Dr. Ambedkar Institute of technology
Department of Electronics and Instrumentation engineering

21EIT504 Machine learning with python programming

Unit 3 : Training models

Prepared by : Ms. Nirmala Bai L

Assistant professor
Dept. of EIE
Dr. AIT

Training models

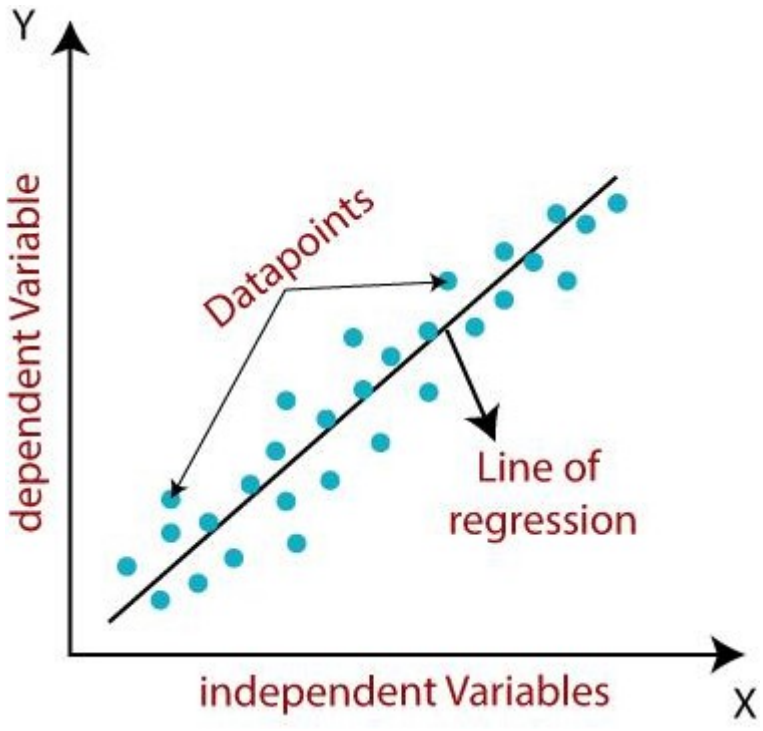
What is a machine learning Algorithm?
A machine learning algorithm is a mathematical method to find patterns in a set of data. Machine Learning algorithms are often drawn from statistics, calculus, and linear algebra. Some popular examples of machine learning algorithms include linear regression, decision trees, random forest.

Linear Regression in Machine Learning

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$$y= a_0 +a_1 x+ \varepsilon$$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

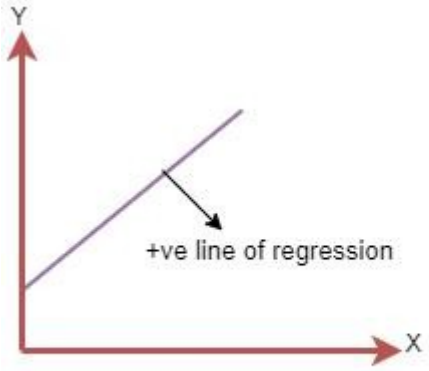
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

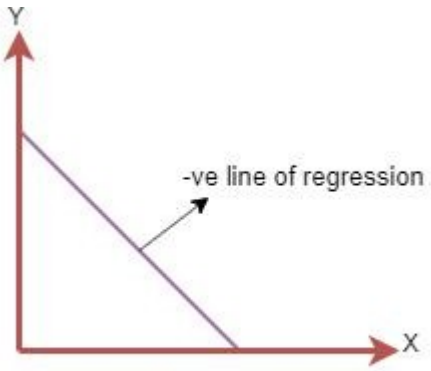
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.

- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.

- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
  Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
  It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.

- **No autocorrelations:**

  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.

Gradient Descent in Machine Learning

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

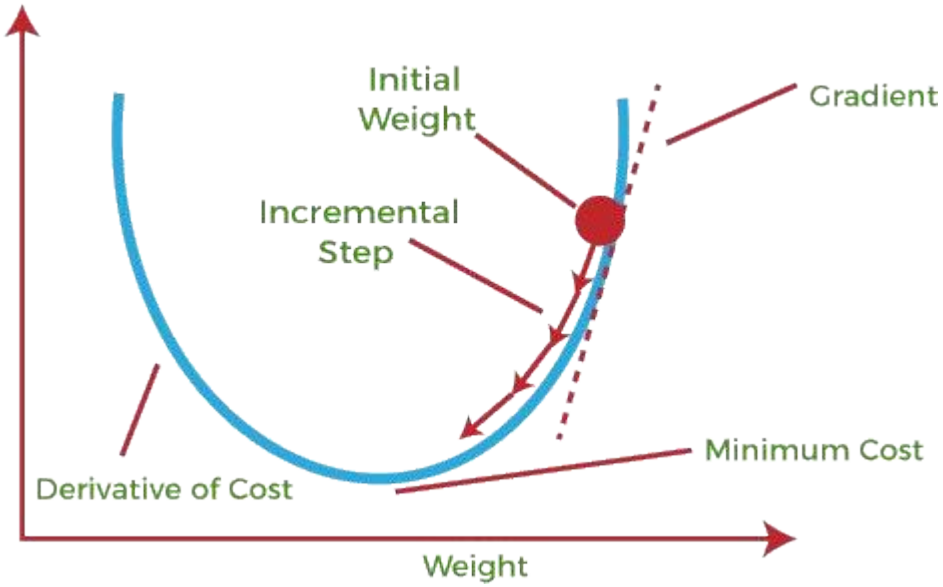
In this tutorial on Gradient Descent in Machine Learning, we will learn in detail about gradient descent, the role of cost functions specifically as a barometer within Machine Learning, types of gradient descents, learning rates, etc.

What is Gradient Descent or Steepest Descent?

Gradient descent was initially discovered by **"Augustin-Louis Cauchy"** in mid of 18th century. **Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.**

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. **The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.** To achieve this goal, it performs two steps iteratively:

- Calculates the first-order derivative of the function to compute the gradient or slope of that function.
- Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

What is Cost-function?

The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number. It helps to increase and improve machine learning efficiency by providing feedback to this model so that it can minimize error and find the local or global minimum. Further, it continuously iterates along the direction of the negative gradient until the cost function approaches zero. At this steepest descent point, the model will stop learning further. Although cost function and loss function are considered synonymous, also there is a minor difference between them. The slight difference between the loss function and the cost function is about the error within the training of machine learning models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

The cost function is calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms over known data to reduce the cost function.

Hypothesis:

Parameters:

Cost function:

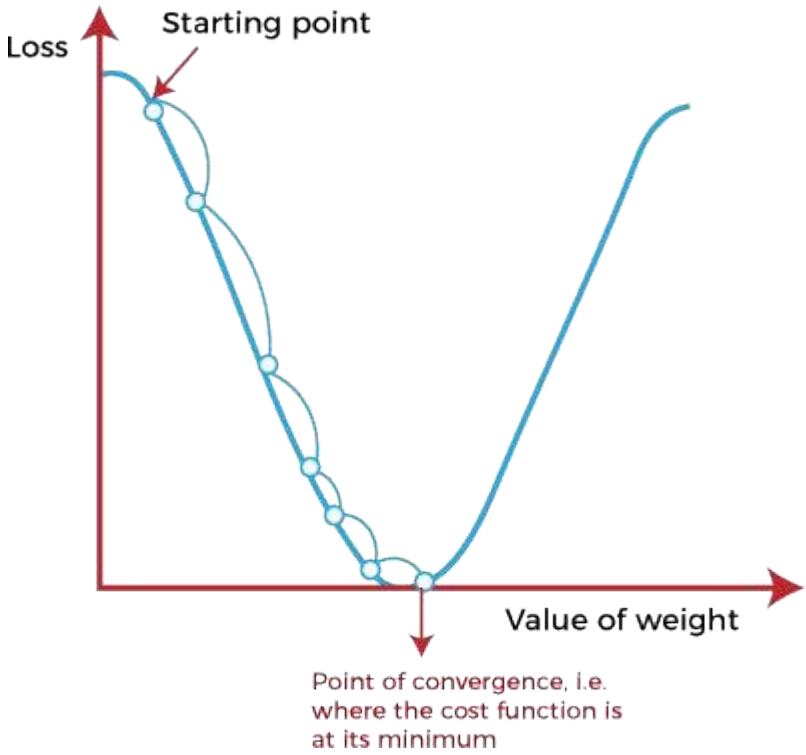
Goal:

How does Gradient Descent work?

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

1. $Y=mX+c$

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called **a point of convergence.**

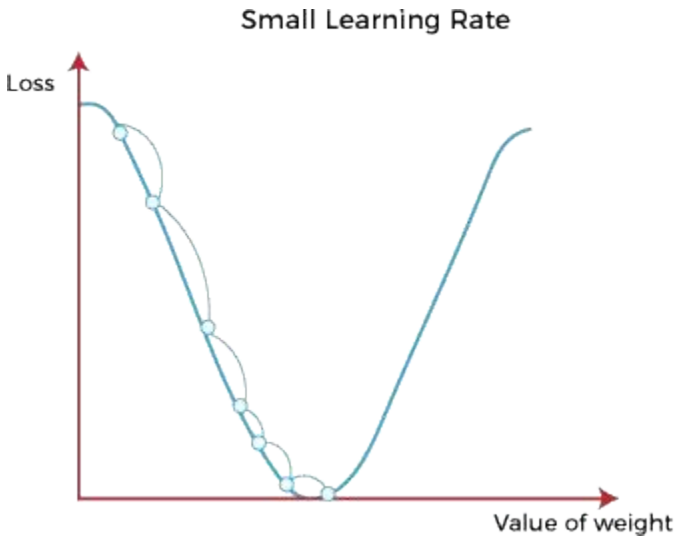
The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:

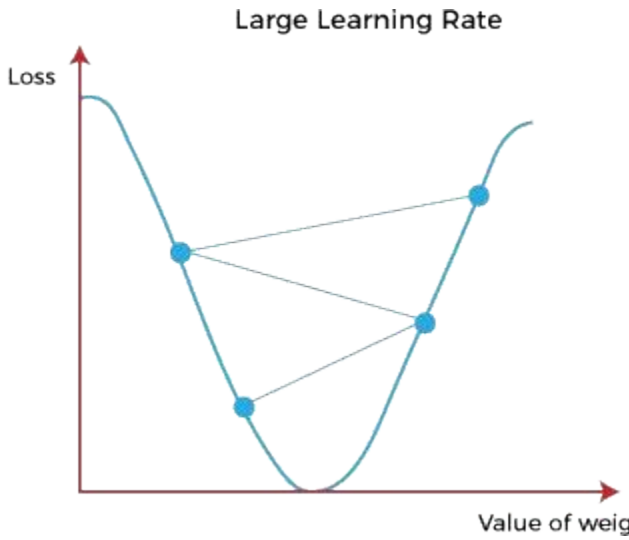
- **Direction & Learning Rate**

These two factors are used to determine the partial derivative calculation of future iteration and allow it to the point of convergence or local minimum or global minimum. Let's discuss learning rate factors in brief;

Learning Rate:

It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function. If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum. At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.





Types of Gradient Descent

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into **Batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.** Let's understand these different types of gradient descent:

1. Batch Gradient Descent:

Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

Advantages of Batch gradient descent:

- It produces less noise in comparison to other gradient descent.
- It produces stable gradient descent convergence.
- It is Computationally efficient as all resources are used for all training samples.

2. Stochastic gradient descent

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed. Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

Advantages of Stochastic gradient descent:

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

- It is easier to allocate in desired memory.
- It is relatively fast to compute than batch gradient descent.
- It is more efficient for large datasets.

3. MiniBatch Gradient Descent:

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent. Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

Advantages of Mini Batch gradient descent:

- It is easier to fit in allocated memory.

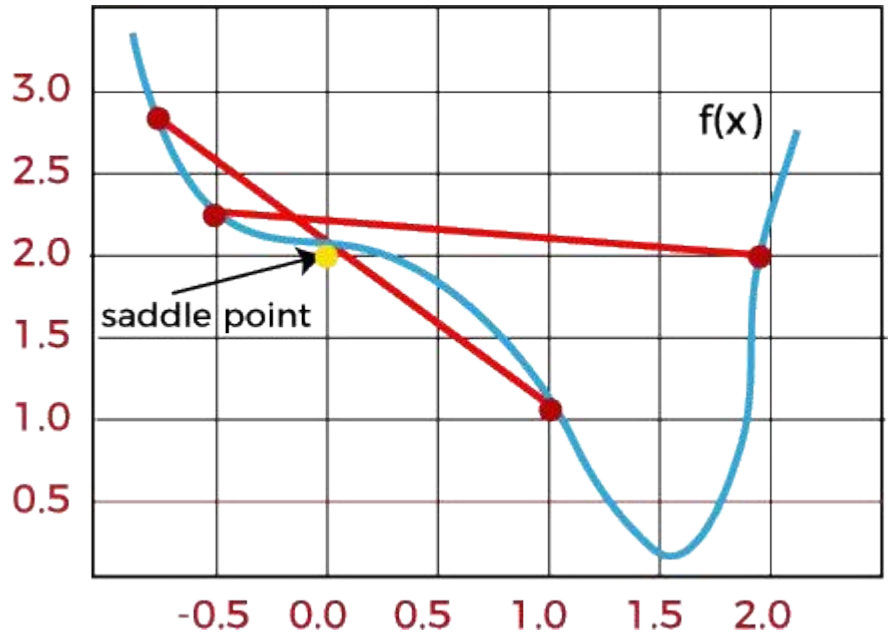
- It is computationally efficient.
- It produces stable gradient descent convergence.

Challenges with the Gradient Descent

Although we know Gradient Descent is one of the most popular methods for optimization problems, it still also has some challenges. There are a few challenges as follows:

1. Local Minima and Saddle Point:

For convex problems, gradient descent can find the global minimum easily, while for non-convex problems, it is sometimes difficult to find the global minimum, where the machine learning models achieve the best results.



Whenever the slope of the cost function is at zero or just close to zero, this model stops learning further. Apart from the global minimum, there occur some scenarios that can show this slop, which is saddle point and local minimum. Local minima generate the shape similar to the global minimum, where the slope of the cost function increases on both sides of the current points.

In contrast, with saddle points, the negative gradient only occurs on one side of the point, which reaches a local maximum on one side and a local minimum on the other side. The name of a saddle point is taken by that of a horse's saddle.

The name of local minima is because the value of the loss function is minimum at that point in a local region. In contrast, the name of the global minima is given so because the value of the loss function is minimum there, globally across the entire domain the loss function.

2. Vanishing and Exploding Gradient

In a deep neural network, if the model is trained with gradient descent and backpropagation, there can occur two more issues other than local minima and saddle point.

Vanishing Gradients:

Vanishing Gradient occurs when the gradient is smaller than expected. During backpropagation, this gradient becomes smaller that causing the decrease in the learning rate of earlier layers than the later layer of the network. Once this happens, the weight parameters update until they become insignificant.

Exploding Gradient:

Exploding gradient is just opposite to the vanishing gradient as it occurs when the Gradient is too large and creates a stable model. Further, in this scenario, model weight increases, and they will be represented as NaN. This problem can be solved using the dimensionality reduction technique, which helps to minimize complexity within the model.

ML Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

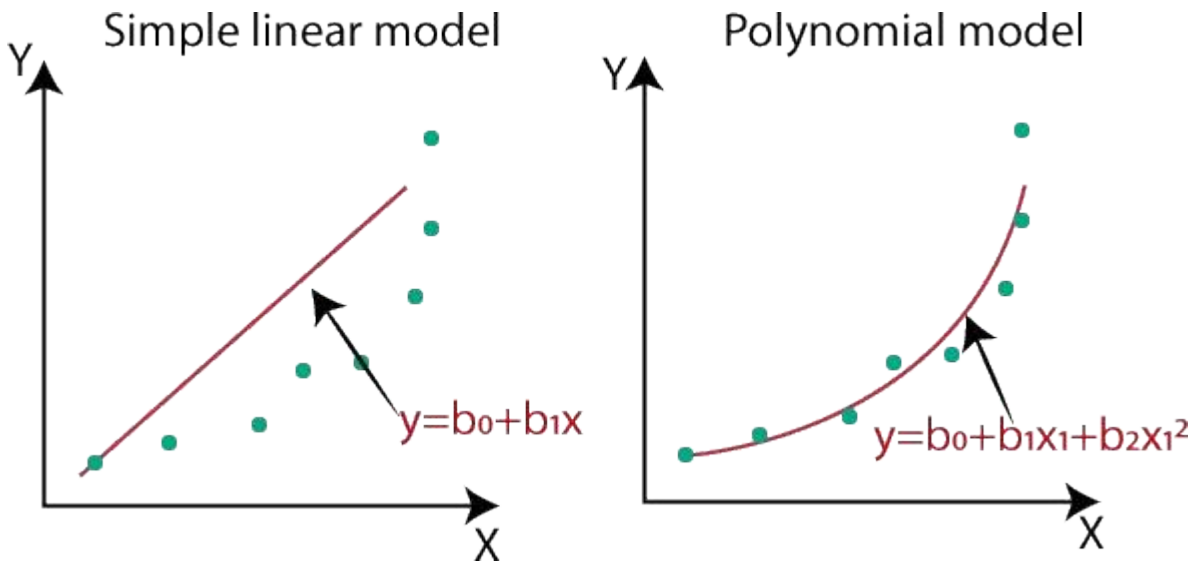
$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, "In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."**

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.

Note: A Polynomial Regression algorithm is also called Polynomial Linear Regression because it does not depend on the variables, instead, it depends on the coefficients, which are arranged in a linear fashion.

Equation of the Polynomial Regression Model:

Simple Linear Regression equation:

$$y = b_0 + b_1 x \quad \text{.....(a)}$$

Multiple Linear Regression equation:

$$y= b_0+b_1x+ b_2x_2+ b_3x_3+....+ b_nx_n \quad .....(b)$$

Polynomial Regression equation:

$$y= b_0+b_1x + b_2x_2+ b_3x_3+....+ b_nx_n \quad .....(c)$$

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Polynomial Regression in Python

To get the Dataset used for the analysis of Polynomial Regression, click here. Import the important libraries and the dataset we are using to perform Polynomial Regression.

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

Pandas – This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
Numpy – Numpy arrays are very fast and can perform large computations in a very short time.
Matplotlib/Seaborn – This library is used to draw visualizations.
Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.

Overfitting Vs Under-fitting

While dealing with the polynomial regression one thing that we face is the problem of overfitting this happens because while we increase the order of the polynomial regression to achieve better and better performance model gets overfit on the data and does not perform on the new data points.

Due to this reason only while using the polynomial regression, do we try to penalize the weights of the model to regularize the effect of the overfitting problem. Regularization techniques like Lasso regression and Ridge regression methodologies are used whenever we deal with a situation in which the model may overfit the data at hand.

Bias Vs Variance Tradeoff

This technique is the generalization of the approach that is used to avoid the problem of overfitting and underfitting. Here as well this technique helps us to avoid the problem of overfitting by helping us select the appropriate value for the degree of the polynomial we are trying to fit our data on. For example, this is achieved when after increasing the degree of polynomial after a certain level the gap between the training and the validation metrics starts increasing.

Advantages of using Polynomial Regression

- A broad range of functions can be fit under it.
- Polynomial basically fits a wide range of curvatures.
- Polynomial provides the best approximation of the relationship between dependent and independent variables.

Disadvantages of using Polynomial Regression

- These are too sensitive to outliers.
- The presence of one or two outliers in the data can seriously affect the results of nonlinear analysis.
- In addition, there are unfortunately fewer model validation tools for the detection of outliers in nonlinear regression than there are for linear regression.

Learning Curves

A learning curve is just a plot showing the progress over the experience of a specific metric related to learning during the training of a machine learning model. They are just a mathematical representation of the learning process.

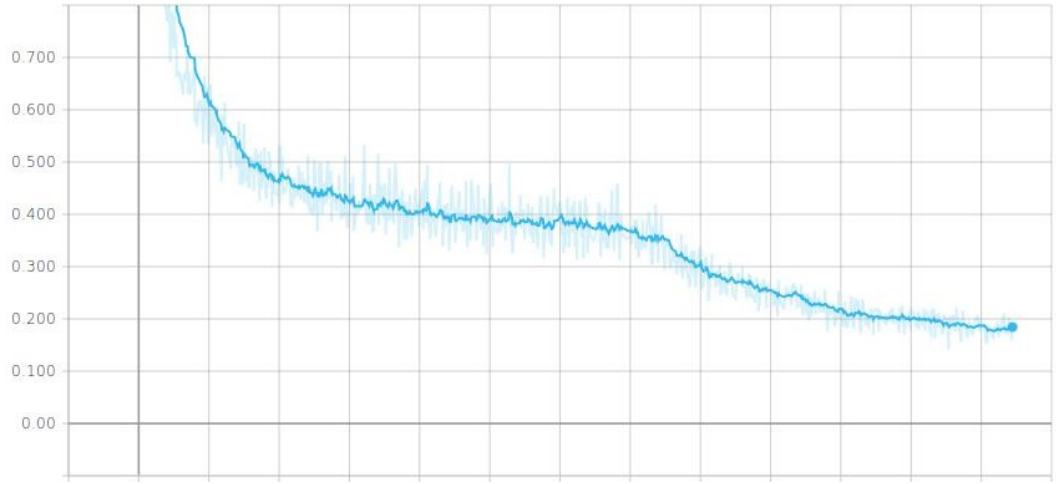
According to this, we'll have a measure of time or progress in the x-axis and a measure of error or performance in the y-axis.

We use these charts to monitor the evolution of our model during learning so we can diagnose problems and optimize the prediction performance.

Single Curves

The most popular example of a learning curve is loss over time. Loss (or cost) measures our model error, or "how bad our model is doing". So, for now, the lower our loss becomes, the better our model performance will be.

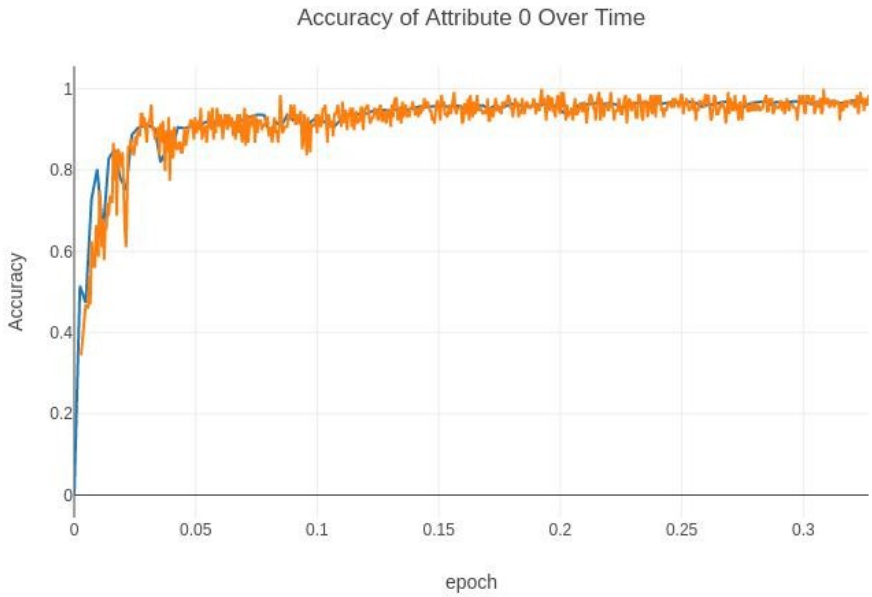
In the picture below, we can see the expected behavior of the learning process:



Despite the fact it has slight ups and downs, in the long term, the loss decreases over time, so the model is learning.

Other examples of very popular learning curves are accuracy, precision, and recall. All of these capture model performance, so the higher they are, the better our model becomes.

See below an example of a typical accuracy curve over time:



The model performance is growing over time, which means the model is improving with experience (it's learning).

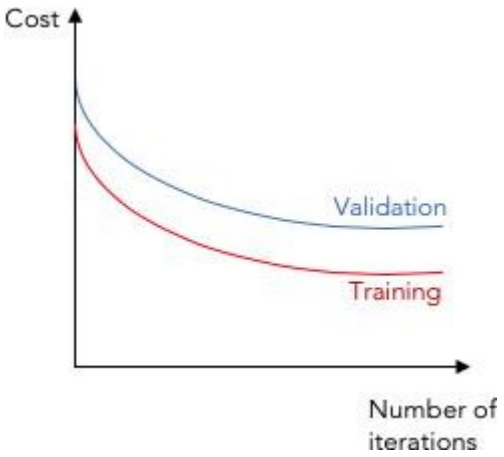
We also see it grows at the beginning, but over time it reaches a plateau, meaning it's not able to learn anymore.

2.3. Multiple Curves

One of the most widely used metrics combinations is training loss + validation loss over time.

The training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

We will see this combination later on, but for now, see below a typical plot showing both metrics:



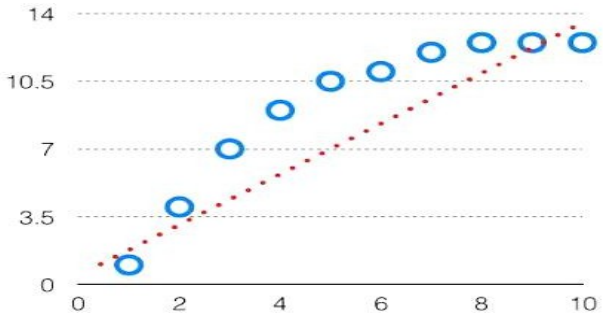
Another common practice is to have multiple metrics in the same chart as well as those metrics for different models.

Bias-Variance Trade Off – Machine Learning

It is important to understand prediction errors (bias and variance) when it comes to accuracy in any machine-learning algorithm. There is a tradeoff between a model's ability to minimize bias and variance which is referred to as the best solution for selecting a value of Regularization constant. A proper understanding of these errors would help to avoid the overfitting and underfitting of a data set while training the algorithm**.**

What is Bias?

The bias is known as **the difference between the prediction of the values by the Machine Learning model and the correct value.** Being high in biasing gives a large error in training as well as testing data. It recommended that an algorithm should always be low-biased to avoid the problem of underfitting. By high bias, the data predicted is in a straight line format, thus not fitting accurately in the data in the data set. Such fitting is known as the Underfitting of Data. This happens when the hypothesis is too simple or linear in nature. Refer to the graph given below for an example of such a situation.



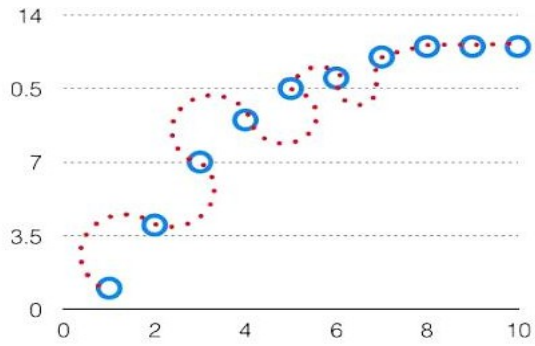
High Bias in the Model

In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

What is Variance?

The variability of model prediction for a given data point which tells us **the spread of our data is called the variance of the model.** The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data. As a result, such models perform very well on training data but have high error rates on test data. When a model is high on variance, it is then said to as Overfitting of Data. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high. While training a data model variance should be kept low. The high variance data looks as follows.



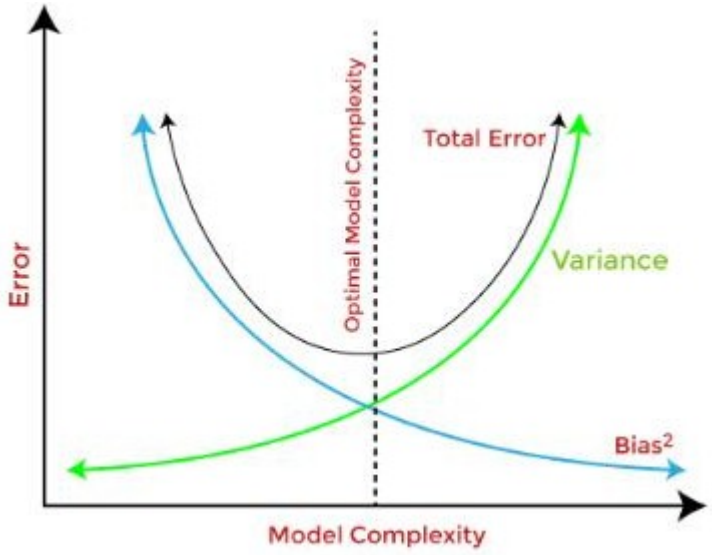
High Variance in the Model

In such a problem, a hypothesis looks like follows.

$$h_\theta(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4)$$

Bias-Variance Trade-Off

While building the machine learning model, it is really important to take care of bias and variance in order to avoid **overfitting** and **underfitting** in the model. *If the model is very simple with fewer parameters, it may have low variance and high bias.* Whereas, *if the model has a large number of parameters, it will have high variance and low bias.* So, it is required to make a balance between bias and variance errors, **and this balance between the bias error and variance error is known as the Bias-Variance trade-off.**

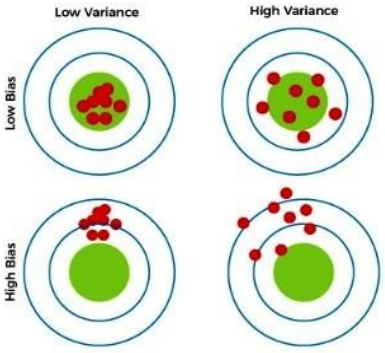


For an accurate prediction of the model, algorithms need a low variance and low bias. But this is not possible because bias and variance are related to each other:

- If we decrease the variance, it will increase the bias.
- If we decrease the bias, it will increase the variance.

Bias-Variance trade-off is a central issue in supervised learning. Ideally, we need a model that accurately captures the regularities in training data and simultaneously generalizes well with the unseen dataset. Unfortunately, doing this is not possible simultaneously. Because a high variance algorithm may perform well with training data, but it may lead to overfitting to noisy data. Whereas, high bias algorithm generates a much simple model that may not even capture important regularities in the data. So, we need to find a sweet spot between bias and variance to make an optimal model.

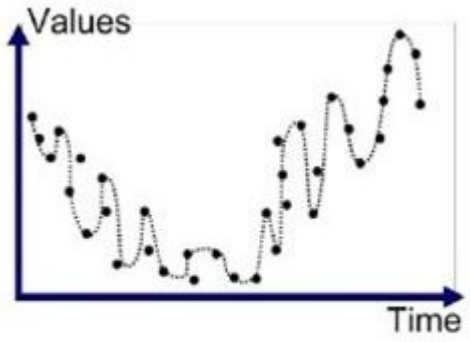
Hence, the ***Bias-Variance trade-off is about finding the sweet spot to make a balance between bias and variance errors***



1. **Low-Bias, Low-Variance:**
   The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.
2. **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting**
3. **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.
4. **High-Bias, High-Variance:**
   With high bias and high variance, predictions are inconsistent and also inaccurate on average.

Regularized Linear Models

In machine learning, we often face the problem when our model behaves well on training data but behaves very poorly on test data. This happens when the model closely follows the training data i.e overfits the data.
Regularization is a technique to reduce overfitting. The term regularization means the act of bringing to uniformity.



Overfitted

Complex models can detect a subtle pattern in the data, but if the data is noisy(contains irrelevant information) or the dataset is too small, the model will end up detecting the pattern in the noise itself. When we use this model to predict our results, the result will not be accurate and the error will be more than the expected error.
In linear regression, the final output is the weighted sum of the feature variables which is represented by the below equation.

$$y = w1x1+w2x2+w3x3+...+wn\,xn+w_0$$

Here weights w1, w2, …, wn represent the importance of the features(x1, x2,..xn). A feature will be of high importance if it has a large weight associated with it.

The error in linear regression will be the mean squared error which is given below:

$$\text{MSE} = 1/n \sum_{i=0}^{n} (predicted - actual)^2$$

To improve the model or reduce the effect of the noise in our model, we need to reduce the weights associated with noise. Smaller the weight associated with the noise will be the less contribution it will have in predicting the output.

For a linear model, regularization is achieved by constraining the weights of the model. To constrain the weights first we need to understand how these weights are calculated. Weights are calculated as per the cost function, for linear regression cost function is mean squared error. Weights are tweaked each time and MSE is calculated and the set that has minimum MSE will be considered as the final output.

To regularize the model, the regularization term will be added to the cost function.

Regularized Cost Function = MSE+ Regularization term

Here we will see three different regularization term to constrain the weights of the model, thus three different regularized linear regression algorithms:

1. Ridge Regression
2. Lasso Regression
3. Elastic Net

Ridge Regression:

Cost Function: MSE + $\alpha \cdot 1/2 \sum_{i=0}^{n}(wi)^2$

In ridge regression, the regularization term **is the sum of the square of the weights of the model.** In statistics, this regularization term is called L2 norm. It forces the model to keep the weight as small as possible.

In the above equation, alpha is a hyperparameter, it controls how much we want to regularize our regression model. If we choose a very large alpha then the learning algorithm will try to keep weights as small as possible because large weights will increase the cost function, thus the result will be a flat line passing through the mean of the data. If alpha is 0 then ridge regression is nothing but linear regression. To choose the best hyperparameter value, we do **hyperparameter tuning**.

Lasso Regression:

Cost Function: MSE + $\alpha \sum_{i=0}^{n}| wi|^2$

Least absolute shrinkage and selection operator regression(Lasso) uses L1 norm for regularization . i.e **absolute sum of the weights of the model**.

An important characteristic of the lasso regression is, it tends to eliminate the features which have less importance by shrinking the weights to zero, and because of this it is used in feature selection also.

Elastic Net:

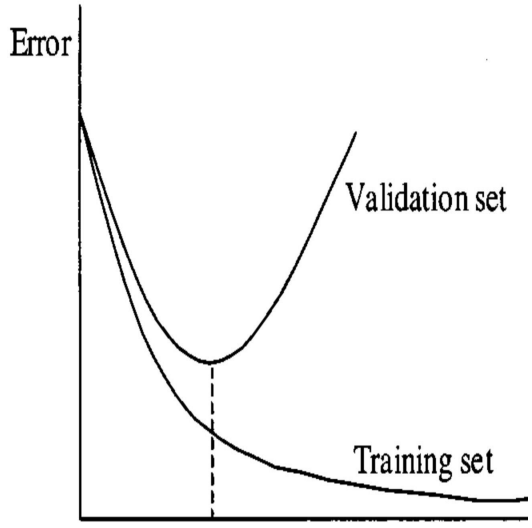
Cost Function: MSE + $\alpha \cdot (1 - r)/2 \sum_{i=0}^{n}(wi)^2 + \alpha \cdot \mathrm{r} \left|\sum_{i=0}^{n}\right| wi|^2$

Elastic net is a mix of ridge and lasso regression, how much you want to mix depends on the value of ‘r’. For example, if r is set to zero then it will be equal to lasso and if it is one then it will become ridge regression.

The important question if how we will decide which regression we should follow. The answer is, we should always prefer to have some regularization, so we use ridge regression by default but when you think some features are important than others use lasso regression or elastic net but when the data set has large number of features prefer elastic net. Elastic net behaves much better than lasso when the dataset has large number of features.

Early stopping

In machine learning, early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration.



Early stopping approaches

There are three main ways early stopping can be achieved. Let’s look at each of them:

1. Training model on a preset number of epochs: By running a set number of epochs, we run the risk of not reaching a satisfactory training point. With a higher learning rate, the model might possibly converge with fewer epochs, but this method requires a lot of trial and error.

2. Stop when the loss function update becomes small: The training is stopped when the update becomes as small as 0.001, as stopping at this point minimizes loss and saves computing power by preventing any unnecessary epochs. However, overfitting might still occur.

3. Validation set strategy: The training error decreases exponentially until increasing epochs no longer have such a large effect on the error. The validation error, however, initially decreases with increasing epochs, but after a certain point, it starts increasing. This is the point where a model should be early stopped as beyond this the model will start to overfit.