UNIT II

The Perceptron: Bio-inspired Learning, The Perceptron Algorithm, Geometric Interpretation, Interpreting Perceptron Weights, Perceptron Convergence and Linear Separability, Improved Generalization, Limitations of the Perceptron [Reference 1] Practical Issues: Importance of Good Features, Irrelevant and Redundant Features, Feature Pruning and Normalization, Combinatorial Feature Explosion, Evaluating Model Performance, Cross Validation, Hypothesis Testing and Statistical Significance, Debugging Learning Algorithms, Bias Variance tradeoff [Reference 1] Linear Models: The Optimization Framework for Linear Models, Convex Surrogate Loss Functions, Weight Regularization, Optimization and Gradient Descent, Support Vector Machines [Reference 1].

2.1 THE PERCEPTRON

Q1. What is the Perceptron model in Machine Learning?

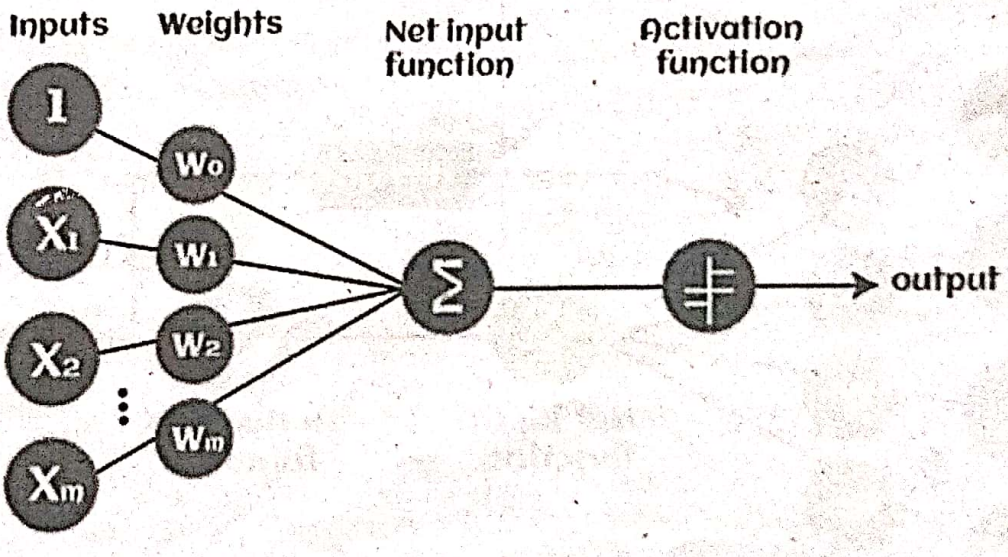
Ans:

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

Basic Components of Perceptron:

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



➢ **Input Nodes or Input Layer**

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias**

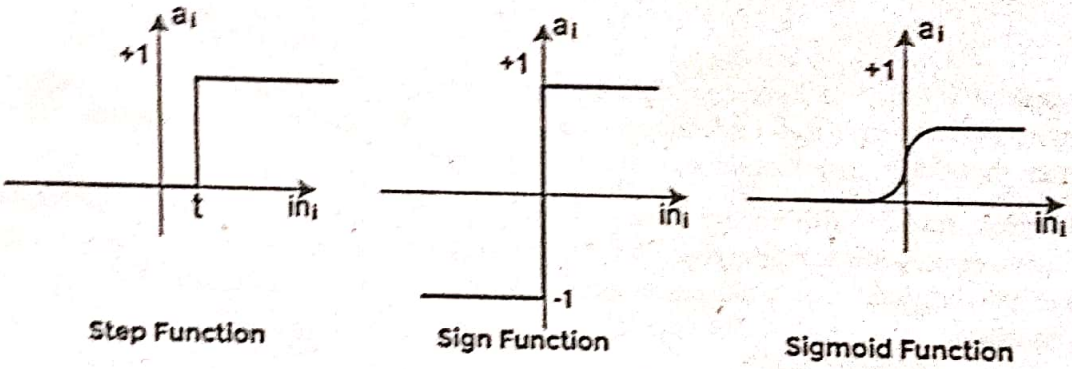
  Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function**

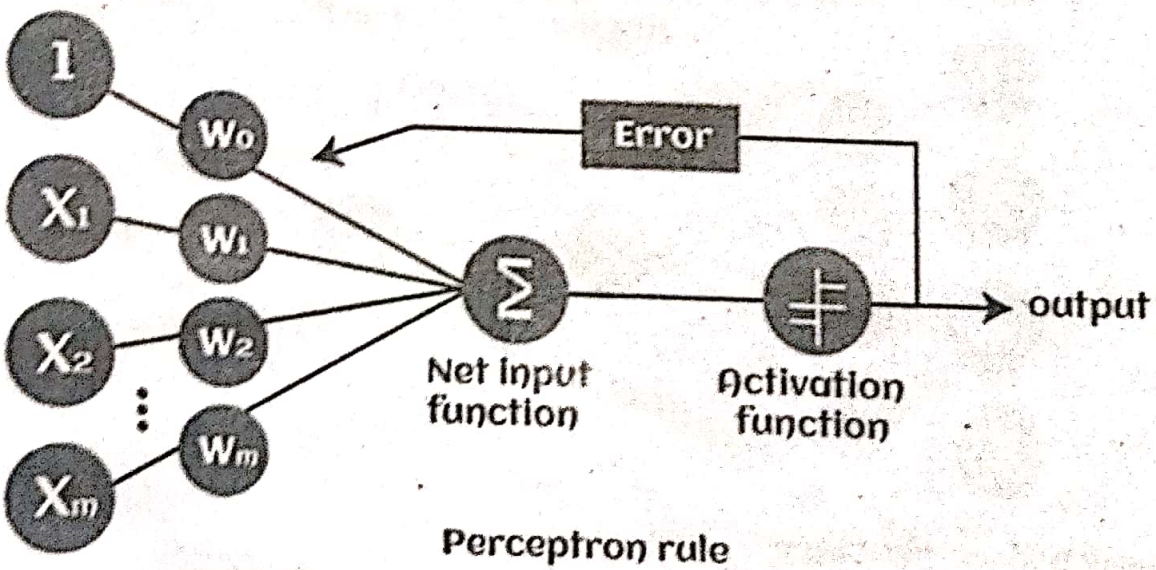
  These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function



The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.



This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input.

The working of Perceptron model

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f'.

This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$\Sigma xi = x1*w1 + x2*w2 + \ldots wn*xn$

Add a special term called bias 'b' to this weighted sum to improve the model's performance.

$\Sigma wi*xi + b$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

Q2. Explain in detail about types of Perceptron Models?

Ans :

Types of Perceptron Models

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

1. Single Layer Perceptron Model

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

If the outcome is same as pre-determined or threshold value, then the performance of this model is stated as satisfied, and weight demand does not change. However, this model consists of a few discrepancies triggered when multiple weight inputs values are fed into the model. Hence, to find desired output and minimize errors, some changes should be necessary for the weights input.

"Single-layer perceptron can learn only linearly separable patterns."

2. Multi-Layered Perceptron Model

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage**

  Activation functions start from the input layer in the forward stage and terminate on the output layer.

- **Backward Stage**

  In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

Hence, a multi-layered perceptron model has considered as multiple artificial neural networks having various layers in which activation function does not remain linear, similar to a single layer perceptron model.

Instead of linear, activation function can be executed as sigmoid, TanH, ReLU, etc., for deployment.

A multi-layer perceptron model has greater processing power and can process linear and non-linear patterns. Further, it can also implement logic gates such as AND, OR, XOR, NAND, NOT, XNOR, NOR.

Advantages

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

Perceptron Function

Perceptron function "f(x)" can be achieved as output by multiplying the input 'x' with the learned weight coefficient 'w'.

Mathematically, we can express it as follows:

$f(x)=1$; if $w.x+b>0$

otherwise, $f(x)=0$

- 'w' represents real-valued weights vector
- 'b' represents the bias
- 'x' represents a vector of input x values.

Characteristics of Perceptron

The perceptron model has the following characteristics.

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.
6. If the added sum of all input values is more than the threshold value, it must have an output signal; otherwise, no output will be shown.

2.2 Bio-Inspired Learning

Q3. What is Bio-inspired Learning?

Ans:

Bio-inspired computing is a research method aimed at solving problems using computer models based on the principles of biology and the natural world. Commonly seen as a philosophical approach, bio-inspired computing is used in a number of related fields of study within computing, rather than a field of study itself. Bio-inspired computing is an extension of the related field of biomimicry.

Bio-inspired computing puts less focus on optimized, high-speed algorithms and more focus on tractability and dependability. Generally, the approach is ground-up, rather than taking a large foundation of knowledge and adding artificial intelligence to it. Bio-inspired computing often takes a small foundation of set rules and builds upon them by way of unsupervised deep learning in training.

In cases where a problem has no clear solution, a useful change of perspective can make for a new chance at solutions. A useful question to ask might be "does this problem have any parallels in nature?" If the answer is "yes" then that parallel can be studied to similarly find parallels in solutions. Humanity has found this philosophy to be useful in solving numerous problems.

2.3 The Perceptron Algorithm

Q4. Explain in detail about perceptron algorithm?

Ans:

The Perceptron is a linear machine learning algorithm for binary classification tasks.

It may be considered one of the first and one of the simplest types of artificial neural networks. It is definitely not "deep" learning but is an important building block.

Like logistic regression, it can quickly learn a linear separation in feature space for two-class classification tasks, although unlike logistic regression, it learns using the stochastic gradient descent optimization algorithm and does not predict calibrated probabilities.

The Perceptron Classifier is a linear algorithm that can be applied to binary classification tasks.

How to fit, evaluate, and make predictions with the Perceptron model with Scikit-Learn.

How to tune the hyperparameters of the Perceptron algorithm on a given dataset.

Perceptron Algorithm

The Perceptron algorithm is a two-class (binary) classification machine learning algorithm.

It is a type of neural network model, perhaps the simplest type of neural network model.

It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating

Because of this, the learning algorithm is stochastic and may achieve different results each time it is run. As such, it is good practice to summarize the performance of the algorithm on a dataset using repeated evaluation and reporting the mean classification accuracy.

The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using heuristics or hyperparameter tuning.

The weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.

- Activation = Weights * Inputs + Bias

  If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.

- Predict 1: If Activation > 0.0
- Predict 0: If Activation <= 0.0

Given that the inputs are multiplied by model coefficients, like linear regression and logistic regression, it is good practice to normalize or standardize data prior to using the model.

The Perceptron is a linear classification algorithm. This means that it learns a decision boundary that separates two classes using a line (called a hyperplane) in the feature space. As such, it is appropriate for those problems where the classes can be separated well by a line or linear model, referred to as linearly separable.

The coefficients of the model are referred to as input weights and are trained using the stochastic gradient descent optimization algorithm.

Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated. The weights of the model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process is repeated for all examples in the training dataset, called an epoch. This process

of updating the model using examples is then repeated for many epochs.

Model weights are updated with a small proportion of the error each batch, and the proportion is controlled by a hyperparameter called the learning rate, typically set to a small value. This is to ensure learning does not occur too quickly, resulting in a possibly lower skill model, referred to as premature convergence of the optimization (search) procedure for the model weights.

•weights(t + 1) = weights(t) + learning_rate * (expected_i – predicted_) * input_i

Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed.

The initial values for the model weights are set to small random values. Additionally, the training dataset is shuffled prior to each training epoch. This is by design to accelerate and improve the model training process. Because of this, the learning algorithm is stochastic and may achieve different results each time it is run. As such, it is good practice to summarize the performance of the algorithm on a dataset using repeated evaluation and reporting the mean classification accuracy.

The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using heuristics or hyperparameter tuning.

2.4 Geometric Interpretation

Q5. What is Geometric Interpretation?

Ans :

For a perceptron, the decision boundary is precisely where the sign of the activation, a, changes from "1 to +1. In other words, it is the set of points x that achieve zero activation. The points that are not clearly positive nor negative. For simplicity, we'll first consider the case where there is no "bias" term (or, equivalently, the bias is zero). Formally, the decision boundary B is:

B = x: Σ w Ans: For a perceptron, the decision boundary is precisely where the sign of the activation, a, changes from "1 to +1. In other words, it is the set of points x that achieve zero activation. The points that are not clearly positive nor negative. For simplicity, we'll first consider the case where there is no "bias" term (or, equivalently, the bias is zero). Formally, the decision boundary B is:

$$B=x:\Sigma w_d x_d=0$$

We can now apply some linear algebra. Recall that "$_d w_d x_d$ is just the dot product between the vector w = (w1, w2, . . . ,wD) and the vector x. We will write this as w · x. Two vectors have a zero-dot product if and only if they are perpendicular. Thus, if we think of the weights as a vector w, then the decision boundary is simply the plane perpendicular to w.

This is shown pictorially in Figur. Here, the weight vector is shown, together with it's perpendicular plane. This plane forms the decision boundary between positive points and negative points. The vector points in the direction of the positive examples and away from the negative examples.

One thing to notice is that the scale of the weight vector is irrelevant from the perspective of classification. Suppose you take a weight vector w and replace it with 2w. All activations are now doubled.

But their sign does not change. This makes complete sense geometrically, since all that matters is which side of the plane a test point falls on, now how far it is from that plane. For this reason, it is common to work with normalized weight vectors, w, that have length one; i.e.,

$$||w|| = 1.$$

The geometric intuition can help us even more when we realize that dot products compute projections. That is, the value w · x is just the distance of x from the origin when projected onto the vector

w. This is shown in Figure. In that figure, all the data points are projected onto w. Below, we can think of this as a one-dimensional version of the data, where each data point is placed according to its projection along w. This distance along w is exactly the activation of that example, with no bias.

From here, you can start thinking about the role of the bias term.

Previously, the threshold would be at zero. Any example with a negative projection onto w would be classified negative; any example with a positive projection, positive. The bias simply moves this threshold. Now, after the projection is computed, b is added to get the overall activation. The projection plus b is then compared against zero.

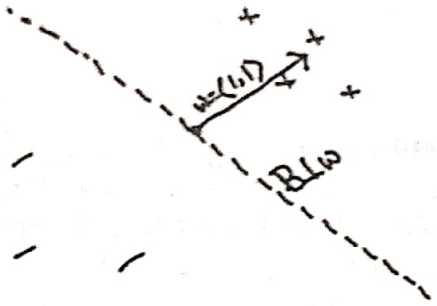


Fig.: Picture of data points with hyper plane and weight vector

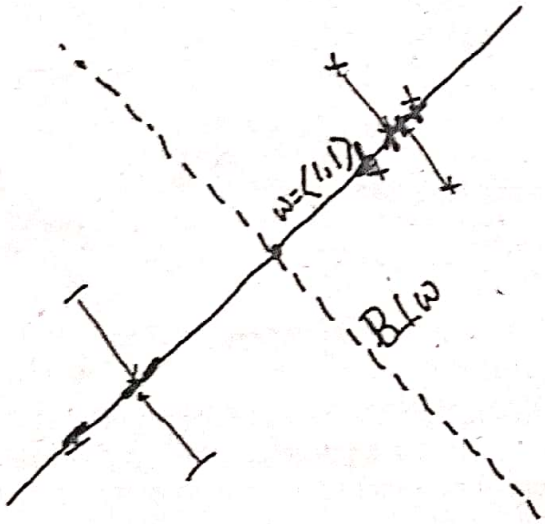


Fig.: The same picture as before, but with projections on to weightvector; then, below, those points along a one-dimensional axis with zeromarked

Thus, from a geometric perspective, the role of the bias is to shift the decision boundary away from the origin, in the direction of w. It is shifted exactly "b units. So if b is positive, the boundary is shifted away from w and if b is negative, the boundary is shifted toward w. This is shown in Figure . This makes intuitive sense: a positive bias means that more examples should be classified positive. By moving the decision boundary in the negative direction, more space yields a positive classification.

The decision boundary for a perceptron is a very magical thing. In D dimensional space, it is always a D–1-dimensional hyperplane. (In two dimensions, a 1-d hyperplane is simply a line. In three dimensions, a 2-d hyperplane is like a sheet of paper.) This hyperplane divides space in half. In the rest of this book, we'll refer to the weight vector, and to hyperplane it defines, interchangeably.

The perceptron update can also be considered geometrically. (For simplicity, we will consider the unbiased case.) Consider the situation in Figure. Here, we have a current guess as to the hyperplane, and positive training example comes in that is currently mis-classified. The weights are updated: $w \leftarrow w + yx$. This yields the new weight vector, also shown in the Figure. In this case, the weight vector changed enough that this training example is now correctly classified.

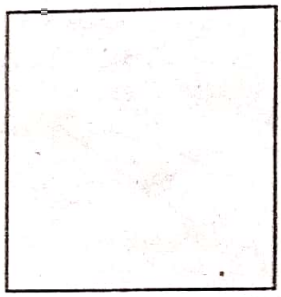


Fig.:perc:bias:perceptron picture with bias

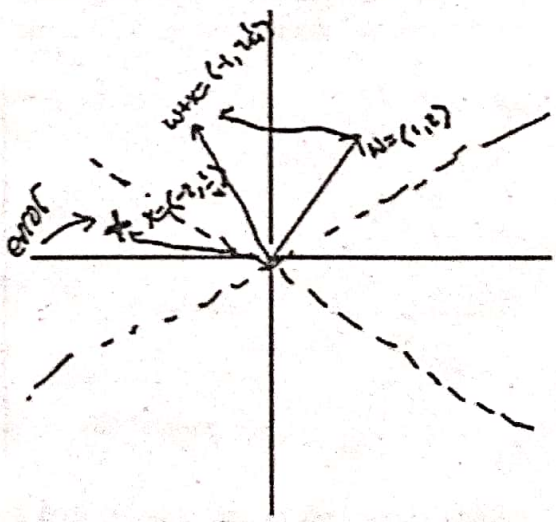
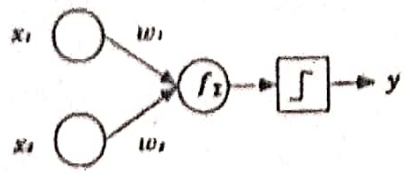


Fig.9: perceptron picture with update, nobias

2.5 Interpreting Perceptron Weights

Q6. What are weights in perceptron?

Ans :

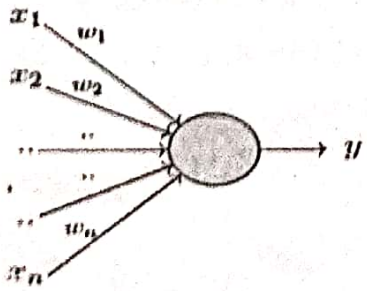


The weights are just scalar values that you multiple each input by before adding them and applying the nonlinear activation function i.e. w1 and w2 in the image. So putting it all together, if we have inputs x1 and x2 which produce a known output y then a perceptron using activation function A can be written as

$$y' = \Sigma A(xi^{*}wi)$$

The values of these weights are what get trained in the perceptron in such a way as to minimize some error metric between y and y'.

A perceptron is not the Sigmoid neuron we use in ANNs or any deep learning networks today.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq 0$$

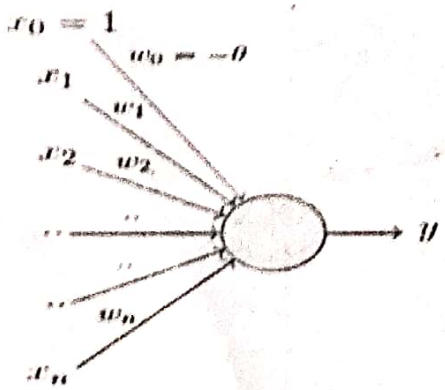
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < 0$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - 0 \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - 0 < 0$$

The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. Rewriting the threshold as shown above andmaking it a constant input with a variable weight, we would end up with something like the following:



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias (the threshold thing

mentioned above). We learn the weights, we get the function. Let's use a perceptron to learn an OR function.

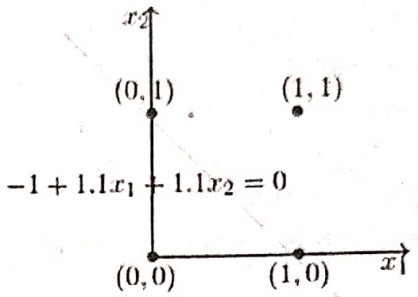
OR Function Using A Perceptron

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

One possible solution is

$w_0 = -1, w_1 = 1.1, w_2 = 1.1$



What's going on above is that we defined a few conditions (the weighted sum has to be more than or equal to 0 when the output is 1) based on the OR function output for various sets of inputs, we solved for weights based on those conditions and we got a line that perfectly separates positive inputs from those of negative.

2.6 Perceptron Convergence and Linear Separability

Q7. What is perceptron convergence and Lineaner Separability?

Ans :

Perceptron Convergence theorem states that a classifier for two linearly separable classes of patterns is always trainable in a finite number of training steps. In summary, the training of a single discrete perceptron two class classifier requires a change of weights if and only if a misclassification occurs.

Perceptron Convergence Theorem

In the classification of linearly separable patterns belonging to two classes only, the training task for the classifier was to find the weight w such that.

Completion of training with the fixed correction training rule for any initial weight vector and any correction increment constant leads to the following weights:

w"=wk0=wk0+1=wk0+2.....

with w" as the solution vector for equation.

Integer k0 is the training step number starting at which no more misclassification occurs, and thus no right adjustments take place for (k_0>=0)

This theorem is called as the "Perceptron Convergence Theorem".

Perceptron Convergence theorem states that a classifier for two linearly separable classes of patterns is always trainable in a finite number of training steps.

In summary, the training of a single discrete perceptron two class classifier requires a change of weights if and only if a misclassification occurs.

In the reason for misclassification is (w^tx<0\) then all weights are increased in proportion wo xi . If \(w^tx>0) then all weights are decreased in proportion to xi

Summary of the Perceptron Convergence Algorithm:

Variables and Parameters: x(n)=(m+1) by 1 input vector=[+1,x1(n),x2(n),.....xm(n)]T

w(n)=(m+1) by 1 weight vector=[b(n), w1(n),w2(n),.....wm(n)]T

b(n)= bias

y(n)= actual response

d(n)= desired response

η= learning rate parameter, a +ve constant less than unity

1. **Initialization**

   Set w(0)=0 , then perform the following computations for time step n=1,2

2. **Activation**

   At time step n, activate the perceptron by applying input vector x(n) and desired response d(n).

3. **Computation of actual response**

   Compute the actual response of the perceptron:

   y(n)=sgn[wT(x)x(n)]

4. **Adaptation of weight vector**

   Update the weight vector of the perceptron:

   w(n+1)=w(n)+ç[d(n)"y(n)]x(n)

5. Continuation: Increment time step n by 1, go to step 1

In Euclidean geometry, linear separability is a property of two sets of points. This is most easily visualized in two dimensions (the Euclidean plane) by thinking of one set of points as being colored blue and the other set of points as being colored red.

2.7 Improved Generalization, Limitions of Perception

Q8. How do you improve generalization in neural network and what are the limitions of perception?

Ans :

One method for improving network generalization is to use a network that is just large enough to provide an adequate fit. The larger network you use, the more complex the functions the network can create. If you use a small enough network, it will not have enough power to overfit the data.

2.7.1 Limitations of the Perceptron

Q9. How Limitations of the Perceptron.

Ans :

A perceptron model has limitation as follows:

The output of a perceptron can only be a binary number (0 or 1) due to the hard limit transfer function. Perceptron can only be used to classify. The Linearly separable sets of input vectors. If input vectors are non-Linear, it is not easy to classify then properly.

2.8 Importance of Good Features

Q10. What are the importance of Good Features?

(OR)

What are good features for machine learning?

Ans:

There are three main goals to feature selection. Improve the accuracy with which the model is able to predict for new data. Reduce computational cost. Produce a more interpretable model.

A huge challenge for engineers and researchers in the fields of data mining and Machine

Learning (ML) is high-dimensional data analysis. Feature selection offers a simple yet effective way to overcome this challenge by eliminating redundant and irrelevant data. Removing the irrelevant data improves learning accuracy, reduces the computation time, and facilitates an enhanced understanding for the learning model or data. While developing a ML model in real-life, most often not all the variables in the dataset are useful. Addition of redundant variables decreases the generalization competence of the model and may also decrease the overall precision of a classifier. Also, if more variables are added to a model, it results in a complex model being developed.

Feature Selection

In statistics and Machine learning, feature selection (also known as variable selection, attribute selection, or variable subset selection) is the practice of choosing a subset of relevant features (predictors and variables) for use in a model construction. It is the automatic selection of attributes present in the data (such as columns in tabular data) that are most significant and appropriate to the predictive modeling problem that one is working on.

Feature Selection and Dimensionality Reduction

Feature selection is different from dimensionality reduction. Both methods work to decrease the number of attributes in the dataset; however, dimensionality reduction works by creating new groupings of attributes, whereas feature selection methods include and remove attributes available in the data without modifying the attributes. Examples of dimensionality reduction methods are principal component analysis, singular value decomposition, and sammon's mapping.

Objective of Feature Selection

The objective of feature selection in ML is to identify the best set of features that enable one to build useful and constructive models of the subject one is trying to study. The methods for feature selection in Machine Learning can be classified into the following categories:

Supervised methods: These methods are used for labeled data, and are also used to classify the relevant features for Increasing the efficiency of supervised models, such as classification and regression.

Unsupervised methods: These methods are used for unlabelled data.

From a taxonomic point of view these methods can be classified under:

Filter methods

These methods collect the fundamental properties of the features that are measured through univariate statistics instead of using cross-validation performance. These methods are quicker and less expensive computationally than wrapper methods. While dealing with high-dimensional data, it is computationally cheaper to use filter methods.

Wrapper methods: Wrappers necessitate a method to search the space of all possible subsets of features, assessing a classifier with that feature subset, and evaluating their quality by learning. The feature selection process is based on a particular ML algorithm that one tries to fit on a given dataset. The wrapper methods usually provide a better predictive accuracy than filter methods.

Embedded methods: These methods cover the advantages of both filter and wrapper methods by not only comprising interactions of features but also by retaining a reasonable computational cost.

Conclusion

The advantages of feature selection can be summed up as:

Decreases over-fitting: Less redundant data means less chances of making decisions based on noise.

Reduces training time: Less data means that the algorithms train sooner.

Improved accuracy: Less ambiguous data means improvement of modeling accuracy

2.9 Irrelevant and Redundant Features

Q11. Explain about Irrelevant and Redundant Features?

Ans :

One big difference between learning models is how robust they are tothe addition of noisy or irrelevant features. Intuitively, an irrelevantfeature is one that is completely uncorrelated with the predictiontask. A feature 'f' whose expectation does not depend on the labelE[f | Y] = E[f] might be irrelevant. For instance, the presence ofthe word "the" might be largely irrelevant for predicting whether acourse review is positive or negative.

A secondary issue is how well these algorithms deal with redundant features. Two features are redundant if they are highly correlated, regardless of whether they are correlated with the task ornot. For example, having a bright red pixel in an image at position(20, 93) is probably highly redundant with having a bright red pixelat position (21, 93). Both might be useful (e.g., for identifying fire hydrants), but because of how images are structured, these two features are likely to co-occur frequently.

When thinking about robustness to irrelevant or redundant features, it is usually not worthwhile thinking of the case where one has999 great features and 1 bad feature. The interesting case is when thebad features outnumber the good features, and often outnumber by alarge degree. The question is how robust are algorithms in this case.

For shallow decision trees, the model explicitly selects featuresthat are highly correlated with the label. In particular, by limiting thedepth of the decision tree, one can at least hope that the model will beable to throw away irrelevant features. Redundant features are almostcertainly thrown out: once you select one feature, the second featurenow looks mostly useless. The only possible issue with irrelevantfeatures is that even though they're irrelevant, they happen to correlatewith the class label on the training data, but chance.

As a thought experiment, suppose that we have N training examples, and exactly half are positive examples and half are negativeexamples. Suppose there's some binary feature, f, that is completelyuncorrelated with the label. This feature has a 50/50 chance of appearing in any example, regardless of the label. In principle, the decision tree should not select this feature. But, by chance, especially if Nis small, the feature might look correlated with the label. This is analogous to flipping two coins simultaneously N times. Even though thecoins are independent, it's entirely possible that you will observe asequence like (H, H),(T, T),(H, H),(H, H), which makes them lookentirely correlated! The hope is that as N grows, this becomes lessand less likely. In fact, we can explicitly compute how likely this is tohappen.

To do this, let's fix the sequence of N labels. We now flip a coin Ntimes and consider how likely it is that it exactly matches the label. This is easy: the probability is 0.5^N. Now, we would also be confusedif it exactly matched not the label, which has the same probability. Sothe chance that it looks perfectly correlated is $0.5^N + 0.5^N = 0.^{5N-1}$.Thankfully, this shrinks down very small (e.g., 10"6) after only 21data points, meaning that even with a very small training set, thechance that a randomfeature happens to correlate exactly with thelabel is tiny.

This makes us happy. The problem is that we don't have one irrelevant feature: we have many! If we randomly pick two irrelevant features, each has the same probability of perfectly correlating: 0.5^{N-1}.But since there are two and they're independent coins, the chancethat either correlates perfectly is $2 \times 0.5^{N-1} = 0.5^{N-2}$. In general,if we have K irrelevant features, all of which are random independent coins, the chance that at least one of them perfectly correlates is0.5^{N-K}. This suggests that if we have a sizeable number K of irrelevant features, we'd better have at least K + 21 training examples.

Unfortunately, the situation is actually worse than this. In theabove analysis we only considered the case of perfect correlation. Wecould also consider the case of partial correlation, which would yieldeven higher probabilities. Suffice it to say that even decision trees canbecome confused.

In the case of K-nearest neighbors, the situation is perhaps moredire. Since KNN weighs each feature just as much as another feature,the introduction of irrelevant features can completely

mess up KNNprediction. In fact, as you saw, in high dimensional space, randomlydistributed points all look approximately the same distance apart. Ifwe add lots and lots of randomly distributed features to a data set,then all distances still converge.In the case of the perceptron, one can hope that it might learn toassign zero weight to irrelevant features. For instance, consider abinary feature is randomly one or zero independent of the label. Ifthe perceptron makes just as many updates for positive examplesas for negative examples, there is a reasonable chance this featureweight will be zero. At the very least, it should be small.

2.10 Feature Pruning and Normalization

Q12. What is normalization in machine learning?

Ans:

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

Normalization in Machine Learning:

"Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale".

Although Normalization is no mandate for all datasets available in machine learning, it is used whenever the attributes of the dataset have different ranges. It helps to enhance the performance and reliability of a machine learning model. In this article, we will discuss in brief various Normalization techniques in machine learning, why it is used, examples of normalization in an ML model, and much more. So, let's start with the definition of Normalization in Machine Learning.

What is Normalization in Machine Learning?

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Mathematically, we can calculate normalization with the below formula:

Xn = (X - Xminimum) / (Xmaximum - Xminimum)

- Xn = Value of Normalization
- Xmaximum = Maximum value of a feature
- Xminimum = Minimum value of a feature

Example:

Let's assume we have a model dataset having maximum and minimum values of feature as mentioned above. To normalize the machine learning model, values are shifted and rescaled so their range can vary between 0 and 1. This technique is also known as Min-Max scaling. In this scaling technique, we will change the feature values as follows:

Case 1:

If the value of X is minimum, the value of Numerator will be 0; hence Normalization will also be 0.

Xn = (X - Xminimum) / (Xmaximum - Xminimum)

Put X =Xminimum in above formula, we get;

Xn = Xminimum- Xminimum/ (Xmaximum - Xminimum)

Xn = 0

Case 2:

If the value of X is maximum, then the value of the numerator is equal to the denominator; hence Normalization will be 1.

Xn = (X - Xminimum) / (Xmaximum - Xminimum)

Put X =Xmaximum in above formula, we get;

Xn = Xmaximum - Xminimum/ (Xmaximum - Xminimum)

Xn = 1

Case 3:

On the other hand, if the value of X is neither maximum nor minimum, then values of normalization will also be between 0 and 1.

Hence, Normalization can be defined as a scaling method where values are shifted and rescaled to maintain their ranges between 0 and 1, or in other words; it can be referred to as Min-Max scaling technique.

Normalization techniques in Machine Learning:

Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

Min-Max Scaling: This technique is also referred to as scaling. As we have already discussed above, the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.

Standardization scaling: Standardization scaling is also known as Z-score normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.

Hence, standardization can be expressed as follows:

Here, μ represents the mean of feature value, and ó represents the standard deviation of feature values.

However, unlike Min-Max scaling technique, feature values are not restricted to a specific range in the standardization technique.

This technique is helpful for various machine learning algorithms that use distance measures such as KNN, K-means clustering, and Principal component analysis, etc. Further, it is also important that the model is built on assumptions and data is normally distributed.

Difference between Normalization and Standardization:

Normalization	Standardization
This technique uses minimum and max values for scaling of model.	This technique uses mean and standard deviation for scaling of model.
It is helpful when features are of different scales.	It is helpful when the mean of a variable is set to 0 and the standard deviation is set to 1.
Scales values ranges between [0, 1] or [-1, 1].	Scale values are not restricted to a specific range.
It got affected by outliers.	It is comparatively less affected by outliers.
Scikit-Learn provides a transformer called MinMaxScaler for Normalization.	Scikit-Learn provides a transformer called StandardScaler for Normalization.
It is also called Scaling normalization.	It is known as Z-score normalization.
It is useful when feature distribution is unknown.	It is useful when feature distribution is normal.

When to use Normalization or Standardization:

Which is suitable for our machine learning model, Normalization or Standardization? This is probably a big confusion among all data scientists as well as machine learning engineers. Although both terms have the almost same meaning choice of using normalization or standardization will depend on your problem and the algorithm you are using in models.

1. Normalization is a transformation technique that helps to improve the performance as well as the accuracy of your model better. Normalization of a machine learning model is useful when you don't know feature distribution exactly. In other words, the feature distribution of data does not follow a Gaussian (bell curve) distribution. Normalization must have an abounding range, so if you have outliers in data, they will be affected by Normalization.

   Further, it is also useful for data having variable scaling techniques such as KNN, artificial neural networks. Hence, you can't use assumptions for the distribution of data.

2. Standardization in the machine learning model is useful when you are exactly aware of the feature distribution of data or, in other words, your data follows a Gaussian distribution. However, this does not have to be necessarily true. Unlike Normalization, Standardization does not necessarily have a bounding range, so if you have outliers in your data, they will not be affected by Standardization.

   Further, it is also useful when data has variable dimensions and techniques such as linear regression, logistic regression, and linear discriminant analysis.

   Example: Let's understand an experiment where we have a dataset having two attributes, i.e., age and salary. Where the age ranges from 0 to 80 years old, and the income varies from 0 to 75,000 dollars or more. Income is assumed to be 1,000 times that of age. As a result, the ranges of these two attributes are much different from one another.

Because of its bigger value, the attributed income will organically influence the conclusion more when we undertake further analysis, such as multivariate linear regression. However, this does not necessarily imply that it is a better predictor. As a result, we normalize the data so that all of the variables are in the same range.

Further, it is also helpful for the prediction of credit risk scores where normalization is applied to all numeric data except the class column. It uses the tanh transformation technique, which converts all numeric features into values of range between 0 to 1.

Conclusion

Normalization avoids raw data and various problems of datasets by creating new values and maintaining general distribution as well as a ratio in data. Further, it also improves the performance and accuracy of machine learning models using various techniques and algorithms. Hence, the concept of Normalization and Standardization is a bit confusing but has a lot of importance to build a better machine learning model.

2.11 COMBINATORIAL FEATURE EXPLOSION

Q13. What is combinatorial explosion?

Ans :

Combinatorial explosion is the exponential growth rate at which most programs grow. The goal of AI is to avoid the combinatorial explosion issue as much as possible. The combinatorial explosion issue occurs when a number of possible combinations are created by increasing the number of entities.

The combinatorial explosion issue occurs when a number of possible combinations are created by increasing the number of entities. A small increase in the number of entities quickly increases the number of computations that need to be done and will cause it to reach the computational limit.

2.12 Evaluating Model Performance, Cross Validation

Q14. Explain Evaluating Model Performance over Cross Validation.

Ans :

Cross-validation is a model evaluation technique. The central intuition behind model evaluation is to figure out if the trained model is generalizable, that is, whether the predictive power we observe while training is also to be expected on unseen data.

The ultimate goal of a Machine Learning Engineer or a Data Scientist is to develop a Model in order to get Predictions on New Data or Forecast some events for future on Unseen data. A Good Model is not the one that gives accurate predictions on the known data or training data but the one which gives good predictions on the new data and avoids overfitting and underfitting.

After completing this tutorial, you will know:

- That why to use cross validation is a procedure used to estimate the skill of the model on new data.
- There are common tactics that you can use to select the value of k for your dataset.
- There are commonly used variations on cross-validation such as stratified and LOOCV that are available in scikit-learn.
- Practical Implementation of k-Fold Cross Validation in Python

To derive a solution, we should first understand the problem. Before we proceed to Understanding Cross Validation let us first understand Overfitting and Underfitting.

Understanding Underfitting and Overfitting:

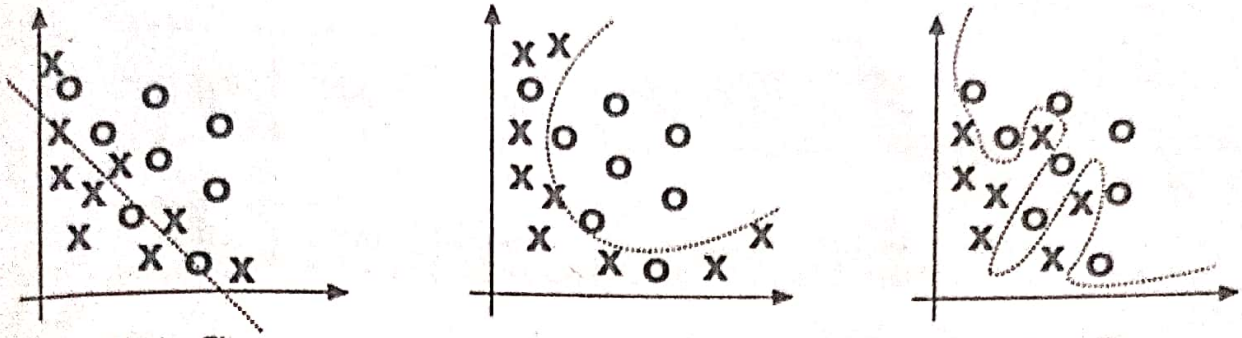
Overfit Model

Overfitting occurs when a statistical model or machine learning algorithm captures the noise of the data. Intuitively, overfitting occurs when the model or the algorithm fits the data too well.

Overfitting a model result in good accuracy for training data set but poor results on new data sets. Such a model is not of any use in the real world as it is not able to predict outcomes for new cases.

Underfit Model

Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough. Underfitting is often a result of an excessively simple model. By simple we mean that the missing data is not handled properly, no outlier treatment, removing of irrelevant features or features which do not contribute much to the predictor variable.



How to tackle Problem of Overfitting:

The answer is Cross Validation

A key challenge with overfitting, and with machine learning in general, is that we can't know how well our model will perform on new data until we actually test it.

To address this, we can split our initial dataset into separate training and test subsets.There are different types of Cross Validation Techniques but the overall concept remains the same,

- To partition the data into a number of subsets
- Hold out a set at a time and train the model on remaining set
- Test model on hold out set

Repeat the process for each subset of the dataset

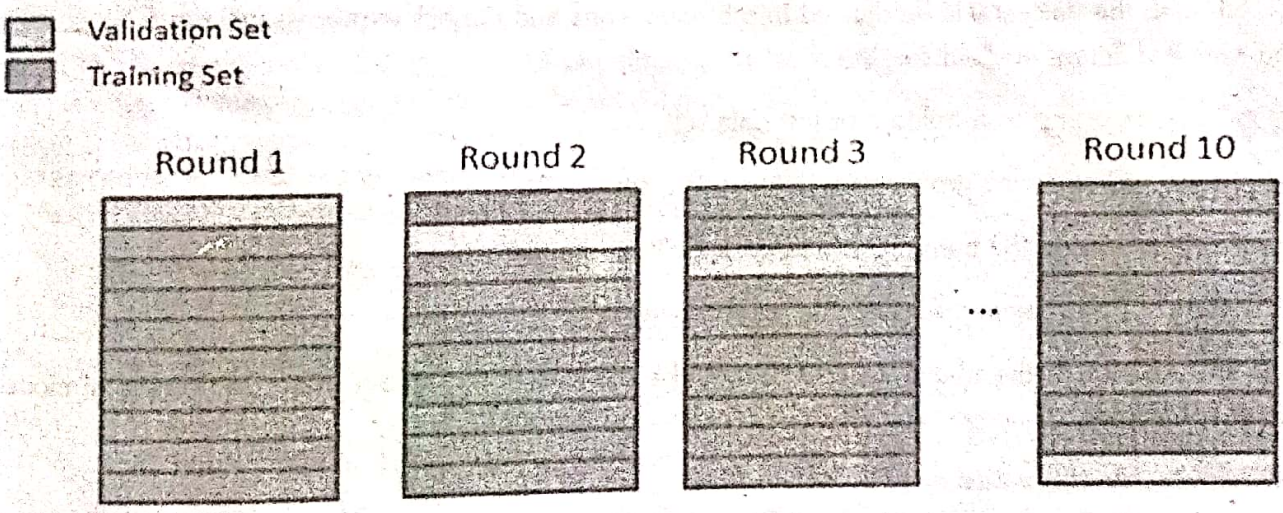


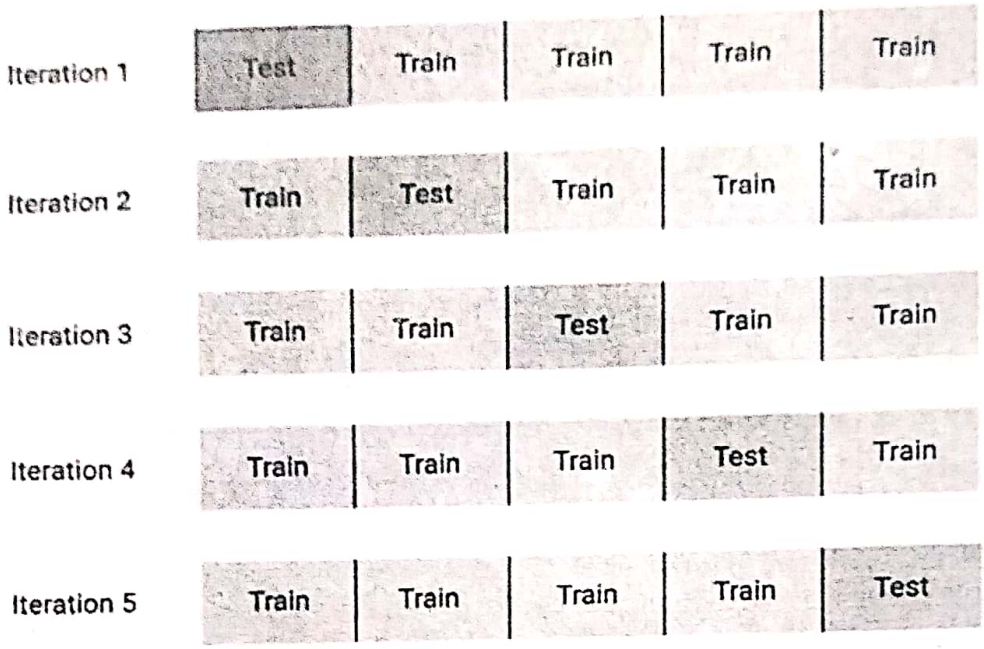
Fig:The process of cross validation in general

Types of Cross Validation:

- K-Fold Cross Validation
- Stratified K-fold Cross Validation
- Leave One Out Cross Validation

Let's understand each type one by one

k-Fold Cross Validation:



The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

If k=5 the dataset will be divided into 5 equal parts and the below process will run 5 times, each time with a different holdout set.

1. Take the group as a holdout or test data set
2. Take the remaining groups as a training data set
3. Fit a model on the training set and evaluate it on the test set
4. Retain the evaluation score and discard the model

At the end of the above process Summarize the skill of the model using the sample of model evaluation scores.

How to decide the value of k?

The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset.

A value of k=10 is very common in the field of applied machine learning, and is recommend if you are struggling to choose a value for your dataset.

If a value for k is chosen that does not evenly split the data sample, then one group will contain a remainder of the examples. It is preferable to split the data sample into k groups with the same number of samples, such that the sample of model skill scores are all equivalent.

Stratified k-Fold Cross Validation

Same as K-Fold Cross Validation, just a slight difference

The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class outcome value. This is called stratified cross-validation.

In below image, the stratified k-fold validation is set on basis of Gender whether M or F

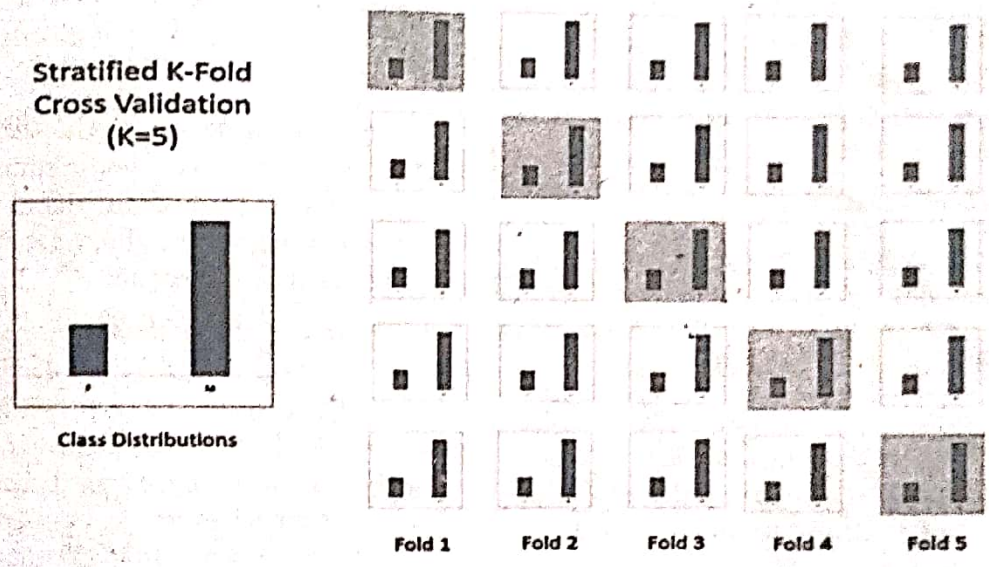


Fig: stratified k-fold cross validation

Leave One Out Cross Validation (LOOCV):

This approach leaves 1 data point out of training data, i.e. if there are n data points in the original sample then, n-1 samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which the original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness.

The number of possible combinations is equal to the number of data points in the original sample or n.

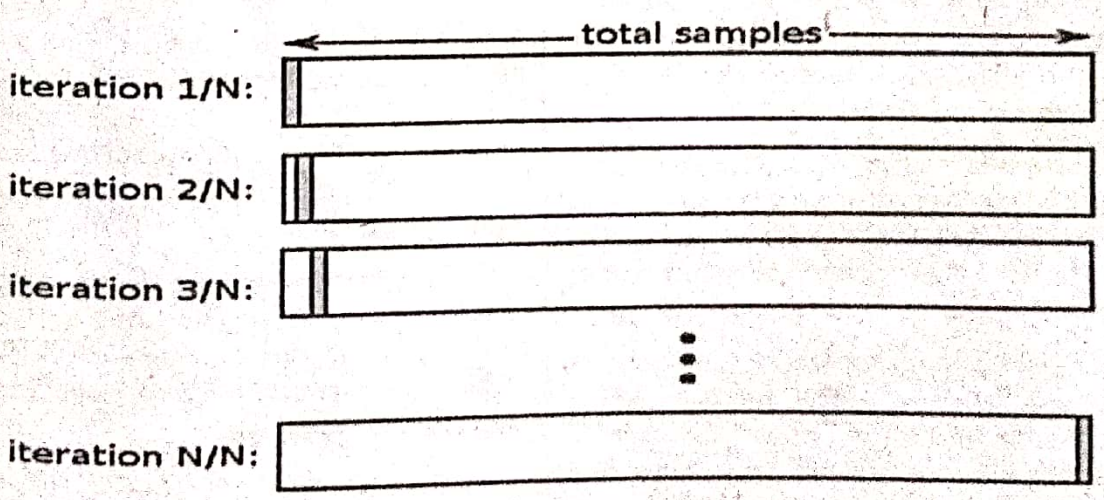


Fig: Representation of leave one out cross validation

Cross Validation is a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate over-fitting.

2.13 Hypothesis Testing and Statistical Significance

Q15. What is the significance of using statistical test in machine learning?

Ans:

Statistical test is a way to determine whether the random variable is following the null hypothesis or alternate hypothesis. It basically tells whether the sample and population or two/ more samples have significant differences.

Statistical Hypothesis Tests:

Generally, a statistical hypothesis test for comparing samples quantifies how likely it is to observe two data samples given the assumption that the samples have the same distribution.

The assumption of a statistical test is called the null hypothesis and we can calculate statistical measures and interpret them in order to decide whether or not to accept or reject the null hypothesis.

In the case of selecting models based on their estimated skill, we are interested to know whether there is a real or statistically significant difference between the two models.

If the result of the test suggests that there is insufficient evidence to reject the null hypothesis, then any observed difference in model skill is likely due to statistical chance.

If the result of the test suggests that there is sufficient evidence to reject the null hypothesis, then any observed difference in model skill is likely due to a difference in the models.

The results of the test are probabilistic, meaning, it is possible to correctly interpret the result and for the result to be wrong with a type I or type II error. Briefly, a false positive or false negative finding.

Comparing machine learning models via statistical significance tests imposes some expectations that in turn will impact the types of statistical tests that can be used; for example:Skill Estimate. A specific measure of model skill must be chosen. This could be classification accuracy (a proportion) or mean absolute error (summary statistic) which will limit the type of tests that can be used Repeated Estimates. A sample of skill scores is required in order to calculate statistics. The repeated training and testing of a given model on the same or different data will impact the type of test that can be used.

Distribution of Estimates. The sample of skill score estimates will have a distribution, perhaps Gaussian or perhaps not. This will determine whether parametric or nonparametric tests can be used.

Central Tendency. Model skill will often be described and compared using a summary statistic such as a mean or median, depending on the distribution of skill scores. The test may or may not take this directly into account.

The results of a statistical test are often a test statistic and a p-value, both of which can be interpreted and used in the presentation of the results in order to quantify the level of confidence or significance in the difference between models. This allows stronger claims to be made as part of model selection than not using statistical hypothesis tests.

2.14 Debugging Learning Algorithms

Q16. How do you debug a machine learning algorithm?

Ans:

Debugging steps in the software world:

Inspect the system thoroughly to find it. Analyze the Error: Analyze the code to identify more issues and estimate the risk that the error creates. Prove the Analysis: Work with automated tests, after analyzing the bug you might find a few more errors in the application.

In traditional software development, a bug usually leads to the program crashing. While this is annoying for the user, it is critical for the developer - when the program fails the developer can inspect errors to understand why.

With machine learning models, the developer sometimes encounters errors but all too often the program crashes without a clear reason why. While these issues can be debugged manually, machine learning models most often fail because of poor

output predictions. What's worse is that when they fail, there is usually no signal about why or when the models failed. And to make thesituation even more complicated, it might be a result of a number of things, including bad training data, high loss error, or a lack of convergence rate.

Six Ways to Debug a Machine Learning Model.

1. How to find flaws in the input data
2. How to make the model learn more from less data
3. How to prepare data for training and avoid common pitfalls
4. How to find the optimal model hyper-parameters
5. How to schedule learning rates in order to reduce overfitting
6. How to monitor training progress with Weights and Biases

1. How to find flaws in the input data:

There are two aspects to consider when wanting to know whether our data is up to the task of training a good model:

Can the data predict what we want to predict?

Is there enough data?

To figure out whether our model contains predictive information, we can ask ourselves: can a human make a prediction given this data?

If a human cannot understand an image or a text, chances are our model won't make much sense of it either. If there is not enough predictive information, adding more inputs to our model won't make it better; in contrast, the model will overfit and become less accurate.

Once our data has enough predictive information, we need to figure out whether we have enough data to train a model to extract the signal. There are a couple of rules of thumb to follow:

1. For classification, we should have at least 30 independent samples per class
2. We should have at least 10 samples for any feature, especially for structured data problems, and finally
3. The size of our dataset is directly proportional to the number of parameters in our model. These rules might need to be tailored to your specific application. If you can make use of transfer learning, then you can drastically reduce the number of samples needed.

2. How to make the model learn more from less data

In many situations, we simply do not have enough data. In this case, one of the best options is to augment the data. Taking augmentation, a step further, we can generate our own data with generative models such as autoencoders and generative adversarial networks.

Likewise, we can find external public data, which can be found available on the Internet. Even if the data were not originally collected for our purpose, we can potentially relabel it or use it for transfer learning. We can train a model on a large dataset for a different task and then use that model as a basis for our task. Similarly, we can find a model that someone else has trained for a different task and repurpose it for our task.

It's important to remember that in most cases the quality of the data trumps the quantity of the data. Having a small, high-quality dataset and training a simple model is the best practice to find problems in the data early in the training process. A mistake many data scientists make is that they spend time and money on getting a big dataset, only to figure out later that they have the wrong kind of data for their project.

3. How to prepare data for training and avoid common pitfalls

There are 3 common ways to pre-process the data features for the training process:

- Standardization ensures that all of the data has a mean of 0 and a standard deviation of 1. This is the most common way of scaling features. It is even more useful if you suspect that the data contains outliers.
- Min-Max rescaling scales all data between 0 and 1 by subtracting the minimum value and then dividing by the range of values.

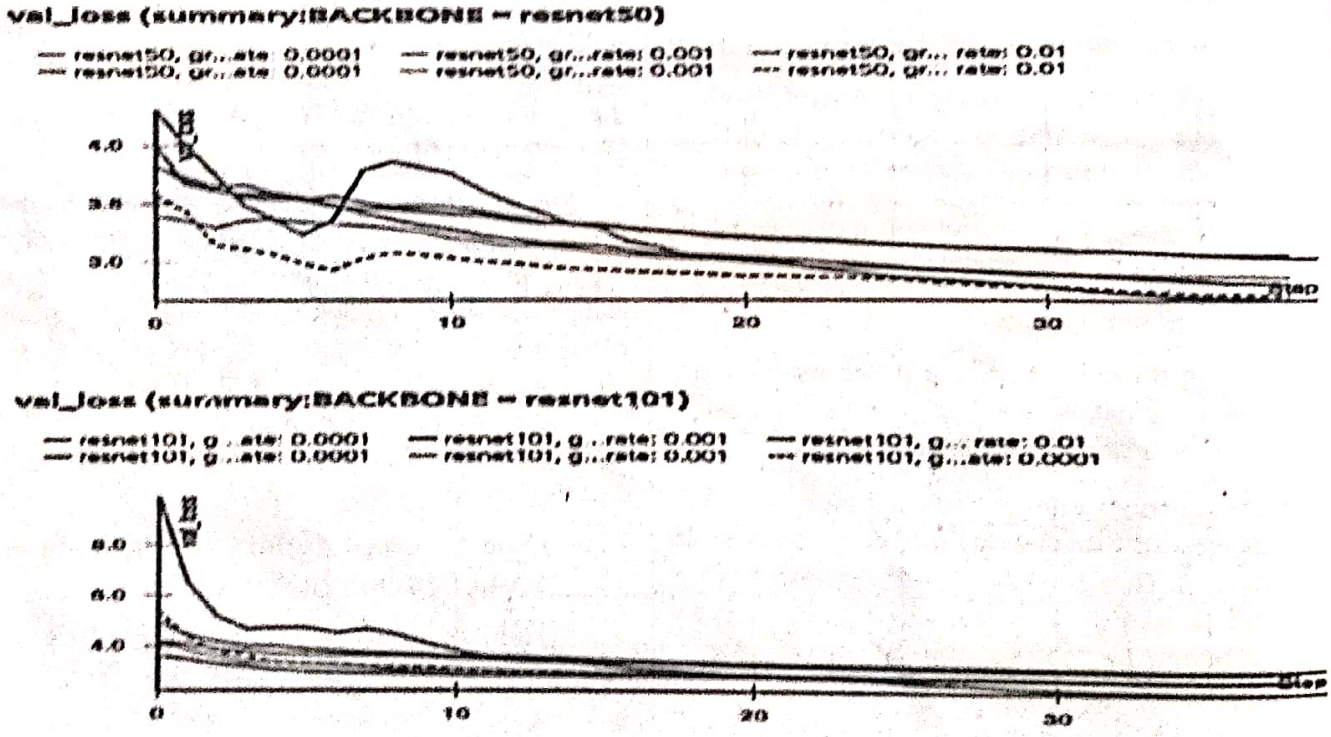
- Mean normalization ensures your data has values between (-1) and 1 with a mean of 0. We subtract the mean and divide by the range of the data.

  However, we prepare the features, it is important to only measure the scaling factors, the mean, and the standard deviation on the test set. If we measure these factors over the entire dataset, the algorithm might perform better on the test set than it will in production, due to this information exposure.

4. How to find the optimal model hyper-parameters

Manually tuning the hyperparameters of a neural network model can be extremely tedious. That is because there is not a scientific rule to use when it comes to hyper-parameter tuning. That is why many data scientists have moved towards automatic hyperparameter search, using some sort of non-gradient based optimization algorithm.

To see how we can find the optimal hyperparameters for our model with Weight and Biases, let's take a look at this example from a Mask R-CNN computer vision model. With the goal to implement Mask R-CNN for semantic segmentation tasks, Connor and Trent tweak different hyper-parameters that govern how the model operates: learning rate, gradient clip normalization, momentum, weight decay, scale ratios, weights of various loss functions... They want to know how the semantic segmentation of the images progressed as the model trained with different hyper-parameters, so they integrated an ImageCallback() class to sync to wandb. Furthermore, they included a script to run parameters sweeps that can be adapted to work with different hyper-parameters or different values of the same ones.

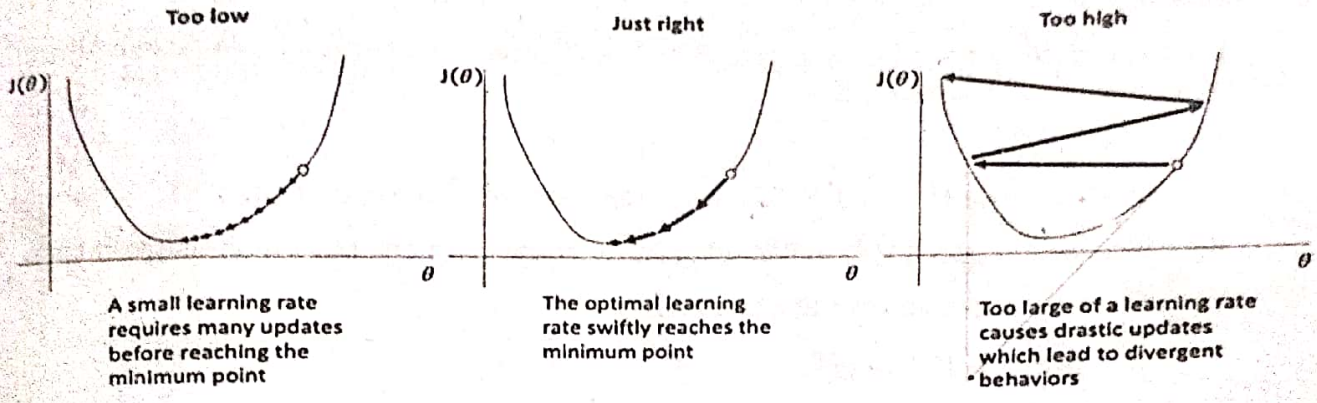


Their results can be found on the wandb run page. It seems like a high gradient clipping set and a high learning rate can lead to better model accuracy, with the validation loss scores decreasing quickly given an increasing number of iterations.

5. How to schedule learning rates in order to reduce overfitting

One of the most important hyperparameters is the learning rate, which is difficult to optimize. Small learning rate leads to slow training, and large learning rate leads to model overfitting.

When it comes to finding a learning rate, standard hyperparameter search techniques are not the best choice. For the learning rate, it is better to perform a line search and visualize the loss for different learning rates, as this will give you an understanding of how the loss function behaves. When doing a line search, it is better to increase the learning rate exponentially. You are more likely to care about the region of smaller learning rates than about very large learning rates.



In the beginning, our model might be far away from the optimal solution, and so because of that, we want to move as fast as possible. As we approach the minimum loss, however, we want to move slower to avoid overshooting. A popular method is to anneal the learning rate over time, which recommends starting with a relatively high learning rate and then gradually lowering the learning rate during training. The intuition is that we would like to move quickly from the initial parameters to a range of good-enough parameter values, and then we can explore the "deeper, but narrower parts of the loss function." The most popular form of learning rate annealing is a step decay where the learning rate is reduced by some percentage after a set number of training epochs. More generically, we should define a learning rate schedule to update the rate during training according to a specified rule.

6. How to monitor training progress with Weights and Biases

An important part of debugging a model is knowing when things go wrong before you have invested significant amounts of time training the model. WandB provides a seamless way to visualize and track machine learning experiments. As described in this GitHub repo, you can search/compare/visualize training runs, analyze system usage metrics alongside runs, replicate historic results, and many more.

2.15 Bias Variance Tradeoff

Q17. What is bias and variance tradeoff in machine learning?

Ans :

Bias

Bias is the simplifying assumptions made by the model to make the target function easier to approximate. Varianceis the amount that the estimate of the target function will change given different training data. Trade-off is tension between the error introduced by the bias and the variance.

(OR)

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 training and test data.

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 training data but has high error rates on test data.

Mathematically

Let the variable we are trying to predict as Y and other covariates as X. We assume there is a relationship between the two such that

$Y=f(X) + e$

Where e is the error term and it's normally distributed with a mean of 0.

We will make a model $f\hat{}(X)$ of f(X) using linear regression or any other modeling technique.

So the expected squared error at a point x is

$$Err(x) = E[(Y-\hat{f}(x)^2]$$

The Err(x) can be further decomposed as

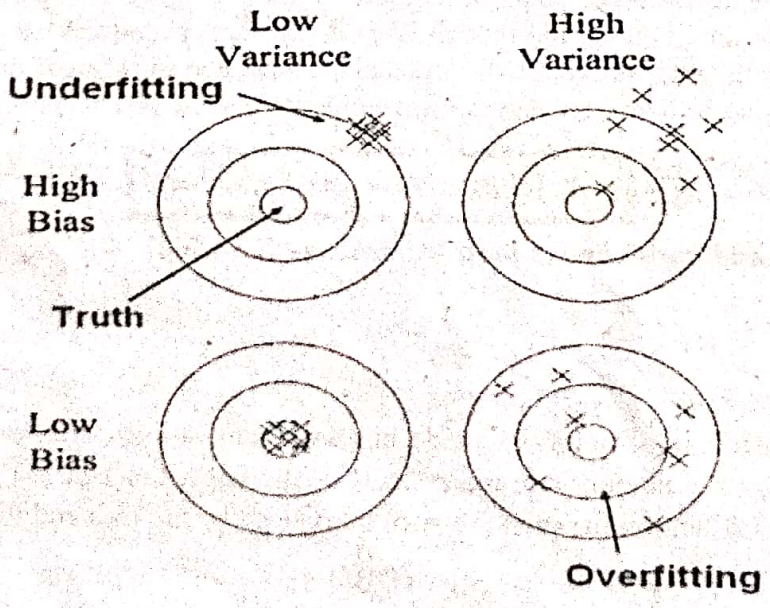
$$Err(x) = (E|\hat{f}(x)|-f(x)^2 + E[(\hat{f}(x)-E[\hat{f}(x)|)^2]+\sigma_c^2$$

$$Err(x) = Bias^2 + variance + Inreducible\ Error$$

Err(x) is the sum of $Bias^2$, variance and the irreducible error.

Irreducible error is the error that can't be reduced by creating good models. It is a measure of the amount of noise in our data. Here it is important to understand that no matter how good we make our model, our data will have certain amount of noise or irreducible error that cannot be removed.

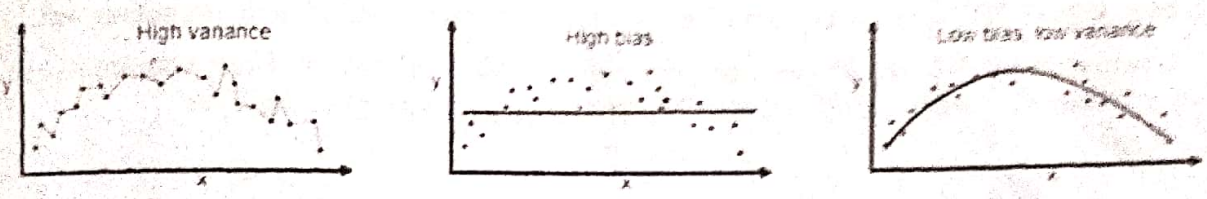
Bias and variance using bulls-eye diagram



In the above diagram, center of the target is a model that perfectly predicts correct values. As we move away from the bulls-eye our predictions become get worse and worse. We can repeat our process of model building to get separate hits on the target.

In supervised learning, underfitting happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kinds of models are very simple to capture the complex patterns in data like Linear and logistic regression.

In supervised learning, overfitting happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance. These models are very complex like Decision trees which are prone to overfitting.



overfitting underfitting Good balance

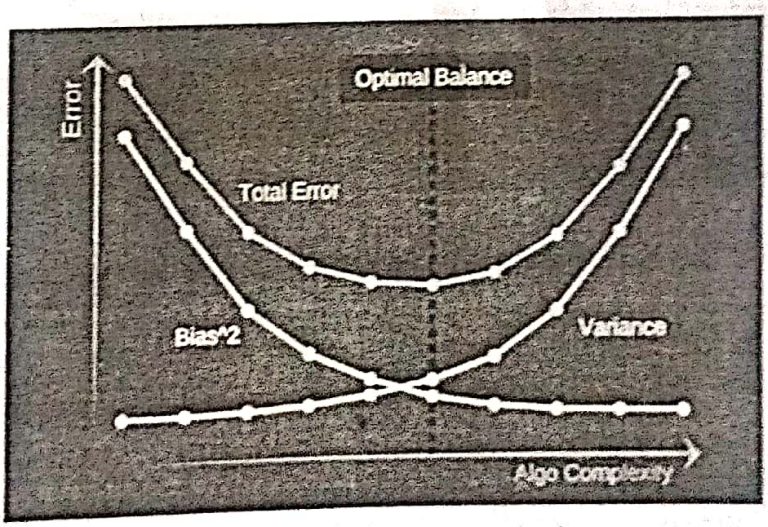
Bias Variance Trade off

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.

Total Error = Bias^2+ Variance + Irreducible Error

To build a good model, we need to find a good balance between bias and variance such that it minimizes the total error.



An optimal balance of bias and variance would never overfit or underfit the model.

2.16 Linear Models: The Optimization Framework for Linear Models

Q18. Explain Optimization Framework for Linear Models in machine learning?

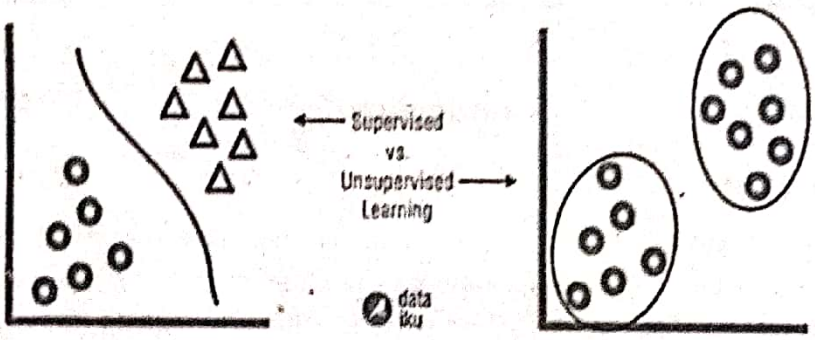
Ans:

The AI/ML that we actually interact with in our day-to-day lives is usually "Weak AI," which means that it is programmed to do one specific task. This includes things like credit card fraud detection, spam email classification, and movie recommendations on Netflix.

We can break machine learning into two key subcategories:

Supervised ML, which uses a set of input variables to predict the value of an output variable.

Unsupervised ML, which infers patterns from an unlabeled dataset. Here, you aren't trying to predict anything, you're just trying to understand patterns and groupings in the data.



The idea is that we will look at historical data to train a model to learn the relationships between features, or variables, and a target, the thing we're trying to predict. This way, when new data comes in, we can use the feature values to make a good prediction of the target, whose value we do not yet know.

Supervised learning can be further split into regression (predicting numerical values) and classification (predicting categorical values). Some algorithms can only be used for regression, others only for classification, and many for both.

Machine Learning Algorithms in Action: Practical Examples

Let's say we're the owners of a candy store, Willy Wonka's Candy, and we want to do a better job of predicting how much our customers will spend this week, in order to stock our shelves more appropriately. To get even more specific, let's explore one specific customer named George. George is a 65-year-old mechanic who has children and spent $10 at our store last week. We're going to try to predict the following:

How much George will spend this week (hint: this is regression, because it is a dollar amount).

Whether George will be a "high spender," which we've defined as someone who will spend at least $25 at Willy Wonka's Candy this week (hint: this is a classification, because we're predicting a distinct category, high spender or not).

Linear Models:

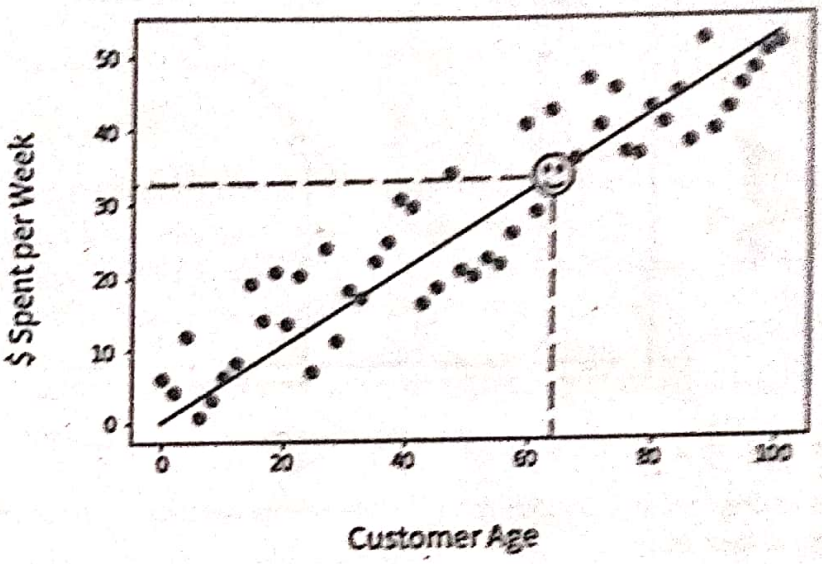
So now let's dive in and see how we can use a linear model. Remember, linear models generate a formula to create a best-fit line to predict unknown values. Linear models are considered "old school" and often not as predictive as newer algorithm classes, but they can be trained relatively quickly and are generally more straightforward to interpret, which can be a big plus!

We'll explore two types of linear models:

- Linear regression, which is used for regression (numerical predictions).
- Logistic regression, which is used for classification (categorical predictions). Don't get thrown off by the word "regression" in the name. I know, it's confusing. I didn't make the rules.

Linear Regression

Okay, let's imagine we have a simple model in which we're trying to just use age to predict how much George will spend at Willy Wonka's Candy this week.



The data points we used to train our model are in blue. The red line is the line of best fit, which the model generated, and captures the direction of those points as best as possible.

Here, it looks like the older somebody is, the more money they will spend. We know George is 65, so we'll find 65 on the x-axis and follow the green dotted line up until we meet the red "line of best fit." Now we can follow the second dotted line across to the y-axis, and land on our prediction — we would predict that George will spend $33 this week.

Where does this red "line of best fit" come from? Well, you may be familiar with the formula y = mx + b, the formula for a straight line. This is the foundation of linear regression. All we need to do is reformat a few variables, add an error term (e) to account for randomness, and fill in our target ($ spent) and features (age).

$$y = mx + b$$

$$y = b + m_1 * x$$

$$\$ \text{ spent this week} = b + m_1 * age + e$$

$$\$ \text{ spent this week} = b + m_1 * age + m_2 * has\_kids + \ldots + e$$

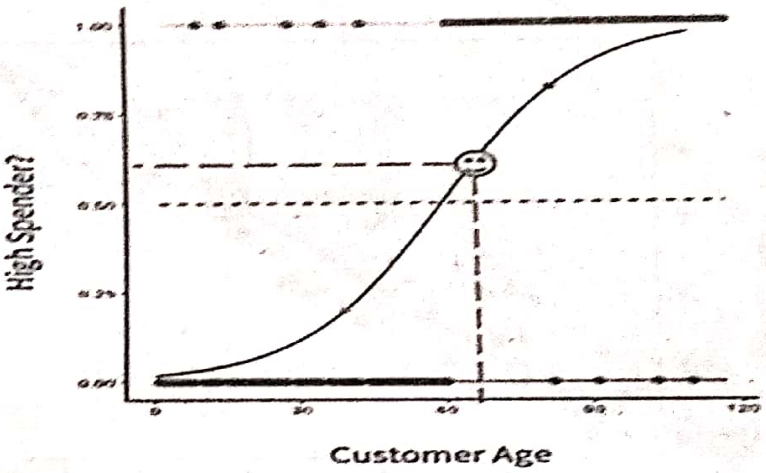
We'll train a model to learn the relationship between age and dollars spent this week from past data points. Our model will determine the values of m1 and b that best predict the dollars spent this week, given the age. We can easily add in more features, such as has_kids, and the model will then learn the value of m2 as well.

In the real world, of course, building a straight line like this is usually not realistic, as we often have more complex, non-linear relationships. We can manipulate our features manually to deal with this, but

that can be cumbersome, and we'll often miss out on some more complex relationships. However, the benefit is that it's quite straightforward to interpret — with a certain increase in age, we can expect a specific corresponding increase in dollars spent.

Logistic Regression:

Now, rather than trying to predict George's exact spending, let's just try to predict whether or not George will be a high spender. We can use logistic regression, an adaptation of linear regression for classification problems, to solve this.



The black dots at the top and bottom are the data points we used to train our model, and the S-shaped line is the line of best fit.

You may have noticed that all data points in the above chart are either a 0 or a 1. This is because each point is marked as either a low spender (0) or a high spender (1). Now, we will use a logistic function to generate an S-shaped line of best fit, also called a Sigmoid curve, to predict the likelihood of a data point belonging to one category, in this case high spender. We also could have predicted the likelihood of being a low spender, it doesn't matter. We'll then use a predefined threshold to make a final prediction.

Let's predict for George again — we'll find 65 on the x-axis and then map it up to the S-shaped line and then across. Now, we think there is a 60%chance that George is a high spender. We'll now use our threshold, which is indicated by the black dotted line in the chart above, to decide whether we will predict that he is a high spender or not.

Our threshold is 50%, so since our point is above that line, we'll predict that George is a high spender. For this use case, a 50%threshold makes sense, but that's not always the case. For example, in the case of credit card fraud, a bank might only want to predict that a transaction is fraudulent if they're, say, 95%sure, so they don't annoy their customers by frequently declining valid transactions.

2.17 Convex Surrogate Loss Functions

Q19. What is surrogate loss in machine learning?

Ans :

In general, the loss function that we care about cannot be optimized efficiently. For example, the 0-1 loss function is discontinuous. So, we consider another loss function that will make our life easier, which we call the surrogate loss function.

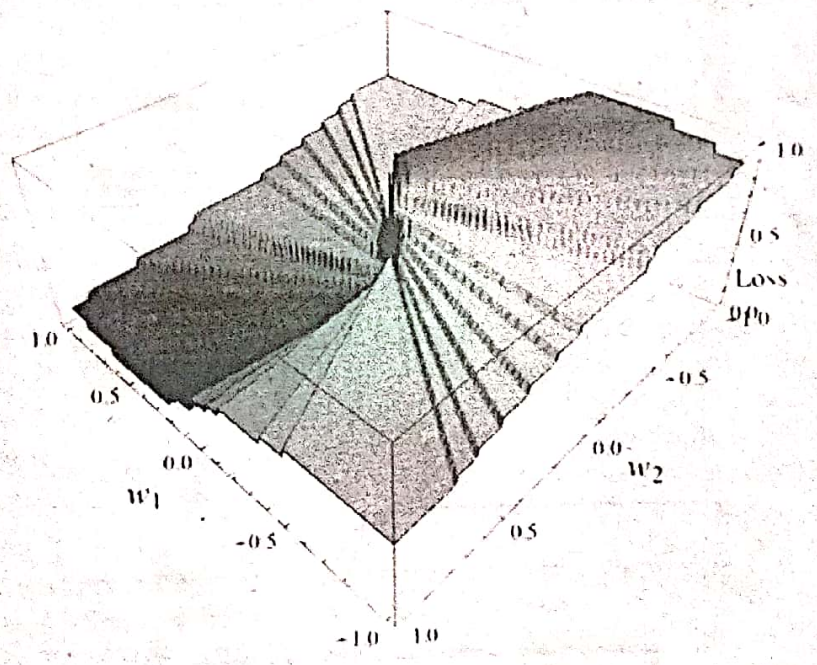
In the binary-class classification setting we are given n training samples {(X1,Y1),...,(Xn,Yn)}, where Xi belongs to some sample space X, usually Rp but for the purpose of this post we can keep i abstract, and yi"{"1,1} is an integer representing the class label.

We are also given a loss function –!:{–1,1}×{–1,1}'!R that measures the error of a given prediction. The value of the loss function –! at an arbitrary point (y,y^) is interpreted as the cost incurred by predicting y^ when the true label is y. In classification this function is often the zero-one loss, that is, –!(y,y^) is zero when y=y^ and one otherwise.

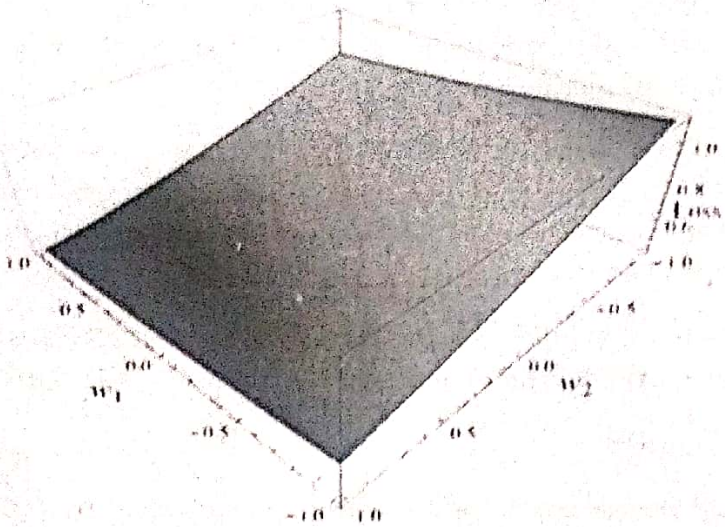
The goal is to find a function h:X'![k], the classifier, with the smallest expected loss on a new sample. In other words, we seek to find a function h that minimizes the expected –!-risk, given by

$$R_l(h)=E_{X\times Y}[l(Y,h(X))]$$

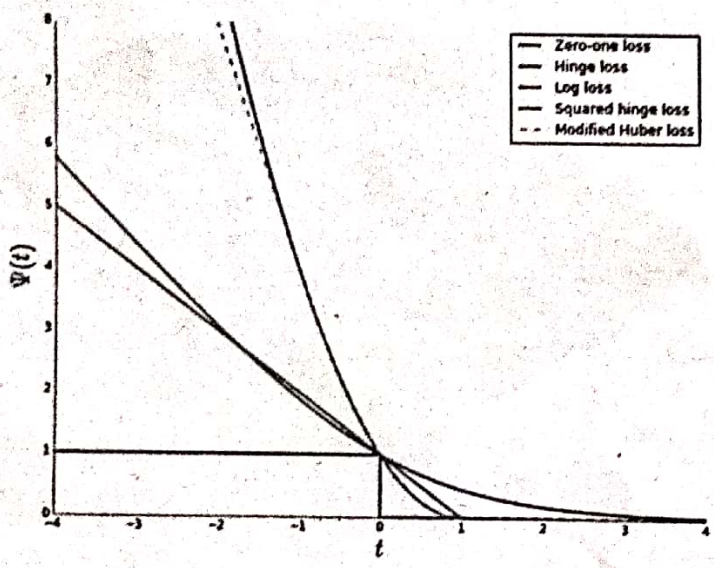
In theory, we could directly minimize the –!-risk and we would have the optimal classifier, also known as Bayes predictor. However, there are several problems associated with this approach. One is that the probability distribution of X×Y is unknown, thus computing the exact expected value is not feasible. It must be approximated by the empirical risk. Another issue is that this quantity is difficult to optimize because the function –! is discontinuous. Take for example a problem in which X=R2,k=2, and we seek to find the linear function f(X)=sign(Xw), ∈R2 and that minimizes the –!-risk. As a function of the parameter w this function looks something like



This function is discontinuous with large, flat regions and is thus extremely hard to optimize using gradient-based methods. For this reason it is usual to consider a proxy to the loss called a surrogate loss function. For computational reasons this is usually convex function ψR → R + . An example of such surrogate loss functions is the hinge loss, ψ(t)=max(1–t,0), which is the loss used by Support Vector Machines SVMs. Another example is the logistic loss, ψ(t)=1/(1+exp(1 – tt)), used by the logistic regression model. If we consider the logistic loss, minimizing the ψ-risk, given by $E_{X\times Y}[\psi(Y,f(X))]$, of the function f(X)=Xw becomes a much more more tractable optimization problem:



In short, we have replaced the –*l*-risk which is computationally difficult to optimize with the Ø-risk which has more advantageous properties. A natural questions to ask is how much have we lost by this change. The property of whether minimizing the ψ-risk leads to a function that also minimizes the –*l*-risk is often referred to as consistency or calibration. For a more formal definition see [1] and [2]. This property will depend on the surrogate function ψ: for some functions ψit will be verified the consistency property and for some not. One of the most useful characterizations was given in [1] and states that if ψ is convex then it is consistent if and only if it is differentiable at zero and ψ2 (0)<0. This includes most of the commonly used surrogate loss functions, including hinge, logistic regression and Huber loss functions



2.18 Weight Regularization

Q20. What is weight regularization in machine learning?

Ans :

Regularization is the technique in which slight modifications are made to learning algorithm such that the model generalizes better. This in turn results in the improvement of the model's performance on the test data or unseen data. In weight regularization, It penalizes the weight matrices of nodes.

Weight regularization results in simpler linear network and slight underfitting of training data. Optimization of the value of regularization coefficient is done in order to obtain a well-fitted model. Weight regularization helps in reducing underfitting in model and thus making model a robust and improving accuracy

Explanation of Weight Regularization.

In L1 weight regularization, the sum of squared values of the weights are used to calculate size of the weights. L1 regularization encourages weights to 0.0 thereby resulting in more sparse weights that is weights with more 0.0 values. L1 regularization sometimes calculate the size of the weight as penalty.

In L2 weight regularization, the sum of absolute values of the weights is used to calculate size of the weights. L2 regularization offers more nuance that is both penalizing larger weights more severely and also resulting in less sparse weights. L2 regularization use in linear regression and logistic regression is often referred as Ridge Regression or Tikhonov regularization. L2 regularization can sometimes calculate the size of the weight as penalty. The L2 regularization approach is the most used and traditionally referred to as "weight decay" in the neural networks field. It is also called as "shrinkage" in statistics that given a name that encourages to think of the impact of the penalty on model weights during learning process. L2 regularization is preferred over L1 regularization as in L1 regularization, the weights may be reduced to zero.

2.19 Optimization and Gradient Descent

Q21. Define Gradient Descent algorithm?

Ans :

Gradient descent (GD) is an iterative first-order optimization algorithm used to find a local minimum/maximum of a given function. This method is commonly used in machine learning (ML) and deep learning (DL) to minimise a cost/loss function (e.g. in a linear regression).

Which is gradient descent technique for solving optimization problem?

A: Gradient descent is a simple optimization procedure that you can use with many machine learning algorithms. Batch gradient descent refers to calculating the derivative from all training data before calculating an update.

Batch Gradient Descent for Machine Learning:

The goal of all supervised machine learning algorithms is to best estimate a target function (f) that maps input data (X) onto output variables (Y). This describes all classification and regression problems.

Some machine learning algorithms have coefficients that characterize the algorithms estimate for the target function (f). Different algorithms have different representations and different coefficients, but many of them require a process of optimization to find the set of coefficients that result in the best estimate of the target function.

Common examples of algorithms with coefficients that can be optimized using gradient descent are Linear Regression and Logistic Regression.

The evaluation of how close a fit a machine learning model estimates the target function can be calculated a number of different ways, often specific to the machine learning algorithm. The cost function involves evaluating the coefficients in the machine learning model by calculating a prediction for the model for each training instance in the dataset and comparing the predictions to the actual output values and calculating a sum or average error (such as the Sum of Squared Residuals or SSR in the case of linear regression).

From the cost function a derivative can be calculated for each coefficient so that it can be updated using exactly the update equation described above.

The cost is calculated for a machine learning algorithm over the entire training dataset for each iteration of the gradient descent algorithm. One iteration of the algorithm is called one batch and this form of gradient descent is referred to as batch gradient descent.

Batch gradient descent is the most common form of gradient descent described in machine learning.

Tips for Gradient Descent:

This section lists some tips and tricks for getting the most out of the gradient descent algorithm for machine learning.

Plot Cost versus Time: Collect and plot the cost values calculated by the algorithm each iteration. The expectation for a well performing gradient descent run is a decrease in cost each iteration. If it does not decrease, try reducing your learning rate.

Learning Rate: The learning rate value is a small real value such as 0.1, 0.001 or 0.0001. Try different values for your problem and see which works best.

Rescale Inputs: The algorithm will reach the minimum cost faster if the shape of the cost function is not skewed and distorted. You can achieved this by rescaling all of the input variables (X) to the same range, such as [0, 1] or [-1, 1].

Few Passes: Stochastic gradient descent often does not need more than 1-to-10 passes through the training dataset to converge on good or good enough coefficients.

Plot Mean Cost: The updates for each training dataset instance can result in a noisy plot of cost over time when using stochastic gradient descent. Taking the average over 10, 100, or 1000 updates can give you a better idea of the learning trend for the algorithm.

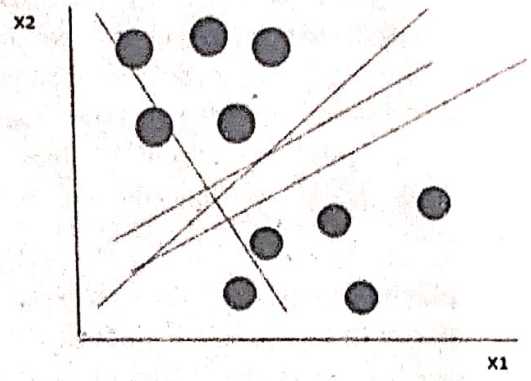
2.20 Support Vector Machines

Q22. Explain about Support Vector Machine Algorithm

Ans: **(Imp.)**

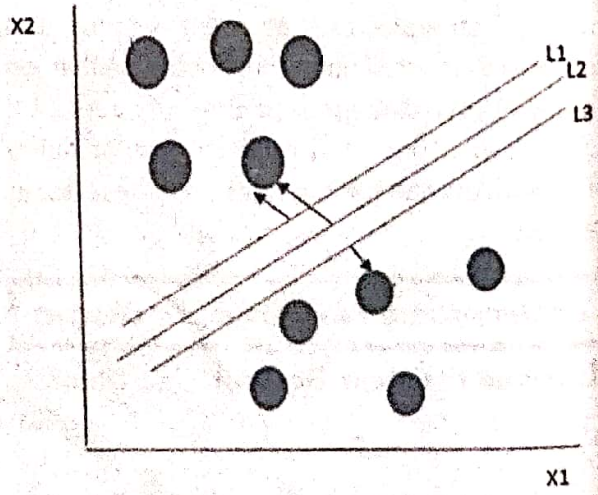
Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Let's consider two independent variables x1, x2 and one dependent variable which is either a blue circle or a red circle.



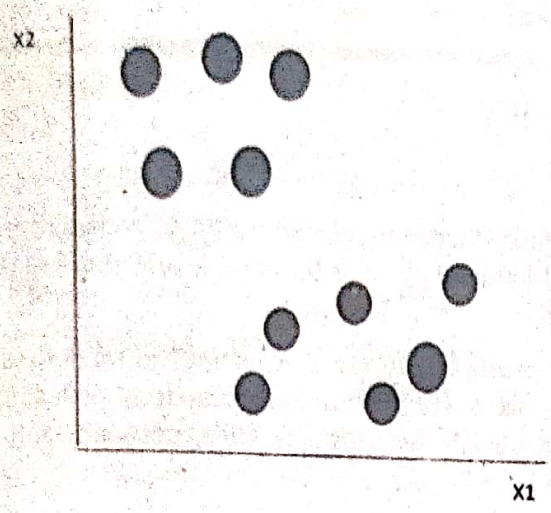
Linearly Separable Data points

From the figure above its very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregates our data points or does a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points.

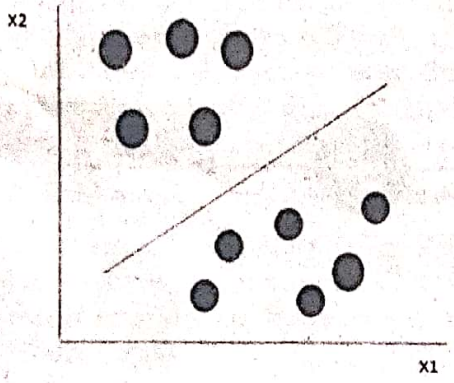


So,we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin. So, from the above figure, we choose L2.

Let's consider a scenario like shown below

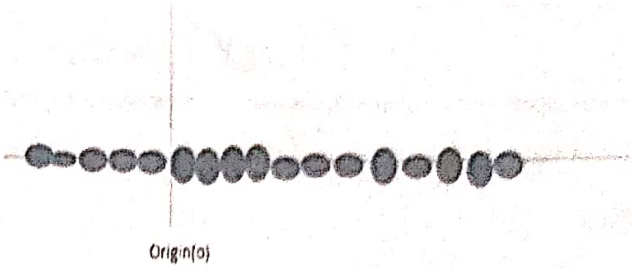


Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

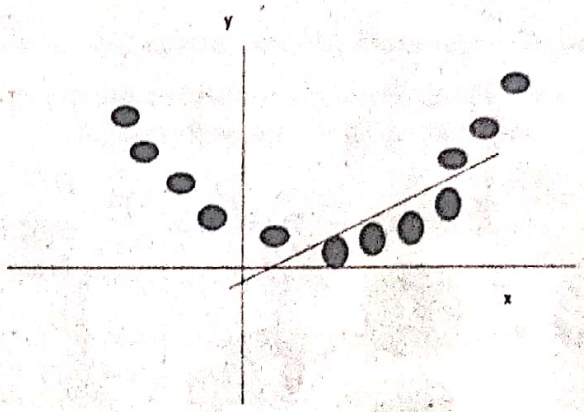


So in this type of data points what SVM does is, it finds maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So the margins in these type of cases are called soft margin. When there is a soft margin to the data set, the SVM tries to minimize (1/margin+'"("penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss.If violations hinge loss proportional to the distance of violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



Say, our data is like shown in the figure above.SVM solves this by creating a new variable using a kernel. We call a point xi on the line and we create a new variable yi as a function of distance from origin o.so if we plot this we get something like as shown below



In this case, the new variable y is created as a function of distance from the origin. A non-linear function that creates a new variable is referred to as kernel.

SVM Kernel:

The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space, ie it converts not separable problem to separable problem. It is mostly useful in non-linear separation problems. Simply put the kernel, it does some extremely complex data transformations then finds out the process to separate the data based on the labels or outputs defined.

Advantages of SVM:

- Effective in high dimensional cases
- Its memory efficient as it uses a subset of training points in the decision function called support vectors
- Different kernel functions can be specified for the decision functions and its possible to specify custom kernels

Short Question and Answers

1. What is the Perceptron model in Machine Learning?

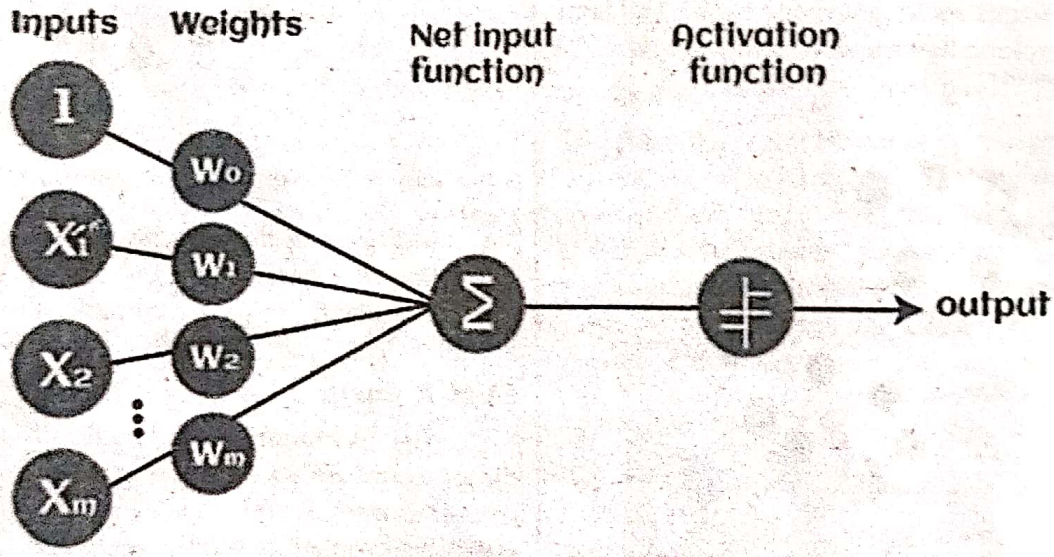
Ans:

Perceptron is Machine Learning algorithm for supervised learning of various binary classification tasks. Further, Perceptron is also understood as an Artificial Neuron or neural network unit that helps to detect certain input data computations in business intelligence.

Perceptron model is also treated as one of the best and simplest types of Artificial Neural networks. However, it is a supervised learning algorithm of binary classifiers. Hence, we can consider it as a single-layer neural network with four main parameters, i.e., input values, weights and Bias, net sum, and an activation function.

Basic Components of Perceptron:

Mr. Frank Rosenblatt invented the perceptron model as a binary classifier which contains three main components. These are as follows:



- **Input Nodes or Input Layer**

  This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

- **Wight and Bias**

  Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

- **Activation Function**

  These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

2. Write about types of perception models.

Ans:

Based on the layers, Perceptron models are divided into two types. These are as follows:

1. Single-layer Perceptron Model
2. Multi-layer Perceptron model

Types

1. Single Layer Perceptron Model

This is one of the easiest Artificial neural networks (ANN) types. A single-layered perceptron model consists feed-forward network and also includes a threshold transfer function inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes.

In a single layer perceptron model, its algorithms do not contain recorded data, so it begins with inconstantly allocated input for weight parameters. Further, it sums up all inputs (weight). After adding all inputs, if the total sum of all inputs is more than a pre-determined value, the model gets activated and shows the output value as +1.

2. Multi-Layered Perceptron Model

Like a single-layer perceptron model, a multi-layer perceptron model also has the same model structure but has a greater number of hidden layers.

The multi-layer perceptron model is also known as the Backpropagation algorithm, which executes in two stages as follows:

- **Forward Stage**

Activation functions start from the input layer in the forward stage and terminate on the output layer.

- **Backward Stage**

In the backward stage, weight and bias values are modified as per the model's requirement. In this stage, the error between actual output and demanded originated backward on the output layer and ended on the input layer.

3. What are the advantages & disadvantages of perception model?

Ans:

- A multi-layered perceptron model can be used to solve complex non-linear problems.
- It works well with both small and large input data.
- It helps us to obtain quick predictions after the training.
- It helps to obtain the same accuracy ratio with large as well as small data.

Disadvantages

- In Multi-layer perceptron, computations are difficult and time-consuming.
- In multi-layer Perceptron, it is difficult to predict how much the dependent variable affects each independent variable.
- The model functioning depends on the quality of the training.

4. What is Bio-inspired Learning?

Ans:

Bio-inspired computing is a research method aimed at solving problems using computer models based on the principles of biology and the natural world. Commonly seen as a philosophical approach, bio-inspired computing is used in a number of related fields of study within computing, rather than a field of study itself. Bio-inspired computing is an extension of the related field of biomimicry.

Bio-inspired computing puts less focus on optimized, high-speed algorithms and more focus on tractability and dependability. Generally, the approach is ground-up, rather than taking a large foundation of knowledge and adding artificial intelligence to it. Bio-inspired computing often takes a small foundation of set rules and builds upon them by way of unsupervised deep learning in training.

In cases where a problem has no clear solution, a useful change of perspective can make for a new chance at solutions. A useful question to ask might be "does this problem have any parallels in nature?" If the answer is "yes" then that parallel can be studied to similarly find parallels in solutions. Humanity has found this philosophy to be useful in solving numerous problems.

5. Explain about Interpreting Perceptron Weights?

Ans:

You may find yourself having run the perceptron, learned a really awesome classifier, and then wondering "what the heck is this classifier doing?" You might ask this question because you're curious to learn something about the underlying data. You might ask this question because you want to make sure that the perceptron is learning "the right thing." You might ask this question because you want to remove a bunch of features that aren't very useful because they're expensive to compute or take a lot of storage.

The perceptron learns a classifier of the form $x \rightarrow$ sign (Σ wdxd + b).

A reasonable question to ask is: how sensitive is the final classification to small changes in some particular feature. We can answer this question by taking a derivative. If we arbitrarily take the 7th feature we can compute $\frac{\partial}{\partial}(\Sigma_d w_d x_d + b) = w_7$. This says: the rate at which the activation changes as a function of the 7th feature is exactly w_7. This gives rise to a useful heuristic for interpreting percep- tron weights: sort all the weights from largest (positive) to largest (negative), and take the top ten and bottom ten. The top ten are the features that the perceptron is most sensitive to for making positive predictions.The bottom tenare the features thattheperceptronis most sensitivet of or making negative predictions.

This heuristic is useful,especially when the inputsxconsis tentirely of binary values(like a bag of wordsreprese ntation).The heuristic is less useful when the range of the individual features varies signific antly. The issue is that if you have one feat x_5t hat's either 0or1, and another feature $x_7$that's either0or100.but$w_5 = w_7$, it's reasonable to say that w_7 is more important because it is likely to have a much larger influence on the final prediction. The easiest way to compensate for this is simply to scale your features a head of time this is another reason why features caling is a useful preprocessing step.

6. What is perceptron convergence?

Ans :

Perceptron Convergence theorem states that a classifier for two linearly separable classes of patterns is always trainable in a finite number of training steps. In summary, the training of a single discrete perceptron two class classifier requires a change of weights if and only if a misclassification occurs.

Perceptron Convergence Theorem

In the classification of linearly separable patterns belonging to two classes only, the training task for the classifier was to find the weight w such that.

Completion of training with the fixed correction training rule for any initial weight vector and any correction increment constant leads to the following weights:

w"=wk0=wk0+1=wk0+2.....

with w" as the solution vector for equation.

Integer k0 is the training step number starting at which no more misclassification occurs, and thus no right adjustments take place for (k_0>=0)

This theorem is called as the "Perceptron Convergence Theorem".

Perceptron Convergence theorem states that a classifier for two linearly separable classes of patterns is always trainable in a finite number of training steps.

In summary, the training of a single discrete perceptron two class classifier requires a change of weights if and only if a misclassification occurs.

In the reason for misclassification is (w ^ tx<0) then all weights are increased in proportion wo xi . If \(w ^ tx>0) then all weights are decreased in proportion to xi

Summary of the Perceptron Convergence Algorithm:

Variables and Parameters: x(n)=(m+1) by 1 input vector=[+1,x1(n),x2(n),.....xm(n)]T

w(n)=(m+1) by 1 weight vector=[b(n), w1(n),w2(n),.....wm(n)]T

b(n)= bias

y(n)= actual response

d(n)= desired response

η= learning rate parameter, a +ve constant less than unity

1. **Initialization**

   Set w(0)=0 , then perform the following computations for time step n=1,2

2. **Activation**

   At time step n, activate the perceptron by applying input vector x(n) and desired response d(n).

3. **Computation of actual response**

   Compute the actual response of the perceptron:

   y(n)=sgn[wT(x)x(n)]

4. **Adaptation of weight vector**

   Update the weight vector of the perceptron:

   w(n+1)=w(n)+ç[d(n)"y(n)]x(n)

5. Continuation: Increment time step n by 1, go to step 1

7. What is normalization in machine learning?

Ans:

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

Normalization in Machine Learning:

"Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale".

Although Normalization is no mandate for all datasets available in machine learning, it is used whenever the attributes of the dataset have different ranges. It helps to enhance the performance and reliability of a machine learning model. In this article, we will discuss in brief various Normalization techniques in machine learning, why it is used, examples of normalization in an ML model, and much more. So, let's start with the definition of Normalization in Machine Learning.

8. What are the Normalization techniques in Machine Learning.

Ans:

Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

Min-Max Scaling

This technique is also referred to as scaling. As we have already discussed above, the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.

Standardization scaling

Standardization scaling is also known as Z-score normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.

Hence, standardization can be expressed as follows:

Here, μ represents the mean of feature value, and ó represents the standard deviation of feature values.

However, unlike Min-Max scaling technique, feature values are not restricted to a specific range in the standardization technique.

This technique is helpful for various machine learning algorithms that use distance measures such as KNN, K-means clustering, and Principal

component analysis, etc. Further, it is also important that the model is built on assumptions and data is normally distributed.

9. Difference between Normalization and Standardization

Ans:

Normalization	Standardization
This technique uses minimum and max values for scaling of model.	This technique uses mean and standard deviation for scaling of model.
It is helpful when features are of different scales.	It is helpful when the mean of a variable is set to 0 and the standard deviation is set to 1.
Scales values ranges between [0, 1] or [-1, 1].	Scale values are not restricted to a specific range.
It got affected by outliers.	It is comparatively less affected by outliers.
Scikit-Learn provides a transformer called MinMaxScaler for Normalization.	Scikit-Learn provides a transformer called StandardScaler for Normalization.
It is also called Scaling normalization.	It is known as Z-score normalization.
It is useful when feature distribution is unknown.	It is useful when feature distribution is normal.

10. What is the significance of using statistical test in machine learning?

Ans:

Statistical test is a way to determine whether the random variable is following the null hypothesis or alternate hypothesis. It basically tells whether the sample and population or two/ more samples have significant differences.

Statistical Hypothesis Tests:

Generally, a statistical hypothesis test for comparing samples quantifies how likely it is to observe two data samples given the assumption that the samples have the same distribution.

The assumption of a statistical test is called the null hypothesis and we can calculate statistical measures and interpret them in order to decide whether or not to accept or reject the null hypothesis.

In the case of selecting models based on their estimated skill, we are interested to know whether there is a real or statistically significant difference between the two models.

If the result of the test suggests that there is insufficient evidence to reject the null hypothesis, then any observed difference in model skill is likely due to statistical chance.

If the result of the test suggests that there is sufficient evidence to reject the null hypothesis, then any observed difference in model skill is likely due to a difference in the models.

The results of the test are probabilistic, meaning, it is possible to correctly interpret the result and for the result to be wrong with a type I or type II error. Briefly, a false positive or false negative finding.

Comparing machine learning models via statistical significance tests imposes some expectations that in turn will impact the types of statistical tests that can be used; for example:Skill Estimate. A specific measure of model skill must be chosen. This could be classification accuracy (a proportion) or mean absolute error (summary statistic) which will limit the type of tests that can be used.Repeated Estimates. A sample of skill scores is required in order to calculate statistics. The repeated training and testing of a given model on the same or different data will impact the type of test that can be used.

Distribution of Estimates. The sample of skill score estimates will have a distribution, perhaps Gaussian or perhaps not. This will determine whether parametric or nonparametric tests can be used.

Central Tendency. Model skill will often be described and compared using a summary statistic such as a mean or median, depending on the distribution of skill scores. The test may or may not take this directly into account.

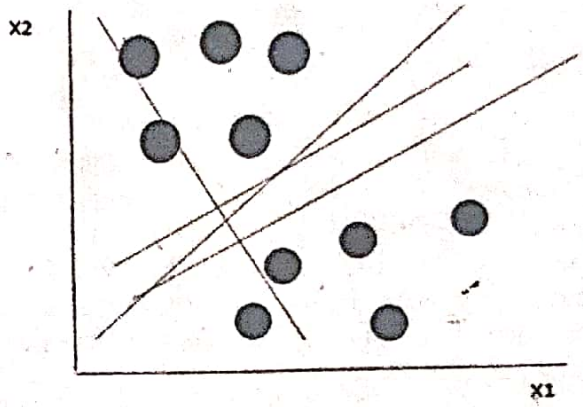
The results of a statistical test are often a test statistic and a p-value, both of which can be interpreted and used in the presentation of the results in order to quantify the level of confidence or significance in the difference between models. This allows stronger claims to be made as part of model selection than not using statistical hypothesis tests.

11. Explain about Support Vector Machine Algorithm

Ans:

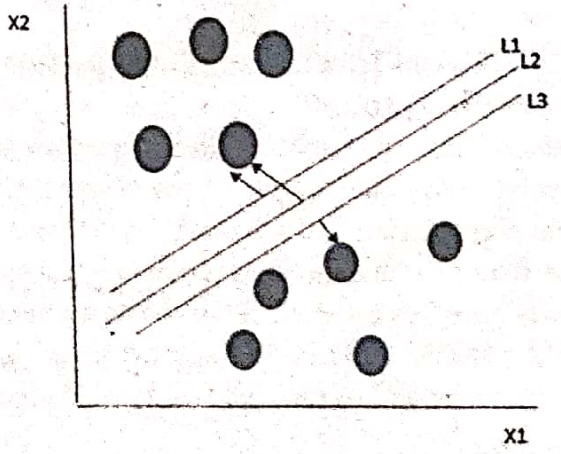
Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

Let's consider two independent variables x1, x2 and one dependent variable which is either a blue circle or a red circle.



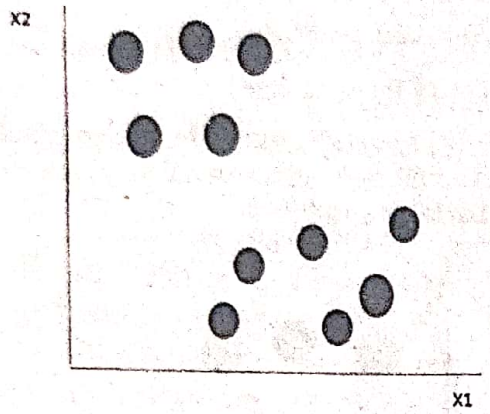
Linearly Separable Data points

From the figure above its very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregates our data points or does a classification between red and blue circles. So how do we choose the best line or in general the best hyperplane that segregates our data points.

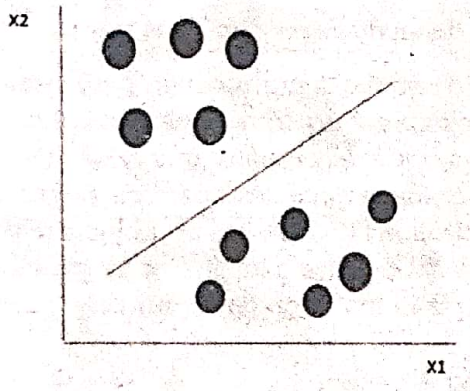


So,we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin. So, from the above figure, we choose L2.

Let's consider a scenario like shown below

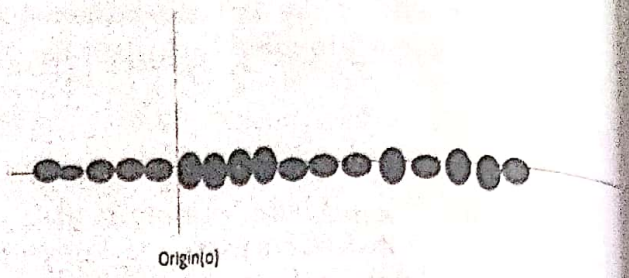


Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It's simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.

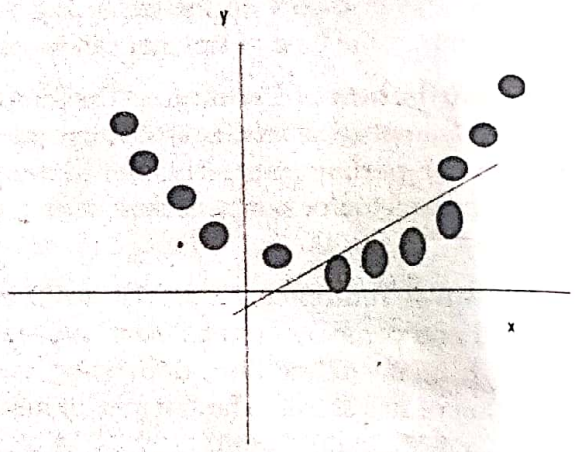


So in this type of data points what SVM does is, it finds maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So the margins in these type of cases are called soft margin. When there is a soft margin to the data set, the SVM tries to minimize (1/margin+'"("penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss.If violations hinge loss proportional to the distance of violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



Say, our data is like shown in the figure above.SVM solves this by creating a new variable using a kernel. We call a point xi on the line and we create a new variable yi as a function of distance from origin o.so if we plot this we get something like as shown below



In this case, the new variable y is created as a function of distance from the origin. A non-linear function that creates a new variable is referred to as kernel.