Decision Tree Classifier

Related terms:

Decision Tree, Classification (Machine Learning), Neural Networks, Classification, Random Decision Forest, Classification Accuracy, Nearest Neighbour

View all Topics

Management of Complexity and Information Flow

E. Szczerbicki, in Agile Manufacturing: The 21st Century Competitive Strategy, 2001

3.1 Decision tree classifiers

Decision tree classifiers are used successfully in many diverse areas. Their most important feature is the capability of capturing descriptive decisionmaking knowledge from the supplied data. Decision tree can be generated from training sets. The procedure for such generation based on the set of objects (**S**), each belonging to one of the classes $\mathbf{C}_1$, $\mathbf{C}_2$, …, $\mathbf{C}_k$ is as follows [9]:

Step 1.	If all the objects in **S** belong to the same class, for example $\mathbf{C}_i$, the decision tree for **S** consists of a leaf labelled with this class.
Step 2.	Otherwise, let T be some test with possible outcomes O_1, O_2,..., O_n. Each object in **S** has one outcome for T so the test partitions **S** into subsets $\mathbf{S}_1$, $\mathbf{S}_2$,... $\mathbf{S}_n$ where each object in $\mathbf{S}_i$ has outcome O_i for T. **T** becomes the root of the decision tree and for each outcome O_i we build a subsidiary decision tree by invoking the same procedure recursively on the set $\mathbf{S}_i$.

The above procedure is applied to the training set of objects related to the flow of information for a given AMS subsystem in Table 1. The training sets are delivered from the initial analysis based on the quantitative model of AMS functioning as

presented in Section 2. Each object is described by the relating attributes and belongs to one of the agent decision classes exchange_information ("yes" in the last column) or do_not_exchange_information ("no" in the last column).

Table 1. Training set for agent functioning

external environment	internal environment	type of dynamics	correlation	delay of information	decision
static	independent_actions	0	0	0	no
static	independent_actions	0	0.7	1	no
dynamic	dependent_actions	1.5	–0.5	1	yes
static	dependent_actions	0	0	0	yes
static	independent_actions	0	1	2	no
static	dependent_actions	0	0.5	2	yes
static	dependent_actions	0	–1	3	no
static	independent_actions	0	–1	0	no
static	dependent_actions	0	0.9	1	yes
static	dependent_actions	0	1	1	no

Suppose, as it was done in Section 2, that we are interested in decision making situations involving static environment only. When for this case the set is partitioned by testing on internal_environment and then on correlation, the resulting structure is equivalent to the decision tree shown in Figure 2.

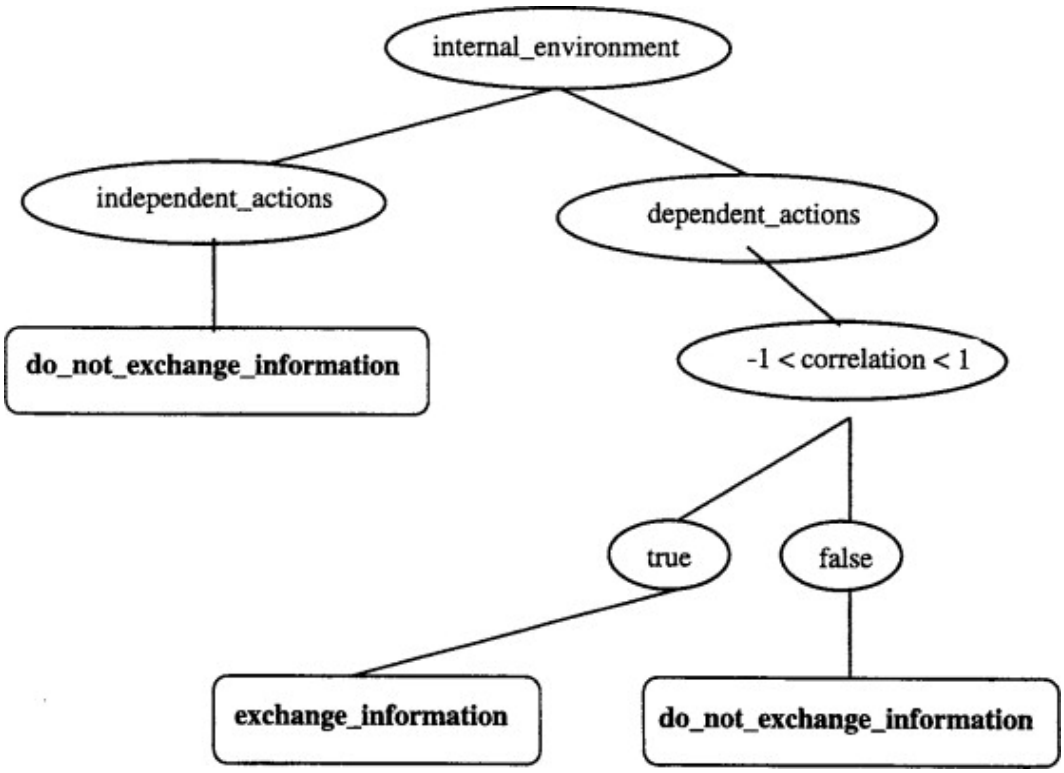

Figure 2. Decision tree classifier for AMS information flow related decisionmaking

The following rules can be delivered from Figure 2.

RULE 1

IF	an external environment of an agent is static
AND	it is described by random variables
AND	there is no interaction in the internal environment
THEN	communication (exchange of information) between agent elements is not necessary

RULE 2

IF	an external environment of an agent is static
AND	it is described by random variables
AND	there is interaction in the internal environment
AND	the relationship between variables describing the external environment is of statistical character
THEN	exchange of information between agent elements should be organised

THEN exchange of information between agent elements should be organised

RULE 3

IF	an external environment of an agent is static
AND	it is described by random variables
AND	there is interaction in the internal environment

AND	the relationship between variables describing the external environment is given by function dependence
THEN	exchange of information between agent elements is not necessary

The above shows how the simple decision tree in Figure 2 can be used to retrieve some of the knowledge concerning the functioning of an AMS subsystem in static environment. The resulting rules are exactly the same as those that were developed using the analytical model presented in Section 2. The decision tree, once developed, can support decision situations that are not covered by the training set. That is why the production rules can be formulated as a generalised statements. As it has been illustrated the use of decision trees is simple and as effective as the analysis based on a rigorous mathematical model. Another tool that can be of much help in the process of knowledge retrieval for AMS information flow and evaluation is the technique based on connectionist systems.

> Read full chapter

Machine learning and its application in microscopic image analysis

F. Xing, L. Yang, in Machine Learning and Medical Imaging, 2016

4.3.1.1 Structured edge detection

Since a decision tree classifier generates the actual prediction at the leaf nodes, more information (instead of only class likelihoods) can be stored at the leaf nodes. For example, in Kontschieder et al. (2011), structured class label information is stored at leaf nodes for semantic image segmentation. Similar to Dollar and Zitnick (2014) and Chen et al. (2015), we have stored edge structure information at the leaf nodes for structured muscle image edge detection. Different from traditional edge detection algorithms (Arbelaez et al., 2011), which take an image patch x as an input and compute the probability of the edge existence at the center pixel p, the output of our proposed structured edge detection algorithm is an edge mask around the central pixel p instead of the likelihood value. After the decision tree is learned, the median or mean of the edge masks sent to the leaf node will be stored as the leaf node output, as shown in Fig. 4.5.

The information gain criterion in Eq. (4.14) is effective in practice for decision tree training. In order to follow this criterion, the edge masks must be explicitly assigned proper class labels at each internal node of the tree during the training stage. One straightforward idea is to group the edge masks at a node into several clusters by an

unsupervised clustering algorithm such as *k*-means or mean-shift (Comaniciu and Meer, 2002), and then treat each cluster *id* as the class label for the sample belonging to that cluster. However, the edge masks do not reside in the Euclidean space so that direct grouping may not generate desired results. In addition, clustering in a high-dimension space (for a 16 × 16 edge mask) is computationally expensive. To address this problem, we propose to reduce the high-dimension edge masks to a lower dimensional subspace ($m << n$) using an autoencoder (Hinton and Salakhutdinov, 2006) before clustering the edge masks. For notation convenience, we use the matrix form and vector form of edge mask space interchangeably in this section.

Although the transformed data is used to choose a split function $h(x, \tau n)$ during the training of the decision tree, only the original edge masks are stored at leaf nodes for the prediction. Several sample edge masks learned and stored at the leaf nodes are shown in Fig. 4.6. As one can tell, many edge structures are unique for muscle cell boundaries, which demonstrates the effectiveness of the structured edge detection procedure. The proposed structured edge detection algorithm takes a 32 × 32 image patch as input and generates a 16 × 16 edge mask around the input's center pixel. The image patch is represented with the same high-dimensional feature used in Dollar and Zitnick (2014) and Arbelaez et al. (2014), and it is effective and computationally efficient. In total, two million samples are randomly generated to train the structured decision random forest, which consists of eight decision trees. The autoencoder model used in our work consists of an encoder with layers of sizes (16 × 16) – 512 – 256 – 30 and a symmetric decoder. The autoencoder model is trained once offline and applied to all decision trees, and the data compression is only performed at the root node.

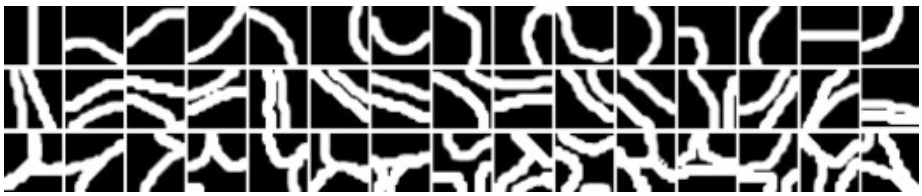

Fig. 4.6. Several sample edge masks learned and stored at the leaf nodes of the random decision trees.

> Read full chapter

Classification and Analysis of Facebook Metrics Dataset Using Supervised Classifiers

Ranjit Panigrahi, Samarjeet Borah, in Social Network Analytics, 2019

2.6 Tree Classifiers

The principle of splitting criteria is behind the intelligence of any decision tree classifier. Decision trees are presented similar to a flow chart, with a tree structure wherein instances are classified according to their feature values. A node in a decision tree represents an instance, outcomes of the test represented by branch, and the leaf node epitomized the class label. Three variations of decision trees are explored here, viz., Best First Decision Tree (BFTree) [62, 63], ForestPA [64], and SysFor [65] because of the fast model build time and processing speed.

> Read full chapter

Classification

Jiawei Han, ... Jian Pei, in Data Mining (Third Edition), 2012

8.6.4 Random Forests

We now present another ensemble method called random forests. Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a "forest." The individual decision trees are generated using a random selection of attributes at each node to determine the split. More formally, each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. During classification, each tree votes and the most popular class is returned.

Random forests can be built using bagging (Section 8.6.2) in tandem with random attribute selection. A training set, D, of D tuples is given. The general procedure to generate k decision trees for the ensemble is as follows. For each iteration,), a training set, D_i, of D tuples is sampled with replacement from D. That is, each D_i is a bootstrap sample of D (Section 8.5.4), so that some tuples may occur more than once in D_i, while others may be excluded. Let F be the number of attributes to be used to determine the split at each node, where F is much smaller than the number of available attributes. To construct a decision tree classifier, M_i, randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees. The trees are grown to maximum size and are not pruned. Random forests formed this way, with *random input selection*, are called Forest-RI.

Another form of random forest, called Forest-RC, uses random linear combinations of the input attributes. Instead of randomly selecting a subset of the attributes,

it creates new attributes (or features) that are a linear combination of the existing attributes. That is, an attribute is generated by specifying *L*, the number of original attributes to be combined. At a given node, *L* attributes are randomly selected and added together with coefficients that are uniform random numbers on . *F* linear combinations are generated, and a search is made over these for the best split. This form of random forest is useful when there are only a few attributes available, so as to reduce the correlation between individual classifiers.

Random forests are comparable in accuracy to AdaBoost, yet are more robust to errors and outliers. The generalization error for a forest converges as long as the number of trees in the forest is large. Thus, overfitting is not a problem. The accuracy of a random forest depends on the strength of the individual classifiers and a measure of the dependence between them. The ideal is to maintain the strength of individual classifiers without increasing their correlation. Random forests are insensitive to the number of attributes selected for consideration at each split. Typically, up to are chosen. (An interesting empirical observation was that using a single random input attribute may result in good accuracy that is often higher than when using several attributes.) Because random forests consider many fewer attributes for each split, they are efficient on very large databases. They can be faster than either bagging or boosting. Random forests give internal estimates of variable importance.

> Read full chapter

Object Classification Methods

Cheng-Jin Du, Da-Wen Sun, in Computer Vision Technology for Food Quality Evaluation, 2008

5 Decision tree

The decision tree acquires knowledge in the form of a tree, which can also be rewritten as a set of discrete rules to make it easier to understand. The main advantage of the decision tree classifier is its ability to using different feature subsets and decision rules at different stages of classification. As shown in Figure 4.6, a general decision tree consists of one root node, a number of internal and leaf nodes, and branches. Leaf nodes indicate the class to be assigned to a sample. Each internal node of a tree corresponds to a feature, and branches represent conjunctions of features that lead to those classifications. For food quality evaluation using computer vision, the decision tree has been applied to the problem of meat quality grading

(Song *et al.*, 2002) and the classification of "in the shell" pistachio nuts (Ghazanfari *et al.*, 1998).

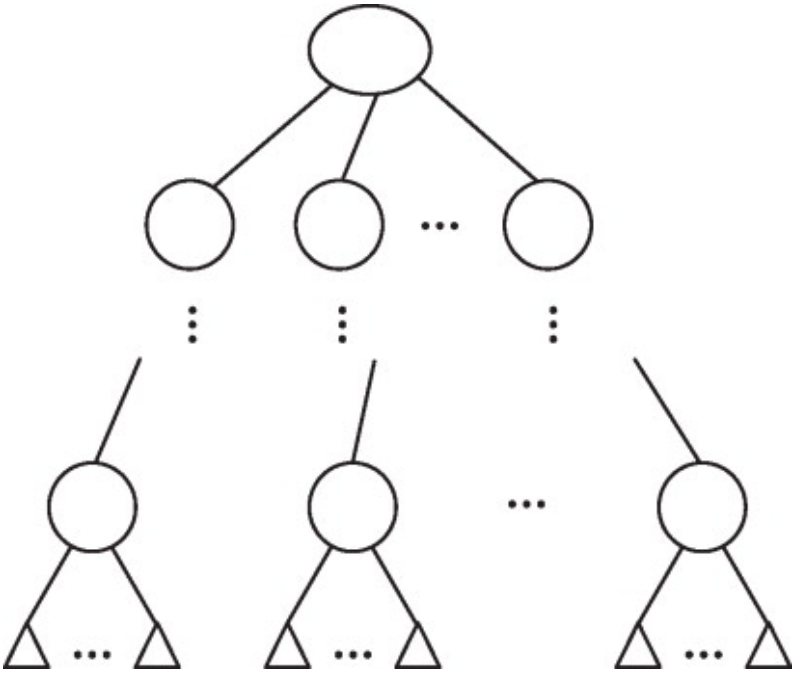

Figure 4.6. A general decision tree structure; , and Δ represent root, internal, and leaf nodes respectively.

The performance of a decision tree classifier depends on how well the tree is constructed from the training data. A decision tree normally starts from a root node, and proceeds to split the source set into subsets, based on a feature value, to generate subtrees. This process is repeated on each derived subset in a recursive manner until leaf nodes are created. The problem of constructing a truly optimal decision tree seems not to be easy. As one of the well-known decision tree methods, C4.5 is an inductive algorithm developed by Quinlan (1993); this is described in detail below.

To build a decision tree from training data, C4.5 employs an approach which uses information theoretically measured based on "gain" and "gain ratio." Given a training set TS, each sample has the same structure. Usually, the training set TS of food products is partitioned into two classes – AL (acceptable level) and UL (unacceptable level). The information (I) needed to identify the class of an element of TS is then given by

4.25

If the training set TS is partitioned on the basis of the value of a feature x_k into sets TS_1, TS_2, ..., TS_n, the information needed to identify the class of an element of TS can be calculated by the weighted average of $I(TS_i)$ as follows:

4.26

The information gained on a given feature is the difference between the information needed to identify an element of TS and the information needed to identify an element of TS after the value of the feature has been obtained. Therefore, the information gained on x_k is

(4.27)

The root of the decision tree is the attribute with the greatest gain. The process of building the decision tree is repeated, where each node locates the feature with the greatest gain among the attributes not yet considered in the path from the root.

The gain measurement has disadvantageous effects regarding the features with a large number of values. To cope with this problem, the gain ratio is introduced instead of the gain. For example, the gain ratio of x_k is defined as:

(4.28)

(4.29)

where $split(x_k, TS)$ is the information due to the split of TS on the basis of the value of feature x_k.

Sometimes, the decision tree obtained by recursively partitioning a training set as described above may become quite complex, with long and uneven paths. To deal with this shortcoming, the decision tree is pruned by replacing a whole sub-tree with a leaf node through an error-based strategy (Quinlan, 1993).

> Read full chapter

13th International Symposium on Process Systems Engineering (PSE 2018)

Gioele Casola, ... Hirokazu Sugiyama, in Computer Aided Chemical Engineering, 2018

Abstract

In this work, a data mining-based algorithm is presented for the pre-processing—e.g., noise removal, batch isolation—of continuously measured historical records of biopharmaceutical manufacturing. The algorithm applies approximate string match and decision tree classifiers to remove noise from commercial production data automatically. Single batches are isolated using k-means clustering,

after which algebraic semantic is used to characterize whether the data points within a batch describe the normal process execution or failures. The algorithm was applied to a dataset containing two months of manufacturing data in a cleaning and sterilization process. The performance of the algorithm was evaluated, resulting in a yield of 95 %, a mean time deviation of 1.3 ± 4.6 %, and a rate of misclassification of 1.5 %, which showed a high performance of the algorithm. This study supports the introduction of data-driven automation approaches as well as smart manufacturing in pharmaceutical manufacturing.

> Read full chapter

Irony, Sarcasm, and Sentiment Analysis

D.I. Hernández Farias, P. Rosso, in Sentiment Analysis in Social Networks, 2017

2.1 Irony Detection

One of the first studies in irony detection was by Carvalho et al. [11]. They worked on the identification of a set of surface patterns to identify ironic sentences in a Portuguese online newspaper. The most relevant features were the use of punctuation marks and emoticons. Veale and Hao [12] conducted an experiment by harvesting the web, looking for a commonly used framing device for linguistic irony: the simile (two queries "as * as *" and "about as * as *" were used to retrieve snippets from the web). They analyzed a very large corpus to identify characteristics of ironic comparisons, and presented a set of rules to classify a simile as ironic or nonironic.

Reyes et al. [13] analyzed tweets tagged with the hashtags #irony and #humor to identify textual features for distinguishing between them. They proposed a model that includes structural, morphosyntactic, semantic and psychological features. Additionally, they considered the polarity expressed in a tweet using the Macquarie Semantic Orientation Lexicon.[2] They experimented with different feature sets and a decision tree classifier, obtaining encouraging results (*F* measure of approximately 0.80).

Afterward, Reyes et al. [14] collected a corpus composed of 40,000 tweets, relying on the "self-tagged" approach. Four different hashtags were selected: #irony, #education, #politics, and #humor. Their model is organized according to four types of conceptual features—signatures (such as punctuation marks, emoticons, and discursive terms), unexpectedness (opposition, incongruency, and inconsistency in a text), style (recurring sequences of textual elements), and emotional scenarios (elements that symbolize sentiment, attitude, feeling, and mood)—by exploiting the Dictionary of Affect in Language (DAL).[3] They addressed the problem as a binary

classification task, distinguishing ironic tweets from nonironic tweets by using naïve Bayes and decision tree classifiers. They achieved an average *F* measure of 0.70.

Barbieri and Saggion [15] proposed a model to detect irony using lexical features, such as the frequency of rare and common terms, punctuation marks, emoticons, synonyms, adjectives, and positive and negative terms. They compared their approach with that of Reyes et al. [14] on the same corpus using a decision tree, and obtained results slightly better than those previously obtained. They concluded that rare words, synonyms ,and punctuation marks seem to be the most discriminating features. Hernández-Farías et al. [16] described an approach for irony detection that uses a set of surface text properties enriched with sentiment analysis features. They exploited two widely applied sentiment analysis lexicons: Hu&Liu[4] and AFINN.[5] They experimented with the same dataset used in [14, 15]. Their proposal was evaluated with use of a set of classifiers composed of Naïve bayes, decision tree, support vector machine (SVM), multilayer perceptron, and logistic regression classifiers. The proposed model improved on the previous results (*F* measure of approximately 0.79). The features related to sentiment analysis were the most relevant.

Buschmeier et al. [17] presented a classification approach using the Amazon review corpus collected by Filatova [10], which contains both ironic and nonironic reviews annotated by Mechanical Turk crowdsourcing. They proposed a model that takes into account features such as *n*-grams, punctuation marks, interjections, emoticons, and the star rating of each review (a particular feature from Amazon reviews, which, according to the authors, seems to help result in good performance in the task. They experimented with a set of classifiers (composed of naïve Bayes, logistic regression, decision tree, random forest, and SVM classifiers), achieving an *F*-measure rate of 0.74.

Wallace et al. [18] attempted to undertake the study of irony detection using contextual features, specifically by combining noun phrases and sentiment extracted from comments. They proposed exploiting information regarding the conversational threads to which comments belong. Their approach capitalizes on the intuition that members of different user communities are likely to be sarcastic about different things. A dataset of comments posted on Reddit[6] was used.[7]

Karoui et al. [19] recently presented an approach to separate ironic from nonironic tweets written in French. They proposed a two-stage model. In the first part they addressed the irony detection as a binary classification problem. Then the misclassified instances are processed by an algorithm that tries to correct them by querying Google to check the veracity of tweets with negation. They represented each tweet with a vector composed of six groups of features: surface (such as punctuation marks, emoticons, and uppercase letters), sentiment (positive and negative words), sentiment shifter (positive and negative words in the scope of an intensifier),

shifter (presence of an intensifier, a negation word, or reporting speech verbs), opposition (sentiment opposition or contrast between a subjective and an objective proposition), and internal contextual (the presence/absence of personal pronouns, topic keywords, and named entities). The authors experimented with an SVM as a classifier, achieving an *F* measure of 0.87.

To sum up, several approaches have been proposed to detect irony as a classification task. Many of the features employed have already been used in various tasks related to sentiment analysis such as polarity classification. The ironic intention is captured by the exploitation of mainly surface features such as punctuation marks and emoticons. These kinds of lexical cues have been shown to be useful to distinguish ironic content, especially in tweets. It may confirm in some way the necessity of users to add textual markers to deal with the absence of paralinguistic cues. Besides, many authors point out the importance of capturing the inherent incongruity in ironic utterances. To achieve this goal, the presence of opposite polarities (positive and negative words) and the use of semantically unrelated terms (synonyms and antonyms) have been considered in many approaches. Both kinds of features seem to be relevant to distinguish ironic from nonironic utterances. Decision trees are among the classifiers that produced the best results.

> Read full chapter

Enhancing energy efficiency in buildings through innovative data analytics technologiesa

A. Capozzoli, ... M.S. Piscitelli, in Pervasive Computing, 2016

5.3.2 Classification algorithms

Classification is the task of creating a model, which assigns objects to one of several predefined categories, to analyze the properties of classes and to automatically classify a new object. A classification model is typically used (i) to predict the class label for a new unlabeled data object, (ii) to provide a descriptive model explaining what features characterize objects in each class. Classification methods include various techniques such as the decision tree, rule-based, naive Bayes, neural networks, and support vector machines (SVM) classifiers. Each technique employs different learning algorithms to build models with good generalization capability, ie, models that accurately predict the class labels of previously unknown records. Generally, the construction of the classification model is performed by dividing the available

dataset into a *training set*, which is to be used in the construction phase of the classifier, and a *test set* for validation.

Decision tree classifiers provide a readable classification model that is potentially accurate in many different application contexts, including energy-based applications. The decision tree classifier (Pang-Ning et al., 2006) creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute. Each leaf represents class labels associated with the instance. Instances in the training set are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Starting from the root node of the tree, each node splits the instance space into two or more sub-spaces according to an attribute test condition. Then moving down the tree branch corresponding to the value of the attribute, a new node is created. This process is then repeated for the subtree rooted at the new node, until all records in the training set have been classified. The decision tree construction process usually works in a top-down manner, by choosing an attribute test condition at each step that best splits the records. There are many measures that can be used to determine the best way to split the records. The Gini index impurity-based criterion for growing the tree (Pang-Ning et al., 2006) is often exploited. It measures how often a randomly chosen instance from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the subset.

SVMs (Pang-Ning et al., 2006) were first proposed in statistical learning theory. SVM is able to deal with high-dimensional data and it generates a quite comprehensive (geometric) model. An SVM predictor is based on a kernel function K that defines a particular type of similarity measure between data objects. Examples of kernel functions are linear, RBF (radial basis function), polynomial, or sigmoid kernel. The SVM learning problem can be formulated as a convex optimization problem, in which different algorithms can be exploited to find the global minimum of the objective function.

ANNs (Pang-Ning et al., 2006) simulate biological neural systems. The network consists of an input layer, *n* hidden layers, and an output layer. Each layer is made up of nodes. Each node in a layer takes as input a weighted sum of the outputs of all the nodes in the previous layer, and it applies a nonlinear activation function to the weighted input. The network is trained with back propagation and learns by iteratively processing the set of training data objects. For each training data object, the network predicts the target value. Then, weights in the network nodes are modified to minimize the mean squared prediction error. These modifications are made in a backwards direction, that is, from the output layer through each hidden layer down to the first hidden layer.

> Read full chapter

Introduction to Data Mining

Domenico Talia, ... Fabrizio Marozzo, in Data Analysis in the Cloud, 2016

1.2.1 Parallel Classification

As described in Section 1.1.1.1, decision trees are an effective and popular technique for classification. They are tree-shaped structures that represent a way to classify items. Paths in those trees, from the root to a leaf, correspond to rules for classifying a data set whereas the tree leaves represent the classes and the tree nodes represent attribute values.

Independent parallelism can be exploited in decision tree construction assigning to a process the goal to construct a decision tree according to some parameters. If several processes are executed in parallel on different computing elements, a set of decision tree classifiers can be obtained at the same time. One or more of such trees can be selected as classifiers for the data.

Using the Task parallelism *approach* one process is associated to each subtree of the decision tree that is built to represent a classification model. The search occurs in parallel in each subtree, thus the degree of parallelism **P** is equal to the number of active processes at a given time. This approach can be implemented using the farm parallelism pattern in which one master process controls the computation and a set of **W** workers that are assigned to the subtrees. The result is a single decision tree built in a shorter time with respect to the sequential tree building.

According to *SPMD parallelism*, a set of processes executes the same code to classify the items of a subset of the data set. The **P** processes search in parallel in the whole tree using a partition **D/P** of the data set **D**. The global result is obtained by exchanging partial results among the processes. The data set partitioning may be operated in two different ways: (i) by partitioning the **D** tuples of the data set assigning **D/P** tuples per processor or (ii) by partitioning the **n** attributes of each tuple and assigning **D** tuples of **n/P** attributes per processor.

Kufrin (1997) proposed a parallel implementation of the C4.5 algorithm that uses the independent parallelism approach. Some other examples of parallel algorithms for building decision trees are Top-Down Induction of Decision Trees (Pearson, 1994) and SPRINT (Shafer et al., 1996).

> Read full chapter

E-CART

Pardeep Kumar, in Trends in Deep Learning Methodologies, 2021

4 Experiment

This chapter compares the E-CART with the E-CVFDT and CART decision tree with Gaussian approximation. The experiment is done by using a SEA concept generator and rotating hyperplane generator on Massive Online Analysis [20].

The data examples generated from the hyperplane generator are similar to the gradual concept drift data stream. By adjusting the data generation rate, the instantaneous concept drift stream is tested, whereas the examples generated from the SEA generator are similar to the instantaneous concept drift.

Computation time for building the decision tree by using the E-CART is calculated for the hyperplane generator and SEA generator and compared with the CVFDT algorithm. It is observed that the computation time for the SEA generator is very similar to the CVFDT approach, whereas in the case of the hyperplane generator, E-CART takes less computation time for building the decision tree classifier than CVFDT.

Table 10.3 represents the time taken by the CVFDT algorithm and the E-CART algorithm for the data examples generated by the SEA generator. The comparison is visualized in Fig. 10.2.

Table 10.3. Computation time comparison using the SEA generator.

Classification		**SEA generator**	
	n = 1,000,000	n = 100,000	N = 10,000
CVFDT	3.20 s	0.25 s	0.05 s
E-CART	3.37 s	0.32 s	0.06 s

CVFDT, Concept-adapting Very Fast Decision Tree; *E-CART*, Efficient Classification and Regression Tree.

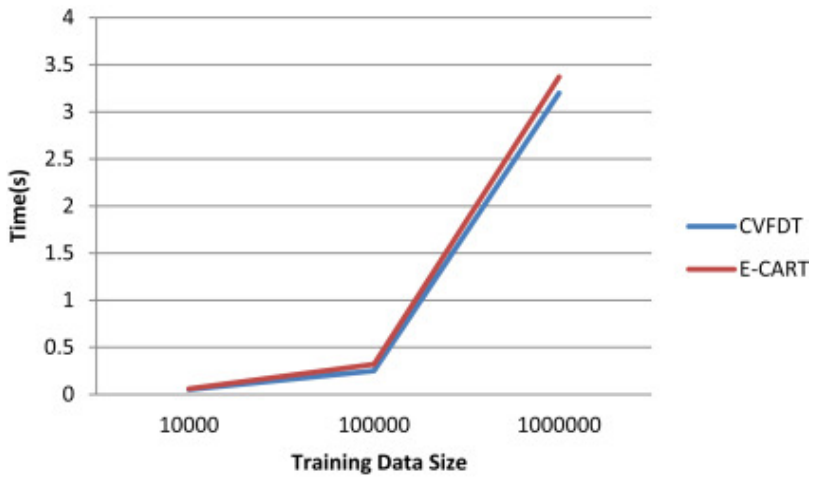

Figure 10.2. Computation time for the SEA generator. *CVFDT*, Concept-adapting Very Fast Decision Tree; *E-CART*, Efficient Classification and Regression Tree.

This graph shows that the performance of the E-CART is somewhat similar to the CVFDT approach given by G. Hulten and P. Domingos [7]. As said earlier, in the instantaneous concept drift data stream, the concept drift occurs instantaneously and then the older concept disappears immediately. When the new concept arrives in the stream, decision tree construction starts immediately rather than caching it as is done in the CVFDT approach to decision tree construction to mine the data stream.

Table 10.4 represents the comparison of E-CART and CVFDT approach in terms of performance efficiency on the dataset generated by the hyperplane generator. The computation time decreases with the proposed approach compared to the CVFDT.

Table 10.4. Computation time comparison using the hyperplane generator.

Classification		**Hyperplane generator**	
	n = 1,000,000	n = 100,000	N = 10,000
CVFDT	9.87 s	0.73 s	0.16 s
E-CART	7.30 s	0.34 s	0.15 s

CVFDT, Concept-adapting Very Fast Decision Tree; *E-CART*, Efficient Classification and Regression Tree.

The tabulated data shown in Table 10.4 is visualized in Fig. 10.3.

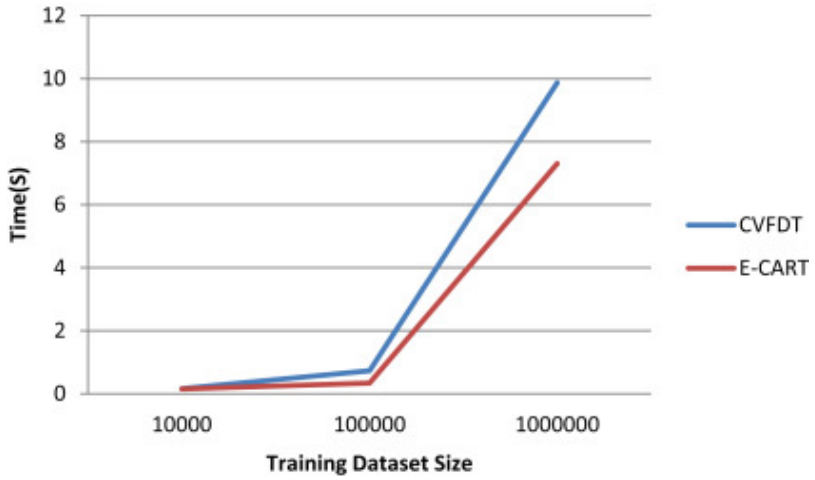

Figure 10.3. Computation time for a rotating hyperplane generator. *CVFDT*, Concept-adapting Very Fast Decision Tree; *E-CART*, Efficient Classification and Regression Tree.

The graph represented in Fig. 10.3 shows that the computation time decreases by using the proposed approach compared to the CVFDT. This happens because the proposed approach E-CART takes the concept drift into consideration. Thus in case of accidental concept drift, it ignores the new concept because the probability of the new concept appearing is very small. Thus rebuilding the decision tree for such a new concept is totally inefficient. Generally, the accidental concept drifts result in case of any noise in the data stream. In case of gradual concept drift, the cached system is used by the proposed approach, which makes it more efficient. This proposed approach results in 93.23% accuracy with the data size of 1,000,000,000 examples as compared to the accuracy of the CART decision tree approach proposed by Rutkowski et al., which is 90%. Accuracy visualization is shown in Fig. 10.4.

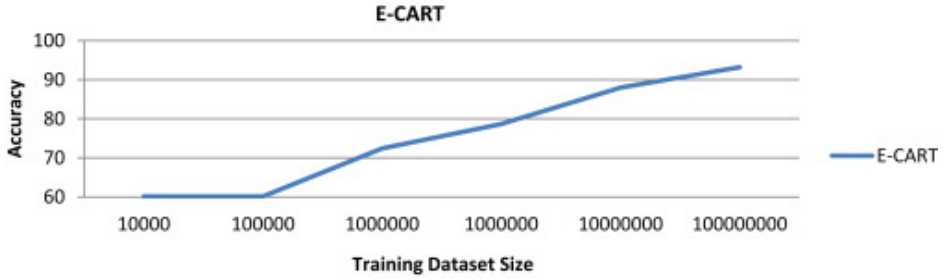

Figure 10.4. Accuracy of the proposed approach. *E-CART*, Efficient Classification and Regression Tree.

This graph shows an accuracy of 93.23%. It shows that the proposed approach for data stream mining is more efficient and better than the previous approaches.

> Read full chapter