Signitive TECHNOLOGIES

Signify the Learning

Certification Training Courses for Graduates and Professionals

www.signitivetech.com

Regression

Linear Regression

Linear Regression

- Machine learning, more specifically the field of predictive modeling is primarily concerned with minimizing the error of a model or making the most accurate predictions possible, at the expense of explainability.
- Linear regression is a **linear model**, e.g. a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).
- When there is a single input variable (x), the method is referred to as **simple linear regression**. When there are **multiple input variables**, literature from statistics often refers to the method as multiple linear regression.
- The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

Linear Regression

Linear regression is very good to answer the following questions:

- Is there a relationship between 2 variables?
- How strong is the relationship?
- Which variable contributes the most?
- How accurately can we estimate the effect of each variable?
- How accurately can we predict the target?
- Is the relationship linear?
- Is there an interaction effect?

Linear regression is a statistical model that examines the linear relationship between two (Simple Linear Regression) or more (Multiple Linear Regression) variables — a dependent variable and independent variable(s). Linear relationship basically means that when one (or more) independent variables increases (or decreases), the dependent variable increases (or decreases) too

Linear Regression

- The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

A relationship between variables Y and X is represented by this equation:

$$\boxed{Y = \beta_0 + \beta_1 * X}$$

Y: Dependent variable—or the variable we are trying to predict or estimate
X: Independent variable—the variable we are using to make predictions
β_1: Slope of the regression line—it represent the effect X has on Y. In other words, if X increases by 1 unit, Y will increase by exactly β_1 units. (This is true only if we know that X and Y have a linear relationship. In almost all linear regression cases, this will not be true!)
β_0: Constant or Y-intercept. If X equals 0, Y would be equal to β_0

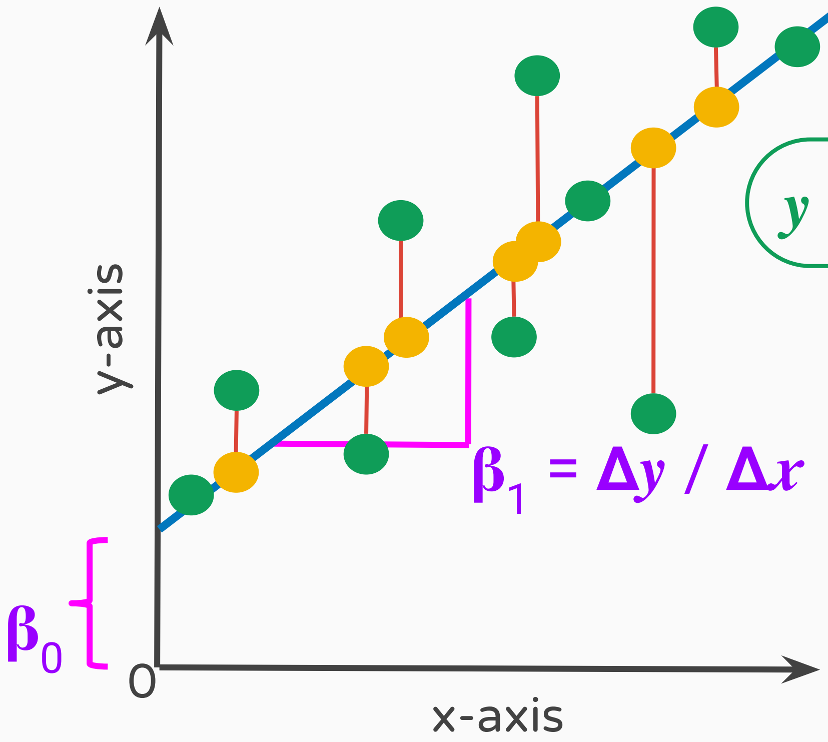
Linear Regression

- The values β_0 and β_1 must be chosen so that they minimize the error. If sum of squared error is taken as a metric to evaluate the model, then goal to obtain a line that best reduces the error.

$$\boxed{\text{Error} = \sum_{i=1}^{n} (\text{actual\_output } Y - \text{predicted\_output } \hat{Y})^2}$$

- We square the error, because the prediction can be either above or below the true value, resulting in a negative or positive difference respectively. If we don’t square the error, then positive and negative point will cancel out each other.
- While doing linear regression our objective is to fit a line through the distribution which is nearest to most of the points. Hence reducing the distance (error term) of data points from the fitted line.

Linear Regression



$$y = \beta_0 + \beta_1 * x$$

Linear Regression

- The Green Dots are the observed or actual values of x and y
- The Yellow Dots are the predicted values of x and y
- The Blue Line is the least squares line or regression line or best fit line
- The Red Lines are residuals which are the distance between the observed value and the least square line or predicted value i.e. Error

Linear Regression

Exploring '$\boldsymbol{\beta_1}$':

- $\boldsymbol{\beta_1}$ explains the change in Y with a change in X by one unit. In other words, if we increase the value of 'X' by one unit then what will be the change in value of Y
- If $\boldsymbol{\beta_1} > 0$, then x(predictor) and y(target) have a positive relationship. That is increase in x will increase y.
- If $\boldsymbol{\beta_1} < 0$, then x(predictor) and y(target) have a negative relationship. That is increase in x will decrease y.

Linear Regression

Exploring 'β_0':

- If the model does not include x=0, then the prediction will become meaningless with only β_0. For example, we have a dataset that relates height(x) and weight(y). Taking x=0(that is height as 0), will make equation have only β_0 value which is completely meaningless as in real-time height and weight can never be zero. This resulted due to considering the model values beyond its scope.
- If the model includes value 0, then 'β_0' will be the average of all predicted values when x=0. But, setting zero for all the predictor variables is often impossible.
- The value of β_0 guarantee that residual have mean zero. If there is no 'β_0' term, then regression will be forced to pass over the origin. Both the regression coefficient and prediction will be biased.

Linear Regression

For model with one Predictor (X):

Intercept Calculation:

$$\beta_0 = \text{mean}(Y) - \beta_1 * \text{mean}(X)$$

Coefficient Calculation:

$$\beta_1 = \frac{{}_{i=1}\Sigma^n (x_i - \text{mean}(x))(y_i - \text{mean}(y))}{{}_{i=1}\Sigma^n (x_i - \text{mean}(x))^2}$$

$$\beta_1 = \text{Correlation\_Coefficient}(r)*(\text{std\_dev}(Y) / \text{std\_dev}(X))$$

Linear Regression

Residual Analysis:

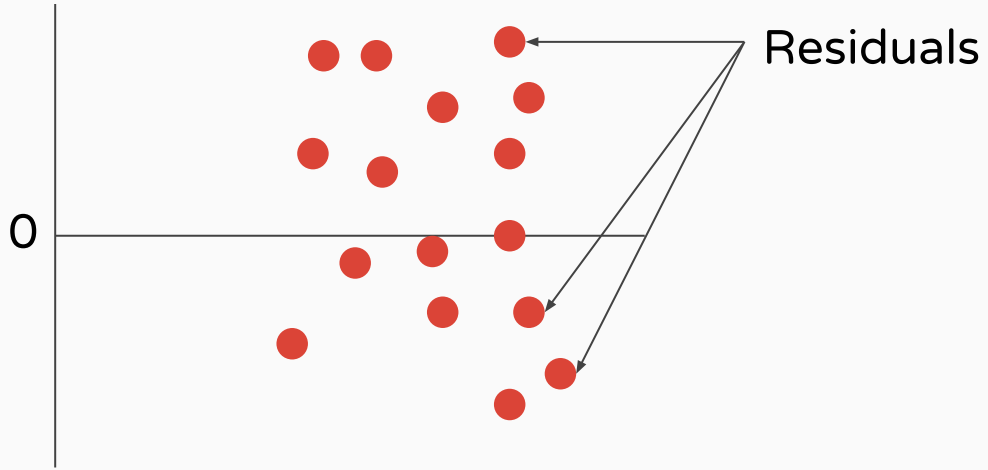
- Randomness and unpredictability are the two main components of a regression model.
- Consider, we have a dataset which predicts sales of juice when given a temperature of place. Value predicted from regression equation will always have some difference with the actual value. Sales will not match exactly with the true output value. This difference is called as residue.
- Residual plot helps in analyzing the model using the values of residues. It is plotted between predicted values and residue. Their values are standardized. The distance of the point from 0 specifies how bad the prediction was for that value. If the value is positive, then the prediction is low. If the value is negative, then the prediction is high. 0 value indicates prefect prediction. Detecting residual pattern can improve the model.

Linear Regression

Residual Analysis:

Non-random pattern of the residual plot indicates that the model is,

- Missing a variable which has significant contribution to the model target
- Missing to capture non-linearity (using polynomial term)
- No interaction between terms in model



Linear Regression

Metrics for model evaluation:

R-Squared value (R^2 or R2 Score**):**
This value ranges from 0 to 1. Value ‘1’ indicates predictor perfectly accounts for all the variation in Y. Value ‘0’ indicates that predictor ‘x’ accounts for no variation in ‘y’.

- R^2 of 0 means that the dependent variable cannot be predicted from the independent variable
- R^2 of 1 means the dependent variable can be predicted without error from the independent variable
- An R^2 between 0 and 1 indicates the extent to which the dependent variable is predictable. An R^2 of 0.20 means that 20 percent of the variance in Y is predictable from X; an R^2 of 0.40 means that 40 percent is predictable; and so on.

Linear Regression

Metrics for model evaluation:

1. Regression sum of squares (SSR): This gives information about how far estimated regression line is from the horizontal ‘no relationship’ line (average of actual output).

$$SSR = {}_{i=1}\Sigma^{n} (\text{predicted\_output } \hat{Y} - \text{mean}(Y))^2$$

2. Sum of Squared error (SSE): How much the target value varies around the regression line (predicted value).

$$SSE = {}_{i=1}\Sigma^{n} (\text{actual\_output } Y - \text{predicted\_output } \hat{Y})^2$$

3. Total sum of squares (SSTO): This tells how much the data point move around the mean.

$$SSTO = {}_{i=1}\Sigma^{n} (\text{actual\_output } Y - \text{mean}(Y))^2$$

Linear Regression

Metrics for model evaluation:
Coefficient of Determination (R Square): It suggests the proportion of variation in Y which can be explained with the independent variables.

$$\text{R-Square } (R^2) = 1 - (\text{ SSE / SSTO})$$

Is the range of R-Square always between 0 to 1?

Value of R^2 may end up being negative if the regression line is made to pass through a point forcefully. This will lead to forcefully making regression line to pass through the origin (no intercept) giving an error higher than the error produced by the horizontal line. This will happen if the data is far away from the origin.

Linear Regression

Metrics for model evaluation:

Mean Squared Error (MSE):

$$MSE = \frac{{}_{i=1}\Sigma^{n} (Y_i - \hat{Y}_i)^2}{N}$$

Mean Absolute Error (MAE):

$$MSE = \frac{{}_{i=1}\Sigma^{n} |Y_i - \hat{Y}_i|}{N}$$

Here N is the total number of observations.

Linear Regression

Metrics for model evaluation:

Root Mean Squared Error (RMSE):

$$\sqrt{\frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{N}}$$

Root Mean Squared Log Error (RMSLE):

$$\sqrt{\frac{\sum_{i=1}^{n} (\log(p_i + 1) - \log(a_i + 1))^2}{N}}$$

p_i Predicted Value & a_i Actual Value

- If both predicted and actual values are small: RMSE = RMSLE
- If either predicted or the actual value is big: RMSE > RMSLE
- If both predicted and actual values are big: RMSE > RMSLE (RMSLE becomes almost negligible)

Linear Regression

Multilinear Regression:

If we have more than one independent variables then we will use Multilinear Regression. It takes the following equation:

$$\boxed{y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \beta_3 * x_3}$$

What does each term represent?

- y is the response
- $x_1 x_2 x_3$ are features
- β_0 is the intercept
- β_1 β_2 β_3 are different coefficients for $x_1 x_2 x_3$ respectively

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Import Libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
```

- Load Dataset:

```
df = pd.read_csv('../Sample.csv')
```

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Check the relationship between Independent Variables and Dependent Variables:
- If there is One independent Variable, use scatter plot

```
plt.plot(df.X, df.Y)
plt.show()
```

- If there are More than One independent Variables, use pairplot plot

```
sb.pairplot(df)
plt.show()
```

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Check the correlation coefficient matrix to determine the relationship between independent variable and dependent variables and also it is used to check relationship between independent variables (if more than one) among themselves.

  ```
  df.corr()
  ```

- Do the Data preprocessing if any including missing value handling, rescaling the features, categorical to numeric conversion and outlier detection and removal

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Preparing the Data i.e. Separating Features and Target: Divide the data into "attributes" and "labels". Attributes are the independent variables while labels are dependent variables whose values are to be predicted.

  x = *Contains Dataset Attributes or Features*
  y = *Contains Dataset Labels or Target*

- Split the Data: Split the Dataset into the training and testing sets.

  ```
  from sklearn.model_selection import train_test_split
  x_train, x_test, y_train, y_test = train_test_spilt(x, y, test_size = 0.2)
  ```

Here test_size=0.2 refers that Testing Data is 20% and Training Data is 80%

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Training the Algorithm: Import the Linear Regression class, make the object of this class and by using fit() method, training data will get fitted to this class.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train, Y_train)
```

The linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data.

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Retrieve the Intercept:

  ```
  print(model.intercept_)
  ```

- Retrieve the Slope (coefficient of x):

  ```
  print(model.coef_)
  ```

- Making Predictions: Make predictions on testing data using predict()

  ```
  y_predicted = model.predict(x_test)
  ```

The y_predicted is a numpy array that contains all the predicted values for the input values x_test

Linear Regression

Steps to Perform Linear Regression using Scikit-Learn:

- Evaluating the Algorithm: Calculate R^2 value, MSE, MAE, Mean Squared Log Error and RMSE

```
        from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error, mean_squared_log_error

        r2 = r2_score(y_test, y_predicted)
        mae = mean_absolute_error(y_test, y_predicted)
        mse = mean_squared_error(y_test, y_predicted)
        msle = mean_squared_log_error(y_test, y_predicted)
        rmse = np.sqrt(mse)
```

R^2 value should be MAXIMUM close to 1 & RMSE should be LESS close to 0

Linear Regression: Assumptions in Linear Regression Analysis

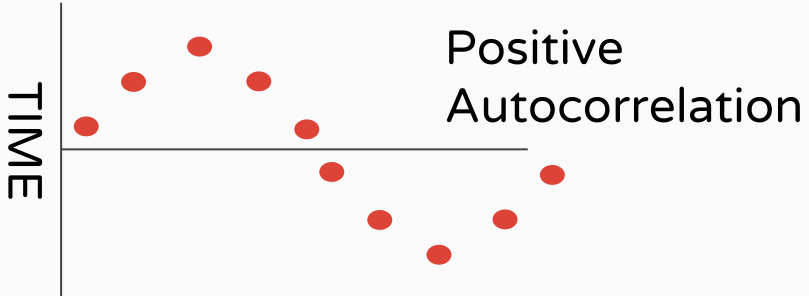
1. Linear and Additive:

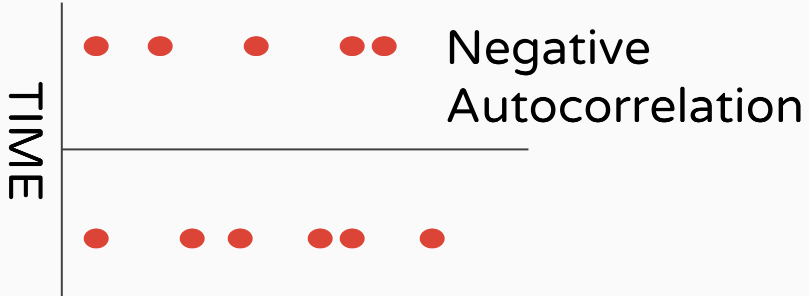
- There should be a linear and additive relationship between dependent (response) variable and independent (predictor) variable(s). A linear relationship suggests that a change in response Y due to one unit change in X is constant, regardless of the value of X. An additive relationship suggests that the effect of X on Y is independent of other variables.
- How to Check: Plotting Scatter Plots or Pair Plots
- If you fit a linear model to a non-linear, non-additive data set, the regression algorithm would fail to capture the trend mathematically, thus resulting in an inefficient model. Also, this will result in erroneous predictions on an unseen data set.

Linear Regression: Assumptions in Linear Regression Analysis

2. Autocorrelation:

- There should be no correlation between the residual (error) terms. Absence of this phenomenon is known as Autocorrelation.
- The presence of correlation in error terms drastically reduces model's accuracy. This usually occurs in time series models where the next instant is dependent on previous instant. If the error terms are correlated, the estimated standard errors tend to underestimate the true standard error.
- How to check: Plot residual vs time plot and look for the seasonal or correlated pattern in residual values.





Linear Regression: Assumptions in Linear Regression Analysis

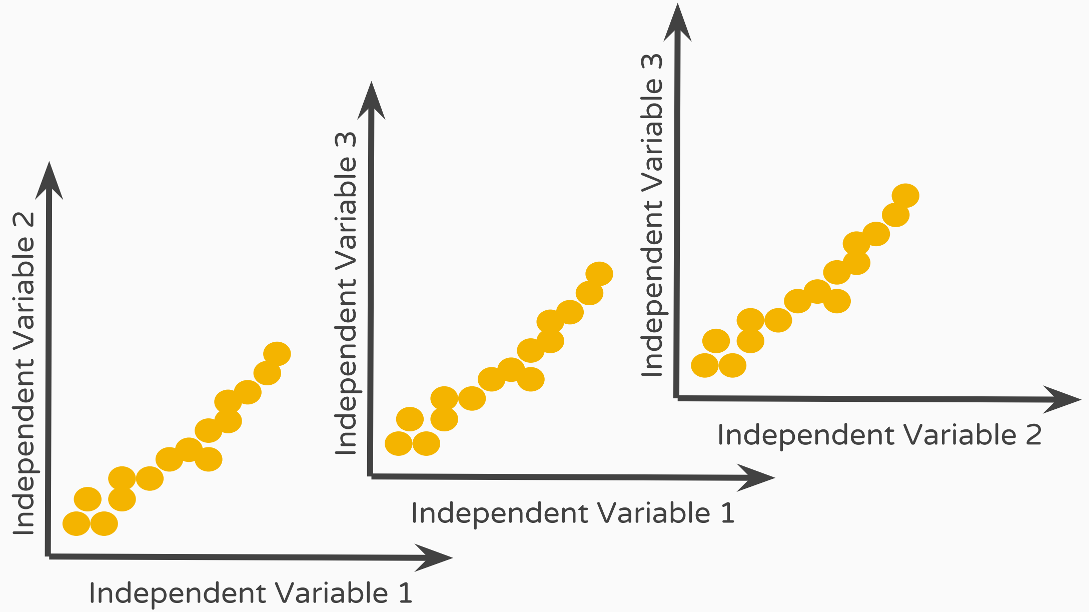
3. Multicollinearity:

- This phenomenon exists when the independent variables are found to be moderately or highly correlated. In a model with correlated variables, it becomes a tough task to figure out the true relationship of a predictors with response variable. In other words, it becomes difficult to find out which variable is actually contributing to predict the response variable.
- Another point, with presence of correlated predictors, the standard errors tend to increase. And, with large standard errors, the confidence interval becomes wider leading to less precise estimates of slope parameters
- Also, when predictors are correlated, the estimated regression coefficient of a correlated variable depends on which other predictors are available in the model. If this happens, you'll end up with an incorrect conclusion that a variable strongly / weakly affects target variable. Since, even if you drop one correlated variable from the model, its estimated regression coefficients would change. That's not good!

Linear Regression: Assumptions in Linear Regression Analysis

3. Multicollinearity:

- How to check: Using scatter plot to visualize correlation effect among variables. Check correlation matrix to determine the correlation coefficient value for each independent variable



Correlation Matrix

	IV1	IV2	IV3
IV1	1.00	0.98	0.91
IV2	0.98	1.00	0.95
Iv3	0.91	0.95	1.00

Linear Regression: Assumptions in Linear Regression Analysis

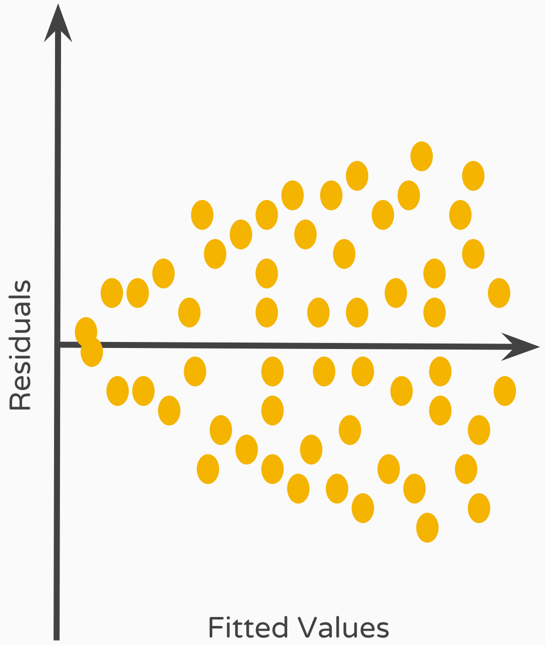
4. Heteroskedasticity:

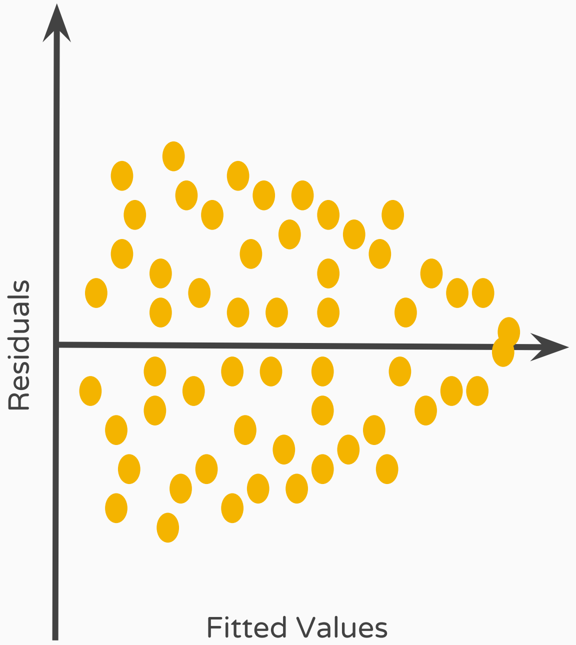
- The error terms must have constant variance. This phenomenon is known as homoscedasticity. The presence of non-constant variance is referred to heteroskedasticity.
- The presence of non-constant variance in the error terms results in heteroskedasticity. Generally, non-constant variance arises in presence of outliers or extreme leverage values. Look like, these values get too much weight, thereby disproportionately influences the model's performance.
- How to check: Draw the residual vs fitted values plot

Linear Regression: Assumptions in Linear Regression Analysis

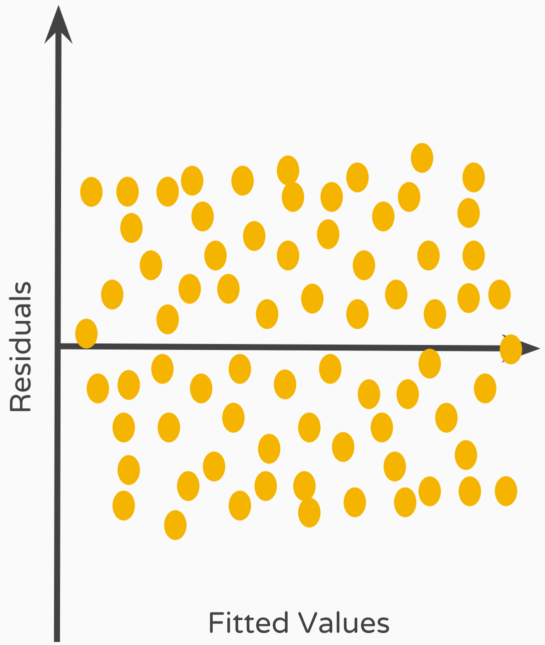
4. Heteroskedasticity:

Heteroskedasticity





Homoscedasticity



Linear Regression: Assumptions in Linear Regression Analysis

5. Normal Distribution:

- Error terms should have a Normal Distribution.

6. Gaussian Distributions:

- Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution.

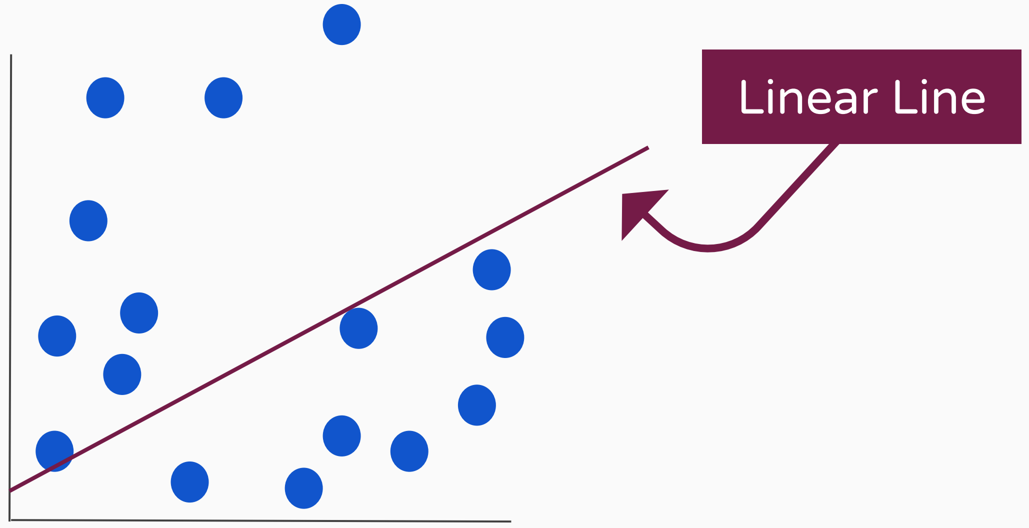
7. Rescale Inputs:

- Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

Polynomial Regression

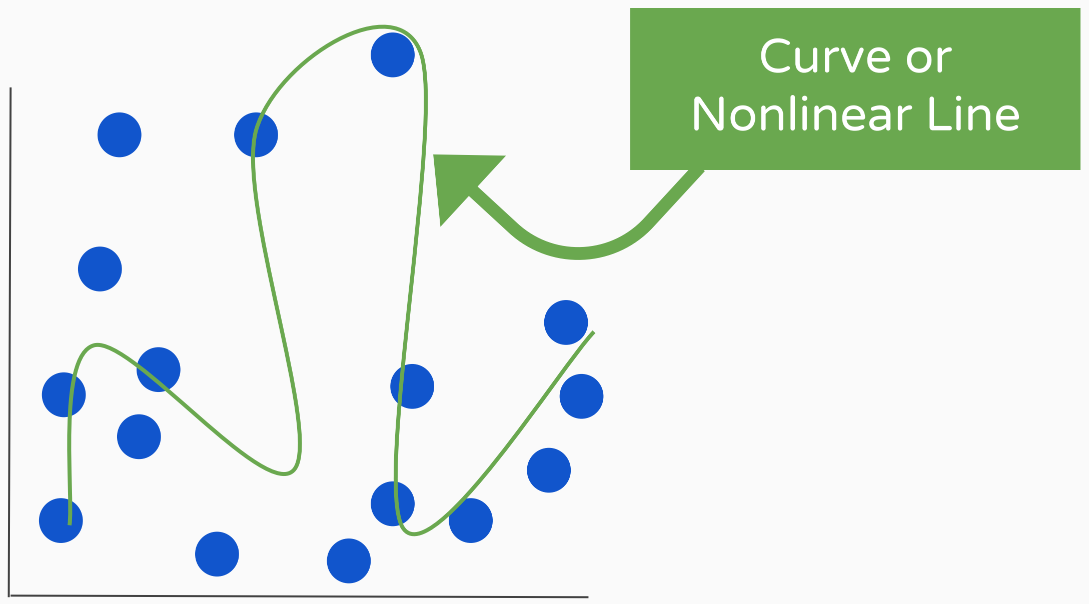
Polynomial Regression

- Linear regression requires the relation between the dependent variable and the independent variable to be linear. What if the distribution of the data was more complex as shown in the below figure? Can linear models be used to fit non-linear data?
- If we try to fit linear regression, errors which will get leads to misprediction which encounters high errors in the model.



Polynomial Regression

- When such problem occurs where the data seems to be more scattered or diversified, straight line or linear regression might not be the best way to describe the data.
- A curved or non linear line might be a better fit for such data.



Polynomial Regression

- Linear relationship between target X and feature X can be represented as:

$$\boxed{Y = \beta_0 + \beta_1 * X}$$

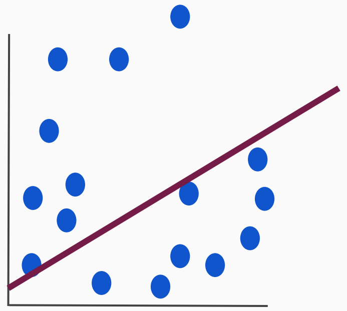
- However, linear line might not be best way to describe the relationship between X and Y. Here we want curve or nonlinear line which can be depicted by higher orders. To generate a higher order equation we can add powers of the original features as new features:

$$\boxed{Y = \beta_0 + \beta_1 * X + \beta_2 * X^2 + \beta_3 * X^3}$$

- This is still considered to be **linear model** as the coefficients/weights associated with the features are still linear. X^2 is only a feature. However the curve that we are fitting is **quadratic** in nature.

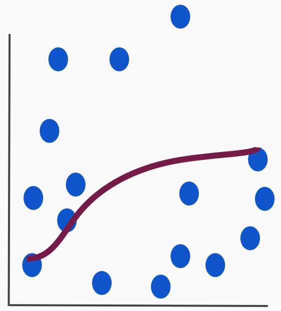
Polynomial Regression

- As we increase the power or degree of features, the line will get curved so as to cover maximum data points and minimize the errors between predicted and actual data points.



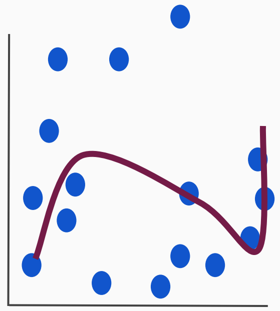
Linear Line of Degree = 1

$Y = \beta_0 + \beta_1 * X$



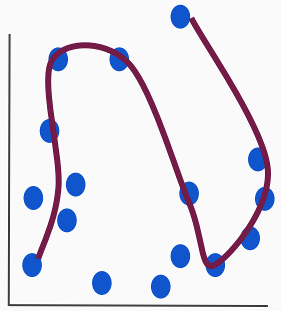
Curve Line of Degree = 2

$Y = \beta_0 + \beta_1 * X + \beta_2 * X^2$



Curve Line of Degree = 3

$Y = \beta_0 + \beta_1 * X + \beta_2 * X^2 + \beta_3 * X^3$



Curve Line of Degree = 4

$Y = \beta_0 + \beta_1 * X + \beta_2 * X^2 + \beta_3 * X^3 + \beta_4 * X^4$

Polynomial Regression

Steps to Perform Polynomial Regression using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables:
  - If there is One independent Variable, use scatter plot
  - If there are More than One independent Variables, use pairplot plot
- Check the correlation coefficient matrix to determine the relationship between independent variable and dependent variables and also it is used to check relationship between independent variables (if more than one) among themselves.
- Do the Data preprocessing if any including missing value handling, rescaling the features, categorical to numeric conversion and outlier detection and removal

Polynomial Regression

Steps to Perform Polynomial Regression using Scikit-Learn:

- Preparing the Data i.e. Separating Features and Target: Divide the data into "attributes" and "labels". Attributes are the independent variables while labels are dependent variables whose values are to be predicted.

  x = *Contains Dataset Attributes or Features*
  y = *Contains Dataset Labels or Target*

Polynomial Regression

Steps to Perform Polynomial Regression using Scikit-Learn:

- Convert Features to Polynomial Features i.e. increase the power or degree of feature variables

  ```
  from sklearn.preprocessing import PolynomialFeatures
  poly = PolynomialFeatures(degree = 2)
  X_poly = poly.fit_transform(x)
  ```

How does PolynomialFeatures() function work exactly?

Let's say we have a matrix of one feature: X=[a]
After increasing power of features i.e for degree = 2
X_poly_degree2 = [1, a, a^2]
Further increasing power of features i.e for degree = 3
X_poly_degree3 = [1, a, a^2, a^3]

Polynomial Regression

How does PolynomialFeatures() function work exactly?

Let's say we have a matrix of one feature: X=[a, b]

After increasing power of features i.e for degree = 2

X_poly_degree2 = [1, a, b, a^2, a*b, b^2]

Further increasing power of features i.e for degree = 3

X_poly_degree3 = [1, a, b, a^2, a*b, b^2, a^3, a^2*b, a*b^2, b^3]

Polynomial Regression

Steps to Perform Polynomial Regression using Scikit-Learn:

- Split the Data: Split the Dataset into the training and testing sets.

  ```
  from sklearn.model_selection import train_test_split
  x_train, x_test, y_train, y_test = train_test_spilt(X_poly, y, test_size = 0.2)
  ```

- Training the Algorithm: Import the Linear Regression class, make the object of this class and by using fit() method, training data will get fitted to this class.

  ```
  from sklearn.linear_model import LinearRegression
  model = LinearRegression()
  model.fit(x_train, Y_train)
  ```

Polynomial Regression

Steps to Perform Polynomial Regression using Scikit-Learn:

- Making Predictions: Make predictions on testing data using predict()

```
y_predicted = model.predict(x_test)
```

- Evaluating the Algorithm: Calculate R^2 value, MSE, MAE, MSLE, and RMSE

```
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error, mean_squared_log_error

r2 = r2_score(y_test, y_predicted)
mae = mean_absolute_error(y_test, y_predicted)
mse = mean_squared_error(y_test, y_predicted)
msle = mean_squared_log_error(y_test, y_predicted)
rmse = np.sqrt(mse)
```

Underfitting & Overfitting

Underfitting and Overfitting

Underfitting:

- A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.
- Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough.
- It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data.
- In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions.
- Underfitting can be avoided by using more data and also reducing the features by feature selection.

Underfitting and Overfitting

Overfitting:

- A statistical model is said to be overfitted, when we train it with a lot of data.
- When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too much of details and noise.
- The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.
- A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Underfitting and Overfitting

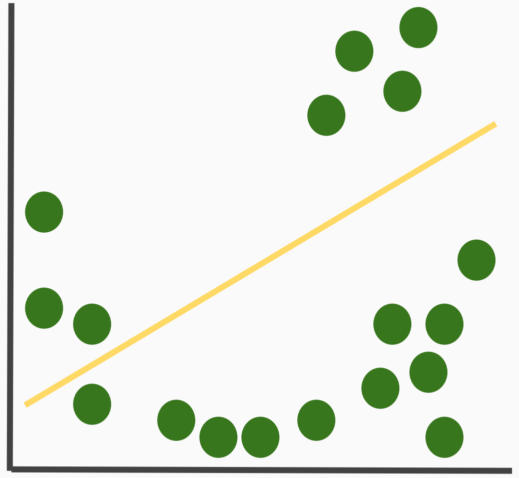
- The problem of Overfitting and Underfitting appears in the polynomial degree.
- The degree represents how much flexibility is in the model, with a higher power allowing the model freedom to hit as many data points as possible.
- An underfit model will be less flexible and cannot account for the data.
- Underfit model has **LOW VARIANCE**. Variance is how much a model changes in response to the training data. Variance refers to how much the model is dependent on the training data.
- Underfit model has **HIGH BIAS**. Bias means model makes a strong assumption about the data. When the model makes test predictions, the bias leads it to make inaccurate estimates. The model will fail to learn the relationship between features and target because of the bias.

Underfitting and Overfitting

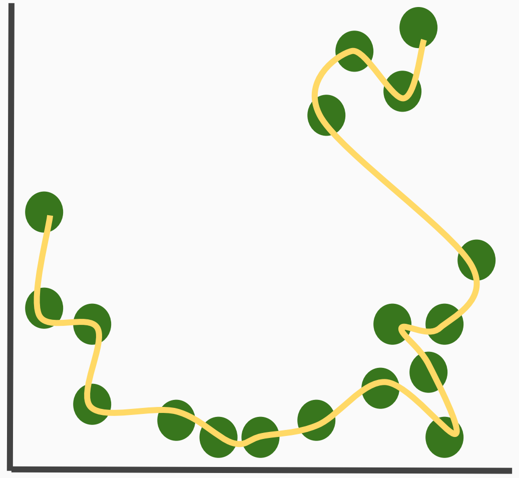
- To avoid Underfitting we can increase the power of degrees of features.
- With such a high degree of flexibility, the model does its best to account for every single training point.
- Further, the model has a great score on the training data because it gets close to all the points.
- However, model will cover almost all data points including Noise which would be not acceptable to make a Perfect Model, because this leads again wrong predictions of new or unseen data.
- Such a model is called as Overfit model. An Overfit model has **HIGH VARIANCE** and **LOW BIAS**.

Underfitting and Overfitting

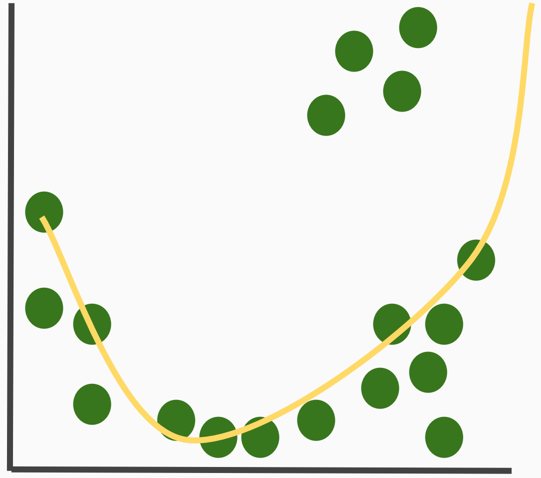
Underfitting v/s Overfitting: Visualizations:



Undefitting
High Bias
Low Variance



Overfitting
Low Bias
High Variance



Good/Robust Fit
Low Bias
Low Variance

Underfitting and Overfitting

How to avoid Overfitting:

The commonly used methodologies are:

- **Cross- Validation:** A standard way to find out-of-sample prediction error is to use 5-fold cross validation.
- **Early Stopping:** Its rules provide us the guidance as to how many iterations can be run before learner begins to over-fit.
- **Pruning:** Pruning is extensively used while building related models. It simply removes the nodes which add little predictive power for the problem in hand.
- **Regularization:** 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.

Underfitting and Overfitting

Good Fit in a Statistical Model:

- Ideally, the case when the model makes the predictions with 0 error, is said to have a *good fit* on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it we will have to look at the performance of our model with the passage of time, while it is learning from training dataset.
- With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to presence of noise and less useful details. Hence the performance of our model will decrease.
- In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point the model is said to have good skills on training dataset as well our unseen testing dataset.

Underfitting and Overfitting

Short Summary:

- **Overfitting:** This means too much reliance on the training data.
- **Underfitting:** This means a failure to learn the relationships in the training data.
- **High Variance:** This means model changes significantly based on training data.
- **High Bias**: This means assumptions about model lead to ignoring training data.

Bias-Variance Trade-Off

Bias-Variance Trade-Off

- The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X). The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.
- The prediction error for any machine learning algorithm can be broken down into three parts:
  - ➔ Bias Error
  - ➔ Variance Error
  - ➔ Irreducible Error

- The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

Bias-Variance Trade-Off

What is bias?

- Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.
- Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on train and test data.
- Bias are the simplifying assumptions made by a model to make the target function easier to learn.
- Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

Bias-Variance Trade-Off

What is bias?

- **Low-Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.
- Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.
- Bias occurs when an algorithm has limited flexibility to learn the true signal from a dataset.

Bias-Variance Trade-Off

What is variance?

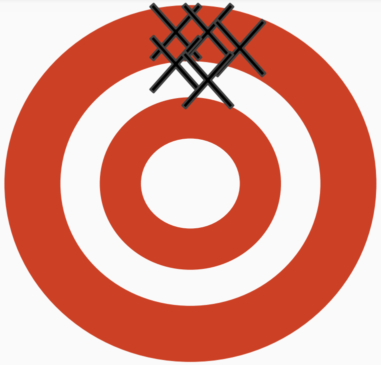
- Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.
- Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on train data but has high error rates on test data.
- Variance is the amount that the estimate of the target function will change if different training data was used.
- The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

Bias-Variance Trade-Off

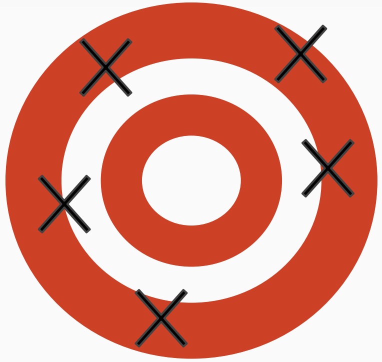
What is variance?

- Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.
- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.
- Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance.
- Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.
- Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Bias-Variance Trade-Off



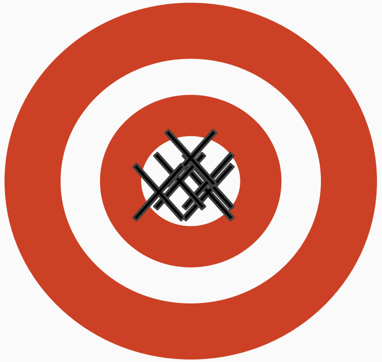
Low Variance
High Bias
(Undefitting)



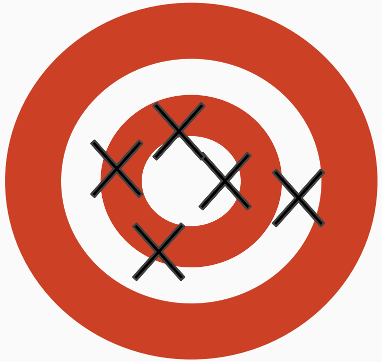
High Variance
High Bias

High Bias and Low Variance algorithms train models that are consistent but inaccurate on average

High Variance and Low Bias algorithms train models that are accurate on average but inconsistent



Low Variance
Low Bias



High Variance
Low Bias
(Overfitting)

Bias-Variance Trade-Off

Bias-Variance Trade-Off:

- The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.
- Parametric or linear machine learning algorithms often have a high bias but a low variance.
- Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.
- The parameterization of machine learning algorithms is often a battle to balance out bias and variance.
- Low variance algorithms tend to be less complex with simple or rigid underlying structure. Examples: Regress, Naive Bayes
- Low bias algorithms tend to be more complex with flexible underlying structure. Examples: Decision Trees, KNN

Bias-Variance Trade-Off

Bias-Variance Trade-Off:

- If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.
- This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time. There is no escaping the relationship between bias and variance in machine learning.

➔ Increasing the bias will decrease the variance.

➔ Increasing the variance will decrease the bias.

- There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem.

Bias-Variance Trade-Off

Bias-Variance Trade-Off: How to configure?

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.
- Regression can be regularized to further reduce complexity.
- Decision Trees can be pruned to reduce complexity.
- Algos that are not complex enough produce underfit models that can’t learn the signal from the data.
- Algos that are too complex produce overfit models that memorize the noise instead of the signal.

Bias-Variance Trade-Off

Bias-Variance Trade-Off:
Total Error:

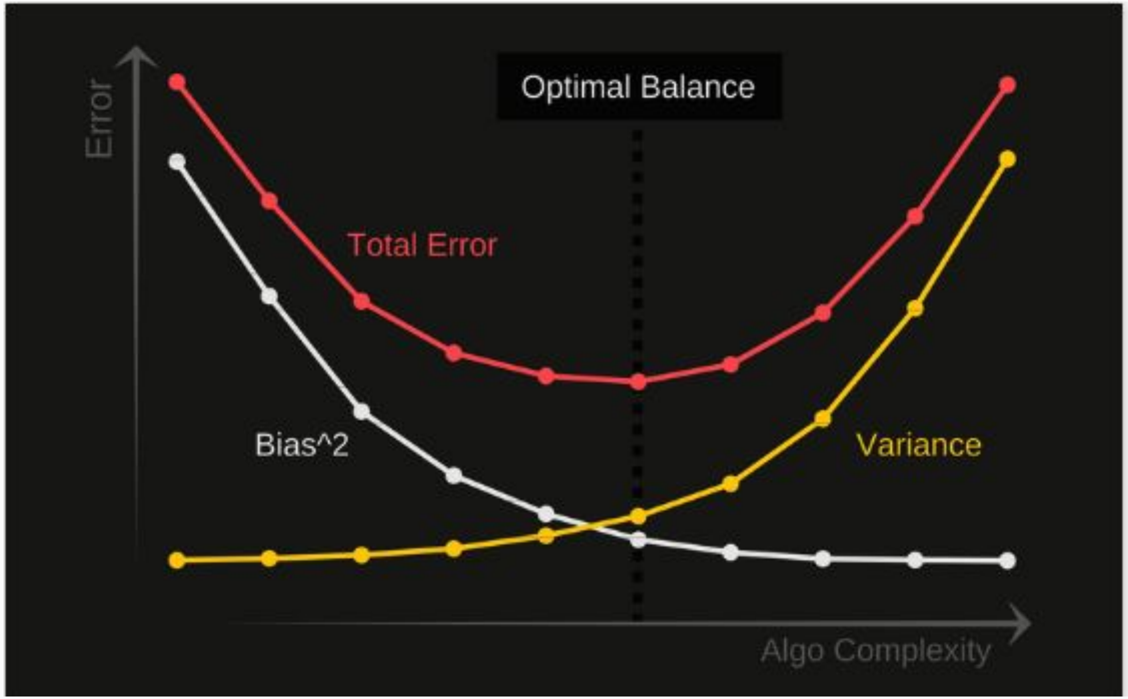
- To build a good model, we need to find a good balance between bias and variance such that it minimizes the total error.
- Total error breaks down as:

$$\text{Total\_Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible\_Error}$$

- Irreducible error is noise that can’t be reduced by algorithms. It can sometimes be reduced by better data cleaning.
- An optimal balance of bias and variance would never overfit or underfit the model.

Bias-Variance Trade-Off

Bias-Variance Trade-Off:
Total Error:



Regularization Techniques

Regularization Techniques

Overfit in machine learning algorithms:

- Having more features may seem like a perfect way for improving the accuracy of our trained model (reducing the loss or error) — because the model that will be trained will be more flexible and will take into account more parameters.
- On the other hand, we need to be extremely careful about **overfitting** the data. As we know, every dataset has noisy samples.
- The inaccuracies can lead to a low-quality model if not trained carefully. The model might end up memorizing the noise instead of learning the trend of the data.
- Overfit can happen in linear models as well when dealing with multiple features. If not filtered and explored up front, some features can be more destructive than helpful, repeat information that already expressed by other features and add high noise to the dataset.

Regularization Techniques

- With any predictive model, we seek to strike a balance between bias and variance.
- If our model has too few features or oversimplifies reality, the algorithm is going to underfit the training data and probably not perform much better.
- If our model has too many features or features with multicollinearity, the algorithm will overfit the training data and not generalize well.
- Regularization is an effective tool to deal with overfit or high bias.
- When fitting a model, we want to minimize the loss function or error but if the loss function is very close to zero, we are in danger of overfitting. If the loss function is too far removed from zero, we are in danger of underfitting.
- Regularization adds a penalty term to the loss function to help deal with either of these scenarios.
- Regularization favors simpler models to more complex models to prevent our model from overfitting to the data.

Regularization Techniques

Regularization techniques in Generalized Linear Models (GLM) are used during a modeling process for many reasons. A regularization technique helps in the following many ways:

1. It doesn’t assume any particular distribution of the dependent variable (DV). The DV can follow any distribution. Hence it is called as the Generalized Linear Models (GLMs)
2. It addresses Variance-Bias Tradeoffs. Generally, it will lower the variance from the model.
3. It is more robust to handle multicollinearity.
4. It helps in sparse data (observations < features) handling.
5. It gives natural feature selection.
6. It provides more accurate prediction on new data as it minimizes overfitting on the train data.
7. It provides easier interpretation of the output.

Regularization Techniques

What is a regularization technique?

- A regularization technique is in simple terms a penalty mechanism which applies shrinkage (driving them closer to zero) of coefficient to build a more robust and parsimonious model.
- Regularized machine learning model, is a model that its loss function contains another element that should be minimized as well.
- **Regularization** is a technique to avoid high variance in our model. To do this we will introduce a **penalty term** to the cost function above. By adding this penalty term we will prevent the coefficients of the linear function from getting too large. Introducing this bias should decrease the variance and thus prevent overfitting.

Regularization Techniques

Which are regularization techniques?

- There are Three forms of regularization: Ridge, Lasso and Elastic Net

  Ridge Regularization, also called as L2 Penalty or L2 Norm.

  Lasso Regularization, also called as L1 Penalty or L1 Norm.

  Elastic Net Regularization: Ridge+Lasso (L1 + L2) Norms

Note: Before using either regularization method, we must standardize all the data. Regularization penalizes the magnitude of coefficients so all predictor variables must be on the same scale.

Regularization Techniques

Lasso Regularization:

- Lasso stands for Least Absolute Shrinkage Selector Operator.
- Lasso assigns a penalty to the coefficients in the linear model and eliminates variables with coefficients that zero. This is called shrinkage or the process where data values are shrunk to a central point such as a mean.
- This add regularization terms in the model which are function of absolute value of the coefficients of parameters. The coefficient of the parameters can be driven to zero as well during the regularization process. Hence this technique can be used for feature selection and generating more parsimonious model.
- By reducing the sum of absolute values of the coefficients, what Lasso Regularization (L1 Norm) does is to reduce the number of features in the model altogether to predict the target variable.

Regularization Techniques

Lasso Regularization:
Lasso = Sum of Squared Error + Sum of the absolute value of coefficients

$$L = {}_{i=1}\Sigma^{n} (Y_i - \hat{Y}_i)^2 + \lambda * {}_{i=1}\Sigma^{n} |\beta_i|$$

- Lasso adds a penalty equal to the absolute value of the magnitude of the coefficients multiplied by lambda.
- The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients. This penalty reduces the value of many coefficients to zero, all of which are eliminated.
- As λ gets significantly large β is forced to 0.
- Because we are using the absolute value, instead of the square of our coefficients, less important features' coefficients can be assigned a value of 0. This is particularly helpful for feature selection.

Regularization Techniques

Ridge Regularization:

- Ridge assigns a penalty that is the squared magnitude of the coefficients to the loss function multiplied by lambda.
- As Lasso does, ridge also adds a penalty to coefficients the model overemphasizes.
- This add regularization terms in the model which are function of square of coefficients of parameters.
- Coefficient of parameters can approach to zero but never become zero.
- Coefficients of unimportant features will shrink to **near zero.**
- By reducing the sum of square of coefficients, Ridge Regularization (L2 Norm) doesn’t necessarily reduce the number of features per se, but rather reduces the magnitude/impact that each features has on the model by reducing the coefficient value.

Regularization Techniques

Ridge Regularization:
Ridge = Sum of Squared Error + Sum of the squares value of coefficients

$$R = {}_{i=1}\Sigma^{n} (Y_i - \hat{Y}_i)^2 + \lambda * {}_{i=1}\Sigma^{n} \beta_i^2$$

- The value of lambda also plays a key role in how much weight you assign to the penalty for the coefficients.
- The larger your value of lambda, the more likely your coefficients get closer and closer to zero.
- When λ is significantly large the sum of squared errors term in our cost function becomes insignificant. Since we are looking to minimize this cost function, Ridge forces β to 0.
- As we increase λ value, the values in β decrease.

Regularization Techniques

Elastic Net Regularization:

- Elastic Net combines characteristics of both lasso and ridge.
- Elastic Net reduces the impact of different features while not eliminating all of the features.
- Both regularization does indeed prevent the model from overfitting.
- Lasso Regularization reduces the quantity of features while Ridge Regularization reduces the quality of features.
- In essence, both types of reductions are needed, and therefore it makes much more sense why Elastic Net (combination of Lasso and Ridge Regularization) would be the ideal type of regularization to perform on a model.

Regularization Techniques

Elastic Net Regularization:
Elastic Net = Ridge + Lasso

$$EN = {}_{i=1}\Sigma^{n} (Y_i - \hat{Y}_i)^2 + \lambda * \{ (1 - \alpha) {}_{i=1}\Sigma^{n} \beta_i^{2} + (\alpha) {}_{i=1}\Sigma^{n} | \beta_i | \}$$

- The α takes the value from 0 and 1
- If $\alpha = 0$, Ridge Regularization will be used and if $\alpha = 1$, Lasso Regularization will be used.
- However, if $\alpha = 0.5$ both regularizations will play an important role.

Regularization Techniques

Steps to Perform Regularization using Scikit-Learn:

- Import required libraries
- Load the Dataset
- Check the relationship between Independent and Dependent Variables
- Do the Data preprocessing
- Separate features and target
- Split the data into train and test part

Regularization Techniques

Steps to Perform Regularization using Scikit-Learn:

For Ridge Regularization:

- Training the Algorithm: Import the Ridge class, make the object of this class and by using fit() method, training data will get fitted to this class.

  ```
  from sklearn.linear_model import Ridge
  model = Ridge(alpha = 1.0)
  model.fit(x_train, Y_train)
  ```

- Here alpha stands for lambda value, by default its value is 1.0

- Check for coefficients values:
  ```
  print(model.coef_)
  ```

- Evaluate the model by checking MSE and R2 Score values

Regularization Techniques

Steps to Perform Regularization using Scikit-Learn:
For Lasso Regularization:

- Training the Algorithm: Import the Lasso class, make the object of this class and by using fit() method, training data will get fitted to this class.

  ```
  from sklearn.linear_model import Lasso
  model = Lasso(alpha = 1.0)
  model.fit(x_train, Y_train)
  ```

- Here alpha stands for lambda value, by default its value is 1.0

- Check for coefficients values:
  ```
  print(model.coef_)
  ```

- Evaluate the model by checking MSE and R2 Score values

Regularization Techniques

Steps to Perform Regularization using Scikit-Learn:
For Elastic Net Regularization:

- Training the Algorithm: Import the ElasticNet class, make the object of this class and by using fit() method, training data will get fitted to this class.

```
from sklearn.linear_model import ElasticNet
model = ElasticNet(alpha = 1.0, l1_ratio = 0.5)
model.fit(x_train, Y_train)
```

- Here alpha stands for lambda value, by default its value is 1.0
- Here l1_ratio = 0.5 stands, ElasticNet uses both regularizations
- Check for coefficients values:
  `print(model.coef_)`

- Evaluate the model by checking MSE and R2 Score values

Gradient Descent

Gradient Descent

Linear Regression:

- Linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables.
- Let **X** be the independent variable and **Y** be the dependent variable. We will define a linear relationship between these two variables as follows:

$$Y = mX + c$$

Loss Function:

- The loss is the error in our predicted value of **m** and **c**. Our goal is to minimize this error to obtain the most accurate value of **m** and **c**.

Loss Function i.e. $\text{MSE} = \dfrac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{N}$

Gradient Descent

The Gradient Descent Algorithm:

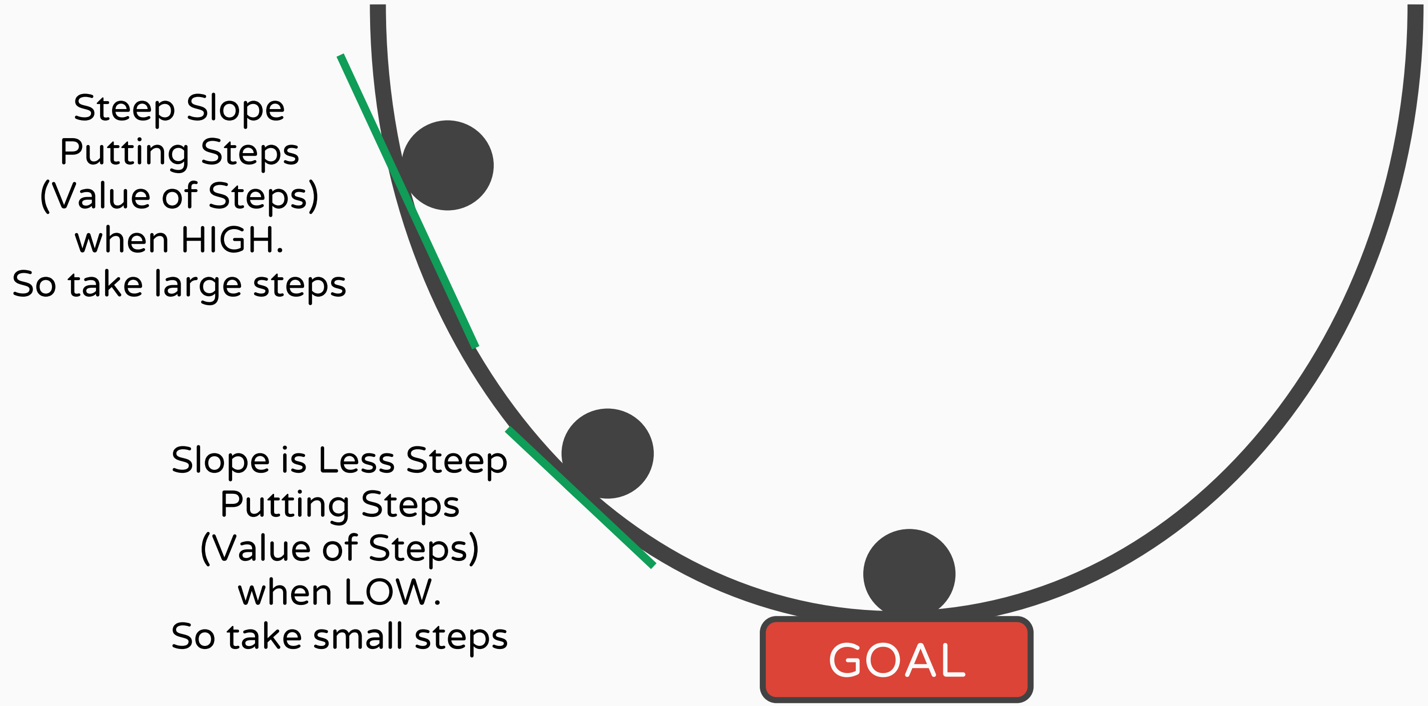
- Gradient descent is an iterative optimization algorithm to find the minimum of a cost or loss function i.e. minimize the error.
- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

Understanding the Gradient Descent Algorithm:

- Imagine a valley and a person with no sense of direction who wants to get to the bottom of the valley.
- He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep.
- He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Gradient Descent

Understanding the Gradient Descent Algorithm: Visualization



Gradient Descent

Let’s try applying gradient descent to **m** and **c** and approach it step by step:

1. Initially let m = 0 and c = 0. Let L be our learning rate. This controls how much the value of **m** changes with each step. L could be a small value like 0.0001 for good accuracy.
2. Calculate the partial derivative of the loss function with respect to m, plug in the current values of x, y, m and c in it to obtain the derivative value **D**.

$$\partial/\partial m = D_m = 1/n * \left({}_{i=1}\Sigma^n \, 2*(Y_i - \hat{Y}_i)*(-X_i) \right)$$

$$\partial/\partial m = D_m = -2/n * \left({}_{i=1}\Sigma^n \, (-X_i)*(Y_i - \hat{Y}_i) \right)$$

3. D_m is the value of the partial derivative with respect to **m**.

Gradient Descent

4. Similarly lets find the partial derivative with respect to **c**. D_c is the value of the partial derivative with respect to **c**.

$$\partial/\partial c = D_c = 1/n * ({}_{i=1}\Sigma^n -2*(Y_i - \hat{Y}_i))$$

$$\partial/\partial c = D_c = -2/n * ({}_{i=1}\Sigma^n (Y_i - \hat{Y}_i))$$

5. Now we update the current value of **m** and **c** using the following equation:

$$m = m - L * D_m$$

$$c = c - L * D_c$$

L is the Learning Rate value

Gradient Descent

6. We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy). The value of **m** and **c** that we are left with now will be the optimum values.

Summary:

- Now going back to our analogy, **m** can be considered the current position of the person. **D** is equivalent to the steepness of the slope and **L** can be the speed with which he moves.
- Now the new value of **m** that we calculate using the above equation will be his next position, and **L×D** will be the size of the steps he will take.
- When the slope is more steep (**D** is more) he takes longer steps and when it is less steep (**D** is less), he takes smaller steps.
- Finally he arrives at the bottom of the valley which corresponds to our loss = 0. Now with the optimum value of **m** and **c** our model is ready to make predictions.

Gradient Descent

Types of Gradient Descent Algorithms:

There are three types of Gradient Descent Algorithms:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-Batch Gradient Descent

Batch Gradient Descent:

- In the batch gradient descent, to calculate the gradient of the cost function, we need to sum all training examples for each steps.
- If we have 3 millions samples (m training examples) then the gradient descent algorithm should sum 3 millions samples for every epoch. To move a single step, we have to calculate each with 3 million times.
- Batch Gradient Descent is not good fit for large datasets

Gradient Descent

Stochastic Gradient Descent (SGD):

- In stochastic Gradient Descent, we use one example or one training sample at each iteration instead of using whole dataset to sum all for every steps.
- SGD is widely used for larger dataset trainings and computationally faster and can be trained in parallel.
- We need to randomly shuffle the training examples before calculating it.

Mini-Batch Gradient Descent:

- It is similar like SGD, it uses n samples instead of 1 at each iteration.

Thanks!

Signitive Technologies

Sneh Nagar, Behind ICICI Bank,
Chhatrapati Square, Nagpur 15

Landmark:
Bharat Petrol Pump
Chhatrapati Square

Contact: 9011033776
www.signitivetech.com

“Keep Learning, Happy Learning”

Best Luck!

Have a Happy Future