Back to the Roots: Decision Trees within Modern Database Systems

Lena Wolfschmidt[1], Maximilian E. Schüle[2]

Abstract: HUK-COBURG relies on decision trees to classify e-mails into topics. As the data is stored within relational database systems, this paper investigates the performance improvement of in-database evaluation of decision trees. The paper first summarises the theory behind binary decision trees, then describes the implementation as SQL case-when statements. The evaluation shows the benefit of the in-database application of decision trees by eliminating expensive extraction-transform-load time. In the future, we intend to optimise the training as well as inference of decision trees in SQL to fully eliminate the need for data extraction.

1 Introduction

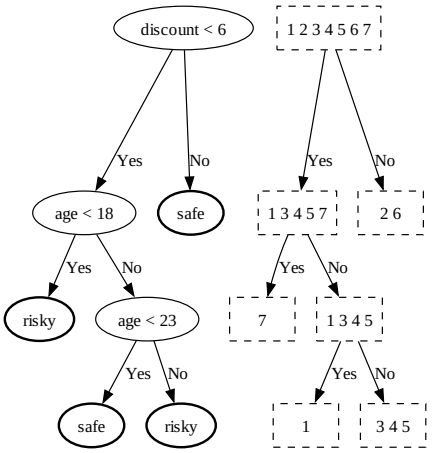
Machine learning (ML) is becoming increasingly important in the insurance industry for a wide range of applications. So far, only a few workflows at HUK-COBURG have been enhanced with ML, allowing the exploration of innovative approaches such as breaking up the traditional separation into data and application layers [Sc23; SKN22]. Until now, data storage and processing have been largely separated. The research question of this paper is to investigate whether and how machine learning algorithms can be integrated directly into the database to enable more efficient and direct data processing.

Supervised learning represents an important task in the field of data mining, especially when it comes to classification and regression [CFB99; SD01]. At present, HUK-COBURG can have e-mails from certain domains processed automatically. Incoming e-mails should therefore be classified according to a specific subject.

Example: A dataset consists of m features, A_1 to A_m, and a target feature C. The aim is to develop a model that predicts the value of C for each data item based on the values of the m features [CFB99]. Depending on the type of C, there are two different prediction methods: classification and regression. Classification means assigning a class from a set of given classes to a specific input. A regression task is a task that predicts a numerical value resulting from a specific input, e.g. predicting the risk class from a driver's age and current discount [Na11]. There are several models for classification, including Bayesian classifiers, neural networks, decision trees and decision tables [LL00]. Decision trees are considered intuitive because humans can understand why a data item is assigned to a certain class, as

[1] HUK-COBURG, 96444 Coburg, lena.wolfschmidt@huk-coburg.de
[2] University of Bamberg, An der Weberei 5, 96047 Bamberg, maximilian.schuele@uni-bamberg.de

doi:10.18420/BTW2025-37



id	age	discount	label	prediction
1	19	2	safe	safe
2	40	20	safe	safe
3	27	4	safe	risky
4	27	4	risky	risky
5	27	4	risky	risky
6	25	8	safe	safe
7	17	1	risky	risky

Fig. 1: Binary decision tree with exemplary evaluation.

the decisions are based on simple binary conditions. Ordoñez [Or22] gives an example of calculating summary matrices, such as the mean and variance of a data set, which can be used for further calculations or processes such as feature selection. Sattler and Dunemann use partial calculations in SQL to train decision trees. They use what are called CC tables, which are tables that contain statistics about the data set needed to make the best possible split at a decision point. This means that the entire data set does not have to be extracted from the database, but only the statistics mentioned above [SD01].

Figure 1 shows the structure of an example decision tree. This tree represents a simple classification for car insurance policies based on the age of the main driver and their no-claims discount. A policy is classified as *risky* if the insurance company expects high claims, otherwise *safe*. If the discount is higher than or equal to six, the policy is considered *safe*. If the discount is less than six, we take into account the driver's age. All drivers under the age of 18 fall under the *risky* policy. For drivers with a discount less than six and older than or equal to 18, we reconsider the age. For customers under the age of 23 (but still older than or equal to 18) the *safe* policy is applied, otherwise *risky*. With this approach, only two classes are distinguished, which also applies to our use case.

Performance comparisons show that for high-dimensional data, decision trees tend to perform better than neural networks or clustering algorithms. However, performance always depends on the specific use case and therefore cannot be universally assessed. Decision trees run the risk of overfitting, but there are techniques to minimise this risk, such as pruning or limiting the maximum depth [CFB99; SD01]. In the following sections we will explain how binary classification decision trees are trained and used for inference, and give a brief overview of the parameters available when constructing a decision tree.

2 Construction of Decision Trees

A decision tree follows the structure of a basic tree, i.e. it consists of nodes and edges. Trees, and therefore decision trees, are called binary decision trees if each node has two children. This results in a hierarchical sequence of attribute-threshold combinations for each path from root to leaves [KS08].

2.1 Training

The general procedure for training a decision tree consists of two main phases: the growth phase and the pruning phase [SD01]. Growth phase: The algorithm first creates a root node that represents the entire data set. At this root, the first partitioning takes place, for which a suitable condition has to be found. A greedy approach is used to subsequently create each node. An optimal result is achieved by the algorithm making the locally optimal choice at each stage of data partitioning. The condition is captured by using metrics such as entropy or Gini impurity to achieve the best possible partitioning of the data into homogeneous subgroups.

Depending on the specific algorithm, there are different approaches [Na11; Qu86]. By calculating the existing intermediate values for all features of all elements in the current node, all possible conditions and consequently the best partition are determined. This process is repeated iteratively, so that the tree grows and the dataset is divided into subsets. The algorithm stops when there are only items of one class left within a node, as this node then becomes a leaf node representing the class of its homogeneous items. The algorithm also stops if any split would lead to the same result, i.e. the items cannot be further divided into homogeneous classes. In this case, the leaf node represents the most frequent data item.

Depending on the implementation, there may be additional termination criteria, such as the maximum depth a node can reach in the path starting from the root. If a node has reached the maximum depth, no more conditions are searched for and the node represents exactly the class whose data items occur most frequently in the node as a leaf node [CFB99; KS08].

For example, ID3 (Iterative Dichotomizer 3) uses entropy as a measure to build decision trees [Qu86]. C4.5 is a further development of ID3 and, in addition to entropy, uses information gain to be less sensitive to features with a large number of values, as opposed to ID3 which is overly sensitive [Mi10]. In contrast, CART uses the Gini index instead of entropy [Br17]. Entropy, information gain and Gini index differ as follows: Gini impurity measures the frequency with which an element of the dataset is mislabelled when it is randomly labelled. Entropy is an information measure that indicates the disorder of the features within the groups. Information gain is based on entropy and indicates the expected decrease in entropy when a node is divided by a particular attribute. For our research, we use ID3 because it is simple and easy to understand using entropy.

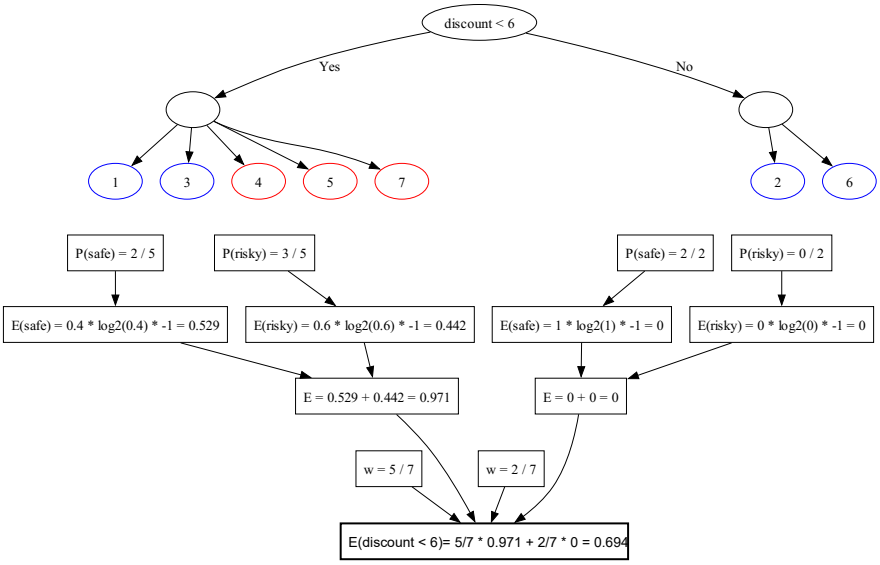


Fig. 2: Entropy for split by `discount < 6`: Leaves labeled risky (red) or safe (blue).

2.2 Entropy

Entropy is an information measure for finding the best split. As possible splits, we use all the attribute-intermediate value pairs as explained above. The entropy is calculated for each of these possible conditions; the condition with the lowest entropy is used as the optimal split.

$$\text{Node Entropy: } E(\text{split condition}) = w_{\text{left}} \cdot E_{\text{left}} + w_{right} \cdot E_{right} \tag{1}$$

$$\text{Weight: } w_{\text{child}} = \frac{\text{number of tuples in child}}{\text{number of tuples in both children}} \tag{2}$$

$$\text{Child Entropy: } E_{\text{child}} = E_{\text{child}}(c_1) + E_{\text{child}}(c_2) \tag{3}$$

$$E_{\text{child}}(c) = P_{\text{child}}(c) \cdot log_2(P_{\text{child}}(c)) \cdot -1 \tag{4}$$

$$\text{Probability: } P_{\text{child}}(c) = \frac{\text{number of tuples in child of class c}}{\text{number of all tuples in child}} \tag{5}$$

With child either left/right and the class c either c_1/c_2.

```
SELECT *, CASE WHEN discount >= 6 THEN 'safe'
               WHEN discount < 6 AND age < 18 THEN 'risky'
               WHEN discount < 6 AND age >= 18 AND age < 23 THEN 'safe'
               WHEN discount < 6 AND age >= 23 THEN 'risky'
               ELSE 'None' END AS PredictedClass
FROM data;
```

List. 1: Decision tree in SQL.

2.3 Inference

The inference in a decision tree describes the prediction process of the model in which a data item is assigned to a class for a classification task or a numerical value for regression. The resources required, such as time, are therefore the most important factor in the inference of a decision tree. Efficiency also plays a crucial role, especially when it comes to real-time applications [Ch22]. The basic inference process works as follows: For each individual data item, the process starts at the root node of the tree. The value of the attribute of the data item is compared with the threshold of the condition. The process is repeated iteratively, stopping when a leaf node is reached. The data item is then assigned to the class represented by that leaf node [KS08].

We will now explain how a tree is used to predict the class of data items from the values of their features. In our approach, a tree exists in the form of CASE-WHEN statements, each of which represents the conditions that lead to a leaf node in the tree. For example, the tree shown in Figure 1 as an SQL query is shown in List. 1. There is a statement for every path from the root to every leaf, but not all intermediate nodes are shown. We can shorten the representation for intermediate values: If *CASE WHEN* $n < x$ *AND* $n < y$ *THEN class1* holds and also $x < y$, then we can shorten the first expression to *CASE WHEN* $n < x$ *THEN class1*.

2.4 In-Database Training

In addition to using decision trees as case-when statements in SQL, we want to explore their performance for training. To do this, we calculate the metrics from Section 2.2 to determine the optimal split criterion at each level (`Path`). Table 1 shows the database table that calculates the entropy for each attribute value. The table is actually a cube that sums the entropy for each class and branch per node and attribute value (child entropy). Then a roll-up summarises the entropy first for both classes per child (determined by the `branch` yes/no), then for the whole node (both branches together weighted by the number of tuples contained).

Attribute	Path	LessThan	Branch	Class	Tuples	Entropy
...	...	...	...	...	...	...
discount	0	6	yes	safe	2	0.529
discount	0	6	yes	risky	3	0.442
discount	0	6	yes		5	0.971
discount	0	6	no	safe	2	0
discount	0	6	no	risky	0	0
discount	0	6	no		2	0
discount	0	6				0.694
...	...	...	...	...	...	...
discount	0	14				0.857
...	...	...	...	...	...	...

Tab. 1: Table containing the entropy per class/branch and the number of tuples for each attribute value.

3 Use Case

This section describes the specific use case we have chosen to evaluate the efficiency of our approach. We want to use machine learning to classify incoming e-mails at HUK-COBURG, an insurance company. The goal is to automatically recognise e-mails related to a contract. The following sections explain the use case in detail and describe the data preprocessing and feature generation methods used to train the model (cf. Figure 3).

3.1 Description

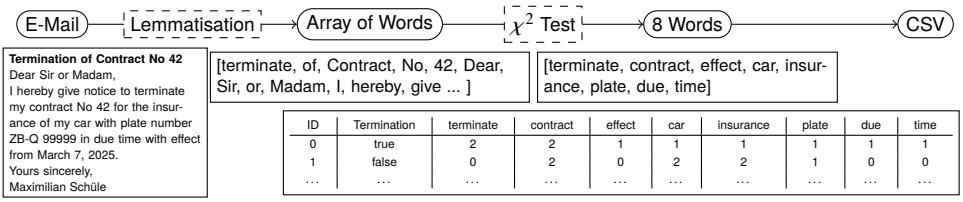
Every day, HUK-COBURG receives a large number of e-mails on various topics such as contract changes, contact requests or contract terminations. In order for the result to be valuable for improving customer service at HUK-Coburg, the success rate of recognising e-mails containing pure contract terminations must be significantly high.

A dataset of collected e-mails is available for training a tree. The dataset consists of 3615 pre-labelled e-mails with different labels, which can be grouped into two main categories: termination and no termination.

3.2 Feature Engineering

The data initially consists of a screenshot or image capture of each e-mail, which is converted to plain text documents using OCR. Metadata is lost at this point as no distinction is made between the type of text, i.e. sender, subject line and others. Despite this possibility, it is very rare for individual characters to be misrecognised.

Each e-mail is then split into its individual words, creating an array. We use lemmatisation to reduce inflected words to their root word. The word lists obtained for each e-mail are



ID	Termination	terminate	contract	effect	car	insurance	plate	due	time
0	true	2	2	1	1	1	1	1	1
1	false	0	2	0	2	2	1	0	0
...	...	...	...	...	...	...	...	...	...

Fig. 3: Use case: E-mails are first parsed and tokenised into an array of words from which the eight most significant are taken. The number of occurrences per e-mail and the label are stored as CSV file.

assigned to their respective classes. Using a chi-square test (χ^2), the ten words that best distinguish the two sets or classes are identified [Zh18]. The number ten is an empirical choice. Among these ten words, those that do not fit the context are manually eliminated, leaving us with eight keywords.

Based on the list of lemmatised words for each e-mail and the eight identified keywords, a CSV file is created, which serves as input for the tool. Each row represents an e-mail and consists of an assigned ID, the class of the e-mail, i.e. *termination* or *no termination*, and each of the eight identified keywords as columns, where the value of each column reflects how often that word occurs in the represented e-mail.

4 Evaluation

System: Ubuntu 22.04 LTS, Intel(R) Xeon(R) Gold 6248 CPU (20 CPU cores, two sockets, 2.50GHz.), 132 GB DDR4 RAM. Python 3.10.10 with sklearn, IBM DB2, DuckDB [RM19].
Data: eight attributes, binary label, 3615 tuples (= scale factor 1)

We simulate a production setup where the data is stored in the database system and data is extracted for training and inference. Therefore, we model a data pipeline in Python that extracts the data from the database system and evaluates a decision tree trained with sklearn. We first run the decision tree as case-when statements within SQL (without extraction). Then we extract the data from the IBM DB2 database (or DuckDB) to run the decision tree in Python. Figure 4 shows the throughput in terms of tuples per second as a function of the input size (in multiples of the original dataset). The more tuples, the higherh the data extraction overhead. In both setups—with either DB2 or DuckDB as the underlying database—the extraction time pays off from a scale factor of 10.

Figure 5 shows the runtime for training, including the time needed to compute the entropies. The current implementation is very experimental and could be optimised by avoiding unnecessary CTEs used so far. Thus, the database systems cannot excel, but demonstrate the ability to compute the metrics necessary for training.

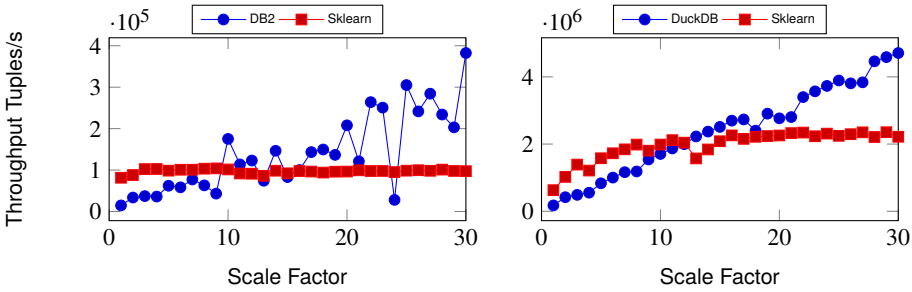


Fig. 4: Inference (scale factor refers to the relative size to the original data set).

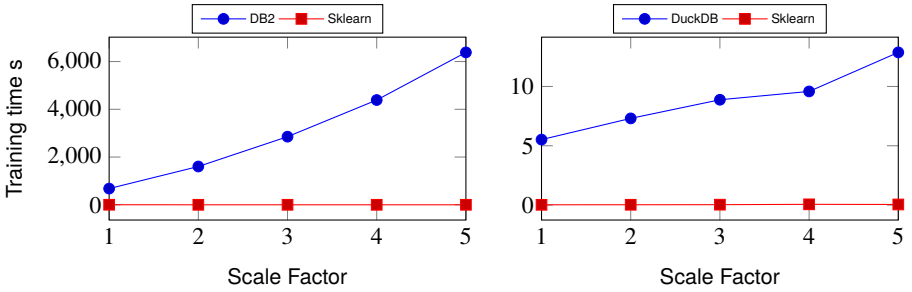


Fig. 5: Training (scale factor refers to the relative size to the original data set).

5 Conclusion

This paper investigated the performance gain when evaluating binary decision trees in SQL. We trained a decision tree on insurance data from HUK-COBURG to classify e-mails into topics. The evaluation showed the performance gain in terms of elimination of extraction time from the database system.

The use of DuckDB for data storage is currently being investigated. In general, the question is how to organise data so that it can be accessed quickly. The easiest way to improve the use of a modern database system like DuckDB for in-database machine learning would be to provide an abstraction between data storage and processing. This could be achieved by providing modules that make it easier for users to load data into DuckDB, for example with a focus on decision tree training. For large database server systems, it could also be considered to provide a Docker environment in which training could then be run. In the future, we plan to decode decision trees as bit vectors for storage within the database system, rather than as part of an SQL query.

Acknowledgement: We thank Claudia Shooter for proof-reading.

References

[Br17] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J.: Classification And Regression Trees. Routledge, 2017, ISBN: 9781315139470.

[CFB99] Chaudhuri, S.; Fayyad, U. M.; Bernhardt, J.: Scalable Classification over SQL Databases. In: ICDE. IEEE Computer Society, pp. 470–479, 1999.

[Ch22] Chen, K.-H.; Su, C.; Hakert, C.; Buschjäger, S.; Lee, C.-L.; Lee, J.-K.; Morik, K.; Chen, J.-J.: Efficient Realization of Decision Trees for Real-Time Inference. ACM Trans. Embed. Comput. Syst. 21/6, 68:1–68:26, 2022.

[KS08] Kingsford, C.; Salzberg, S. L.: What are decision trees? Nature biotechnology 26/9, pp. 1011–1013, 2008.

[LL00] Lu, H.; Liu, H.: Decision Tables: Scalable Classification Exploring RDBMS Capabilities. In: VLDB. Morgan Kaufmann, pp. 373–384, 2000.

[Mi10] Mitchell, T. M.: Machine learning. McGraw-Hill, New York, NY, 2010, ISBN: 0070428077.

[Na11] Navada, A.; Ansari, A. N.; Patil, S.; Sonkamble, B. A.: Overview of use of decision tree algorithms in machine learning. In: 2011 IEEE Control and System Graduate Research Colloquium. IEEE, pp. 37–42, 2011, ISBN: 978-1-4577-0337-9.

[Or22] Ordonez, C.: Scalable Parallel Machine Learning Computing a Summarization Matrix with SQL Queries. In: IEEE Big Data. IEEE, pp. 151–160, 2022.

[Qu86] Quinlan, J. R.: Induction of decision trees. Machine Learning 1/1, pp. 81–106, 1986, ISSN: 0885-6125.

[RM19] Raasveldt, M.; Mühleisen, H.: DuckDB: an Embeddable Analytical Database. In: SIGMOD Conference. ACM, pp. 1981–1984, 2019.

[Sc23] Schüle, M. E.; Scalerandi, L.; Kemper, A.; Neumann, T.: Blue Elephants Inspecting Pandas: Inspection and Execution of Machine Learning Pipelines in SQL. In: EDBT. OpenProceedings.org, pp. 40–52, 2023.

[SD01] Sattler, K.-U.; Dunemann, O.: SQL Database Primitives for Decision Tree Classifiers. In: CIKM. ACM, pp. 379–386, 2001.

[SKN22] Schüle, M. E.; Kemper, A.; Neumann, T.: Recursive SQL for Data Mining. In: SSDBM. ACM, 21:1–21:4, 2022.

[Zh18] Zhai, Y.; Song, W.; Liu, X.; Liu, L.; Zhao, X.: A Chi-Square Statistics Based Feature Selection Method in Text Classification. In: 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). IEEE, pp. 160–163, 2018, ISBN: 978-1-5386-6565-7.