = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2 + \sum_{i=1}^{n} e_i^2$$

R-Square Matrix

The determination coefficient also known as **R2 (R-squared) score** is used for the performance evaluation of a linear regression model.

R2 displays the proportion of data points inside the regression equation line.

A higher R2 value means improved results.

It is calculated as follows:

$$R^2 = 1 - \frac{SSE}{SSR}$$

$$Or$$

$$R^2 = 1 - \frac{RSS}{ESS}$$

The highest possible score is 1, which is achieved when the predicted and actual values are the same.

The R2 score is 0 for a baseline model.

In the worst-case scenario, the R2 score can also be negative.

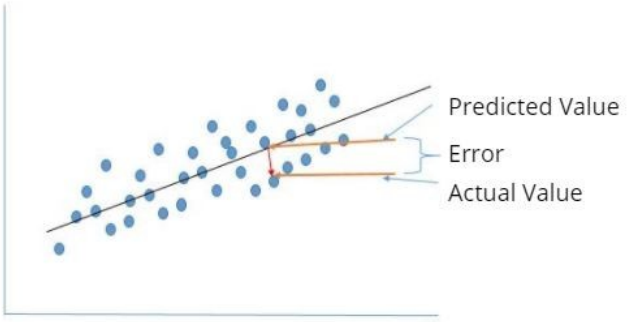
Import *statsmodels* Library for Linear Regression

```
#Importing the statsmodel library
import statsmodels.api as sm

#Applying linear regression
model1 = sm.OLS(y_train, X_train)

model1 = model1.fit()

#Finding the summary
model1.summary()
```

Identified values for *model1*:

- R^2 **(uncentered): 0.987**

Overview of R^2 (R-square):



$$\text{R-square} = 1 - \frac{\sum(Y\_actual - Y\_predicted)^2}{\sum(Y\_actual - Y\_mean)^2}$$

R-square is the most common metric to judge the performance of regression models

R^2 lies between 0 -100 %

- **Adjusted R^2 (uncentered): 0.987**

  Overview of adjusted R^2:

  The disadvantage with R-squared is that it assumes every independent variable in the model and explains variations in the dependent variable. Use adjusted R-squared when working on a multiple linear regression problem.

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

where R^2 is R-squared value
P is number of predictor variables
N is number data points

- **F-statistic: 3717**

  Overview of F-statistic:

  The F-test indicates whether the linear regression model provides a better fit to the data in comparison to a model that contains no independent variables.

  Note: In general, the models with high F-statistic are considered as optimum.

  An F-test provides 2 values:

  - F-critical: This value is also called as F statistic
  - F-value: This value is without the “critical” part

Note: The F-value is always observed along with the p-value. In general, the higher the F-value of a variable, the lesser the p-value.

- **AIC: 8811**
- **BIC: 9646**

  Overview of AIC and BIC:

  - AIC and BIC are penalized-likelihood criteria.
  - They are used for selecting the best predictor subsets in regression and comparing the nonnested models.
  - AIC: "It is an estimate of a constant plus the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model."
  - BIC: "It is an estimate of a function of the posterior probability of a model being true, under a certain Bayesian setup."

    Source: https://www.methodology.psu.edu/resources/AIC-vs-BIC/

- **Formula:**

  The AIC or BIC for a model is usually written in the form:

  $[-2 log L + k p)$

  Where:

  - L = likelihood function
  - p = number of parameters in the model
  - k = 2 for AIC and log(n) for BIC
- In popular opinion:
  - Lower AIC tells that the model is closer to the truth
  - Lower BIC tells that the model is more likely to be the true model

Prediction and evaluation using "model1"

```
#Prediction and evaluation
import sklearn.metrics as metrics
y_test_pred= model1.predict(X_test)
```

Metrics used for regression models:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Lower values of RMSE indicate better fit.

```
#Assessing the model using RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, y_test_pred)))

#RMSE of "model1" = 0.49203326281981113
```

MPE:

Mean Percentage Error measures the errors that indicate whether or not the prediction is biased.

Bias is one component of the mean squared error and is measured as the variance of the errors plus the square of the mean error.

$$MSE = VAR(E) + (MPE)^2$$

Therefore, minimizing the MSE will implicitly minimize the bias and the variance of the errors.

MPE is the computed average of percentage errors by which prediction of a model differ from actual values of the quantity being predicted.

Formula:

$$MPE = \frac{100\,\%}{n}\sum_{i=1}^{n}\square\frac{a_i - p_i}{a_i}$$

Where:

- a_i = Actual value of the quantity being predicted
- p_i = Predicted value
- n = Number of different times for which the variable is predicted

```
#Calculating MPE
def MPE(y_test_pred,y_test):
```

```
    return ( ((y_test - y_test_pred) / y_test).mean()) * 100

print ('MPE: ' + str(MPE(y_test_pred,y_test)) )
```

MAPE:

Mean Absolute Percentage Error measures the prediction accuracy of a model.

Formula:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \frac{|A_i - P_i|}{|A_i|}$$

Where:

- A_i = Actual value of the quantity being predicted
- P_i = Predicted value
- n = Number of different times for which the variable is predicted

Mean Absolute Percentage Error is similar to Mean Absolute Error but is normalized by true observation.

The mean absolute percentage error is useful for reporting purpose and is expressed in generic percentage terms.

Note: In general, the desirable MAPE value is less than 20.

```
#Calculating MAPE
def MAPE(y_test_pred,y_test):
    return np.mean(np.abs((y_test - y_test_pred) / y_test)) * 100

print ('MAPE: ' + str(MAPE(y_test_pred,y_test)) )
```

Reiterating the Model

1. Feature selection using p-value:

In regression analysis, p-values and coefficients together indicate which relationships in the model are statistically significant and the nature of those relationships.

Coefficients describe the mathematical relationship between each independent variable and the dependent variable.

p-values for the coefficients indicate whether these relationships are statistically significant.

In general, the threshold for the p-value is taken as ≤ 0.05.

Let's find the variable with value ≤ 0.05.

2. List:

Identified 99 variables that have p-value ≤ 0.05.

'Rating','Reviews','Installs','Category_AUTO_AND_VEHICLES','Category_BEAUTY','Category_BOOKS_AND_REFERENCE','Category_BUSINESS','Category_COMICS','Category_COMMUNICATION','Category_DATING','Category_EDUCATION','Category_EDUCATION','Category_ENTERTAINMENT','Category_EVENTS','Category_FAMILY','Category_FINANCE','Category_FOOD_AND_DRINK','Category_GAME','Category_HEALTH_AND_FITNESS','Category_HOUSE_AND_HOME','Category_LIBRARIES_AND_DEMO','Category_LIFESTYLE','Category_MAPS_AND_NAVIGATION','Category_MEDICAL','Category_NEWS_AND_MAGAZINES','Category_PARENTING','Category_PERSONALIZATION','Category_PHOTOGRAPHY','Category_PRODUCTIVITY','Category_SHOPPING','Category_SOCIAL','Category_SPORTS','Category_TOOLS','Category_TRAVEL_AND_LOCAL','Category_VIDEO_PLAYERS','Category_WEATHER','Type_Paid','Content_Rating_Everyone','Content_Rating_Everyone_10+','Content_Rating_Mature_17+','Content_Rating_Teen','Content_Rating_Unrated','Genres_Art_&*Design*','*Genres_Art&Design;Creativity*','*Genres_Art&Design;Pretend_Play*','*Genres_Auto&Vehicles*','*Genres_Beauty*','*Genres_Board;Brain_Games*','*Genres_Books&Reference*','*Genres_Business*','*Genres_Casual*','*Genres_Casual;Action&Adventure*','*Genres_Casual;Brain_Games*','*Genres_Casual;Pretend_Play*','*Genres_Comics*','*Genres_Comics;Creativity*','*Genres_Communication*','*Genres_Dating*','*Genres_Education*','*Genres_Education;Action&Adventure*','*Genres_Education;Creativity*','*Genres_Education;Education*','*Genres_Education;Pretend_Play*','*Genres_Educational;Education*','*Genres_Educational;Pretend_Play*','*Genres_Entertainment*','*Genres_Entertainment;Brain_Games*','*Genres_Entertainment;Creativity*','*Genres_Entertainment;Music&Video*','*Genres_Events*','*Genres_Finance*','*Genres_Food&Drink*','*Genres_Health&Fitness*','*Genres_House&Home*','*Genres_Libraries&Demo*','*Genres_Lifestyle*','*Genres_Maps&Navigation*','*Genres_Medical*','*Genres_Music;Music&Video*','*Genres_News&Magazines*','*Genres_Parenting*','*Genres_Parenting;Education*','*Genres_Parenting;Music&Video*','*Genres_Personalization*','*Genres_Photography*','*Genres_Productivity*','*Genres_Puzzle*','*Genres_Puzzle;Brain_Games*','*Genres_Racing;Action&Adventure*','*Genres_Role_Playing*','*Genres_Shopping*','*Genres_Simulation*','*Genres_Simulation;Action&Adventure*','*Genres_Social*','*Genres_Strategy*','*Genres_Tools*','*Genres_Tools;Education*','*Genres_Travel&Local*','*Genres_Travel&Local;Action*&_Adventure','Genres_Weather'

3. Create a new dataset with chosen variables and split it into training and testing dataset

```
#Create inp3
inp3 =
inp2[['Rating','Reviews','Installs','Category_AUTO_AND_VEHICLES','Category_BEAUTY',

'Category_BOOKS_AND_REFERENCE','Category_BUSINESS','Category_COMICS',

'Category_COMMUNICATION','Category_DATING','Category_EDUCATION','Category_EDUCATION',

'Category_ENTERTAINMENT','Category_EVENTS','Category_FAMILY','Category_FINANCE',

'Category_FOOD_AND_DRINK','Category_GAME','Category_HEALTH_AND_FITNESS',

'Category_HOUSE_AND_HOME','Category_LIBRARIES_AND_DEMO','Category_LIFESTYLE',
```

```
'Category_MAPS_AND_NAVIGATION','Category_MEDICAL','Category_NEWS_AND_MAGAZINES',

'Category_PARENTING','Category_PERSONALIZATION','Category_PHOTOGRAPHY','Category_PRODUCTIVITY',

'Category_SHOPPING','Category_SOCIAL','Category_SPORTS','Category_TOOLS','Category_TRAVEL_AND_LOCAL',

'Category_VIDEO_PLAYERS','Category_WEATHER','Type_Paid','Content_Rating_Everyone',

'Content_Rating_Everyone_10+','Content_Rating_Mature_17+','Content_Rating_Teen','Content_Rating_Unrated',

'Genres_Art_&_Design','Genres_Art_&_Design_Creativity','Genres_Art_&_Design_Pretend_Play',

'Genres_Auto_&_Vehicles','Genres_Beauty','Genres_Board_Brain_Games','Genres_Books_&_Reference',

'Genres_Business','Genres_Casual','Genres_Casual_Action_&_Adventure','Genres_Casual_Brain_Games',

'Genres_Casual_Pretend_Play','Genres_Comics','Genres_Comics_Creativity','Genres_Communication',

'Genres_Dating','Genres_Education','Genres_Education_Action_&_Adventure','Genres_Education_Creativity',

'Genres_Education_Education','Genres_Education_Pretend_Play','Genres_Educational_Education',

'Genres_Educational_Pretend_Play','Genres_Entertainment','Genres_Entertainment_Brain_Games',

'Genres_Entertainment_Creativity','Genres_Entertainment_Music_&_Video','Genres_Events',

'Genres_Finance','Genres_Food_&_Drink','Genres_Health_&_Fitness','Genres_House_&_Home',

'Genres_Libraries_&_Demo','Genres_Lifestyle','Genres_Maps_&_Navigation','Genres_Medical',

'Genres_Music_Music_&_Video','Genres_News_&_Magazines','Genres_Parenting','Genres_Parenting_Education',
```

```
'Genres_Parenting_Music_&_Video','Genres_Personalization','Genres_Photography','Genres_Productivity',

'Genres_Puzzle','Genres_Puzzle_Brain_Games','Genres_Racing_Action_&_Adventure','Genres_Role_Playing',

'Genres_Shopping','Genres_Simulation','Genres_Simulation_Action_&_Adventure','Genres_Social',

'Genres_Strategy','Genres_Tools','Genres_Tools_Education','Genres_Travel_&_Local',

'Genres_Travel_&_Local_Action_&_Adventure','Genres_Weather']]

#Checking datatypes
inp3.dtypes

inp3.shape

#split imp3
df_train, df_test = train_test_split(inp3, train_size = 0.7,
random_state = 100)

df_train.shape, df_test.shape
```

4. Separate the new dataframes into X_train, y_train, X_test, y_test

```
#Separating the new dataframes
y_train1 = df_train.pop("Rating")
X_train1 = df_train

X_train1.head(1)

y_test1 = df_test.pop("Rating")
X_test1 = df_test

X_test1.head(1)
```

5. Create the second model

```
#Apply linear regression
model2 = sm.OLS(y_train1, X_train1)

model2 = model2.fit()

#Find the summary
model2.summary()
```

- Rating R-squared (uncentered): 0.987
- Adj. R-squared (uncentered): 0.987

- F-statistic: 6412
- AIC: 8749
- BIC: 9234

How Well Does the Model Fit the Data?

***model2* performs better:**

- R-squared value:

  The most common way to evaluate the overall fit of a linear model is by the **R-squared** value.

  R-squared is between 0 and 1 (or between 0 to 100%), and higher is better because it means that more variance is explained by the model.

  In case of our models, model1 and model2 have same R-squared values.

- F-statistics:

  In case of dataset that only have numerical values, it is said higher the F-statistic better the model.

  In case of our models, model2 has more F-statistic value than model1.

- AIC and BIC:

  In case of models, lesser the AIC and BIC, better the model.

  In case of models, model2 has lesser AIC and BIC values than the model1.

Prediction using "model2"

Let us perform prediction using model2 by importing **sklearn.metrics**

```
#Importing sklearn.metrics
import sklearn.metrics as metrics
y_test_pred1= model2.predict(X_test1)
print(np.sqrt(metrics.mean_squared_error(y_test, y_test_pred1)))


#RMSE value of model1 = 0.49203326281981113

#Calculating MAPE
def MAPE(y_test_pred1,y_test1):
    return np.mean(np.abs((y_test1 - y_test_pred1) / y_test1)) * 100
print ('MAPE: ' + str(MAPE(y_test_pred1,y_test1)) )
```

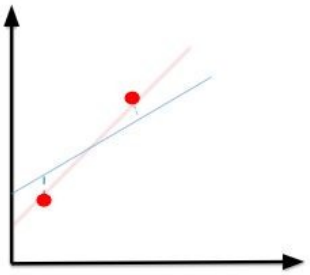
model2 shows slight improvement as the RMSE and MAPE value of this model is lesser than that of **model1**.

4. Ridge Regression:

- Ridge Regression (L2) is used when there is a problem of multicollinearity.
- By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

The main idea is to find a new line that has some bias with respect to the training data

In return for that small amount of bias, a significant drop in variance is achieved



Minimization objective = LS Obj + λ * (sum of the square of coefficients)

LS Obj refers to least squares objective

λ controls the strength of the penalty term

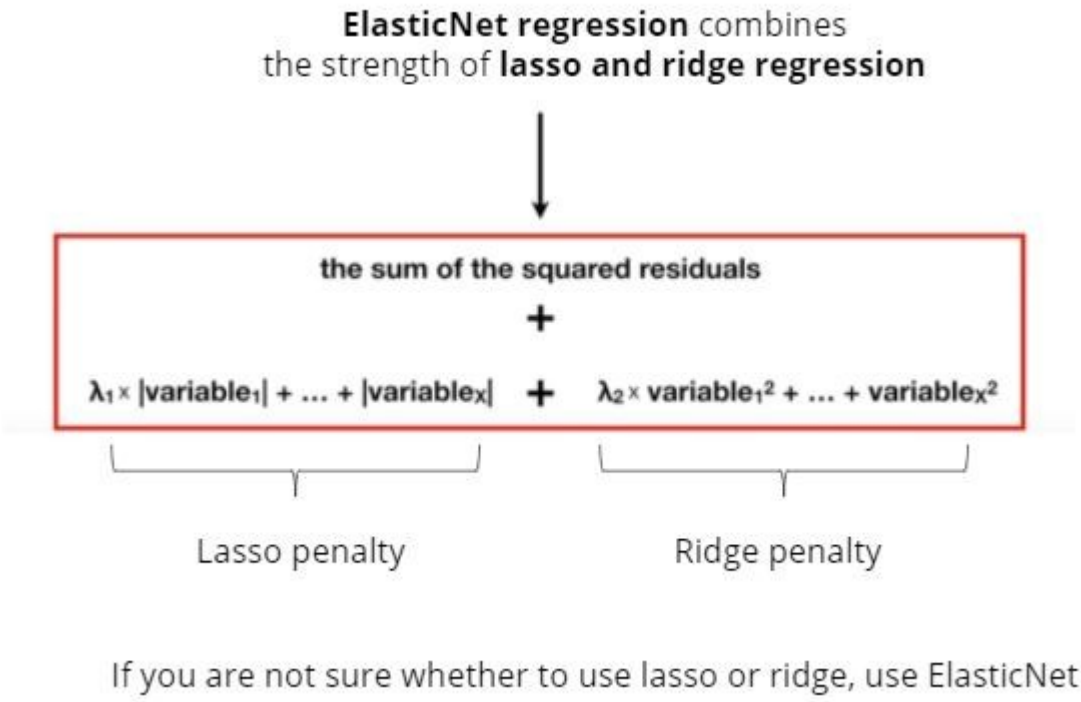
5. Lasso Regression:

- Lasso Regression (L1) is similar to ridge, but it also performs feature selection.
- It will set the coefficient value for features that do not help in decision making very low, potentially zero.

Minimization objective = LS Obj + λ * (sum of absolute coefficient values)

- Lasso regression tends to exclude variables that are not required from the equation, whereas ridge tends to do better when all variables are present

6. ElasticNet Regression:



If you are not sure whether to use lasso or ridge, use ElasticNet

Use Case: Ridge, Lasso, ElasticNet Regression for Training and Prediction:

We are going to use the same dataset that we used in the previous use case at the time of training of "model2" i.e. inp3

1. Ridge Regression:

```
#Importing Ridge
import sklearn
from sklearn.linear_model import Ridge
ridgeReg = Ridge(alpha=0.001, normalize=True)
ridgeReg.fit(X_train1,y_train1)
```

Evaluating using RMSE:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1,
ridgeReg.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1,
ridgeReg.predict(X_test1))))
print('R2 Value/Coefficient of Determination:
{}'.format(ridgeReg.score(X_test1, y_test1)))
```

2. Lasso Regression:

```
#Importing Lasso
from sklearn.linear_model import Lasso
lassoreg = Lasso(alpha=0.001, normalize=True)
lassoreg.fit(X_train1,y_train1)
```

Evaluating using RMSE:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1,
lassoreg.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1,
lassoreg.predict(X_test1))))
print('R2 Value/Coefficient of Determination:
{}'.format(lassoreg.score(X_test1, y_test1)))
```

3. ElasticNet Regression:

```
#Importing ElasticNet
from sklearn.linear_model import ElasticNet
Elastic = ElasticNet(alpha=0.001, normalize=True)
Elastic.fit(X_train1,y_train1)
```

Evaluating using RMSE:

```
print(np.sqrt(sklearn.metrics.mean_squared_error(y_train1,
Elastic.predict(X_train1))))
print(np.sqrt(sklearn.metrics.mean_squared_error(y_test1,
Elastic.predict(X_test1))))
print('R2 Value/Coefficient of Determination:
{}'.format(Elastic.score(X_test1, y_test1)))
```

Exercise:

- Perform the iteration of the model with Lasso, Ridge, and ElasticNet Regression by using the original dataset i.e., **inp0** as done in the case of Linear Regression.
- Use the following metrics to evaluate the model:
  - RMSE
  - MAPE
  - R^2 error

Cost Function:

- A cost function is a function that evaluates a model's performance for a given dataset.
- It evaluates and expresses the error between predicted values and expected values as a single, real number.

Gradient:

- A gradient is a measurement of how much a function's output varies as its inputs are changed.

Gradient Descent:

- Gradient descent is an optimization algorithm that is used to find the values of the parameters (coefficients) of a function that minimizes the cost function by iteratively moving in the direction of steepest descent as determined by the gradient's negative.
- It's an optimization algorithm to discover the local minimum of a differentiable function or feature.
- We use gradient descent to update the parameters of our model. In linear regression, parameters correspond to coefficients, and in neural networks, parameters correspond to weights.
- The gradient descent equation is as follows:

Cost Function

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x_i) - y_i]^2$$

Predicted Value

True Value

Gradient Descent

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Learning Rate

Now,

$$\frac{\partial}{\partial \theta} J_\theta = \frac{\partial}{\partial \theta} \frac{1}{2m} \sum_{i=1}^{m} [h_\theta(x_i) - y_i]^2$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i) . \frac{\partial}{\partial \theta_j} (\theta x_i - y_i)$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m}$$

Types of Gradient Descents:

Gradient descents are divided into three categories, which vary primarily in the amount of data they use. These categories are:

1. Batch Gradient Descent (BGD)
2. Stochastic Gradient Descent (SGD)
3. Mini-Batch Gradient Descent (Mini BGD)

1. Batch Gradient Descent (BGD):

- Batch gradient descent (BGD), also known as vanilla gradient descent, measures the error for each example in the training dataset, but the model is updated only after all of the training examples have been evaluated.
- This whole process is referred to as a training epoch because it resembles a loop.
- For each gradient descent iteration, it processes all the training samples. However, batch gradient descent is computationally very expensive when the number of training examples is high.
- Thus, if the number of training examples is high, we tend to use stochastic gradient descent (SGD) or mini-batch gradient descent (Mini BGD) instead.

2. Stochastic Gradient Descent (SGD):

- Stochastic gradient descent (SGD) is a form of gradient descent that processes one training example per iteration.
- It estimates the error for each example in the training dataset and updates the parameters one by one.
- Based on the problem statement, batch gradient descent (BGD) can be much faster because the parameters are modified even after an iteration in which only a single example has been processed.
- One advantage is that the frequent updates allow us to track our progress in great detail.
- However, even if the number of training examples is high, it can only process one of them, which will add to the system's overhead and the number of iterations needed.

3. Mini-Batch Gradient Descent (Mini BGD):

- Mini-batch gradient descent (Mini BGD) combines the principles of stochastic gradient descent (SGD) and batch gradient descent (BGD) and is faster than both.
- It divides the training dataset into small batches and updates each of those batches. This establishes a balance between the robustness of stochastic gradient descent (SGD) and the efficiency of batch gradient descent (BGD).
- It is compatible with both larger and smaller training examples.

Use Case: Stochastic Gradient Descent (SGD):

Importing Required Libraries

```
#Importing libraries
import numpy as np
import pandas as pd

from tabulate import tabulate

from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LinearRegression, SGDRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score

import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
%matplotlib inline
```

Load the Data

```
#Load the data dictionary
boston = load_boston()

#Find the dict keys
print(boston.keys())
```

Dataset Description

```
#To print the boston dataset description
print(boston.DESCR)
```

Find Features and Target

Let us split the data into **test** and **train.**

```
X = boston.data
Y = boston.target

#Splitting the data to test and train
x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.3)
```

Find Feature's Name

```
columns = boston.feature_names
columns
```

```
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
boston_df["MEDV"] = Y

boston_df.head()

print(boston_df.describe())
```

Standardizing Data

Let us standardize the **test** and **train** data using the **StandardScaler()** function.

```
scaler = StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)

train_data=pd.DataFrame(x_train)
train_data['price']=y_train
train_data.head(3)

x_test = np.array(x_test)
y_test = np.array(y_test)

#Shape of test and train data metrics
print(x_train.shape)
print(y_train.shape)

print(x_test.shape)
print(y_test.shape)
```

Linear Regression: Boston Housing Prediction

Let us calculate **R-Squared**, **Linear Regressor Model Accuracy**, **MAE**, **MSE**, and **RMSE**.

```
print("Linear Regression: Boston Housing Prediction")
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
lin_score = lin_reg.score(x_train, y_train)
print("R-squared:", lin_score)

lin_y_pred = lin_reg.predict(x_test)
lin_accuracy = explained_variance_score(y_test, lin_y_pred)
lin_accuracy = round(lin_accuracy*100, 6)
print("Linear Regressor Model Accuracy:", lin_accuracy, "%")
print()

lin_mae = mean_absolute_error(y_test, lin_y_pred)
lin_mse = mean_squared_error(y_test, lin_y_pred)
lin_rmse = lin_mse*(1/2.0)
```

```
print("MAE:", lin_mae)
print("MSE:", lin_mae)
print("RMSE:", lin_rmse)
print()
```

Let us plot a graph of the **Actual vs. Predicted Target**.

```
plt.scatter(y_test, lin_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```

Let us plot a graph of the **Test vs. Predicted Data**.

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, lin_y_pred, label="predicted")
plt.title("Test vs. Predicted Data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

SGD Regression: Boston Housing Prediction

Let us calculate **R-squared**, **SGD Regressor Model Accuracy**, **MAE**, **MSE**, and **RMSE**.

```
print("SGD Regression: Boston Housing Prediction")
sgd_reg = SGDRegressor()
sgd_reg.fit(x_train, y_train)
sgd_score = sgd_reg.score(x_train, y_train)
print("R-squared:", sgd_score)

sgd_y_pred = sgd_reg.predict(x_test)
sgd_accuracy = explained_variance_score(y_test, sgd_y_pred)
sgd_accuracy = round(sgd_accuracy*100, 6)
print("SGD Regressor Model Accuracy:", sgd_accuracy, "%")
print()

sgd_mae = mean_absolute_error(y_test, sgd_y_pred)
sgd_mse = mean_squared_error(y_test, sgd_y_pred)
sgd_rmse = sgd_mse*(1/2.0)

print("MAE:", sgd_mae)
print("MSE:", sgd_mse)
```

```
print("RMSE:", sgd_rmse)
print()
```

Let us plot a graph of the **Actual vs. Predicted Target**.

```
plt.scatter(y_test, sgd_y_pred)
plt.grid()
plt.xlabel('Actual Y')
plt.ylabel('Predicted Y')
plt.title('Actual vs. Predicted Target')
plt.show()
```

Let us plot a graph of the **Test vs. Predicted Data**.

```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, label="original")
plt.plot(x_ax, sgd_y_pred, label="predicted")
plt.title("Test vs. Predicted Data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

Model Comparison

1. Evaluation Matrix Comparison

```
#Performing evaluation matrix comparison
model_table = pd.DataFrame(columns = ["Comparison Matrix", "LR Model",
"SGD Model"])
model_table["Comparison Matrix"] = ["Accuracy (Variance Score)", "R2
Score", "MAE", "MSE", "RMSE"]
model_table["LR Model"] = [lin_accuracy, lin_score, lin_mae, lin_mse,
lin_rmse]
model_table["SGD Model"] = [sgd_accuracy, sgd_score, sgd_mae, sgd_mse,
sgd_rmse]

print(tabulate(model_table, headers = 'keys', tablefmt = 'psql',
numalign="left"))
```

2. Prediction Comparison - A: Scatter Plot

Let us perform a comparison of the **Actual vs. Predicted Target** for the **Scatter Plot**.

```
plt.scatter(y_test, lin_y_pred, c="b", marker="d", label='LR')
plt.scatter(y_test, sgd_y_pred, c="r", marker=".", label='SGD')
plt.xlabel('Actual Y')
```

```
plt.ylabel('Predicted Y')
plt.title('Actual Vs. Predicted Target - LR Vs. SGD')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid()
plt.show()
```

3. Prediction Comparison - B: Line Graph

Let us perform a comparison of **LR vs. SGD Prediction** for the **Line Graph**.

```
x_ax = range(len(sgd_y_pred))
plt.plot(x_ax, lin_y_pred, c="C0", linestyle="-", linewidth = 2,
label="LR Prediction")
plt.plot(x_ax, sgd_y_pred, c="C2", linestyle=":", linewidth = 4,
label="SGD Prediction")
plt.title("LR Vs. SGD Prediction")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```

4. Prediction Comparison - C: Target Prediction Table

Let us compare the **LR Predicted value** and **SGD Predicted value**.

```
prediction_table = pd.DataFrame(columns=["LR Predicted Value", "SGD
Predicted Value"])
prediction_table["LR Predicted Value"] = lin_y_pred
prediction_table["SGD Predicted Value"] = sgd_y_pred

print(tabulate(prediction_table, headers = 'keys', tablefmt = 'psql',
numalign="left"))
```

Note: In this topic, we saw the use of the linear regression methods, but in the next topic we will be working on "Logistic Regression".