Stream mining. Machine learning methods for data streams. Part II: decision trees

Maciej Grzenda, PhD, DSc
Maciej.Grzenda@pw.edu.pl

Faculty of Mathematics and Information Science
Warsaw University of Technology, Poland

Warszawa, 2025

Introduction

Classification defined

Classification is one key concepts in ML. A classifier is a ML model that maps object features to one of discrete classes [5].

Sample classification tasks:

Object	Features of the object	Classes
Bank client	Socio-demographic features of the client	accept credit card application — reject the application
E-mail message	List of words/phrases present in the message	spam — ham (i.e. non-spam)
A patient	Results of medical tests	Patient suffering from cardiovascular diseases — not suffering

Assuming all S features of the objects can be expressed with real numbers, a classifier is a function:

$f : \mathbb{R}^S \longrightarrow \{C_1, C_2, ..., C_k\}, k \geq 2$, where $k \geq 2$ is the number of classes, and C_i - the label of i-th class.

Classification and machine learning

A classifier is developed based on the set of vectors $\mathbf{v_i} \in \mathbb{R}^S$ having their class labels $\tilde{f}(\mathbf{v_i})$ available i.e. *the training set*.

The labels needed in training data can be provided by an expert or based on observations.

The objective is to develop a function $f()$ that assigns a class to any vector from the domain of interest. ML methods make this process possible. The only requirement is to provide a sufficiently large training data set.

A classifier in batch approach is developed based on the fixed set of training instances. In online learning, we will use a sequence of labelled examples $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k = \{(\mathbf{x}_k, y_k)\}$ received so far to train and incrementally update a model, whereas $\mathbf{x}_k$ is a vector of input feature values, y_k is a true label.

Batch training revisited

Batch training: decision trees

Decision tree is composed of decision nodes, leaves and edges.

Every tree encodes the algorithm of mapping input vector $\mathbf{x} \in \mathbb{R}^S$ to class i.e. defines a mapping function $f()$.

One of the most popular algorithms of constructing decision trees is CART algorithm proposed by Leo Breiman in [9].

When a decision tree is built with CART algorithm, first root is established. Then, in every step a condition to partition input space based on the value of a single variable is verified. The best condition is selected. When stop condition is reached, such as a minimum number of patterns linked to a node, the process is stopped, and a leaf is created.

Once a tree is constructed, it can be used to classify new vectors. In each node the feature of the vector $\mathbf{x} \in \mathbb{R}^S$ present in node condition is investigated and left or right subtree is selected. Finally, a leaf is reached and class linked to this leaf is returned as a class $f(\mathbf{x})$

CART tree for Iris data set

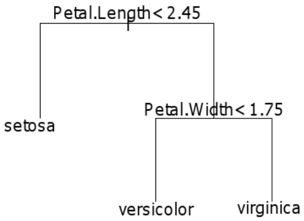


Figure: Decision tree developed for the Iris data set [4]

Tree interpretation

In each node the feature of the vector $\mathbf{x} \in \mathbb{R}^S$ present in node condition is investigated and left or right subtree is selected. Finally, a leaf is reached and class linked to this leaf is returned as a class $f(\mathbf{x})$. Hence, always a path from the root towards one of the leaves is followed to make a decision.

Parametrising construction of decision tree

One of key issues to decide upon when constructing a decision tree is when to stop, i.e. how detailed partitioning of input space is appropriate.

Typical stopping conditions [16]:

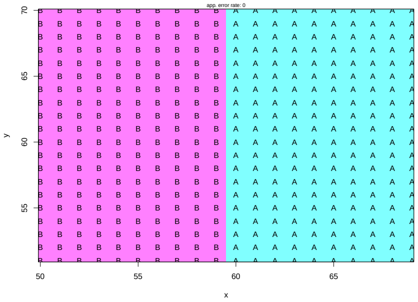
1. All training instances linked to the tree node belong to a single class.
2. The maximum tree depth has been reached.
3. The number of instances linked to a node is lower than a minimum number assumed to be sufficient to justify further space partitioning.
4. The best possible split generates improvement lower than a threshold value assumed to be sufficient.

Moreover, pruning is applied to reduce the risk of overfitting, i.e. some subtrees may be eliminated and replaced with nodes. This results in less complex trees.

Two linearly separable classes

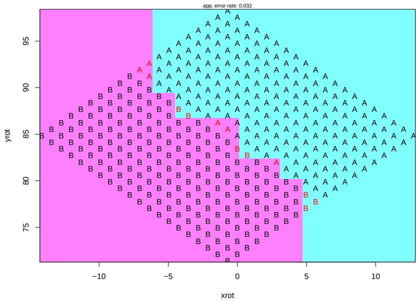
Let us assume that the data set includes two linearly separable classes and $\mathbf{v_i} = [x_i, y_i] \in \mathbb{R}^2$. For every $x \geq 50$, A class is assigned. All the remaining vectors belong to B class, irrespective of y value.

This classification task is trivial for a decision tree.



Two linearly separable classes

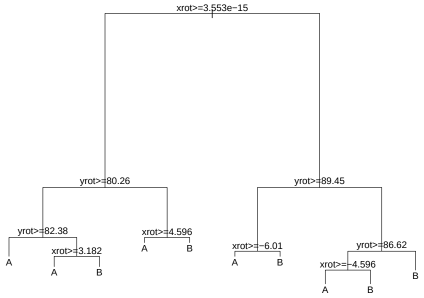
Let us rotate the previous input data by 45° and develop a decision tree for the rotated data. Will this change decision borders defined by a decision tree?



Discussion

The limitations of investigating a condition based on one variable only in every node are revealed.

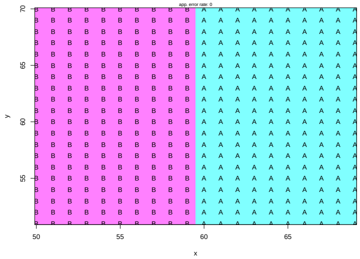
Linearly separable classes - decision tree



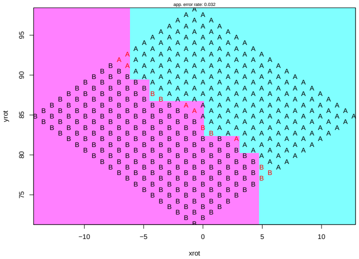
Interpretation

Even though, the classification task remains simple, the decision tree is no longer simple.

Linearly separable classes - analysis



(a) Before rotation



(b) After rotation

Discussion

This example illustrates the major impact of data transformation on classifier accuracy.

Classification vs. regression

We focus on classification trees i.e. decision trees used to assign a label y to each vector $\mathbf{x} = [x_1, \dots, x_n]$ of values of independent features. In the case of regression, a real value can be assigned.

Batch vs. online learning

CART method assumes that all training data is represented by a single data set with fixed content. Hence, each candidate split can be and is evaluated with the whole training data to see the impurity reduction it would yield, if introduced.
What if we have larger and larger training data set to work with?

First online learning method - Hoeffding trees

Hoeffding trees

One of the first stream mining method algorithms is known as Hoeffding tree (HT) algorithm proposed in [3]. The algorithm provides an incremental method of constructing tree-based classifiers.

VFDT (Very Fast Decision Tree learner) is a learning system implemented based on Hoeffding tree algorithm, described also in [3]. VFDT provides a stream mining implementation i.e.:

- it processes examples an instance after instance
- it does not store any examples in memory
- its memory requirements are proportional only to tree size
- it provides anytime prediction as the tree is incrementally updated after a single instance

Comparison with batch learning

HT develops trees similar to these learned with CART. However, compared to CART, memory footprint and model availability is improved.

Hoeffding tree algorithm - key symbols

Input: $\mathcal{S}_1, \mathcal{S}_2, ...; \mathcal{S}_k = \{(\mathbf{x}_k, y_k)\}$ - data stream, $A = dim(\mathbf{x}_k)$ - the number of attributes, all attributes are assumed to be nominal, $Y = \{1, \ldots, C\}$ - the set of classes, $y_k \in Y$, $G(\cdot)$ - a split evaluation function, $\delta \in (0, 1)$ - parameter such that $1 - \delta$ is the required probability that a correct attribute will be selected for splits at any node, X - set of discrete attributes

Data: l - leaf of a tree; $y(l)$ - label predicted by leaf l i.e. the most frequent class among instances sorted to the leaf; $n_{ijk}(l)$- the number of times an instance having class y_k and $j - th$ value of $i - th$ attribute, was sorted into leaf l so far (initialised with 0); $n_{ijk}(l) = 0$ for newly added leaf; $\overline{G}(X_i)$ - the average value of heuristic measure used to choose test attributes for X_i attribute

Result: h_i - Hoeffding tree model after processing i instances, $\mathcal{H}_1, \mathcal{H}_2, \ldots$ - the list of model predictions generated for individual instances,

Algorithm 0: Hoeffding tree method, based on [3] - part I.

Hoeffding tree algorithm - key solutions [3]

```
begin
    /* Initialise a tree with a leaf predicting most frequent class */
    $h_0$= a tree with a single leaf $l_1$;
    /* For every instance */
    for $s = 1, \ldots$ do
        $\mathcal{H}_s = h_{s-1}(\mathbf{x}_s)$;
        sort $\mathcal{S}_s$ into a leaf $l$ using $h_{i-1}$ tree;
        /* For every attribute */
        for $i = 1, \ldots, A$ do
            $n_{ijy_s}(l) = n_{ijy_s}(l) + 1$;
        end
        $y(l)$=majority class among the examples sorted so far into $l$;
        if Examples sorted to $l$ so far are not of the same class then
            /* Consider possible splits */
            calculate $\overline{G_l}(X_i)$ for every attribute $X_i$;
            $X_{i_1}, \ldots, X_{i_A}$=sortDescending($\overline{G_l}(X_i)$);
            if $\overline{G_l}(X_{i_1}) - \overline{G_l}(X_{i_2}) > \epsilon$ then
                Replace $l$ with a node splitting on $X_{i_1}$;
                for each branch of a split do
                    add a new leaf $l_m$;
                end
    end
end
```

Algorithm 1: Hoeffding tree algorithm, based on [3] - part II. All attributes are assumed to be discrete.

Split evaluation functions

$G(X_i)$ is the heuristic measure used to evaluate possible splits based on X_i values.

The objective is to use the measure to quantify possible benefits arising from splitting based on X_i attribute.

Functions mentioned by Domingos and Hulten in [3] are:

- *information gain*
- *Gini index*

These functions are also supported in recent implementations of the method e.g. in the `river` library.

Gini index

The Gini index is defined as follows [10]:

$I_{\mathrm{G}}(m) = \sum_{i=1}^{g} f_i(1 - f_i) = \sum_{i=1}^{g}(f_i - {f_i}^2) = \sum_{i=1}^{g} f_i - \sum_{i=1}^{g} {f_i}^2 = 1 - \sum_{i=1}^{C} {f_i}^2$, where C is the number of classes.

Note that the $G()$ function in CART method (i.e. in batch mode) is calculated for $S_m \subset S$, i.e. for the observations routed to the m leaf of a tree. Moreover, $f_i = \frac{n_{mi}}{n_m}, i = 1, \ldots, C$, where $n_m = card(S_m)$ - the number of records related to leaf m, while n_{mi} is the number of records $x_k \in S_m : y_k = i$, related to class i.

In Hoeffding trees, the $G()$ function is calculated based on $n_{ijk}(l)$ counters maintained for each leaf l.

Gini index as impurity measure

We expect different impurity measures to take the value of 0 when all related records belong to one class, and a maximum value, when $f_0 = f_1 = \ldots = f_C$. See [10] for a number of other impurity-based split criteria.

Hoeffding bound

Domingos and Hulten in [3] refer to Hoeffding bound described as follows. Let:

- r be a real-valued random variable with a range R
- n be the number of independent observations of this variable
- $\overline{r}$ be the mean value of this variable

Hoeffding bound states that with probability $1 - \delta$, the true mean value of r is at least $\overline{r} - \epsilon$, where:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

The use of Hoeffding bound in Hoeffding Tree algorithm

Hoeffding bound guarantees that with probability $1 - \delta$ true $\Delta G \geq \overline{G_l}(X_{i_1}) - \overline{G_l}(X_{i_2}) - \epsilon > 0$ i.e. X_{i_1} is the best attribute to perform the split with probability $1 - \delta$.

Key features of Hoeffding tree algorithm

The algorithms requires $O(AvYL)$ memory, where A - the number of attributes, v - the maximum number of values per attribute, Y - the number of classes, L - the number of leaves [3].

The memory allocation does not depend on the number of instances, which can be unbounded (subject to n_{ijk} counts)

The tree is potentially constantly grown by adding new splits. There is no way to replace a subtree with a new one.

VFDT implementation of Hoeffding tree

VFDT adds further simplifications aimed at increasing instance throughput. These include [3]: recalculating G only after user-specified minimum number of samples C_{min}, initialising HT with batch-trained model, eliminating keeping n_{ijk} counts for some of the attributes once they are found unpromising.

Extending Hoeffding trees with Naive Bayes

In every leaf of the original Hoeffding tree, actually Majority Class is implemented, i.e. the most frequently observed class in the leaf is used as a prediction.

Next, it has been shown that instead Naive Bayes can be implemented in each leaf, which yields even better predictions.

However, even later it was shown in [8] that Majority Class in tree leaves can be superior in some cases to Naive Bayes. Hence,

1. keeping both Majority Class and Naive Bayes in every leaf
2. and using the more accurate so far of these two for generating predictions

has been recommended, as it was shown to yield the best predictions [8].

Adding support for numeric attributes

The original method proposed by Domingos and Hulten does not support numeric attributes. Different ways of addressing this limitation have been proposed [1].

One of the best solutions is to maintain a separate Gaussian distribution per each class label, as proposed in [13].

The values of a numerical feature j are grouped into a predefined number of bins, such as 10. Next, when a split is attempted, impurity reduction is estimated by analysing distributions calculated for every class. In an ideal case, e.g. all instances with $x^j \leq 17$ are instances of class A, while instances with $x^j \geq 19$ are of class B.

Hoeffding tree nowadays

The implementations of the Hoeffding tree method nowadays go beyond original proposals of Domingos and Hulten in [3]. As an example, the `GaussianNumericAttributeClassObserver` is by default used for numeric attributes in MOA.

Key configuration settings: Hoeffding trees in river

Parameter	Default	Description
grace_period	200	Number of instances a leaf should observe between split attempts.
max_depth	None	The maximum depth a tree can reach. If None, the tree will grow until the system recursion limit.
split_criterion	info_gain	Split criterion to use: 'gini' (Gini), 'info_gain' (Information Gain), 'hellinger' (Hellinger Distance).
delta	1e-07	Significance level to calculate the Hoeffding bound. The significance level is given by 1 - delta. Values closer to zero imply longer split decision delays.
leaf_prediction	nba	Prediction mechanism used at leafs: 'mc' (Majority Class), 'nb' (Naive Bayes), 'nba' (Naive Bayes Adaptive).
nb_threshold	0	Number of instances a leaf should observe before allowing Naive Bayes.
nominal_attributes	None	List of nominal attributes identifiers. If empty, assume that all numeric attributes should be treated as continuous.
splitter	None	The Splitter or Attribute Observer (AO) used to monitor class statistics of numeric features and perform splits. By default, tree.splitter.GaussianSplitter is used if splitter is None.
binary_split	False	If True, only allow binary splits.

Cited from: https://riverml.xyz/dev/api/tree/HoeffdingTreeClassifier/

Selected issues in stream mining

Key aspects: labels

Similarly to batch learning tasks, not all instances have to be labelled.

Hence, the following cases can be identified:

- learning with fully labelled data streams
- semi-supervised learning with partly labelled data streams. This includes learning with *initially labeled* data streams i.e. streams which include completely labeled examples only during some initial period [15].

Semi-supervised learning for data streams

For many problems labels are costly and difficult to obtain. In some cases, they require off-line process to complement instance data available as a data stream [6]. Hence, semi-supervised stream mining methods attract particular attention [12].

Key aspects: delayed labels

Some (or even all) labels may be available with some latency only. This means, a stream $\mathcal{S}_1, \mathcal{S}_2, ...$ includes two types of tuples i.e.

$$\mathcal{S}_a = \begin{cases} \{(\mathbf{x}_k, ?)\} & \text{an instance arrives, but there is no label yet} \\ \{(\mathbf{x}_k, y_k)\} & \text{true label of the instance arrives} \end{cases}$$

Hence, the stream may include for instance:
$\mathcal{S}_a = \{(\mathbf{x}_k, ?)\}, \ldots, \mathcal{S}_b = \{(\mathbf{x}_j, y_j)\}, \ldots, \mathcal{S}_c = \{(\mathbf{x}_k, y_k)\}$. This means *delayed labels* occur.

Example: prediction of device failure is made at t_0 based on device features. Whether device actually fails (i.e. a real label) will be known at $t_1 > t_0$.

Consequences:

- prediction $\hat{y}_k$ made at the time $t(\mathcal{S}_a)$ of receiving an instance cannot be immediately evaluated
- this means *verification latency* [2] occurs

Further consequences of transition to stream mining

Some other challenges to address after transition from batch learning to stream mining are:

- *novel class detection* [11] as the assumption that the set of classes is fixed may be no longer true.
- *feature selection* methods reflecting the fact that the optimal subset of features may change with time
- *online feature selection* dealing with selection of features arriving over time [17]
- *instance selection* methods not requiring multiple iterations on the top of the same data
- *online discretisation* addressing the fact that the best discretisation method may change with time.

Data reduction techniques

A survey of data reduction techniques incl. feature and instance selection can be found in [14].

Hoeffding trees revisited

Hoeffding trees are fundamental for stream mining.

Still, the method does not consider some more advanced aspects.

First of all, the method includes no explicit drift-focused mechanisms such as drift detection and adaptation

Moreover, the training and evaluation process relies on test-then-train approach i.e. does not consider delayed labels

The algorithm assumes the set of classes is a priori known.

It depends (as for many stream mining methods) on its implementation, whether new classes can be introduced with time i.e. after processing some instances or not. Hence, as in many other cases, the ultimate capabilities of the method depend on its implementation.

Summary

Stream mining is an active field of research, with many issues to be addressed.

Various studies undertake different problems such as learning for concept-drifting data streams, with or without label latency, with fully or partly labelled data in a fully or semi-supervised manner.

One more aspect is the need for novel performance indicators and evaluation procedures matching stream mining setting.

Bridging the gap between batch learning and stream mining is yet another interesting challenge. As an example, in [7] we exploit a hybrid method combining batch-trained MLP networks and evolving stream mining models for public transport delay prediction.

References I

[1] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. `https://moa.cms.waikato.ac.nz/book/`.

[2] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[3] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *6th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[4] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[5] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, USA, 2012.

[6] Maciej Grzenda and Andres Bustillo. Semi-supervised roughness prediction with partly unlabeled vibration data streams. *Journal of Intelligent Manufacturing*, 30(2):933–945, Feb 2019.

References II

[7] Maciej Grzenda, Karolina Kwasiborska, and Tomasz Zaremba. Hybrid short term prediction to address limited timeliness of public transport data streams. *Neurocomputing*, 2019.

[8] Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. Stress-testing hoeffding trees. In Alípio Mário Jorge, Luís Torgo, Pavel Brazdil, Rui Camacho, and João Gama, editors, *Knowledge Discovery in Databases: PKDD 2005*, pages 495–502, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[9] Charles J. Stone R.A. Olshen Leo Breiman, Jerome Friedman. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[10] O. Maimon and L. Rokach. *Data mining and knowledge discovery handbook*, volume 14. Springer, 2010.

[11] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE TKDE*, 23(6):859–874, 2011.

References III

[12] Mohammad M. Masud, Clay Woolam, Jing Gao, Latifur Khan, Jiawei Han, Kevin W. Hamlen, and Nikunj C. Oza. Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowl. Inf. Syst.*, 33(1):213–244, 2011.

[13] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. Handling numeric attributes in hoeffding trees. In *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'08, page 296–307, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] Sergio Ramrez-Gallego, Bartosz Krawczyk, Salvador Garca, Micha Woniak, and Francisco Herrera. A survey on data preprocessing for data stream mining. *Neurocomput.*, 239(C):39–57, May 2017.

[15] Jerzy Stefanowski and Dariusz Brzezinski. Stream classification. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 1–9. Springer US, Boston, MA, 2016.

[16] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

References IV

[17] Peng Zhou, Xuegang Hu, Peipei Li, and Xindong Wu. Online feature selection for high-dimensional class-imbalanced data. *Know.-Based Syst.*, 136(C):187–199, November 2017.