Linear Regression:

Regression is a method of modelling a target value based on independent predictors. Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.

- Minimize the cost function

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

Minimization and Cost Function

  ○

- Mean squared Error Function, minimize the value of MSE. The idea is that we start with some values for a_0 and a_1 and then we change these values iteratively to reduce the cost. Gradient descent helps us on how to change the values.

Logistic Regression:

- Generalized linear model (GLM) for binary classification problems
- Apply the sigmoid function to the output of linear models, squeezing the target to range [0, 1]
- Threshold to make prediction: usually if the output > .5, prediction 1; otherwise prediction 0
- A special case of softmax function, which deals with multi-class problems

KNN

KNN algorithm is a supervised learning algorithm, based on the fact data points belonging to the same class will likely to occur in close proximity. It captures the idea of similarity. Euclidean distance is used to calculate the distance between two points.

Algorithm:

1. Load the data
2. Initialize value of K, which is equal to choosing K nearest data points to calculate the distance.
3. Take all the data points in the dataset and find the distance between to test point and all the other points.
4. Sort all the points according to the distance with the test point and select K nearest points among those.
5. In these K points, select the category having majority of data points which is also the category of selected point.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric= 'minkowski', p = 2)
classifier.fit(X_train, Y_train)
```

As the volume of data increases, KNN model difficult to train, as calculating distance between each data points takes lot of time.

SVM

SVM is a supervised learning algorithm used for classification and regression problems. SVM tries to create a hyperplane which best segregates the two classes in case of classification problems. The hyperplane created has the largest margin in high dimensional space. The objective is to find a plane that has the maximum margin i.e. the maximum distance between data points of both classes. Maximising the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. *Support vector* are the data points that are closer to the hyperplane and influence the position and orientation of the plane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is **hinge loss.**

```
from sklearn.svm import SVC

Clf = SVC(kernel = ‘linear’)

Clf.fit(x_train, y_train)
```

A kernel is a similarity function. The function of kernel take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. Ex: linear, non linear, polynomial, radial basis function(RBF), and sigmoid.

$$k(x, y) = \exp\left(-\frac{||x - y||^2}{2\sigma^2}\right)$$

Gaussian kernel equation

C = 1/lambda

Large C: Lower bias, high variance (small lambda)

Small C: Higher bias, low variance   (high lambda)

Large sigma^2 : Higher bias, low variance, more smooth variation

Small sigma^2 : Lower bias, high variance, less smooth variation

Types of kernel: Polynomial kernel, Chi-square kernel, histogram intersection

Let n = number of features, m = number of training examples

- If n is large, (relative to m): Use logistic regression or Linear SVM
- If n is small, m is intermediate : Use Gaussian Kernel
- If n is small, m is large: Use Linear kernel, or logistic regression.

Naïve Bayes: -Bayes Theorem: determining conditional probability.

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}$$

Bayes' theorem relies on incorporating prior probability distributions in order to generate posterior probabilities. *Prior Probability P(A)* is the probability of an event before the new data point is collected. *Posterior probability P(A/B)* is the revised probability of an event occurring after taking into consideration new information. The probability that the word 'FREE' is used in previous spam message is likelihood P(B/A). Marginal likelihood P(B) is, the probability that the word 'FREE' is used in any message.

Let 2 classes be Spam and Non spam, and to classify a new data point X between these two classes, compare the two values P(Spam/X) and P(Non-Spam/X) and classify the data point accordingly.

Why Naïve: This implies the absolute independence of features - a condition probably never met in real life.

Applications: text classification, spam filtering, sentiment analysis, recommendation systems, etc.

Gaussian Naïve Bayes: continuous values associated with each feature are assumed to be distributive according to a Gaussian distribution (Normal distribution).

Decision Tree:

Decision trees are built based on available dataset. The dataset is splits the nodes based on available input variables. Given a data point to be classified, walk through the tree built using training dataset.

These are prone to over-fitting. To overcome the disadvantages of decision tree, random forests are used.

Random Forest:

It is an ensemble of randomized decision trees. Each decision tree gives a vote for the prediction of target variable. Random forest choses the prediction that gets the most vote. An ensemble learning model aggregates multiple machine learning models to give a better performance. In random forest we use multiple random decision trees for a better accuracy. It reduces the variance of the individual decision trees by randomly selecting trees and then either average them or picking the class that gets the most vote. High predictive accuracy. Efficiency on large datasets.

Ensemble Models:

A collection of predictors which come together (e.g. Mean of all predictions) to give a final prediction. The reason we use ensembles is that many different predictors trying to predict same target variable will perform a better job than any single predictor alone. Ensembling techniques are further classified into Bagging and Boosting.

Bagging:

A simple ensembling technique in which we build many independent predictors/models/learners and combine them using some model averaging techniques. (E.g. Weighted average, majority vote, normal average). Example of bagging ensemble is Random Forest models.

Boosting:

It's an ensembling technique in which the predictors are not made independently but sequentially. The technique works on the principle that subsequent predictors learn from the mistakes of the previous predictors. Because new predictors are learning from mistakes committed by previous predictors, it takes less time/iterations to reach close to actual predictions. But we have to choose the stopping criteria carefully or it could lead to overfitting on training data. Example is boosting is gradient boosting.

Difference from random forest (RF)

- RF grows trees in parallel, while Boosting is sequential.
- RF reduces variance, while Boosting reduces errors by reducing bias.

Multi Layer Perceptron:

A feedforward neural network of multiple layers. In each layer we can have multiple neurons, and each of the neuron in the next layer is a linear/nonlinear combination of the all the neurons in the previous layer. In order to train the network we back propagate the errors layer by layer. In theory MLP can approximate any functions.

Curse of Dimensionality:

Deep Learning:

Deep learning refers to the multi-layered neural networks which are used to learn levels of representation and abstraction that makes sense of data such as images and text. There are hidden layers between input and output made up of neurons (basic unit of a neural network) similar to brain. These hidden layers are not designed by humans, they are learned from data using a general purpose learning procedure. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features.

Simple neural networks have only 1-2 layers in between input and output. Whereas in deep learning there are more layers between input and output (hence “deep”), which allows richer intermediate representations to be built.

Neurons:

Neurons are the basic unit of a neural network. In nature, neurons have a number of dendrites (inputs), a cell nucleus (processor), and an axon (output). A neuron is a mathematical function that model the functioning of a biological neuron.

Weights:

A weight represent the strength of the connection between units. If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2. A weight brings down the importance of the input value. Weights near zero means changing this input will not change the output. Negative weights mean increasing this input will decrease the output. A weight decides how much influence the input will have on the output. Each neuron is associated with weights. Lower value of weight of a neuron indicates the input of this neuron will have an output.

Bias:

Bias is like the intercept added in a linear equation. It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron. Therefore, Bias is a constant which helps the model in a way that it can fit best for the given data. Bias is like the intercept added in a linear equation. It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron. Therefore Bias is a constant which helps the model in a way that it can fit best for the given data. Bias helps in controlling the value at which activation function will trigger.

Input Layer:

This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information (features) to the hidden layer.

Hidden Layer:

Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.

Output Layer:

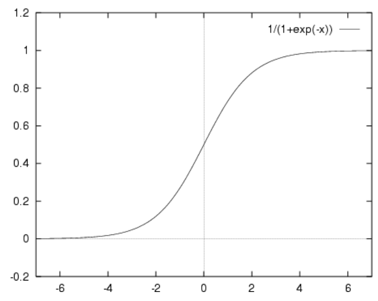
This layer bring up the information learned by the network to the outer world.

Activation function:

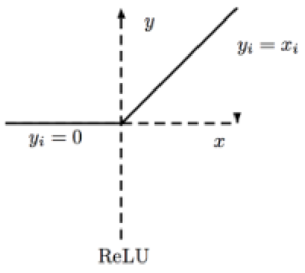
The purpose of activation function is to introduce non-linearity into the output of a neuron. Introducing non-linearity is necessary because in real-world most of the data is not linearly dependent. Without activation function is just a linear regression model. Activation function does the non-linear transformation to the input making it capable to learn more complex tasks to generate non-linear mappings from inputs to outputs.

Commonly used activation functions:

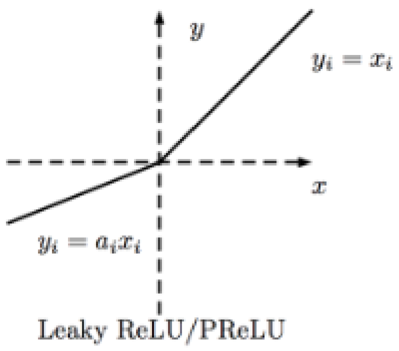
- Sigmoid:
    - g(x) = 1/(1+exp(-x)). Range between (0, 1).
    - Generally used for binary classification



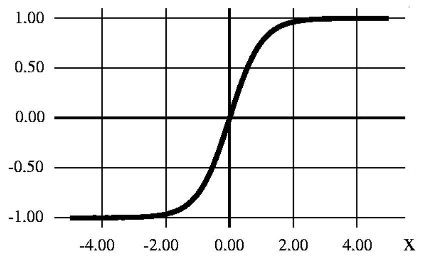
- ReLu: g(x) = max(0, x). Simple and efficient. It avoids and rectifies vanishing gradient problem. Only used with hidden layers. ReLu can result in dead neurons.



- Leaky ReLu: max(0.1x, x).



- Hyperbolic tanh(x): g(x) = (1-exp(-2x))/(1+exp(-2x)). Range(-1, 1)



Backpropogation:

Backpropogation is a method used to train neural networks meaning "back propagating the errors" by calculating the gradients of loss function with respect to learnable parameters Weights (W) and bias (B) at each level in artificial neural networks. It is used to optimize the performance by changing the weights. It is used to obtain the optimized values of weights and biases such that error is minimum.

Initialize weights and biases with some random values. Use *forward pass* to calculate the cost function. To reduce the error in the network, find gradients of cost function with respect to weights and using *backward pass,* propagate these values to previous layers and modify the initialized value accordingly. Follow the steps to obtain optimized cost function.

Gradient Descent:

Optimization Algorithm used to find the values of parameters of a function that minimizes a cost function.

- Initializing weights and biases find the cost function.
- To minimize this cost function, calculate the derivate of the cost function with respect to these weights and biases. Update the weights and biases value by multiplying the derivative with alpha. (learning rate)
    - coefficient = coefficient – (alpha * delta)

Alpha learning rate must be specified that controls how much the coefficient can change on each update. If alpha large, may diverge and never converge to minimum. Similarly, if alpha is small it'll take lot of iterations to converge. Therefore, optimal is required.

Batch (Vanilla) Gradient Descent and Stochastic Gradient Function:

In gradient descent algorithm, update of set of parameters is required to minimize the error function. In Batch GD, for a single update you have to run through all the training samples. Whereas in Stochastic GD, single update requires only one training sample. Therefore, if training samples are very large then Batch GD will be very costly. On the other hand, SGD will be faster and will converge faster. In Stochastic GD, each time we encounter a training example, we update the parameters. Whereas Batch GD, before taking one step of updation, it has to scan all the training examples. Although Stochastic is faster, but the error function is not well minimized as in the case of GD. Often in most cases, the close approximation that you get in SGD for the parameter values are enough because they reach the optimal values and keep oscillating there.

Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in GD, but may go "zig-zag" if we are visualizing the cost surface in a 2D space. However, it has been shown that SGD almost surely converges to the global cost minimum if the cost function is convex (or pseudo-convex).

Mini-batch Gradient descent:

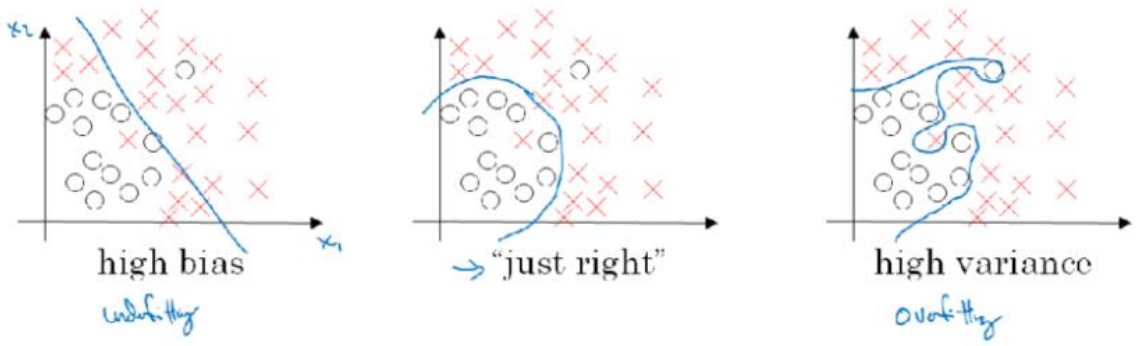
A common approach to addressing large scale data is mini-batch GD, wherein gradients are computed over batches of training data. The reason this works well is that the examples in the training data are correlated. Extreme case of this is when mini-batch contains exactly one sample. This is relatively less common to see because in practice due to vectorized code optimizations it can be computationally much more efficient to evaluate the gradient for 100 examples, than the gradient for one example 100 times. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2. So mini-batches of 32, 64 or 128 is used.

Optimization Algorithms

Problem with convergence of SGD is around the areas where the surface curves more steeply in one direction than other which are common around local minimum. To help with this problem, other optimization algorithms are generally used which accelerate SGD in one direction and dampen oscillations. Important optimization algorithms include:

- SGD with momentum
- RMS Prop
- Adam (most commonly used)

Bias VS Variance:



First diagram depicts the case of high bias. High bias, a situation where in model pays very little attention to the training data and over simplifies the model (Underfitting). It always leads to high error on training data and test data. Bias are the simplifying assumptions made by a model to make the target function easier to learn. So high bias means the model assumes more assumptions about the form of the target function.

Third diagram in the sequence depicts the case of High Variance where in model pays a lot of attention to the training data and tries to predict a function such that it is accurate for almost all the values of training data. Such a function fails to perform well on unseen data and thus has high error rates on test data. Variance is the amount of the estimate of the target function will change if different training data was used. So high variance means as the dataset changes, there will be high variations in the model's performance.

Example cases:

- High Variance Case: Train set error: 1%, Dev set error: 11%
- High Bias Case: Train set error: 15%, Dev set error: 16%
- High Bias and High Variance: Train set error: 15%, Dev set error: 30%
- Low Bias and Low Variance(DESIRABLE): Train set error: 0.5%, Dev set error: 1%

General trend is Parametric or linear machine learning algorithms often have a high bias but a low variance and Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.

In ML algorithms, there is always a trade-off between the two but in deep-learning, this trade-off is not very common.

If High bias, more hidden layers, more hidden units, train longer, try optimization algorithms.

If High Variance, More data, regularization.

Overfitting:

One of the most common problems in Machine learning (or in Data Science) is Overfitting. A situation when the model performs exceptionally well on training data but fails to predict on test data i.e. the case of high variance. Commonly used techniques in machine learning are cross-validation, early stopping, pruning, and regularization.

Regularization:

To address the problem of high variance, one of the measure is to get more training data. But in some cases, it may be expensive. So the other measure Regularization is often used in machine learning problems.

$$\min_{w,b} J(w,b) \qquad \underline{w \in \mathbb{R}^{n_x}}, \ \underline{b \in \mathbb{R}}$$

$\lambda =$ regularization parameter

lambda $\quad$ lambd

$$J(w,b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})} + \frac{\boxed{\lambda}}{2m}\underbrace{\|w\|_2^2} \ \cancel{+ \frac{\lambda}{2m} b^2}$$

omit

L2 regularization $\quad \underline{\|w\|_2^2} = \sum_{j=1}^{n_x} w_j^2 = w^T w \leftarrow$

L1 regularization $\quad \frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m}\|w\|_1$

w will be sparse

Add a regularization parameter lambda to the cost function. This parameter penalizes or shrinks the learned parameters of the network. As the value of lambda rises, it reduces the value of coefficients and thus reducing the parameters. Till a point, this increase in lambda is beneficial as it is only reducing the variance (hence avoiding overfitting), without loosing any important properties in the data. But after a certain value, model starts loosing its properties and thus underfitting. Therefore, value of lambda should be carefully used.

Why penalize Weights and not Bias?

Often, W (Weights) are penalized and not B (biases) because W is usually high dimensional parameter vector which is the main reason of high variance and B is just a real number. So, most of the parameters are in W and not in b.

The two commonly used techniques of regularization are L1 (lasso) and L2 (ridge) regularization.

L1 Regularization (Lasso):

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

Also known as Least absolute deviations (LAD). It is minimizing the sum of the absolute differences between the target value and the estimated values.

L2 Regularization (Ridge):

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Also known as least squares error (LSE). It is minimizing the sum of the squares of the difference between target value and the estimated values.

The key difference is that lasso shrinks the less important feature’s coefficients to zero thus, removing some features altogether. So, this works for features selection when there are a large number of features and thus yields sparse models. However, L2 norm shrinks the coefficients for least important predictors very close to zero. But it will never make them exactly zero. In other words, the final model will include all predictors.

Weight Decay:

Dropout:

Dropout is a regularization technique. In this technique, some neurons are randomly removed (or shut down) in each iteration, i.e. these neurons are not used in both forward and backward direction in computation of cost function and gradients. In every pass (iteration) some probability is set with each neuron in each layer, and according to that some of the neurons are eliminated. Thus, much smaller network is trained on every example. By avoiding to use every neuron in every iteration, it shrinks the weights related to every neuron and thus eliminating the cause of over-fitting by shrinking the weights.

The value of keep_prob = 0.8 in a hidden layer means that on each iteration, each unit has 80% probability of being included and 20% probability of being dropped out. The lower the value of dropout, higher the bias and lower the variance.

Early Stopping:

Early stopping is also a regularization technique which uses number of training epochs as a hyperparameter and aims to find for the iteration where the neural network is doing best and stop trading on neural network halfway and take whatever value is achieved on dev set error.

Batch Normalization: Why it works? :

CNN basics:

- **Translation invariation.**

RNN Basics:

LSTM Basics:

GRU Basics:

Supervised Learning:

Supervised learning is where you have input variables and an output variable(Y) and you use an algorithm to learn the mapping function from the input to the output. Y = f(x). The goal is to approximate the mapping function well so that for new input x, output Y can be predicted using the predicted mapping function. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process.

Unsupervised Learning:

Unsupervised learning is where only input data x is present and no corresponding labels are available. The goal of unsupervised learning algorithms is to model the underlying structure or distribution in the data in order to learn more about the data. Unsupervised because no teacher or no correct labels to teach the model.

- Clustering: The data is divided into several groups. Aim is to segregate groups with similar traits and assign them into clusters. Example: K-means Clustering
- Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y. Example: Apriori algorithm for association rule learning problem.
- Clustering, dimensionality reduction, feature learning without any external labels, underlying density estimation.

Semi-Supervised Learning:

Problems having a large amount of data (X) and only some of the data is labelled (Y) is called semi-supervised learning. Example: Photo-archive where only some of the images are labelled and majority is unlabelled. Unlabelled data is cheap and easily available, most of the ml problems fall into this area.

K-means Clustering:

Objective of K-means clustering is to group data on the basis of similarity and discover underlying patterns. To achieve this objective, K-means try to figure out K-clusters in the data. The 'means' in the K-means refers to averaging of the data i.e. finding the centroid.

Algorithm:

```
- Choose the number of clusters (K) and obtain the data points.
- Place the centroids c_1, c_2, ….. c_k randomly.
- Repeat the following:
- (Cluster assignment step) For each data point x_i:
    o Find the nearest centroid (c_1, c_2, c_3, …., c_k)
    o Assign the data point to that cluster centroid c_i := min|| x_i – mean(k) ||
      closest to x_i
- (Move Centroid Step) For each cluster j = 1 .. k
    o New_centroid = mean of all points assigned to that cluster
    o Move the centroid to that new_centroid point.
- End
```

Time complexity is O(n). Noisy data is not allowed.

Elbow Method: To find out appropriate value of number of clusters (K) in the dataset, plot a graph between cost function(J) and different values of K.

Applications: Market Segmentation, Social Network Analysis, Anomaly Detection, Behavioural Segmentation.

Dimensionality Reduction:

Reducing the p dimensions of the data into a subset of k dimensions (k << n). This is dimensionality reduction.

When the size of the data increases, i.e. it may contain million features, it is better to select a subset of features or variables as it will reduce the disk space used by the data and also speeds up the learning algorithms. Also, keeping correlated features in the dataset is redundant. For example keeping height in cm as well as in inches doesn't make a difference. Memory required will be more. There is no point in storing both as just one of them does what you require. Some algorithms do not perform well when we have large dimensions as they can't extract enough important information from large dataset. So reducing these dimensions need to happen for the algorithm to be useful. Also, helps in visualization of data in lower dimensions helps us to plot the data and observe the patters more clearly.

- Feature selection (removing some of the features).
- Finding smaller set of new variables each being a combination of the input variables containing basically the same information as the input variables.

PCA:

It is one of the dimensionality reduction techniques. Helps in extracting a set of variables from an existing large set of variables. These newly extracted features are called principle components. What PCA does is, it tries to find a surface to project the data to minimise the projected area. Reduce from n-dimension to k-dimension: Find k vectors u(1), u(2), … , u(k) onto which to project the data so as to minimize the projection error.

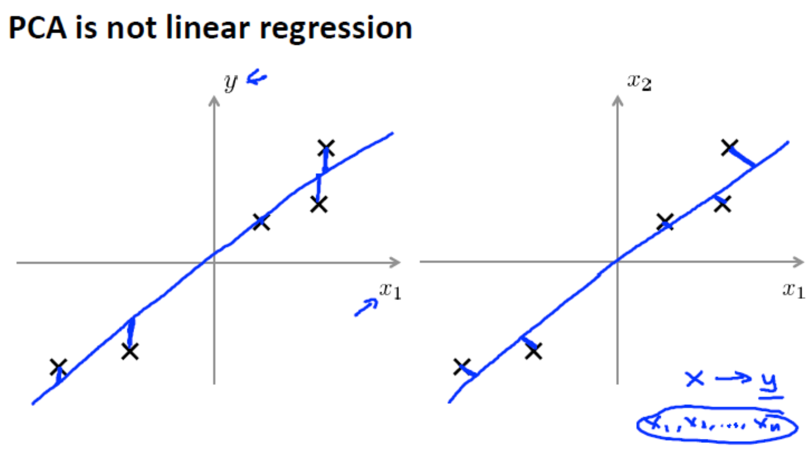
PCA (Principal Component Analysis) helps in producing low dimensional representation of the dataset by identifying a set of linear combination of features which have maximum variance and are mutually un-correlated.

Principle component is a linear combination of the original variables. Principle components are extracted in such a way that the first principle component explain the maximum variance in the dataset. Second principal component tries to explain the remaining variance in the dataset and is uncorrelated to the first principal component. Third principal component tries to explain the variance which is not explained by the first two principal components and so on.

PCA is not linear regression. Linear regression minimises the vertical distance between input and output whereas in PCA it minimises the square of projection error.

Data pre-processing is required i.e. feature scaling and mean normalization.

- Compute the covariance matrix S = 1/N *(X^T * X)
- Compute K largest Eigen vectors of S. These Eigen vectors are the principal components of the data set.



SVD (Singular Value Decomposition):

Singular Value decomposition is used to get rid of the redundant features present in our data. It decomposes the matrix into three constituent matrices. This is called Matrix factorization or matrix decomposition. It is essentially used to remove redundant features from the dataset. It uses the concept of Eigenvalues and Eigenvectors to determine those matrices.

A = U . Sigma . V^T , A = real matrix of m*n, U = m*n (Left Singular Vectors), Sigma = diagonal matrix of r*r (Singular Values), V^T = transpose matrix of V of r*k (Right singular vectors). N =

Every rectangular matrix has a singular value decomposition, although the resulting matrices may contain complex numbers and the limitations of floating point arithmetic may cause some matrices to fail to decompose neatly. Diagonal values in the Sigma matrix are known as the singular values of the original matrix A. Root of Eigen values are called singular values. SVD can also be used in least squares linear regression, image compression, and denoising data.

U, V are column orthonormal i.e. columns have Euclidean length of 1 so sum of squared values in each column is 1. Columns are orthogonal unit vectors i.e. multiply two columns of U and V you get zero, Sigma is diagonal and entries are positive and sorted in decreasing order. U is User-to-concept similarity matrix. Sigma gives strength of each concept. V is movie-to-concept matrix. SVD gives 'best' axis to project on. Best = min sum of squares of projection errors or minimum reconstruction error. Zero out the lower singular values of Sigma and the columns of U and rows of V^T. The obtained matrix is very similar to the original matrix. Can be quantified using Frobenius norm.

It turns out that the variance of each principal component is related to square (diagonal elements of d matrix). The image below shows how I reduce the number of dimensions from k to q (k<q). If you reduce the number of column vectors to q, then you have obtained the q-dimensional hyper-plane in this example. The values of D gives you the amount of variance retained by this reduction. The lower diagonal values can be zero out because they

explain the minimum variance in the dataset. Choose the values of k such that 99% of variance is retained.

t-Stochastic Neighbor Embedding (t-SNE) :

t-SNE a non-linear dimensionality reduction algorithm finds patterns in the data by identifying observed clusters based on similarity of data points with multiple features. But it is not a clustering algorithm, it is a dimensionality reduction algorithm. This is because it maps the multi-dimensional data to a lower dimensional space, the input features are no longer identifiable. Thus you cannot make any inference based only on the output of t-SNE. So essentially it is mainly a data exploration and visualization technique. But t-SNE can be used in the process of classification and clustering by using its output as the input feature for other classification algorithms.

Pre-processing → normalization → t-SNE→ classification algorithm (AdaBoostM2, Random Forests, Logistic Regression, NNs and others as multi-classifier for the expression classification).

Word vector representations capture many linguistic properties such as gender, tense, plurality and even semantic concepts like “capital city of”. Using dimensionality reduction, a 2D map can be computed where semantically similar words are close to each other. This combination of techniques can be used to provide a bird’s-eye view of different text sources, including text summaries and their source material. This enables users to explore a text source like a geographical map.

Contrary to PCA, t-SNE is not a mathematical technique but a probabilistic one. t-Distributed stochastic neighbour embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

PCA is deterministic and t-SNE is non-deterministic.

Generative Models:

- Given training data, generate new samples from same distribution.
- Want to learn generated sampled P_model(x) such that training data P_data(x) is similar.
- Addresses density estimation, core problem in unsupervised learning.
  - Explicit density estimation: explicitly define and solve for Pmodel(x)
  - Implicit density estimation: learn model that can sample from Pmodel(x) via explicitly defining it.
- Realistic samples for artwork, super-resolution, colorization, etc.
- Generative models of time series data can be used for simulation and planning.
- Modeling the distribution of natural images.
- Training generative models can also enable inference of latent representations that can be useful as general features.
- Important generative models include:

- PixelRNN / PixelCNN
- Variational Autoencoder
- Markov Chain (Boltzmann Machine)
- Generative Adversarial Networks

PixelRNN:

Use probabilistic density models (like Gaussian or Normal distribution) to quantify the pixels of an image as a product of conditional distributions. Turns the modelling problem into sequence problem wherein the next pixel value is determined by all the previously generated pixel values. Generate image pixels starting from corner.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Drawback: sequential generation is slow!

PixelCNN:

To overcome the problem of slow training, PixelCNN uses standard convolutional layers to capture a bounded receptive field and compute features for all pixels positions at once. Multiple convolutional layers are used that preserve the spatial resolution. However, pooling layers are not used. Masks are adopted in the convolutions to restrict the model from violating the conditional dependence. Training is faster but generation must still proceed sequentially. PixelCNN define tractable density function, optimize likelihood of training data.

Auto-Encoder:

- The aim of an autoencoder is to learn a representation (encoding) for a set of data.

- An autoencoder always consists of two parts, the encoder and the decoder. The encoder would find a lower dimension representation (latent variable) of the original input, while the decoder is used to reconstruct from the lower-dimension vector such that the distance between the original and reconstruction is minimized.

- Can be used for data denoising and dimensionality reduction and can learn features to initialize a supervised model.

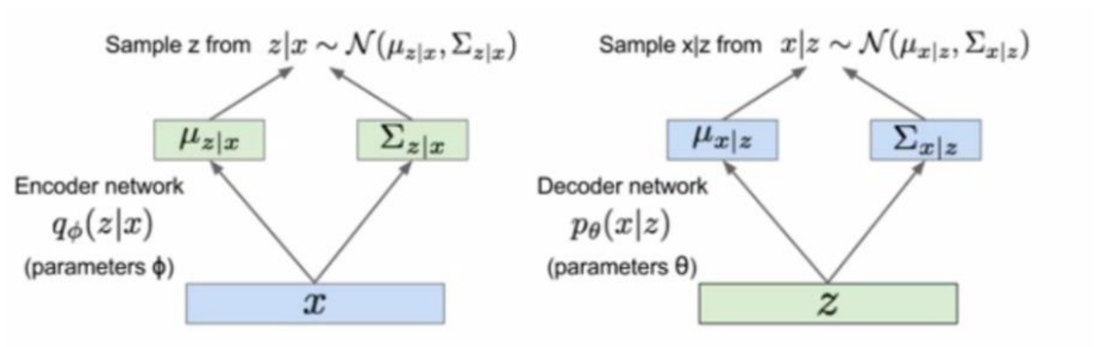
- Loss function where $g_\emptyset$ Encoder function and $f_\theta$is the Decoder function. It is mean-squared error or cross-entropy loss between the output and the input also known as reconstruction loss. Encoders learn to preserve and extract the useful information from the data and discard the irrelevant parts.

$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

Variational Auto Encoders (VAEs):

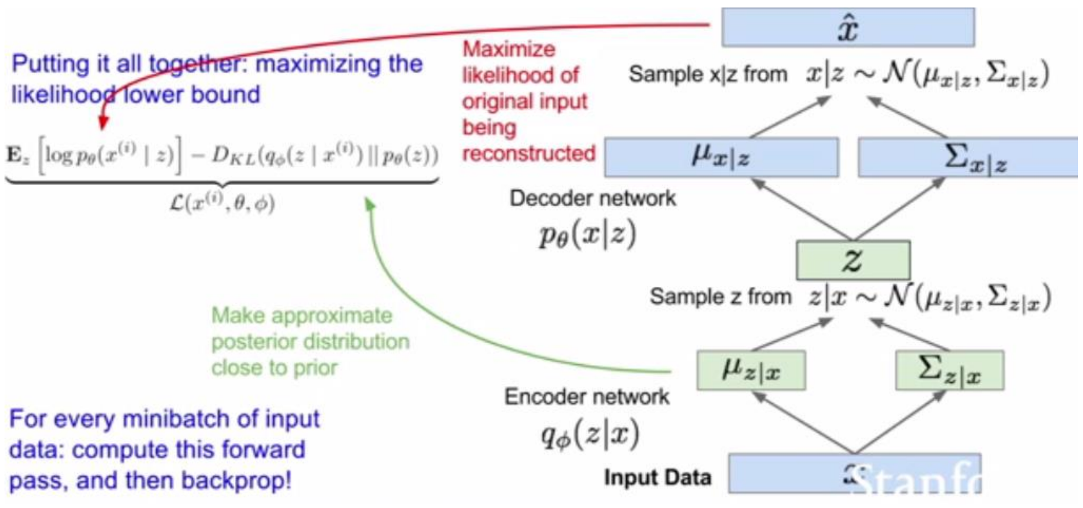
Instead of just learning a function representing the data (a compressed representation) like autoencoders, variational autoencoders learn the parameters of a probability distribution representing the data. Since it learns to model the data, we can sample from the distribution and generate new input data samples. So it is a generative model like GANs.

Variational autoencoders (VAEs) have a unique property that their latent spaces are continuous which allows easy random sampling and interpolation which helps them to generate new samples. In VAEs, probabilistic generation of data is modelled and therefore its encoder and decoder networks are probabilistic. Its encoder outputs two vectors of size n: a vector of means and another vector of standard deviation.



Encoder and decoder networks are called “recognition” and “generation” networks. The loss of VAEs is to minimize both the reconstruction loss (how similar the autoencoder’s output was to its input) and its latent loss or KL divergence term (how much the two distributions diverge from each other).

KL divergence terms are used to tell how much the two distributions diverge from each other. Note: this is not distance metric. Minimizing KL divergence here means optimizing the probability distribution parameters (mean, μ and standard deviation, σ) to closely resemble that of the target distribution. First term forces to reconstruct the input data and KL divergence makes approximate posterior distribution close to prior.


Putting it all together: maximizing the likelihood lower bound
$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$
Maximize likelihood of original input being reconstructed
$\hat{x}$
Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$
$\mu_{x|z}$
$\Sigma_{x|z}$
Decoder network
$p_\theta(x|z)$
z
Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$
Make approximate posterior distribution close to prior
$\mu_{z|x}$
$\Sigma_{z|x}$
Encoder network
$q_\phi(z|x)$
For every minibatch of input data: compute this forward pass, and then backprop!
Input Data
x

Generative Adversarial Networks:

Generative Models are models used for unsupervised learning, wherein only unlabelled data is available for training. In such cases, the goal is to learn some underlying hidden structure of the data. These models are powerful way to learn true data distributions of the available training examples so to generate new samples from the same distribution. It addresses the problem of density estimation in unsupervised learning.

Generative Adversarial Networks are one of these generative models which was first pioneered by Ian Goodfellow. The paper presents a novel technique to generate realistic images similar to existing images. It's basically a 2-player minimax game approach where Generator's (first player) task is to try to fool the Discriminator (other player) by generating fake images and Discriminator's task is to distinguish and estimate between the real images of available dataset and counterfeiting images generated by generator. These networks act as opponents against each other as both tries to minimize their own cost function to reach a global minimal point. These networks are different from most of the deep learning models which try to minimize single cost function. The objective of training jointly these two networks is that generator is forced to generate real looking images similar to training data by learning the data distribution of real images. More realistic images are produced as a result of training the two networks. Both the networks are deep neural networks.

Random noise (z) is fed as an input to the Generator G(z) which outputs an image. Let it be represented by X'. Now, this generated image G(x) is fed to Discriminator D which outputs the value D(G(z)) and an image(X) from training dataset is also fed as an input to Discriminator represented by D(x) which outputs a scalar value describing how close is the generator's output to the real image. Discriminator is trained so as to maximize objective such that D(x) is 1, as input from dataset is real and D(G(z)) is 0 which signifies the image from data is fake. Generator's objective is to minimise such that D(G(z)) is 1 so as to fool the discriminator that data produced by it is real. The cost function J(D) of discriminator is given by the equation(1). Gradient ascent is performed on discriminator.

$$(D) = -12\ (x \sim Pdata)\ \log(x)\ -12\ E(z)\log(1-D(G(z))\ (1)$$

Whereas generator minimizes the log probability of discriminator being mistaken given by the cost function: (G) = $-(D)$. But this equation may saturate if G is poor. So, to avoid this, G is trained to maximize the following equation: (G) =-12 Elog($G(z)$)

It has been shown in the paper that the global optimal value for the two cost functions is: (x) = $Pda(x)Pdata(x)$+$Pmodel(x)$. D(x) = ½ when generator starts generating images similar to real images.

Key Issues: Jointly training two networks is challenging and may cause non convergence issue, due to which GANs are unstable.

Applications: Many areas of applications where GANs have shown good performance. Few of them are: Image to Image translation which involves converting sketch of an object to realistic photo of the same, text translation to an image, aerial view to its map view, better generation of images as compared to other generative models for example Variational Autoencoders which produces blurry images, next video frame generation, drug discovery, and many more.

Word2Vec:

Word2Vec models, introduced by Mikolov Et.al are deep learning based models to compute and generate high quality distributed and continuous dense vector representations of words which capture a large number of precise synctactic and semantic similarity.

C-BOW Architecture:

Given the surrounding context words, the model tries to predict the current word. In the implementation in this repo, two context words on each side of current word are used. Input to the model will be [w(t-2), w(t-1), w(t+1), w(t+2)] and output will be w(t). These models are several times faster to train as compared to skip gram model. Gives better accuracy for frequent words.

Skip-gram Architecture:

Given a word, the model tries to predict the context surrounding words. In the implementation of this repo, model tries to figure out two surrounding words on either side of current word. Input to the model will be w(t) and output will be [w(t-2), w(t-1), w(t+1), w(t+2)]. These models work well with small amount of training data, represents well even rare words or phrases.

N-gram:

Cross Validation:

Generative and Discriminative Models:

Generative classifiers learn a model of the joint probability p(x, y), of the input x and the label y, and make their predictions by using Bayes rules to calculate p(y/x) and then picking the most likely label y. It explicitly models the actual distribution of each class. Example: Naïve Bayes

Discriminative classifiers, on the other hand, model the posterior p(y/x) directly or learn a direct map from input x to the class labels. It models the decision boundary between the classes. Since, it directly learns the conditional probability, discriminative classifiers are faster than generative. Example: Logistic Regression.

Sparse Encoding Models:

- Unsupervised learning Only use the inputs x(t) for learning.

  For each x(t) find a latent representation h(t) such that it is sparse: the vector h(t) has many zeros. we can reconstruct the original input x(t) as well as possible.

  fixed (a problem of sparse encoding and known as LASSO Least Angle Shrinkage and Selection Operator) (a problem of least squares method)

layerwise stacking

of feature extraction often yielded better representations, e.g., in terms of classification accuracy.

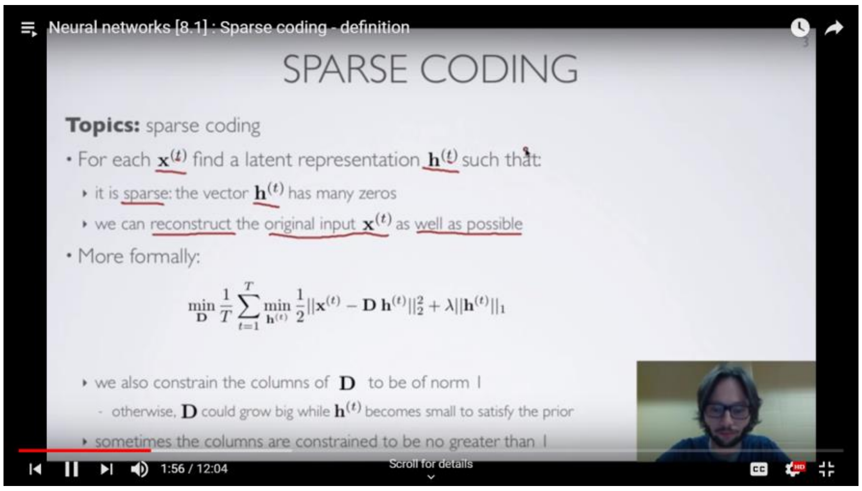
Another advantage of depth is that more abstract features can be generated from less abstract features at deeper levels. The idea comes from the fact that the inputs are transformed to highly non-linear transformations at the deeper levels thus representing more variations.

Constraint the columns of D because if don't put any constraint on D and penalize sparsity, then it would increase the size of elements in D and decrease in h(t).

Thus high-level abstractions in data is modelled by using a deep architecture with multiple processing layers, composed of multiple linear and non-linear transformations.

Dictionary learning: It is an unsupervised learning approach that aims to find a sparse representation of the input data.

$$\underset{D,Z}{min} \|X - DZ\|_F^2$$


Neural networks [8.1] : Sparse coding - definition

SPARSE CODING

Topics: sparse coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can reconstruct the original input $\mathbf{x}^{(t)}$ as well as possible
- More formally:

$$\min_{\mathbf{D}} \frac{1}{T}\sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2}||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda||\mathbf{h}^{(t)}||_1$$

  - we also constrain the columns of $\mathbf{D}$ to be of norm 1
    - otherwise, $\mathbf{D}$ could grow big while $\mathbf{h}^{(t)}$ becomes small to satisfy the prior
  - sometimes the columns are constrained to be no greater than 1

1:56 / 12:04

Scroll for details

D is called dictionary matrix. D*h(t) is called reconstruction. And is close to the original input.

Lambda * h(t) is called l1-norm. lambda term is to control to what extent we are going to get good reconstruction error as compared to achieving high sparsity. Lambda is to control the trade-off between high sparsity vs good reconstruction. Optimize for each training example x(t).

We will constraint D also, otherwise it could penalize h(t) by increasing D and regularization lambda will be good but D will increase. We want to avoid this. D is equivalent to the auto encoder output weight matrix. H(x(t)) is now complicated function of x(t).

Disentangled Representation Learning:

Unsupervised learning technique that breaks down or disentangles each feature into narrowly defined variables and encodes them as separate dimensions. Disentangled representation would mean a single latent unit being sensitive to variations in single generative factors.

Confusion Matrix

-Performance measurement tool in Machine Learning

	Actual Values(1)	(0)
Predictive values (1)	TP	FP
(0)	FN	TN

False Positive: Type 1 error

False Negative: Type 2 error

Recall = (TP)/(TP + FN)

Precision = (TP)/(TP + FP)

F-measure = 2 * R * P/(R + P)

Trade-off between recall and precision. So better to compare them on the basis of F-score.

For Example: In case of 10 apples, prediction made is 10 apples and 5 oranges. So, recall is 100% as out of 10 actual apple, all predicted values are true. Whereas Precision is only 66.7%, as out of 15 predicted values only 10 are correct.

AUC-ROC curve (Area under the curve-Receiver Operating Characteristics)

-In ML, an important tool for measurement.

-To check performance of classification models. Mainly multi-class classification problems can easily be visualized.

-ROC curve is probability curve and AUC represents degree or measure of probability.

-Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s, i.e correct classification.

-ROC curve plotted against TPR(True Posi-tive Rate) and FPR(False Positive Rate)

-TPR/Recall/Sensitivity = (TP)/(TP + FN)

-Specificity = (TN)/(TN + FP)

-FPR = 1 - Specificity = (FP)/(TN + FP)

-Good model : AUC near to 1

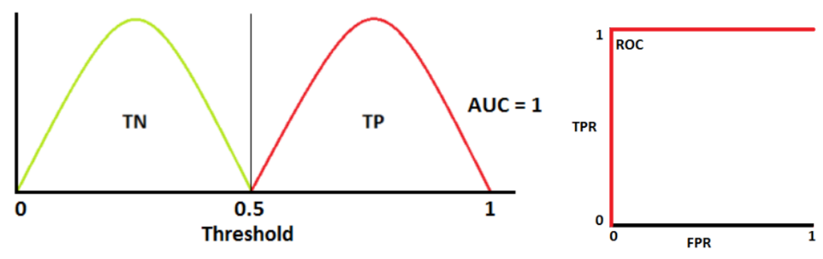
-AUC near to 0 implies reciprocating means it is classifying positive class as negative class and negative class as positive class.

-And when AUC is 0.5, it means model has no class separation capacity whatsoever.

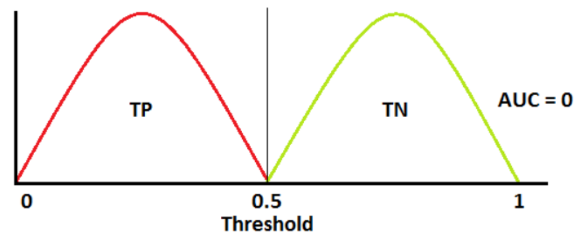
-Sensitivity and Specifity are reciprocal . Decrease threshold, increased the sensitivity and decreases specifity.

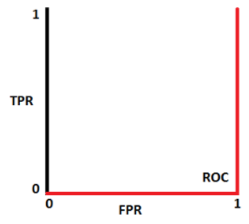
-In multi-class model, we can plot N number of AUC ROC Curves for N number classes using One vs ALL methodology. So for Example, If you have three classes named X, Y and Z, you will have one ROC for X classified against Y and Z, another ROC for Y classified against X and Z, and a third one of Z classified against Y and X.

AUC = 0:

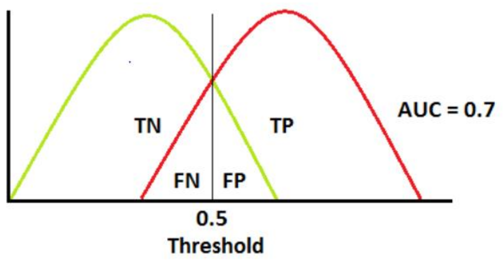


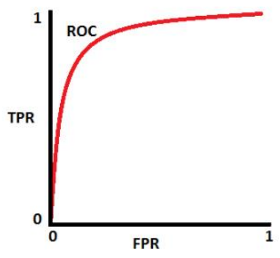
AUC = 1:


TP
TN
AUC = 0
0
0.5
1
Threshold

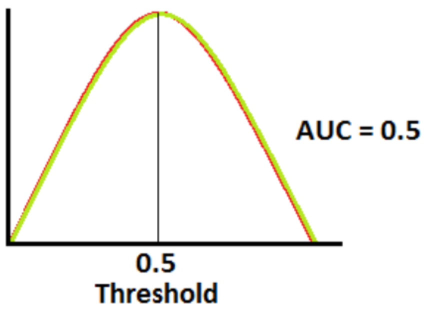

1
TPR
0
ROC
0
FPR
1

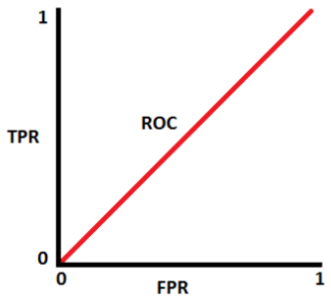
AUC = 0.7:


TN
TP
FN
FP
AUC = 0.7
0.5
Threshold


1
ROC
TPR
0
0
FPR
1

AUC = 0.5:


AUC = 0.5
0.5
Threshold


1
TPR
ROC
0
0
FPR
1