Machine Learning

Model Optimization

Satishkumar L. Varma

Department of Information Technology
SVKM's Dwarkadas J. Sanghvi College of Engineering, Vile Parle, Mumbai.
ORCID | Scopus | Google Scholar | Google Site | Website

Optimization: Outline

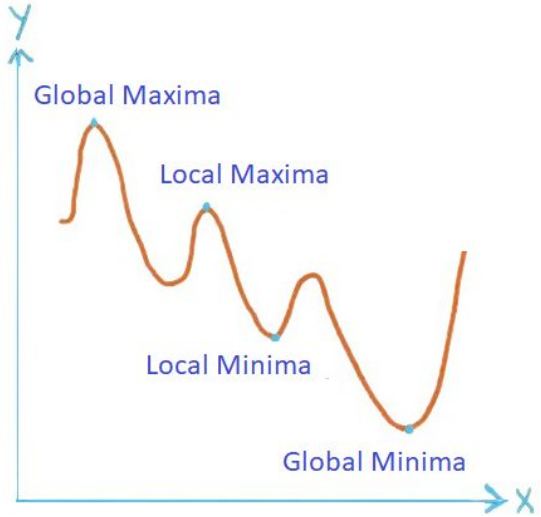
- Optimizers
  - Gradient Descent (GD),
  - Types of GD, Vanishing Gradient Problem and Exploding Gradient Problem,
  - Frobenius norm regularization,
  - Early stopping,
  - Adam optimizer.

Optimization: Introduction

- Optimization
  - Process training the model iteratively to get a maximum and minimum function evaluation
  - It helps AI/ML/DL techniques/models to get to get better results.
- Why do we optimize our any learning models?
  - Result is compared in every iteration by changing hyperparameters in each step until we reach optimum results.
  - We create an accurate model with less error rate.
  - There are different ways using which we can optimize a model.
  - Example of Optimization algorithms:
    - Batch Gradient Descent
    - Stochastic Gradient Descent(SGD)
    - Mini-batch Gradient Descent
    - Adam Optimization

Optimization: Optimization

- **Fundamental Terms used in** Optimization
- Maxima: Largest value of a function within a given range.
- Minima:Smallest value of a function within a given range.
- Global Maxima: Maximum value on the entire domain of the function
- Global Minima: Minimum value on the entire domain of the function
- Local Maxima: Maximum value of the function within a given range.
- Local Minima: Minimum value of the function within a given range.
- There can be only one global minima and maxima
- There can be more than one local minima and maxima.



Optimization: Optimization

- **Fundamental Terms for Optimization**
- **Epoch**: The number of times the algorithm runs on the whole training dataset.
- **Sample**: A single row of a dataset.
- **Batch**: It denotes the number of samples to be taken to for updating the model parameters.
- **Learning rate**: It is a parameter that provides the model a scale of how much model weights should be updated.
- **Cost Function/Loss Function**: A cost function is used to calculate the cost that is the difference between the predicted value and the actual value.
- **Weights/ Bias**: The learnable parameters in a model that controls the signal between two neurons.

Optimization: List of Optimizers

- **Example:**
  - RMSProp
  - SGD
  - Adagrad
  - Adadelta
  - Adam
  - Conjugate Gradients
  - BFGS
  - Momentum
  - Nesterov Momentum
  - Newton's Method

Optimization: Gradient Descent

$\theta_i = \theta_i - \alpha * dJ / d\theta_i$

Parameter | Learning Rate | Gradient / Slop

- **What** is Gradient Descent?
  - It is a powerful optimization algorithm used to minimize the loss function(LF) in a DL/ML.
  - Several variants of the algorithm designed to address different challenges and provide optimizations
  - The goal of GD is to find the set of weights (or coefficients) that minimize the loss function
  - It works by iteratively adjusting the weights in the direction of the steepest decrease in the loss function.
- **How** does Gradient Descent Work?
  - Idea is to start with an initial set of weights and update them in the direction of -ve gradient of the LF
  - Gradient is a vector of partial derivatives that represents the rate of change of the LF wrt the weights
  - By updating weights in the direction of the -ve gradient, the algorithm moves towards a minimum of the LF
  - The learning rate (r) is a hyperparameter that determines the size of the step taken in the weight update
  - Small r results in a slow convergence, while large r lead to overshooting the min. & oscillating around min.
  - Those an appropriate α that balances the speed of convergence and the stability of optimization

Optimization: Gradient Descent

- Equation of the Gradient Descent Algorithm

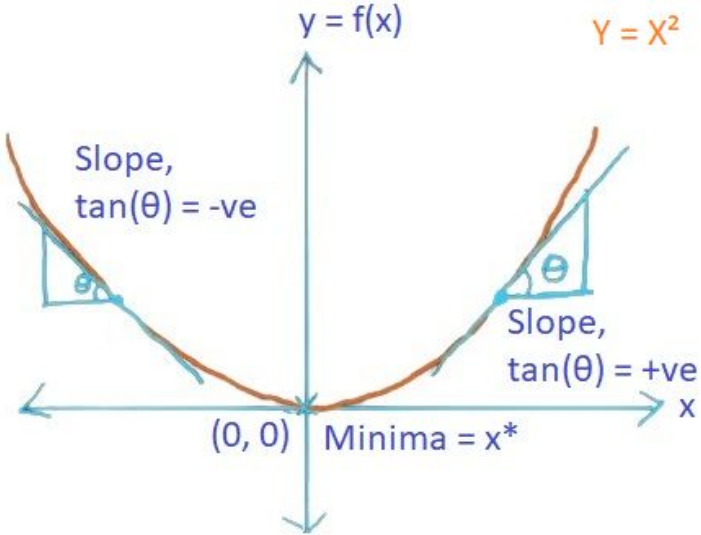
$$\theta_i = \theta_i - \alpha * dJ / d\theta_i$$

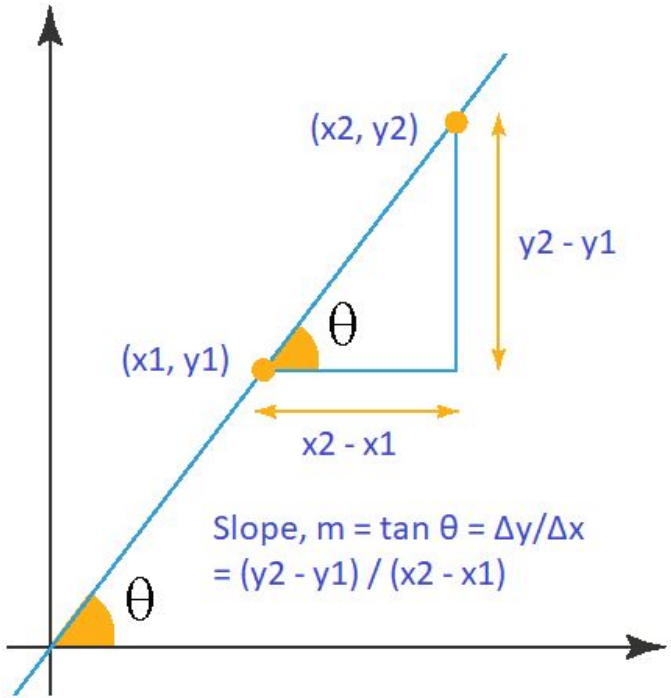
Parameter | Learning Rate | Gradient / Slop

- θ - parameter we wish to update
- dJ/dθ - the partial derivative which tells us the rate of change of error on the cost function wrt the parameter θ
- α - Learning Rate.
- J - represents the cost function and there are multiple ways to calculate this cost
- Based on the way we are calculating this cost function there are **different variants of Gradient Descent**
    - Batch Gradient Descent
    - Stochastic Gradient Descent (SGD)
    - Mini-batch Gradient Descent

Optimization: Optimization Algorithms: Gradient Descent

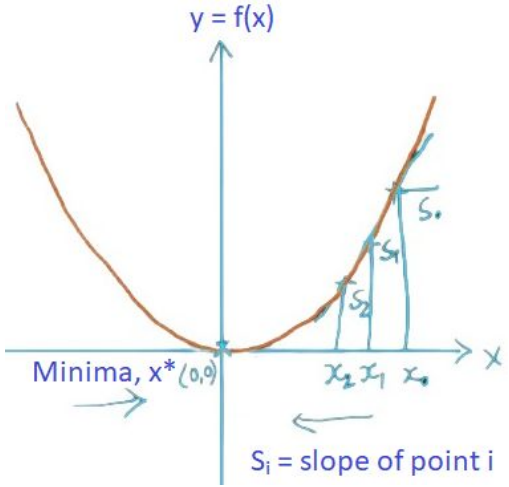
- Optimization algorithm: **Gradient Descent**
- This optimization algorithm uses calculus to modify the values consistently and to achieve the local minimum.
- It finds out the local minima of a differentiable function
- It is a minimization algorithm that minimizes a given function.
- Example: The geometric intuition of Gradient Descent





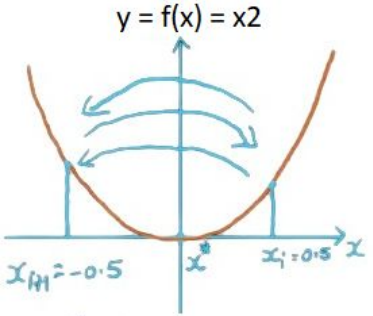
Optimization: Optimization Algorithms: Gradient Descent

- Slope (Tanθ) of points as moved towards minima, x* which is at origin(0, 0)
- Slope on right side is +ve as $0 < \theta < 90$ and its Tanθ is a +ve value
- Slope on left side is -ve as $90 < \theta < 180$ and its Tanθ is a -ve value
- Observation: slope changes its sign from positive to negative at minima
- As we move closer to the minima, the slope reduces.
- Here, df/dx = gradient
- **Gradient Descent Algorithm:** Objective: Calculate X*- local minimum of the function $Y=X^2$
  - 1. Pick an initial point X_0 at random
  - 2. Calculate $X_1 = X_0 - r[df/dx]$ at X_0 and **r** is Learning Rate
  - 3. Let **r** = 1; and here, df/dx = gradient
  - 4. Calculate $X_2 = X_1 - r[df/dx]$ at X_1
  - 5. Calculate for all the points: $X_1, X_2, X_3, \ldots\ldots, X_{i-1}, X_i$
  - 6. Calculate local minima: $X_i = (X_{i-1}) - r[df/dx]$ at X_{i-1}
  - 7. Stop the iteration and declare $X^* = X_i$
    - When $(X_i - X_{i-1})$ is small, i.e., when X_{i-1}, X_i converge



Optimization: Example: Gradient Descent Algorithm

- How does Gradient Descent Algorithm work?
- **Example:** Graph of a parabola, $Y = X^2$
- **Objective**: Calculate local minimum (X^*) of the function $Y=X^2$
  - Pick an initial point $X_0 = 0.5$ at random
  - Let Learning Rate, **r** = 1
  - Here, df/dx = gradient
  - Calculate derivative of $Y = f(X^2) = [df/dx] = 2x$
  - Calculate local minima: $X_i = (X_{i-1}) - \mathbf{r}[df/dx]$ at X_{i-1}
  - Calculate $X_1 = X_0 - \mathbf{r}[df/dx]$ at X_0
  - $X_1 = 0.5 - 1[2*(0.5)] = -0.5$
  - Calculate $X_2 = X_1 - \mathbf{r}[df/dx]$ at X_1
  - $X_2 = -0.5 - 1[2*(-0.5)] = 0.5$
  - When $(X_i - X_{i-1})$ is small, i.e., when X_{i-1}, X_i converge,
  - we stop the iteration and declare $X^* = X_{ii}$



Learning rate, r = 1

$\frac{df}{dx} = 2x$

$x_{i+1} = x_i - \gamma\left[\frac{df}{dx}\right]_{x_i}$

$x_{i+1} = 0.5 - 1(2 \times 0.5]$

$x_{i+1} = -0.5$

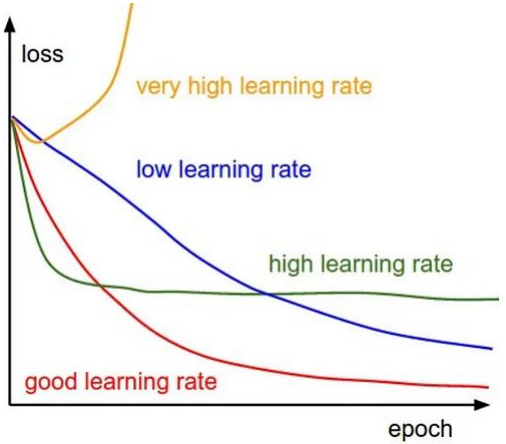
$x_{i+2} = x_{i+1} - \gamma\left[\frac{df}{dx}\right]_{x_{i+1}}$

$x_{i+2} = -0.5 - 1[2(-0.5)]$

$= -0.5 + 1 = 0.5$

Optimization: Learning Rate

- Learning Rate (**r**) is a hyperparameter or tuning parameter
  - **r** determines step size at each iteration while moving towards minima in function.
  - For example, if **r** = 0.1 in the initial step, it can be taken as **r** = 0.01 in next step.
  - Likewise it can be reduced exponentially as we iterate further.
- **Oscillation Problem**: if we keep **r** value as constant
- In the above Gradient Descent Algorithm **example**, we took **r** = 1.
  - Calculate the points X_i, X_{i+1}, X_{i+2},....to find the local minima, X*,
  - we observe oscillation between X = -0.5 and X = 0.5
  - When we keep **r** as constant, we end up with an oscillation problem.
  - So, we have to reduce the "**r**" value with each iteration.
  - Reduce the **r** value as the iteration step increases.



- **Note**: The most commonly used Learning Rate (**r**) are: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3
  - Hyperparameters decide the bias-variance tradeoff.
  - When r value is low, it could overfit the model and cause high variance.
  - When r value is high, it could underfit the model and cause high bias.
  - We can find the correct **r** value with Cross Validation technique.
  - Plot a graph with diff. **r** and check for the training loss with each value and choose one with minimum loss.

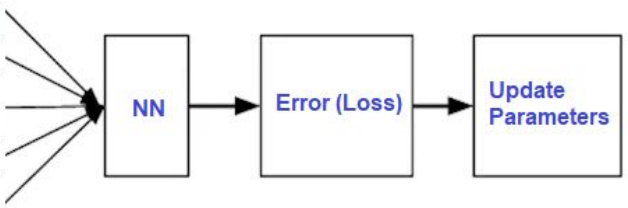
Optimization: Batch Gradient Descent

- Sum up over all examples on each iteration when performing updates to parameter $\theta_i = \theta_i - \alpha * dJ / d\theta_i$ (Parameter; Learning Rate; Gradient / Slop)
- **Batch Gradient Descent Algorithm:**
  - For a total of 'm' observations in a dataset, we use all these observations to calculate the cost function J
  - Pass the entire training set (**one epoch**) to model,
  - Perform forward propagation and calculate the **cost function**
  - And then update the parameters using the rate of change of this loss function (LF) wrt the parameters
  - After **one epoch**, f/w propagation & b/w propagation are performed
  - And the parameters are updated once **per epoch**
- **Advantages**:
  - Use fixed learning rate during training without worrying about learning rate decay
  - It has straight trajectory towards the minimum and it is guaranteed to converge in theory
    - to the global minimum if the LF is convex and to a local minimum if the LF is not convex
  - It has unbiased estimate of gradients
  - The more the examples, the lower the standard error.
- **Disadvantages**:
  - We can use vectorized implementation still slow to go over all examples in case of large datasets
  - Each step of learning happens after going over all examples where some examples may be redundant.

Optimization: Stochastic Gradient Descent(SGD)

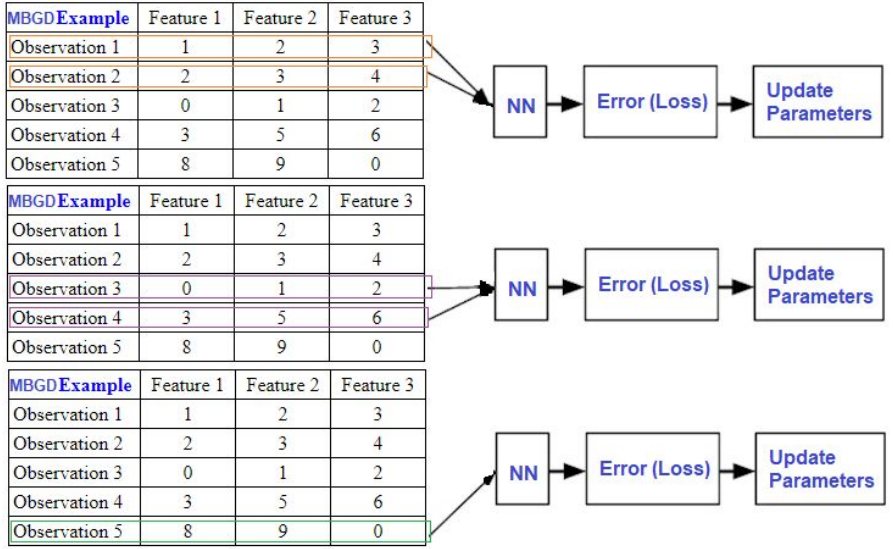
- SGD Algorithm: We pass a single observation at a time, calculate the cost and update the parameters.
  - Step 1: Consider 5 observations and each observation has three features randomly.
  - Step 2: Take 1st observation, then pass it through the NN, calculate error (LF) & then update parameters
  - Step 3: Take 2nd observation, then pass it through the NN, calculate error (LF) & then update parameters
  - Step 4: Repeat until all observations have been passed through NN and the parameters have been updated
- SGD Example:
- Each time the parameter is updated, it is known as an Iteration
- So, 5 observations, the parameters will be updated 5 times so there will be 5 iterations
- SGD: There will be 'm' iterations per epoch, where 'm' is the number of observations in a dataset
- BGD: All the observations are passed together and the parameters have been updated only once
- Note:
  - If we use the entire dataset to calculate the cost function, it is known as Batch Gradient Descent (BGD)
  - If use a single observation to calculate the cost, it is known as Stochastic Gradient Descent(SGD)

SGD Example	Feature 1	Feature 2	Feature 3
Observation 1	1	2	3
Observation 2	2	3	4
Observation 3	0	1	2
Observation 4	3	5	6
Observation 5	8	9	0



Optimization: Mini-batch Gradient Descent

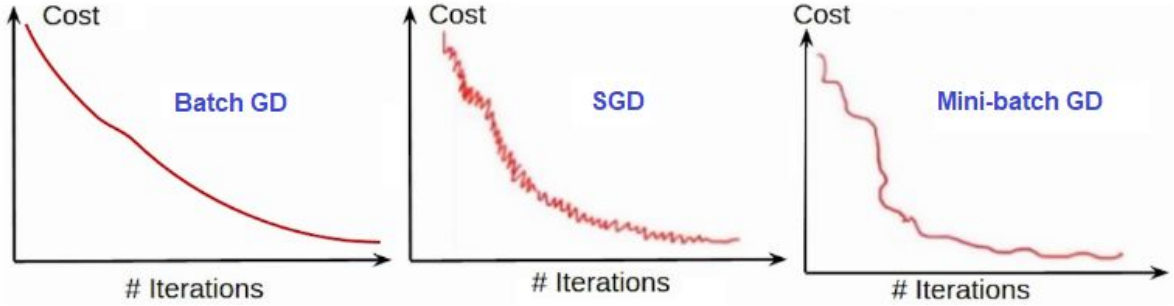
- It takes a subset of the entire dataset to calculate the cost function
- So if there are 'm' observations then each subset or mini-batches (x) => 1 < x < m
- **MBGD Algorithm**: Assume that the batch size is 2
  - Step 1: Take first two observations, pass them through NN, calculate the error and then update parameters
  - Step 2: Take next two observations, pass them through NN, calculate the error and then update parameters
  - Step 3: Take remaining single last observation in final iteration, calculate error and then update parameters
- **Advantages**:
  - Faster than Batch
  - Random selection will help avoid redundant examples
  - It adds noise with batch size < size of training set
- **Disadvantages**:
  - It won't converge
  - Due to noise, it goes back and forth on each iteration
  - So, it wanders around the min. but never converges



Optimization: Compare GD, SGD, and Mini-batch

- Comparison between Batch GD, SGD, and Mini-batch GD

Batch GD	SGD	Mini-batch GD
Take entire dataset for updation	Take single observation for updation	Take a subset of dataset for updation
Cost function reduces smoothly	Updation is not that smooth	Smoother cost function as compared to SGD
Computation cost is very high	Computation time is more	Computation time < SGD & Computation cost < BGD

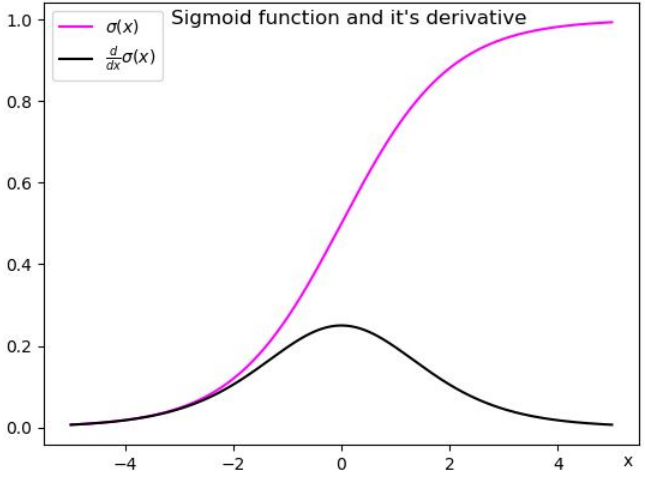


Optimization: Exploding and Vanishing Gradients

- **What is the problem of exploding and vanishing gradients while training a NN?**
- Vanishing Gradient Problem
  - As backpropagation algo advances downwards (or backward) from o/p to i/p layer,
    - Gradients often get smaller & smaller & approach zero which eventually leaves weights nearly unchanged
    - As a result, GD never converges to the optimum and this is known as the vanishing gradients problem.
- Exploding Gradient Problem
  - On the contrary, as the backpropagation algorithm progresses, in some cases,
    - Gradients keep on getting larger and larger, in tern, causes very large weight updates and
    - As a result, causes the GD to diverge and this is known as the exploding gradients problem.

Optimization: Exploding and Vanishing Gradients

- **Why do the gradients even vanish/explode?**
- Certain activation functions (say sigmoid), have a very huge difference between the variance of their i/p and o/p
- That is, they shrink and transform a larger input space into a smaller output space that lies between [0,1]
- In Sigmoid function, for larger inputs (-ve or +ve), it saturates at 0 or 1 with a derivative very close to zero
  - So, during backpropagation, it virtually has no gradients to propagate backward in n/w, and
  - whatever little residual gradients exist keeps on diluting as algo. progresses down through the top layers
  - So, this leaves nothing for the lower layers.
- Similarly, in some cases suppose the initial weights assigned to the network generate some large loss



Optimization: Exploding and Vanishing Gradients

- **How to know if our model is suffering from the Exploding/Vanishing gradient problem?**
- Signs that can indicate that our gradients are exploding/vanishing :
- Neither do we want our signal to explode or saturate nor do we want it to die out.
- The signal needs to flow properly both in the forward direction
  - when making predictions as well as in the backward direction while calculating gradients.

Exploding	Vanishing
There is an exponential growth in the model parameters.	The parameters of the higher layers change significantly whereas the parameters of lower layers would not change much (or not at all).
The model weights may become NaN during training.	The model weights may become 0 during training.
The model experiences avalanche learning.	The model learns very slowly and perhaps the training stagnates at a very early stage just after a few iterations.

Optimization: Exploding and Vanishing Gradients

- **Techniques that can be used to fix Vanishing/Exploding Gradients Problems in NN**
- 1. Proper Weight Initialization
- 2. Using Non-saturating Activation Functions:
  - Saturation of activation functions like sigmoid and tanh, so, use non-saturating functions like ReLu.
  - ReLU can significantly reduce the chances of vanishing/exploding problems at the beginning.
  - However, it does not guarantee that the problem won't reappear during training.
- 3. Batch Normalization
- 4. Gradient Clipping
  - To clip the gradients during backpropagation so that they never exceed some threshold

Summary

- Gradient Descent and Stochastic Gradient Descent are useful in in Logistic Regression.
- Mini-batch GD is the most commonly used variant of the Gradient Descent.
- Also used in Logistic Regression and Linear Regression.
- Hyperparameter tuning, particularly for the initial learning rate and epsilon (ε), is crucial for optimizing Adam's performance.

References

Text books:

1. Ethem Alpaydin, “Introduction to Machine Learning , 4th Edition, The MIT Press, 2020.
2. Peter Harrington, “Machine Learning in Action”, 1st Edition, Dreamtech Press, 2012."
3. Tom Mitchell, “Machine Learning”, 1st Edition, McGraw Hill, 2017.
4. Andreas C, Müller and Sarah Guido, “Introduction to Machine Learning with Python: A Guide for Data Scientists”, 1ed, O'reilly, 2016.
5. Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, 1st Edition, MIT Press, 2012."

Reference Books:

6. Aurélien Géron, “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow”, 2nd Edition, Shroff/O'Reilly, 2019.
7. Witten Ian H., Eibe Frank, Mark A. Hall, and Christopher J. Pal., “Data Mining: Practical machine learning tools and techniques”, 1st Edition, Morgan Kaufmann, 2016.
8. Han, Kamber, “Data Mining Concepts and Techniques”, 3rd Edition, Morgan Kaufmann, 2012.
9. Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar, “Foundations of Machine Learning”, 1ed, MIT Press, 2012.
10. H. Dunham, “Data Mining: Introductory and Advanced Topics”, 1st Edition, Pearson Education, 2006.

Thank You.