**Machine Learning**

**Machine learning**, a branch of artificial intelligence, is about the construction and study of systems that can learn from data. For example, a machine learning system could be trained on email messages to learn to distinguish between spam and non-spam messages. After learning, it can then be used to classify new email messages into spam and non-spam folders.

The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems; for example, in the above email message example we can represent an email as a set of English words by simply discarding the word order. Generalization is the property that the system will perform well on unseen data instances; the conditions under which this can be guaranteed are a key object of study in the subfield of computational learning theory.

There is a wide variety of machine learning tasks and successful applications. Optical character recognition, in which printed characters are recognized automatically based on previous examples, is a classic engineering example of machine learning

**Supervised Machine Learning**

**Supervised learning** is the [machine learning](http://en.wikipedia.org/wiki/Machine_learning) task of inferring a function from labeled training data. The [training data](http://en.wikipedia.org/wiki/Training_set) consist of a set of *training examples*. In supervised learning, each example is a *pair* consisting of an input object (typically a vector) and a desired output value (also called the *supervisory signal*). A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a *classifier.* The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. The parallel task in human and animal psychology is often referred to as [concept learning](http://en.wikipedia.org/wiki/Concept_learning).

**Overview**

In order to solve a given problem of supervised learning, one has to perform the following steps:

1. Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In the case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
2. Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the [curse of dimensionality](http://en.wikipedia.org/wiki/Curse_of_dimensionality); but should contain enough information to accurately predict the output.
4. Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or [decision trees](http://en.wikipedia.org/wiki/Decision_tree_learning).
5. Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a *validation* set) of the training set, or via [cross-validation](http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29).
6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

A wide range of supervised learning algorithms is available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems

**Unsupervised Machine Learning**

In [machine learning](http://en.wikipedia.org/wiki/Machine_learning), **unsupervised learning** refers to the problem of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from [supervised learning](http://en.wikipedia.org/wiki/Supervised_learning). Unsupervised learning is closely related to the problem of [density estimation](http://en.wikipedia.org/wiki/Density_estimation) in [statistics](http://en.wikipedia.org/wiki/Statistics). However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data. Many methods employed in unsupervised learning are based on [data mining](http://en.wikipedia.org/wiki/Data_mining) methods used to preprocess data.

Approaches to unsupervised learning include:

* [clustering](http://en.wikipedia.org/wiki/Data_clustering) (e.g., [k-means](http://en.wikipedia.org/wiki/K-means), [mixture models](http://en.wikipedia.org/wiki/Mixture_models), [hierarchical clustering](http://en.wikipedia.org/wiki/Hierarchical_clustering)),
* [blind signal separation](http://en.wikipedia.org/wiki/Blind_signal_separation) using [feature extraction](http://en.wikipedia.org/wiki/Feature_extraction) techniques for [dimensionality reduction](http://en.wikipedia.org/wiki/Dimensionality_reduction) (e.g., [Principal component analysis](http://en.wikipedia.org/wiki/Principal_component_analysis), [Independent component analysis](http://en.wikipedia.org/wiki/Independent_component_analysis), [Non-negative matrix factorization](http://en.wikipedia.org/wiki/Non-negative_matrix_factorization), [Singular value decomposition](http://en.wikipedia.org/wiki/Singular_value_decomposition)). [[2]](http://en.wikipedia.org/wiki/Unsupervised_learning#cite_note-2)

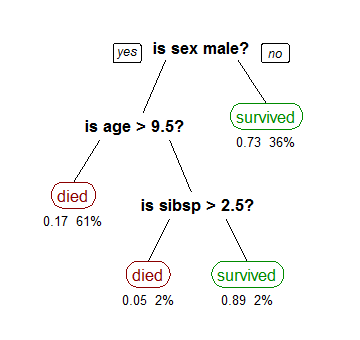
**Decision Tree**

Decision trees are a simple, but powerful form of multiple variable analyses. They

provide unique capabilities to supplement, complement, and substitute for

* traditional statistical forms of analysis (such as multiple linear regression)
* a variety of data mining tools and techniques (such as neural networks)
* recently developed multidimensional forms of reporting and analysis found in the field of business intelligence

Decision trees are produced by algorithms that identify various ways of splitting a data set into branch-like segments. These segments form an inverted decision tree that originates with a root node at the top of the tree. The object of analysis is reflected in this root node as a simple, one-dimensional display in the decision tree interface. The name of the field of data that is the object of analysis is usually displayed, along with the spread or distribution of the values that are contained in that field. A sample decision tree is illustrated in Figure 1.1



***Figure 1.1.***

**Decision Tree Advantages**

Amongst other data mining methods, decision trees have various advantages:

* **Simple to understand and interpret.** People are able to understand decision tree models after a brief explanation.
* **Requires little data preparation.** Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
* **Able to handle both numerical and categorical data.** Other techniques are usually specialised in analysing datasets that have only one type of variable. Ex: relation rules can be used only with nominal variables while neural networks can be used only with numerical variables.
* **Uses a** [**white box**](http://en.wikipedia.org/wiki/White_box_%28software_engineering%29) **model.** If a given situation is observable in a model the explanation for the condition is easily explained by boolean logic. An example of a black box model is an [artificial neural network](http://en.wikipedia.org/wiki/Artificial_neural_network) since the explanation for the results is difficult to understand.
* **Possible to validate a model using statistical tests.** That makes it possible to account for the reliability of the model.
* [**Robust**](http://en.wikipedia.org/wiki/Robust_statistics)**.** Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
* **Performs well with large data in a short time.** Large amounts of data can be analysed using standard computing resources.

**Decision Tree Limitations**

* The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree.
* Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as [pruning](http://en.wikipedia.org/wiki/Pruning_%28decision_trees%29) are necessary to avoid this problem.
* There are concepts that are hard to learn because decision trees do not express them easily, such as [XOR](http://en.wikipedia.org/wiki/XOR), [parity](http://en.wikipedia.org/wiki/Parity_bit#Parity) or [multiplexer](http://en.wikipedia.org/wiki/Multiplexer) problems. In such cases, the decision tree becomes prohibitively large. Approaches to solve the problem involve either changing the representation of the problem domain (known as propositionalisation)[[13]](http://en.wikipedia.org/wiki/Decision_tree_learning#cite_note-13) or using learning algorithms based on more expressive representations (such as [statistical relational learning](http://en.wikipedia.org/wiki/Statistical_relational_learning) or [inductive logic programming](http://en.wikipedia.org/wiki/Inductive_logic_programming)).
* For data including categorical variables with different numbers of levels, [information gain in decision trees](http://en.wikipedia.org/wiki/Information_gain_in_decision_trees) is biased in favor of those attributes with more levels.

**K-Means Clustering**

**The Algorithm**

K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids shoud be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more.  
Finally, this algorithm aims at minimizing an objective function, in this case a squared error function. The objective function

http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/images/image009.gif

where http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/images/image011.gifis a chosen distance measure between a data point http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/images/image013.gifand the cluster centre http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/images/image015.gif, is an indicator of the distance of the n data points from their respective cluster centers.

The algorithm is composed of the following steps:

|  |
| --- |
| 1. *Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.* 2. *Assign each object to the group that has the closest centroid.* 3. *When all objects have been assigned, recalculate the positions of the K centroids.* 4. *Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.* |

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect.

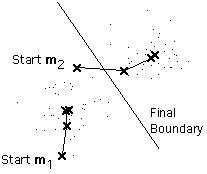
K-means is a simple algorithm that has been adapted to many problem domains. As we are going to see, it is a good candidate for extension to work with fuzzy feature vectors.

**An example**

Suppose that we have n sample feature vectors **x**1, **x**2, ..., **x**n all from the same class, and we know that they fall into k compact clusters, k < n. Let **m**i be the mean of the vectors in cluster i. If the clusters are well separated, we can use a minimum-distance classifier to separate them. That is, we can say that **x** is in cluster i if || **x** - **m**i || is the minimum of all the k distances. This suggests the following procedure for finding the k means:

* Make initial guesses for the means **m**1, **m**2, ..., **m**k
* Until there are no changes in any mean
  + Use the estimated means to classify the samples into clusters
  + For i from 1 to k
    - Replace **m**i with the mean of all of the samples for cluster i
  + end\_for
* end\_until

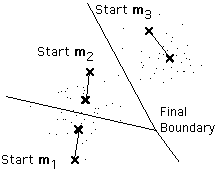
Here is an example showing how the means **m**1 and **m**2 move into the centers of two clusters.



**Remarks**  
This is a simple version of the k-means procedure. It can be viewed as a greedy algorithm for partitioning the n samples into k clusters so as to minimize the sum of the squared distances to the cluster centers. It does have some weaknesses:

* The way to initialize the means was not specified. One popular way to start is to randomly choose k of the samples.
* The results produced depend on the initial values for the means, and it frequently happens that suboptimal partitions are found. The standard solution is to try a number of different starting points.
* It can happen that the set of samples closest to **m**i is empty, so that **m**i cannot be updated. This is an annoyance that must be handled in an implementation, but that we shall ignore.
* The results depend on the metric used to measure || **x** - **m**i ||. A popular solution is to normalize each variable by its standard deviation, though this is not always desirable.
* The results depend on the value of k.

This last problem is particularly troublesome, since we often have no way of knowing how many clusters exist. In the example shown above, the same algorithm applied to the same data produces the following 3-means clustering. Is it better or worse than the 2-means clustering?



Unfortunately there is no general theoretical solution to find the optimal number of clusters for any given data set. A simple approach is to compare the results of multiple runs with different k classes and choose the best one according to a given criterion (for instance the Schwarz Criterion - see [Moore's slides](http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html#moore)), but we need to be careful because increasing k results in smaller error function values by definition, but also an increasing risk of over-fitting.

**References for K-Means Clustering**

* J. B. MacQueen (1967): "Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press, 1:281-297
* Andrew Moore: “K-means and Hierarchical Clustering - Tutorial Slides”  
  [http://www-2.cs.cmu.edu/~awm/tutorials/kmeans.html](http://www-2.cs.cmu.edu/%7Eawm/tutorials/kmeans.html)
* Brian T. Luke: “K-Means Clustering”  
  [http://fconyx.ncifcrf.gov/~lukeb/kmeans.html](http://fconyx.ncifcrf.gov/%7Elukeb/kmeans.html)
* Tariq Rashid: “Clustering”  
  <http://www.cs.bris.ac.uk/home/tr1690/documentation/fuzzy_clustering_initial_report/node11.html>
* Hans-Joachim Mucha and Hizir Sofyan: “Nonhierarchical Clustering”  
  [http://www.quantlet.com/mdstat/scripts/xag/html/xaghtmlframe149.ht](http://www.quantlet.com/mdstat/scripts/xag/html/xaghtmlframe149.html)
* PLUS material taught in the Class

**Search Algorithms**

**Blind Search**

* Depth First Search
  + Iterative DFS
  + Fixed Depth DFS
* Breadth First Search

**Depth First Search**

**Depth-first search** (**DFS**) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hasn't finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration.

Applications

For applications of DFS to search problems in artificial intelligence, however, the graph to be searched is often either too large to visit in its entirety or even infinite, and DFS may suffer from non-termination when the length of a path in the search tree is infinite. Therefore, the search is only performed to a limited depth, and due to limited memory availability one typically does not use data structures that keep track of the set of all previously visited vertices. In this case, the time is still linear in the number of expanded vertices and edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once and others not at all) but the space complexity of this variant of DFS is only proportional to the depth limit, much smaller than the space needed for searching to the same depth using breadth-first search. For such applications, DFS also lends itself much better to heuristic methods of choosing a likely-looking branch. When an appropriate depth limit is not known a priori, iterative deepening depth-first search applies DFS repeatedly with a sequence of increasing limits; in the artificial intelligence mode of analysis, with a branching factor greater than one, iterative deepening increases the running time by only a constant factor over the case in which the correct depth limit is known due to the geometric growth of the number of nodes per level.

DFS may be also used to collect a sample of graph nodes. However, incomplete DFS, similarly to incomplete BFS, is biased towards nodes of high degree.

**Advantages of DFS**

* The advantage of depth-first Search is that memory requirement is only linear with respect to the search graph. This is in contrast with [breadth-first search](http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html) which requires more space. The reason is that the algorithm only needs to store a stack of nodes on the path from the root to the current node.
* The time complexity of a depth-first Search to depth d is O(b^d) since it generates the same set of nodes as [breadth-first search](http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html), but simply in a different order. Thus practically depth-first search is time-limited rather than space-limited.
* If depth-first search finds solution without exploring much in a path then the time and space it takes will be very less.

**Disadvantages**

* The disadvantage of Depth-First Search is that there is a possibility that it may go down the left-most path forever. Even a finite graph can generate an infinite tree. One solution to this problem is to impose a cutoff depth on the search. Although the ideal cutoff is the solution depth d and this value is rarely known in advance of actually solving the problem. If the chosen cutoff depth is less than d, the algorithm will fail to find a solution, whereas if the cutoff depth is greater than d, a large price is paid in execution time, and the first solution found may not be an optimal one.
* Depth-First Search is not guaranteed to find the solution.
* And there is no guarantee to find a minimal solution, if more than one solution exists.

**Breadth First Search**

In graph theory, **breadth-first search** (**BFS**) is a strategy for searching in a graph when search is limited to essentially two operations: (a) visit and inspect a node of a graph; (b) gain access to visit the nodes that neighbor the currently visited node. The BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on. Compare it with the depth-first search.

**Applications**

Breadth-first search can be used to solve many problems in graph theory, for example:

* Finding all nodes within one [connected component](http://en.wikipedia.org/wiki/Connected_component_%28graph_theory%29)
* Copying Collection, Cheney's algorithm
* Finding the shortest path between two nodes *u* and *v* (with path length measured by number of edges)
* Testing a graph for [bipartiteness](http://en.wikipedia.org/wiki/Bipartite_graph)
* (Reverse) Cuthill–McKee mesh numbering
* Ford–Fulkerson method for computing the maximum flow in a flow network
* Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.

**Advantages**

1. Breadth first search will never get trapped exploring the useless path forever.
2. If there is a solution, BFS will definitely find it out.
3. If there is more than one solution then BFS can find the minimal one that requires less number of steps.

**Disadvantages**

1. The main drawback of Breadth first search is its memory requirement. Since each level of the tree must be saved in order to generate the next level, and the amount of memory is proportional to the number of nodes stored, the space complexity of BFS is O(bd). As a result, BFS is severely space-bound in practice so will exhaust the memory available on typical computers in a matter of minutes.
2. If the solution is farther away from the root, breath first search will consume lot of time.

**Heuristic search**

Heuristic search algorithms have knowledge of where the goal or finish of the graph. For example, in a maze, they would know which path leads in the direction of the goal. Blind search algorithms have no knowledge of where the goal is, and wander "blindly" through the graph.

Heuristic search is an [AI search](http://intelligence.worldofcomputing.net/ai-search/ai-search-techniques.html) technique that employs heuristic for its moves. Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number. In [Artificial Intelligence](http://intelligence.worldofcomputing.net/ai-introduction/artificial-intelligence-overview.html), **heuristic search** has a general meaning, and a more specialized technical meaning. In a general sense, the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case. Within the heuristic search architecture, however, the term heuristic usually refers to the special case of a [heuristic evaluation function](http://intelligence.worldofcomputing.net/ai-search/heuristic-evaluation-function.html).

* Best First Search
* A Star Search

**Best First Search**

**Best-first search** is a [search algorithm](http://en.wikipedia.org/wiki/Search_algorithm) which explores a [graph](http://en.wikipedia.org/wiki/Graph_%28data_structure%29) by expanding the most promising node chosen according to a specified rule.

[Judea Pearl](http://en.wikipedia.org/wiki/Judea_Pearl) described best-first search as estimating the promise of node *n* by a "heuristic evaluation function f(n) which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."

Some authors have used "best-first search" to refer specifically to a search with a [heuristic](http://en.wikipedia.org/wiki/Heuristic_function) that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called [**greedy**](http://en.wikipedia.org/wiki/Greedy_algorithm) **best-first search**.

Efficient selection of the current best candidate for extension is typically implemented using a [priority queue](http://en.wikipedia.org/wiki/Priority_queue).

**A-Star Search**

A\* algorithm is a best-first search algorithm in which the cost associated with a node is f(n) = g(n) + h(n), where g(n) is the cost of the path from the initial state to node n and h(n) is the heuristic estimate or the cost or a path from node n to a goal. Thus, f(n) estimates the lowest total cost of any solution path going through node n. At each point a node with lowest f value is chosen for expansion. Ties among nodes of equal f value should be broken in favour of nodes with lower h values. The algorithm terminates when a goal is chosen for expansion.

A\* algorithm guides an optimal path to a goal if the heuristic function h(n) is admissible, meaning it never overestimates actual cost. For example, since airline distance never overestimates actual highway distance, and manhatten distance never overestimates actual moves in the gliding tile.

For Puzzle, A\* algorithm, using these evaluation functions, can find optimal solutions to these problems. In addition, A\* makes the most efficient use of the given heuristic function in the following sense: among all shortest-path algorithms using the given heuristic function h(n). A\* algorithm expands the fewest number of nodes.

The main drawback of A\* algorithm and indeed of any best-first search is its memory requirement. Since at least the entire open list must be saved, A\* algorithm is severely space-limited in practice, and is no more practical than breadth-first search on current machines. For example, while it can be run successfully on the eight puzzle, it exhausts available memory in a matter of minutes on the fifteen puzzle.

**References**

* Material taught in class,