MACHINE LEARNING
UNIT-1

Basic Concepts

Definition of learning systems

A **learning system** is essentially a collection of artefacts that are 'brought together', in an appropriate way, in order to create an environment that will facilitate various types of **learning** process. **Learning system**s can take a variety of different forms - for example, a book, a mobile form, a computer, an online forum, a school and a university. Most **learning system**s will provide various types of **learning** resource and descriptions of procedures for using these to achieve particular **learning** outcomes. They will also embed various strategies for assessing the levels and quality of the achievement of their users.

Goals and applications of machine learning

The Goals of Machine Learning.

The goal of ML, in simple words, is to understand the nature of (human and other forms of) learning, and to build learning capability in computers. To be more specific, there are three aspects of the goals of ML.

1) To make the computers smarter, more intelligent. The more direct objective in this aspect is to develop systems (programs) for specific practical learning tasks in application domains.
2) To develop computational models of human learning process and perform computer simulations. The study in this aspect is also called cognitive modelling.
3) To explore new learning methods and develop general learning algorithms independent of applications.

The Applications of Machine Learning.

Machine learning is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect. We probably use a learning algorithm dozens of time without even knowing it. Applications of Machine Learning include:

- **Web Search Engine:** One of the reasons why search engines like google, bing etc work so well is because the system has learnt how to rank pages through a complex learning algorithm.
- **Photo tagging Applications:** Be it facebook or any other photo tagging application, the ability to tag friends makes it even more happening. It is all possible because of a face recognition algorithm that runs behind the application.
- **Spam Detector:** Our mail agent like Gmail or Hotmail does a lot of hard work for us in classifying the mails and moving the spam mails to spam folder. This is again achieved by a spam classifier running in the back end of mail application.

Today, companies are using Machine Learning to improve business decisions,increase productivity, detect disease, forecast weather, and do many more things. With the exponential growth of technology, we not only need better tools to understand the data we currently have, but we also need to prepare ourselves for the data we will have. To achieve this goal we need to build intelligent machines. We can write a program to do simple things. But for most of times Hardwiring Intelligence in it is difficult. Best way to do it is to have some way for machines to learn things themselves. A mechanism for learning – if a machine can learn from input then it does the hard work for us. This is where Machine Learning comes in action. Some examples of machine learning are:

- **Database Mining for growth of automation:** Typical applications include Web-click data for better UX(User eXperience), Medical records for better automation in healthcare, biological data and many more.
- **Applications that cannot be programmed:** There are some tasks that cannot be programmed as the computers we use are not modelled that way. Examples include Autonomous Driving, Recognition tasks from unordered data (Face Recognition/ Handwriting Recognition), Natural language Processing, computer Vision etc.
- **Understanding Human Learning:** This is the closest we have understood and mimicked the human brain. It is the start of a new revolution, The real AI. Now, After a brief insight lets come to a more formal definition of Machine Learning
- **Arthur Samuel(1959):** "Machine Learning is a field of study that gives computers, the ability to learn without explicitly being programmed."Samuel wrote a Checker playing program which could learn over time. At first it could be easily won. But over time, it learnt all the board position that would eventually lead him to victory or loss and thus became a better chess player than Samuel itself. This was one of the most early attempts of defining Machine Learning and is somewhat less formal.
- **Tom Michel(1999):** "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." This is a more formal and mathematical definition. For the previous Chess program
    - E is number of games.
    - T is playing chess against computer.

- P is win/loss by computer.

Aspects of developing a learning system

Training data

Training data is the real fuel to accelerate the machine learning process. It can only provide the actual inputs to the algorithms to learn the certain patterns and utilize this training to predict with the right results if such data comes again in real-life use.

Actually, training data is generated through labeling process which involves image annotation, text annotation and video annotation using certain techniques to make the objects recognizable to computer vision for machine learning training.

Labeling such data is called "**Data Annotation**" done by the well-trained and experienced annotators to annotate the images available in different formats. The best tools and techniques are used to annotate the object of interest in a image while ensuring the accuracy to train the popular AI models like self-driving cars or robotics.

How Data Annotation is Done?

To annotate the data the combination of humans and machines are used at large scale for mass production and achieve the economies of scale. To annotate the data bulk of images are uploaded on the software servers and then given access to the annotators for annotating the each image as per the requirements.

Types of Data Annotation

The data annotation service basically divided into three categories text, videos or images. And annotation job is done as per the AI project training data needs and algorithms compatibility to utilize the information from such labeled data.

- Text Annotation
- Video Annotation
- Image Annotation

Image annotation is used to train the computer vision based perception model.

And there are different types of techniques adopted in this image annotation service. Bounding box, semantic segmentation, 3D cuboid, polygons, landmark annotation and polylines annotation are the leading image annotation methods used in such tasks.

How Training Data is used in Machine Learning?

Machine learning training is performed through certain amount of data from the relevant field. And to train the AI model through machine learning certain process is followed that involves acquiring training data, using the right algorithm, validation of the model and implementation to check the prediction results in real-life use.

In fact, in machine learning labeled or unlabeled training data is used as per the supervised or unsupervised ML process. In supervised machine learning the objected is categories, classified and segmented to make it recognizable to machines.

While in unsupervised machine learning, the data is not labeled and algorithm have to group the object as per the understandings to group them accordingly with its own segmentation to recognize the same when used in future.

How to Acquire Training Data for Machine Learning?

Machine learning training is possible only when you have quality data sets, and acquiring the training data is one the most challenging job in AI world. But if you get in touch with companies like Cogito, you can get high-quality training data for Artificial Intelligence your machine learning project with best level of accuracy for right prediction.

Concept representation

In deep learning, feature representations are generally learned as a blob of ungrouped features. However, an increasing number of visual applications nourish from inferring knowledge from imagery which requires scene understanding. Semantic segmentation is a task that paves the way towards scene understanding. Deep semantic segmentation [17] uses deep learning for semantic segmentation.

Deep semantic segmentation makes dense predictions inferring labels for every pixel. It can be carried out at three different levels:

- Class segmentation: each pixel is labeled with the class of its enclosing object or region
- Instance segmentation: separate labels for different instances of the same class
- Part segmentation: decomposition of already segmented classes into their component sub-classes

CODL extends and generalizes deep semantic segmentation. In CODL, feature representations are always learned semantically segmented in a concept-oriented manner. Concept orientation means that each feature representation is associated with a concept, an instance or an attribute. These concepts, instances and attributes form a concept graph. In addition, the concept graph are generally linked to Microsoft Concept Graph, thus leveraging and integrating with the common conceptual knowledge and conceptual understanding capability provided by Microsoft Concept Graph.

A concept representation consists of a concept, its instances and attributes, and all the feature representations associated with the concept and its instances and attributes. If a concept has sub-concepts, its concept representation also consists of the 5 concept representations of its sub-concepts. Concept representations, therefore, are the same as concept-oriented feature representations, but provide a different view. The latter is data driven and provides a bottom-up view starting from feature representations; the former is concept driven and provides a top-down view starting from concepts. Due to the focus on concepts instead of low-level feature representations, concept representations provide the proper view to work with in CODL.

Supervised Concept Representation Learning

Concept representations can be learned using supervised learning. Similar to deep semantic segmentation, discussed above, it can be carried out at different levels:

- Concept level: each feature representation is labeled with the concept that owns the feature
- Instance level: separate labels for different instances of the same concept
- Attribute level: separate labels for different attributes of the same concept
- Component level: decomposition of already learned concept representations into their sub-concept representations

The concept, instance and attribute names used for labeling should be taken from Microsoft Concept Graph, if available. This provides direct link to Microsoft Concept Graph to leverage its common conceptual knowledge and conceptual understanding capability.

Function approximation

In general, a function approximation problem asks us to select a function among a well-defined class that closely matches ("approximates") a **target function** in a task-specific

way. The need for **function approximations** arises in many branches of applied mathematics, and computer science in particular.

One can distinguish two major classes of function approximation problems:

First, for known target functions approximation theory is the branch of numerical analysis that investigates how certain known functions (for example, special functions) can be approximated by a specific class of functions (for example, polynomials or rational functions) that often have desirable properties (inexpensive computation, continuity, integral and limit values, etc.).

Second, the target function, call it *g*, may be unknown; instead of an explicit formula, only a set of points of the form (*x*, *g*(*x*)) is provided. Depending on the structure of the domain and codomain of *g*, several techniques for approximating *g* may be applicable. For example, if *g* is an operation on the real numbers, techniques of interpolation, extrapolation, regression analysis, and curve fitting can be used. If the codomain (range or target set) of *g* is a finite set, one is dealing with a classification problem instead.

To some extent, the different problems (regression, classification, fitness approximation) have received a unified treatment in statistical learning theory, where they are viewed as supervised learning problems.

Types of Learning

Supervised learning and unsupervised learning

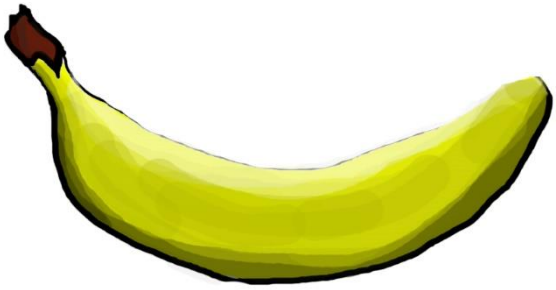
Supervised learning

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

- If shape of object is rounded and depression at top having color Red then it will be labelled as –**Apple**.
- If shape of object is long curving cylinder having color Green-Yellow then it will be labelled as –**Banana**.

Now suppose after training the data, you have given a new separate fruit say Banana from basket and asked to identify it.



Since the machine has already learned the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color and would confirm

the fruit name as BANANA and put it in Banana category. Thus the machine learns the things from training data(basket containing fruits) and then apply the knowledge to test data(new fruit).

Supervised learning classified into two categories of algorithms:

- **Classification**: A classification problem is when the output variable is a category, such as “Red” or “blue” or “disease” and “no disease”.
- **Regression**: A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Unsupervised learning

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabeled data by our-self.
For instance, suppose it is given an image having both dogs and cats which have not seen ever.

Thus the machine has no idea about the features of dogs and cat so we can’t categorize it in dogs and cats. But it can categorize them according to their similarities, patterns, and differences i.e., we can easily categorize the above picture into two parts. First first may contain all pics having **dogs** in it and second part may contain all pics having **cats** in it. Here you didn’t learn anything before, means no training data or examples.
Unsupervised learning classified into two categories of algorithms:

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **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.

Overview of classification

Classification - Machine Learning

This is 'Classification' tutorial which is a part of the Machine Learning course offered by Simplilearn. We will learn Classification algorithms, types of classification algorithms, support vector machines(SVM), Naive Bayes, Decision Tree and Random Forest Classifier in this tutorial.

Objectives

Let us look at some of the objectives covered under this section of Machine Learning tutorial.

- Define Classification and list its algorithms
- Describe Logistic Regression and Sigmoid Probability
- Explain K-Nearest Neighbors and KNN classification Understand Support Vector Machines, Polynomial Kernel, and Kernel Trick
- Analyze Kernel Support Vector Machines with an example
- Implement the Naïve Bayes Classifier
- Demonstrate Decision Tree Classifier
- Describe Random Forest Classifier

Classification: Meaning

Classification is a type of supervised learning. It specifies the class to which data elements belong to and is best used when the output has finite and discrete values. It predicts a class for an input variable as well.
There are 2 types of Classification:

- Binomial
- Multi-Class

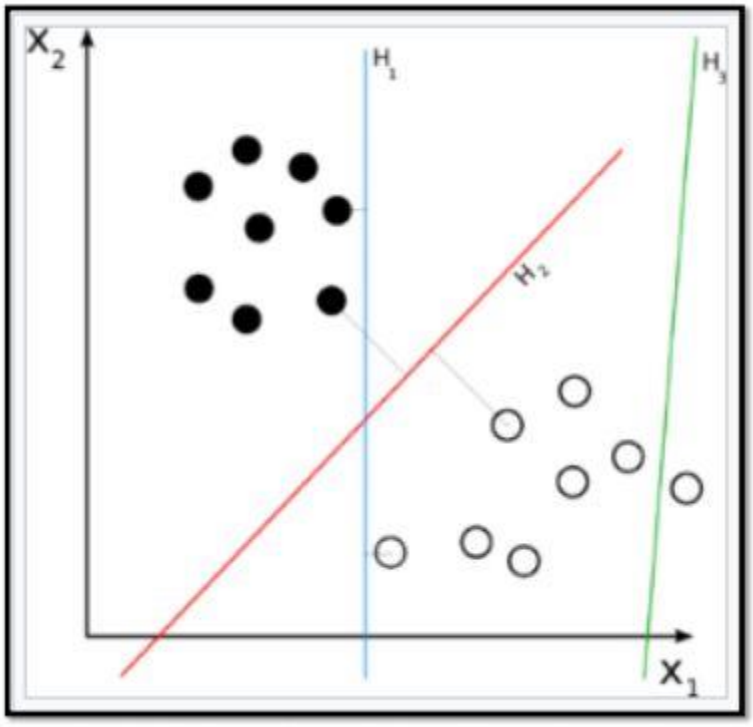
Classification: Use Cases

Some of the key areas where classification cases are being used:

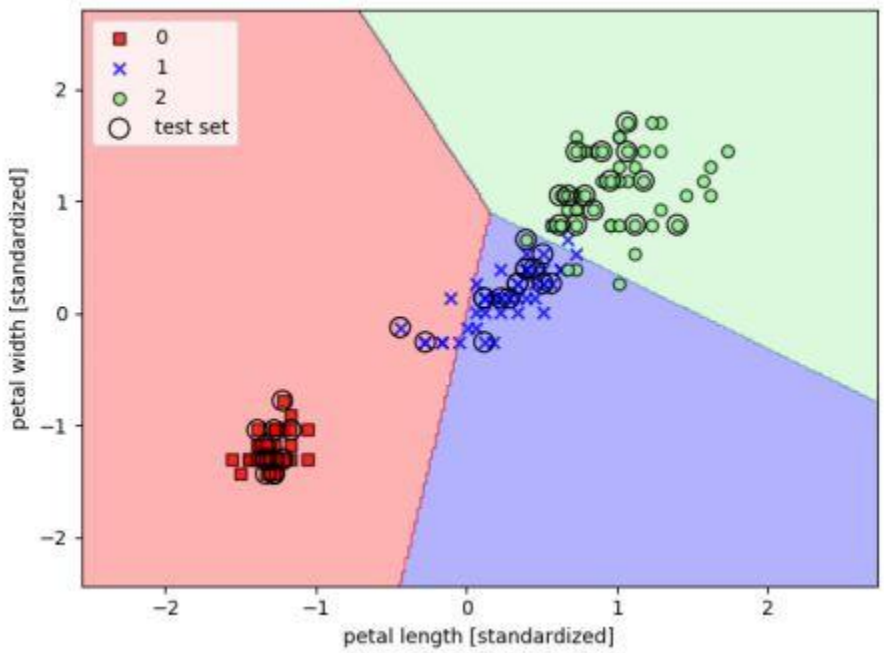
- To find whether an email received is a spam or ham
- To identify customer segments
- To find if a bank loan is granted
- To identify if a kid will pass or fail in an examination

Classification: Example

Social media sentiment analysis has two potential outcomes, positive or negative, as displayed by the chart given below.



- This chart shows the classification of the Iris flower dataset into its three sub-species indicated by codes 0, 1, and 2.



- The test set dots represent the assignment of new test data points to one class or the other based on the trained classifier model.

Types of Classification Algorithms

Let's have a quick look into the types of Classification Algorithm below.

- Linear Models
  - Logistic Regression
  - Support Vector Machines
- Nonlinear models
  - K-nearest Neighbors (KNN)
  - Kernel Support Vector Machines (SVM)
  - Naïve Bayes
  - Decision Tree Classification
  - Random Forest Classification

Training Dataset

Training Dataset*: The sample of data used to fit the model.*

The actual dataset that we use to train the model (weights and biases in the case of Neural Network). The model *sees* and *learns* from this data.

Validation Dataset

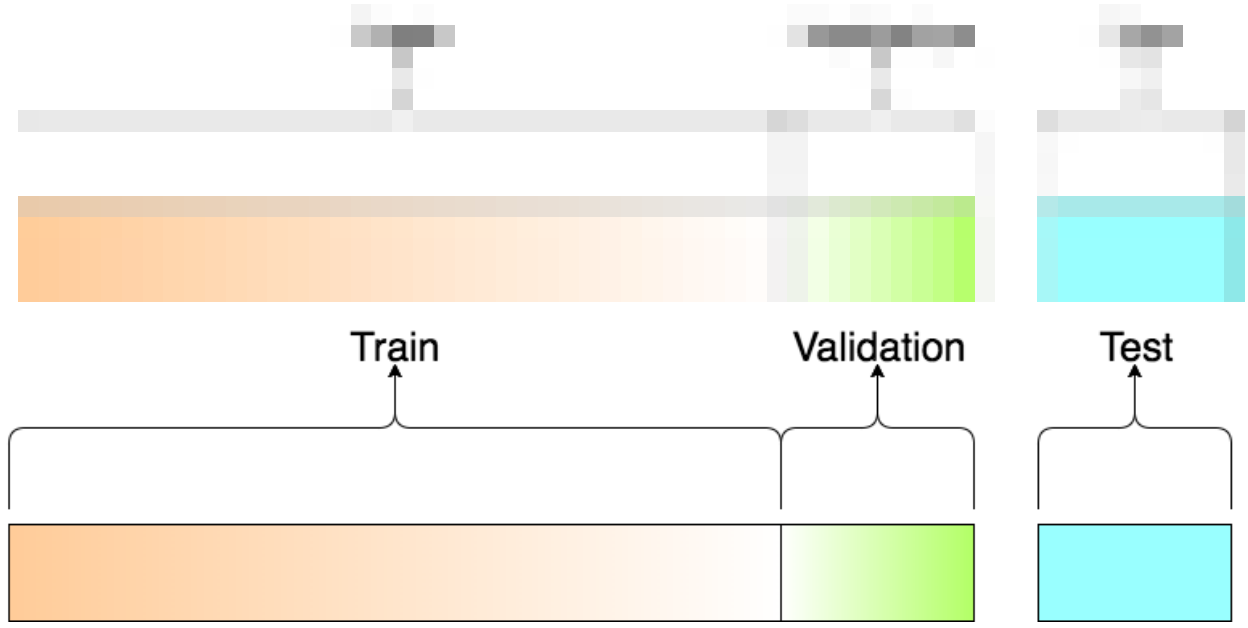
Validation Dataset*: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.*

The validation set is used to evaluate a given model, but this is for frequent evaluation. We as machine learning engineers use this data to fine-tune the model hyperparameters. Hence the model occasionally *sees* this data, but never does it "*Learn*" from this. We(mostly humans, at-least as of 2017 😛) use the validation set results and update higher level hyperparameters. So the validation set in a way affects a model, but indirectly.

Test Dataset

***Test Dataset**: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.*

The Test dataset provides the gold standard used to evaluate the model. It is only used once a model is completely trained(using the train and validation sets). The test set is generally what is used to evaluate competing models (For example on many Kaggle competitions, the validation set is released initially along with the training set and the actual test set is only released when the competition is about to close, and it is the result of the the model on the Test set that decides the winner). Many a times the validation set is used as the test set, but it is not good practice. The test set is generally well curated. It contains carefully sampled data that spans the various classes that the model would face, when used in the real world.



A visualisation of the splits

About the dataset split ratio

Now that you know what these datasets do, you might be looking for recommendations on how to split your dataset into Train, Validation and Test sets...

This mainly depends on 2 things. First, the total number of samples in your data and second, on the actual model you are training.

Some models need substantial data to train upon, so in this case you would optimize for the larger training sets. Models with very few hyperparameters will be easy to validate and tune, so you can probably reduce the size of your validation set, but if your model has many hyperparameters, you would want to have a large validation set as well(although you should also consider cross validation). Also, if you happen to have a model with no hyperparameters or ones that cannot be easily tuned, you probably don't need a validation set too!

All in all, like many other things in machine learning, the train-test-validation split ratio is also quite specific to your use case and it gets easier to make judge ment as you train and build more and more models.

Note on Cross Validation: Many a times, people first split their dataset into 2 — Train and Test. After this, they keep aside the Test set, and randomly choose X% of their Train dataset to be the actual ***Train*** *set and the remaining (100-X)% to be the* ***Validation*** *set, where X is a fixed number(say 80%), the model is then iteratively trained and validated on these different sets. There are multiple ways to do this, and is commonly known as Cross Validation. Basically you use your training set to generate multiple splits of the Train and Validation sets. Cross validation avoids over fitting and is getting more and more popular, with K-fold Cross Validation being the most popular method of cross validation.*

Overfitting in Machine Learning

Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

Classification Families

Discriminant Analysis

During a study, there are often questions that strike the researcher that must be answered. These questions include questions like 'are the groups different?', 'on what variables, are the groups most different?', 'can one predict which group a person belongs to using such variables?' etc. In answering such questions, **discriminant analysis** is quite helpful. Statistics Solutions is the country's leader in discriminant analysis and dissertation statistics.

Discriminant analysis is a technique that is used by the researcher to analyze the research data when the criterion or the dependent variable is categorical and the predictor or the independent variable is interval in nature. The term categorical variable means that the dependent variable is divided into a number of categories. For example, three brands of computers, Computer A, Computer B and Computer C can be the categorical dependent variable.

The objective of discriminant analysis is to develop discriminant functions that are nothing but the linear combination of independent variables that will discriminate between the categories of the dependent variable in a perfect manner. It enables the researcher to examine whether significant differences exist among the groups, in terms of the predictor variables. It also evaluates the accuracy of the classification.

Discriminant analysis is described by the number of categories that is possessed by the dependent variable.

As in statistics, everything is assumed up until infinity, so in this case, when the dependent variable has two categories, then the type used is two-group discriminant analysis. If the dependent variable has three or more than three categories, then the type used is multiple discriminant analysis. The major distinction to the types of discriminant analysis is that for a two group, it is possible to derive only one discriminant function. On the other hand, in the case of multiple discriminant analysis, more than one discriminant function can be computed.

There are many examples that can explain when discriminant analysis fits. It can be used to know whether heavy, medium and light users of soft drinks are different in terms of their consumption of frozen foods. In the field of psychology, it can be used to differentiate between the price sensitive and

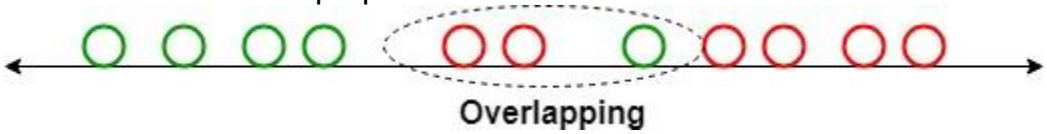
non price sensitive buyers of groceries in terms of their psychological attributes or characteristics. In the field of business, it can be used to understand the characteristics or the attributes of a customer possessing store loyalty and a customer who does not have store loyalty.

For a researcher, it is important to understand the relationship of discriminant analysis with Regression and Analysis of Variance (ANOVA) which has many similarities and differences. Often we can find similarities and differences with the people we come across. Similarly, there are some similarities and differences with discriminant analysis along with two other procedures. The similarity is that the number of dependent variables is one in discriminant analysis and in the other two procedures, the number of independent variables are multiple in discriminant analysis. The difference is categorical or binary in discriminant analysis, but metric in the other two procedures. The nature of the independent variables is categorical in Analysis of Variance (ANOVA), but metric in regression and discriminant analysis.

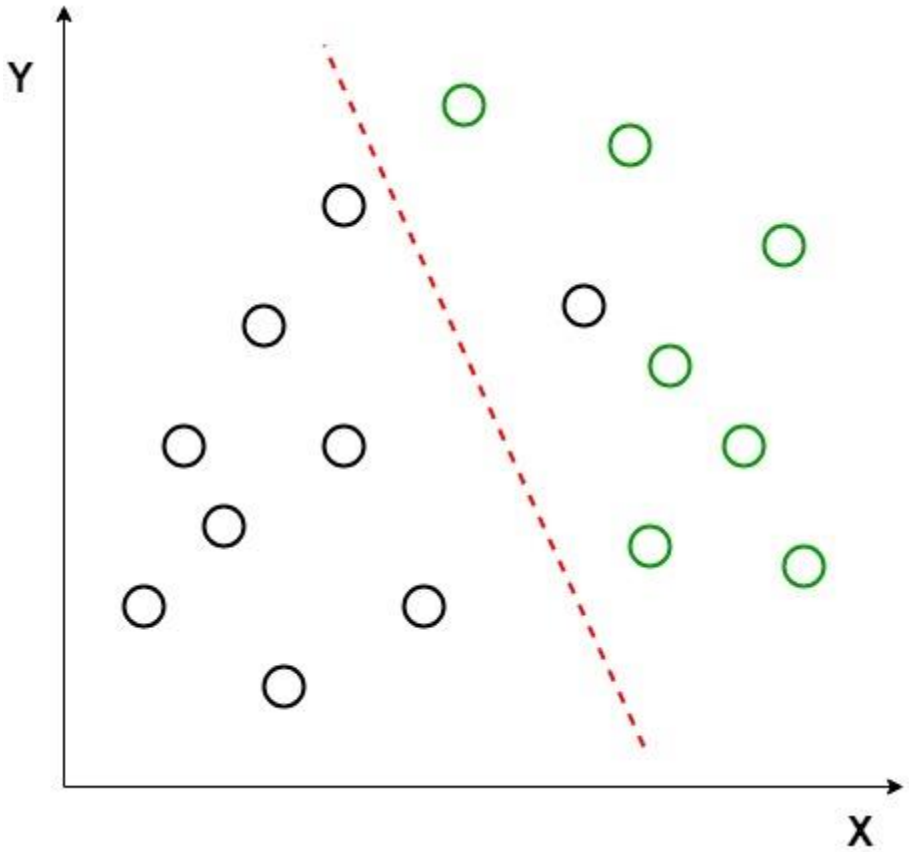
The steps involved in conducting discriminant analysis are as follows:
- The problem is formulated before conducting.
- The discriminant function coefficients are estimated.
- The next step is the determination of the significance of these discriminant functions.
- One must interpret the results obtained.
- The last and the most important step is to assess the validity.

Linear Discriminant Analysis

Linear Discriminant Analysis or **Normal Discriminant Analysis** or **Discriminant Function Analysis** is a dimensionality reduction technique which is commonly used for the supervised classification problems. It is used for modeling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.
For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.



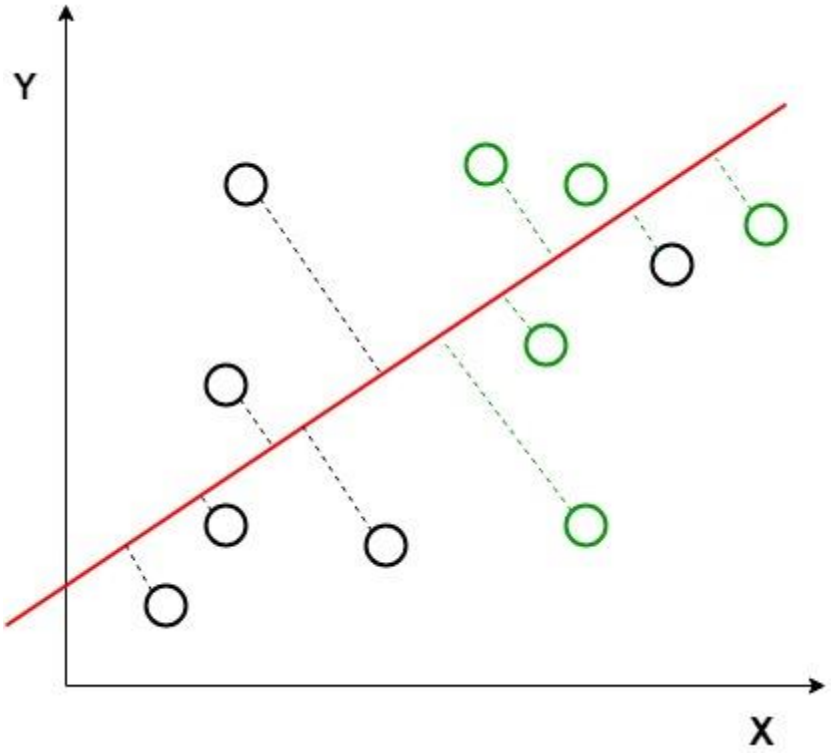
Example:
Suppose we have two sets of data points belonging to two different classes that we want to classify. As shown in the given 2D graph, when the data points are plotted on the 2D plane, there's no straight line that can separate the two classes of the data points completely. Hence, in this case, LDA (Linear Discriminant Analysis) is used which reduces the 2D graph into a 1D graph in order to maximize the separability between the two classes.



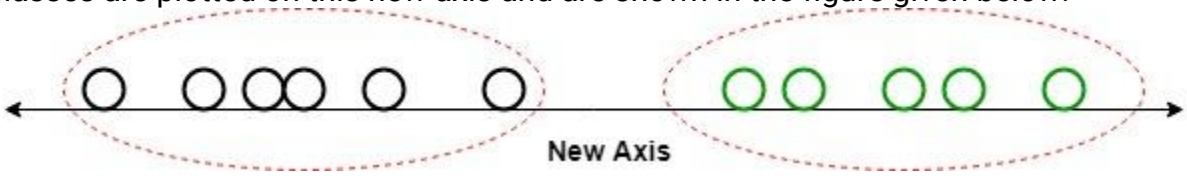
Here, Linear Discriminant Analysis uses both the axes (X and Y) to create a new axis and projects data onto a new axis in a way to maximize the separation of the two categories and hence, reducing the 2D graph into a 1D graph.

Two criteria are used by LDA to create a new axis:

1. Maximize the distance between means of the two classes.
2. Minimize the variation within each class.



In the above graph, it can be seen that a new axis (in red) is generated and plotted in the 2D graph such that it maximizes the distance between the means of the two classes and minimizes the variation within each class. In simple terms, this newly generated axis increases the separation between the dtla points of the two classes. After generating this new axis using the above-mentioned criteria, all the data points of the classes are plotted on this new axis and are shown in the figure given below.



But Linear Discriminant Analysis fails when the mean of the distributions are shared, as it becomes impossible for LDA to find a new axis that makes both the classes linearly separable. In such cases, we use non-linear discriminant analysis.

Extensions to LDA:

1. **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables).
2. **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs is used such as splines.
3. **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

Applications:

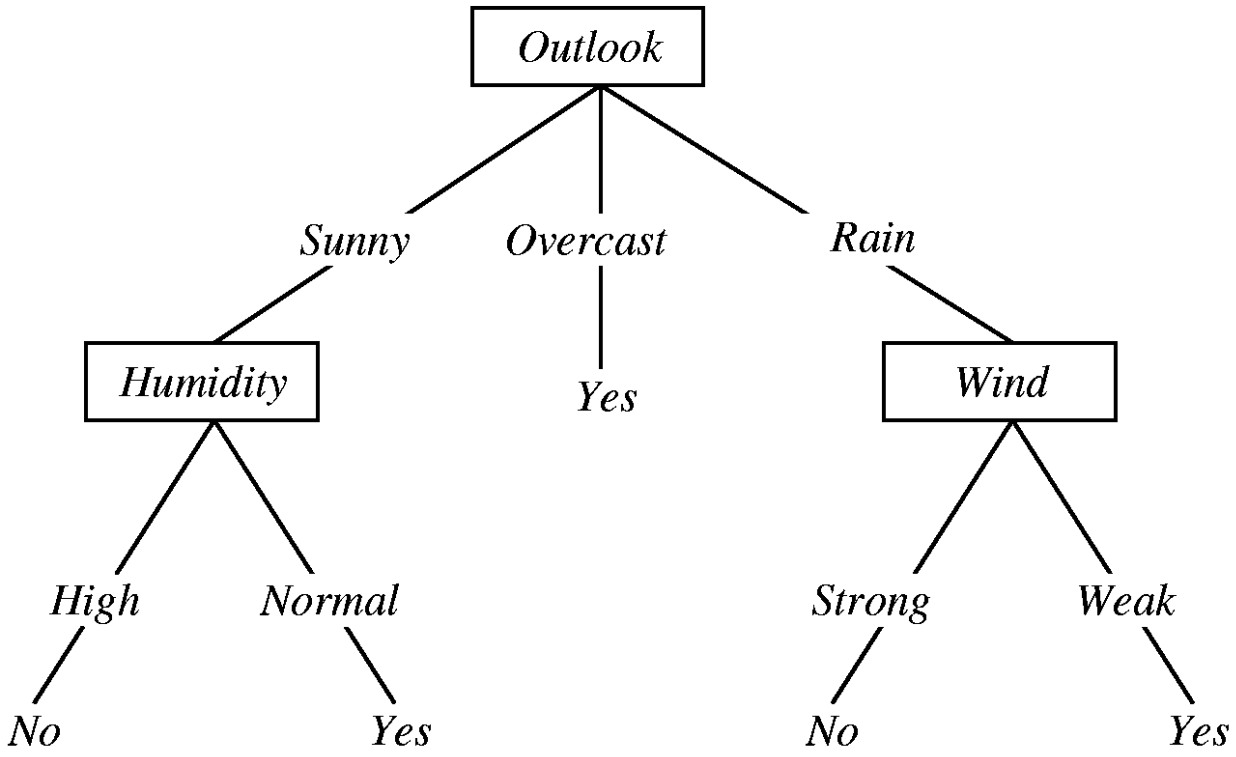
1. **Face Recognition:** In the field of Computer Vision, face recognition is a very popular application in which each face is represented by a very large number of pixel values. Linear discriminant analysis (LDA) is used here to reduce the number of features to a

more manageable number before the process of classification. Each of the new dimensions generated is a linear combination of pixel values, which form a template. The linear combinations obtained using Fisher's linear discriminant are called Fisher faces.

2. **Medical:** In this field, Linear discriminant analysis (LDA) is used to classify the patient disease state as mild, moderate or severe based upon the patient various parameters and the medical treatment he is going through. This helps the doctors to intensify or reduce the pace of their treatment.
3. **Customer Identification:** Suppose we want to identify the type of customers which are most likely to buy a particular product in a shopping mall. By doing a simple question and answers survey, we can gather all the features of the customers. Here, Linear discriminant analysis will help us to identify and select the features which can describe the characteristics of the group of customers that are most likely to buy that particular product in the shopping mall.

Decision Tree in Machine Learning

A decision tree is a flowchart-like structure in which each internal node represents a `test` on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a `class label` (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent `classification rules`. Below diagram illustrate the basic flow of decision tree for decision making with labels (Rain(Yes), No Rain(No)).



Decision Tree for Rain Forecasting

Decision tree is one of the predictive modelling approaches used in `statistics`, `data mining` and `machine learning`.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric **supervised learning** method used for both **classification** and **regression** tasks.

Tree models where the target variable can take a discrete set of values are called **classification trees**. Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**. Classification And Regression Tree (CART) is general term for this.

Throughout this post i will try to explain using the examples.

Data Format

Data comes in records of forms.
`(x,Y)=(x1,x2,x3,....,xk,Y)`

The dependent variable, Y, is the target variable that we are trying to understand, classify or generalize. The vector x is composed of the features, x1, x2, x3 etc., that are used for that task.

Example

```
training_data = [
                  ['Green', 3, 'Apple'],
                  ['Yellow', 3, 'Apple'],
                  ['Red', 1, 'Grape'],
                  ['Red', 1, 'Grape'],
                  ['Yellow', 3, 'Lemon'],
                  ]
 # Header = ["Color", "diameter", "Label"]
 # The last column is the label.
 # The first two columns are features.
```

Approach to make decision tree

While making decision tree, at each node of tree we ask different type of questions. Based on the asked question we will calculate the information gain corresponding to it.

Information Gain

Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes. A commonly used measure of

purity is called information. For each node of the tree, the information value **measures how much information a feature gives us about the class. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.**

Probabilistic classification

In machine learning, a **probabilistic classifier** is a classifier that is able to predict, given an observation of an input, a probability distribution over a set of classes, rather than only outputting the most likely class that the observation should belong to. Probabilistic classifiers provide classification that can be useful in its own right[1] or when combining classifiers into ensembles.

Nearest Neighbour Classifier

Among the various methods of supervised statistical pattern recognition, the Nearest Neighbour rule achieves consistently high performance, without *a priori* assumptions about the distributions from which the training examples are drawn. It involves a training set of both positive and negative cases. A new sample is classified by calculating the distance to the nearest training case; the sign of that point then determines the classification of the sample. The *k*-NN classifier extends this idea by taking the *k* nearest points and assigning the sign of the majority. It is common to select *k* small and odd to break ties (typically 1, 3 or 5). Larger *k* values help reduce the effects of noisy points within the training data set, and the choice of *k* is often performed through cross-validation.

There are many techniques available for improving the performance and speed of a nearest neighbour classification. One approach to this problem is to pre-sort the training sets in some way (such as *kd*-trees or Voronoi cells). Another solution is to choose a subset of the training data such that classification by the 1-NN rule (using the subset) approximates the Bayes classifier. This can result in significant speed improvements as *k* can now be limited to 1 and redundant data points have been removed from the training set. These data modification techniques can also improve the performance through removing points that cause mis-classifications. Several dataset reduction techniques are discussed in the section on target detection.

The above discussion focuses on binary classification problems; there are only two possible output classes. In the digit recognition example there are ten output classes, which changes things slightly. The labelling of training samples and computing the distance are unchanged, but ties can now occur even with k odd. If all of the k nearest neighbours are from different classes we are no closer to a decision than with the single nearest neighbour rule. We will therefore revert to a 1-NN rule when all there is no majority within the k nearest neighbours.

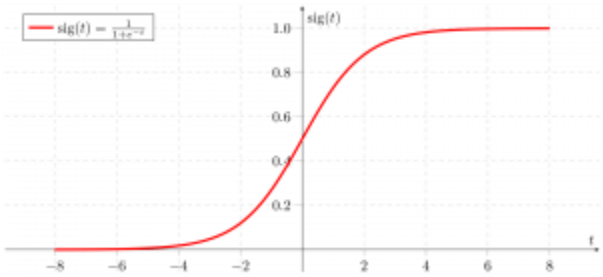
The nearest neighbour rule is quite simple, but very computationally intensive. For the digit example, each classification requires 60,000 distance calculations between 784 dimensional vectors (28x28 pixels). The nearest neighbour code was therefore written in C in order to speed up the Matlab testing. The files are given below, but note that these are set up to read in the image database after it has been converted from the format available on the MNIST web page.

MACHINE LEARNING
UNIT-2

Logistic regression

This article discusses the basics of Logistic Regression and its implementation in Python. Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable(or output), y, can take only discrete values for given set of features(or inputs), X.

Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function.



Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.

The decision for the value of the threshold value is majorly affected by the values of precision and recall. Ideally, we want both precision and recall to be 1, but this seldom is the case. In case of a Precision-Recall tradeoff we use the following arguments to decide upon the thresold:-

1. **Low Precision/High Recall:** In applications where we want to reduce the number of false negatives without necessarily reducing the number false positives, we choose a decision value which has a low value of Precision or high value of Recall. For example, in a cancer diagnosis application, we do not want any affected patient to be classified as not affected without giving much heed to if the patient is being wrongfully diagnosed with cancer. This is because, the absence of cancer can be detected by further medical diseases but the presence of the disease cannot be detected in an already rejected candidate.

2. **High Precision/Low Recall:** In applications where we want to reduce the number of false positives without necessarily reducing the number false negatives, we choose a decision value which has a high value of Precision or low value of Recall. For example, if we are classifying customers whether they will react positively or negatively to a personalised advertisement, we want to be absolutely sure that the customer will react positively to the advertisemnt because otherwise, a negative reaction can cause a loss potential sales from the customer.
Based on the number of categories, Logistic regression can be classified as:

1. **binomial:** target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.
2. **multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.
3. **ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:“very poor”, “poor”, “good”, “very good”. Here, each category can be given a score like 0, 1, 2, 3.

Perceptron

In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

Exponential family

In probability and statistics, an **exponential family** is a parametric set of probability distributions of a certain form, specified below. This special form is chosen for mathematical convenience, based on some useful algebraic properties, as well as for generality, as exponential families are in a sense very natural sets of distributions to consider. The term **exponential class** is sometimes used in place of "exponential family", or the older term **Koopman–Darmois family**. The terms "distribution" and "family" are often used loosely: properly, *an* exponential family is a *set* of distributions, where the specific distribution varies with the parameter; however, a parametric *family* of distributions is often referred to as "*a* distribution" (like "the normal distribution", meaning "the family of normal distributions"), and the set of all exponential families is sometimes loosely referred to as "the" exponential family.

The concept of exponential families is credited to E. J. G. Pitman, G. Darmois, and B. O. Koopman in 1935–1936. Exponential families of distributions provides a general framework for selecting a possible alternative parameterisation of a parametric family of distributions, in terms of **natural parameters**, and for defining useful sample statistics, called the **natural sufficient statistics** of the family.

Generative learning algorithms

Generative approaches **try to build a model** of the *positives* and a model of the *negatives*. You can think of a model as a *"blueprint"* for a class. A decision boundary is formed where one model becomes more likely. As these create models of each class they can be used for generation.

To create these models, a generative learning algorithm learns the **joint probability** distribution **P(x, y)**.

Now time for some maths!

The joint probability can be written as:

$P(x, y) = P(x \mid y) \cdot P(y)$ ....(i)

Also, using Bayes' Rule we can write:

$P(y \mid x) = P(x \mid y) \cdot P(y) / P(x)$ ....(ii)

Since, to predict a class label **y,** we are only interested in the **arg max** , the denominator can be removed from **(ii)**.

Hence to predict the label **y** from the training example **x**, generative models evaluate:

$f(x) = argmax\_y\, P(y \mid x) = argmax\_y\, P(x \mid y) \cdot P(y)$

The most important part in the above is **P(x I y)**. This is what allows the model to be *generative*! **P(x I y)** means – what **x** *(features)* are there given *class* **y**. Hence, with the joint probability distribution function **(i)**, given a **y**, you can calculate (*"generate"*) its corresponding **x**. **For this reason they are called generative models!**

Generative learning algorithms **make strong assumptions** on the data. To explain this let's look at a generative learning algorithm called Gaussian Discriminant Analysis (GDA)

Gaussian discriminant analysis

GDA, is a method for data classification commonly used when data can be approximated with a Normal distribution. As first step, you will need a training set, i.e. a bunch of data yet classified. These data are used to train your classifier, and obtain a discriminant function that will tell you to which class a data has higher probability to belong.

When you have your training set you need to compute the mean μμ and the standard deviation σ2σ2. These two variables, as you know, allow you to describe a Normal distribution.
Once you have computed the Normal distribution for each class, to classify a data you will need to compute, for each one, the probability that that data belongs to it. The class with the highest probability will be chosen as the affinity class.

More information about Discriminant Functions for the Normal Density can be found in textbook as *Pattern Classification DUDA, HART, SOTRK* or *Pattern Recognition and Machine Learning BISHOP*.

Naïve Bayes

In machine learning, **naïve Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models.

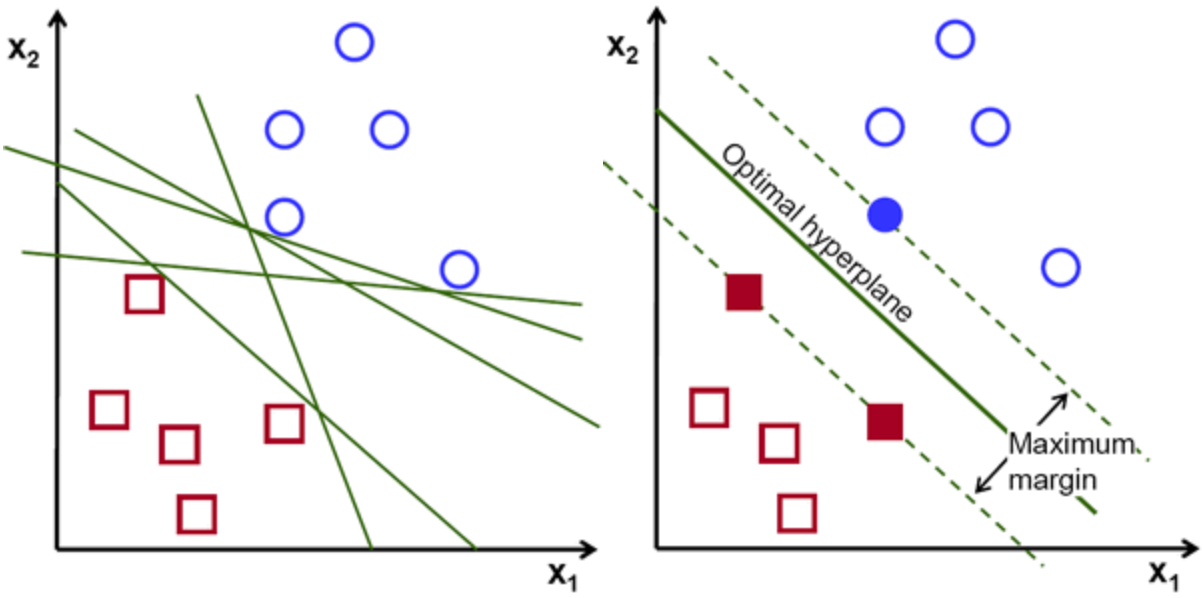
Naïve Bayes has been studied extensively since the 1960s. It was introduced (though not under that name) into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (document categorization)(such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, naive Bayes models are known under a variety of names, including **simple Bayes** and **independence Bayes**. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naïve Bayes is not (necessarily) a Bayesian method.

Support vector machines

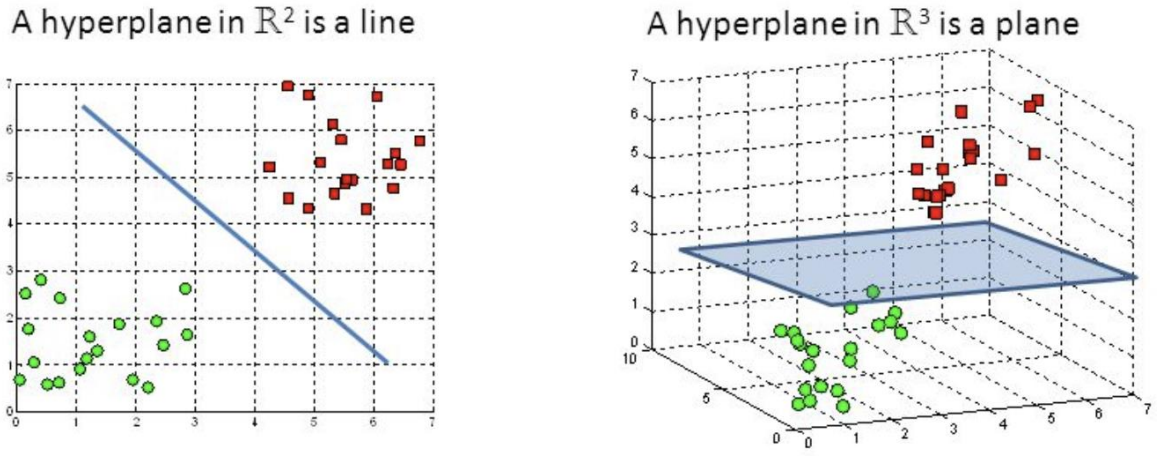
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

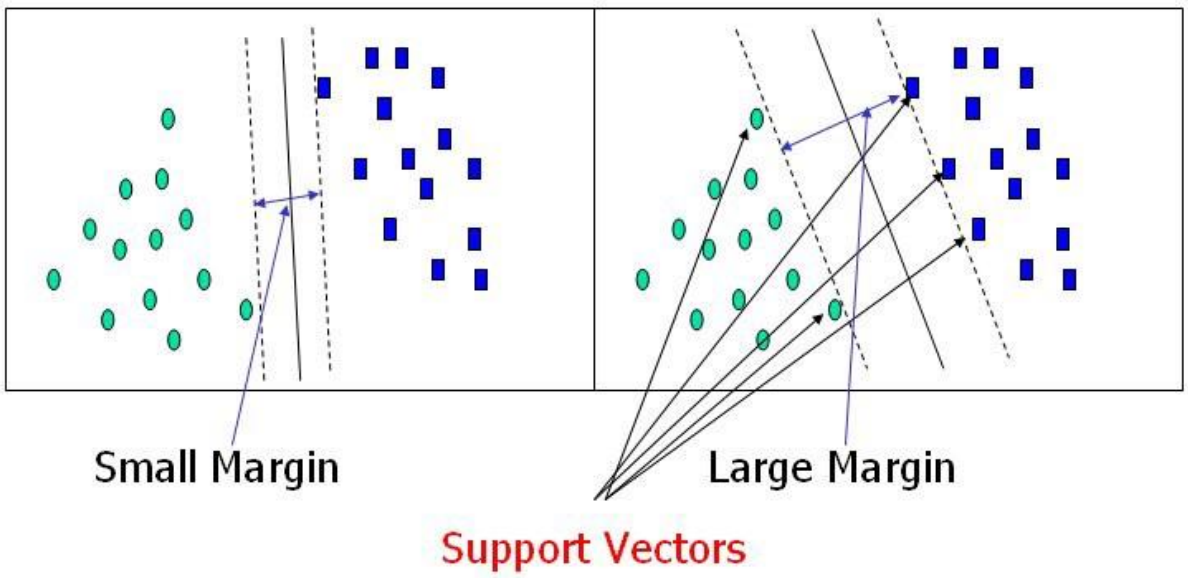
Hyperplanes and Support Vectors



Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the

dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support Vectors

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. 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.

What is Model Selection

Given a set of models $\mathcal{M} = \{M_1, M_2, \ldots, M_R\}$, choose the model that is expected to do the best on the **test data**. $\mathcal{M}$ may consist of:

- Same learning model with different complexities or hyperparameters
  - Nonlinear Regression: Polynomials with different degrees
  - K-Nearest Neighbors: Different choices of K
  - Decision Trees: Different choices of the number of levels/leaves
  - SVM: Different choices of the misclassification penalty hyperparameter C
  - Regularized Models: Different choices of the regularization parameter
  - Kernel based Methods: Different choices of kernels
  - .. and almost any learning problem
- Different learning models (e.g., SVM, KNN, DT, etc.)

Note: Usually considered in supervised learning contexts but unsupervised learning too faces this issue (e.g., "how many clusters" when doing clustering)

Feature Selection

Selecting a useful subset from all the features

Why Feature Selection?

- Some algorithms scale (computationally) poorly with increased dimension
- Irrelevant features can confuse some algorithms
- Redundant features adversely affect regularization
- Removal of features can increase (relative) margin (and generalization)
- Reduces data set and resulting model size
- Note: Feature Selection is different from Feature Extraction
  - The latter transforms original features to get a small set of new features
  - More on feature extraction when we cover **Dimensionality Reduction**

Combining Classifiers

Vote provides a baseline method for combining classifiers. The default scheme is to average their probability estimates or numeric predictions, for classification and regression, respectively. Other combination schemes are available—for example, using majority voting for classification. MultiScheme selects the best classifier from a set of candidates using cross-validation of percentage accuracy or mean-squared error for classification and regression, respectively. The number of folds is a parameter. Performance on training data can be used instead.

Stacking combines classifiers using stacking (see Section 8.7, page 369) for both classification and regression problems. You specify the base classifiers, the metalearner, and the number of cross-validation folds. StackingC implements a more efficient variant for which the metalearner must be a numeric prediction scheme (Seewald, 2002). In Grading, the inputs to the metalearner are base-level predictions that have been marked (i.e., “graded”) as correct or incorrect. For each base classifier, a metalearner is learned that predicts when the base classifier will err. Just as stacking may be viewed as a generalization of voting, grading generalizes selection by cross-validation (Seewald and Fürnkranz, 2001).

Bagging

Bootstrap aggregating, also called **bagging** (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting.

Boosting

The term '**Boosting**' refers to a family of algorithms which converts weak learner to strong learners. **Boosting** is an ensemble method for improving the model predictions of any given learning algorithm. The idea of **boosting** is to train weak learners sequentially, each trying to correct its predecessor.

Evaluating and debugging learning algorithms

Once you have defined your problem and prepared your data you need to apply machine learning algorithms to the data in order to solve your problem.
You can spend a lot of time choosing, running and tuning algorithms. You want to make sure you are using your time effectively to get closer to your goal.
In this post you will step through a process to rapidly test algorithms and discover whether or not there is structure in your problem for the algorithms to learn and which algorithms are effective.

Test Harness

You need to define a test harness. The test harness is the data you will train and test an algorithm against and the performance measure you will use to assess its performance. It is important to define your test harness well so that you can focus on evaluating different algorithms and thinking deeply about the problem.

The goal of the test harness is to be able to quickly and consistently test algorithms against a fair representation of the problem being solved. The outcome of testing multiple algorithms against the harness will be an estimation of how a variety of algorithms perform on the problem against a chosen performance measure. You will know which algorithms might be worth tuning on the problem and which should not be considered further.

The results will also give you an indication of how learnable the problem is. If a variety of different learning algorithms university perform poorly on the problem, it may be an indication of a lack of structure available to algorithms to learn. This may be because there actually is a lack of learnable structure in the selected data or it may be an opportunity to try different transforms to expose the structure to the learning algorithms.

Performance Measure

The performance measure is the way you want to evaluate a solution to the problem. It is the measurement you will make of the predictions made by a trained model on the test dataset.

Performance measures are typically specialized to the class of problem you are working with, for example classification, regression, and clustering. Many standard performance measures will give you a score that is meaningful to your problem domain. For example, classification accuracy for classification (total correct correction divided by the total predictions made multiple by 100 to turn it into a percentage).

You may also want a more detailed breakdown of performance, for example, you may want to know about the false positives on a spam classification problem because good email will be marked as spam and cannot be read.

There are many standard performance measures to choose from. You rarely have to devise a new performance measure yourself as you can generally find or adapt one that best captures the requirements of the problem being solved. Look to similar problems you uncovered and at the performance measures used to see if any can be adopted.

Test and Train Datasets

From the transformed data, you will need to select a test set and a training set. An algorithm will be trained on the training dataset and will be evaluated against the test set. This may be as simple as selecting a random split of data (66% for training, 34% for testing) or may involve more complicated sampling methods.

A trained model is not exposed to the test dataset during training and any predictions made on that dataset are designed to be indicative of the performance of the model in general. As such you want to make sure the selection of your datasets are representative of the problem you are solving.

Cross Validation

A more sophisticated approach than using a test and train dataset is to use the entire transformed dataset to train and test a given algorithm. A method you could use in your test harness that does this is called cross validation.

It first involves separating the dataset into a number of equally sized groups of instances (called folds). The model is then trained on all folds exception one that was left out and the prepared model is tested on that left out fold. The process is repeated so that each fold get's an opportunity at being left out and acting as the test dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem.

For example, a 3-fold cross validation would involve training and testing a model 3 times:

- #1: Train on folds 1+2, test on fold 3
- #2: Train on folds 1+3, test on fold 2
- #3: Train on folds 2+3, test on fold 1

The number of folds can vary based on the size of your dataset, but common numbers are 3, 5, 7 and 10 folds. The goal is to have a good balance between the size and representation of data in your train and test sets.

When you're just getting started, stick with a simple split of train and test data (such as 66%/34%) and move onto cross validation once you have more confidence.

Testing Algorithms

When starting with a problem and having defined a test harness you are happy with, it is time to spot check a variety of machine learning algorithms. Spot checking is useful because it allows you to very quickly see if there is any learnable structures in the data and estimate which algorithms may be effective on the problem.

Spot checking also helps you work out any issues in your test harness and make sure the chosen performance measure is appropriate.

The best first algorithm to spot check is a random. Plug in a random number generator to generate predictions in the appropriate range. This should be the worst "algorithm result" you achieve and will be the measure by which all improvements can be assessed.

Select 5-10 standard algorithms that are appropriate for your problem and run them through your test harness. By standard algorithms, I mean popular methods no special configurations. Appropriate for your problem means that the algorithms can handle regression if you have a regression problem.

Choose methods from the groupings of algorithms we have already reviewed. I like to include a diverse mix and have 10-20 different algorithms drawn from a diverse range of algorithm types. Depending on the library I am using, I may spot check up to a 50+ popular methods to flush out promising methods quickly.
If you want to run a lot of methods, you may have to revisit data preparation and reduce the size of your selected dataset. This may reduce your confidence in the results, so test with various data set sizes. You may like to use a smaller size dataset for algorithm spot checking and a fuller dataset for algorithm tuning.

Classification errors

There are multiple types of *errors* associated with machine learning and predictive analytics. The primary types are *in-sample* and *out-of-sample* errors. In-sample errors (aka *re-substitution errors*) are the error rate found from the training data, i.e., the data used to build predictive models.

Out-of-sample errors (aka *generalisation errors*) are the error rates found on a new data set, and are the most important since they represent the potential performance of a given predictive model on new and unseen data.

In-sample error rates may be very low and seem to be indicative of a high-performing model, but one must be careful, as this may be due to overfitting as mentioned, which would result in a model that is unable to generalise well to new data.

Training and validation data is used to build, validate, and tune a model, but test data is used to evaluate model performance and generalisation capability. One very important point to note is that prediction performance and error analysis should only be done on test data, when evaluating a model for use on non-training or new data (out-of-sample).

Generally speaking, model performance on training data tends to be optimistic, and therefore data errors will be less than those involving test data. There are tradeoffs between the types of errors that a machine learning practitioner must consider and often choose to accept.

For binary classification problems, there are two primary types of errors. *Type 1* errors (false positives) and *Type 2* errors (false negatives). It's often possible through model selection and tuning to increase one while decreasing the other, and often one must choose which error type is more acceptable. This can be a major tradeoff consideration depending on the situation.

A typical example of this tradeoff dilemma involves cancer diagnosis, where the positive diagnosis of having cancer is based on some test. In this case, a false positive means that someone is told that have have cancer when they do not. Conversely, the false negative case is when someone is told that they do not have cancer when they actually do. If no model is perfect, then in the example above, which is the more acceptable error type? In other words, of which one can we accept to a greater degree?

Telling someone they have cancer when they don't can result in tremendous emotional distress, stress, additional tests and medical costs, and so on. On the other hand, failing to detect cancer in someone that actually has it can mean the difference between life and death.

In the spam or ham case, neither error type is nearly as serious as the cancer case, but typically email vendors err slightly more on the side of letting some spam get into your inbox as opposed to you missing a very important email because the spam classifier is too aggressive.

MACHINE LEARNING
UNIT-3

Unsupervised learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses.

The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. The clusters are modelled using a measure of similarity which is defined upon metrics such as Euclidean or probabilistic distance.

Common clustering algorithms include:

- **Hierarchical clustering**: builds a multilevel hierarchy of clusters by creating a cluster tree
- **k-Means clustering**: partitions data into k distinct clusters based on distance to the centroid of a cluster
- **Gaussian mixture models**: models clusters as a mixture of multivariate normal density components
- **Self-organizing maps**: uses neural networks that learn the topology and distribution of the data
- **Hidden Markov models**: uses observed data to recover the sequence of states

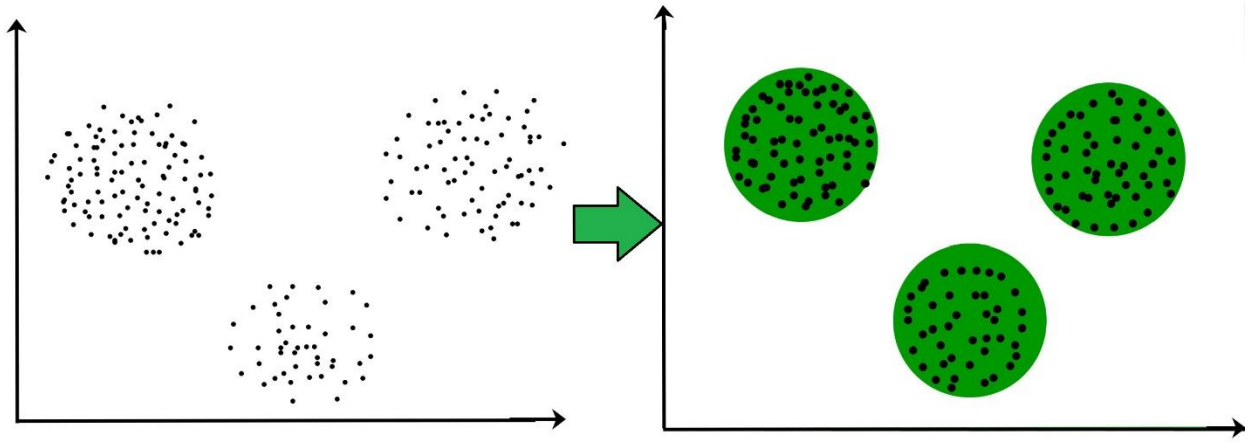
Unsupervised learning methods are used in bioinformatics for sequence analysis and genetic clustering; in data mining for sequence and pattern mining; in medical imaging for image segmentation; and in computer vision for object recognition.

Clustering

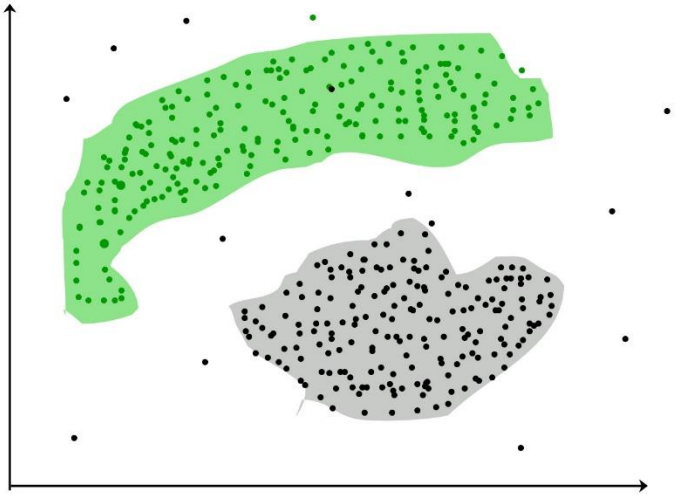
It is basically a type of *unsupervised learning method* . An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



It is not necessary for clusters to be a spherical. Such as :



DBSCAN: Density-based Spatial Clustering of Applications with Noise
These data points are clustered by using the basic concept that the data point lies within the given constraint from the cluster centre. Various distance methods and techniques are used for calculation of the outliers.

Why Clustering ?
Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding "natural clusters" and describe their unknown properties ("natural" data types), in finding useful and suitable groupings ("useful" data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods :

- **Density-Based Methods :** These methods consider the clusters as the dense region having some similarity and different from the lower dense region of the space. These methods have good

accuracy and ability to merge two clusters.Example *DBSCAN (Density-Based Spatial Clustering of Applications with Noise) , OPTICS (Ordering Points to Identify Clustering Structure)* etc.

- **Hierarchical Based Methods :** The clusters formed in this method forms a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category:
  - Agglomerative (*bottom up approach*)
  - Divisive (*top down approach*)

  Examples: *CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies)* etc.

- **Partitioning Methods :** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means, CLARANS (Clustering Large Applications based upon Randomized Search)* etc.

- **Grid-based Methods :** In this method the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operation done on these grids are fast and independent of the number of data objects example *STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest)* etc.

Applications of Clustering in different fields:

- **Marketing :** It can be used to characterize & discover customer segments for marketing purposes.
- **Biology :** It can be used for classification among different species of plants and animals.
- **Libraries :** It is used in clustering different books on the basis of topics and information.
- **Insurance :** It is used to acknowledge the customers, their policies and identifying the frauds.

City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones.

K-means algorithm

***k*-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. It is popular for cluster analysis in data mining. *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using k-medians and k-medoids.

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both *k-means* and *Gaussian mixture modeling*. They both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the *k*-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with *k*-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by *k*-means classifies new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

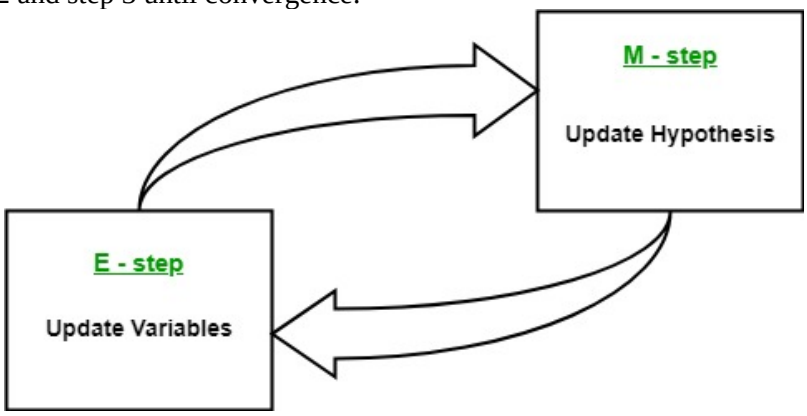
EM algorithm

In the real-world applications of machine learning, it is very common that there are many relevant features available for learning but only a small subset of them are observable. So, for the variables which are sometimes observable and sometimes not, then we can use the instances when that variable is visible is observed for the purpose of learning and then predict its value in the instances when it is not observable.

On the other hand, ***Expectation-Maximization algorithm*** can be used for the latent variables (variables that are not directly observable and are actually inferred from the values of the other observed variables) too in order to predict their values with the condition that the general form of probability distribution governing those latent variables is known to us. This algorithm is actually at the base of many unsupervised clustering algorithms in the field of machine learning.
It was explained, proposed and given its name in a paper published in 1977 by Arthur Dempster, Nan Laird, and Donald Rubin. It is used to find the *local maximum likelihood parameters* of a statistical model in the cases where latent variables are involved and the data is missing or incomplete.

Algorithm:

1. Given a set of incomplete data, consider a set of starting parameters.
2. **Expectation step (E – step):** Using the observed available data of the dataset, estimate (guess) the values of the missing data.
3. **Maximization step (M – step):** Complete data generated after the expectation (E) step is used in order to update the parameters.
4. Repeat step 2 and step 3 until convergence.

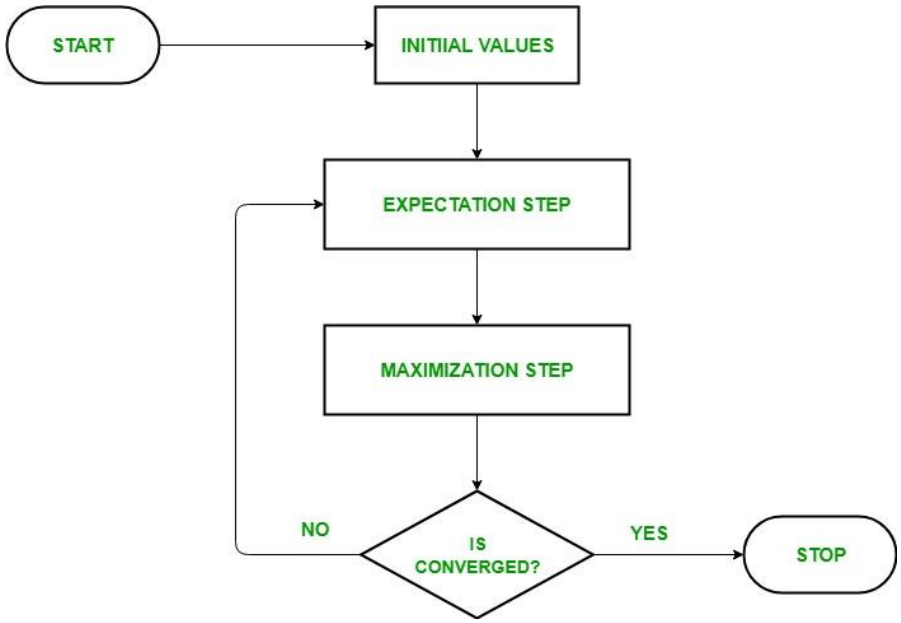


The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters. Let us understand the EM algorithm in detail.

- Initially, a set of initial values of the parameters are considered. A set of incomplete observed data is given to the system with the assumption that the observed data comes from a specific model.
- The next step is known as “Expectation” – step or *E-step*. In this step, we use the observed data in order to estimate or guess the values of the missing or incomplete data. It is basically used to update the variables.

- The next step is known as “Maximization”-step or *M-step*. In this step, we use the complete data generated in the preceding “Expectation” – step in order to update the values of the parameters. It is basically used to update the hypothesis.
- Now, in the fourth step, it is checked whether the values are converging or not, if yes, then stop otherwise repeat *step-2* and *step-3* i.e. “Expectation” – step and “Maximization” – step until the convergence occurs.

Flow chart for EM algorithm –



Usage of EM algorithm –

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm –

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

Disadvantages of EM algorithm –

- It has slow convergence.
- It makes convergence to the local optima only.
- It requires both the probabilities, forward and backward (numerical optimization requires only forward probability).

Mixture of Gaussians

Gaussian mixture models (GMMs) are often used for data clustering. You can use GMMs to perform either *hard* clustering or *soft* clustering on query data.

To perform *hard* clustering, the GMM assigns query data points to the multivariate normal components that maximize the component posterior probability, given the data. That is, given a fitted GMM, `cluster` assigns query data to the component yielding the highest posterior probability. Hard

clustering assigns a data point to exactly one cluster. For an example showing how to fit a GMM to data, cluster using the fitted model, and estimate component posterior probabilities, see Cluster Gaussian Mixture Data Using Hard Clustering.

Additionally, you can use a GMM to perform a more flexible clustering on data, referred to as *soft* (or *fuzzy*) clustering. Soft clustering methods assign a score to a data point for each cluster. The value of the score indicates the association strength of the data point to the cluster. As opposed to hard clustering methods, soft clustering methods are flexible because they can assign a data point to more than one cluster. When you perform GMM clustering, the score is the posterior probability. For an example of soft clustering with a GMM, see Cluster Gaussian Mixture Data Using Soft Clustering.

GMM clustering can accommodate clusters that have different sizes and correlation structures within them. Therefore, in certain applications,, GMM clustering can be more appropriate than methods such as *k*-means clustering. Like many clustering methods, GMM clustering requires you to specify the number of clusters before fitting the model. The number of clusters specifies the number of components in the GMM.

For GMMs, follow these best practices:

- Consider the component covariance structure. You can specify diagonal or full covariance matrices, and whether all components have the same covariance matrix.
- Specify initial conditions. The Expectation-Maximization (EM) algorithm fits the GMM. As in the *k*-means clustering algorithm, EM is sensitive to initial conditions and might converge to a local optimum. You can specify your own starting values for the parameters, specify initial cluster assignments for data points or let them be selected randomly, or specify use of the *k*-means++ algorithm.
- Implement regularization. For example, if you have more predictors than data points, then you can regularize for estimation stability.

Factor Analysis

Factor analysis is a technique that is used to reduce a large number of variables into fewer numbers of factors. This technique extracts maximum common variance from all variables and puts them into a common score. As an index of all variables, we can use this score for further analysis. Factor analysis is part of general linear model (GLM) and this method also assumes several assumptions: there is linear relationship, there is no multicollinearity, it includes relevant variables into analysis, and there is true correlation between variables and factors. Several methods are available, but principal component analysis is used most commonly.

Types of factoring:
There are different types of methods used to extract the factor from the data set:

1. **Principal component analysis:** This is the most common method used by researchers. PCA starts extracting the maximum variance and puts them into the first factor. After that, it removes that variance explained by the first factors and then starts extracting maximum variance for the second factor. This process goes to the last factor.
2. **Common factor analysis:** The second most preferred method by researchers, it extracts the common variance and puts them into factors. This method does not include the unique variance of all variables. This method is used in SEM.
3. **Image factoring:** This method is based on correlation matrix. OLS Regression method is used to predict the factor in image factoring.
4. **Maximum likelihood method:** This method also works on correlation metric but it uses maximum likelihood method to factor.

5. **Other methods of factor analysis:** Alfa factoring outweighs least squares. Weight square is another regression based method which is used for factoring.

Factor loading:

Factor loading is basically the correlation coefficient for the variable and factor. Factor loading shows the variance explained by the variable on that particular factor. In the SEM approach, as a rule of thumb, 0.7 or higher factor loading represents that the factor extracts sufficient variance from that variable.

Eigenvalues: Eigenvalues is also called characteristic roots. Eigenvalues shows variance explained by that particular factor out of the total variance. From the commonality column, we can know how much variance is explained by the first factor out of the total variance. For example, if our first factor explains 68% variance out of the total, this means that 32% variance will be explained by the other factor.

Factor score: The factor score is also called the component score. This score is of all row and columns, which can be used as an index of all variables and can be used for further analysis. We can standardize this score by multiplying a common term. With this factor score, whatever analysis we will do, we will assume that all variables will behave as factor scores and will move.

Criteria for determining the number of factors: According to the Kaiser Criterion, Eigenvalues is a good criteria for determining a factor. If Eigenvalues is greater than one, we should consider that a factor and if Eigenvalues is less than one, then we should not consider that a factor. According to the variance extraction rule, it should be more than 0.7. If variance is less than 0.7, then we should not consider that a factor.

Rotation method: Rotation method makes it more reliable to understand the output. Eigenvalues do not affect the rotation method, but the rotation method affects the Eigenvalues or percentage of variance extracted. There are a number of rotation methods available: (1) No rotation method, (2) Varimax rotation method, (3) Quartimax rotation method, (4) Direct oblimin rotation method, and (5) Promax rotation method. Each of these can be easily selected in SPSS, and we can compare our variance explained by those particular methods.

Assumptions:

1. **No outlier:** Assume that there are no outliers in data.
2. **Adequate sample size:** The case must be greater than the factor.
3. **No perfect multicollinearity:** Factor analysis is an interdependency technique. There should not be perfect multicollinearity between the variables.
4. **Homoscedasticity:** Since factor analysis is a linear function of measured variables, it does not require homoscedasticity between the variables.
5. **Linearity:** Factor analysis is also based on linearity assumption. Non-linear variables can also be used. After transfer, however, it changes into linear variable.
6. **Interval Data:** Interval data are assumed.

Key concepts and terms:

Exploratory factor analysis: Assumes that any indicator or variable may be associated with any factor. This is the most common factor analysis used by researchers and it is not based on any prior theory.

Confirmatory factor analysis (CFA): Used to determine the factor and factor loading of measured variables, and to confirm what is expected on the basic or pre-established theory. CFA assumes that each factor is associated with a specified subset of measured variables. It commonly uses two approaches:

1. **The traditional method:** Traditional factor method is based on principal factor analysis method rather than common factor analysis. Traditional method allows the researcher to know more about insight factor loading.
2. **The SEM approach:** CFA is an alternative approach of factor analysis which can be done in SEM. In SEM, we will remove all straight arrows from the latent variable, and add only that arrow which has to observe the variable representing the covariance between every pair of latents. We will also leave the

straight arrows error free and disturbance terms to their respective variables. If standardized error term in SEM is less than the absolute value two, then it is assumed good for that factor, and if it is more than two, it means that there is still some unexplained variance which can be explained by factor. Chi-square and a number of other goodness-of-fit indexes are used to test how well the model fits.

PCA (Principal components analysis)

The principal components of a collection of points in a real p-space are a sequence of {\displaystyle p}p direction vectors where the {\displaystyle i^{th}}i^{th} vector is the direction of a line that best fits the data while being orthogonal to the first {\displaystyle i-1}i-1 vectors. Here, a best-fitting line is defined as one that minimizes the average squared distance from the points to the line. These directions constitute an orthonormal basis in which different individual dimensions of the data are linearly uncorrelated. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.

PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The first principal component can equivalently be defined as a direction that maximizes the variance of the projected data. The {\displaystyle i^{th}}i^{th} principal component can be taken as a direction orthogonal to the first {\displaystyle i-1}i-1 principal components that maximizes the variance of the projected data.

From either objective, it can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. PCA is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset. Robust and L1-norm-based variants of standard PCA have also been proposed.

ICA (Independent components analysis)

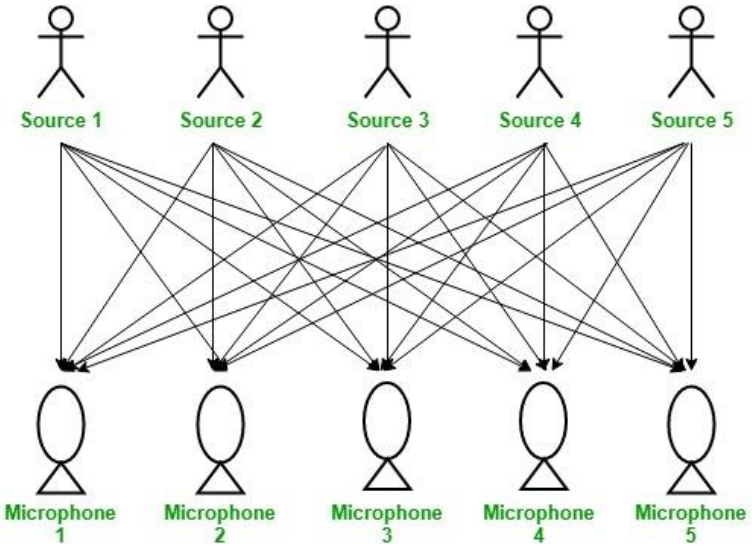
Independent Component Analysis (ICA) is a machine learning technique to separate independent sources from a mixed signal. Unlike principal component analysis which focuses on maximizing the variance of the data points, the independent component analysis focuses on independence, i.e. independent components.

Problem: To extract independent sources' signals from a mixed signal composed of the signals from those sources.
Given: Mixed signal from five different independent sources.
Aim: To decompose the mixed signal into independent sources:

- Source 1
- Source 2
- Source 3

- Source 4
- Source 5

Solution: Independent Component Analysis (ICA).
Consider *Cocktail Party Problem* or *Blind Source Separation* problem to understand the problem which is solved by independent component analysis.



Here, There is a party going into a room full of people. There is 'n' number of speakers in that room and they are speaking simultaneously at the party. In the same room, there are also 'n' number of microphones placed at different distances from the speakers which are recording 'n' speakers' voice signals. Hence, the number of speakers is equal to the number must of microphones in the room.
Now, using these microphones' recordings, we want to separate all the 'n' speakers' voice signals in the room given each microphone recorded the voice signals coming from each speaker of different intensity due to the difference in distances between them. Decomposing the mixed signal of each microphone's recording into independent source's speech signal can be done by using the machine learning technique, independent component analysis.
[X1, X2, ..., Xn] => [Y1, Y2, ..., Yn]
where, X1, X2, …, Xn are the original signals present in the mixed signal and Y1, Y2, …, Yn are the new features and are independent components which are independent of each other.

Restrictions on ICA –

1. The independent components generated by the ICA are assumed to be statistically independent of each other.
2. The independent components generated by the ICA must have non-gaussian distribution.
3. The number of independent components generated by the ICA is equal to the number of observed mixtures.

Difference between PCA and ICA –

PRINCIPAL COMPONENT ANALYSIS	INDEPENDENT COMPONENT ANALYSIS
It reduces the dimensions to avoid the problem of overfitting.	It decomposes the mixed signal into its independent sources' signals.

PRINCIPAL COMPONENT ANALYSIS	INDEPENDENT COMPONENT ANALYSIS
It deals with the Principal Components.	It deals with the Independent Components.
It focuses on maximizing the variance.	It doesn't focus on the issue of variance among the data points.
It focuses on the mutual orthogonality property of the principal components.	It doesn't focus on the mutual orthogonality of the components.
It doesn't focus on the mutual independence of the components.	It focuses on the mutual independence of the components.

Latent Semantic Indexing

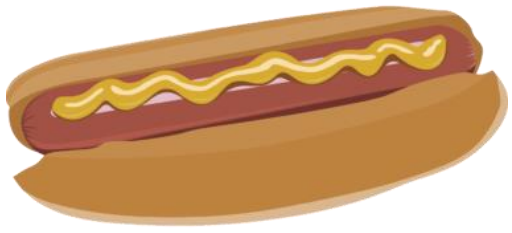
Latent semantic indexing, sometimes referred to as latent semantic analysis, is a mathematical method developed in the late 1980s to improve the accuracy of information retrieval. It uses a technique called singular value decomposition to scan unstructured data within documents and identify relationships between the concepts contained therein.

In essence, it finds the hidden (latent) relationships between words (semantics) in order to improve information understanding (indexing).

It provided a significant step forward for the field of text comprehension as it accounted for the contextual nature of language.

Earlier technologies struggled with the use of synonyms that characterizes natural language use, and also the changes in meanings that come with new surroundings.

For example, the words 'hot' and 'dog' may seem easy to understand, but both have multiple definitions based on how they are used. Put both of them together and you have a whole new concept altogether.



So how can we train a machine to adapt to these nuances?

This is a problem that has troubled great minds for centuries and LSI has helped computers to start understanding language in use.

It works best on static content and on small sets of documents, which was great for its initial purposes. LSI also allows documents to be clustered together based on their thematic commonalities, which was a very useful capability for early search engines.

Latent semantic indexing can be summarized as follows:

- A technology developed in the late 1980s for information retrieval, in response to earlier technologies that could not understand synonymy or polysemy.
- A specific approach that tries to grasp the underlying structure of meaning in language.
- Capable of inducing from these findings the hierarchical categories into which terms and concepts fall.
- Originally useful for working on small sets of static documents.

Spectral clustering

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. The method is flexible and allows us to cluster non graph data as well.

Spectral clustering uses information from the eigenvalues (spectrum) of special matrices built from the graph or the data set. We'll learn how to construct these matrices, interpret their spectrum, and use the eigenvectors to assign our data to clusters.

In multivariate statistics and the clustering of data, spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset.

In application to image segmentation, spectral clustering is known as segmentation-based object categorization.

Markov models: Hidden Markov models (HMMs)

The Hidden Markov Model (HMM) is a relatively simple way to model sequential data. A *hidden* Markov model implies that the Markov Model underlying the data is hidden or unknown to you. More specifically, you only know observational data and not information about the states. In other words, there's a specific type of model that produces the data (a Markov Model) but you don't know what processes are producing it. You basically use your knowledge of Markov Models to make an educated guess about the model's structure.

What is a Markov Model?

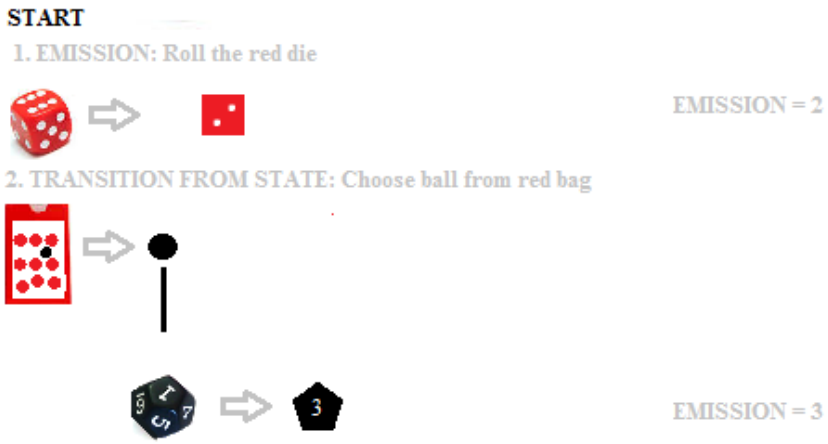
In order to uncover the Hidden Markov Model, you first have to understand what a Markov Model is in the first place. Here I'll create a simple example using two items that are very familiar in probability: dice and bags of colored balls.

The model components, which you'll use to create the random model, are:

- A six-sided red die.
- A ten-sided black die.
- A red bag with ten balls. Nine balls are red, one is black.
- A black bag with twenty balls. One ball is red, nineteen are black.

"Black" and "Red" are the two states in this model (in other words, you can be black, or you can be red).

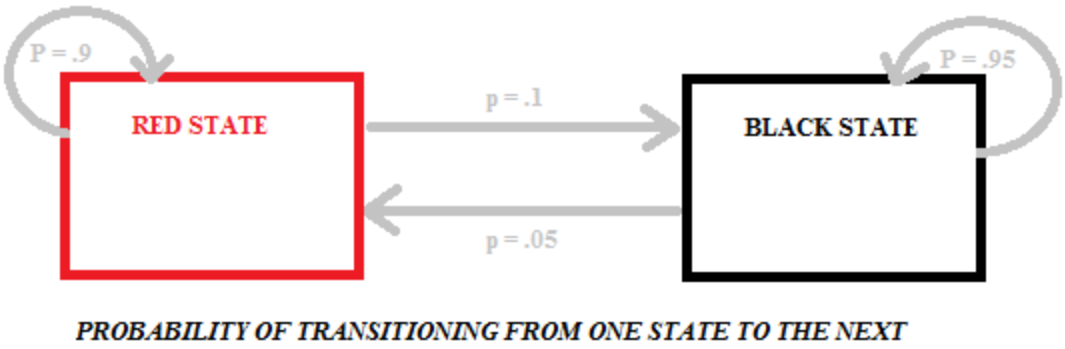
Now create the model by following these steps:



1. **EMISSION STEP:** Roll a die. Note the number that comes up. This is the ***emission***. In the above graphic, I chose a red die to start (arbitrary — I could have chosen black) and rolled 2.
2. **TRANSITION STEP**: Randomly choose a ball from the bag with the color that matches the die you rolled in step 1. I rolled a red die, so I'm going to choose a ball from the red bag. I pulled out a black ball, so I'm going to transition to the black die for the next emission.

You can then repeat these steps to a certain number of emissions. For example, repeating this sequence of steps 10 times might give you the set {2,3,6,1,1,4,5,3,4,1}. **The process of transitioning from one *state* to the next is called a *Markov process*.**

Transitioning from red to black or black to red carries different probabilities as there are different numbers of black and red balls in the bags. The following diagram shows the probabilities for this particular model, which has two states (black and red):



PROBABILITY OF TRANSITIONING FROM ONE STATE TO THE NEXT

Hidden Markov Model Notation

λ = (A,B,π), is shorthand notation for an HMM. Other notation is used in Hidden Markov Models:

- A = state transition probabilities (a_{ij})
- B = observation probability matrix ($b_j(k)$)
- N = number of states in the model {1,2…N} or the state at time t →s_t
- M = number of distinct observation symbols per state
- Q = {q0, q1, . . . , qN−1} = distinct states of the Markov process
- T = length of the observation sequence
- V = {0, 1, . . . , M − 1} = set of possible observations
- O = (O_0, O_1, . . . , O_{T-1}) = observation sequence
- π = initial state distribution (π_i)
- s = state or state sequence (s_1, s_2… s_n)
- x_k = hidden state
- $z_{k\,=\,\text{observation.}}$

Three Basic Problems

Three basic problems can be solved with Hidden Markov Models:

1. Given the Hidden Markov Modelλ = (A, B, π) and a sequence of observations O, find the probability of an observation P(O | λ). This is sometimes called the **Evaluation Problem**.
2. Given the Hidden Markov Modelλ = (A, B, π) and an observation sequence O, find the most likely state sequence (s_1, s_2… s_n). This is sometimes called a **Decoding Problem.**
3. Find an observation sequence (O_1, O_2…O_n and Hidden Markov Model λ = (A, B, π) that maximizes the probability of O. This is sometimes called a **Learning Problem or Optimization Problem**.

MACHINE LEARNING
UNIT-4

Reinforcement Learning and Control

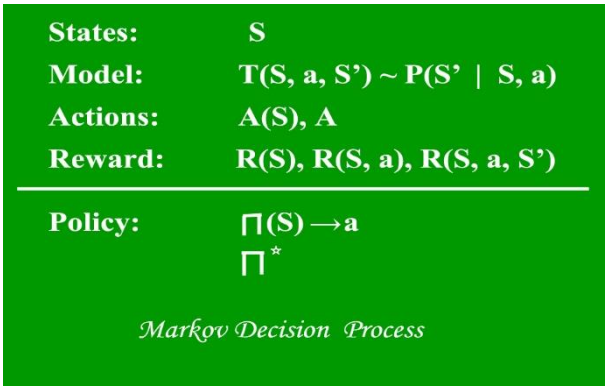
Reinforcement learning (RL) offers powerful algorithms to search for optimal controllers of systems with nonlinear, possibly stochastic dynamics that are unknown or highly uncertain. This review mainly covers artificial-intelligence approaches to RL, from the viewpoint of the control engineer. We explain how approximate representations of the solution make RL feasible for problems with continuous states and control actions. Stability is a central concern in control, and we argue that while the control-theoretic RL subfield called adaptive dynamic programming is dedicated to it, stability of RL largely remains an open question. We also cover in detail the case where deep neural networks are used for approximation, leading to the field of deep RL, which has shown great success in recent years. With the control practitioner in mind, we outline opportunities and pitfalls of deep RL; and we close the survey with an outlook that – among other things – points out some avenues for bridging the gap between control and artificial-intelligence RL techniques.

MDPs (Markov Decision Processes)

Reinforcement Learning is a type of Machine Learning. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed to decide the best action to select based on his current state. When this step is repeated, the problem is known as a **Markov Decision Process**.
A **Markov Decision Process (MDP)** model contains:

- A set of possible world states S.
- A set of Models.
- A set of possible actions A.
- A real valued reward function R(s,a).
- A policy the solution of **Markov Decision Process**.



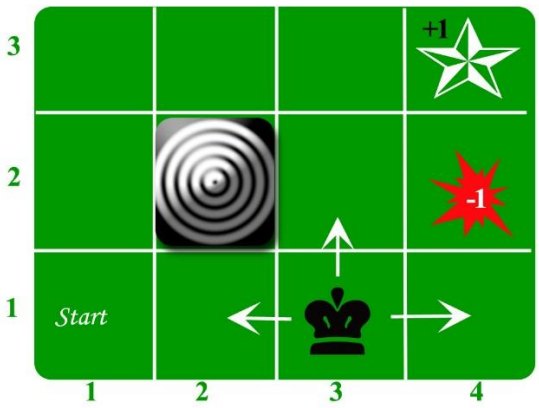
What is a State?
A **State** is a set of tokens that represent every state that the agent can be in.

What is a Model?
A **Model** (sometimes called Transition Model) gives an action's effect in a state. In particular, T(S, a, S') defines a transition T where being in state S and taking an action 'a' takes us to state S' (S and S' may be same). For stochastic actions (noisy, non-deterministic) we also define a probability P(S'|S,a) which represents the probability of reaching a state S' if action 'a' is taken in state S. Note Markov property states that the effects of an action taken in a state depend only on that state and not on the prior history.

What is Actions?
An **Action** A is set of all possible actions. A(s) defines the set of actions that can be taken being in state S.

What is a Reward?
A **Reward** is a real-valued reward function. R(s) indicates the reward for simply being in the state S. R(S,a) indicates the reward for being in a state S and taking an action 'a'. R(S,a,S') indicates the reward for being in a state S, taking an action 'a' and ending up in a state S'.

What is a Policy?
A **Policy** is a solution to the Markov Decision Process. A policy is a mapping from S to a. It indicates the action 'a' to be taken while in state S.
Let us take the example of a grid world:



An agent lives in the grid. The above example is a 3*4 grid. The grid has a START state(grid no 1,1). The purpose of the agent is to wander around the grid to finally reach the Blue Diamond (grid no 4,3). Under all circumstances, the agent should avoid the Fire grid (orange color, grid no 4,2). Also the grid no 2,2 is a blocked grid, it acts like a wall hence the agent cannot enter it.

The agent can take any one of these actions: **UP, DOWN, LEFT, RIGHT**
Walls block the agent path, i.e., if there is a wall in the direction the agent would have taken, the agent stays in the same place. So for example, if the agent says LEFT in the START grid he would stay put in the START grid.

First Aim: To find the shortest sequence getting from START to the Diamond. Two such sequences can be found:

- **RIGHT RIGHT UP UP RIGHT**
- **UP UP RIGHT RIGHT RIGHT**

Let us take the second one (UP UP RIGHT RIGHT RIGHT) for the subsequent discussion.
The move is now noisy. 80% of the time the intended action works correctly. 20% of the time the action agent takes causes it to move at right angles. For example, if the agent says UP the probability of going UP is 0.8 whereas the probability of going LEFT is 0.1 and probability of going RIGHT is 0.1 (since LEFT and RIGHT is right angles to UP).

The agent receives rewards each time step:-

- Small reward each step (can be negative when can also be term as punishment, in the above example entering the Fire can have a reward of -1).
- Big rewards come at the end (good or bad).
- The goal is to Maximize sum of rewards.

Bellman equations

If you have read anything related to reinforcement learning you must have encountered bellman equation somewhere. Bellman equation is the basic block of solving reinforcement learning and is omnipresent in RL. It helps us to solve MDP. To solve means finding the optimal policy and value functions.

The optimal value function V*(S) is one that yields maximum value.

The value of a given state is equal to the max action (action which maximizes the value) of the reward of the optimal action in the given state and add a discount factor multiplied by the next state's Value from the Bellman Equation.

$$V(s) = max_a(R(s,a) + \gamma V(s'))$$

Bellman equation for deterministic environment

Let's understand this equation, V(s) is the value for being in a certain state. V(s') is the value for being in the next state that we will end up in after taking action a. R(s, a) is the reward we get after taking action a in state s. As we can take different actions so we use maximum because our agent wants to be in the optimal state. γ is the discount factor as discussed earlier. This is the bellman equation in the deterministic environment (discussed in part 1). It will be slightly different for a non-deterministic environment or stochastic environment.

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \right)$$

Bellman equation for stochastic environment

In a stochastic environment when we take an action it is not confirmed that we will end up in a particular next state and there is a probability of ending in a particular state. P(s, a,s') is the probability of ending is states' from s by taking action a. This is summed up to a total number of future states. For eg, if by taking an action we can end up in 3 states s_1, s_2 and s_3 from states' with a probability of 0.2, 0.2 and 0.6. The Bellman equation will be:

$$V(s) = max_a(R(s,a) + \gamma(0.2*V(s_1) + 0.2*V(s_2) + 0.6*V(s_3)))$$

We can solve the Bellman equation using a special technique called dynamic programming.

Value iteration and policy iteration

Both value and policy iteration work around The Bellman Equations where we find the optimal utility.

The Bellman Equation is given as :

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

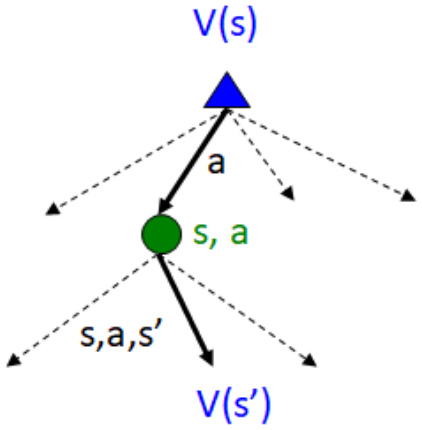
Now , Both value iteration and policy iteration compute the same thing (all optimal values) i.e they work with Bellman updates.

Then how they are different

1. Value iteration is simpler but its computationally heavy.
2. Policy iteration is complicated but its computationally cheap w.r.t value iteration.
3. **Value iteration** includes: **finding optimal value function** + one **policy extraction**. There is no repeat of the two because once the value function is optimal, then the policy out of it should also be optimal (i.e. converged).
4. **Policy iteration** includes: **policy evaluation** + **policy improvement**, and the two are repeated iteratively until policy converges.

Value Iteration :

We start with a random value function and then find a new (improved) value function in a iterative process, until reaching the optimal value function.



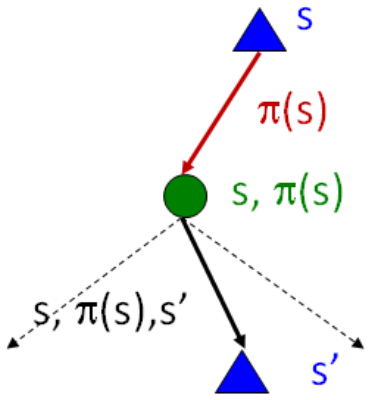
Value iteration computation of Bellman Equation:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Problems with Value Iteration :

1. It's slow – O(S^2A) per iteration
2. The "max" at each state rarely changes
3. The policy often converges long before the values

Policy Iteration :



Policy iteration computations of Bellman Equation:

1. **Evaluation:** For fixed current policy p, find values with policy evaluation.
$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$
2. **Improvement:** For fixed values, get a better policy using policy extraction (One-step look-ahead).
$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Now overcoming with problems of value iteration , We first start with policy iteration where we have fixed policy at start (don't worry whether it is optimal or not.)

Now since we have fixed our polity the actions that we took in value iteration aren't necessary now (i.e. complexity of policy iteration is O(S^2) per iteration). *NOTE : we have relaxed A(i.e actions here).*

Now for policy iteration :

1. **We evaluate the policy evaluation :** i.e. we calculate utilities for some fixed policy (not optimal utilities!) until convergence
2. **We improve the policy**: i.e update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
3. Repeat step 1 and 2 until convergence

Now this convergence of policy iteration is much faster than value iteration.

CONCLUSION :

In value iteration:

- Every iteration updates both the values and (implicitly) the policy
- We don't track the policy, but taking the max over actions implicitly recomputes it

In policy iteration:

- We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
- After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
- The new policy will be better (or we're done)

Both are dynamic programs for solving MDPs.

Linear quadratic regulation (LQR)

The theory of optimal control is concerned with operating a dynamic system at minimum cost. The case where the system dynamics are described by a set of linear differential equations and the cost is described by a quadratic function is called the LQ problem. One of the main results in the theory is that the solution is provided by the **linear–quadratic regulator** (**LQR**), a feedback controller whose equations are given below. The LQR is an important part of the solution to the LQG (linear–quadratic–Gaussian) problem. Like the LQR problem itself, the LQG problem is one of the most fundamental problems in control theory.

The settings of a (regulating) controller governing either a machine or process (like an airplane or chemical reactor) are found by using a mathematical algorithm that minimizes a cost function with weighting factors supplied by a human (engineer). The cost function is often defined as a sum of the deviations of key measurements, like altitude or process temperature, from their desired values. The algorithm thus finds those controller settings that minimize undesired deviations. The magnitude of the control action itself may also be included in the cost function.

The LQR algorithm reduces the amount of work done by the control systems engineer to optimize the controller. However, the engineer still needs to specify the cost function parameters, and compare the results with the specified design goals. Often this means that controller construction will be an iterative process in which the engineer judges the "optimal" controllers produced through simulation and then adjusts the parameters to produce a controller more consistent with design goals.

The LQR algorithm is essentially an automated way of finding an appropriate state-feedback controller. As such, it is not uncommon for control engineers to prefer alternative methods, like full state feedback, also known as pole placement, in which there is a clearer relationship between controller parameters and controller behavior. Difficulty in finding the right weighting factors limits the application of the LQR based controller synthesis.

LQG (Linear Quadratic Gaussian control)

In control theory, the **linear–quadratic–Gaussian** (**LQG**) **control problem** is one of the most fundamental optimal control problems. It concerns linear systems driven by additive white Gaussian noise. The problem is to determine an output feedback law that is optimal in the sense of minimizing

the expected value of a quadratic cost criterion. Output measurements are assumed to be corrupted by Gaussian noise and the initial state, likewise, is assumed to be a Gaussian random vector.

Under these assumptions an optimal control scheme within the class of linear control laws can be derived by a completion-of-squares argument.[1] This control law which is known as the **LQG** controller, is unique and it is simply a combination of a Kalman filter (a linear–quadratic state estimator (LQE)) together with a linear–quadratic regulator (LQR). The separation principle states that the state estimator and the state feedback can be designed independently. LQG control applies to both linear time-invariant systems as well as linear time-varying systems, and constitutes a linear dynamic feedback control law that is easily computed and implemented: the LQG controller itself is a dynamic system like the system it controls. Both systems have the same state dimension.

A deeper statement of the separation principle is that the LQG controller is still optimal in a wider class of possibly nonlinear controllers. That is, utilizing a nonlinear control scheme will not improve the expected value of the cost functional. This version of the separation principle is a special case of the separation principle of stochastic control which states that even when the process and output noise sources are possibly non-Gaussian martingales, as long as the system dynamics are linear, the optimal control separates into an optimal state estimator (which may no longer be a Kalman filter) and an LQR regulator.

In the classical LQG setting, implementation of the LQG controller may be problematic when the dimension of the system state is large. The **reduced-order LQG problem** (fixed-order LQG problem) overcomes this by fixing *a priori* the number of states of the LQG controller. This problem is more difficult to solve because it is no longer separable. Also, the solution is no longer unique. Despite these facts numerical algorithms are available to solve the associated optimal projection equations which constitute necessary and sufficient conditions for a locally optimal reduced-order LQG controller.

LQG optimality does not automatically ensure good robustness properties. The robust stability of the closed loop system must be checked separately after the LQG controller has been designed. To promote robustness some of the system parameters may be assumed stochastic instead of deterministic. The associated more difficult control problem leads to a similar optimal controller of which only the controller parameters are different.

It is possible to compute the expected value of the cost function for the optimal gains, as well as any other set of stable gains.

Finally, the LQG controller is also used to control perturbed non-linear systems.

Q-learning

***Q*-learning** is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.

For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.[1] "Q" names the function that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in a given state.

Value function approximation

In general, a function approximation problem asks us to select a function among a well-defined class that closely matches ("approximates") a **target function** in a task-specific way. The need for **function approximations** arises in many branches of applied mathematics, and computer science in particular.

One can distinguish two major classes of function approximation problems:

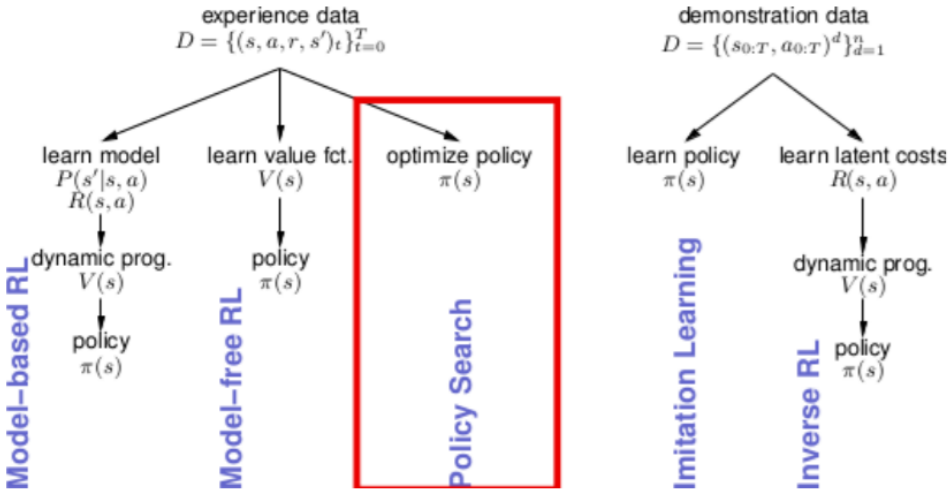
First, for known target functions approximation theory is the branch of numerical analysis that investigates how certain known functions (for example, special functions) can be approximated by a specific class of functions (for example, polynomials or rational functions) that often have desirable properties (inexpensive computation, continuity, integral and limit values, etc.).

Second, the target function, call it *g*, may be unknown; instead of an explicit formula, only a set of points of the form (*x*, *g*(*x*)) is provided. Depending on the structure of the domain and codomain of *g*, several techniques for approximating *g* may be applicable. For example, if *g* is an operation on the real numbers, techniques of interpolation, extrapolation, regression analysis, and curve fitting can be used. If the codomain (range or target set) of *g* is a finite set, one is dealing with a classification problem instead.

To some extent, the different problems (regression, classification, fitness approximation) have received a unified treatment in statistical learning theory, where they are viewed as supervised learning problems.

Policy Search

- Learning in high-dimensional space problems:
  – Using value function gives poor performance (e.g humanoid robots have 35 DoF).
  – Mobel-based learning is expensive.



Reinforce

REINFORCE is a Monte-Carlo variant of policy gradients (Monte-Carlo: taking random samples). The agent collects a trajectory τ of one episode using its current policy, and uses it to update the policy parameter. Since one full trajectory must be completed to construct a sample space, REINFORCE is updated in an off-policy way.

Here is the pseudo code for REINFORCE :

function REINFORCE
Initialise θ arbitrarily
for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
for $t = 1$ to $T - 1$ **do**
$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
end for
end for
return θ
end function

So, the flow of the algorithm is:

1. Perform a trajectory roll-out using the current policy
2. Store log probabilities (of policy) and reward values at each step
3. Calculate discounted cumulative future reward at each step
4. Compute policy gradient and update policy parameter
5. Repeat 1–4

POMDPs (Partially Observable Markov Decision Processes)

A **partially observable Markov decision process** (**POMDP**) is a generalization of a Markov decision process (MDP). A POMDP models an agent decision process in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state. Instead, it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP.

The POMDP framework is general enough to model a variety of real-world sequential decision processes. Applications include robot navigation problems, machine maintenance, and planning under uncertainty in general. The general framework of Markov decision processes with imperfect information was described by Karl Johan Åström in 1965 in the case of a discrete state space, and it was further studied in the operations research community where the acronym POMDP was coined. It was later adapted for problems in artificial intelligence and automated planning by Leslie P. Kaelbling and Michael L. Littman.

An exact solution to a POMDP yields the optimal action for each possible belief over the world states. The optimal action maximizes (or minimizes) the expected reward (or cost) of the agent over a possibly infinite horizon. The sequence of optimal actions is known as the optimal policy of the agent for interacting with its environment.