Support Vector Machine: Tanay Sharma
Random Forest : Tushar Nandy
Naive Bayes : Tushar Nandy

Pattern Recognition Algorithms

1. **Random Forests:**
   Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

   Wikipedia uses this for quality assessment of its articles.

2. **Naive Bayes classifiers:**
   It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Some uses are:

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction features. Here we can predict the probability of multiple classes of target variables.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

3. **Support Vector Machine(SVM):**

a. SVM is a kind of supervised learning algorithm which is used in data classification and pattern recognition.
b. Classification problems generally depend on constructing a hyperplane boundary between the data for classification.
c. Support vector machine helps to find the optimal hyperplane for this type of classification. In this we define a parameter "Margin" which is the minimum distance of any training example from the hyperplane. The optimal hyperplane according to the SVM algorithm is the one which maximizes the "Margin" parameter

d. Used Cases:

i. Face Detection: SVM is used to detect the position and location of faces in an image. It considers an image as an nxn pixel matrix and then it classifies each pixel into 2 classes as either 1(if the pixel is a part of the face) or -1(if the pixel isn't a part of the face).

ii. Text Classification: This is multiclass single label classification i.e. here the Text may be classified to multiple classes e.g. News, documents, spams, etc. But a single training example is assumed to get a single label, i.e. any example cannot belong to more than one class. Here SVM is used to generate multiple hyperplanes so as to classify every example into one of the classes.

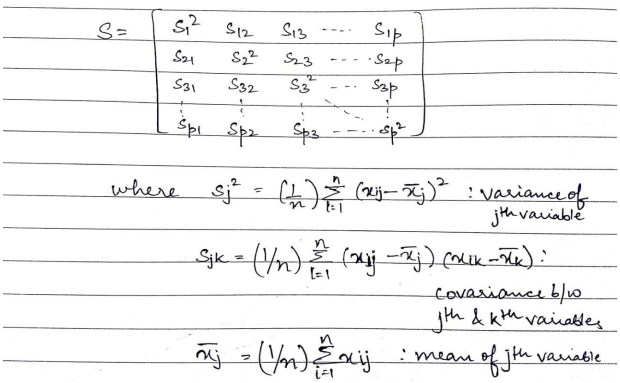
4. Principal Component Analysis:

1. What is it and why do we use this?
   (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.
2. Rough idea about how to use this?
   Assume a dataset with observations on p-numerical variables, for each of n entities or individuals. These data values define pn-dimensional vectors x1,…,xp or, equivalently, an n×p data matrix X, whose jth column is the vector xj of observations on the jth variable.
   a. How do we find new variables?
      We seek a linear combination of the columns of matrix X with maximum variance. Such linear combinations are given by

$\sum_{j=1}^{p} a_j \mathbf{x}_j = \mathbf{Xa}$, where a is a vector of constants a1,a2,…,ap. The most common restriction involves working with unit-norm vectors, i.e. requiring a′a=1.

b. How is variance calculated and how to maximize it?

var(Xa)=a′Sa,where S is the sample covariance matrix associated with the dataset and ′ denotes transpose.



Maximising:

- Hence, identifying the linear combination with maximum variance is equivalent to obtaining a p-dimensional vector a which maximizes the quadratic form a′Sa.
- The problem is equivalent to maximizing a′Sa−λ(a′a−1), where λ is a Lagrange multiplier. Differentiating with respect to the vector a, and equating to the null vector, produces the equation

$$\mathbf{Sa} - \lambda\mathbf{a} = \mathbf{0} \Longleftrightarrow \mathbf{Sa} = \lambda\mathbf{a}.$$

Thus, a must be a (unit-norm) eigenvector, and λ the corresponding eigenvalue, of the covariance matrix S. In particular, we are interested in the largest eigenvalue, λ1 (and corresponding eigenvector a1), since the eigenvalues are the variances of the linear combinations defined by the corresponding eigenvector a: var(Xa)=a′Sa=λa′a=λ.

Conclusion:

A Lagrange multipliers approach, with the added restrictions of orthogonality of different coefficient vectors, can also be used to show that the full set of eigenvectors of S are the solutions to the problem of obtaining up to p new linear combinations

$$\mathbf{Xa}_k = \sum_{j=1}^{p} a_{jk}\mathbf{x}_j$$

It is these linear combinations Xak that are called the principal components of the dataset.

Analyzing PCA

:

5. Self Organizing Maps:

a. This is a type of Artificial Neural Network which is trained in an unsupervised manner.
b. This is used for dimensionality reduction on a high dimensional input space
c. The map provides a discretized representation of the input on a lower(generally two) dimension, while also preserving neighbourhood features thus preserving the input Topology.
d. The algorithm starts with initializing some random vectors in the input dimension. They are called “Weights”. Then for every input vector we find its best matching unit i.e the nearest weight vector to it. We then update the corresponding weight vector and its neighboring weight vectors to shift closer to the training example. This is repeated till the weights become a good enough discretized representation of the input.
e. Used Cases:
    i. Geographic Information: SOM’s can be used to analyse geographic information such as amount of rain, forests, terrain in a visual manner.

6. K-Nearest Neighbors:

What is K-NN ?

1. K-NN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution i.e. the model structure determined from the dataset
2. It is called the Lazy algorithm because it does not need any training data points for model generation. All training data is used in the testing phase which makes training faster and testing phase slower and costlier.
3. K-Nearest Neighbor (K-NN) is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbor
3. For each example in the data
   3.1 Calculate the distance between the query example and the current example from the data.
   3.2 Add the distance and the index of the example to an ordered collection
4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
5. Pick the first K entries from the sorted collection
6. Get the labels of the selected K entries
7. If regression, return the mean of the K labels
8. If classification, return the mode of the K labels

Performance of the K-NN algorithm is influenced by three main factors :

1. The **distance function** or distance metric used to determine the nearest neighbors.

2. The **decision rule used to derive a classification** from the K-nearest neighbors.
3. The **number of neighbors** used to classify the new example.

Advantages of K-NN :

1. The K-NN algorithm is very easy to implement.
2. Nearly optimal in the large sample limit.
3. Uses local information, which can yield highly adaptive behavior.
4. Lends itself very easily to parallel implementation.

Disadvantages of K-NN :

1. Large storage requirements.
2. Computationally intensive recall.
3. Highly susceptible to the curse of dimensionality.

K-NN Algorithm finds its applications in :

1. Finance — financial institutes will predict the credit rating of customers.
2. Healthcare — gene expression.
3. Political Science — classifying potential voters in two classes will vote or won't vote.
4. Handwriting detection.
5. Image Recognition.
6. Video Recognition.
7. Pattern Recognition.

7. Decision Tree Classification Algorithm

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf

  nodes are the output of those decisions and do not contain any further branches.

- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further splits the tree into subtrees.

Why use Decision Trees?

The two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contain possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follows while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the Random Forest algorithm.
- For more class labels, the computational complexity of the decision tree may increase.

8. Independent Component Analysis

What is Independent Component Analysis?

Independent Component Analysis (ICA) is a machine learning technique to separate independent sources from a mixed signal. Unlike principal component analysis which focuses on maximizing the variance of the data points, the independent component analysis focuses on independence, i.e. independent components.

Independent Component Analysis (ICA) Algorithm

At a high level, ICA can be broken down into the following steps.

1. Center **x** by subtracting the mean
2. Whiten **x**
3. Choose a random initial value for the de-mixing matrix **w**
4. Calculate the new value for **w**
5. Normalize **w**
6. Check whether algorithm has converged and if it hasn't, return to step 4
7. Take the dot product of **w** and **x** to get the independent source signals

$$\mathbf{S=wx}$$

Assumptions in Independent Component Analysis.

Independent component analysis only works if the sources are non-Gaussian (i.e. they have non-normal distributions), and so that is one of the first assumptions you make if you use this analysis on a multivariate function. Since you are attempting to break the function down into independent components, you're also making the assumption that the original sources were in fact independent.

Mixing Effects in Independent Component Analysis

There are three principles of mixing signals which make up the foundation for independent component analysis.

1. Mixing signals involves going from a series of independent signals to a series of signals that is dependent. Although your source signals are all independently generated, the composite signals are all made from the same source signals, and so cannot be independent of each other.
2. Although our source signals are non-Gaussian (by the assumption we made to begin with), the composite signals are in fact Gaussian (normally distributed). This is by the *Central Limit Theorem*, which tells us the the *probability distribution function* of the sum of independent variables with finite variance (like our signals) will tend toward the Gaussian distribution.
3. The complexity of a signal mixture must always be greater than, or equal to, the complexity of its simplest source signal.

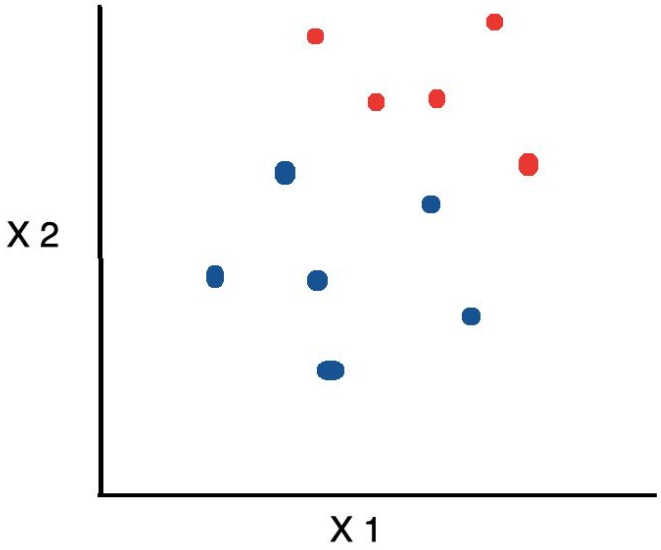
ICA properties

1. **Linear Discriminant Analysis: Abhishek Pai Angle**

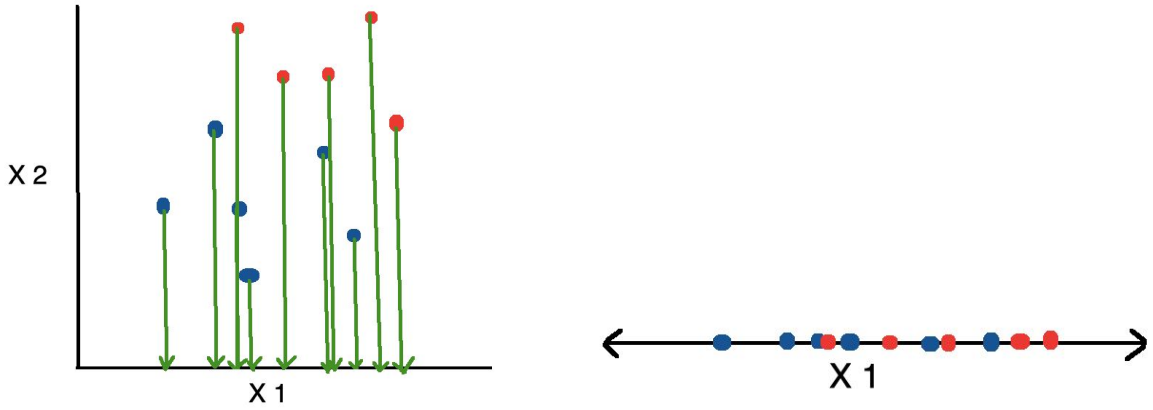
- ICA can only separate linearly mixed sources.
- Since ICA is dealing with clouds of points, changing the order in which the points are plotted (the time points order in EEG) has virtually no effect on the outcome of the algorithm.
- Changing the channel order (for instance swapping electrode locations in EEG) has also no effect on the outcome of the algorithm. For EEG, the algorithm has no a priori about the electrode location and the fact that ICA components can most of the time be resolved to a single equivalent dipole is a proof that ICA is able to isolate compact domains of cortical synchrony.
- Since ICA separates sources by maximizing their non-Gaussianity, perfect Gaussian sources can not be separated
- Even when the sources are not independent, ICA finds a space where they are maximally independent

9.Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique. As the name implies dimensionality reduction techniques reduce the number of dimensions (i.e. variables) in a dataset while retaining as much information as possible.



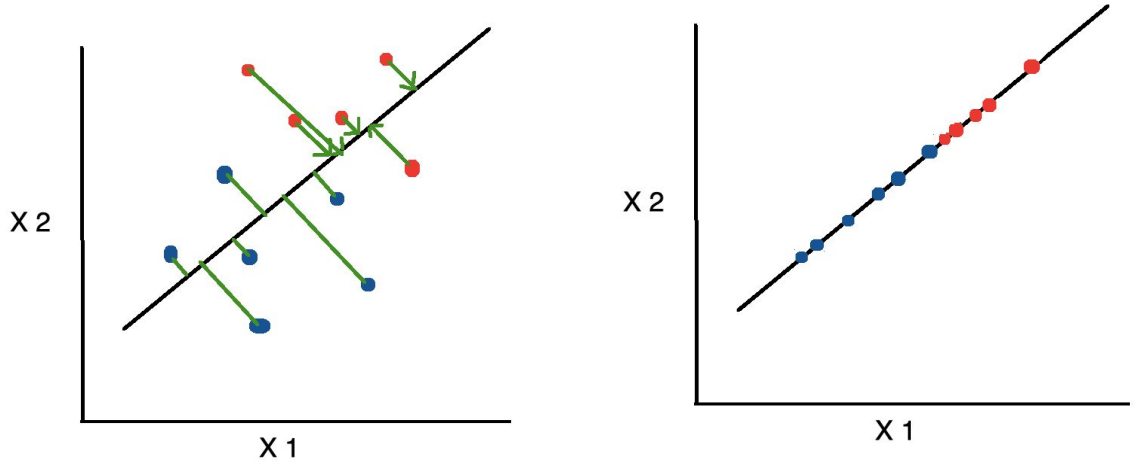
If we'd like to reduce the number of dimensions down to 1, one approach would be to project everything on to the x-axis.



This is bad because it disregards any useful information provided by the second feature.

On the other hand, Linear Discriminant Analysis, or LDA, uses the information from both features to create a new axis and projects the data on to the new axis in such a way

as to minimizes the variance and maximizes the distance between the means of the two classes.



We create a DataFrame containing both the features and classes.

Linear Discriminant Analysis can be broken up into the following steps:

1. Compute the within class and between class scatter matrices
2. Compute the eigenvectors and corresponding eigenvalues for the scatter matrices
3. Sort the eigenvalues and select the top ***k***
4. Create a new matrix containing eigenvectors that map to the ***k*** eigenvalues
5. Obtain the new features (i.e. LDA components) by taking the dot product of the data and the matrix from step 4

Code to directly implement LDA:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda=LDA(n_components=2)

X_train=lda.fit_transform(X_train,y_train)

X_test=lda.transform(X_test)
```

10. Apriori

Apriori algorithm is given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called *Apriori property* which helps by reducing the search space.

Apriori Property –

All non-empty subset of frequent itemset must be frequent. The key concept of Apriori algorithm is its anti-monotonicity of support measure.

Consider the following dataset and we will find frequent itemsets and generate association rules for them.

TID	items
T1	I1, I2 , I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

minimum support count is 2

Step-1: K=1

(I) Create a table containing support count of each item present in dataset – Called **C1(candidate set)**

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

(II) compare candidate set item's support count with minimum support count(here min_support=2 if support_count of candidate set items is less than min_support then remove those items). This gives us itemset L1.

Itemset	sup_count
I1	6
I2	7
I3	6
I4	2
I5	2

Step-2: K=2

- Generate candidate set C2 using L1 (this is called join step). Condition of joining L_{k-1} and L_{k-1} is that it should have (K-2) elements in common.
- Check all subsets of an itemset are frequent or not and if not frequent remove that itemset.(Example subset of{I1, I2} are {I1}, {I2} they are frequent.Check for each itemset)
- Now find support count of these itemsets by searching in dataset.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I4	1
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I3,I4	0
I3,I5	1
I4,I5	0

- (II) compare candidate (C2) support count with minimum support count(here min_support=2 if support_count of candidate set item is less than min_support then remove those items) this gives us itemset L2.

Itemset	sup_count
I1,I2	4
I1,I3	4
I1,I5	2
I2,I3	4
I2,I4	2
I2,I5	2
I2,I5	2

Step-3:

- Generate candidate set C3 using L2 (join step). Condition of joining L_{k-1} and L_{k-1} is that it should have (K-2) elements in common. So here, for L2, first element should match.
  So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}
- Check if all subsets of these itemsets are frequent or not and if not, then remove that itemset.(Here subset of {I1, I2, I3} are {I1, I2},{I2, I3},{I1, I3} which are frequent. For {I2, I3, I4}, subset {I3, I4} is not frequent so remove it. Similarly check for every itemset)
- find support count of these remaining itemset by searching in dataset.

Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

(II) Compare candidate (C3) support count with minimum support count(here min_support=2 if support_count of candidate set item is less than min_support then remove those items) this gives us itemset L3.

Itemset	sup_count
I1,I2,I3	2
I1,I2,I5	2

Step-4:

- Generate candidate set C4 using L3 (join step). Condition of joining L_{k-1} and L_{k-1} (K=4) is that, they should have (K-2) elements in common. So here, for L3, first 2 elements (items) should match.
- Check all subsets of these itemsets are frequent or not (Here itemset formed by joining L3 is {I1, I2, I3, I5} so its subset contains {I1, I3, I5}, which is not frequent). So no itemset in C4
- We stop here because no frequent itemsets are found further

---

Thus, we have discovered all the frequent item-sets. Now generation of strong association rule comes into picture. For that we need to calculate confidence of each rule.

Confidence –

A confidence of 60% means that 60% of the customers, who purchased milk and bread also bought butter.

`Confidence(A->B)=Support_count(A∪B)/Support_count(A)`

So here, by taking an example of any frequent itemset, we will show the rule generation.

Itemset {I1, I2, I3} //from L3

SO rules can be

[I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4*100=50%

[I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/4*100=50%

[I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4*100=50%

[I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/6*100=33%

[I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7*100=28%

[I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/6*100=33%

So if minimum confidence is 50%, then first 3 rules can be considered as strong association rules.

Limitations of Apriori Algorithm

Apriori Algorithm can be slow. The main limitation is time required to hold a vast number of candidate sets with much frequent itemsets, low minimum support or large itemsets i.e. it is not an efficient approach for large number of datasets. For example, if there are 10^4 from frequent 1- itemsets, it need to generate more than 10^7 candidates into 2-length which in turn they will be tested and accumulate. Furthermore, to detect frequent pattern in size 100 i.e. v1, v2… v100, it have to generate 2^100 candidate itemsets that yield on costly and wasting of time of candidate generation. So, it will check for many sets from candidate itemsets, also it will scan database many times repeatedly for finding candidate itemsets. Apriori will be very low and inefficiency when memory capacity is limited with large number of transactions.

11. Upper Confidence Bound(UCB)

In Reinforcement Learning, we use Multi-Armed Bandit Problem to formalize the notion of decision-making under uncertainty using k-armed bandits. A decision-maker or agent is present in Multi-Armed Bandit Problem to choose between k-different actions and receives a reward based on the action it chooses. Bandit problem is used to describe

fundamental concepts in reinforcement learning, such as rewards, timesteps, and values.

Imagine an online advertising trial where an advertiser wants to measure the click-through rate of three different ads for the same product. Whenever a user visits the website, the advertiser displays an ad at random. The advertiser then monitors whether the user clicks on the ad or not. After a while, the advertiser notices that one ad seems to be working better than the others. The advertiser must now decide between sticking with the best-performing ad or continuing with the randomized study.

If the advertiser only displays one ad, then he can no longer collect data on the other two ads. Perhaps one of the other ads is better, it only appears worse due to chance. If the other two ads are worse, then continuing the study can affect the click-through rate adversely. This advertising trial exemplifies decision-making under uncertainty.

In the above example, the role of the agent is played by an advertiser. The advertiser has to choose between three different actions, to display the first, second, or third ad. Each ad is an action. Choosing that ad yields some unknown reward. Finally, the profit of the advertiser after the ad is the reward that the advertiser receives.

Action-Values:

For the advertiser to decide which action is best, we must define the value of taking each action. We define these values using the action-value function using the language of probability. The value of selecting an action $q_*(a)$ is defined as the expected reward R_t we receive when taking an action a from the possible set of actions.

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

The goal of the agent is to maximize the expected reward by selecting the action that has the highest action-value.

Action-value Estimate:

Since the value of selecting an action i.e. $Q_*(a)$ is not known to the agent, so we will use the **sample-average** method to estimate it.

$$Q_t(a) = \frac{sum\ of\ rewards\ when\ (a)\ taken\ prior\ to\ (t)}{number\ of\ times\ (a)\ taken\ prior\ to\ (t)}$$

Exploration vs Exploitation:

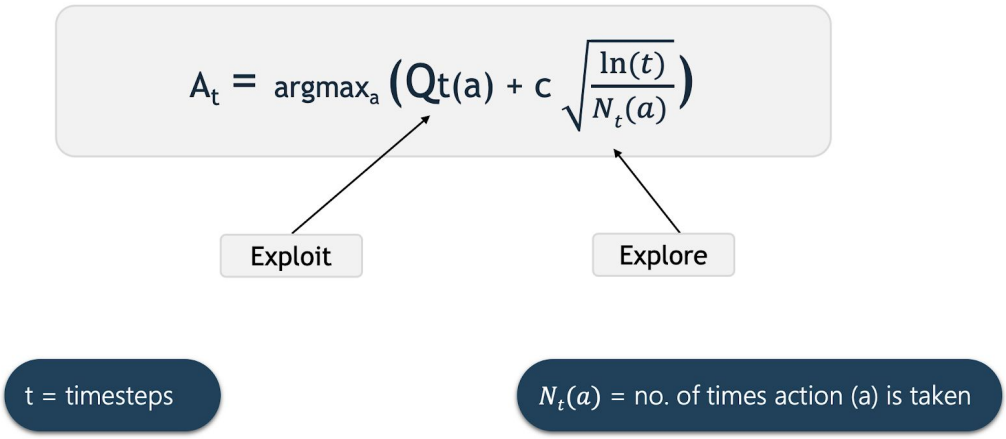
- **Greedy Action**: When an agent chooses an action that currently has the largest estimated value. The agent exploits its current knowledge by choosing the greedy action.
- **Non-Greedy Action**: When the agent does not choose the largest estimated value and sacrifice immediate reward hoping to gain more information about the other actions.
- **Exploration**: It allows the agent to improve its knowledge about each action. Hopefully, leading to a long-term benefit.

- **Exploitation**: It allows the agent to choose the greedy action to try to get the most reward for short-term benefit. A pure greedy action selection can lead to sub-optimal behaviour.

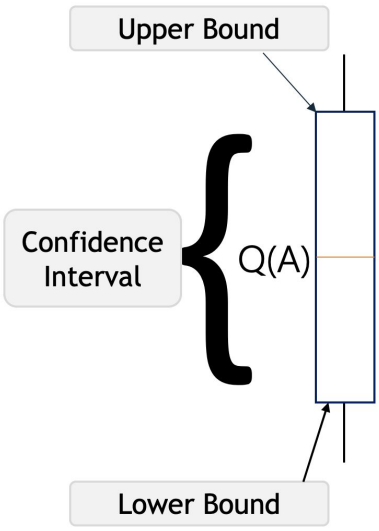
A dilemma occurs between exploration and exploitation because an agent can not choose to both explore and exploit at the same time. Hence, we use the *Upper Confidence Bound* algorithm to solve the exploration-exploitation dilemma

Upper Confidence Bound Action Selection:

Upper-Confidence Bound action selection uses uncertainty in the action-value estimates for balancing exploration and exploitation. Since there is inherent uncertainty in the accuracy of the action-value estimates when we use a sampled set of rewards thus UCB uses uncertainty in the estimates to drive exploration.



$Q_t(a)$ here represents the current estimate for action *a* at time *t*. We select the action that has the highest estimated action-value plus the upper-confidence bound exploration term.



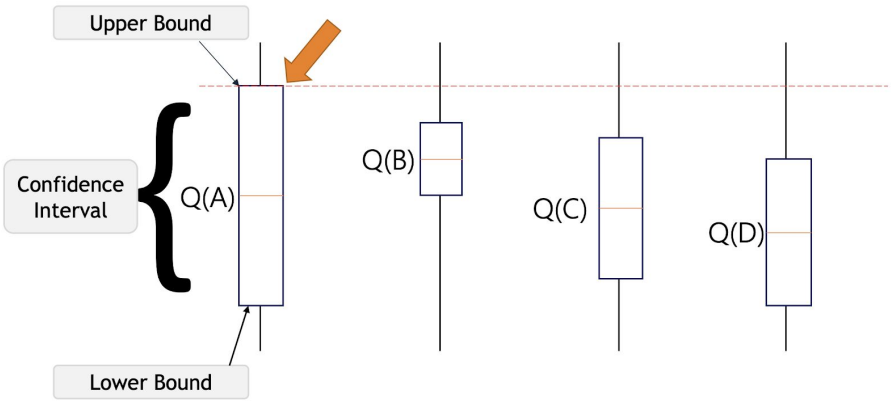
Q(A) in the above picture represents the current action-value estimate for action *A*. The brackets represent a confidence interval around *Q*(A)* which says that we are confident that the actual action-value of action *A* lies somewhere in this region.

The lower bracket is called the lower bound, and the upper bracket is the upper bound. The region between the brackets is the confidence interval which represents the uncertainty in the estimates. If the region is very small, then we become very certain that the actual value of action *A* is near our estimated value. On the other hand, if the region is large, then we become uncertain that the value of action *A* is near our estimated value.

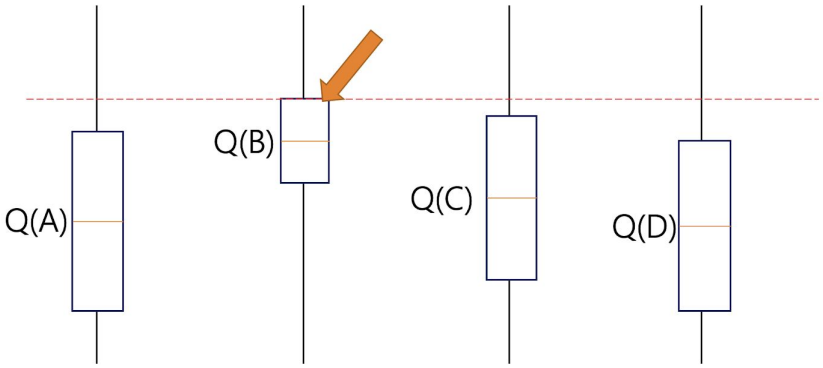
The **Upper Confidence Bound** follows the principle of optimism in the face of uncertainty which implies that if we are uncertain about an action, we should optimistically assume that it is the correct action.

For example, let's say we have these four actions with associated uncertainties in the picture below, our agent has no idea which is the best action. So according to the UCB algorithm, it will optimistically pick the action that has the highest upper bound i.e. *A*. By

doing this either it will have the highest value and get the highest reward, or by taking that we will get to learn about an action we know least about.



Let's assume that after selecting the action *A* we end up in a state depicted in the picture below. This time UCB will select the action *B* since *Q(B)* has the highest upper-confidence bound because it's action-value estimate is the highest, even though the confidence interval is small.



12. Bag Of Words:

1. What is this algo about?

   The bag-of-words model is a way of representing text data when modeling text with machine learning algorithms.

   The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.
2. The Problem with Text:

   Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.A popular and simple method of dealing with text data is called the bag-of-words model of text.
3. What is a Bag-of-Words?

   A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:
   a. A vocabulary of known words.
   b. A measure of the presence of known words.

   It is called a “bag” of words, because *any information about the order or structure of words in the document is discarded.* The model is only concerned with whether known words occur in the document, not where in the document.
4. Logic behind why we use a bag:

   The intuition is that documents are similar if they have similar content.Further, that from the content alone we can learn something about the meaning of the document.
5. How to apply?

   Make a list/dictionary of all the words, key being the word and frequency being it’s value.

   A LITTLE PROBLEM:

   words like, “a”, ”the”, etc. which are repeated very oftenly. Thus we cannot emphasize that the most frequent word is most important.

   POSSIBLE SOLUTIONS:

   a. Clean the data

      I. ignoring case

      II. Ignoring punctuation

      III. Ignoring frequent words that don’t contain much information, called stop words, like “a,” “of,” etc.

      IV. Fixing misspelled words.

      V. Reducing words to their stem (e.g. “play” from “playing”) using stemming algorithms.

      b. Penalizing selectively (TF-IDF):

Rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.
This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:
Term Frequency: is a scoring of the frequency of the word in the current document.
Inverse Document Frequency: is a scoring of how rare the word is across documents.
Thus, the scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

c. Create a vocabulary of grouped words

For example, the bigrams in the first line of text in the previous section: “It was the best of times” are as follows:

- “it was”
- “was the”
- “the best”
- “best of”
- “of times”

I have used bullets deliberately to emphasize the fact that the data is stored without order.

6. How to manage huge lists?

A hash function is a bit of math that maps data to a fixed size set of numbers.
For example, we use them in hash tables when programming where perhaps names are converted to numbers for fast lookup.We can use a hash representation of known words in our vocabulary. This addresses the problem of having a very large vocabulary for a large text corpus because we can choose the size of the hash space, which is in turn the size of the vector representation of the document.

Words are hashed deterministically to the same integer index in the target hash space. A binary score or count can then be used to score the word.This is called the “hash trick” or “feature **hashing**“.

The challenge is to choose a hash space to accommodate the chosen vocabulary size to minimize the probability of collisions and trade-off sparsity.

7. Limitations:

- **Vocabulary**: The vocabulary requires careful design, most specifically in order to manage the size, which impacts the sparsity of the document representations.
- **Sparsity**: Sparse representations are harder to model both for computational reasons (space and time complexity) and also for information reasons, where the challenge is for the models to harness so little information in such a large representational space.
- **Meaning**: Discarding word order ignores the context, and in turn meaning of words in the document (semantics). Context and meaning can offer a lot to the model, that if modeled could tell the difference between the same words differently arranged (“this is interesting” vs “is this interesting”), synonyms (“old bike” vs “used bike”), and much more.

13. Restricted Boltzmann Machines:-

RBMs are a two-layered artificial neural network with generative capabilities. They have the ability to learn a probability distribution over its set of input. RBMs were invented by Geoffrey Hinton and can be used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling.

RBMs are a special class of Boltzmann Machines and they are restricted in terms of the connections between the visible and the hidden units. This makes it easy to implement them when compared to Boltzmann Machines. As stated earlier, they are a two-layered neural network (one being the visible layer and the other one being the hidden layer) and these two layers are connected by a fully bipartite graph. This means that every node in the visible layer is connected to every node in the hidden layer but no two nodes in the same group are connected to each other. This restriction allows for more efficient training algorithms than what is available for the general class of Boltzmann machines, in particular, the gradient-based contrastive divergence algorithm.

A Restricted Boltzmann Machine looks like this:

How do Restricted Boltzmann Machines work?

In an RBM, we have a symmetric bipartite graph where no two units within the same group are connected. Multiple RBMs can also be stacked and can be fine-tuned through the process of gradient descent and back-propagation. Such a network is called a Deep Belief Network. Although RBMs are occasionally used, most people in the deep-learning community have started replacing their use with General Adversarial Networks or Variational Autoencoders. RBM is a Stochastic Neural Network which means that each neuron will have some random behavior when activated. There are two other layers of bias units (hidden bias and visible bias) in an RBM. This is what makes RBMs different from autoencoders. The hidden bias RBM produce the activation on the forward pass and the visible bias helps RBM to reconstruct the input during a backward pass. The reconstructed input is always different from the actual input

as there are no connections among the visible units and therefore, no way of transferring information among themselves.

Image Source
The above image shows the first step in training an RBM with multiple inputs. The inputs are multiplied by the weights and then added to the bias. The result is then passed through a sigmoid activation function and the output determines if the hidden state gets activated or not. Weights will be a matrix with the number of input nodes as the number of rows and the number of hidden nodes as the number of columns. The first hidden node will receive the vector multiplication of the inputs multiplied by the first column of weights before the corresponding bias term is added to it.
And if you are wondering what a sigmoid function is, here is the formula:

Image Source: My Blog
So the equation that we get in this step would be,

where h(1) and v(0) are the corresponding vectors (column matrices) for the hidden and the visible layers with the superscript as the iteration (v(0) means the input that we provide to the network) and a is the hidden layer bias vector.
(Note that we are dealing with vectors and matrices here and not one-dimensional values.)

Image Source
Now this image shows the reverse phase or the reconstruction phase. It is similar to the first pass but in the opposite direction. The equation comes out to be:

where v(1) and h(1) are the corresponding vectors (column matrices) for the visible and the hidden layers with the superscript as the iteration and b is the visible layer bias vector.
The learning process
Now, the difference v(0)-v(1) can be considered as the reconstruction error that we need to reduce in subsequent steps of the training process. So the weights are adjusted in each iteration so as to minimize this error and this is what the learning process essentially is. Now, let us try to understand this process in mathematical terms without going too deep into the mathematics. In the forward pass, we are calculating the probability of output h(1) given the input v(0) and the weights W denoted by:

and in the backward pass, while reconstructing the input, we are calculating the probability of output v(1) given the input h(1) and the weights W denoted by:

The weights used in both the forward and the backward pass are the same. Together, these two conditional probabilities lead us to the joint distribution of inputs and the activations:

Reconstruction is different from regression or classification in that it estimates the probability distribution of the original input instead of associating a continuous/discrete value to an input example. This means it is trying to guess multiple values at the same time. This is known as generative learning as opposed to discriminative learning that happens in a classification problem (mapping input to labels).
Let us try to see how the algorithm reduces loss or simply put, how it reduces the error at each step. Assume that we have two normal distributions, one from the input data (denoted by p(x)) and one from the reconstructed input approximation (denoted by q(x)). The difference between these two distributions is our error in the graphical sense and our goal is to minimize it, i.e., bring the graphs as close as possible. This idea is represented by a term called the Kullback–Leibler divergence. KL-divergence measures the non-overlapping areas under the two graphs and the RBM's optimization algorithm tries to minimize this difference by changing the weights so that the reconstruction closely resembles the input. The graphs on the right-hand side show the integration of the difference in the areas of the curves on the left.

Image by Mundhenk on Wikimedia
This gives us an intuition about our error term. Now, to see how actually this is done for RBMs, we will have to dive into how the loss is being computed. All common training algorithms for RBMs approximate the log-likelihood gradient given some data and perform gradient ascent on these approximations.
Contrastive Divergence
Boltzmann Machines (and RBMs) are Energy-based models and a joint configuration, (v,h) of the visible and hidden units has an energy given by:

where vi, hj, are the binary states of the visible unit i and hidden unit j, ai, bj are their biases and wij is the weight between them.
The probability that the network assigns to a visible vector, v, is given by summing over all possible hidden vectors:

Z here is the partition function and is given by summing over all possible pairs of visible and hidden vectors:

This gives us:

The log-likelihood gradient or the derivative of the log probability of a training vector with respect to a weight is surprisingly simple:

where the angle brackets are used to denote expectations under the distribution specified by the subscript that follows. This leads to a very simple learning rule for performing stochastic steepest ascent in the log probability of the training data:

where alpha is a learning rate. For more information on what the above equations mean or how

they are derived, refer to the Guide on training RBM by Geoffrey Hinton. The important thing to note here is that because there are no direct connections between hidden units in an RBM, it is very easy to get an unbiased sample of ⟨vi hj⟩data. Getting an unbiased sample of ⟨vi hj⟩model, however, is much more difficult. This is because it would require us to run a Markov chain until the stationary distribution is reached (which means the energy of the distribution is minimized — equilibrium!) to approximate the second term. So instead of doing that, we perform Gibbs Sampling from the distribution. It is a Markov chain Monte Carlo (MCMC) algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution, when direct sampling is difficult (like in our case). The Gibbs chain is initialized with a training example v(0) of the training set and yields the sample v(k) after k steps. Each step t consists of sampling h(t) from p(h | v(t)) and sampling v(t+1) from p(v | h(t)) subsequently (the value k = 1 surprisingly works quite well). The learning rule now becomes:

The learning works well even though it is only crudely approximating the gradient of the log probability of the training data. The learning rule is much more closely approximating the gradient of another objective function called the Contrastive Divergence which is the difference between two Kullback-Liebler divergences.
When we apply this, we get:

where the second term is obtained after each k steps of Gibbs Sampling.

14. Autoencoders (Madhumitha)

An **autoencoder** is a type of artificial feed-forward neural network capable of learning complex feature representations in an unsupervised manner. It maps the data fed to a lower dimensionspace by combining the data's most important features. (They *encode* the original data into a more compact representation and decide how the data is combined, hence the *auto* in Autoencoder.)

They are similar to PCA in the sense that both these algorithms result in dimension reduction.

The difference though, lies in the fact that Autoencoders combine the data's important features, resulting in the production of latent features.

An analogy- Say you've bought a lego car set for your friend's birthday and you have a box that's too small to fit all the pieces. PCA corresponds to an approach where you'd cut the pieces systematically into smaller pieces which also allows you to fill the box more.

On the other hand, Autoencoders correspond to the approach where you melt and reshape the pieces representing the most important features of the car, while fitting within the constraints of the box.

The autoencoder involves an encoder and a decoder. The encoder compresses the input into a lower dimension and the decoder reconstructs the input from this.

Pros-

- Reduces dimensionality
- Can learn non-linear feature representations

Cons-

- Extremely uninterpretable
- Since it's a neural network, prone to overfitting

(Main Source- towardsdatascience.com)

15. Eclat: -
The ECLAT algorithm stands for Equivalence Class Clustering and bottom-up Lattice Traversal. It is one of the popular methods of Association Rule mining. It is a more efficient and scalable version of the Apriori algorithm. While the Apriori algorithm works in a horizontal sense imitating the Breadth-First Search of a graph, the ECLAT algorithm works in a vertical manner just like the Depth-First Search of a graph. This vertical approach of the ECLAT algorithm makes it a faster algorithm than the Apriori algorithm.

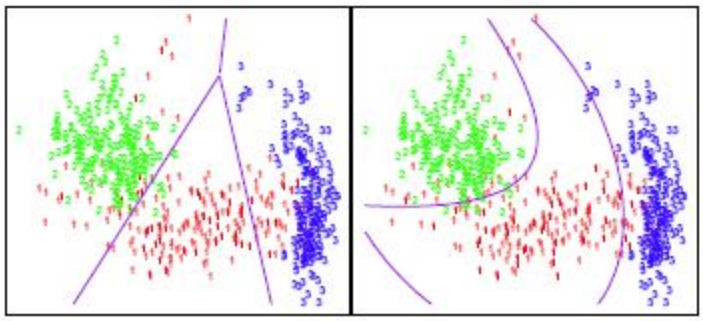
How the algorithm work? :
The basic idea is to use Transaction Id Sets(tidsets) intersections to compute the support value of a candidate and avoiding the generation of subsets which do not exist in the prefix tree. In the first call of the function, all single items are used along with their tidsets. Then the function is called recursively and in each recursive call, each item-tidset pair is verified and combined with other item-tidset pairs. This process is continued until no candidate item-tidset pairs can be combined.

16. Quadratic Discriminant Analysis
(Madhumitha)

Quadratic classifier uses a quadratic decision surface to separate measurements of two or more classes of objects or events. In a way, the linear classifier is a specific case of it (or QDA can be considered the general version of LDA)

Quadratic discriminant analysis (QDA) is closely related to linear discriminant analysis (LDA)/ The main difference is that in LDA, it is assumed that the measurements from each class are normally distributed, i.e the covariance of each of the classes is identical. Unlike LDA however, in QDA each class has its own covariance matrix. (A **covariance** refers to the measure of how two random variables will change together and is used to calculate the correlation between variables).As a result, the quadratic classifier is able to capture differing covariances and provide more accurate non-linear classification decision boundaries.

An example of when QDA proves to be more efficient than LDA-



In the image above, the left plot represents LDA which provides linear decision boundaries that are based on the assumption that the observations vary consistently across all the classes. However, it is apparent when we look at the data that the variability of the observations (covariance) within each class differ. Since QDA uses a non-linear surface, it proves more efficient.

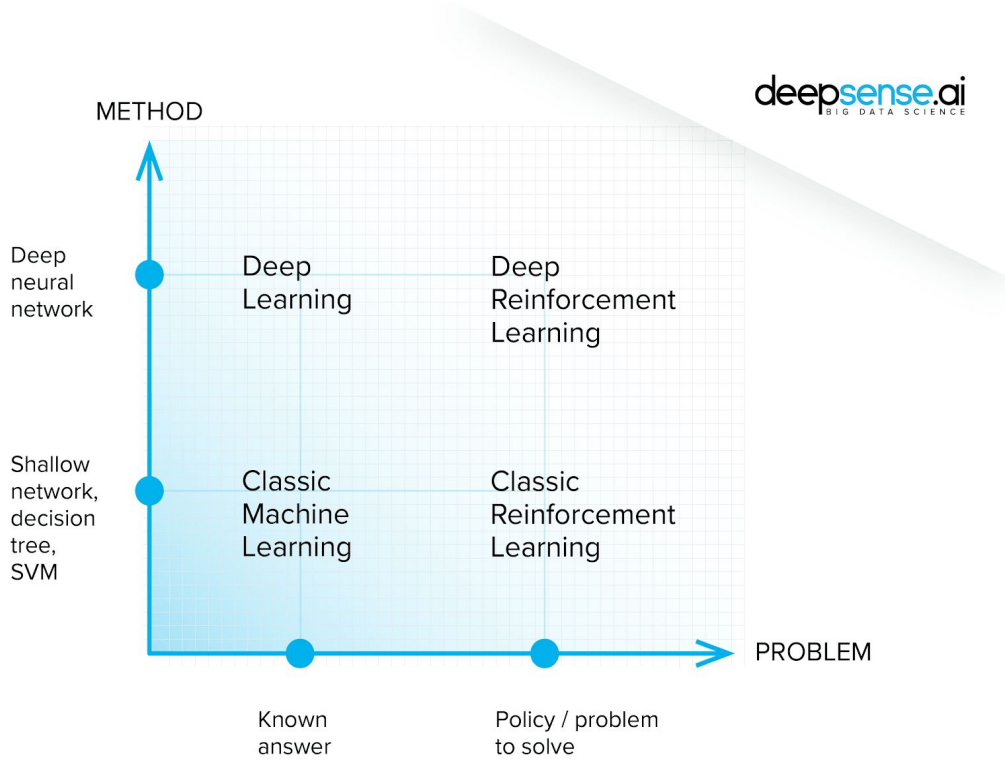
17. REINFORCEMENT LEARNING:

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint at a machine's creativity. In contrast to human beings, artificial intelligence can gather experience from thousands of parallelgameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure.

What distinguishes reinforcement learning from deep learning and machine learning?

In fact, there should be no clear divide between machine learning, deep learning and reinforcement learning. It is like a parallelogram – rectangle – square relation, where machine learning is the broadest category and deep reinforcement learning the most narrow one.In the same way, reinforcement learning is a specialized application of machine and deep learning techniques, designed to solve problems in a particular way.



Although the ideas seem to differ, there is no sharp divide between these subtypes. Moreover, they merge within projects, as the models are designed not to stick to a "pure type" but to perform the task in the most effective way possible. So "what precisely distinguishes machine learning, deep learning and reinforcement learning" is actually a tricky question to answer.

- **Machine learning** – is a form of AI in which computers are given the ability to progressively improve the performance of a specific task with data, without being directly programmed (this is Arthur Lee Samuel's definition. He coined the term "machine learning", of which there are two types, supervised and unsupervised machine learning

Supervised machine learning happens when a programmer can provide a label for every training input into the machine learning system.

- **Example –** by analyzing the historical data taken from coal mines, deepsense.ai prepared an automated system for predicting dangerous seismic events up to 8 hours before they occur. The records of seismic events were taken from 24 coal mines that had collected data for several months. The model was able to recognize the likelihood of an explosion by analyzing the readings from the previous 24 hours.

18 LATENT DIRICHLET ALLOCATION:

In natural language processing, the **Latent Dirichlet Allocation** (**LDA**) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics. LDA is an example of a topic model and belongs to the machine learning toolbox and in a wider sense to the artificial intelligence toolbox.

LDA was applied in machine learning by David Blei, Andrew Ng and Michael I. Jordan in 2003.

Evolutionary biology and biomedicine

In evolutionary biology and biomedicine, the model is used to detect the presence of structured genetic variation in a group of individuals. The model assumes that alleles carried by individuals under study have origin in various extant or past populations. The model and various inference algorithms allow scientists to estimate the allele frequencies in those source populations and the origin of alleles carried by individuals under study. The source populations can be interpreted ex-post in terms of various evolutionary scenarios. In association studies, detecting the presence of genetic structure is considered a necessary preliminary step to avoid confounding.

For diving deeper into the model:
https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation#Overview

19. Thompson Sampling

Contrary to my beliefs, Thompson Sampling is actually a very simple concept and has an equally simple algorithm. Thompson Sampling is a kind of reinforcement learning which makes the use of probability distribution and bayes theorem to make its predictions, and generate success rate probability distributions.

Thompson Sampling is an algorithm that follows exploration and exploitation to maximize the cumulative rewards obtained by performing an action. Thompson Sampling is also sometimes referred to as Posterior Sampling or Probability Matching.

It could be best explained by the multi-armed bandit problem. So the idea is all simple. Given multiple slot machines, the goal is to maximize the reward for a person pulling the lever of each machine randomly. Thompson Sampling makes use of Probability Distribution and Bayes Rule to predict the success rates of each Slot machine.

Basic Intuition Behind Thompson Sampling:-

1. To begin with, all machines are assumed to have a uniform distribution of the probability of success, in this case getting a reward.
2. For each observation obtained from a Slot machine, based on the reward a new distribution is generated with probabilities of success for each slot machine.
3. Further observations are made based on these prior probabilities obtained on each round or observation which then updates the success distributions
4. After sufficient observations, each slot machine will have a success. distribution associated with it which can help the player in choosing the machines wisely to get the maximum rewards.

20. XG Boost

Ever since its introduction in 2014, XGBoost has been lauded as the holy grail of machine learning hackathons and competitions. The accuracy it consistently gives, and the time it saves, demonstrates how useful it is. XGBoost stands for eXtreme Gradient Boosting. The beauty of this powerful algorithm lies in its scalability, which drives fast learning through parallel and distributed computing and offers efficient memory usage.

XGBoost is an ensemble learning method. Sometimes, it may not be sufficient to rely upon the results of just one machine learning model. Ensemble learning offers a systematic solution to combine the predictive power of multiple learners. The resultant is a single model which gives the aggregated output from several models. Bagging and boosting are two widely used ensemble learners.

The mathematics and algorithm to implement XG boost is far beyond my comprehensive abilities however, fret not, as XG Boost comes in an available library of python so it's ez.

If you can make sense of how XG boost works then:

"Gradient descent helps us minimize any differentiable function. Earlier, the regression tree for hm(x) predicted the mean residual at each terminal node of the tree. In gradient boosting, the average gradient component would be computed.

For each node, there is a factor γ with which hm(x) is multiplied. This accounts for the difference in impact of each branch of the split. Gradient boosting helps in predicting the optimal gradient for the additive model, unlike classical gradient descent techniques which reduce error in the output at each iteration."

The following steps are involved in gradient boosting:

- *F0(x) – with which we initialize the boosting algorithm – is to be defined:*

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$

- *The gradient of the loss function is computed iteratively:*

$$r_{im} = -\alpha \left[\frac{\delta(L(y_i, F(x_i))}{\delta F(x_i)} \right]_{F(x)=F_{m-1}(x)}, \text{ where } \alpha \text{ is the learning rate}$$

- *Each hm(x) is fit on the gradient obtained at each step*

- *The multiplicative factor γm for each terminal node is derived and the boosted model Fm(x) is defined:*

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

Here are some popular benefits of XG boost.

- RegularizationHandling sparse data
- Weighted quantile sketch
- Block structure for parallel learning
- Cache awareness
- Out-of-core computing

Package; `from xgboost import XGBClassifier`

References

https://xgboost.readthedocs.io/en/latest/

https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/