Machine Learning

Rajendra Akerkar

Technomathematics Research Foundation
TMRF Report 08- 2008

Machine Learning

Techniques for Analysis of Data

TMRF-report-08-2008

TMRF White Paper

By Rajendra Akerkar

August 2008

Contents

Section 1

Summary

This report presents the fundamental concepts of machine learning (ML) and describes various ML algorithms.

Section 2

Machine Learning

Learning can be defined in general as a process of gaining knowledge through experience. We humans start the process of learning new things from the day we are born. This learning process continues throughout our life where we try to gather more knowledge and try to improve what we have already learned through experience and from information gathered from our surroundings.

Artificial Intelligence (AI) is a field of computer science whose objective is to build a system that exhibits intelligent behavior in the tasks it performs. A system can be said to be intelligent when it has learned to perform a task related to the process it has been assigned to without any human interference and with high accuracy. Machine Learning (ML) is a sub-field of AI whose concern is the development, understanding and evaluation of algorithms and techniques to allow a computer to learn. ML intertwines with other disciplines such as statistics, human psychology and brain modeling. Human psychology and neural models obtained from brain modeling help in understanding the workings of the human brain, and especially its learning process, which can be used in the formulation of ML algorithms. Since many ML algorithms use analysis of data for building models, statistics plays a major role in this field.

A process or task that a computer is assigned to deal with can be termed the knowledge or task domain (or just the domain). The information that is generated by or obtained from the domain constitutes its knowledge base. The knowledge base can be represented in various ways using Boolean, numerical, and discrete values, relational literals and their combinations. The knowledge base is generally represented in the form of input-output pairs, where the information represented by the input is given by the domain and the result generated by the domain is the output. The information from the knowledge base can be used to depict the data generation process (i.e., output classification for a given input) of the domain. Knowledge of the data generation process does not define the internals of the working of the domain, but can be used to classify new inputs accordingly.

As the knowledge base grows in size or gets complex, inferring new relations about the data generation process (the domain) becomes difficult for humans. ML algorithms try to learn from the domain and the knowledge base to build computational models that represent the domain in an accurate and efficient way. The model built captures the data generation process of the domain, and by use of this model the algorithm is able to match previously unobserved examples from the domain.

The models built can take on different forms based on the ML algorithm used. Some of the model forms are decision lists, inference networks, concept hierarchies, state transition networks and search-control rules. The concepts and working of various ML algorithms are different but their common goal is to learn from the domain they represent.

ML algorithms need a dataset, which constitutes the knowledge base, to build a model of the domain. The dataset is a collection of instances from the domain. Each instance consists of a set of attributes which describe the properties of that example from the domain. An attribute takes in a range of values based on its attribute type, which can be discrete or continuous. Discrete (or nominal) attributes take on distinct values (e.g., *car = Honda, weather = sunny*) whereas continuous (or numeric) attributes take on numeric values (e.g., *distance = 10.4 meters, temperature = 20^{o}F*).

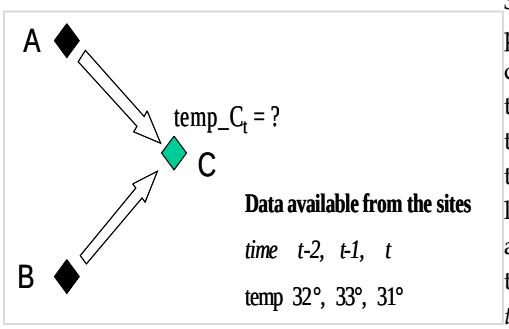
Each instance consists of a set of input attributes and an output attribute. The input attributes are the information given to the learning algorithm and the output attribute contains the feedback of the activity on that information. The value of the output attribute is assumed to depend on the values of the input attributes. The attribute along with the value assigned to it define a feature, which makes an instance a feature vector. The model built by an algorithm can be seen as a function that maps the input attributes in the instance to a value of the output attribute.

Huge amounts of data may look random when observed with the naked eye, but on a closer examination, we may find patterns and relations in it. We also get an insight into the mechanism that generates the data. Witten & Frank [2005] define data mining as a process of discovering patterns in data. It is also referred to as the process of extracting relationships from the given data. In general data mining differs from machine learning in that the issue of the efficiency of learning a model is considered along with the effectiveness of the learning. In data mining problems, we can look at the data generation process as the domain and the data generated by the domain as the knowledge base. Thus, ML algorithms can be used to learn a model that describes the data generation process based on the dataset given to it. The data given to the algorithm for building the model is called the training data, as the computer is being trained to learn from this data, and the model built is the result of the learning process. This model can now be used to predict or classify previously unseen examples. New examples used to evaluate the model are called a test set. The accuracy of a model can be estimated from the difference between the predicted and actual value of the target attribute in the test set.

Predicting weather conditions can also be considered as an example of data mining. Using the weather data collected from a location for a certain period of time, we obtain a model to predict variables such as temperature at a given time based on the input to the model. As weather conditions tend to follow patterns and are not totally random, we can use current meteorological readings along with those taken a few hours earlier at a location and also readings taken from

nearby locations to predict a condition such as the temperature at that location. Thus, the data instances that will be used to build the model may contain present and previous hour's readings from a set of nearby locations as input attributes. The variable that is to be predicted at one of these locations for the present hour is the target attribute. The type and number of conditions that are included in an instance depend on the variable we are trying to predict and on the properties of the ML algorithm used.

WEKA [Witten & Frank, 2005], for Waikato Environment for Knowledge Analysis, is a collection of various ML algorithms, implemented in Java, that can be used for data mining problems. Apart from applying ML algorithms on datasets and analyzing the results generated, WEKA also provides options for pre-processing and visualization of the dataset. It can be extended by the user to implement new algorithms.



Suppose that we want to predict the present temperature at a site *C*. To do this we use eight input attributes: the previous two hours temperature together with the present hour temperature at *C* and two nearby locations *A* and *B*. The output attribute is the present hour temperature at *C*. Let *temp_<site><hour>* denote temperature taken at hour <hour> at location <site>, then the data instance will take the form,

$temp\_A_{t-2}$, $temp\_A_{t-1}$, $temp\_A_t$, $temp\_B_{t-2}$, $temp\_B_{t-1}$, $temp\_B_t$, $temp\_C_{t-2}$, $temp\_C_{t-1}$, $temp\_C_t$

with the last attribute, $temp\_C_t$, being the output attribute. We will refer to this example as 'our weather example' in the following sections in this report.

ML algorithms can be broadly classified into two groups, classification and regression algorithms. We describe these two types of classifications and some of the ML algorithms from each of these groups.

Section 3

Classification Algorithms

Algorithms that classify a given instance into a set of discrete categories are called classification algorithms. These algorithms work on a training set to come up with a model or a set of rules that classify a given input into one of a set of discrete output values. Most classification algorithms can take inputs in any form, discrete or continuous although some of the classification algorithms require all of the inputs also to be discrete. The output is always in the form of a discrete value. Decision trees and Bayes nets are examples of classification algorithms.

To be able to apply classification algorithms on our weather example we need to convert the output attribute into classes. This is generally done by discretization, which is the process of dividing a continuous variable into classes. Discretization can be done in many ways, a simple approach would be to divide the temperature into ranges of 5 degrees and giving each range a name or by using entropy-based algorithms [Fayyad & Irani, 1993; Dougherty et al., 1995]. Inputs attributes can be left as continuous if the algorithm deals with them or they can be converted into discrete values depending on the algorithm.

We describe in detail the classification algorithms that have been used in this thesis in the sub-sections below.

The J48 Decision Tree Algorithm

J48 is a decision tree learner based on C4.5 [Quinlan, 1993]. C4.5 is an update of the ID3 algorithm [Quinlan, 1986]. We describe here the ID3 algorithm.

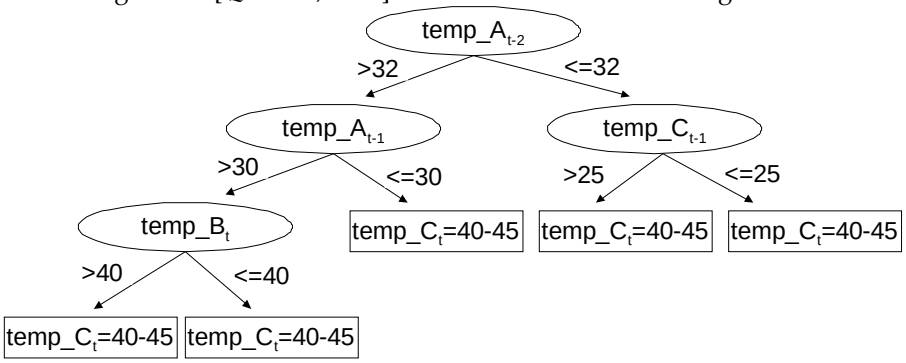


Figure 2: A Decision Tree to predict the current temperature at site *C* based on temperature readings taken from a set of nearby sites.

A decision tree classifies a given instance by passing it through the tree starting at the top and moving down until a leaf node is reached. The value at that leaf node gives the predicted output for the instance. At each node an attribute is tested and the branches from the node correspond to the values that attribute can take. When the instance reaches a node, the branch taken depends on the value it has for the attribute being tested at the node. A decision tree that can be used to predict the present hour temperature for site *C* in our weather example is given Figure 2. So if we were to classify an instance in our weather example with this tree we would start at the root node that tests the attribute $temp\_A_{t-2}$ and based on the value taken by this attribute in the given instance we will take the left or right branch. When we reach a node after taking a branch, the attribute associated with it is tested and the corresponding branch taken until we reach a leaf node, which gives the value taken for the output attribute $temp\_C_t$.

The ID3 algorithm builds a decision tree based on the set of training instances given to it. It takes a greedy top-down approach for the construction of the tree, starting with the creation of the root node. At each node the attribute that best classifies all the training instances that have reached that node is selected as the test attribute. At a node only those attributes are considered which were not used for classification at other nodes above it in the tree. To select the best attribute at a node, the *information gain* for each attribute is calculated and the attribute with the highest information gain is selected. Information gain for an attribute is defined as the reduction in *entropy* caused by splitting the instances based on values taken by the attribute. The information gain for an attribute *A* at a node is calculated using

$$InformationGain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \left(\frac{|S_v|}{|S|} Entropy(S) \right),$$

where *S* is the set of instances at that node and $|S|$ is its cardinality, S_v is the subset of *S* for which attribute *A* has value *v*, and entropy of the set *S* is calculated as

$$Entropy(S) = \sum_{i=1}^{numclasses} - p_i \log_2 p_i,$$

where p_i is the proportion of instances in S that have the i^{th} class value as output attribute.

A new branch is added below the node for each value taken by the test attribute. The training instances that have the test attribute value associated with the branch taken are passed down the branch, and this subset of training instances is used for the creation of further nodes. If this subset of training instances has the same output class value then a leaf is generated at the branch end, and the output attribute is assigned that class value. In the case where no instances are passed down a branch then a leaf node is added at the branch end that assigns the most common class value in the training instances to the output attribute. This process of generating nodes is continued until all the instances are correctly

classified or all the attributes have been used or when its not possible to divide the examples.

Extensions were added to the basic ID3 algorithm to (1) deal with continuous valued attributes, (2) deal with instances that have missing attribute values and to (3) prevent overfitting the data.

When a discrete valued attribute is selected at a node the number of branches formed is equal to the number of possible values taken by the attribute. In the case of a continuous valued attribute two branches are formed based on a threshold value that best splits the instances into two. For example, in Figure 2.4 the attribute at the root node, $temp\_A_{t-2}$, has a threshold value of 32. The threshold is the selected as the value of the attribute that maximizes the information gain of the given training instances. Fayyad & Irani [1993] extended this approach to split a continuous-valued attribute into more than two intervals.

There may arise cases where an instance has no value for an attribute (i.e., missing values) or has an unknown attribute value. The missing value can be replaced by the most common value for that attribute among the training instances that reach the node where this attribute is tested. In C4.5, the probability for each possible value taken by the attribute with missing value is calculated, based on the number of times it is seen in the training instances at a node. The probability values are then used for calculation of information gain at the node.

In the ID3 algorithm, sometimes due to too small of a training set being used, the tree built correctly classifies the training instances but fails when applied on the entire distribution of data because it focuses on the spurious correlation in the data when the remaining amount of data is small; this is known as *overfitting*. To avoid overfitting, C4.5 uses a technique called rule-post pruning. In rule post-pruning, after the tree is built, it is converted into a set of rules. For example, the rule generated for leftmost path of the tree in Figure 2 is

IF ($temp\_A_{t-2} > 32$ AND $temp\_A_t > 30$ AND $temp\_B_t > 40$)
THEN $temp\_C_t = 40\text{-}45$.

From each rule generated for the tree, those antecedents are pruned (i.e., removed) which do not reduce the accuracy of the model. Accuracy is measured based on the instances present in the validation set, which is a subset of the training set not used for building the model.

Naive Bayes

Naive Bayes [Good, 1965; Langley et al., 1992] is a simple probabilistic classifier based on Bayes' rule. The naive Bayes algorithm builds a probabilistic model by learning the conditional probabilities of each input attribute given a possible value taken by the output attribute. This model is then used to predict an output value when we are given a set of inputs. This is done by applying Bayes' rule on the conditional probability of seeing a possible output value when the attribute

values in the given instance are seen together. Before describing the algorithm we first define the Bayes' rule.

Bayes' rule states that

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)},$$

where $P(A|B)$ is defined as the probability of observing A given that B occurs. $P(A|B)$ is called posterior probability, and $P(B|A)$, $P(A)$ and $P(B)$ are called prior probabilities. Bayes' theorem gives a relationship between the posterior probability and the prior probability. It allows one to find the probability of observing A given B when the individual probabilities of A and B are known, and the probability of observing B given A is also known.

The naive Bayes algorithm uses a set of training examples to classify a new instance given to it using the Bayesian approach. For an instance, the Bayes rule is applied to find the probability of observing each output class given the input attributes and the class that has the highest probability is assigned to the instance. The probability values used are obtained from the counts of attribute values seen in the training set.

In our weather example, for a given instance with two input attributes $temp\_A_t$ and $temp\_B_t$, with values a and b respectively, the value v_{MAP} assigned by the naive Bayes algorithm to the the output attribute $temp\_C_t$ is the one that has the highest probability across all possible values taken by output attribute; this is known as the maximum-a-posteriori (MAP) rule. The probability of the output attribute taking a value v_j when the given input attribute values are seen together is given by

$$P(v_j \mid a, b).$$

This probability value as such is difficult to calculate. By applying Bayes theorem on this equation we get

$$P(v_j \mid a, b) = \frac{P(a, b \mid v_j)P(v_j)}{P(a, b)} = P(a, b \mid v_j)P(v_j),$$

where $P(v_j)$ is the probability of observing v_j as the output value, $P(a,b|v_j)$ is the probability of observing input attribute values a, b together when output value is v_j. But if the number of input attributes $(a, b, c, d, ...)$ is large then we likely will not have enough data to estimate the probability $P(a, b, c, d, ... | v_j)$.

The naive Bayes algorithm solves this problem by using the assumption of conditional independence for the all the input attributes given the value for the output. This means it assumes that the values taken by an attribute are not dependent on the values of other attributes in the instance for any given output. By applying the conditional independence assumption, the probability of observing an output value for the inputs can be obtained by multiplying the probabilities of individual inputs given the output value. The probability value $P(a, b | v_j)$ can then be simplified as

$$P(a,b \mid v_j) = P(a \mid v_j)P(b \mid v_j),$$

where P($a \mid v_j$) is the probability of observing the value a for the attribute $temp\_A_t$ when output value is v_j. Thus the probability of an output value v_j to be assigned for the given input attributes is

$$P(v_j \mid a,b) = P(v_j)P(a \mid v_j)P(b \mid v_j).$$

Learning in the Naive Bayes algorithm involves finding the probabilities of $P(v_j)$ and $P(a_i \mid v_j)$ for all possible values taken by the input and output attributes based on the training set provided. $P(v_j)$ is obtained from the ratio of the number of time the value v_j is seen for the output attribute to the total number of instances in the training set. For an attribute at position i with value a_i, the probability $P(a_i \mid v_j)$ is obtained from the number of times a_i is seen in the training set when the output value is v_j.

The naive Bayes algorithm requires all attributes in the instance to be discrete. Continuous valued attributes have to be discretized before they can be used. Missing values for an attribute are not allowed, as they can lead to difficulties while calculating the probability values for that attribute. A common approach to deal with missing values is to replace them by a default value for that attribute.

Bayesian Belief Networks (Bayes Nets)

The naive Bayes algorithm uses the assumption that the values of all the input attributes are conditionally independent given the value of the output attribute. But there may be cases when assuming conditional independence of all the given inputs, may not lead to appropriate predictions. Bayesian Belief Networks or Bayes Nets introduce the idea of applying conditional independence on a certain number of inputs rather than on all of them. This notion avoids the global assumption of conditional independence while maintaining some amount of conditional independence among the inputs.

A Bayesian Belief Network [Friedman et al., 1997; Pearl, 1988] is a directed acyclic graphical network model that gives the joint probability distribution for a set of attributes. Each attribute in the instance is represented in the network in the form of a node. In the network a directed connection from node X to node Y is made when X is a parent of Y which means that there is a dependence relation of Y on X, or in other words X has an influence on Y. Thus in this network an attribute at a node is conditionally independent of its non-dependents in the network given the state of its parent nodes. These influences are represented by conditional probabilities, which gives the probability of a value at a node that is conditional on the value of its parents. These probability values for a node are

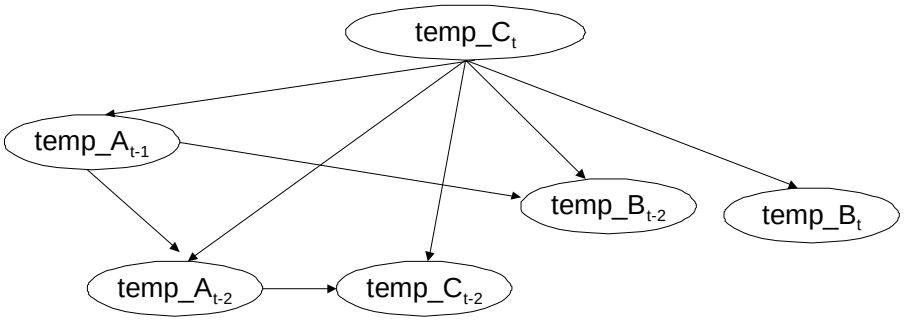


Figure 3: A Bayesian network to predict temperature *temp_C*$_t$ at a site. The arrows represent a direct relation between nodes. Each node is associated with a CPT.

arranged in a tabular form called a Conditional Probability Table (CPT). In the case of nodes with no parents, the CPT gives the distribution of the attribute at that node.

When a node is connected to a set of nodes, which are one step above in the hierarchy, these parent nodes have an influence on its behavior. This node is not affected by other nodes present in the given pool of nodes. It means the node is conditionally independent of all non-parent nodes when given its parents. The nodes which are more than one step above in hierarchy, that is parents of parents of a node, are not considered as directly influencing the node, as these nodes affect the nodes which are parents to the node in question and thus indirectly influence it. Thus the parents are considered for calculating the joint probability, as only the direct parents of a node influence the conditional probabilities at this node. Using conditional independence between nodes, the joint probability for a set of attribute values $y_1, y_2, .., y_n$ represented by the nodes $Y_1, Y_2, ..., Y_n$ is given by

$$P(y_1, ..., y_n) = \prod_{i=1}^{n} P(y_i \mid Parents(Y_i)),$$

where $Parents(Y_i)$ are the immediate parents of node Y_i. The probability values can be obtained directly from the CPTs associated with the node.

A Bayesian network requires that both input and output attributes be discrete. A simple Bayesian network for predicting temperature at a site in our weather example, using only a few of the input instances, is shown in Figure 3. Each node in the tree is associated with a CPT. For example, the CPT for the node *temp_A*$_{t-2}$ will contain the probability of each value taken by it when all possible values for *temp_A*$_{t-1}$ and *temp_C*$_t$ (i.e., its parents) are seen together. For a given instance, the Bayesian network can be used to determine the probability distribution of the target class by multiplying all the individual probabilities of values taken up by the individual nodes. The class value that has the highest probability is selected. The probability of a class value taken by the output attribute *temp_C*$_t$ for the given input attributes, using parental information of nodes from the Bayesian network in Figure 3 is

$P(temp\_C_t \mid temp\_A_{t-1}, temp\_A_{t-2}, temp\_B_t, temp\_B_{t-2}, temp\_C_{t-2}) =$

$P(temp\_C_t)*P(temp\_A_{t-1} | temp\_C_t)*P(temp\_A_{t-2} | temp\_A_{t-1}, temp\_C_t)*P(temp\_B_t | temp\_C_t)*$
$P(temp\_B_{t-2} | temp\_A_{t-1}, temp\_C_t)*P(temp\_C_{t-2} | temp\_A_{t-2}, temp\_C_t)$.

Learning in Bayes' Nets from a given training set involves finding the best performing network structure and calculating CPTs. To build the network structure, we start by assigning each attribute a node. Learning the network connections involves moving though the set of possible connections and finding the accuracy of the network for the given training set. The accuracy of the network can be determined by using a scoring criterion such as the Akaike's Information Criterion [Akaike, 1974], the Minimum Description Criterion [Rissanen, 1978] or the Cross-Validation Criterion. Allen & Greiner [2000] present a brief description of these scoring criterions along with their empirical comparisons. For a network, the CPTs are calculated at each node based on the information obtained from the training set.

The K2 algorithm [Cooper & Herskovits, 1992] can be used to learn the Bayesian network structure. K2 puts the given nodes in an order and then processes one node at a time. It adds an edge to this node from previously added nodes only when the network accuracy is increased after this addition. When no further connections can be added to the current node that increase the accuracy, the algorithm then moves to another node. This process continues until all nodes have been processed.

When all variables present in the network are seen in the training data, the probability values in the CPTs can be filled by counting the required terms. In the case of training data with missing variables the gradient ascent training [Russel et al., 1995] method can be used to learn values for the CPTs.

Section 4

Regression Algorithms

Algorithms that develop a model based on equations or mathematical operations on the values taken by the input attributes to produce a continuous value to represent the output are called of regression algorithms. The input to these algorithms can take both continuous and discrete values depending on the algorithm, whereas the output is a continuous value. We describe in detail the regression algorithms that have been used in this thesis below.

Linear Regression

The Linear Regression algorithm of WEKA [Witten & Frank, 2005] performs standard least squares regression to identify linear relations in the training data. This algorithm gives the best results when there is some linear dependency among the data. It requires the input attributes and target class to be numeric and it does not allow missing attributes values. The algorithm calculates a regression equation to predict the output (x) for a set of input attributes $a_1,a_2,.....,a_k$. The equation to calculate the output is expressed in the form of a linear combination of input attributes with each attribute associated with its respective weight $w_0,w_1,....,w_k$, where w_1 is the weight of a_1 and a_0 is always taken as the constant 1. An equation takes the form

$$x = w_0 + w_1 a_1 + ... + w_k a_k .$$

For our weather example the equation learned would take the form

$$temp\_C_t = w_0 + w_{At-2}\, temp\_A_{t-2} + w_{At-1}\, temp\_A_{t-1} + w_{At}\, temp\_A_t + w_{Bt-2}\, temp\_B_{t-2} + w_{Bt-1}\, temp\_B_{t-1} + w_{Bt}\, temp\_B_t + w_{Ct-2}\, temp\_C_{t-2} + w_{Ct-1}\, temp\_C_{t-1} ,$$

where $temp\_C_t$ is value assigned to the output attribute, and each term on the right hand side is the product of the values of the input attributes and the weight associated with each input.

The accuracy of predicting the output by this algorithm can be measured as the absolute difference between the actual output observed and the predicted output as obtained from the regression equation, which is also the error. The weights must be chosen in such as way that they minimize the error. To get better accuracy higher weights must be assigned to those attributes that influence the result the most.

A set of training instances is used to update the weights. At the start, the weights can be assigned random values or all set to a constant (such as 0). For the first instance in the training data the predicted output is obtained as

$$w_0 + w_1 a_1^{(1)} + \ldots\ldots\ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)},$$

where the superscript for attributes gives the instance position in the training data. After the predicted outputs for all instances are obtained, the weights are reassigned so as to minimize the sum of squared differences between the actual and predicted outcome. Thus the aim of the weight update process is to minimize

$$\sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right),$$

which is the sum of the squared differences between the observed output for the i^{th} training instance ($x^{(i)}$) and the predicted outcome for that training instance obtained from the linear regression equation.

LeastMedSquare

The WEKA LeastMedSquare or Least Median Squares of Regression algorithm [Rousseeuw, 1984] is a linear regression method that minimizes the median of the squares of the differences from the regression line. The algorithm requires input and output attributes to be continuous, and it does not allow missing attribute values. Standard linear regression is applied to the input attributes to get the predict the output. The predicted output x is obtained as

$$w_0 + w_1 a_1^{(1)} + \ldots\ldots\ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)},$$

where the a_i are input attributes and w_i are the weights associated with them.

In the LeastMedSquare algorithm, using the training data, the weights are updated in such a way that they minimize the median of the squares of the difference between the actual output and the predicted outcome using the regression equation. Weights can be initially set to random values or assigned a scalar value. The aim of the weight update process is to determine new weights to minimize

$$\underset{i}{median} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right),$$

where i ranges from 1 to the number of instances in the training data that is being used, $x^{(i)}$ is the actual output for the training instance i, and the predicted outcome for that training instance is obtained from the regression equation.

M5P

The M5P or M5Prime algorithm [Wang & Witten, 1997] is a regression-based decision tree algorithm, based on the M5 algorithm by Quinlan [1992]. M5P is developed using M5 with some additions made to it. We will first describe the M5 algorithm and then the

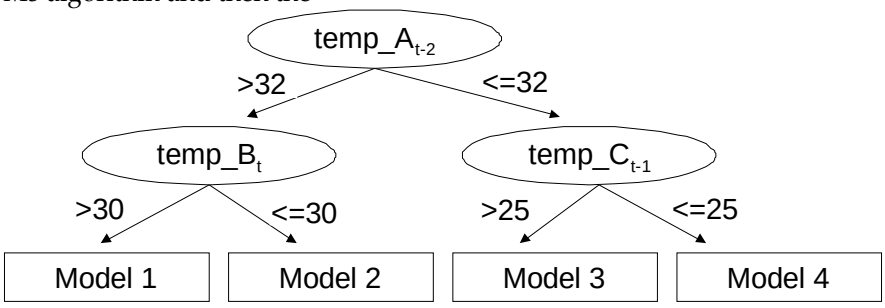


Figure 4: A M5 model tree for predicting temperature at a site. The decision taken at a node is based on the test of the attributes mentioned at that node. Each model at a leaf takes the form $w_0+w_1a_1+ .... +w_ka_k$ where k is the number of input attributes.

features added to it in M5P.

M5 builds a tree to predict numeric values for a given instance. The algorithm requires the output attribute to be numeric while the input attributes can be either discrete or continuous. For a given instance the tree is traversed from top to bottom until a leaf node is reached. At each node in the tree a decision is made to follow a particular branch based on a test condition on the attribute associated with that node. Each leaf has a linear regression model associated with it of the form

$$w_o + w_1 a_1 + ... + w_k a_k,$$

based on some of the input attributes $a_1,a_2,.....,a_k$ in the instance and whose respective weights $w_0,w_1,....,w_k$ are calculated using standard regression. As the leaf nodes contain a linear regression model to obtain the predicted output, the tree is called a model tree. When the M5 algorithm is applied on our weather example, the model tree generated will take a form as shown in Figure 4.

To build a model tree, using the M5 algorithm, we start with a set of training instances. The tree is built using a divide-and-conquer method. At a node, starting with the root node, the instance set that reaches it is either associated with a leaf or a test condition is chosen that splits the instances into subsets based on the test outcome. A test is based on an attributes value, which is used to decide which branch to follow. There are many potential tests that can be used at a node. In M5 the test that maximizes the error reduction is used. For a test the expected error reduction is found using

$$\Delta error = stdev(S) - \sum_i \left(\frac{|S_i|}{|S|} stdev(S_i) \right),$$

where S is the set of instance passed to the node, *stdev(S)* is its standard deviation, S_i is the subset of S resulting from splitting at the node with the i^{th} outcome for the test. This process of creating new nodes is repeated until a there are too few instances to proceed further or the variation in the output values in the instances that reach the node is small.

Once the tree has been built, a linear model is constructed at each node. The linear model is a regression equation. The attributes used in the equation are those that are tested or are used in linear models in the sub-trees below this node. The attributes tested above this node are not used in the equation as their effect on predicting the output has already been captured in the tests done at the above nodes. The linear model built is further simplified by eliminating attributes in it. The attributes whose removal from the linear model leads to a reduction in the error are eliminated. The error is defined as the absolute difference between the output value predicted by the model and the actual output value seen for a given instance.

The tree built can take a complex form. The tree is pruned so as to make it simpler without losing the basic functionality. Starting from the bottom of the tree, the error is calculated for the linear model at each node. If the error for the linear model at a node is less than the model sub-tree below then the sub-tree for this node is pruned. In the case of missing values in training instances, M5P changes the expected error reduction equation to

$$\Delta error = \frac{m}{|S|} * \beta(i) * \left[stdev(S) - \sum_i \left(\frac{|S_i|}{|S|} stdev(S_i) \right) \right],$$

where m is the number of instances without missing values for that attribute, S is the set of instances at the node, $\beta(i)$ is the factor multiplied in case of discrete attributes, j takes values L and R with S_L and S_R being the sets obtained from splitting at that attribute.

MultiLayer Perceptron

A MultiLayer Perceptron (MLP) [Bishop, 1995] is a neural network that is trained using backpropagation. MLPs consist of multiple layers of computational units that are connected in a feed-forward way forming a directed connection from lower units to a unit in a subsequent layer. The basic structure of MLP consists of an input layer, one or more hidden layers and one output layer. Units in the hidden layer are termed *hidden* as their output is used only in the network and is not seen outside the network. An MLP with two hidden layers that can be used to predict temperature in our weather example is shown in Figure 5. The output from a unit is used as input to units in the subsequent layer. The connection between units in subsequent layers has an associated weight.

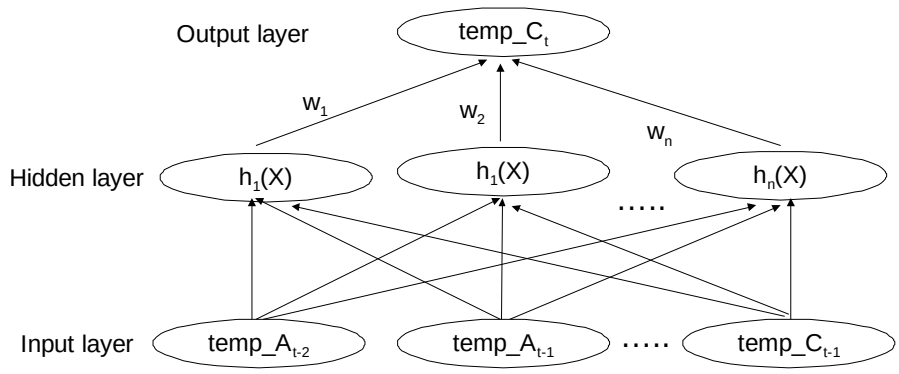


Figure 5: A multilayer perceptron with two hidden layers to predict temperature at a site. Each connection is associated with a weight. Hidden and output units are sigmoid units.

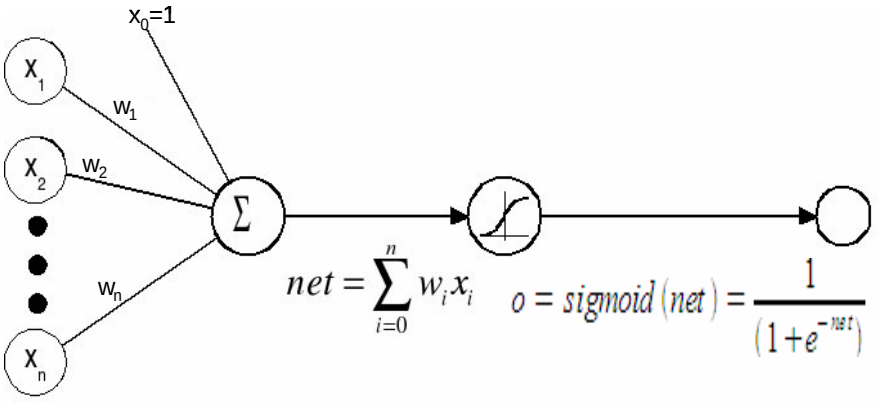


Figure 6: A sigmoid unit that takes inputs x_i, w_i the weights associated with the inputs and *sigmoid* the resulting output from the unit.

The hidden and output units are based on sigmoid units. A sigmoid unit calculates a linear combination of its input and then applies the sigmoid function on the result. The sigmoid function, for net input x is

$$sigmoid(x) = \frac{1}{(1+e^{-x})}.$$

The output of a sigmoid unit, *sigmoid(x)*, is a continuous function of its input (x) and is in the range of 0 to 1. A sigmoid unit is shown in Figure 6. In addition to the inputs supplied to it, the sigmoid unit also takes in a constant input of 1.

An MLP learns its weights using the backpropagation algorithm [Rumelhart et al., 1986]. The backpropagation algorithm takes a set of training instances for the learning process. For the given feed-forward network, the weights are initialized to small random numbers. Each training instance is passed through the network and the output from each unit is computed. The target output is compared with the output computed by the network to calculate the error and this error value is fed back through the network. To adjust the weights, backpropagation uses gradient descent to minimize the squared error between the target output and the computed output. At each unit in the network, starting from the output unit and moving down to the hidden units, its error value is used to adjust weights of its connections so as to reduce the error. The weights are updated using

$$w_{ji} = w_{ji} + \eta \delta_j x_{ji},$$

where w_{ji} is the weight from unit i to unit j, x_{ij} is the input from unit i to unit j, η is the learning rate and δ_j is error obtained at unit j. This process of adjusting the weights using training instances is iterated for a fixed number of times or is continued until the error is small or cannot be reduced.

To improve the performance of the backpropagation algorithm, the weight-update made at the n^{th} iteration of the backpropagation is made partially dependent to the amount of weight changed in the $n\text{-}1^{st}$ iteration. The amount by which the $n\text{-}1^{st}$ iteration contributes is determined by a constant term called momentum (α). The new rule used for weight-update at the n^{th} iteration is

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \, \Delta w_{ji}(n-1).$$

This momentum term is added to achieve faster convergence to a minimum in some cases.

RBF Network

An RBF or Radial Basis Function Network [Buhmann & Albovitz, 2003; Orr, 1996] is another type of a feed-forward neural network. It has three layers: the input, hidden and output layer. It differs from an MLP in the way the hidden layer units perform calculations. An RBF Network can build both regression and classification models. We will describe the regression model.

In an RBF Network, inputs from the input layer are mapped to each of the hidden units. The hidden units use radial functions for activation such as Gaussian, multiquadric, inverse-multiquadric and Cauchy [Orr, 1996]. The RBF Network of WEKA [Witten & Wang, 2005] uses the bell-shaped Gaussian function. The activation $h(x)$ of the Gaussian function for a given input x decreases monotonically as the distance between the center c of the Gaussian and x increases. A Gaussian function is useful in finding the activation at a hidden unit, as the activation of inputs depends on their closeness to center of the hidden unit and thus can be used as a effective method to distinguish between inputs. The Gaussian function is of the form

$$h(x) = \exp\left(\frac{-(x-c)^2}{r^2}\right).$$

The output layer takes in linear combination of outputs from hidden units and is similar to a regression model. An RBF Network for our weather models will be of the form shown in Figure 7.

AN RBF Network takes the inputs and the hidden units as points in space. The activation of a hidden unit depends on the distance between the point in space representing the input values and the point for that hidden unit. The distance is converted

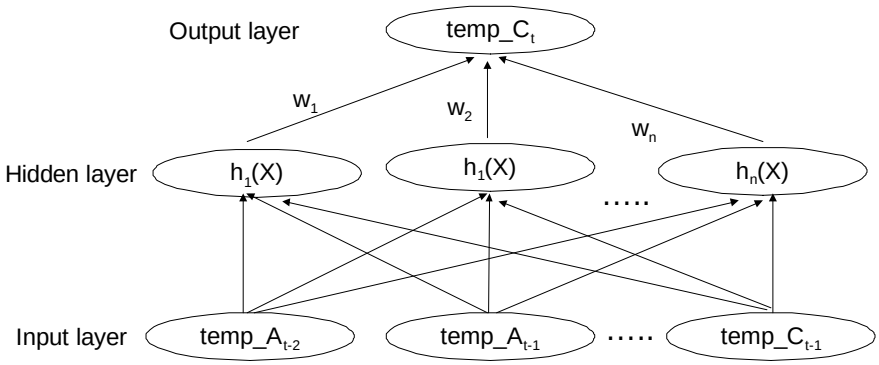


Figure 7: An RBF Network with *n* hidden units to predict temperature at a site. X is the input vector given to the network and output. *temp_Ct* is the sum of the products of activations from the hidden units and the weights associated with them. Each hidden unit has its own learned center.

into a similarity measure by the Gaussian function. The point in space for the hidden unit is obtained from the center of the Gaussian for that hidden unit. The width of the Gaussian is a learned parameter as well.

An RBF Network is trained to learn the centers and widths of the Gaussian function for hidden units, and then to adjust weights in the regression model that is used at the output unit. To learn the centers of the Gaussian functions the k-means clustering algorithm can be used that clusters the training instances to obtain k Gaussian functions for each attribute in the instance. After the parameters for the Gaussian function at the hidden units have been found, the weights from these units to the output unit are adjusted using Linear Regression. The process can be repeated to learn in an EM manner.

The Conjunctive Rule Algorithm

The Conjunctive Rule algorithm in WEKA learns a single rule that can predict an output value. It can predict both discrete and numeric classes. An example of a conjunctive rule that can be developed from our weather example is

IF $temp\_A_{t-2} > 30$ *AND* $temp\_C_{t-1} < 90$ *AND* $temp\_B_{t-1} > 40$ *THEN* $temp\_C_t = 30$.

A conjunctive rule consists of a set of antecedents (e.g., $temp\_A_{t-2} > 30$, $temp\_C_{t-1} < 90$) *AND*ed together to give the consequent (e.g., $temp\_C_t$). Antecedents consist of relations between relevant attributes and the consequent indicates an output value.

The learning process in the Conjunctive Rule algorithm attempts to come up with a rule for all relevant attributes based on the training data. The algorithm learns by calculating the variance reduction for all possible antecedents and then selects the one that reduces the variance the most. In cases when the learned rule becomes too complex then it is pruned using reduced error pruning similar to that in the J48 system.

In cases when a test instance is seen that is not covered by the rule, then the attribute whose value is not included in the rule is assigned the default value for that attribute.

Section 5

Conclusions

Machine learning (ML) methods are useful for building data models. We have discussed classification algorithms such as J48 decision trees, Naive Bayes and Bayesian Networks, regression algorithms such as Linear Regression, Least Median Square (LMS), M5P regression trees, MultiLayer Perceptron, and RBF Networks.

Section 6

Bibliography

[Akaike, 1974] Akaike, H., *A new look at the statistical model identification*. IEEE Transaction on Automatic Control , vol. AC-19, pp. 716-723, 1974.

[Allen & Greiner, 2000] Allen, T. and Greiner, R., *A model selection criteria for learning belief nets: An empirical comparison*, Proceedings of the International Conference on Machine Learning, pp. 1047-1054, 2000.

[Bishop, 1995] Bishop, C., *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[Buhmann & Albovitz, 2003] Buhmann, M., Albowitz, M., *Radial Basis Functions: Theory and Implementations*, Cambridge University Press, 2003.

[Cooper & Herskovits, 1992] Cooper, G. and Herskovits, E., *A Bayesian Method for the Induction of Probabilistic Networks from Data*, Machine Learning, vol. 9, pp. 309-347, 1992

[Dougherty et al., 1995] Dougherty, J., Kohavi, R. and Sahami, M., *Supervised and unsupervised discretization of continuous features*, Proceedings of the Twelfth International Conference on Machine Learning, pp 94-202, 1995.

[Fayyad & Irani, 1993] Fayyad, U. and Irani, K., *Multi-interval discretization of continuous-valued attributes for classification learning*, Proceedings of 13th International Joint Conference on Artificial Intelligence, pp 1022-1027, Morgan Kaufmann, 1993.

[Friedman et al., 1997] Friedman, N., Geiger, D. and Goldszmidt, M., *Bayesian network classifiers*, Machine Learning, vol. 29, pp. 131-163, 1997.

[Good, 1965] Good, I., *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*, M.I.T. Press, 1992.

[Langley et al., 1992] Langley, P., Iba, W. and Thompson, K., *An Analysis of Bayesian Classifiers*, Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 223-228, AAAI Press, 1992.

[Orr, 1996] Orr, M., *Introduction to radial basis function networks*. Technical report, Institute for Adaptive and Neural Computation of the Division of Informatics at

Edinburgh University, Scotland, UK, 1996, http://www.anc.ed.ac.uk/~mjo/papers/intro.ps.gz.

[Pearl, 1988] Pearl, J., *Probabilistic reasoning in intelligent systems*, Morgan Kaufman, 1988

[Quinlan, 1986] Quinlan, R., *Induction of Decision Trees,* Machine Learning, vol. 1, pp. 81-106, 1986.

[Quinlan, 1992] Quinlan, R., *Learning with Continuous Classes*, Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, pp. 343-348. World Scientific, Singapore, 1992.

[Quinlan, 1993] Quinlan, R., C4.5: *Programs for Machine Learning*, Morgan Kaufmann, San Francisco, 1993.

[Rissanen, 1978] Rissanen, J., *Modeling by shortest data description*. Automatica, vol. 14, pp. 465-471, 1978.

[Rousseeuw, 1984] Rousseeuw, P., *Least Median Squares of Regression*, Journal of American Statistical Association, vol. 49, pp. 871-880, December 1984.

[Rumelhart et al., 1986] Rumelhart, D., Hinton, G. and Williams, R., *Learning internal representations by error propagation*, Parallel Distributed Processing: Explorations in the Microstructures of Cognition, vol.I, pp. 318–362, MIT Press, 1986.

[Russell et al., 1995] Russell, S., Binder, J., Koller, D. and Kanazawa, K., *Local Learning in Probabilistic Networks with Hidden Variables*, Proceedings of the 14th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1995.

[Wang & Witten, 1997] Wang, Y. and Witten, I., *Inducing Model Trees for Continuous Classes*, In Poster Papers of the Ninth European Conference on Machine Learning, pp. 128-137, Prague, Czech Republic, April, 1997.

[Witten & Frank, 2005] Witten, I. and Frank, E., *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.