Advances in Machine Learning

(Branch of Artificial Intelligence)

Margert Cooney

First Edition, 2012

ISBN 978-81-323-4477-3

Published by:
White Word Publications
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter- 1

Introduction to Machine Learning

Machine learning, a branch of artificial intelligence, is a scientific discipline that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data). Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases. Machine Learning, like all subjects in artificial intelligence, require cross-disciplinary proficiency in several areas, such as probability theory, statistics, pattern recognition, cognitive science, data mining, adaptive control, computational neuroscience and theoretical computer science.

Definition

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.

Generalization

The core objective of a learner is to generalize from its experience. The training examples from its experience come from some generally unknown probability distribution and the learner has to extract from them something more general, something about that distribution, that allows it to produce useful answers in new cases.

Human interaction

Some machine learning systems attempt to eliminate the need for human intuition in data analysis, while others adopt a collaborative approach between human and machine. Human intuition cannot, however, be entirely eliminated, since the system's designer must specify how the data is to be represented and what mechanisms will be used to search for a characterization of the data.

Algorithm types

Machine learning algorithms are organized into a taxonomy, based on the desired outcome of the algorithm.

- **Supervised learning** generates a function that maps inputs to desired outputs. For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- **Unsupervised learning** models a set of inputs, like clustering.
- **Semi-supervised learning** combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- **Reinforcement learning** learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.
- **Transduction** tries to predict new outputs based on training inputs, training outputs, and test inputs.
- **Learning to learn** learns its own inductive bias based on previous experience.

Theory

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield absolute guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

There are many similarities between machine learning theory and statistics, although they use different terms.

Approaches

Decision tree learning

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value.

Association rule learning

Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial neural networks

An artificial neural network (ANN), usually called "neural network" (NN), is a mathematical model or computational model that tries to simulate the structure and/or functional aspects of biological neural networks. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs or to find patterns in data.

Genetic programming

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task. It is a specialization of genetic algorithms (GA) where each individual is a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task.

Inductive logic programming

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program which entails all the positive and none of the negative examples.

Support vector machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Clustering

Cluster analysis or clustering is the assignment of a set of observations into subsets (called *clusters*) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Bayesian networks

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

Reinforcement learning

Reinforcement learning is concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Applications

Applications for machine learning include machine perception, computer vision, natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics, brain-machine interfaces and cheminformatics, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing, software engineering, adaptive websites, robot locomotion, and structural health monitoring.

Machine learning techniques helped win a major software competition: In 2006, the online movie company Netflix held the first "Netflix Prize" competition to find a program to better predict user preferences and beat its existing Netflix movie recommendation system by at least 10%. The AT&T Research Team BellKor won over several other teams with their machine learning program called Pragmatic Chaos. After winning several minor prizes, it won the 2009 grand prize competition for $1 million.

Software

RapidMiner, KNIME, Weka, ODM, Shogun toolbox and Orange are software suites containing a variety of machine learning algorithms.

Chapter- 2

Supervised Learning

Supervised learning is the machine learning task of inferring a function from *supervised* training data. The training data 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* or a *regression function*. 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. (Compare with unsupervised learning.) The parallel task in human and animal psychology is often referred to as 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 engineer should decide what kind of data is to be used as an example. For instance, 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; 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.
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.
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.

There are four major issues to consider in supervised learning:

Bias-variance tradeoff

A first issue is the tradeoff between *bias* and *variance* . Imagine that we have available several different, but equally good, training data sets. A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x. A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets. The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm . Generally, there is a tradeoff between bias and variance. A learning algorithm with low bias must be "flexible" so that it can fit the data well. But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance. A key aspect of many supervised learning methods is that they are able to adjust this tradeoff between bias and variance (either automatically or by providing a bias/variance parameter that the user can adjust).

Function complexity and amount of training data

The second issue is the amount of training data available relative to the complexity of the "true" function (classifier or regression function). If the true function is simple, then an "inflexible" learning algorithm with high bias and low variance will be able to learn it from a small amount of data. But if the true function is highly complex (e.g., because it involves complex interactions among many different input features and behaves differently in different parts of the input space), then the function will only be learnable from a very large amount of training data and using a "flexible" learning algorithm with low bias and high variance. Good learning algorithms therefore automatically adjust the bias/variance tradeoff based on the amount of data available and the apparent complexity of the function to be learned.

Dimensionality of the input space

A third issue is the dimensionality of the input space. If the input feature vectors have very high dimension, the learning problem can be difficult even if the true function only depends on a small number of those features. This is because the many "extra" dimensions can confuse the learning algorithm and cause it to have high variance. Hence, high input dimensionality typically requires tuning the classifier to have low variance and high bias. In practice, if the engineer can manually remove irrelevant features from the input data, this is likely to improve the accuracy of the learned function. In addition, there are many algorithms for feature selection that seek to identify the relevant features and discard the irrelevant ones. This is an instance of the more general strategy of dimensionality reduction, which seeks to map the input data into a lower dimensional space prior to running the supervised learning algorithm.

Noise in the output values

A fourth issue is the degree of noise in the desired output values (the supervisory targets). If the desired output values are often incorrect (because of human error or sensor errors), then the learning algorithm should not attempt to find a function that exactly matches the training examples. This is another case where it is usually best to employ a high bias, low variance classifier.

Other factors to consider

Other factors to consider when choosing and applying a learning algorithm include the following:

1. Heterogeneity of the data. If the feature vectors include features of many different kinds (discrete, discrete ordered, counts, continuous values), some algorithms are easier to apply than others. Many algorithms, including Support Vector Machines, linear regression, logistic regression, neural networks, and nearest neighbor methods, require that the input features be numerical and scaled to similar ranges (e.g., to the [-1,1] interval). Methods that employ a distance function, such as nearest neighbor methods and support vector machines with Gaussian kernels, are particularly sensitive to this. An advantage of decision trees is that they easily handle heterogeneous data.
2. Redundancy in the data. If the input features contain redundant information (e.g., highly correlated features), some learning algorithms (e.g., linear regression, logistic regression, and distance based methods) will perform poorly because of numerical instabilities. These problems can often by solved by imposing some form of regularization.
3. Presence of interactions and non-linearities. If each of the features makes an independent contribution to the output, then algorithms based on linear functions (e.g., linear regression, logistic regression, Support Vector Machines, naive Bayes) and distance functions (e.g., nearest neighbor methods, support vector machines with Gaussian kernels) generally perform well. However, if there are complex interactions among features, then algorithms such as decision trees and neural networks work better, because they are specifically designed to discover

these interactions. Linear methods can also be applied, but the engineer must manually specify the interactions when using them.

When considering a new application, the engineer can compare multiple learning algorithms and experimentally determine which one works best on the problem at hand (Tuning the performance of a learning algorithm can be very time-consuming. Given fixed resources, it is often better to spend more time collecting additional training data and more informative features than it is to spend extra time tuning the learning algorithms.

The most widely used learning algorithms are Support Vector Machines, linear regression, logistic regression, naive Bayes, linear discriminant analysis, decision trees, k-nearest neighbor algorithm, and Neural Networks (Multilayer perceptron).

How supervised learning algorithms work

Given a set of training examples of the form $\{(x_1, y_1), ..., (x_N,\ y_N)\}$, a learning algorithm seeks a function $g : X \to Y$, where *X* is the input space and *Y* is the output space. The function *g* is an element of some space of possible functions *G*, usually called the *hypothesis space*. It is sometimes convenient to represent *g* using a scoring function $f : X \times Y \to \mathbb{R}$ such that *g* is defined as returning the *y* value that gives the highest score: $g(x) = \arg\max_y\ f(x, y)$. Let *F* denote the space of scoring functions.

Although *G* and *F* can be any space of functions, many learning algorithms are probabilistic models where *g* takes the form of a conditional probability model $g(x) = P(y \mid x)$, or *f* takes the form of a joint probability model $f(x,y) = P(x,y)$. For example, naive Bayes and linear discriminant analysis are joint probability models, whereas logistic regression is a conditional probability model.

There are two basic approaches to choosing *f* or *g*: empirical risk minimization and structural risk minimization . Empirical risk minimization seeks the function that best fits the training data. Structural risk minimize includes a *penalty function* that controls the bias/variance tradeoff.

In both cases, it is assumed that the training set consists of a sample of independent and identically distributed pairs, $(x_i,\ y_i)$. In order to measure how well a function fits the training data, a loss function $L : Y \times Y \to \mathbb{R}^{\geq 0}$ is defined. For training example $(x_i,\ y_i)$, the loss of predicting the value $\hat{y}$ is $L(y_i, \hat{y})$.

The *risk* $R(g)$ of function *g* is defined as the expected loss of *g*. This can be estimated from the training data as

$$R_{emp}(g) = \frac{1}{N} \sum_i L(y_i, g(x_i))$$

.

Empirical risk minimization

Empirical risk minimization (ERM) is a principle in statistical learning theory which defines a family of learning algorithms and is used to give theoretical bounds on the performance of learning algorithms.

Background

Consider the following situation, which is a general setting of many supervised learning problems. We have two spaces of objects *X* and *Y* and would like to learn a function $h : X \to Y$ (often called *hypothesis*) which outputs an object $y \in Y$, given $x \in X$. To do so, we have in our disposal a *training set* of a few examples $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$ is an input and $y_i \in Y$ is the corresponding response that we wish to get from $h(x_i)$.

To put it more formally, we assume that there is a joint probability distribution *P*(*x*,*y*) over *X* and *Y*, and that the training set consists of *m* instances $(x_1, y_1), \ldots, (x_m, y_m)$ drawn i.i.d. from *P*(*x*,*y*). Note that the assumption of a joint probability distribution allows us to model uncertainty in predictions (e.g. from noise in data) because *y* is not a deterministic function of *x*, but rather a random variable with conditional distribution *P*(*y* | *x*) for a fixed *x*.

We also assume that we are given a non-negative real-valued loss function $L(\hat{y}, y)$ which measures how different is the prediction $\hat{y}$ of a hypothesis from the true outcome *y*. The risk associated with hypothesis *h*(*x*) is then defined as the expectation of the loss function:

$$R(h) = \mathbf{E}[L(h(x), y)] = \int L(h(x), y)\, dP(x, y).$$

A loss function commonly used in theory is the 0-1 loss function: $L(\hat{y}, y) = I(\hat{y} \neq y)$, where *I*(...) is the indicator notation.

The ultimate goal of a learning algorithm is to find a hypothesis h^* among a fixed class of functions $\mathcal{H}$ for which the risk *R*(*h*) is minimal:

$$h^* = \arg\min_{h \in \mathcal{H}} R(h).$$

Empirical risk minimization

In general, the risk $R(h)$ cannot be computed because the distribution $P(x,y)$ is unknown to the learning algorithm (this situation is referred to as agnostic learning). However, we can compute an approximation, called *empirical risk*, by averaging the loss function on the training set:

$$R_{\text{emp}}(h) = \frac{1}{m}\sum_{i=1}^{m} L(h(x_i), y_i).$$

Empirical risk minimization principle states that the learning algorithm should choose a hypothesis $\hat{h}$ which minimizes the empirical risk:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} R_{\text{emp}}(h).$$

Thus the learning algorithm defined by ERM principle consists in solving the above optimization problem.

Properties

Computational complexity

Empirical risk minimization for a classification problem with 0-1 loss function is known to be an NP-hard problem even for such relatively simple class of functions as linear classifiers. Though, it can be solved efficiently when minimal empirical risk is zero, i.e. data is linearly separable.

In practice, machine learning algorithms cope with that either by employing a convex approximation to 0-1 loss function (like hinge loss for SVM), which is easier to optimize, or by posing assumptions on the distribution $P(x,y)$ (and thus stop being agnostic learning algorithms to which the above result applies,)

In empirical risk minimization, the supervised learning algorithm seeks the function g that minimizes $R(g)$. Hence, a supervised learning algorithm can be constructed by applying an optimization algorithm to find g.

When g is a conditional probability distribution $P(y \mid x)$ and the loss function is the negative log likelihood: $L(y, \hat{y}) = -\log P(y|x)$, then empirical risk minimization is equivalent to maximum likelihood estimation.

When G contains many candidate functions or the training set is not sufficiently large, empirical risk minimization leads to high variance and poor generalization. The learning algorithm is able to memorize the training examples without generalizing well. This is called overfitting.

Structural risk minimization

Structural risk minimization seeks to prevent overfitting by incorporating a regularization penalty into the optimization. The regularization penalty can be viewed as implementing a form of Occam's razor that prefers simpler functions over more complex ones.

A wide variety of penalties have been employed that correspond to different definitions of complexity. For example, consider the case where the function g is a linear function of the form

$$g(x) = \sum_{j=1}^{d} \beta_j x_j$$

.

A popular regularization penalty is $\sum_j \beta_j^2$, which is the squared Euclidean norm of the weights, also known as the L_2 norm. Other norms include the L_1 norm,

$$\sum_j |\beta_j|$$

, and the L_0 norm, which is the number of non-zero β_js. The penalty will be denoted by $C(g)$.

The supervised learning optimization problem is to find the function g that minimizes

$$J(g) = R_{emp}(g) + \lambda C(g).$$

The parameter λ controls the bias-variance tradeoff. When $\lambda = 0$, this gives empirical risk minimization with low bias and high variance. When λ is large, the learning algorithm will have high bias and low variance. The value of λ can be chosen empirically via cross validation.

The complexity penalty has a Bayesian interpretation as the negative log prior probability of g, $-\log P(g)$, in which case $J(g)$ is the posterior probabability of g.

Generative training

The training methods described above are *discriminative training* methods, because they seek to find a function g that discriminates well between the different output values. For the special case where $f(x,y) = P(x,y)$ is a joint probability distribution and the loss function is the negative log likelihood

$$-\sum \log P(x_i,y_i),$$

a risk minimization algorithm is said to perform *generative training*, because *f* can be regarded as a generative model that explains how the data were generated. Generative training algorithms are often simpler and more computationally efficient than discriminative training algorithms. In some cases, the solution can be computed in closed form as in naive Bayes and linear discriminant analysis.

Generalizations of supervised learning

There are several ways in which the standard supervised learning problem can be generalized:

Semi-supervised learning

In computer science, **semi-supervised learning** is a class of machine learning techniques that make use of both labeled and unlabeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. The acquisition of labeled data for a learning problem often requires a skilled human agent to manually classify training examples. The cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas acquisition of unlabeled data is relatively inexpensive. In such situations, semi-supervised learning can be of great practical value.

One example of a semi-supervised learning technique is co-training, in which two or possibly more learners are each trained on a set of examples, but with each learner using a different, and ideally independent, set of features for each example.

An alternative approach is to model the joint probability distribution of the features and the labels. For the unlabelled data the labels can then be treated as 'missing data'. Techniques that handle missing data, such as Gibbs sampling or the EM algorithm, can then be used to estimate the parameters of the model.

Active learning (machine learning)

Active learning is a form of supervised machine learning in which the learning algorithm is able to interactively query the user (or some other information source) to obtain the desired outputs at new data points. In statistics literature it is sometimes also called optimal experimental design.

There are situations in which unlabeled data is abundant but labeling data is expensive. In such a scenario the learning algorithm can actively query the user/teacher for labels. This type of iterative supervised learning is called active learning. Since the learner chooses the examples, the number of examples to learn a concept can often be much lower than the number required in normal supervised learning. With this approach there is a risk that the algorithm might focus on unimportant or even invalid examples.

Active learning can be especially useful in biological research problems such as Protein engineering where a few proteins have been discovered with a certain interesting function and one wishes to determine which of many possible mutants to make next that will have a similar function .

Definitions

Let T be the total set of all data under consideration. For example, in a protein engineering problem, T would include all proteins that are known to have a certain interesting activity and all additional proteins that one might want to test for that activity.

During each iteration, i, T is broken up into three subsets

1. $\mathbf{T}_{K,i}$: Data points where the label is **known**.
2. $\mathbf{T}_{U,i}$: Data points where the label is **unknown**.
3. $\mathbf{T}_{C,i}$: A subset of $T_{U,i}$ that is **chosen** to be labeled.

Most of the current research in active learning involves the best method to chose the data points for $T_{C,i}$.

Minimum Marginal Hyperplane

Some active learning algorithms are built upon Support vector machines (SVMs) and exploit the structure of the SVM to determine which data points to label. Such methods usually calculate the margin, W, of each unlabeled datum in $T_{U,i}$ and treat W as an n-dimensional distance from that datum to separating hyperplane.

Minimum Marginal Hyperplane methods assume that the data with the smallest W are those that the SVM is most uncertain about and therefore should be placed in $T_{C,i}$ to be labeled. Other similar methods, such as Maximum Marginal Hyperplane, choose data with the largest W. Tradeoff methods choose a mix of the smallest and largest Ws.

Maximum Curiosity

Another active learning method, that typically learns a data set with fewer examples than Minimum Marginal Hyperplane but is more computationally intensive and only works for discrete classifiers is Maximum Curiosity .

Maximum curiosity takes each unlabeled datum in $T_{U,i}$ and assumes all possible labels that datum might have. This datum with each assumed class is added to $T_{K,i}$ and then the new $T_{K,i}$ is cross-validated. It is assumed that when the datum is paired up with its correct label, the cross-validated accuracy (or correlation coefficient) of $T_{K,i}$ will most improve. The datum with the most improved accuracy is placed in $T_{C,i}$ to be labeled

Structured prediction

Structured prediction is an umbrella term for machine learning and regression techniques that involve predicting structured objects. For example, the problem of translating a natural language sentence into a semantic representation such as a parse tree can be seen as a structured prediction problem in which the structured output domain is the set of all possible parse trees. Structured prediction generalizes supervised learning where the output domain is usually a small or simple set.

Probabilistic graphical models form a large class of structured prediction models. In particular, Bayesian networks and random fields are popularly used to solve structured prediction problems in a wide variety of application domains including bioinformatics, natural language processing, speech recognition, and computer vision.

Similar to commonly used supervised learning techniques, structured prediction models are typically trained by means of observed data in which the true prediction value is used to adjust model parameters. Due to the complexity of the model and the interrelations of predicted variables the process of prediction using a trained model and of training itself is often computationally infeasible and approximate inference and learning methods are used.

Another commonly used term for structured prediction is structured output learning.

Chapter- 3

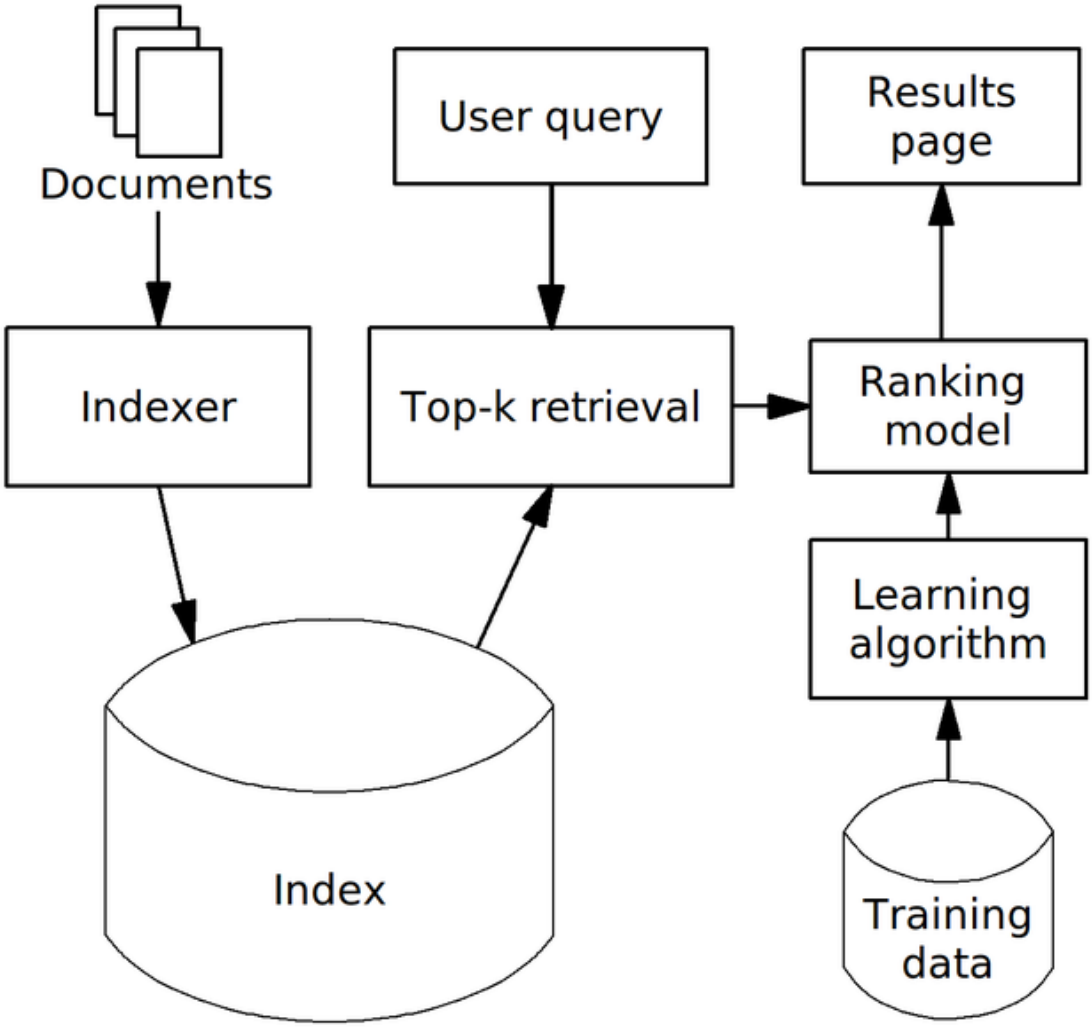
Learning to Rank

Learning to rank or **machine-learned ranking** (MLR) is a type of supervised or semi-supervised machine learning problem in which the goal is to automatically construct a ranking model from training data. Training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g. "relevant" or "not relevant") for each item. Ranking model's purpose is to rank, i.e. produce a permutation of items in new, unseen lists in a way, which is "similar" to rankings in the training data in some sense.

Learning to rank is a relatively new research area which has emerged in the past decade.

Applications

In information retrieval



A possible architecture of a machine-learned search engine.

Ranking is a central part of many information retrieval problems, such as document retrieval, collaborative filtering, sentiment analysis, computational advertising (online ad placement).

When applied to document retrieval, the task of learning to rank is to construct a ranking function for a search engine. In this case each list in training data represents documents which match a search query and they are ordered according to relevance to the query.

A possible architecture of a machine-learned search engine is shown in the figure to the right.

Training data consists of queries and documents matching them together with relevance degree of each match. It may be prepared manually by human *assessors* (or *raters*, as Google calls them), who check results for some queries and determine relevance of each result. It is not feasible to check relevance of all documents, and so typically a technique

called pooling is used — only top few documents, retrieved by some existing ranking models are checked. Alternatively, training data may be derived automatically by analyzing *clickthrough logs* (i.e. search results which got clicks from users), *query chains*, or such search engines' features as Google's.

Training data is used by a learning algorithm to produce a ranking model which computes relevance of documents for actual queries.

Typically, users expect a search query to complete in a short time (such as a few hundred milliseconds for web search), which makes it impossible to evaluate a complex ranking model on each document in the corpus, and so a two-phase scheme is used. First, a small number of potentially relevant documents are identified using simpler retrieval models which permit fast query evaluation, such as vector space model, boolean model, weighted AND , BM25. This phase is called *top-k document retrieval* and many good heuristics were proposed in the literature to accelerate it, such as using document's static quality score and tiered indexes. In the second phase, a more accurate but computationally expensive machine-learned model is used to re-rank these documents.

In other areas

Learning to rank algorithms have been applied in areas other than information retrieval:

- In machine translation for ranking a set of hypothesized translations;
- In computational biology for ranking candidate 3-D structures in protein structure prediction problem.
- In proteomics for the identification of frequent top scoring peptides.

Feature vectors

For convenience of MLR algorithms, query-document pairs are usually represented by numerical vectors, which are called *feature vectors*. Such approach is sometimes called *bag of features* and is analogous to bag of words and vector space model used in information retrieval for representation of documents.

Components of such vectors are called *features*, *factors* or *ranking signals*. They may be divided into three groups (features from document retrieval are shown as examples):

- *Query-independent* or *static* features — those features, which depend only on the document, but not on the query. For example, PageRank or document's length. Such features can be precomputed in off-line mode during indexing. They may be used to compute document's *static quality score* (or *static rank*), which is often used to speed up search query evaluation.
- *Query-dependent* or *dynamic* features — those features, which depend both on the contents of the document and the query, such as TF-IDF score or other non-machine-learned ranking functions.

- *Query features*, which depend only on the query. For example, the number of words in a query.

Some examples of features, which were used in the well-known LETOR dataset:

- TF, TF-IDF, BM25, and language modeling scores of document's zones (title, body, anchors text, URL) for a given query;
- Lengths and IDF sums of document's zones;
- Document's PageRank, HITS ranks and their variants.

Selecting and designing good features is an important area in machine learning, which is called feature engineering.

Evaluation measures

There are several measures (metrics) which are commonly used to judge how well an algorithm is doing on training data and to compare performance of different MLR algorithms. Often a learning-to-rank problem is reformulated as an optimization problem with respect to one of these metrics.

Examples of ranking quality measures:

Information retrieval

Information retrieval (**IR**) is the science of searching for documents, for information within documents, and for metadata about documents, as well as that of searching relational databases and the World Wide Web. There is overlap in the usage of the terms data retrieval, document retrieval, information retrieval, and text retrieval, but each also has its own body of literature, theory, praxis, and technologies. IR is interdisciplinary, based on computer science, mathematics, library science, information science, information architecture, cognitive psychology, linguistics, and statistics.

Automated information retrieval systems are used to reduce what has been called "information overload". Many universities and public libraries use IR systems to provide access to books, journals and other documents. Web search engines are the most visible IR applications.

History

> “But do you know that, although I have kept the diary [on a phonograph] for ”

months past, it
never once struck
me how I was
going to find any
particular part of
it in case I
wanted to look it
up?

—Dr Seward, Bram Stoker's
Dracula, 1897

The idea of using computers to search for relevant pieces of information was popularized in the article *As We May Think* by Vannevar Bush in 1945. The first automated information retrieval systems were introduced in the 1950s and 1960s. By 1970 several different techniques had been shown to perform well on small text corpora such as the Cranfield collection (several thousand documents). Large-scale retrieval systems, such as the Lockheed Dialog system, came into use early in the 1970s.

In 1992, the US Department of Defense along with the National Institute of Standards and Technology (NIST), cosponsored the Text Retrieval Conference (TREC) as part of the TIPSTER text program. The aim of this was to look into the information retrieval community by supplying the infrastructure that was needed for evaluation of text retrieval methodologies on a very large text collection. This catalyzed research on methods that scale to huge corpora. The introduction of web search engines has boosted the need for very large scale retrieval systems even further.

The use of digital methods for storing and retrieving information has led to the phenomenon of digital obsolescence, where a digital resource ceases to be readable because the physical media, the reader required to read the media, the hardware, or the software that runs on it, is no longer available. The information is initially easier to retrieve than if it were on paper, but is then effectively lost.

Timeline

- Before the **1900s**

  1880s: Herman Hollerith invents the recording of data on a machine readable medium.
  1890 Hollerith cards, key punches and tabulators used to process the 1890 US Census data.

- **1940s–1950s**

  late 1940s: The US military confronted problems of indexing and retrieval of wartime scientific research documents captured from Germans.

1945: Vannevar Bush's *As We May Think* appeared in *Atlantic Monthly*.
1947: Hans Peter Luhn (research engineer at IBM since 1941) began work on a mechanized punch card-based system for searching chemical compounds.
1950s: Growing concern in the US for a "science gap" with the USSR motivated, encouraged funding and provided a backdrop for mechanized literature searching systems (Allen Kent *et al.*) and the invention of citation indexing (Eugene Garfield).
1950: The term "information retrieval" appears to have been coined by Calvin Mooers.
1951: Philip Bagley conducted the earliest experiment in computerized document retrieval in a master thesis at MIT.
1955: Allen Kent joined Case Western Reserve University, and eventually became associate director of the Center for Documentation and Communications Research. That same year, Kent and colleagues published a paper in American Documentation describing the precision and recall measures as well as detailing a proposed "framework" for evaluating an IR system which included statistical sampling methods for determining the number of relevant documents not retrieved.
1958: International Conference on Scientific Information Washington DC included consideration of IR systems as a solution to problems identified. See: *Proceedings of the International Conference on Scientific Information, 1958* (National Academy of Sciences, Washington, DC, 1959)
1959: Hans Peter Luhn published "Auto-encoding of documents for information retrieval."

- **1960s**:

  early 1960s: Gerard Salton began work on IR at Harvard, later moved to Cornell.
  1960: Melvin Earl (Bill) Maron and John Lary Kuhns published "On relevance, probabilistic indexing, and information retrieval" in the Journal of the ACM 7(3):216–244, July 1960.
  1962:

  - Cyril W. Cleverdon published early findings of the Cranfield studies, developing a model for IR system evaluation. See: Cyril W. Cleverdon, "Report on the Testing and Analysis of an Investigation into the Comparative Efficiency of Indexing Systems". Cranfield Collection of Aeronautics, Cranfield, England, 1962.
  - Kent published *Information Analysis and Retrieval*.

  1963:

  - Weinberg report "Science, Government and Information" gave a full articulation of the idea of a "crisis of scientific information." The report was named after Dr. Alvin Weinberg.

- Joseph Becker and Robert M. Hayes published text on information retrieval. Becker, Joseph; Hayes, Robert Mayo. *Information storage and retrieval: tools, elements, theories*. New York, Wiley (1963).

1964:

- Karen Spärck Jones finished her thesis at Cambridge, *Synonymy and Semantic Classification*, and continued work on computational linguistics as it applies to IR.
- The National Bureau of Standards sponsored a symposium titled "Statistical Association Methods for Mechanized Documentation." Several highly significant papers, including G. Salton's first published reference (we believe) to the SMART system.

mid-1960s:

- National Library of Medicine developed MEDLARS Medical Literature Analysis and Retrieval System, the first major machine-readable database and batch-retrieval system.
- Project Intrex at MIT.

1965: J. C. R. Licklider published *Libraries of the Future*.

1966: Don Swanson was involved in studies at University of Chicago on Requirements for Future Catalogs.

late 1960s: F. Wilfrid Lancaster completed evaluation studies of the MEDLARS system and published the first edition of his text on information retrieval.

1968:

- Gerard Salton published *Automatic Information Organization and Retrieval*.
- John W. Sammon, Jr.'s RADC Tech report "Some Mathematics of Information Storage and Retrieval..." outlined the vector model.

1969: Sammon's "A nonlinear mapping for data structure analysis" (IEEE Transactions on Computers) was the first proposal for visualization interface to an IR system.

- **1970s**

early 1970s:

- First online systems—NLM's AIM-TWX, MEDLINE; Lockheed's Dialog; SDC's ORBIT.

- Theodor Nelson promoting concept of hypertext, published *Computer Lib/Dream Machines*.

1971: Nicholas Jardine and Cornelis J. van Rijsbergen published "The use of hierarchic clustering in information retrieval", which articulated the "cluster hypothesis." (Information Storage and Retrieval, 7(5), pp. 217–240, December 1971)

1975: Three highly influential publications by Salton fully articulated his vector processing framework and term discrimination model:

- *A Theory of Indexing* (Society for Industrial and Applied Mathematics)
- *A Theory of Term Importance in Automatic Text Analysis* (JASIS v. 26)
- *A Vector Space Model for Automatic Indexing* (CACM 18:11)

1978: The First ACM SIGIR conference.
1979: C. J. van Rijsbergen published *Information Retrieval* (Butterworths). Heavy emphasis on probabilistic models.

- **1980s**

1980: First international ACM SIGIR conference, joint with British Computer Society IR group in Cambridge.

1982: Nicholas J. Belkin, Robert N. Oddy, and Helen M. Brooks proposed the ASK (Anomalous State of Knowledge) viewpoint for information retrieval. This was an important concept, though their automated analysis tool proved ultimately disappointing.

1983: Salton (and Michael J. McGill) published *Introduction to Modern Information Retrieval* (McGraw-Hill), with heavy emphasis on vector models.

mid-1980s: Efforts to develop end-user versions of commercial IR systems.

1985–1993: Key papers on and experimental systems for visualization interfaces. Work by Donald B. Crouch, Robert R. Korfhage, Matthew Chalmers, Anselm Spoerri and others.

1989: First World Wide Web proposals by Tim Berners-Lee at CERN.

- **1990s**

1992: First TREC conference.

1997: Publication of Korfhage's *Information Storage and Retrieval* with emphasis on visualization and multi-reference point systems.

late 1990s: Web search engines implementation of many features formerly found only in experimental IR systems. Search engines become the most common and maybe best instantiation of IR models, research, and implementation.

Overview

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

An object is an entity that is represented by information in a database. User queries are matched against the database information. Depending on the application the data objects may be, for example, text documents, images , audio, mind maps or videos. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates or metadata.

Most IR systems compute a numeric score on how well each object in the database match the query, and rank the objects according to this value. The top ranking objects are then shown to the user. The process may then be iterated if the user wishes to refine the query.

Performance measures

Many different measures for evaluating the performance of information retrieval systems have been proposed. The measures require a collection of documents and a query. All common measures described here assume a ground truth notion of relevancy: every document is known to be either relevant or non-relevant to a particular query. In practice queries may be ill-posed and there may be different shades of relevancy.

Precision

Precision is the fraction of the documents retrieved that are relevant to the user's information need.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*.

Note that the meaning and usage of "precision" in the field of Information Retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall

Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

In binary classification, recall is called sensitivity. So it can be looked at as *the probability that a relevant document is retrieved by the query*.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

Fall-Out

The proportion of non-relevant documents that are retrieved, out of all non-relevant documents available:

$$\text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}$$

In binary classification, fall-out is closely related to specificity (1 − specificity). It can be looked at as *the probability that a non-relevant document is retrieved by the query*.

It is trivial to achieve fall-out of 0% by returning zero documents in response to any query.

F-measure

The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}.$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1+\beta^2)\cdot(\mathrm{precision}\cdot\mathrm{recall})}{(\beta^2\cdot\mathrm{precision}+\mathrm{recall})}$$

Two other commonly used F measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure $E = 1 - (1 / (\alpha / P + (1 - \alpha) / R))$. Their relationship is $F_\beta = 1 - E$ where $\alpha = 1 / (\beta^2 + 1)$.

Average precision

Precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. Average precision emphasizes ranking relevant documents higher. It is the average of precisions computed at the point of each of the relevant documents in the ranked sequence:

$$\mathrm{AveP} = \frac{\sum_{r=1}^{N}(P(r)\times \mathrm{rel}(r))}{\text{number of relevant documents}}$$

where *r* is the rank, *N* the number retrieved, *rel()* a binary function on the relevance of a given rank, and *P(r)* precision at a given cut-off rank:

$$\mathrm{P(r)} = \frac{|\{\text{relevant retrieved documents of rank r or less}\}|}{r}$$

This metric is also sometimes referred to geometrically as the area under the Precision-Recall curve.

Note that the denominator (number of relevant documents) is the number of relevant documents in the entire collection, so that the metric reflects performance over all relevant documents, regardless of a retrieval cutoff.

Mean average precision

Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\mathrm{MAP} = \frac{\sum_{q=1}^{Q}\mathrm{AveP(q)}}{Q}$$

where Q is the number of queries.

Discounted cumulative gain

DCG uses a graded relevance scale of documents from the result set to evaluate the usefulness, or gain, of a document based on its position in the result list. The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result.

The DCG accumulated at a particular rank position p is defined as:

$$\mathrm{DCG_p} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

Since result set may vary in size among different queries or systems, to compare performances the normalised version of DCG uses an ideal DCG. To this end, it sorts documents of a result list by relevance, producing an ideal DCG at position p ($IDCG_p$), which normalizes the score:

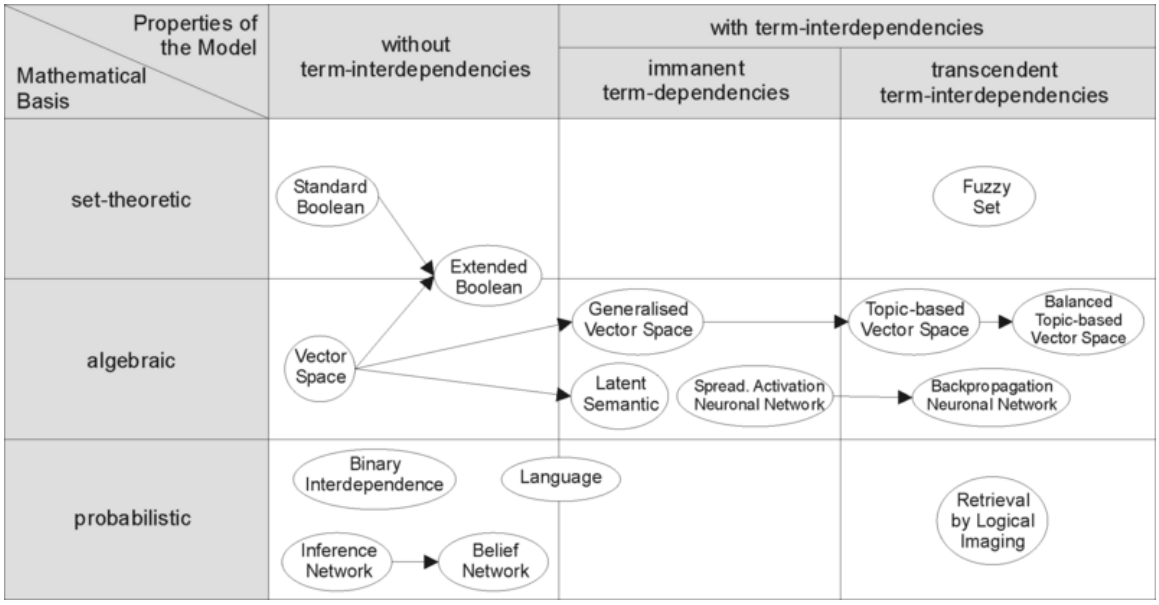
$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCGp}$$

The nDCG values for all queries can be averaged to obtain a measure of the average performance of a ranking algorithm. Note that in a perfect ranking algorithm, the DCG_p will be the same as the $IDCG_p$ producing an nDCG of 1.0. All nDCG calculations are then relative values on the interval 0.0 to 1.0 and so are cross-query comparable.

Other Measures

- Mean reciprocal rank
- Spearman's rank correlation coefficient

Model types



Categorization of IR-models (translated from German entry, original source Dominik Kuropka).

For the information retrieval to be efficient, the documents are typically transformed into a suitable representation. There are several representations. The picture on the right illustrates the relationship of some common models. In the picture, the models are categorized according to two dimensions: the mathematical basis and the properties of the model.

First dimension: mathematical basis

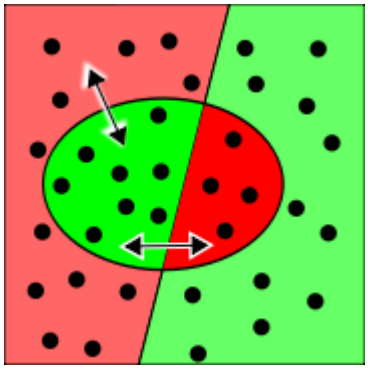
- *Set-theoretic models represent documents as sets of words or phrases. Similarities are usually derived from set-theoretic operations on those sets. Common models are:*
  - Standard Boolean model
  - Extended Boolean model
  - Fuzzy retrieval
- *Algebraic models* represent documents and queries usually as vectors, matrices, or tuples. The similarity of the query vector and document vector is represented as a scalar value.
  - Vector space model
  - Generalized vector space model
  - (Enhanced) Topic-based Vector Space Model
  - Extended Boolean model
  - Latent semantic indexing aka latent semantic analysis
- *Probabilistic models* treat the process of document retrieval as a probabilistic inference. Similarities are computed as probabilities that a document is relevant for a given query. Probabilistic theorems like the Bayes' theorem are often used in these models.

    - Binary Independence Model
    - Probabilistic relevance model on which is based the okapi (BM25) relevance function
    - Uncertain inference
    - Language models
    - Divergence-from-randomness model
    - Latent Dirichlet allocation
- *Machine-learned ranking* models view documents as vectors of ranking features (some of which often incorporate other ranking models mentioned above) and try to find the best way to combine these features into a single relevance score by machine learning methods.

Second dimension: properties of the model

- *Models without term-interdependencies* treat different terms/words as independent. This fact is usually represented in vector space models by the orthogonality assumption of term vectors or in probabilistic models by an independency assumption for term variables.
- *Models with immanent term interdependencies* allow a representation of interdependencies between terms. However the degree of the interdependency between two terms is defined by the model itself. It is usually directly or indirectly derived (e.g. by dimensional reduction) from the co-occurrence of those terms in the whole set of documents.
- *Models with transcendent term interdependencies* allow a representation of interdependencies between terms, but they do not allege how the interdependency between two terms is defined. They relay an external source for the degree of interdependency between two terms. (For example a human or sophisticated algorithms.)

Precision and recall



Recall and precision depend on the outcome (oval) of a query and its relation to all relevant documents (left) and the non-relevant documents (right). The more correct results (green), the better. **Precision**: horizontal arrow. **Recall**: diagonal arrow.

Precision and **recall** are two widely used metrics for evaluating the correctness of a pattern recognition algorithm. They can be seen as extended versions of *accuracy*, a simple metric that computes the fraction of instances for which the correct result is returned.

When using precision and recall, the set of possible labels for a given instance is divided into two subsets, one of which is considered "relevant" for the purposes of the metric. Recall is then computed as the fraction of correct instances among all instances that *actually* belong to the relevant subset, while precision is the fraction of correct instances among those that the algorithm *believes* to belong to the relevant subset.

Precision can be seen as a measure of exactness or fidelity, whereas recall is a measure of completeness.

Introduction

As an example, in an information retrieval scenario, the instances are documents and the task is to return a set of relevant documents given a search term; or equivalently, to assign each document to one of two categories, "relevant" and "not relevant". In this case, the "relevant" documents are simply those that belong to the "relevant" category. Recall is defined as the *number of relevant documents* retrieved by a search *divided by the total number of existing relevant documents*, while precision is defined as the *number of relevant documents* retrieved by a search *divided by the total number of documents retrieved* by that search.

In a classification task, the precision for a class is the *number of* ***true positives*** (i.e. the *number of items correctly labeled as belonging to the positive class*) *divided by the total number of elements labeled as belonging to the positive class* (i.e. the sum of true positives and **false positives**, which are items incorrectly labeled as belonging to the class). Recall in this context is defined as the *number of true positives divided by the total number of elements that actually belong to the positive class* (i.e. the sum of true positives and **false negatives**, which are items which were not labeled as belonging to the positive class but should have been).

In information retrieval, a perfect precision score of 1.0 means that every result retrieved by a search was relevant (but says nothing about whether all relevant documents were retrieved) whereas a perfect recall score of 1.0 means that all relevant documents were retrieved by the search (but says nothing about how many irrelevant documents were also retrieved).

In a classification task, a precision score of 1.0 for a class C means that every item labeled as belonging to class C does indeed belong to class C (but says nothing about the number of items from class C that were not labeled correctly) whereas a recall of 1.0 means that every item from class C was labeled as belonging to class C (but says nothing about how many other items were incorrectly also labeled as belonging to class C).

Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. For example, an information retrieval system (such as a search engine) can often increase its recall by retrieving more documents, at the cost of increasing number of irrelevant documents retrieved (decreasing precision). Similarly, a classification system for deciding whether or not, say, a fruit is an orange, can achieve high precision by only classifying fruits with the exact right shape and color as oranges, but at the cost of low recall due to the number of *false negatives* from oranges that did not quite match the specification.

Usually, precision and recall scores are not discussed in isolation. Instead, either values for one measure are compared for a fixed level at the other measure (e.g. *precision at a recall level of 0.75*) or both are combined into a single measure, such as the **F-measure**, which is the *weighted harmonic mean of precision and recall* (see below), or the Matthews correlation coefficient.

Definition (information retrieval context)

In information retrieval contexts, precision and recall are defined in terms of a set of **retrieved documents** (e.g. the list of documents produced by a web search engine for a query) and a set of **relevant documents** (e.g. the list of all documents on the internet that are relevant for a certain topic).

Precision

In the field of information retrieval, **precision** is the fraction of retrieved documents that are relevant to the search:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Precision takes all retrieved documents into account, but it can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called **precision at n** or **P@n**.

For example for a text search on a set of documents precision is the number of correct results divided by the number of all returned results.

Precision is also used with recall, the percent of *all* relevant documents that is returned by the search. The two measures are sometimes used together in the F1 Score (or f-measure) to provide a single measurement for a system.

Note that the meaning and usage of "precision" in the field of Information Retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall

Recall in Information Retrieval is the fraction of the documents that are relevant to the query that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

For example for text search on a set of documents recall is the number of correct results divided by the number of results that should have been returned

In binary classification, recall is called sensitivity. So it can be looked at as the probability that a relevant document is retrieved by the query.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

Definition (classification context)

In the context of classification tasks, the terms **true positives**, **true negatives**, **false positives** and **false negatives** are used to compare the given classification of an item (the class label assigned to the item by a classifier) with the desired correct classification (the class the item actually belongs to). This is illustrated by the table below:

		correct result / classification	
		E1	**E2**
obtained result / classification	**E1**	**tp** (true positive)	**fp** (false positive)
	E2	**fn** (false negative)	**tn** (true negative)

Precision and recall are then defined as:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Recall in this context is also referred to as the True Positive Rate, other related measures used in classification include True Negative Rate and Accuracy: . True Negative Rate is also called Specificity.

$$\text{True Negative Rate} = \frac{tn}{tn + fp}$$

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

Probabilistic interpretation

It is possible to interpret precision and recall not as ratios but as probabilities:

- **Precision** is the probability that a (randomly selected) retrieved document is relevant.
- **Recall** is the probability that a (randomly selected) relevant document is retrieved in a search.

F-measure

A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional F-measure or balanced F-score:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

It is a special case of the general F_β measure (for non-negative real values of β):

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

Two other commonly used F measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure $E = 1 - (1/(\alpha/P + (1 - \alpha)/R))$. Their relationship is $F_\beta = 1 - E$ where $\alpha = 1/(\beta^2 + 1)$.

Chapter- 4

Types of Machine Learning

Reinforcement learning

Inspired by old behaviorist psychology, **reinforcement learning** is an area of machine learning in computer science, concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of cumulative *reward*. The problem, due to its generality, is studied in many other disciplines, such as control theory, operations research, information theory, simulation-based optimization, statistics, and Genetic Algorithms. In the operations research and control literature the field where reinforcement learning methods are studied is called *approximate dynamic programming*. The problem has been studied in the theory of optimal control, though most studies there are concerned with existence of optimal solutions and their characterization, and not with the learning or approximation aspects. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

In machine learning, the environment is typically formulated as a Markov decision process (MDP), and many reinforcement learning algorithms for this context are highly related to dynamic programming techniques. The main difference to these classical techniques is that reinforcement learning algorithms do not need the knowledge of the MDP and they target large MDPs where exact methods become infeasible.

Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). The exploration vs. exploitation trade-off in reinforcement learning has been most thoroughly studied through the multi-armed bandit problem and in finite MDPs.

The basic reinforcement learning model consists of:

1. a set of environment states *S*;

2. a set of actions A;
3. rules of transitioning between states;
4. rules that determine the *scalar immediate reward* of a transition; and
5. rules that describe what the agent observes.

The rules are often stochastic. The observation typically involves the scalar immediate reward associated to the last transition. In many works, the agent is also assumed to observe the current environmental state, in which case we talk about *full observability*, whereas in the opposing case we talk about *partial observability*. Sometimes the set of actions available to the agent is restricted (e.g., you cannot spend more money than what you possess).

A reinforcement learning agent interacts with its environment in discrete time steps. At each time t, the agent receives an observation o_t, which typically includes the reward r_t. It then chooses an action a_t from the set of actions available, which is subsequently sent to the environment. The environment moves to a new state s_{t+1} and the reward r_{t+1} associated with the *transition* (s_t,a_t,s_{t+1}) is determined. The goal of a reinforcement learning agent is to collect as much reward as possible. The agent can choose any action as a function of the history and it can even randomize its action selection.

When the agent's performance is compared to that of an agent which acts optimally from the beginning, the difference in performance gives rise to the notion of *regret*. Note that in order to act near optimally, the agent must reason about the long term consequences of its actions: In order to maximize my future income I better go to school now, although the immediate monetary reward associated with this might be negative.

Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including robot control, elevator scheduling, telecommunications, backgammon and chess.

Two components make reinforcement learning powerful: The use of samples to optimize performance and the use of function approximation to deal with large environments. Thus, reinforcement learning is most successful when the environment is big or cannot be precisely described. However, reinforcement learning methods can also be applied when the environment is big, but can be reasonably simulated, a problem studied in simulation-based optimization.

Exploration

The reinforcement learning problem as described requires clever exploration mechanisms. Randomly selecting actions is known to give rise to very poor performance. The case of (small) finite MDPs is relatively well understood by now. However, due to the lack of algorithms that would provably scale well with the number of states (or scale to problems with infinite state spaces), in practice people resort to simple exploration methods. One such method is ε-greedy, when the agent chooses the action that it believes

has the best long-term effect with probability $1 - \varepsilon$, and it chooses an action uniformly at random, otherwise. Here, $0 < \varepsilon < 1$ is a tuning parameter, which is sometimes changed, either according to a fixed schedule (making the agent explore less as time goes by), or adaptively based on some heuristics (Tokic, 2010).

Algorithms for control learning

Even if the issue of exploration is disregarded and even if the state was observable (which we assume from now on), the problem remains to find out which actions are good based on past experience.

Criterion of optimality

For simplicity, assume for a moment that the problem studied is *episodic*, an episode ending when some *terminal state* is reached. Assume further that no matter what course of actions the agent takes, termination is inevitable with probability one. Under some additional mild regularity conditions the expectation of the total reward is then well-defined, for *any* policy π and any initial distribution over the states. Given a fixed initial distribution μ, we can thus assign the expected return ρ^π to policy π:

$$\rho^\pi = E[R \mid \pi],$$

where the random variable R denotes the *return* and is defined by

$$R = \sum_{t=0}^{N-1} r_{t+1},$$

where r_{t+1} is the reward received after the t-th transition, the initial state is sampled at random from μ and actions are selected by policy π. Here, N denotes the (random) time when a terminal state is reached, i.e., the time when the episode terminates.

In the case of non-episodic problems the return is often *discounted*,

$$R = \sum_{t=0}^{\infty} \gamma^t r_{t+1},$$

giving rise to the total expected discounted reward criterion. Here $0 \leq \gamma \leq 1$is the so-called *discount-factor*. Since the undiscounted return is a special case of the discounted return, from now on we will assume discounting. Although this looks innocent enough, discounting is in fact problematic if one cares about online performance. This is because discounting makes the initial time steps more important. Since a learning agent is likely to make mistakes during the first few steps after its "life" starts, no uninformed learning algorithm can achieve near-optimal performance under discounting even if the class of environments is restricted to that of finite MDPs. (This does not mean though that, given

enough time, a learning agent cannot figure how to act near-optimally, if time was restarted.)

The problem then is to specify an algorithm that can be used to find a policy with maximum expected return. From the theory of MDPs it is known that, without the loss of generality, the search can be restricted to the set of the so-called *stationary* policies. A policy is called stationary if the action-distribution returned by it depends only the last state visited (which is part of the observation history of the agent, by our simplifying assumption). In fact, the search can be further restricted to *deterministic* stationary policies. A deterministic stationary policy is one which deterministically selects actions based on the current state. Since any such policy can be identified with a mapping from the set of states to the set of action, these policies can be identified with such mappings with no loss of generality.

Brute force

The naive brute force approach entails the following two steps:

1. For each possible policy, sample returns while following it
2. Choose the policy with the largest expected return

One problem with this is that the number of policies can be extremely large, or even infinite. Another is that variance of the returns might be large, in which case a large number of samples will be required to accurately estimate the return of each policy.

These problems can be ameliorated if we assume some structure and perhaps allow samples generated from one policy to influence the estimates made for another. The two main approaches for achieving this are value function estimation and direct policy search.

Value function approaches

Value function approaches attempt to find a policy that maximizes the return by maintaining a set of estimates of expected returns for some policy (usually either the "current" or the optimal one).

These methods rely on the theory of MDPs, where optimality is defined in a sense which is stronger than the above one: A policy is called optimal if it achieves the best expected return from *any* initial state (i.e., initial distributions play no role in this definition). Again, one can always find an optimal policy amongst stationary policies.

To define optimality in a formal manner, define the value of a policy π by

$$V^{\pi}(s) = E[R \mid s,\pi],$$

where R stands for the random return associated with following π from the initial state s. Define $V^{*}(s)$ as the maximum possible value of $V^{\pi}(s)$, where π is allowed to change:

$$V^*(s) = \sup_\pi V^\pi(s).$$

A policy which achieves these *optimal values* in *each* state is called *optimal*. Clearly, a policy optimal in this strong sense is also optimal in the sense that it maximizes the expected return ρ^π, since $\rho^\pi = E[V^\pi(S)]$, where S is a state randomly sampled from the distribution μ.

Although state-values suffice to define optimality, it will prove to be useful to define action-values. Given a state s, an action a and a policy π, the action-value of the pair (s,a) under π is defined by

$$Q^\pi(s,a) = E[R|s,a,\pi],$$

where, now, R stands for the random return associated with first taking action a in state s and following π, thereafter.

It is well-known from the theory of MDPs that if someone gives us Q for an optimal policy, we can always choose optimal actions (and thus act optimally) by simply choosing the action with the highest value at each state. The *action-value function* of such an optimal policy is called the *optimal action-value function* and is denoted by Q^*. In summary, the knowledge of the optimal action-value function *alone* suffices to know how to act optimally.

Assuming full knowledge of the MDP, there are two basic approaches to compute the optimal action-value function, value iteration and policy iteration. Both algorithms compute a sequence of functions Q_k ($k = 0, 1, 2, \ldots$,) which converge to Q^*. Computing these functions involves computing expectations over the whole state-space, which is impractical for all, but the smallest (finite) MDPs, never mind the case when the MDP is unknown. In reinforcement learning methods the expectations are approximated by averaging over samples and one uses function approximation techniques to cope with the need to represent value functions over large state-action spaces.

Monte Carlo methods

The simplest Monte Carlo methods can be used in an algorithm that mimics policy iteration. Policy iteration consists of two steps: *policy evaluation* and *policy improvement*. The Monte Carlo methods are used in the policy evaluation step. In this step, given a stationary, deterministic policy π, the goal is to compute the function values $Q^\pi(s,a)$ (or a good approximation to them) for all state-action pairs (s,a). Assume (for simplicity) that the MDP is finite and in fact a table representing the action-values fits into the memory. Further, assume that the problem is episodic and after each episode a new one starts from some random initial state. Then, the estimate of the value of a given state-action pair (s,a)can be computed by simply averaging the sampled returns which originated from (s,a) over time. Given enough time, this procedure can thus construct a precise estimate Q of the action-value function Q^π. This finishes the description of the

policy evaluation step. In the policy improvement step, as it is done in the standard policy iteration algorithm, the next policy is obtained by computing a *greedy* policy with respect to *Q*: Given a state *s*, this new policy returns an action that maximizes $Q(s,\cdot)$. In practice one often avoids computing and storing the new policy, but uses lazy evaluation to defer the computation of the maximizing actions to when they are actually needed.

A few problems with this procedure are as follows:

- The procedure may waste too much time on evaluating a suboptimal policy;
- It uses samples inefficiently in that a long trajectory is used to improve the estimate only of the *single* state-action pair that started the trajectory;
- When the returns along the trajectories have *high variance*, convergence will be slow;
- It works in *episodic problems only*;
- It works in *small, finite MDPs only*.

Temporal difference methods

The first issue is easily corrected by allowing the procedure to change the policy (at all, or at some states) before the values settle. However good this sounds, this may be dangerous as this might prevent convergence. Still, most current algorithms implement this idea, giving rise to the class of *generalized policy iteration* algorithm. We note in passing that actor critic methods belong to this category.

The second issue can be corrected within the algorithm by allowing trajectories to contribute to any state-action pair in them. This may also help to some extent with the third problem, although a better solution when returns have high variance is to use Sutton's temporal difference (TD) methods which are based on the recursive Bellman equation. Note that the computation in TD methods can be incremental (when after each transition the memory is changed and the transition is thrown away), or batch (when the transitions are collected and then the estimates are computed once based on a large number of transitions). Batch methods, a prime example of which is the least-squares temporal difference method due to Bradtke and Barto (1996), may use the information in the samples better, whereas incremental methods are the only choice when batch methods become infeasible due to their high computational or memory complexity. In addition, there exist methods that try to unify the advantages of the two approaches. Methods based on temporal differences also overcome the second but last issue.

In order to address the last issue mentioned in the previous section, *function approximation methods* are used. In *linear function approximation* one starts with a mapping φ that assigns a finite dimensional vector to each state-action pair. Then, the action values of a state-action pair (s,a) are obtained by linearly combining the components of $\varphi(s,a)$ with some *weights* θ:

$$Q(s,a) = \sum_{i=1}^{d} \theta_i \phi_i(s,a)$$.

The algorithms then adjust the weights, instead of adjusting the values associated with the individual state-action pairs. However, linear function approximation is not the only choice. More recently, methods based on ideas from nonparametric statistics (which can be seen to construct their own features) have been explored.

So far, the discussion was restricted to how policy iteration can be used as a basis of the designing reinforcement learning algorithms. Equally importantly, value iteration can also be used as a starting point, giving rise to the Q-Learning algorithm (Watkins 1989) and its many variants.

The problem with methods that use action-values is that they may need highly precise estimates of the competing action values, which can be hard to obtain when the returns are noisy. Though this problem is mitigated to some extent by temporal difference methods and if one uses the so-called compatible function approximation method, more work remains to be done to increase generality and efficiency. Another problem specific to temporal difference methods comes from their reliance on the recursive Bellman equation. Most temporal difference methods have a so-called λ parameter $(0 \leq \lambda \leq 1)$ that allows one to continuously interpolate between Monte-Carlo methods (which do not rely on the Bellman equations) and the basic temporal difference methods (which rely entirely on the Bellman equations), which can thus be effective in palliating this issue.

Direct policy search

An alternative method to find a good policy is to search directly in (some subset) of the policy space, in which case the problem becomes an instance of stochastic optimization. The two approaches available are gradient-based and gradient-free methods.

Gradient-based methods (giving rise to the so-called *policy gradient methods*) start with a mapping from a finite dimensional (parameter) space to the space of policies: given the parameter vector θ, let π_θ denote the policy associated to θ. Define the performance function by

$$\rho(\theta) = \rho^{\pi_\theta}.$$

Under mild conditions this function will be differentiable as a function of the parameter vector θ. If the gradient of ρ was known, one could use gradient ascent. Since an analytic expression for the gradient is not available, one must rely on a noisy estimate. Such an estimate can be constructed in many ways, giving rise to algorithms like Williams' REINFORCE method (which is also known as the likelihood ratio method in the simulation-based optimization literature). Policy gradient methods have received a lot of attention in the last couple of years (e.g., Peters et al. (2003)), but they remain an active

field. The issue with many of these methods is that they may get stuck in local optima (as they are based on local search).

A large class of methods avoids relying on gradient information. These include simulated annealing, cross-entropy search or methods of evolutionary computation. Many gradient-free methods can achieve (in theory and in the limit) a global optimum. In a number of cases they have indeed demonstrated remarkable performance.

The issue with policy search methods is that they may converge slowly if the information based on which they act is noisy. For example, this happens when in episodic problems the trajectories are long and the variance of the returns is large. As argued beforehand, value-function based methods that rely on temporal differences might help in this case. In recent years, several actor-critic algorithms have been proposed following this idea and were demonstrated to perform well on various benchmarks.

Theory

The theory for small, finite MDPs is quite mature. Both the asymptotic and finite-sample behavior of most algorithms is well-understood. As mentioned beforehand, algorithms with provably good online performance (addressing the exploration issue) are known. The theory of large MDPs needs more work. Efficient exploration is largely untouched (except for the case of bandit problems). Although finite-time performance bounds appeared for many algorithms in the recent years, these bounds are expected to be rather loose and thus more work is needed to better understand the relative advantages, as well as the limitations of these algorithms. For incremental algorithm asymptotic convergence issues have been settled. Recently, new incremental, temporal-difference-based algorithms have appeared which converge under a much wider set of conditions than was previously possible (for example, when used with arbitrary, smooth function approximation).

Current research

Current research topics include: adaptive methods which work with fewer (or no) parameters under a large number of conditions, addressing the exploration problem in large MDPs, large scale empirical evaluations, learning and acting under partial information (e.g., using Predictive State Representation), modular and hierarchical reinforcement learning, improving existing value-function and policy search methods, algorithms that work well with large (or continuous) action spaces, transfer learning, lifelong learning, efficient sample-based planning (e.g., based on Monte-Carlo tree search). Multiagent or Distributed Reinforcement Learning is also a topic of interest in current research. There is also a growing interest in real life applications of reinforcement learning. Successes of reinforcement learning are collected on here and here.

Reinforcement learning algorithms such as TD learning are also being investigated as a model for Dopamine-based learning in the brain. In this model, the dopaminergic

projections from the substantia nigra to the basal ganglia function as the prediction error. Reinforcement learning has also been used as a part of the model for human skill learning, especially in relation to the interaction between implicit and explicit learning in skill acquisition (the first publication on this application was in 1995-1996, and there have been many follow-up studies).

Literature

Conferences, journals

Most reinforcement learning papers are published at the major machine learning and AI conferences (ICML, NIPS, AAAI, IJCAI, UAI, AI and Statistics) and journals (JAIR, JMLR, Machine learning journal). Some theory papers are published at COLT and ALT. However, many papers appear in robotics conferences (IROS, ICRA) and the "agent" conference AAMAS. Operations researchers publish their papers at the INFORMS conference and, for example, in the Operation Research, and the Mathematics of Operations Research journals. Control researchers publish their papers at the CDC and ACC conferences, or, e.g., in the journals IEEE Transactions on Automatic Control, or Automatica, although applied works tend to be published in more specialized journals. The Winter Simulation Conference also publishes many relevant papers. Other than this, papers also published in the major conferences of the neural networks, fuzzy, and evolutionary computation communities. The annual IEEE symposium titled Approximate Dynamic Programming and Reinforcement Learning (ADPRL) and the biannual European Workshop on Reinforcement Learning (EWRL) are two regularly held meetings where RL researchers meet.

Unsupervised learning

In machine learning, **unsupervised learning** is a class of problems in which one seeks to determine how the data are organized. Many methods employed here are based on data mining methods used to preprocess data. It is distinguished from supervised learning (and reinforcement learning) in that the learner is given only unlabeled examples.

Unsupervised learning is closely related to the problem of density estimation in statistics. However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data.

One form of unsupervised learning is clustering. Another example is blind source separation based on Independent Component Analysis (ICA).

Among neural network models, the Self-organizing map (SOM) and Adaptive resonance theory (ART) are commonly used unsupervised learning algorithms. The SOM is a topographic organization in which nearby locations in the map represent inputs with similar properties. The ART model allows the number of clusters to vary with problem size and lets the user control the degree of similarity between members of the same

clusters by means of a user-defined constant called the vigilance parameter. ART networks are also used for many pattern recognition tasks, such as automatic target recognition and seismic signal processing. The first version of ART was "ART1", developed by Carpenter and Grossberg(1988).

Semi-supervised learning

In computer science, **semi-supervised learning** is a class of machine learning techniques that make use of both labeled and unlabeled data for training - typically a small amount of labeled data with a large amount of unlabeled data. Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. The acquisition of labeled data for a learning problem often requires a skilled human agent to manually classify training examples. The cost associated with the labeling process thus may render a fully labeled training set infeasible, whereas acquisition of unlabeled data is relatively inexpensive. In such situations, semi-supervised learning can be of great practical value.

One example of a semi-supervised learning technique is co-training, in which two or possibly more learners are each trained on a set of examples, but with each learner using a different, and ideally independent, set of features for each example.

An alternative approach is to model the joint probability distribution of the features and the labels. For the unlabelled data the labels can then be treated as 'missing data'. Techniques that handle missing data, such as Gibbs sampling or the EM algorithm, can then be used to estimate the parameters of the model.

Transduction (machine learning)

In logic, statistical inference, and supervised learning, **transduction** or **transductive inference** is reasoning from observed, specific (training) cases to specific (test) cases. In contrast, induction is reasoning from observed training cases to general rules, which are then applied to the test cases. The distinction is most interesting in cases where the predictions of the transductive model are not achievable by any inductive model. Note that this is caused by transductive inference on different test sets producing mutually inconsistent predictions.

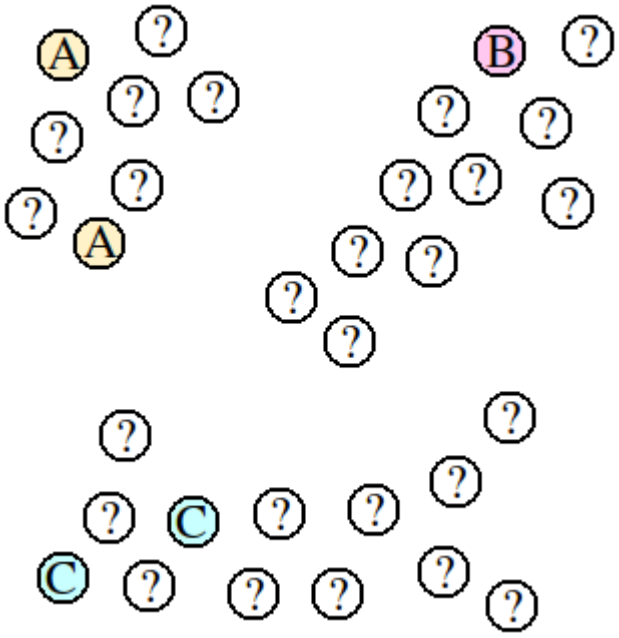
Transduction was introduced by Vladimir Vapnik in the 1990s, motivated by his view that transduction is preferable to induction since, according to him, induction requires solving a more general problem (inferring a function) before solving a more specific problem (computing outputs for new cases): "When solving a problem of interest, do not solve a more general problem as an intermediate step. Try to get the answer that you really need but not a more general one."

An example of learning which is not inductive would be in the case of binary classification, where the inputs tend to cluster in two groups. A large set of test inputs may help in finding the clusters, thus providing useful information about the classification labels. The same predictions would not be obtainable from a model which induces a function based only on the training cases. Some people may call this an example of the closely related semi-supervised learning, since Vapnik's motivation is quite different. An example of an algorithm in this category is the Transductive Support Vector Machine (TSVM).

A third possible motivation which leads to transduction arises through the need to approximate. If exact inference is computationally prohibitive, one may at least try to make sure that the approximations are good at the test inputs. In this case, the test inputs could come from an arbitrary distribution (not necessarily related to the distribution of the training inputs), which wouldn't be allowed in semi-supervised learning. An example of an algorithm falling in this category is the Bayesian Committee Machine (BCM).

Example Problem

The following example problem contrasts some of the unique properties of transduction against induction.



A collection of points is given, such that some of the points are labeled (A, B, or C), but most of the points are unlabeled (?). The goal is to predict appropriate labels for all of the unlabeled points.

The inductive approach to solving this problem is to use the labeled points to train a supervised learning algorithm, and then have it predict labels for all of the unlabeled

points. With this problem, however, the supervised learning algorithm will only have five labeled points to use as a basis for building a predictive model. It will certainly struggle to build a model that captures the structure of this data. For example, if a nearest-neighbor algorithm is used, then the points near the middle will be labeled "A" or "C", even though it is apparent that they belong to the same cluster as the point labeled "B".

Transduction has the advantage of being able to consider all of the points, not just the labeled points, while performing the labeling task. In this case, transductive algorithms would label the unlabeled points according to the clusters to which they naturally belong. The points in the middle, therefore, would most likely be labeled "B", because they are packed very close to that cluster.

An advantage of transduction is that it may be able to make better predictions with fewer labeled points, because it uses the natural breaks found in the unlabeled points. One disadvantage of transduction is that it builds no predictive model. If a previously unknown point is added to the set, the entire transductive algorithm would need to be repeated with all of the points in order to predict a label. This can be computationally expensive if the data is made available incrementally in a stream. Further, this might cause the predictions of some of the old points to change (which may be good or bad, depending on the application). A supervised learning algorithm, on the other hand, can label new points instantly, with very little computational cost.

Transduction Algorithms

Transduction algorithms can be broadly divided into two categories: 1- Those that seek to assign discrete labels to unlabeled points, and 2- Those that seek to regress continuous labels for unlabeled points. Algorithms that seek to predict discrete labels tend to be derived by adding partial-supervision to a clustering algorithm. These can be further sub-divided into two categories: those that cluster by partitioning, and those that cluster by agglomerating. Algorithms that seek to predict continuous labels tend to be derived by adding partial-supervision to a manifold learning algorithm.

Partitioning Transduction

Partitioning-transduction can be thought of as top-down transduction. It is a semi-supervised extension of partition-based clustering. It is typically performed as follows:

```
Consider the set of all points to be one large partition.
while any partition P contains two points with conflicting labels:
  Partition P into smaller partitions.
for each partition P:
  Assign the same label to all of the points in P.
```

Of course, any reasonable partitioning technique could be used with this algorithm. Max flow min cut partitioning schemes are very popular for this purpose.

Agglomerative Transduction

Agglomerative transduction can be thought of as bottom-up transduction. It is a semi-supervised extension of agglomerative clustering. It is typically performed as follows:

```
Compute the pair-wise distances, D, between all the points.
Sort D in ascending order.
Consider each point to be a cluster of size 1.
for each pair of points {a,b} in D:
  if (a is unlabeled) or (b is unlabeled) or (a and b have the same
label)
    Merge the two clusters that contain a and b.
    Label all points in the merged cluster with the same label.
```

Manifold Transduction

Manifold-learning-based transduction is still a very young field of research.

Multi-task learning

Multi-task learning is an approach to machine learning, that learns a problem together with other related problems at the same time, using a shared representation. This often leads to a better model for the main task, because it allows the learner to use the commonality among the tasks. Therefore, multi-task learning is a kind of inductive transfer.

Chapter- 5

Computational Learning Theory

In theoretical computer science, **computational learning theory** is a mathematical field related to the analysis of machine learning algorithms.

Overview

Theoretical results in machine learning mainly deal with a type of inductive learning called supervised learning. In supervised learning, an algorithm is given samples that are labeled in some useful way. For example, the samples might be descriptions of mushrooms, and the labels could be whether or not the mushrooms are edible. The algorithm takes these previously labeled samples and uses them to induce a classifier. This classifier is a function that assigns labels to samples including the samples that have never been previously seen by the algorithm. The goal of the supervised learning algorithm is to optimize some measure of performance such as minimizing the number of mistakes made on new samples.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results:

- Positive results – Showing that a certain class of functions is learnable in polynomial time.
- Negative results – Showing that certain classes cannot be learned in polynomial time.

Negative results are proven only by assumption. The assumptions that are common in negative results are:

- Computational complexity - $P \neq NP$
- Cryptographic - One-way functions exist.

There are several different approaches to computational learning theory. These differences are based on making assumptions about the inference principles used to

generalize from limited data. This includes different definitions of probability and different assumptions on the generation of samples. The different approaches include:

Probably approximately correct learning

In computational learning theory, **probably approximately correct learning** (**PAC learning**) is a framework for mathematical analysis of machine learning. It was proposed in 1984 by Leslie Valiant.

In this framework, the learner receives samples and must select a generalization function (called the *hypothesis*) from a certain class of possible functions. The goal is that, with high probability (the "probably" part), the selected function will have low generalization error (the "approximately correct" part). The learner must be able to learn the concept given any arbitrary approximation ratio, probability of success, or distribution of the samples.

The model was later extended to treat noise (misclassified samples).

An important innovation of the PAC framework is the introduction of computational complexity theory concepts to machine learning. In particular, the learner is expected to find efficient functions (time and space requirements bounded to a polynomial of the example size), and the learner itself must implement an efficient procedure (requiring an example count bounded to a polynomial of the concept size, modified by the approximation and likelihood bounds).

Definitions and terminology

In order to give the definition for something that is PAC-learnable, we first have to introduce some terminology.

For the following definitions, two examples will be used. The first is the problem of character recognition given an array of n bits. The other example is the problem of finding an interval that will correctly classify points within the interval as positive and the points outside of the range as negative.

Let X be a set called the *instance space* or the encoding of all the samples. In the character recognition problem, the instance space is $X = \{0,1\}^n$. In the interval problem the instance space is $X = \mathbb{R}$, where $\mathbb{R}$ denotes the set of all real numbers.

A *concept* is a subset $c \subset X$. One concept is the set of all of the bits that encode for the letter "P" in $X = \{0,1\}^n$. An example concept from the second example is the set of all of the numbers between $\pi / 2$ and $\sqrt{10}$. A *concept class* is a set of concepts over X. This could be the set of all of the array of bits that are skeletonized 4-connected (width of the font is 1).

Let $EX(c,D)$ be a procedure that draws an example, x, using a probability distribution D and gives the correct label $c(x)$.

Say that there is an algorithm A that given access to $EX(c,D)$ and inputs ε and δ that, with probability of at least $1 - \delta$, A outputs a hypothesis $h \in C$ that has error less than or equal to ε with examples drawn from X with the distribution D. If there is such an algorithm for every concept $c \in C$, for every distribution D over X, and for all $0 < \varepsilon < 1 / 2$ and $0 < \delta < 1 / 2$ then C is **PAC learnable**. We can also say that A is a **PAC learning algorithm** for C.

Vapnik–Chervonenkis theory

Vapnik–Chervonenkis theory (also known as **VC theory**) was developed during 1960–1990 by Vladimir Vapnik and Alexey Chervonenkis. The theory is a form of computational learning theory, which attempts to explain the learning process from a statistical point of view.

VC theory is related to **statistical learning theory** and to empirical processes. Richard M. Dudley and Vladimir Vapnik himself, among others, apply VC-theory to empirical processes.

VC theory covers at least four parts (as explained in *The Nature of Statistical Learning Theory*):

- Theory of consistency of learning processes
  - What are (necessary and sufficient) conditions for consistency of a learning process based on the empirical risk minimization principle ?
- Nonasymptotic theory of the rate of convergence of learning processes
  - How fast is the rate of convergence of the learning process?
- Theory of controlling the generalization ability of learning processes
  - How can one control the rate of convergence (the generalization ability) of the learning process?
- Theory of constructing learning machines
  - How can one construct algorithms that can control the generalization ability?

In addition, VC theory and VC dimension are instrumental in the theory of empirical processes, in the case of processes indexed by VC classes.

The last part of VC theory introduced a well-known learning algorithm: the support vector machine.

VC theory contains important concepts such as the VC dimension and structural risk minimization. This theory is related to mathematical subjects such as:

- reproducing kernel Hilbert spaces
- regularization networks
- kernels
- empirical processes

Algorithmic learning theory

Algorithmic learning theory (or **algorithmic inductive inference**) is a framework for machine learning.

The framework was introduced in E. Mark Gold's seminal paper "Language identification in the limit". The objective of language identification is for a machine running one program to be capable of developing another program by which any given sentence can be tested to determine whether it is "grammatical" or "ungrammatical". The language being learned need not be English or any other natural language - in fact the definition of "grammatical" can be absolutely anything known to the tester.

In the framework of algorithmic learning theory, the tester gives the learner an example sentence at each step, and the learner responds with a hypothesis, which is a suggested program to determine grammatical correctness. It is required of the tester that every possible sentence (grammatical or not) appears in the list eventually, but no particular order is required. It is required of the learner that at each step the hypothesis must be correct for all the sentences so far.

A particular learner is said to be able to "learn a language in the limit" if there is a certain number of steps beyond which its hypothesis no longer changes. At this point it has indeed learned the language, because every possible sentence appears somewhere in the sequence of inputs (past or future), and the hypothesis is correct for all inputs (past or future), so the hypothesis is correct for every sentence. The learner is not required to be able to tell when it has reached a correct hypothesis, all that is required is that it be true.

Gold showed that any language which is defined by a Turing machine program can be learned in the limit by another Turing-complete machine using enumeration. This is done by the learner testing all possible Turing machine programs in turn until one is found which is correct so far - this forms the hypothesis for the current step. Eventually, the correct program will be reached, after which the hypothesis will never change again (but note that the learner does not know that it won't need to change).

Gold also showed that if the learner is given only positive examples (that is, only grammatical sentences appear in the input, not ungrammatical sentences), then the language can only be guaranteed to be learned in the limit if there are only a finite number of possible sentences in the language (this is possible if, for example, sentences are known to be of limited length).

Language identification in the limit is a very theoretical model. It does not allow for limits of runtime or computer memory which can occur in practice, and the enumeration method may fail if there are errors in the input. However the framework is very powerful, because if these strict conditions are maintained, it allows the learning of any program known to be computable. This is because a Turing machine program can be written to mimic any program in any conventional programming language.

Other frameworks of learning consider a much more restricted class of function than Turing machines, but complete the learning more quickly (in polynomial time). An example of such a framework is Probably approximately correct learning.

Online machine learning

In machine learning, **online learning** is a model of induction that learns one instance at a time. The goal in online learning is to predict labels for instances. For example, the instances could describe the current conditions of the stock market, and an online algorithm predicts tomorrow's value of a particular stock. The key defining characteristic of online learning is that soon after the prediction is made, the true label of the instance is discovered. This information can then be used to refine the prediction hypothesis used by the algorithm. The goal of the algorithm is to make predictions that are close to the true labels.

More formally, an online algorithm proceeds in a sequence of trials. Each trial can be decomposed into three steps. First the algorithm receives an instance. Second the algorithm predicts the label of the instance. Third the algorithm receives the true label of the instance. The third stage is the most crucial as the algorithm can use this label feedback to update its hypothesis for future trials. The goal of the algorithm is to minimize some performance criteria. For example, with stock market prediction the algorithm may attempt to minimize sum of the square distances between the predicted and true value of a stock. Another popular performance criteria is to minimize the number of mistakes when dealing with classification problems.

Because online learning algorithms continually receive label feedback, the algorithms are able to adapt and learn in difficult situations. Many online algorithms can give strong guarantees on performance even when the instances are not generated by a distribution. As long a reasonably good classifier exists, the online algorithm will learn to predict correct labels. This good classifier must come from a previously determined set that depends on the algorithm. For example, two popular on-line algorithms perceptron and winnow can perform well when a hyperplane exists that splits the data into two categories. These algorithms can even be modified to do provably well even if the hyperplane is allowed to infrequently change during the online learning trials.

Unfortunately, the main difficulty of online learning is also a result of the requirement for continual label feedback. For many problems it is not possible to guarantee that accurate label feedback will be available in the near future. For example, when designing a system

that learns how to do optical character recognition, typically some expert must label previous instances to help train the algorithm. In actual use of the OCR application, the expert is no longer available and no inexpensive outside source of accurate labels is available. Fortunately, there is a large class of problems where label feedback is always available. For any problem that consists of predicting the future, an online learning algorithm just needs to wait for the label to become available. This is true in our previous example of stock market prediction and many other problems.

Computational learning theory has led to several practical algorithms. For example, PAC theory inspired boosting, VC theory led to support vector machines, and Bayesian inference led to belief networks (by Judea Pearl).

Chapter- 6

Support Vector Machine

Support vector machines (**SVMs**) are a set of related supervised learning methods that analyze data and recognize patterns, used for classification and regression analysis. The original SVM algorithm was invented by Vladimir Vapnik and the current standard incarnation (soft margin) was proposed by Corinna Cortes and Vladimir Vapnik . The standard SVM takes a set of input data, and predicts, for each given input, which of two possible classes the input is a member of, which makes the SVM a non-probabilistic binary linear classifier. Since an SVM is a classifier, then given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Intuitively, an SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

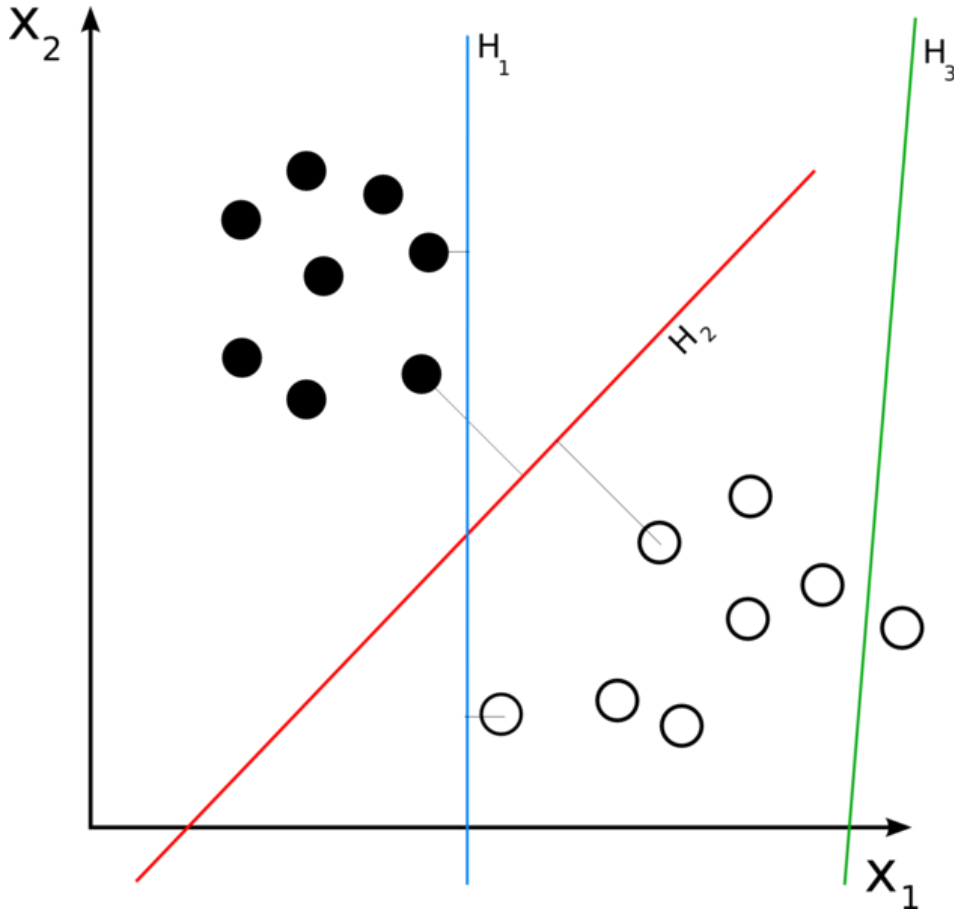
Whereas the original problem may be stated in a finite dimensional space, it often happens that in that space the sets to be discriminated are not linearly separable. For this reason it was proposed that the original finite dimensional space be mapped into a much higher dimensional space presumably making the separation easier in that space. SVM schemes use a mapping into a larger space so that cross products may be computed easily in terms of the variables in the original space making the computational load reasonable. The cross products in the larger space are defined in terms of a kernel function $K(x,y)$ which can be selected to suit the problem. The hyperplanes in the large space are defined as the set of points whose cross product with a vector in that space is constant. The

vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors which occur in the data base. With this choice of a hyperplane the points x in the feature space which are mapped into the hyperplane are defined by the relation:

$$\sum_i \alpha_i K(x_i, x) = constant$$

Note that if $K(x,y)$ becomes small as y grows further from x, each element in the sum measures the degree of closeness of the test point x to the corresponding data base point x_i. In this way the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of points x mapped into any hyperplane can be quite convoluted as a result allowing much more complex discrimination between sets which are far from convex in the original space.

Motivation



H3 (green) doesn't separate the 2 classes. H1 (blue) does, with a small margin and H2 (red) with the maximum margin.

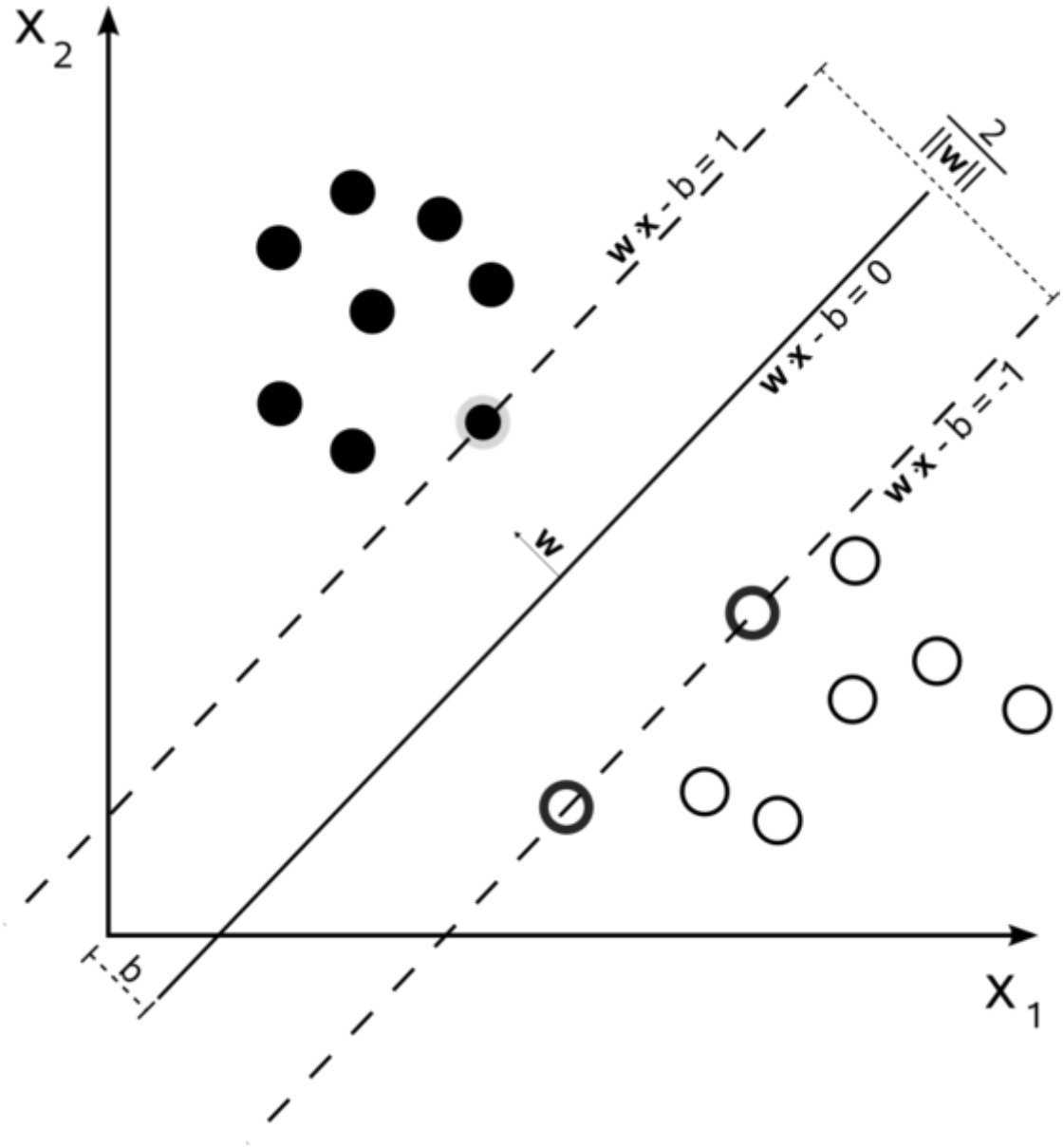
Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a *new* data point will be in. In the case of support vector machines, a data point is viewed as a *p*-dimensional vector (a list of *p* numbers), and we want to know whether we can separate such points with a $p-1$-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the *maximum-margin hyperplane* and the linear classifier it defines is known as a *maximum margin classifier*.

Formalization

We are given some training data $\mathcal{D}$, a set of *n* points of the form

$$\mathcal{D} = \{(\mathbf{x}_i, c_i) \mid \mathbf{x}_i \in \mathbb{R}^p,\ c_i \in \{-1, 1\}\}_{i=1}^{n}$$

where the c_i is either 1 or −1, indicating the class to which the point $\mathbf{x}_i$ belongs. Each $\mathbf{x}_i$ is a *p*-dimensional real vector. We want to find the maximum-margin hyperplane that divides the points having $c_i = 1$ from those having $c_i = -1$. Any hyperplane can be written as the set of points $\mathbf{x}$ satisfying



Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

$$\mathbf{w} \cdot \mathbf{x} - b = 0,$$

where $\cdot$ denotes the dot product. The vector $\mathbf{w}$ is a normal vector: it is perpendicular to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$.

We want to choose the $\mathbf{w}$ and b to maximize the margin, or distance between the parallel hyperplanes that are as far apart as possible while still separating the data. These hyperplanes can be described by the equations

$$\mathbf{w} \cdot \mathbf{x} - b = 1$$

and

$$\mathbf{w} \cdot \mathbf{x} - b = -1.$$

Note that if the training data are linearly separable, we can select the two hyperplanes of the margin in a way that there are no points between them and then try to maximize their distance. By using geometry, we find the distance between these two hyperplanes is $\frac{2}{\|\mathbf{w}\|}$, so we want to minimize $\|\mathbf{w}\|$. As we also have to prevent data points falling into the margin, we add the following constraint: for each *i* either

$$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1 \qquad \text{for } \mathbf{x}_i \text{of the first class}$$

or

$$\mathbf{w} \cdot \mathbf{x}_i - b \leq -1 \qquad \text{for } \mathbf{x}_i \text{of the second.}$$

This can be rewritten as:

$$c_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, \quad \text{for all } 1 \leq i \leq n. \qquad (1)$$

We can put this together to get the optimization problem:

Minimize (in $\mathbf{w}, b$)

$$\|\mathbf{w}\|$$

subject to (for any $i = 1, \ldots, n$)

$$c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1.$$

Primal form

The optimization problem presented in the preceding section is difficult to solve because it depends on ||**w**||, the norm of **w**, which involves a square root. Fortunately it is possible to alter the equation by substituting ||**w**|| with $\frac{1}{2}\|\mathbf{w}\|^2$ (the factor of 1/2 being used for mathematical convenience) without changing the solution (the minimum of the original and the modified equation have the same **w** and b). This is a quadratic programming (QP) optimization problem. More clearly:

Minimize (in $\mathbf{w}, b$)

$$\frac{1}{2}\|\mathbf{w}\|^2$$

subject to (for any $i = 1, \ldots, n$)

$$c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1.$$

One could be tempted to express the previous problem by means of non-negative Lagrange multipliers α_i as

$$\min_{\mathbf{w},b,\boldsymbol{\alpha}} \{\frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [c_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1]\}$$

but this would be wrong. The reason is the following: suppose we can find a family of hyperplanes which divide the points; then all $c_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1 \geq 0$. Hence we could find the minimum by sending all α_i to $+\infty$, and this minimum would be reached for all the members of the family, not only for the best one which can be chosen solving the original problem.

Nevertheless the previous constrained problem can be expressed as

$$\min_{\mathbf{w},b} \max_{\boldsymbol{\alpha}} \{\frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [c_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1]\}$$

that is we look for a saddle point. In doing so all the points which can be separated as $c_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1 > 0$ do not matter since we must set the corresponding α_i to zero.

This problem can now be solved by standard quadratic programming techniques and programs. The solution can be expressed by terms of linear combination of the training vectors as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i c_i \mathbf{x_i}$$

Only a few α_i will be greater than zero. The corresponding $\mathbf{x_i}$ are exactly the *support vectors*, which lie on the margin and satisfy $c_i(\mathbf{w} \cdot \mathbf{x_i} - b) = 1$. From this one can derive that the support vectors also satisfy

$$\mathbf{w} \cdot \mathbf{x_i} - b = 1/c_i = c_i \iff b = \mathbf{w} \cdot \mathbf{x_i} - c_i$$

which allows one to define the offset *b*. In practice, it is more robust to average over all N_{SV} support vectors:

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (\mathbf{w} \cdot \mathbf{x_i} - c_i)$$

Dual form

Writing the classification rule in its unconstrained dual form reveals that the maximum margin hyperplane and therefore the classification task is only a function of the *support vectors*, the training data that lie on the margin.

Using the fact, that $\|\mathbf{w}\|^2 = w \cdot w$ and substituting $\mathbf{w} = \sum_{i=1}^{n} \alpha_i c_i \mathbf{x_i}$, one can show that the dual of the SVM reduces to the following optimization problem:

Maximize (in α_i)

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j c_i c_j \mathbf{x}_i^T \mathbf{x}_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j c_i c_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to (for any $i = 1, \ldots, n$)

$$\alpha_i \geq 0,$$

and to the constraint from the minimization in *b*

$$\sum_{i=1}^{n} \alpha_i c_i = 0.$$

Here the kernel is defined by $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$.

The α terms constitute a dual representation for the weight vector in terms of the training set:

$$\mathbf{w} = \sum_i \alpha_i c_i \mathbf{x}_i.$$

Biased and unbiased hyperplanes

For simplicity reasons, sometimes it is required that the hyperplane passes through the origin of the coordinate system. Such hyperplanes are called *unbiased*, whereas general hyperplanes not necessarily passing through the origin are called *biased*. An unbiased

hyperplane can be enforced by setting $b = 0$ in the primal optimization problem. The corresponding dual is identical to the dual given above without the equality constraint

$$\sum_{i=1}^{n} \alpha_i c_i = 0.$$

Transductive support vector machines

Transductive support vector machines extend SVMs in that they could also treat partially labeled data in semi-supervised learning. Here, in addition to the training set $\mathcal{D}$, the learner is also given a set

$$\mathcal{D}^\star = \{\mathbf{x}_i^\star | \mathbf{x}_i^\star \in \mathbb{R}^p\}_{i=1}^k$$

of test examples to be classified. Formally, a transductive support vector machine is defined by the following primal optimization problem:

Minimize (in $\mathbf{w}, b, \mathbf{c}^\star$)

$$\frac{1}{2}\|\mathbf{w}\|^2$$

subject to (for any $i = 1, \ldots, n$ and any $j = 1, \ldots, k$)

$$c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1,$$
$$c_j^\star(\mathbf{w} \cdot \mathbf{x_j^\star} - b) \geq 1,$$

and

$$c_j^\star \in \{-1, 1\}.$$

Transductive support vector machines were introduced by Vladimir Vapnik in 1998.

Properties

SVMs belong to a family of generalized linear classifiers. They can also be considered a special case of Tikhonov regularization. A special property is that they simultaneously minimize the empirical *classification error* and maximize the *geometric margin*; hence they are also known as **maximum margin classifiers**.

A comparison of the SVM to other classifiers has been made by Meyer, Leisch and Hornik.

Extensions to the linear SVM

Soft margin

In 1995, Corinna Cortes and Vladimir Vapnik suggested a modified maximum margin idea that allows for mislabeled examples. If there exists no hyperplane that can split the "yes" and "no" examples, the *Soft Margin* method will choose a hyperplane that splits the examples as cleanly as possible, while still maximizing the distance to the nearest cleanly split examples. The method introduces slack variables, ξ_i, which measure the degree of misclassification of the datum x_i

$$c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1 - \xi_i \quad 1 \leq i \leq n. \qquad (2)$$

The objective function is then increased by a function which penalizes non-zero ξ_i, and the optimization becomes a trade off between a large margin, and a small error penalty. If the penalty function is linear, the optimization problem becomes:

$$\min_{\mathbf{w},\xi} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right\}$$

subject to (for any $i = 1, \ldots n$)

$$c_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1 - \xi_i, \qquad \xi_i \geq 0.$$

This constraint in (2) along with the objective of minimizing $\|\mathbf{w}\|$ can be solved using Lagrange multipliers as done above. One has then to solve the following problem

$$\min_{\mathbf{w},\xi,b} \max_{\alpha,\beta} \left\{ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i [c_i(\mathbf{w} \cdot \mathbf{x_i} - b) - 1 + \xi_i] - \sum_{i=1}^{n} \beta_i \xi_i \right\}$$

with $\alpha_i, \beta_i \geq 0$.

The key advantage of a linear penalty function is that the slack variables vanish from the dual problem, with the constant *C* appearing only as an additional constraint on the Lagrange multipliers. For the above formulation and its huge impact in practice, Cortes and Vapnik received the 2008 ACM Paris Kanellakis Award . Nonlinear penalty functions have been used, particularly to reduce the effect of outliers on the classifier, but unless care is taken, the problem becomes non-convex, and thus it is considerably more difficult to find a global solution.

Nonlinear classification

The original optimal hyperplane algorithm proposed by Vladimir Vapnik in 1963 was a linear classifier. However, in 1992, Bernhard Boser, Isabelle Guyon and Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick (originally proposed by Aizerman et al.) to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; thus though the classifier is a hyperplane in the high-dimensional feature space, it may be nonlinear in the original input space.

If the kernel used is a Gaussian radial basis function, the corresponding feature space is a Hilbert space of infinite dimension. Maximum margin classifiers are well regularized, so the infinite dimension does not spoil the results. Some common kernels include:

- Polynomial (homogeneous): $k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$
- Polynomial (inhomogeneous): $k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$
- Gaussian or Radial Basis Function: $k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$, for $\gamma > 0$. Sometimes parametrized using $\gamma = 1 / 2\sigma^2$
- Hyperbolic tangent: $k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\kappa\mathbf{x_i} \cdot \mathbf{x_j} + c)$, for some (not every) $\kappa > 0$ and $c < 0$

The kernel is related to the transform $\varphi(\mathbf{x_i})$ by the equation $k(\mathbf{x_i}, \mathbf{x_j}) = \varphi(\mathbf{x_i}) \cdot \varphi(\mathbf{x_j})$. The value **w** is also in the transformed space, with $\mathbf{w} = \sum_i \alpha_i c_i \varphi(\mathbf{x}_i)$. Dot products with **w** for classification can again be computed by the kernel trick, i.e. $\mathbf{w} \cdot \varphi(\mathbf{x}) = \sum_i \alpha_i c_i k(\mathbf{x}_i, \mathbf{x})$. However, there does not in general exist a value **w'** such that $\mathbf{w} \cdot \varphi(\mathbf{x}) = k(\mathbf{w}', \mathbf{x})$.

Parameter selection

The effectiveness of SVM depends on the selection of kernel, the kernel's parameters, and soft margin parameter C.

A common choice is a Gaussian kernel, which has a single parameter γ. Best combination of C and γ is often selected by a grid-search with exponentially growing sequences of C and γ, for example, $C \in \{2^{-5}, 2^{-3}, \ldots, 2^{13}, 2^{15}\}$; $\gamma \in \{2^{-15}, 2^{-13}, \ldots, 2^{1}, 2^{3}\}$. Typically, each combination of parameter choices is checked using cross validation, and the parameters with best cross-validation accuracy are picked. The final model, which is used for testing and for classifying new data, is then trained on the whole training set using the selected parameters.

Issues

Potential drawbacks of the SVM are the following two aspects:

- Uncalibrated Class membership probabilities
- The SVM is only directly applicable for two-class tasks. Therefore, algorithms that reduce the multi-class task to several binary problems have to be applied.

Multiclass SVM

Multiclass SVM aims to assign labels to instances by using support vector machines, where the labels are drawn from a finite set of several elements. The dominating approach for doing so is to reduce the single multiclass problem into multiple binary classification problems. Each of the problems yields a binary classifier, which is assumed to produce an output function that gives relatively large values for examples from the positive class and relatively small values for examples belonging to the negative class. Two common methods to build such binary classifiers are where each classifier distinguishes between (i) one of the labels to the rest (one-versus-all) or (ii) between every pair of classes (one-versus-one). Classification of new instances for one-versus-all case is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class (it is important that the output functions be calibrated to produce comparable scores). For the one-versus-one approach, classification is done by a max-wins voting strategy, in which every classifier assigns the instance to one of the two classes, then the vote for the assigned class is increased by one vote, and finally the class with most votes determines the instance classification.

Structured SVM

SVMs have been generalized to structured SVMs, where the label space is structured and of possibly infinite size.

Regression

A version of SVM for regression was proposed in 1996 by Vladimir Vapnik, Harris Drucker, Chris Burges, Linda Kaufman and Alex Smola. This method is called support vector regression (SVR). The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction (within a threshold ε). There is also a least squares version of support vector machine (SVM) called least squares support vector machine (LS-SVM) proposed in Suykens and Vandewalle.

Implementation

The parameters of the maximum-margin hyperplane are derived by solving the optimization. There exist several specialized algorithms for quickly solving the QP problem that arises from SVMs, mostly reliant on heuristics for breaking the problem down into smaller, more-manageable chunks. A common method for solving the QP problem is Platt's Sequential Minimal Optimization (SMO) algorithm, which breaks the problem down into 2-dimensional sub-problems that may be solved analytically, eliminating the need for a numerical optimization algorithm.

Another approach is to use an interior point method that uses Newton-like iterations to find a solution of the Karush-Kuhn-Tucker conditions of the primal and dual problems. Instead of solving a sequence of broken down problems, this approach directly solves the problem as a whole. To avoid solving a linear system involving the large kernel matrix, a low rank approximation to the matrix is often used to use the kernel trick.

Chapter- 7

Machine Learning Concepts & Methods

Backpropagation

Backpropagation, or **propagation of error**, is a common method of teaching artificial neural networks how to perform a given task. It was first described by Arthur E. Bryson and Yu-Chi Ho in 1969, but it wasn't until 1986, through the work of David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams, that it gained recognition, and it led to a “renaissance” in the field of artificial neural network research.

It is a supervised learning method, and is an implementation of the Delta rule. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Backpropagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

Summary

For better understanding, the backpropagation learning algorithm can be divided into two phases: propagation and weight update.

Phase 1: Propagation

Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
2. Back propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

This ratio influences the speed and quality of learning; it is called the *learning rate*. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

Repeat the phase 1 and 2 until the performance of the network is good enough.

Modes of learning

There are basically two modes of learning to choose from, one is on-line learning and the other is batch learning. In on-line learning, each propagation is followed immediately by a weight update. In batch learning, many propagations occur before weight updating occurs. Batch learning requires more memory capacity, but on-line learning requires more updates.

Algorithm

Actual algorithm for a 3-layer network (only one hidden layer):

```
  Initialize the weights in the network (often randomly)
  Do
         For each example e in the training set
              O = neural-net-output(network, e) ; forward pass
              T = teacher output for e
              Calculate error (T - O) at the output units
              Compute delta_wh for all weights from hidden layer to
output layer ; backward pass
              Compute delta_wi for all weights from input layer to
hidden layer ; backward pass continued
              Update the weights in the network
  Until all examples classified correctly or stopping criterion
satisfied
  Return the network
```

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, backpropagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient

descent. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network. Non-linear activation functions that are commonly used include the logistic function, the softmax function, and the gaussian function.

The backpropagation algorithm for calculating a gradient has been rediscovered a number of times, and is a special case of a more general technique called automatic differentiation in the reverse accumulation mode.

It is also closely related to the Gauss–Newton algorithm, and is also part of continuing research in neural backpropagation.

Multithreaded Backpropagation

Backpropagation is an iterative process that can often take a great deal of time to complete. When multicore computers are used multithreaded techniques can greatly decrease the amount of time that backpropagation takes to converge. If batching is being used, it is relatively simple to adapt the backpropagation algorithm to operate in a multithreaded manner.

The training data is broken up into equally large batches for each of the threads. Each thread executes the forward and backward propagations. The weight and threshold deltas are summed for each of the threads. At the end of each iteration all threads must pause briefly for the weight and threshold deltas to be summed and applied to the neural network. This process continues for each iteration. This multithreaded approach to backpropagation is used by the Encog Neural Network Framework.

Limitations

- The convergence obtained from backpropagation learning is very slow.
- The convergence in backpropagation learning is not guaranteed.
- The result may generally converge to any local minimum on the error surface, since stochastic gradient descent exists on a non-linear surface.
- The backpropagation learning is associated with the problem of scaling.

Case-based reasoning

Case-based reasoning (CBR), broadly construed, is the process of solving new problems based on the solutions of similar past problems. An auto mechanic who fixes an engine

by recalling another car that exhibited similar symptoms is using case-based reasoning. A lawyer who advocates a particular outcome in a trial based on legal precedents or a judge who creates case law is using case-based reasoning. So, too, an engineer copying working elements of nature (practicing biomimicry), is treating nature as a database of solutions to problems. Case-based reasoning is a prominent kind of analogy making.

It has been argued that case-based reasoning is not only a powerful method for computer reasoning, but also a pervasive behavior in everyday human problem solving; or, more radically, that all reasoning is based on past cases personally experienced. This view is related to prototype theory, which is most deeply explored in cognitive science.

Process

Case-based reasoning has been formalized for purposes of computer reasoning as a four-step process :

1. **Retrieve:** Given a target problem, retrieve cases from memory that are relevant to solving it. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived. For example, suppose Fred wants to prepare blueberry pancakes. Being a novice cook, the most relevant experience he can recall is one in which he successfully made plain pancakes. The procedure he followed for making the plain pancakes, together with justifications for decisions made along the way, constitutes Fred's retrieved case.
2. **Reuse:** Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation. In the pancake example, Fred must adapt his retrieved solution to include the addition of blueberries.
3. **Revise:** Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise. Suppose Fred adapted his pancake solution by adding blueberries to the batter. After mixing, he discovers that the batter has turned blue – an undesired effect. This suggests the following revision: delay the addition of blueberries until after the batter has been ladled into the pan.
4. **Retain:** After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory. Fred, accordingly, records his newfound procedure for making blueberry pancakes, thereby enriching his set of stored experiences, and better preparing him for future pancake-making demands.

Comparison to other methods

At first glance, CBR may seem similar to the rule induction algorithms of machine learning. Like a rule-induction algorithm, CBR starts with a set of cases or training examples; it forms generalizations of these examples, albeit implicit ones, by identifying commonalities between a retrieved case and the target problem.

If for instance a procedure for plain pancakes is mapped to blueberry pancakes, a decision is made to use the same basic batter and frying method, thus implicitly generalizing the set of situations under which the batter and frying method can be used. The key difference, however, between the implicit generalization in CBR and the generalization in rule induction lies in when the generalization is made. A rule-induction algorithm draws its generalizations from a set of training examples before the target problem is even known; that is, it performs eager generalization.

For instance, if a rule-induction algorithm were given recipes for plain pancakes, Dutch apple pancakes, and banana pancakes as its training examples, it would have to derive, at training time, a set of general rules for making all types of pancakes. It would not be until testing time that it would be given, say, the task of cooking blueberry pancakes. The difficulty for the rule-induction algorithm is in anticipating the different directions in which it should attempt to generalize its training examples. This is in contrast to CBR, which delays (implicit) generalization of its cases until testing time – a strategy of lazy generalization. In the pancake example, CBR has already been given the target problem of cooking blueberry pancakes; thus it can generalize its cases exactly as needed to cover this situation. CBR therefore tends to be a good approach for rich, complex domains in which there are myriad ways to generalize a case.

Criticism

Critics of CBR argue that it is an approach that accepts anecdotal evidence as its main operating principle. Without statistically relevant data for backing and implicit generalization, there is no guarantee that the generalization is correct. However, all inductive reasoning where data is too scarce for statistical relevance is inherently based on anecdotal evidence.

History

CBR traces its roots to the work of Roger Schank and his students at Yale University in the early 1980s. Schank's model of dynamic memory was the basis for the earliest CBR systems: Janet Kolodner's CYRUS and Michael Lebowitz's IPP .

Other schools of CBR and closely allied fields emerged in the 1980s, investigating such topics as CBR in legal reasoning, memory-based reasoning (a way of reasoning from examples on massively parallel machines), and combinations of CBR with other reasoning methods. In the 1990s, interest in CBR grew in the international community, as evidenced by the establishment of an International Conference on Case-Based Reasoning in 1995, as well as European, German, British, Italian, and other CBR workshops.

CBR technology has produced a number of successful deployed systems, the earliest being Lockheed's CLAVIER, a system for laying out composite parts to be baked in an industrial convection oven. CBR has been used extensively in help desk applications such as the Compaq SMART system.

Prominent CBR systems

- SMART: Support management automated reasoning technology for Compaq customer service
- Appliance Call Center automation at General Electric
- CLAVIER: Applying case-based reasoning on to composite part fabrication
- FormTool: Plastics Color Matching
- CoolAir: HVAC specification and pricing system
- Vidur - A CBR based intelligent advisory system, by C-DAC Mumbai, for farmers of North-East India.
- jCOLIBRI - A CBR framework that can be used to build your own CBR system.

Decision tree learning

Decision tree learning, used in data mining and machine learning, uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. More descriptive names for such tree models are **classification trees** or **regression trees**. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making. This page deals with decision trees in data mining.

General

Decision tree learning is a common method used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the

same value of the target variable, or when splitting no longer adds value to the predictions.

In data mining, trees can be described also as the combination of mathematical and computational techniques to aid the description, categorisation and generalisation of a given set of data.

Data comes in records of the form:

$$(\mathbf{x}, Y) = (x_1, x_2, x_3, ..., x_k, Y)$$

The dependent variable, Y, is the target variable that we are trying to understand, classify or generalise. The vector **x** is composed of the input variables, x_1, x_2, x_3 etc., that are used for that task.

Types

In data mining, trees have additional categories:

- **Classification tree** analysis is when the predicted outcome is the class to which the data belongs.
- **Regression tree** analysis is when the predicted outcome can be considered a real number (e.g. the price of a house, or a patient's length of stay in a hospital).
- **Classification And Regression Tree (CART)** analysis is used to refer to both of the above procedures, first introduced by Breiman et al.
- **CHi-squared Automatic Interaction Detector (CHAID)**. Performs multi-level splits when computing classification trees.
- A **Random Forest** classifier uses a number of decision trees, in order to improve the classification rate.
- **Boosted Trees** can be used for regression-type and classification-type problems.

Formulae

The algorithms that are used for constructing decision trees usually work by choosing a variable at each step that is the next best variable to use in splitting the set of items. "Best" is defined by how well the variable splits the set into subsets that have the same value of the target variable. Different algorithms use different formulae for measuring "best". This section presents a few of the most common formulae. These formulae are applied to each candidate subset, and the resulting values are combined (e.g., averaged) to provide a measure of the quality of the split.

Gini impurity

Used by the CART algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it were randomly labelled

according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability of each item being chosen times the probability of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

To compute Gini impurity for a set of items, suppose y takes on values in {1, 2, ..., m}, and let f_i = the fraction of items labelled with value i in the set.

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = \sum_{i=1}^{m} (f_i - {f_i}^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} {f_i}^2 = 1 - \sum_{i=1}^{m} {f_i}^2$$

Information gain

Used by the ID3, C4.5 and C5.0 tree generation algorithms. Information gain is based on the concept of entropy used in information theory.

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i$$

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 normalisation, 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 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 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.** Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.
- **Perform well with large data in a short time.** Large amounts of data can be analysed using personal computers in a time short enough to enable stakeholders to take decisions based on its analysis.

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. Recent developments suggest the use of genetic algorithms to avoid local optimal decisions and search the decision tree space with little *a priori* bias .
- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning 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, parity or 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) or using learning algorithms based on more expressive representations (such as statistical relational learning or inductive logic programming).

Extending decision trees with decision graphs

In a decision tree, all paths from the root node to the leaf node proceed by way of conjunction, or *AND*. In a decision graph, it is possible to use disjunctions (ORs) to join two more paths together using Minimum Message Length (MML). Decision graphs have been further extended to allow for previously unstated new attributes to be learnt dynamically and used at different places within the graph. The more general coding scheme results in better predictive accuracy and log-loss probabilistic scoring. In general, decision graphs infer models with fewer leaves than decision trees.

Group method of data handling

Group method of data handling (GMDH) is a family of inductive algorithms for computer-based mathematical modeling of multi-parametric datasets that features fully-automatic structural and parametric optimization of models.

GMDH is used in such fields as data mining, knowledge discovery, prediction, complex systems modeling, optimization and pattern recognition.

GMDH algorithms are characterized by inductive procedure that performs sorting-out of gradually complicated polynomial models and selecting the best solution by means of the so-called *external criterion*.

A GMDH model with multiple inputs and one output is a subset of components of the *base function* (1):

$$Y(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{m} a_i f_i$$

where *f* are elementary functions dependent on different sets of inputs, *a* are coefficients and *m* is the number of the base function components.

In order to find the best solution GMDH algorithm consider various component subsets of the base function (1) called *partial models*. Coefficients of these models estimated by the least squares method. GMDH algorithm gradually increase the number of partial model components and find a model structure with optimal complexity indicated by the minimum value of an *external criterion*. This process is called self-organization of models.

The most popular base function used in GMDH is the gradually complicated Kolmogorov-Gabor polynomial (2):

$$Y(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} a_{ijk} x_i x_j x_k + \cdots$$

GMDH is also known as polynomial neural networks and statistical learning networks thanks to implementation of the corresponding algorithms in several commercial software products.

History

GMDH author - Ukrainian scientist Prof. Alexey G. Ivakhnenko.

The method was originated in 1968 by Prof. Alexey G. Ivakhnenko in the Institute of Cybernetics in Kiev (Ukraine). This approach from the very beginning was a computer-based method so, a set of computer programs and algorithms were the primary practical

results achieved at the base of the new theoretical principles. Thanks to the author's policy of open code sharing the method was quickly settled in the large number of scientific laboratories world wide. At that time code sharing was quite a physical action since the Internet is at least 5 years younger than GMDH. Despite this fact the first investigation of GMDH outside the Soviet Union had been made soon by R.Shankar in 1972. Later on different GMDH variants were published by Japanese and Polish scientists.

Period 1968-1971 is characterized by application of only regularity criterion for solving of the problems of identification, pattern recognition and short-term forecasting. As reference functions polynomials, logical nets, fuzzy Zadeh sets and Bayes probability formulas were used. Authors were stimulated by very high accuracy of forecasting with the new approach. Noiseimmunity was not investigated.

Period 1972-1975. The problem of modeling of noised data and incomplete information basis was solved. Multicriteria selection and utilization of additional priory information for noiseimmunity increasing were proposed. Best experiments showed that with extended definition of the optimal model by additional criterion noise level can be ten times more than signal. Then it was improved using Shannon's Theorem of General Communication theory.

Period 1976-1979. The convergence of multilayered GMDH algorithms was investigated. It was shown that some multilayered algorithms have "multilayerness error" - analogous to static error of control systems. In 1977 a solution of objective systems analysis problems by multilayered GMDH algorithms was proposed. It turned out that sorting-out by criteria ensemble finds the only optimal system of equations and therefore to show complex object elements, their main input and output variables.

Period 1980-1988. Many important theoretical results were received. It became clear that full physical models cannot be used for long-term forecasting. It was proved, that non-physical models of GMDH are more accurate for approximation and forecast than physical models of regression analysis. Two-level algorithms which use two different time scales for modeling were developed.

Since 1989 the new algorithms (AC, OCC, PF) for non-parametric modeling of fuzzy objects and SLP for expert systems were developed and investigated. Present stage of GMDH development can be described as blossom out of twice-multilayered neuronets and parallel combinatorial algorithms for multiprocessor computers.

External criteria

External criterion is one of the key features of GMDH. Criterion describes requirements to the model, for example minimization of Least squares. It is always calculated with a separate part of data sample that have not been used for estimation of coefficients. There are several popular criteria:

- Criterion of Regularity (CR) - Least squares of a model at the sample B.
- Criterion of Unbiasdness - Sum of CR value and special CR for which A is B and B is A. Ratio of sample lengthes must be 1:1 i.e. size of A must be the same as size of B.

If a criterion does not define the number of observations for external dataset then the problem of data dividing ratio appears because the forecasting abilities of identified model are very dependent on the dividing ratio.

GMDH-type neural networks

There are many different ways to choose an order for partial models consideration. The very first consideration order used in GMDH and originally called multilayered inductive procedure is the most popular one. It is a sorting-out of gradually complicated models generated from Kolmogorov-Gabor polinomial. The best model is indicated by the minimum of the external criterion characteristic. Multilayered procedure is equivalent to the Artificial Neural Network with polynomial activation function of neurons. Therefore the algorithm with such an approach usually referred as GMDH-type Neural Network or Polynomial Neural Network.

Combinatorial GMDH

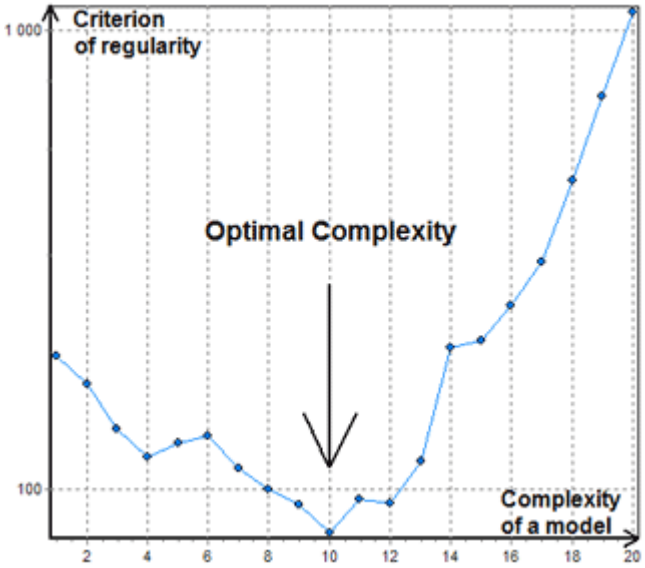


Fig.1. A typical distribution of minimal values of criterion of regularity for Combinatorial GMDH models with different complexity.

Another important approach to partial models consideration that becomes more and more popular is a brute force combinatorial search that is either limited or full. This approach has some advantages against Polynomial Neural Networks but requires considerable computational power and thus is not effective for objects with more than 30 inputs in

case of full search. An important achievement of Combinatorial GMDH is that it fully outperforms linear regression approach if noise level in the input data is greater than zero.

Basic combinatorial algorithm makes the following steps:

- Divides data sample onto parts A and B.
- Generates structures for partial models.
- Estimates coefficients of partial models using Least squares method and sample A.
- Calculates value of external criterion for partial models using sample B.
- Chooses the best model (set of models) indicated by minimal value of the criterion.

In contrast to GMDH-type neural networks Combinatorial algorithm can't be stopped at the certain level of complexity because a point of increase of criterion value can be simply a local minimum, see Fig.1.

Algorithms

- Combinatorial (COMBI)
- Multilayered Iterative (MIA)
- GN
- Objective System Analysis (OSA)
- Harmonical
- Two-level (ARIMAD)
- Multiplicative-Additive (MAA)
- Objective Computer Clusterization (OCC);
- Pointing Finger (PF) clusterization algorithm;
- Analogues Complexing (AC)
- Harmonical Rediscretization
- Algorithm on the base of Multilayered Theory of Statistical Decisions (MTSD)
- Group of Adaptive Models Evolution (GAME)

Learning Automata

A branch of the theory of Adaptive control is devoted to **learning automata** surveyed by Narendra and Thathachar which were originally described explicitly as finite state automata. Learning automata select their current action based on past experiences from the environment.

History

Research in Learning Automata can be traced back to the work of Tsetlin in the early 1960s in the Soviet Union. Together with some colleagues, he published a collection of

papers on how to use matrices to describe automata functions. Additionally, Tsetlin worked on *reasonable* and *collective automata behaviour*, and on *automata games*. Learning automata were also investigated by researches in the United States in the 1960s. However, the term *learning automaton* was not used until Narendra and Thathachar introduced it in a survey paper in 1974.

Definition

A learning automaton is an adaptive decision-making unit situated in a random environment that learns the optimal action through repeated interactions with its environment. The actions are chosen according to a specific probability distribution which is updated based on the environment response the automaton obtains by performing a particular action.

Finite action-set learning automata

Finite action-set learning automata (FALA) are a class of learning automata for which the number of possible actions is finite or, in more mathematical terms, for which the size of the action-set is finite.

Minimum message length

Minimum message length (MML) is a formal information theory restatement of Occam's Razor: even when models are not equal in goodness of fit accuracy to the observed data, the one generating the shortest overall message is more likely to be correct (where the message consists of a statement of the model, followed by a statement of data encoded concisely using that model). MML was invented by Chris Wallace, first appearing in the seminal (Wallace and Boulton, 1968).

MML is intended not just as a theoretical construct, but as a technique that may be deployed in practice. It differs from the related concept of Kolmogorov complexity in that it does not require use of a Turing-complete language to model data. The relation between Strict MML (SMML) and Kolmogorov complexity is outlined in Wallace and Dowe (1999a). Further, a variety of mathematical approximations to "Strict" MML can be used.

Definition

Shannon's *A Mathematical Theory of Communication* (1949) states that in an optimal code, the message length (in binary) of an event *E*, $\text{length}(E)$, where *E* has probability *P*(*E*), is given by $\text{length}(E) = -\log_2(P(E))$.

Bayes's theorem states that the probability of a hypothesis (*H*) given evidence (*E*) is proportional to *P*(*E* | *H*)*P*(*H*), which is just $P(H \wedge E)$. We want the model (hypothesis) with the highest such probability. Therefore, we want the model which generates the shortest (two-part) encoding of the data. Since $\text{length}(H \wedge E) = -\log_2(P(H \wedge E))$, the most probable model will have the shortest such message. The message breaks into two parts: $-\log_2(P(H \wedge E)) = -\log_2(P(H)) + -\log_2(P(E|H))$. The first is the length of the model, and the second is the length of the data, given the model.

MML naturally and precisely trades model complexity for goodness of fit. A more complicated model takes longer to state (longer first part) but probably fits the data better (shorter second part). So, an MML metric won't choose a complicated model unless that model pays for itself.

Continuous-valued parameters

One reason why a model might be longer would be simply because its various parameters are stated to greater precision, thus requiring transmission of more digits. Much of the power of MML derives from its handling of how accurately to state parameters in a model, and a variety of approximations that make this feasible in practice. This allows it to usefully compare, say, a model with many parameters imprecisely stated against a model with fewer parameters more accurately stated.

Key features of MML

- MML can be used to compare models of different structure. For example, its earliest application was in finding mixture models with the optimal number of classes. Adding extra classes to a mixture model will always allow the data to be fitted to greater accuracy, but according to MML this must be weighed against the extra bits required to encode the parameters defining those classes.
- MML is a method of Bayesian model comparison. It gives every model a score.
- MML is scale-invariant and statistically invariant. Unlike many Bayesian selection methods, MML doesn't care if you change from measuring length to volume or from Cartesian co-ordinates to polar co-ordinates.
- MML is statistically consistent. For problems like the Neyman-Scott (1948) problem or factor analysis where the amount of data per parameter is bounded above, MML can estimate all parameters with statistical consistency.
- MML accounts for the precision of measurement. It uses the Fisher information (in the Wallace-Freeman 1987 approximation, or other hyper-volumes in other approximations) to optimally discretize continuous parameters. Therefore the posterior is always a probability, not a probability density.
- MML has been in use since 1968. MML coding schemes have been developed for several distributions, and many kinds of machine learners including unsupervised classification, decision trees and graphs, DNA sequences, Bayesian networks,

neural networks (one-layer only so far), image compression, image and function segmentation, etc.

Lazy learning

In artificial intelligence, **lazy learning** is a learning method in which generalization beyond the training data is delayed until a query is made to the system, as opposed to in eager learning, where the system tries to generalize the training data before receiving queries.

The main advantage gained in employing a lazy learning method, such as Case based reasoning, is that the target function will be approximated locally, such as in the k-nearest neighbor algorithm. Because the target function is approximated locally for each query to the system, lazy learning systems can simultaneously solve multiple problems and deal successfully with changes in the problem domain.

The disadvantages with lazy learning include the large space requirement to store the entire training dataset. Particularly noisy training data increases the case base unnecessarily, because no abstraction is made during the training phase. Another disadvantage is that lazy learning methods are usually slower to evaluate, though this is coupled with a faster training phase.

Instance-based learning

In machine learning, **instance-based learning** or **memory-based learning** is a family of learning algorithms that, instead of performing explicit generalization, compare new problem instances with instances seen in training, which have been stored in memory. Instance-based learning is a kind of lazy learning.

It is called instance-based because it constructs hypotheses directly from the training instances themselves. This means that the hypothesis complexity can grow with the data: in the worst case, a hypothesis is a list of n training items and classification takes $O(n)$. One advantage that instance-based learning has over other methods of machine learning is its ability to adapt its model to previously unseen data. Where other methods generally require the entire set of training data to be re-examined when one instance is changed, instance-based learners may simply store a new instance or throw an old instance away.

A simple example of an instance-based learning algorithm is the k-nearest neighbor algorithm. Daelemans and Van den Bosch describe variations of this algorithm for use in natural language processing (NLP), claiming that memory-based learning is both more psychologically realistic than other machine-learning schemes and practically effective.

k-nearest neighbor algorithm

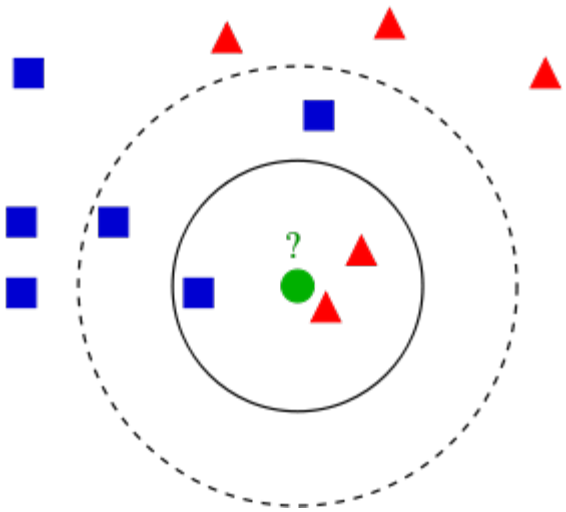
In pattern recognition, the ***k*-nearest neighbors algorithm** (*k*-NN) is a method for classifying objects based on closest training examples in the feature space. *k*-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The *k*-nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its *k* nearest neighbors (*k* is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of its nearest neighbor.

The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its *k* nearest neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. (A common weighting scheme is to give each neighbor a weight of $1/d$, where *d* is the distance to the neighbor. This scheme is a generalization of linear interpolation.)

The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. The *k*-nearest neighbor algorithm is sensitive to the local structure of the data.

Nearest neighbor rules in effect compute the decision boundary in an implicit manner. It is also possible to compute the decision boundary itself explicitly, and to do so in an efficient manner so that the computational complexity is a function of the boundary complexity.

Algorithm



Example of *k*-NN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If *k = 3* it is classified to the second class because there are 2 triangles and only 1 square inside the

inner circle. If $k = 5$ it is classified to first class (3 squares vs. 2 triangles inside the outer circle).

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, *k* is a user-defined constant, and an unlabelled vector (a query or test point) is classified by assigning the label which is most frequent among the *k* training samples nearest to that query point.

Usually Euclidean distance is used as the distance metric; however this is only applicable to continuous variables. In cases such as text classification, another metric such as the **overlap metric** (or Hamming distance) can be used. Often, the classification accuracy of "k"-NN can be improved significantly if the distance metric is learned with specialized algorithms such as e.g. Large Margin Nearest Neighbor or Neighbourhood components analysis.

A drawback to the basic "majority voting" classification is that the classes with the more frequent examples tend to dominate the prediction of the new vector, as they tend to come up in the *k* nearest neighbors when the neighbors are computed due to their large number. One way to overcome this problem is to weight the classification taking into account the distance from the test point to each of its *k* nearest neighbors.

KNN is a special case of a variable-bandwidth, kernel density "balloon" estimator with a uniform kernel.

Parameter selection

The best choice of *k* depends upon the data; generally, larger values of *k* reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good *k* can be selected by various heuristic techniques, for example, cross-validation. The special case where the class is predicted to be the class of the closest training sample (i.e. when $k = 1$) is called the nearest neighbor algorithm.

The accuracy of the *k*-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features by the mutual information of the training data with the training classes.

In binary (two class) classification problems, it is helpful to choose *k* to be an odd number as this avoids tied votes. One popular way of choosing the empirically optimal *k* in this setting is via bootstrap method.

Properties

The naive version of the algorithm is easy to implement by computing the distances from the test sample to all stored vectors, but it is computationally intensive, especially when the size of the training set grows. Many nearest neighbor search algorithms have been proposed over the years; these generally seek to reduce the number of distance evaluations actually performed. Using an appropriate nearest neighbor search algorithm makes *k*-NN computationally tractable even for large data sets.

The nearest neighbor algorithm has some strong consistency results. As the amount of data approaches infinity, the algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data). *k*-nearest neighbor is guaranteed to approach the Bayes error rate, for some value of *k* (where *k* increases as a function of the number of data points). Various improvements to *k*-nearest neighbor methods are possible by using proximity graphs.

For estimating continuous variables

The *k*-NN algorithm can also be adapted for use in estimating continuous variables. One such implementation uses an inverse distance weighted average of the *k*-nearest multivariate neighbors. This algorithm functions as follows:

1. Compute Euclidean or Mahalanobis distance from target plot to those that were sampled.
2. Order samples taking for account calculated distances.
3. Choose heuristically optimal *k* nearest neighbor based on RMSE done by cross validation technique.
4. Calculate an inverse distance weighted average with the *k*-nearest multivariate neighbors.

The optimal k for most datasets is 10 or more. That produces much better results than 1-NN. Using a weighted k-NN, where the weights by which each of the k nearest points' class (or value in regression problems) is multiplied are proportional to the inverse of the distance between that point and the point for which the class is to be predicted also significantly improves the results.

Ripple down rules

Ripple Down Rules is a way of approaching knowledge acquisition. Knowledge acquisition refers to the transfer knowledge from human experts to knowledge based systems.

Introductory material

Ripple Down Rules (RDR) is an incremental approach to knowledge acquisition and covers a family of techniques. RDR were proposed by Compton and Jansen based on experience maintaining the expert system GARVAN-ES1 (Compton and Jansen 1988). The original GARVAN-ES1 (Horn et al. 1985) employed a knowledge acquisition process where new cases, that were poorly classified by the system, were added to a data base and then used to incrementally refine the knowledge base. The added cases, whose conclusions conflicted with the advice of the system were termed "cornerstone cases". Consequently the data base grew iteratively with each refinement to the knowledge. The data base could then be used to test changes to the knowledge. Knowledge acquisition tools, similar to those provided by Teiresias were developed to find and help modify the conflicting rules. The tools would display the rules fired by each case and suggestions to "edit" the knowledge to remove the conflicts.

In the RDR framework, the human expert’s knowledge is acquired based on the current context and is added incrementally. Compton and Jansen argued that the expert's knowledge is to some extent 'made up' to justify why she was right, not to explain how she reached this right interpretation (or conclusion). The justification is based on features that are identified from the current case. The expert creates a rule for classifying cases corresponding to a particular context. This rule is unlikely to classify all cases belonging to the class. Compton and Jansen asserted that it is not possible to create a single elegant context free rule as the knowledge we communicate is a justification in a context. This implies that there is no absolute knowledge that acts as foundation of other knowledge, since knowledge is only true in a context (Compton and Jansen 1990).

Methodology

Ripple Down Rules (RDR) consist of a data structure and knowledge acquisition scenarios. Human experts' knowledge is stored in the data structure. The knowledge is coded as a set of rules. The process of transferring human experts's knowledge to Knowledge Based Systems in RDR is explained in Knowledge Acquisition Scenario.

Data Structure

There are various structures of Ripple Down Rules, for example Single Classification Ripple Down Rules (SCRDR), Multiple Classification Ripple Down Rules (MCRDR), Nested Ripple Down Rules (NRDR) and Repeat Inference Multiple Classification Ripple Down Rules (RIMCRDR). The data structure of RDR described here is SCRDR, which is the simplest structure.

The data structure is similar to a decision tree. Each node has a rule, the format of this rule is IF cond1 AND cond2 AND ... AND condN THEN conclusion. Cond1 is a condition (boolean evaluation), for example A=1, isGreater(A,5) and average(A,">",average(B)). Each node has exactly two successor nodes, these successor nodes are connected to predecessor node by "ELSE" or "EXCEPT".

An example of SCRDR tree (defined recursively) is shown below:

IF (OutLook = "SUNNY" AND Temperature = "COOL") THEN PLAY="TENNIS" EXCEPT Child-1 ELSE Child-2

where Child-1 and Child-2 are also SCRDR trees. For example Child-1 is:

IF (Wind = "WINDY" AND Humidity = "HIGH") THEN Play="SQUASH" EXCEPT NoChild ELSE NoChild

Knowledge Acquisition Scenario

Human experts provide a case to the systems and they add a new rule to correct the classification of a misclassified case. For example rule Child-1 is added to correct classification of case [OutLook="SUNNY", Temperature="COOL", Wind="WINDY", Humidity="HIGH", ForeCast="STORM", Play="SQUASH"]. This case is misclassified as Play="TENNIS".

When a rule is constructed by the human experts, the conditions of this rule should be satisfied by the misclassified case and also they should NOT be satisfied by any previous cases classified correctly by the parent rule (in this context is the first rule).

Random forest

Random forest (or **random forests**) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman and Adele Cutler, and "Random Forests" is their trademark. The term came from **random decision forests** that was first proposed by Tin Kam Ho of Bell Labs in 1995. The method combines Breiman's "bagging" idea and the random selection of features, introduced independently by Ho and Amit and Geman in order to construct a collection of decision trees with controlled variation.

The selection of a random subset of features is an example of the random subspace method, which, in Ho's formulation, is a way to implement stochastic discrimination proposed by Eugene Kleinberg.

Learning algorithm

Each tree is constructed using the following algorithm:

1. Let the number of training cases be *N*, and the number of variables in the classifier be *M*.
2. We are told the number *m* of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*.

3. Choose a training set for this tree by choosing *N* times with replacement from all *N* available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
4. For each node of the tree, randomly choose *m* variables on which to base the decision at that node. Calculate the best split based on these *m* variables in the training set.
5. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

Advantages

The advantages of random forest are:

- For many data sets, it produces a highly accurate classifier
- It handles a very large number of input variables
- It estimates the importance of variables in determining classification
- It generates an internal unbiased estimate of the generalization error as the forest building progresses
- It includes a good method for estimating missing data and maintains accuracy when a large proportion of the data are missing
- It provides an experimental way to detect variable interactions
- It can balance error in class population unbalanced data sets
- It computes proximities between cases, useful for clustering, detecting outliers, and (by scaling) visualizing the data
- Using the above, it can be extended to unlabeled data, leading to unsupervised clustering, outlier detection and data views
- Learning is fast

Disadvantages

- Random forests are prone to overfitting for some datasets. This is even more pronounced in noisy classification/regression tasks.
- Random forests do not handle large numbers of irrelevant features as well as ensembles of entropy-reducing decision trees.
- It is more efficient to select a random decision boundary than an entropy-reducing decision boundary, thus making larger ensembles more feasible. Although this may seem to be an advantage at first, it has the effect of shifting the computation from training time to evaluation time, which is actually a disadvantage for most applications.

Ensemble learning

In statistics and machine learning, **ensemble methods** use multiple models to obtain better predictive performance than could be obtained from any of the constituent models.

Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble refers only to a concrete finite set of alternative models.

Overview

Supervised learning algorithms are commonly described as performing the task of searching through a hypothesis space to find a suitable hypothesis that will make good predictions with a particular problem. Even if the hypothesis space contains hypotheses that are very well-suited for a particular problem, it may be very difficult to find a good one. Ensembles combine multiple hypotheses to form a (hopefully) better hypothesis. In other words, an ensemble is a technique for combining many *weak learners* in an attempt to produce a *strong learner*.

Evaluating the prediction of an ensemble typically requires more computation than evaluating the prediction of a single model, so ensembles may be thought of as a way to compensate for poor learning algorithms by performing a lot of extra computation. Fast algorithms, such as Decision trees are commonly used with ensembles, although slower algorithms can benefit from ensemble techniques as well.

Ensemble Theory

An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. The trained ensemble, therefore, represents a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, some ensemble techniques (especially bagging) tend to reduce problems related to over-fitting of the training data.

Empirically, ensembles tend to yield better results when there is a significant diversity among the models . Many ensemble methods, therefore, seek to promote diversity among the models they combine . Although perhaps non-intuitive, more random algorithms (like random decision trees) can be used to produce a stronger ensemble than very deliberate algorithms (like entropy-reducing decision trees). Using a variety of strong learning algorithms, however, has been shown to be more effective than using techniques that attempt to *dumb-down* the models in order to promote diversity.

Common Types of Ensembles

Bayes Optimal Classifier

The Bayes Optimal Classifier is an optimal classification technique. It is an ensemble of all the hypotheses in the hypothesis space. On average, no other ensemble can outperform it, so it is the ideal ensemble. Each hypothesis is given a vote proportional to the likelihood that the training dataset would be sampled from a system if that hypothesis were true. To facilitate training data of finite size, the vote of each hypothesis is also multiplied by the prior probability of that hypothesis. The Bayes Optimal Classifier can be expressed with following equation:

$$y = \operatorname{argmax}_{c_j \in C} \sum_{h_i \in H} P(c_j|h_i)P(T|h_i)P(h_i)$$

where y is the predicted class, C is the set of all possible classes, H is the hypothesis space, P refers to a *probability*, and T is the training data. As an ensemble, the Bayes Optimal Classifier represents a hypothesis that is not necessarily in H. The hypothesis represented by the Bayes Optimal Classifier, however, is the optimal hypothesis in *ensemble space* (the space of all possible ensembles consisting only of hypotheses in H).

Unfortunately, Bayes Optimal Classifier cannot be practically implemented for any but the most simple of problems. There are several reasons why the Bayes Optimal Classifier cannot be practically implemented:

1. Most interesting hypothesis spaces are too large to iterate over, as required by the *argmax*.
2. Many hypotheses yield only a predicted class, rather than a probability for each class as required by the term $P(c_j \mid h_i)$.
3. Computing an unbiased estimate of the probability of the training set given a hypothesis ($P(T \mid h_i)$) is non-trivial.
4. Estimating the prior probability for each hypothesis ($P(h_i)$) is rarely feasible.

Bayesian Model Averaging

Bayesian model averaging is an ensemble technique that seeks to approximate the Bayes Optimal Classifier by sampling hypotheses from the hypothesis space, and combining them using Bayes' law. Unlike the Bayes optimal classifier, Bayesian model averaging can be practically implemented. Hypotheses are typically sampled using a Monte Carlo sampling technique such as MCMC. For example, Gibbs sampling may be used to draw hypotheses that are representative of the distribution $P(T \mid H)$. It has been shown that under certain circumstances, when hypotheses are drawn in this manner and averaged according to Bayes' law, this technique has an expected error that is bounded to be at most twice the expected error of the Bayes optimal classifier. Despite the theoretical correctness of this technique, however, it has a tendency to promote over-fitting, and does not perform as well empirically as simpler ensemble techniques such as bagging.

Bootstrap Aggregating (Bagging)

Bootstrap aggregating, often abbreviated as *bagging*, involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly-drawn subset of the training set. As an example, the random forest algorithm combines random decision trees with bagging to achieve very high classification accuracy.

Boosting

Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified. In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to over-fit the training data. By far, the most common implementation of Boosting is Adaboost, although some newer algorithms are reported to achieve better results.

Bucket Of Models

A "bucket of models" is an ensemble in which a model selection algorithm is used to choose the best model for each problem. When tested with only one problem, a bucket of models can produce no better results than the best model in the set, but when evaluated across many problems, it will typically produce much better results, on average, than any model in the set.

The most common approach used for model-selection is cross-validation selection. It is described with the following pseudo-code:

```
For each model m in the bucket:
  Do c times: (where 'c' is some constant)
    Randomly divide the training dataset into two datasets: A, and B.
    Train m with A
    Test m with B
Select the model that obtains the highest average score
```

Cross-Validation Selection can be summed up as: "try them all with the training set, and pick the one that works best".

Gating is a generalization of Cross-Validation Selection. It involves training another learning model to decide which of the models in the bucket is best-suited to solve the problem. Often, a perceptron is used for the gating model. It can be used to pick the "best" model, or it can be used to give a linear weight to the predictions from each model in the bucket.

When a bucket of models is used with a large set of problems, it may be desirable to avoid training some of the models that take a long time to train. Landmark learning is a meta-learning approach that seeks to solve this problem. It involves training only the fast (but imprecise) algorithms in the bucket, and then using the performance of these

algorithms to help determine which slow (but accurate) algorithm is most likely to do best.

Stacking

The crucial prior belief underlying the scientific method is that one can judge among a set of models by comparing them on data that was not used to create any of them. This same prior belief underlies the use in machine learning of *bake-off contests* to judge which of a set of competitor learning algorithms is actually best.

This prior belief can also be used by a single practitioner, to choose among a set of models based on a single data set. This is done by partitioning the data set into a *held-in* data set and a *held-out* data set; training the models on the held-in data; and then choosing whichever of those trained models performs best on the held-out data. This is the cross-validation technique, mentioned above.

Stacking (sometimes called *stacked generalization*) exploits this prior belief further. It does this by using performance on the held-out data to combine the models rather than choose among them, thereby typically getting performance better than any single one of the trained models.. It has been successfully used on both supervised learning tasks (regression) and unsupervised learning (density estimation). It has also been used to estimate Bagging's error rate.

Because the prior belief concerning held-out data is so powerful, stacking often out-performs Bayesian model-averaging. Indeed, renamed *blending*, stacking was extensively used in the two top performers in the recent Netflix competition.

Activity recognition

Activity recognition aims to recognize the actions and goals of one or more agents from a series of observations on the agents' actions and the environmental conditions. Since the 1980s, this research field has captured the attention of several computer science communities due to its strength in providing personalized support for many different applications and its connection to many different fields of study such as medicine, human-computer interaction, or sociology.

To understand activity recognition better, consider the following scenario. An elderly man wakes up at dawn in his small studio apartment, where he stays alone. He lights the stove to make a pot of tea, switches on the toaster oven, and takes some bread and jelly from the cupboard. After taking his morning medication, a computer-generated voice gently reminds him to turn off the toaster. Later that day, his daughter accesses a secure website where she scans a check-list, which was created by a sensor network in her father's apartment. She finds that her father is eating normally, taking his medicine on schedule, and continuing to manage his daily life on his own. That information puts her mind at ease.

Many different applications have been studied by researchers in activity recognition; examples include assisting the sick and disabled. For example, Pollack et al. show that by automatically monitoring human activities, home-based rehabilitation can be provided for people suffering from traumatic brain injuries. One can find applications ranging from security-related applications and logistics support to location-based services. Due to its many-faceted nature, different fields may refer to activity recognition as plan recognition, goal recognition, intent recognition, behavior recognition, location estimation and location-based services.

Types of activity recognition

Sensor-based, single-user activity recognition

Sensor-based activity recognition integrates the emerging area of sensor networks with novel data mining and machine learning techniques to model a wide range of human activities. Mobile devices (e.g. smart phones) provide sufficient sensor data and calculation power to enable physical activity recognition to provide an estimation of the energy consumption during everyday life. Sensor-based activity recognition researchers believe that by empowering ubiquitous computers and sensors to monitor the behavior of agents (under consent), these computers will be better suited to act on our behalf.

Levels of sensor-based activity recognition

Sensor-based activity recognition is a challenging task due to the inherent noisy nature of the input. Thus, statistical modeling has been the main thrust in this direction in layers, where the recognition at several intermediate levels is conducted and connected. At the lowest level where the sensor data are collected, statistical learning concerns how to find the detailed locations of agents from the received signal data. At an intermediate level, statistical inference may be concerned about how to recognize individuals' activities from the inferred location sequences and environmental conditions at the lower levels. Furthermore, at the highest level a major concern is to find out the overall goal or subgoals of an agent from the activity sequences through a mixture of logical and statistical reasoning. Scientific conferences where activity recognition work from wearable and environmental often appears are ISWC and UbiComp.

Sensor-based, multi-user activity recognition

Recognizing activities for multiple users using on-body sensors first appeared in the work by ORL using active badge systems in the early 90´s. Other sensor technology such as acceleration sensors were used for identifying group activity patterns during office scenarios . Activities of Multiple Users in intelligent environments are addressed in Gu et al. . In this work, they investigate the fundamental problem of recognizing activities for multiple users from sensor readings in a home environment, and propose a novel pattern mining approach to recognize both single-user and multi-user activities in a unified solution. Many interesting research topics can be spawn from this work.

Vision-based activity recognition

It is a very important and challenging problem to track and understand the behavior of agents through videos taken by various cameras. The primary technique employed is computer vision. Vision-based activity recognition has found many applications such as human-computer interaction, user interface design, robot learning, and surveillance, among others. Scientific conferences where vision based activity recognition work often appears are ICCV and CVPR.

In vision-based activity recognition, a great deal of work has been done. Researchers have attempted a number of methods such as optical flow, Kalman filtering, hidden Markov models, etc., under different modalities such as single camera, stereo, and infrared. In addition, researchers have considered multiple aspects on this topic, including single pedestrian tracking, group tracking, and detecting dropped objects.

Levels of vision-based activity recognition

In vision-based activity recognition, the computational process is often divided into four steps, namely human detection, human tracking, human activity recognition and then a high-level activity evaluation.

Approaches of activity recognition

Activity recognition through logic and reasoning

Logic-based approaches keep track of all logically consistent explanations of the observed actions. Thus, all possible and consistent plans or goals must be considered. Kautz provided a formal theory of plan recognition. He described plan recognition as a logical inference process of circumscription. All actions, plans are uniformly referred to as goals, and a recognizer's knowledge is represented by a set of first-order statements called event hierarchy encoded in first-order logic, which defines abstraction, decomposition and functional relationships between types of events.

Kautz's general framework for plan recognition has an exponential time complexity in worst case, measured in the size of input hierarchy. Lesh and Etzioni went one step further and presented methods in scaling up goal recognition to scale up his work computationally. In contrast to Kautz's approach where the plan library is explicitly represented, Lesh and Etzioni's approach enables automatic plan-library construction from domain primitives. Furthermore, they introduced compact representations and efficient algorithms for goal recognition on large plan libraries.

Inconsistent plans and goals are repeatedly pruned when new actions arrive. Besides, they also presented methods for adapting a goal recognizer to handle individual idiosyncratic behavior given a sample of an individual's recent behavior. Pollack et al. described a

direct argumentation model that can know about the relative strength of several kinds of arguments for belief and intention description.

A serious problem of logic-based approaches is their inability or inherent infeasibility to represent uncertainty. They offer no mechanism for preferring one consistent approach to another and incapable of deciding whether one particular plan is more likely than another, as long as both of them can be consistent enough to explain the actions observed. There is also a lack of learning ability associated with logic based methods.

Activity recognition through probabilistic reasoning

Probability theory and statistical learning models are more recently applied in activity recognition to reason about actions, plans and goals.

Plan recognition can be done as a process of reasoning under uncertainty, which is convincingly argued by Charniak and Goldman. They argued that any model that does not incorporate some theory of uncertainty reasoning cannot be adequate. In the literature, there have been several approaches which explicitly represent uncertainty in reasoning about an agent's plans and goals.

Using sensor data as input, Hodges and Pollack designed machine learning-based systems for identifying individuals as they perform routine daily activities such as making coffee. Intel Research (Seattle) Lab and University of Washington at Seattle have done some important works on using sensors to detect human plans. Some of these works infer user transportation modes from readings of radio-frequency identifiers (RFID) and global positioning systems (GPS).

Wi-Fi-based activity recognition

When activity recognition is performed indoors and in cities using the widely available Wi-Fi signals and 802.11 access points, there is much noise and uncertainty. These uncertainties are modeled using a dynamic Bayesian network model by Yin et al. A multiple goal model that can reason about user's interleaving goals is presented by Chai and Yang, where a deterministic state transition model is applied. A better model that models the concurrent and interleaving activities in a probabilistic approach is proposed by Hu and Yang. A user action discovery model is presented by Yin et al., where the Wi-Fi signals are segmented to produce possible actions.

A fundamental problem in Wi-Fi-based activity recognition is to estimate the user locations. Two important issues are how to reduce the human labelling effort and how to cope with the changing signal profiles when the environment changes. Yin et al. dealt with the second issue by transferring the labelled knowledge between time periods. Chai and Yang proposed a hidden Markov model-based method to extend labelled knowledge by leveraging the unlabelled user traces. J. Pan et al. propose to perform location estimation through online co-localization, and S. Pan et al. proposed to apply multi-view learning for migrating the labelled data to a new time period.

Data mining based approach to activity recognition

Different from traditional machine learning approaches, a novel approach based on data mining has been recently proposed by a research group led by Dr. Gu. In this approach, the problem of activity recognition is formulated as a pattern-based classification problem. They proposed a novel data mining approach based on discriminative patterns which describe significant changes between any two activity classes of data to recognize sequential, interleaved and concurrent activities in a unified solution. The advantages of such an approach are: Firstly, they provide a solution which recognizes sequential, interleaved and concurrent activities in a unified computation framework. Secondly, no training is required for interleaved and concurrent activities since these two activity models can be derived directly from the sequential activity model. Thirdly, their solution is more noise tolerant because mining the differences of classes will not include noise patterns provided the noise distribution is random among classes. The noise-tolerant feature is particularly important in sensor-based activity recognition.