Journal of Universal Computer Science, vol. 20, no. 9 (2014), 1174-1231
submitted: 9/4/12, accepted: 31/7/14, appeared: 1/9/14 © J.UCS

Decisions: Algebra, Implementation, and First Experiments

Antonina Danylenko, Jonas Lundberg, and Welf Löwe
(Linnaeus University, Software Technology Group,
35195 Växjö, Sweden
{antonina.danylenko, jonas.lundberg, welf.lowe}@lnu.se)

Abstract: Classification is a constitutive part in many different fields of Computer Science. There exist several approaches that capture and manipulate classification information in order to construct a specific classification model. These approaches are often tightly coupled to certain learning strategies, special data structures for capturing the models, and to how common problems, e.g. fragmentation, replication and model overfitting, are addressed.

In order to unify these different classification approaches, we define a *Decision Algebra* which defines models for classification as higher order decision functions abstracting from their implementations using decision trees (or similar), decision rules, decision tables, etc. Decision Algebra defines operations for learning, applying, storing, merging, approximating, and manipulating models for classification, along with some general algebraic laws regardless of the implementation used.

The Decision Algebra abstraction has several advantages. First, several useful Decision Algebra operations (e.g., learning and deciding) can be derived based on the implementation of a few core operations (including merging and approximating). Second, applications using classification can be defined regardless of the different approaches. Third, certain properties of Decision Algebra operations can be proved regardless of the actual implementation. For instance, we show that the merger of a series of probably accurate decision functions is even more accurate, which can be exploited for efficient and general online learning.

As a proof of the Decision Algebra concept, we compare decision trees with *decision graphs*, an efficient implementation of the Decision Algebra core operations, which capture classification models in a non-redundant way. Compared to classical decision tree implementations, decision graphs are 20% faster in learning and classification without accuracy loss and reduce memory consumption by 44%. This is the result of experiments on a number of standard benchmark data sets comparing accuracy, access time, and size of decision graphs and trees as constructed by the standard C4.5 algorithm.

Finally, in order to test our hypothesis about increased accuracy when merging decision functions, we merged a series of decision graphs constructed over the data sets. The result shows that on each step the accuracy of the merged decision graph increases with the final accuracy growth of up to 16%.

Key Words: decision algebra, decision function, decision graph, decision tree, classification

Category: D.3.1, I.2.4, I.2.6, I.2.m.

[1] This is a revised and extended version of the article which appeared in Proceedings of the 7th International Conference on Machine Learning and Data Mining (MLDM 2011), Lecture Notes in Computer Science, vol. 6871. Springer, pp. 31-45

1 Introduction

Classification is a constitutive part in different application domains of Computer Science, such as in information storage, retrieval and manipulation, knowledge management, artificial intelligence, image processing, data processing and visualization in social and behavioral sciences, software and hardware engineering, and in many others. In general, *classification* is used to come to a certain decision in a certain context. A *context* is a set of attribute values (e.g. a set of symptoms) that can be derived from a particular situation or state (e.g., a patient's health state), and a *decision* is an inference based on this context, often a class (e.g., the diagnosis of the patient). We refer to this type of classification information as to *decision information*.

Decision models represent the information necessary for classification, e.g., distributions, coefficients, probabilities. Decision models are captured in data structures like decision trees, support vectors, neural networks, etc. Decision models are often constructed automatically using machine learning. *Machine learning* processes, howsoever, a set of contexts and corresponding classes.

Learning is not an easy task; appropriate learning algorithms and decision models need to address several issues [King 1967, P.-N. Tan and Kumar 2005]:

- Accuracy, i.e. the ratio of correct classifications in all classifications, is an issue, especially, with missing or contradicting training data.
- Robustness, i.e., the accuracy of decision models learned with only a limited amount of decision information is a related issue. Learning needs to avoid decision model overfitting, i.e., basing classifications on statistically insignificant data.
- Scalability of learning and classifications, i.e., the time required for constructing and applying a decision model, resp., is another issue, since decision model size grows, in the worst case, exponentially with the number of context attributes. Data replication, i.e., redundancy in the decision models, adds to this problem.

Learning algorithms and corresponding decision models address these problems, e.g., by approximating decision models and by reducing redundancy in the information captured in decision models [Feng et al. 2010, Seredin et al. 2009, Ceci et al. 2007].

Selecting an appropriate decision model is a difficult task too, since no single model has been found superior to all others [Han and Kamber 2000]. Accuracy, robustness, and scalability are actually contradicting goals, which leads to trade-offs. For instance, accuracy of a decision model might require to exactly reflect all training data points while robustness and scalability may require to abstract from some of them (referred to as pruning). Therefore, different decision models may be appropriate in different application domains [Mitchell 1997].

Learning algorithms can be presented in a general framework as suggested, e.g., in [Rokach and Maimon 2008]. Also the data structures backing up decision models are generally well-known, along with efficient implementations thereof. However, when adapting learning algorithms and decision models to the needs (in accuracy, robustness, and scalability) of specific application domains, generality gets lost: decision models become incomparable and, hence, benchmarking difficult. Moreover, advances made in one domain are hardly propagated to others.

For instance, static program analysis uses *decision graphs*, a kind of decision model, for capturing context-sensitive analysis information (constructed by program analysis not learning) [Trapp 1999]. Precise program analysis is quite expensive in terms of time and memory consumption. Therefore, decision graphs optimize memory consumption by removing any redundancy and trade accuracy off against scalability. Decision graphs might even be beneficial in classification problems of other application domains with similar requirements, but it is hard to compare them with other, also highly specified decision models. Moreover, the approach of trading accuracy for scalability used in decision graphs might be applicable even to other decision models but, again, it is hard to transfer this approach before commonalities of the different models are understood.

Because of this variety of application domains with classification problems each coming with different learning algorithms, decision models, variants thereof, and tailored implementations – sometimes even with different notations – we consider it worthwhile to introduce a theoretical generalization, referred to as *Decision Algebra*. We separate interface and implementation of decision models making them (re-)usable as interchangeable black-box components. Several interface operations can be implemented on the abstract level using primitive operations which are specific to individual decision models. This does not exclude more efficient algorithms and data structures overriding the abstract implementations. Due to this generalization, insights can be gained at an abstract level or reused between different domains, paving the way for a deeper problem understanding. Some properties, for instance, can be proved on Decision Algebra level and hold for all its implementations.

In this paper, we mainly focus on generalizing tree-based decision models—including decision trees, decision graphs, and decision tables—and define Decision Algebra, a common abstraction of these models. Decision Algebra defines the operations *learn*, *decide*, and *prune*, based on abstract operations *restrict*, *merge*, *approximate*, *apply*, and *evert*. *Restrict* serves as a basic auxiliary operation. *Merge* enables different pruning and simple learning approaches. Different *approximate* implementations allow for further pruning approaches. *Apply* allows for symbolic computations with information stored in decision models in general. *Evert* can serve as the basis for different learning algorithms in which selecting an appropriate attribute order is essential. In fact, several existing approaches suggested for decision trees and tables so far come out as alternative implementation variants of the above operations. We further exploit the Decision Algebra abstraction by benchmarking tree-based models. Finally, we generalize tree-based models even further to also include other decision models such as, fore example, probability-based models.

In detail, [Section 2] discusses the results of a literature study backing up the observations discussed in this introduction. [Section 3] and [Section 4] introduce Decision Algebra, the theoretical framework that describes both tree-based decision models and the main operations required for classification. These two sections contain the main theoretical contribution of this article. [Section 5] introduces *decision graphs*, a specific decision model implementing Decision Algebra. Decision graphs capture training sample in a non-redundant way and perform the Decision Algebra operations efficiently. This is confirmed by experiments presented in [Section 6]. [Section 7] discusses related work. Finally, [Section 8] concludes the results and points out directions of future work.

2 Literature Study Motivating Decision Algebra

Despite the vast body of literature on applications of decision models in different application domains of Computer Science, no systematic study has been performed on the usage of decision models in different domains and the rationales behind their selection. In this section we perform such a study of the research papers published in the Journal of Universal Computer Science (J.UCS) from January 2010 till December 2012.

2.1 Objective

The objective is to study and summarize recent existing research in different application domains of Computer Science where decision models are used for classification and to:

A identify what decision models are typically used,

B assess the connection between the problem domains and the decision models used,

C retrieve the rationales for applying specific decision models in particular problem domains.

2.2 Method and Conduction of the Study

Our study comprises the primary steps of a systematic literature review as suggested by [Kitchenham and Charters 2007]. It is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest.

We searched for papers to be studied further in five steps:

1. We automatically searched—the actual search string is given below—for papers that use well-known or develop special decision models as a tool for solving other Computer Science research problems.

2. We manually inspected the papers found in Step 1 and selected those that we consider relevant: our primary objective is to understand reasoning and consequences of the choice of decision models applied to Computer Science research problems. Therefore, we excluded papers about theoretical aspects, surveys, roadmap papers, as well as papers that addressed non Computer Science problems including, e.g., e-learning, decisions making in society, classification of general methods. We also excluded short papers of 1 or 2 pages as well as papers mentioning decision models only briefly in related or future work. Finally, we excluded special issues.

3. We calculated matching frequencies of the search string in the papers found in Step 1.[1]

4. We assessed the accuracy of the automated search by calculating the F-score[2] based on precision P[3] and recall R[4] of the retrieved papers of Step 1 and the relevant papers analyzed in Step 2.

5. We adjusted the search string to increase the accuracy of the automated search.

These steps were repeated iteratively until the F-score did not further increased. The search string used to produce the final set of papers is: *("genetic algorithm", "bayesian", "bayes", "neural network", "neural networks", "clustering", "support vector", "support vectors", "reinforcement learning", "incremental learning", "collaborative filtering", "continuous learning", "learn continuously", "decision tree", "decision graph", "decision table", "dispatch table", "opinion mining", "hidden-markov-model", "hidden markov model", "utility function", "utility-based technique", "logistic regression", "linear regression", "BDD", "nearest neighbors") AND ("unsupervised learning", "supervised learning", "classifier", "decision model", "machine learning",*

[1] using PDF-XChange Viewer http://pdf-xchange-viewer.en.softonic.com/

[2] $F = (2PR)/(P + R)$

[3] $P = |\text{relevant papers}| \cap |\text{retrieved papers}|/|\text{retrieved papers}|$

[4] $R = |\text{relevant papers}| \cap |\text{retrieved papers}|/|\text{relevant papers}|$

"data mining", "pattern recognition", "artificial intelligence", "image processing", "decision tree", "genetic algorithm", "incremental learning", "classification", "linear regression", "BDD") with 111 (56) retrieved (relevant) papers, a precision (recall) of 0.51 (1) and an F-score of 0.67. We studied the 56 relevant out of a total of 430 papers.

For each paper the following data items were collected: (F1) a title of the paper and (F2) a year of the paper, both for documentation. (F3) a category of the paper as selected by the author(s) based on the list of topics pre-defined by JUCS; a paper can have more than one category. (F4) a short description of a problem addressed in the paper. (F5) a decision model that was used or implemented in the paper. It is the model that captures decision information required for learning, deciding or continuous learning. This could be, e.g. decision trees, Naïve Bayes (probabilistic model), support vector machines or neural networks (maximum-margin model) or others found in the paper. (F6) a short description of the rational for using this decision model. Such rationales can be given by a discussion in the paper, by formal proofs, or by some references justifying the choice. (F7) any relevant additional information, e.g. the purpose of using the decision model or a tool that was used as an implementation of the decision model. Every paper was read carefully; data was extracted in a form as described.

2.3 Results

We now discuss the study results based on the objectives A, B, and C.
Objective A: *Identify what decision models are typically used.*

Altogether around 30 different types of decision models were used in the 56 papers. We further classified them based on the type of data a decision model captures for the actual decision making:

DM1 Tree-based models capture a seach tree for decision making. For each attribute value, the search space is restricted which finally leads to a class. They include, e.g., decision trees, decision tables, decision rules, multi-variant binary decision diagrams, and decision graphs.

DM2 Probability-based models capture probabilities of attribute values belonging to different classes. They contain, e.g., Naïve Bayes classifiers, Bayesian networks, conditional-probability models, and hidden Markov models.

DM3 Maximum-margin models capture hyperplanes separating vectors of attribute values belonging to different classes. They include support vector machines (svm), artificial neural networks, and similar.

DM4 Vector-based models define vectors of attribute values as centroids of different classes. They are results of instance-based learning (such as k-nearest

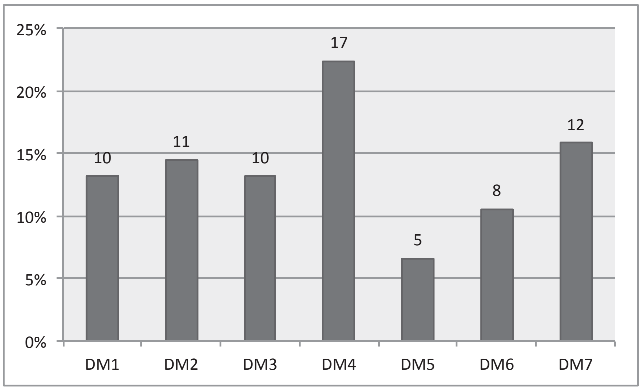


Figure 1: Decision models distribution

neighbours) and clustering algorithms (such as k-means, hierarchical clustering, distribution- and density-based clustering).

DM5 Regression models capture coefficients of certain function families that map attribute values to classes. They capture, e.g, the coefficients of linear and logistic functions derived from linear and logistic regression, resp.

DM6 Ad hoc solutions are self-developed decision models that do not fall into any of the above categories.

DM7 Related papers discuss the learning method, not the decision model. Hence, the decision model itself is unclear as the learning methods are not implying a particular model of any of the above categories. These generic learning methods include, e.g., genetic algorithms, collaborative filtering, population-based incremental learning, reinforcement learning, etc.

Figure 1 shows the categories of decision models introduced in the 56 relevant papers of the study. We are particularly interested in the first category (DM1). This contains decision models that are covered by our theoretical framework of Decision Algebra in Sections 3 and 4. Also, we should look at the first six categories of decision models (DM1 – DM6). These are decision models that can be generalized by Decision Algebras. We discuss this generalization in Section 3.6. The first (six) category (-ies) cover more than 13% (almost 84%) of the decision models used in the papers. In total we found 10 (73) decision models in DM1 (DM1 – DM6). Note that some papers introduce more than one model. Most popular are vector-based models (DM4) with 22% of the papers.

Around 16% of the decision models fall into the ”others” category (DM7). It cannot be excluded that there are decision models of one of the categories DM1 – DM6 even among those.

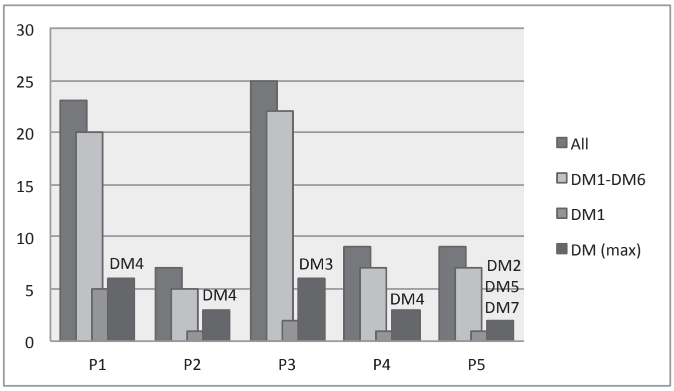


Figure 2: Decision models in problem domains

Objective B: *Assess the connection between the problem domains and the decision models used in these domains.*

Problem domains were derived from the data items F3 and F4, the category and the problem description of the papers. We define five problem domains addressed in the 56 relevant papers:

P1 Storage, retrieval and manipulation of information,

P2 Knowledge management,

P3 Applied mathematics including artificial intelligence, image processing, logics, and formal languages,

P4 Data processing and visualization in social and behavioral sciences, and

P5 Software and hardware engineering including software technology, programming, operating and control systems, and logic circuit design.

Figure 2 shows how the decision models are distributed over problem domains and decision model categories: the bars are the number of all decision models used in a problem domain, the number of decision model of categories DM1 – DM6, the number of decision models of DM1, and the number of decision model in the most popular category for each problem domain, respectively.

In all problem domains, the decision models of DM1 are used, and the decision models of DM1 – DM6 are dominating. Vector-based models (DM4) are most popular in most domains. Tree-based models (DM1) are not far behind in the respectively most popular models (DM4 and DM2, DM5, DM7, resp.) in the domains P1 and P5. However, there is no single decision model category dominating all problem domains or any particular one.

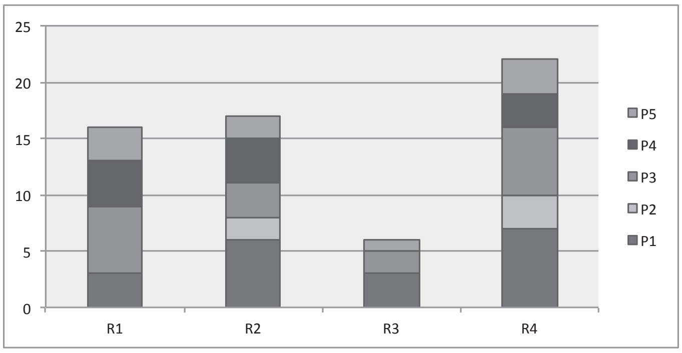


Figure 3: Distribution of rationales for choosing decision models in problem domains

Objective C: *Retrieve the rationales for applying specific decision models in particular problem domains.*

To answer this question we drew on data extracted from a short description of the rational for using the decision model (F6) and any relevant additional information (F7). The set of rationales derived from the papers can be generalized into four groups:

R1 references to the previous studies in this problem domain (choosing well-known decision model for this particular problem),

R2 references to requirements of the specific type of input or output data suggesting a decision model,

R3 references to requirements of the specific performance or representation properties suggesting a decision model,

R4 none of the above; choice in favor of a popular, commonly used, random decision model.

Figure 3 shows how the rationales are distributed among the papers and problem domains. Notice, that several papers use more than one rational to motivate the choice of a specific decision model. Around 21 papers (38% of all papers) do not specify any particular rationales for using one or the other decision model. In four out of five problem domains (P1, P2, P3, and P5) the majority of papers do not motivate the choice of a specific decision model. A reference to non-functional properties of a decision model (R3) is the least frequently used (in six papers alltogether) and does not at all occur in three out of five problem domains (P2, P4, and P5).

2.4 Conclusions

In this study we observed that

A decision models [DM1 - DM6] that we attempt to abstract with our theoretical framework Decision Algebra are popular (84%) and tree-based models [DM1] constitute an important category among them (13%),

B decision models that we abstract with Decision Algebra including tree-based models are popular in all (considered) Computer Science problem domains but, there is no single decision model dominating any problem domain, and

C the selection of a decision model is mostly ad-hoc.

As we have only looked at a limited set of papers of J.UCS (2010 - 2012), there is a threat to external validity of a generalization of these observations. However, the results indicate that (A) different decision models co-exist, they are (B) applicable across problem domains and (C) culture of comparing pros and cons of the decision models to select one could be further developed in general and in any individual problem domain (assessed). The reason is the difficulties in adapting, configuring or even re-implementing decision models for a specific problem domain which lead to problems in benchmarking their accuracy, robustness, and scalability. This motivates the present work: Decision Algebra allows using decision models as black-box components hiding the different kinds (categories) of decision models and their implementation details behind a common interface.

3 Decision Algebra

In this section we introduce a set of notions that characterize decision information and decision functions (which formalize decision information). Moreover, we also give a formal definition of the theoretical framework of Decision Algebra (DA).

We define the notion of *decision information* in [Section 3.1] and the notion of a *decision function* and its term representation in [Section 3.2]. Furthermore, in [Section 3.3] we show how to *learn* a decision function and how to *decide* using a decision function. For implementing the other operations in a number of variants, we define the auxiliary operations *restrict*, *merge*, *approximate*, *apply*, and *evert* in [Section 3.4]. In [Section 3.5] we formally define DA. Finally, in [Section 3.6] we define a generalization of DA.

3.1 Decision Information

Definition 1. A **decision tuple** (**a**, c) is a tuple that relates an actual context $\mathbf{a} \in \mathbf{A}$ with an actual decision $c \in C$, where $\mathbf{A}$ is a formal context and C is a formal decision.

The decision tuple can also be referred to as a *decision fact* or a *training instance*. Notice that we distinguish between an actual context and a formal context.

Definition 2. An **actual context** $\mathbf{a} = (a_1, \ldots, a_n)$ is a tuple of attribute values $a_i \in A_i$, where A_i is an attribute that corresponds to a property in a certain problem domain. A **formal context A** is the set of all actual contexts $\mathbf{a} = (a_1, \ldots, a_n)$ for all possible $a_i \in A_i$. Hence, it is the Cartesian product $\mathbf{A} = A_1 \times A_2 \ldots \times A_n$ over sets of possible values of attributes $A_1, \ldots, A_n$. Finally, an **actual decision** $c \in C$ is one out of a set of alternative decisions. A **formal decision** C is the set of all alternative decisions.

As an example of a formal context, consider weather attributes like "Temperature" and "Rain probability"; an actual context could be values like "hot" and "0.86". A corresponding formal decision could be "Should we go out?" with actual decisions "yes" or "no".

Definition 3. **Decision Information** is a set of decision tuples:
$DI = \{(\mathbf{a}_1, c_1), \ldots, (\mathbf{a}_n, c_n)\}$. Decision information is:

- **complete** if and only if: $\forall \mathbf{a} \in \mathbf{A} : (\mathbf{a}, c) \in DI$
- **non-contradictive** if and only if: $\forall (\mathbf{a}_i, c_i), (\mathbf{a}_j, c_j) \in DI : \mathbf{a}_i = \mathbf{a}_j \Rightarrow c_i = c_j$

Decision information can also be referred to as *dataset*, *training set* or *training sample*. Complete decision information contains decisions for all possible actual contexts within a given problem domain; in non-contradictive decision information there are no two tuples having the same actual context **a** leading to different decisions.

3.2 Decision Functions

A decision function is a representation of complete and non-contradictive decision information defined as:

Definition 4. A **decision function** df is a mapping of a formal context **A** (domain of *df*) to a formal decision C (co-domain of *df*):

$$df : A_1 \times \ldots \times A_n \to C.$$

Basically, a descision function is a mapping of attributes $A_1, \ldots, A_n$ to a classification decision C. We assume here that $A_1, \ldots, A_n$ and C are discrete (or categorical) domains; a discretization of continuous domains is outside the scope of this section but will be captured by our implementation, cf. [Section 6].

We denote by *DF* the set of all decision functions with the same signature $A_1 \times \ldots \times A_n \to C$. The *arity* of a decision function $df : A_1 \times \ldots \times A_n \to C$, denoted by *arity*(*df*), is the number n of attributes. If important we annotate the definition function with its arity n as a superfix, df^n.

A decision function can be represented in different ways by a specific representation, referred to as *decision model*.

Definition 5. A **decision model** is a representation of a decision function.

Example: A decision function $df : A_1 \times \ldots \times A_n \to C$ over finite domains A_i can be defined explicitly by all its tuples $(\mathbf{a}, c)$ with $\mathbf{a} \in A_1 \times \ldots \times A_n$ and $c \in C$. The tuples can be captured in a *decision table* which is a decision model with $n + 1$ columns, one for each attribute a_i and a final column for the actual decision c_j. Such a decision table would have $|A_1| \cdot \ldots \cdot |A_n|$ rows. Decision tables are one type of decision models but there exist alternative decision models that often differ from one domain to another as discussed in [Section 2.3].

For convenience of defining alternative decision models of and operations on decision functions, we will understand them as higher order functions where 0-ary (constant) decision functions $df^0 :\to C$ are the result of an 1-ary decision function $df^1 : A_1 \to (\to C)$ and so forth. A decision function is then represented by a *curried* function.

$$df^n : A_1 \to (A_2 \to (\ldots \to (A_n \to C)\ldots)$$

A *curried decision function* df^n takes one argument (attribute) of type A_1, and generates a new decision function of type $(A_2 \to (\ldots \to (A_n \to C)\ldots)$ which in its turn takes the next argument A_2 and yields new decision function $(A_3 \to \ldots)$. The final decision function will be of a type $\to C$, the actual decision. These curried functions can easily be represented as a *decision tree* or *decision term* $df^n = x^1(df_1^{n-1}, \ldots, df_{|A_1|}^{n-1})$ where the $|A_1|$-ary selection operator x^1 is applied to the arguments of A_1. There are $|A_1|$ result functions, one for each attribute value $a \in A_1$, which are $(n-1)$-ary decision functions:

$$df_{idx_i(a)}^{n-1} : A_2 \to \ldots \to A_n \to C$$

with $idx_i(a)$ being a bijective mapping of each attribute value $a \in A_i$ to a unique Natural index number. If necessary for distinction, we index a selection operator x^i with the index of the attribute A_i it switches on.

As a simple example, a 3-ary decision function df^3 with three boolean attributes A_1, A_2, A_3 can be represented as:

$$df^3 = x^1(x^2(x^3(1, 2), x^3(1, 2)), x^2(x^3(1, 2), x^3(2, 2))).$$

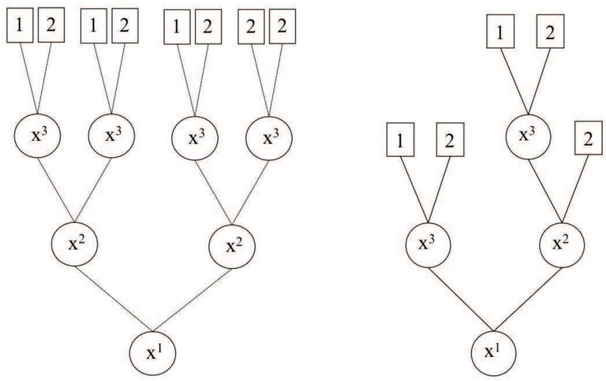


Figure 4: *left:* A straight forward tree representation of $df^3 = x^1(x^2(x^3(1,2), x^3(1,2)), x^2(x^3(1,2), x^3(2,2)))$. *Right:* a non-redundant tree representation of the same decision function.

The outermost selection operator x^1 gives the result $x^2(x^3(1,2), x^3(1,2))$ if A_1 is *true*, otherwise $x^2(x^3(1,2), x^3(2,2))$. In both cases the result is 2-ary decision function containing selection operators x^2 and x^3 that switch depending on the values of the remaining attributes A_2 and A_3, resp.

We refer to this decision model as decision trees since each decision function can easily be depicted as a tree. The left-hand side of [Fig. 4] shows a tree representation of the above mentioned decision function df^3. Each circle node represents a decision term with a selection operator x^n, each square leaf node corresponds to a certain decision $c \in C = \{1, 2\}$.

3.2.1 Redundancy and Equivalence

Definition 6. An n-ary decision function $df^n = x(df^{n-1}, \ldots, df^{n-1}) \equiv df^{n-1}$ is **redundant** if all its sub-functions df^{n-1} are equivalent, i.e., represent the same decision, and can therefore be replaced with this decision.

That is, a decision function df containing a redundant sub-function $df_r = x^r(df_j, \ldots, df_j)$, where each branch leads to the same decision df_j, can without any loss of information be rewritten as:

$$df = x(\ldots, x^r(df_j, \ldots, df_j), \ldots) \equiv df' = x(\ldots, df_j, \ldots)$$

In a decision tree representation, this corresponds to replacing a term with with root df_r by any of its (equivalent) sub-terms. The process of removing redundancy is called *redundancy elimination*. Our previous example decision

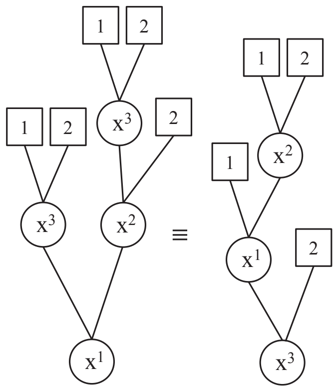


Figure 5: Equivalent decision functions: $df \equiv df'$

function df^3 contains two redundant sub-functions $df^1 = x^3(2,2) \equiv 2$ and $df^2 = x^2(x^3(1,2), x^3(1,2)) \equiv x^3(1,2)$, and can therefore be rewritten as:

$$df^3 = x^1(x^2(x^3(1,2), x^3(1,2)), x^2(x^3(1,2), x^3(2,2))) = x^1(x^3(1,2), x^2(x^3(1,2), 2))$$

The right-hand side of [Fig. 4] shows a non-redundant representations of df^3. Notice, that for df^3 the first child of x^1 refers to x^3 due to redundancy elimination applied to $x^2(x^3(1,2), x^3(1,2))$, where two children of x^2 correspond to $x^3(1,2)$.

Definition 7. Two decision functions df and df' are **equivalent**, denoted by $df \equiv df'$, if they give the same decisions c for the same attribute values $\mathbf{a}$ disregarding permutations of attribute values.

As an example, Figure 5 depicts two equivalent decision functions since for each set of actual boolean attributes $(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3$ they give the same decision $c_j \in C$.

3.3 Learning and Deciding of Decision Functions

Capturing decision information is an important part of the learning decision functions from training sets. The training set may be *incomplete*, i.e., it does not contain a decision for all possible combinations of attribute values, or *contradictive*, i.e., it contains different decisions for the same combination of attribute values. The latter requires a generalization of the decision terms as introduced before. Let C be the co-domain of a decision function (a finite set of discrete decisions) and define $d(C) = \{(c, n) \mid c \in C, n \in \mathrm{N}\}$ (with N the Natural numbers) a discrete *distribution* over C, i.e., a total mapping of the elements of C to their frequencies (or some weights). We denote the set of all possible distributions over C by $D(C)$. For capturing decision information, which can be seen as a simple learning, we replace 0-ary (constant) decision functions $df^0 :\rightarrow C$ with 0-ary decision distribution functions $df^0 :\rightarrow D(C)$. For each tuple $(\mathbf{a}, c)$ in a training dataset, we update $d(C)$ of the corresponding leaf(s) by incrementing the frequency of c in $d(C)$ by one.

3.3.1 Deciding

Deciding means to come to a unique decision c for a given attribute tuple $\mathbf{a}$ using a decision function df. Therefore, we evaluate df for $\mathbf{a}$ and then select the *mode* element of the resulting distribution, i.e., the most frequently occurring element in the distribution: let $df(\mathbf{a}) = d(C)$ then $decide(df(\mathbf{a})) = mode(d(C))$:

$$\begin{aligned}
df^n &: A_1 \rightarrow \ldots \rightarrow A_n \rightarrow D(C)\\
mode &: D(C) \rightarrow C\\
eval &: (A_1 \rightarrow \ldots \rightarrow A_n \rightarrow D(C)) \times A_1 \rightarrow (A_2 \rightarrow \ldots \rightarrow A_n \rightarrow D(C))\\
decide &: (A_1 \rightarrow \ldots \rightarrow A_n \rightarrow D(C)) \times (A_1 \times \cdots \times A_n) \rightarrow C\\
decide(df^n, \mathbf{a}) &:= decide(eval(df^n, a_1), \mathbf{a'}) \quad \mathbf{a} = (a_1, \mathbf{a'})\\
decide(df^0, _) &:= mode \circ df^0
\end{aligned}$$

where "_" corresponds to an empty attribute. Obviously, we can pre-compute *decide* if the learning phase precedes and is not interleaved with the decision phase. That is, we apply the *mode*-function on each of the distributions learned for a *df*-function. This saves space and decision time but loses information captured in the decision distributions.

3.3.2 Learning

Learning is the construction of a decision function from a decision information. It is actually independent of possible decision function implementations. Different learning algorithms exist [Moshkov 1997]. As one example, [Algorithm 1] presents (a sketch of) the commonly used C4.5 algorithm [Quinlan 1993] used in the experiments presented in [Section 6].

The algorithm works by recursively selecting the best attribute to split the training set (lines 6-10) and expanding the terms of the decision function (lines 14-17) until the stopping criteria are met (lines 3,11). Every decision function is created only when each corresponding dataset DS_v is processed (line 16).

3.4 Auxiliary Decision Function Operations

We now motivate and define a set of auxiliary operations on decision functions: *restrict*, *merge*, *approximate*, *apply*, and *evert*.

Algorithm 1 Learning

Dataset $DS \rightarrow$ decision function df

compute *distribution* of classes $D(C)$ in DS;
compute *error* of just selecting the *mode* of $D(C)$ as the decision;
if *error* acceptable **then**
  return $df\colon \rightarrow D(C)$
end if
let A - set of the dataset attributes;
for all $A_i \in A$ **do**
  compute $gain(A_i)$
end for
choose the attribute A_i with $max\ gain(A_i)$
if *gain* not acceptable **then**
  return $df\colon \rightarrow D(C)$
end if
for all $v \in A_i$ **do**
  let $DS_v \leftarrow \{(\mathbf{a}, c) \in DS | A_i = v\}$
  recursively construct df_v for a data subset DS_v
end for
return df, where $\forall v \in A_i, df(v) = df_v$

3.4.1 Approximating and Merging Decision Functions

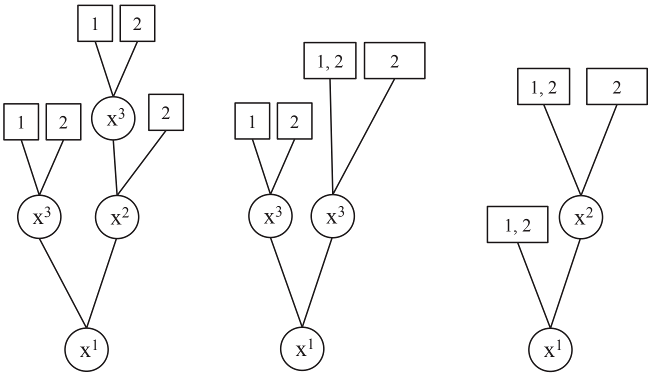
Pre-computing *decide* after learning saves representation space without sacrificing decision accuracy. Alternatively, space can be traded off against information accuracy if we *approximate* a decision by ignoring one attribute and *merge* the different decisions functions of alternative values of that attribute. For defining approximations and mergers of decision functions, we first define restriction of an n-ary decision function df to the kth value of the ith attribute, referred to as *restrict* operation and denoted by $df^n|_{i:k}$, as a new $(n-1)$-ary decision function where the ith attribute is bound to the value $a \in A_i$ with index $k = idx_i(a)$. For example, for $df^3 = x^1(x^3(1,2), x^2(x^3(1,2),2))$ we have

$$df^3|_{1:2} = x^2(x^3(1,2),2) \quad \textit{and} \quad df^3|_{3:1} = x^1(1, x^2(1,2))$$

The merge operation $\sqcup$ of two distributions $d(C)$ and $d'(C)$ is defined as:

$$\begin{gathered} \sqcup : D(C) \times D(C) \rightarrow D(C) \\ d(C) \sqcup d'(\mathrm{C}) = \{(c, w + w') | (c, w) \in d(C), (c, w') \in d'(\mathrm{C})\} \end{gathered}$$

Here $(c, w) \in d(C)$ denotes that decision c is supported to a degree w by $d(C)$, corresponding to the frequency w of decision c in $d(C)$. The $+$ operator on



(a) df^3 (b) $approx(2, df^3)$ (c) k-$approx(2, df^3)$

Figure 6: Approximation and k-approximation of $x^1(x^3(1, 2), x^2(x^3(1, 4), 2))$

frequencies w is the plus of Natural numbers. We consider $\sqcup$ of two distributions as the base case of our merge operation of decision functions. We can recursively define this merge $\sqcup$ of two decision functions:

$$df, df' : A_1 \times \ldots \times A_n \rightarrow D(C)$$
$$df \sqcup df' = x^1(df|_{1:1} \sqcup df'|_{1:1}, \ldots, df|_{1:k} \sqcup df'|_{1:k})$$

where $k = |A_1|$. Note that $df|_{1:i}$ and $df'|_{1:i}$ are $(n-1)$-ary functions. Hence, we reduce the $\sqcup$-definition of decision functions eventually to $\sqcup$ on distributions (0-ary decision functions, base case).

We approximate a decision function df by ignoring an attribute A_i using $\sqcup$:

$$\begin{aligned} approx(i) : &(A_1 \rightarrow \ldots \rightarrow A_n \rightarrow D(C)) \rightarrow \\ &(A_1 \rightarrow \ldots \rightarrow A_{i-1} \rightarrow A_{i+1} \rightarrow \ldots \rightarrow A_n \rightarrow D(C)) \\ approx(i, df) = &\bigsqcup_{a \in A_i} df|_{i:idx_i(a)} \end{aligned}$$

In fact, the decision function df can be approximated by ignoring any number $i \leq n$ of attributes, where n is the $arity(df)$.

Based on the above operation, it is possible to derive another approximation operation called *k-approx*, which represents an approximation of a decision function df of all attributes indexed i starting with a certain attribute index $k < i \leq n$. $k - approx$ is a repeated *approx* operation starting from the leaves and proceeding to a certain k-index of a decision function df. This operation can be formalized as follows:

$$\begin{aligned}
k - approx(k) &: (A_1 \to \ldots \to A_k \to \ldots \to A_n \to D(C)) \to \\
& \quad (A_1 \to \ldots \to A_k \to D(C)) \\
k - approx(k, df) &= approx(i, df_i), \forall i > k
\end{aligned}$$

Figure 6 shows an example of approximation and k-approximation of decision function $df^3 = x^1(x^3(1,2), x^2(x^3(1,2),2))$. This particular approximation will also be used in our experiment section.

We can use $\sqcup$ to formalize a simple learning algorithm: a neutral element $\perp \in C$ is a default class representing "don't know", and learning starts without any knowledge, i.e., with the initial decision function $df \equiv \perp$. Each tuple $(\mathbf{a}, c)$ in the training set corresponds to a decision function:

$$df'(\mathbf{b}) = \begin{cases} c & \text{if } \mathbf{b} = \mathbf{a} \\ \perp & \text{otherwise} \end{cases}$$

For each df' corresponding to a tuple of the dataset, learning incrementally sets $df := \sqcup(df, df')$.

3.4.2 Applying Functions to Decision Functions

We can *apply* functions g, defined on the co-domain of a decision function, to the leaves of this decision function. In [Section 3.4.1] we applied *mode* to the leaf distributions of a decision function and $\sqcup$ of two distributions to the corresponding leaves of two decision functions. These examples are just useful special cases of applying general functions to decision functions. We define such a general *apply* of arbitrary k-ary functions g to k-tuples of decision functions:

$$\begin{aligned}
g &: C_1 \times \ldots \times C_k \to C \\
apply(g, c_1 \ldots c_k) &= g(c_1 \ldots c_k) \\
df_i &: A_1 \to \ldots \to A_n \to C_i \\
apply(g, df_1 \ldots df_k) &= x^1(apply(g, df_1|_{1:1} \ldots df_k|_{1:1}), \ldots, \\
& \quad apply(g, df_1|_{1:K} \ldots df_k|_{1:K}))
\end{aligned}$$

where $i \in [1 \ldots k]$, $K = |A_1|$. $apply(g, df_1 \ldots df_k)$ recursively applies g to the respective subtrees of the arguments and eventually evaluates it on the leaves; the result is a decision function over C. The special cases *decide* and *merge* discussed earlier in Sections [3.3.1] and [3.4.1], respectively, could be redefined as:

$$\begin{aligned}
decide(df, \mathbf{a}) &= eval(apply(mode, df), \mathbf{a}'), \quad \text{where} \\
eval(df, \mathbf{a}) &= eval(\ldots eval(eval(df, a_1), a_2), \ldots, a_n) \\
df_1 \sqcup df_2 &= apply(\sqcup, df_1, df_2)
\end{aligned}$$

In general all functions defined on the classes of decision functions could be lifted to the decision functions over these classes.

3.4.3 Everting Decision Functions

Everting alters the order in which attributes occur in the decision functions. This is, e.g., used in heuristics for saving space during learning. *Evert* is naturally defined as a generalization of the so-called *Shannon expansion* of OBDDs [Bryant 1986] over a decision function *df* with the *i*th attribute:

$$
\begin{aligned}
&evert(i) : (A_1 \to \ldots \to A_n \to D(C)) \to \\
&\qquad (A_i \to A_1 \to \ldots \to A_{i-1} \to A_{i+1} \to \ldots \to A_n \to D(C)) \\
&evert(i, df) = x^i(df|_{i:1}, df|_{i:2}, \ldots, df|_{i:k}) \\
&evert(i, df) \equiv df
\end{aligned}
$$

where $k = |A_i|$. The Shannon expansion creates a new decision function corresponding to a new decision tree but does not change the decisions. It is just a rewrite rule that can be used to reorder the attributes of a decision function, sometimes making the representation more compact. Hence, a decision function *df* is equivalent to $evert(i, df)$ (equivalence was discussed in [Section 5]). For example, for $df^3 = x^1(x^3(1, 2), x^2(x^3(1, 2), 2))$ it holds:

$$
\begin{aligned}
df^3_1 &= evert(3, df^3) = x^3(x^1(1, x^2(1, 2)), x^1(2, x^2(2, 2))) \\
df^3_2 &= evert(2, df^3_1) = x^2(x^3(1, 2), x^3(x^1(1, 2), 2)) \\
df^3_3 &= evert(1, df^3_2) = df^3
\end{aligned}
$$

The everted decision function df^3_1 was previously depicted in Figure 5 in Section 3.2.

3.5 Decision Algebra

Finally, DA is a theoretical framework that generalizes over different decision models of decision functions by defining common properties and operations of all decision functions.

Definition 8. A **Decision Algebra** is a triple $DA =< DF, \Omega, R >$, where *DF* is a set of decision functions, Ω is a set of operations defined over *DF*, and *R* is an equivalence relation.

The equivalence relation *R* partitions the terms of the algebra *DA* into a number of equivalent classes. This relation enables us to determine that two decision functions which are syntactically distinct do, nevertheless, represent the same decisions.

3.6 Decision Algebra: Generalization

In the previous sections we introduced DA operations limited to tree-based models, such as decision trees and graphs. However, in Section 2 we observed several other decision models that are also frequently used for decision making, e.g., probability-based models, maximum-margin models, vector-based, and regression models. Therefore, the aim of this section is to introduce a theoretical generalization of DA over other decision models. In [Section 3.6.1] we present the operations over generalized decision functions, and in [Section 3.6.2] we define a constant decision function which serves as a base case scenario for defined operations.

In [Section 3.2] we introduced the notion of decision function that maps a formal context $\mathbf{A} = A_1 \dots A_n$ to a finite set of discrete decisions C. In practice, we have fuzzy decision bases, i.e., the decision function is rather modeled as a mapping to a more general co-domain $D(C)$ (a distribution over C). For instance, a general discrete distribution is used in: decision trees and graphs (represented by discrete distributions in the leaves), Naïve Bayes (represented by the result of multiplications of conditional probabilities) and support vector machines (represented by a distance between a vector and a maximum-margin [Cortes and Vapnik, 1995]). Therefore, DA can be defined in terms of a parameterized specification, with $\mathbf{A}$ and $D(C)$ as parameters, that provides a general representation of decision information as an abstract decision model along with a set of operations. Such a generalization requires decision models [see Section 2.3] to implement the core operations of DA with its defined pre- and post-conditions. The core operations are those that have to be implemented on each specific type of decision model. The derived operations are implemented based on the core operations, i.e., their implementation is given on the abstract specification level using the core operations.

3.6.1 Decision Function

Let us define general parameterized decision function $df : A_1 \times \cdots \times A_n \to D(C)$, $df \in DF[\mathbf{A}, D(C)]$ as a mapping of context attributes $\mathbf{A}$ to a formal domain $D(C)$. The constructor of such a distribution function $d \in D(C)$ takes a $k = |C|$ pairs (p, c) of frequencies of $p \in P$ and a corresponding classes $c \in C$, and returns a distribution $d \in D(C)$:

$$cons_D : (C \times P)^k \to D(C)$$

The operations over distributions are the following ($D_i \widehat{=} D(C_i)$):

$$
\begin{aligned}
g_D &: D_1 \times \cdots \times D_k \to D' \quad \text{any function to } \textit{apply}\text{, especially} \\
\sqcup_D{}^5 &: D \times \cdots \times D \to D \quad \text{merge function, or} \\
\equiv_D &: D \times D \to \textit{Boolean} \quad \text{equivalence function, or} \\
\textit{mode} &: D \to C \\
\textit{freq} &: D \times C \to P
\end{aligned}
$$

where *mode* returns a most-probable class, and *freq* returns a frequency for a certain class $c \in C$. The implementation of these two operations is straightforward:

$$
\begin{aligned}
\textit{mode}(\textit{cons}_D(c_1, p_1, \ldots, c_k, p_k)) &:= \arg\max_{c \in C} p(\textit{cons}_D(c_1, p_1, \ldots, c_k, p_k), c) \\
\textit{freq}(\textit{cons}_D(c_1, p_1, \ldots, c_k, p_k), c) &:= p_i \quad \text{where } c_i = c
\end{aligned}
$$

The general constructor of decision function $df \in DF[\mathbf{A}, D]$ reflects its higher-order representation discussed in [Section 3.2]. It takes attribute values of the first attribute $a_i \in A_1$ and corresponding decision functions $df^{n-1} \in DF[\mathbf{A}', D]$, where $\mathbf{A}' = A_2 \times \cdots \times A_n$. Notice, that the constructor also fulfills the non-redundancy property of the decision function:

$$
\begin{aligned}
&\textit{cons} : \underbrace{(A_1 \times DF[\mathbf{A}', D]) \times \cdots \times (A_1 \times DF[\mathbf{A}', D])}_{|A_1| \text{ times}} \to DF[\mathbf{A}, D] \\
&\textit{cons}\ (a_1, df^{n-1}, \ldots, a_{|A_1|}, df^{n-1}) \equiv df^{n-1}
\end{aligned}
$$

The only core operation of DA is *restrict* operation defined in [Section 3.4.1] which restricts an n-ary decision function $df^n \in DF[\mathbf{A}, D]$ to the $a \in A_i$ attribute value of the ith attribute, and as a result it returns a new $(n-1)$-ary decision function $df^{n-1} \in DF[\mathbf{A}', D]$ where the ith attribute is bound to the value A_i:

$$
\begin{aligned}
&\textit{restrict}(i) : DF[\mathbf{A}, D] \times A_i \to DF[\mathbf{A}', D], \quad \mathbf{A}' = A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\
&\textit{restrict}(1, \textit{cons}(a_1, df_1, \cdots, a_{|A_1|}, df_{|A_1|}), a) \equiv df_i \quad \text{where } a = a_i \text{ and } \quad i \in [1 \ldots n]
\end{aligned}
$$

The derived operations of DA are *eval*, *evert*, *decide*, *apply*, *equals*, *merge* and *approx*. All these operations were discussed in [Section 3] and concrete examples were given.

Eval evaluates the given decision function *df* with regard to a specific attribute value $a \in A$, as a result it returns another decision function with decreased set

[5] The discussion of a merge operation over distribution functions is outside the scope of this paper, as it can be implemented as shown in [Section 3.4.1] or it can refer to the theory of random variables[Lawrence Marple 1987].

of context attributes:

$$\begin{aligned} eval\ &:\ DF[\mathbf{A}, D] \times A \rightarrow DF[\mathbf{A}', D] \quad \mathbf{A}' = A_2 \times \cdots \times A_n \\ eval(df, a) &:= restrict(1, df, a) \end{aligned}$$

Evert changes the order in which attributes occur in the decision function. This operation was discussed in [Section 3.4]:

$$\begin{aligned} evert(i)\ &:\ DF[\mathbf{A}, D] \rightarrow DF[\mathbf{A}', D] \quad \mathbf{A}' = A_i \times A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\ &\quad i \in [1 \dots n] \\ evert(i, df) &:= cons(a_1, restrict(i, df, a_1), \dots, a_{|A_i|}, restrict(i, df, a_{|A_i|})) \\ &\quad a_1, \dots, a_{|A_i|} \in A_i \end{aligned}$$

Decide [see Section 3.3.1] is a process of applying a decision function *df* to a given actual context $\mathbf{a} \in \mathbf{A} = (A_1 \times \dots \times A_n)$ in order to determine a concrete decision $c \in C$:

$$\begin{aligned} decide\ &:\ DF[\mathbf{A}, D] \times \mathbf{A} \rightarrow C \\ decide(df, \mathbf{a}) &:= decide(eval(df, a_1), \mathbf{a}')^6 \quad \mathbf{a} = (a_1, \mathbf{a}') \end{aligned}$$

Apply [see Section 3.4.2] applies a function g_D on a set of decision functions $df_1, \dots, df_k \in DF$. We define such a general *apply* of arbitrary k-ary function g_D to k-tuples of decision functions:

$$\begin{aligned} apply\ &:(D \times D_2 \times \cdots \times D_k \rightarrow D') \times DF[\mathbf{A}, D] \times DF[\mathbf{A}, D_2] \times \cdots \times DF[\mathbf{A}, D_k] \rightarrow \\ &\rightarrow DF[\mathbf{A}, D'] \\ apply\ (g_D, df_1, \dots, df_k) &:= cons(a_1, apply(g_D, eval(df_1, a_1), \dots, eval(df_k, a_1)), \dots \\ &\quad a_{|A_1|}, apply(g_D, eval(df_1, a_{|A_1|}), \dots, eval(df_k, a_{|A_1|})))^6, \\ &\text{where} \quad a_1, \dots, a_{|A_1|} \in A_1 \end{aligned}$$

Equals is based on apply operation and checks equivalence ($\equiv_D$) of decisions for all attribute vectors $\mathbf{a} \in \mathbf{A}$:

$$\begin{aligned} \equiv\ &:\ DF[\mathbf{A}, D] \times DF[\mathbf{A}', D] \rightarrow Boolean \\ \equiv (df_1, df_2) &:= apply(\equiv_D, df_1, df_2) \equiv df_{True}{}^6 \end{aligned}$$

Merge is also based on *apply*; it applies a merge function $\sqcup_D$ on two decision functions:

$$\begin{aligned} merge\ &:\ DF[\mathbf{A}, D] \times DF[\mathbf{A}, D] \rightarrow DF[\mathbf{A}, D] \\ merge(df_1, \dots, df_k) &:= apply(\sqcup_D, df_1, \dots, df_k) \end{aligned}$$

[6] base case defined in $DF_\emptyset$

Approx (defined in [Section 3.4.1]) approximates the decision function by ignoring one (or more) attribute(s) A_i. It is based on *merge* and *restrict* operations:

$$\begin{aligned} approx(i) \;:\; & DF[\mathbf{A}, D] \to DF[\mathbf{A}', D] \quad \mathbf{A}' = A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\ & i \in [1 \ldots n] \\ approx(i, df) := & \; merge(restrict(i, df, a_1), \ldots, restrict(i, df, a_{|A_i|})) \end{aligned}$$

3.6.2 Constant Decision Function

Constant decision function serves as a base case scenario for DA operations. Basically it refers to a 0-ary decision functions $df^0 :\to D$, $df^0 \in DF_\emptyset[\{0\}, D]$ with zero-dimensional attribute vector space has the following constructor:

$$cons_\emptyset : D \to DF_\emptyset[\{0\}, D]$$

All other operations of DA have the same signatures as described above. Implementation of these operations in the base case scenario is straightforward:

$$\begin{aligned} eval(df) &:= df \\ decide(cons(d), \_) &:= class(d) \\ restrict(0, df, \_) &:= df \\ evert(0, df) &:= df \\ \equiv (df_1, df_2) &:= eval(df_1) \equiv_D eval(df_2) \\ apply(g_D, df_1, \ldots, df_k) &:= cons(g_D(eval(df_1), \ldots, eval(df_k))) \end{aligned}$$

where "_" corresponds to an empty value.

In order to create a specific decision model implementing DA interface one has to implement its core operations. In [Section 5] we show implementation of *restrict*, *apply*, and special form of approximation *k-approx* for tree-based models, i.e., decision trees and graphs. Additionally, we discuss further instantiation of DA towards Naïve Bayes classifier.

4 Accuracy versus Scalability

In [Section 3.4.1], we discussed, among other operations, the merge operation $\sqcup$ of decision functions which could be used in simple learning and in different approximation approaches. In this section, we discuss consequences of merging decision functions on their accuracy.

Intuitively, a decision function df_1 is *more accurate* than a decision function df_2 iff it more often gives the "right" classification, i.e., we define it based on some ground truth. However, this ground truth is, in general, not known to us usually due to the fact that the formal context **A** does not model all properties that have an inpact on a decision. For one and the same actual context $\mathbf{a} \in A$ different decisions $c \in C$ are possible following a probability distribution. This probalility distribution is usually not known to us either (just samples thereof with the training data sets). Still we can define a "more accurate" relation based on the idea of "right" classification from a theoretical "oracle".

Definition 9. Let us define $oracle_{\mathbf{a}} : C \rightarrow \mathbb{R}$ as the accurate classification probability distribution of a decision given a concrete context $\mathbf{a} \in \mathbf{A}$, and $oracle : \mathbf{A} \rightarrow D(C)$ as the corresponding accurate decision function with $\forall \mathbf{a} \in \mathbf{A} : oracle(\mathbf{a}) = oracle_{\mathbf{a}}$.

Given the *oracle* we can express the classification error in a concrete context **a** and a general error of df.

Definition 10. Let $df : \mathbf{A} \rightarrow D(C)$ be any decision function, and $error_{df}(\mathbf{a})$ of a distribution of decisions $d_a = df(\mathbf{a})$ in a concrete context $\mathbf{a} \in \mathbf{A}$ is defined as

$$error_{df}(\mathbf{a}) = \frac{1}{|C|} \sum_{c \in C} (oracle_{\mathbf{a}}(c) - d_a(c))^2$$

and a general $error_{df}$ of df is defined as

$$error_{df} = \frac{1}{|\mathbf{A}|} \sum_{\mathbf{a} \in \mathbf{A}} error_{df}(\mathbf{a}).$$

Finally,

Definition 11. The decision function $df_1 : \mathbf{A} \rightarrow D(C)$ is **more accurate** than another decision function $df_2 : \mathbf{A} \rightarrow D(C)$ iff df_1 is closer to the $oracle : \mathbf{A} \rightarrow D(C)$ accurate decision function than df_2:

$$error_{df_1} \leq error_{df_2}$$

4.1 Accuracy of Learning by Merging Decision Functions

What we wish to establish now is that merging decision functions gives a new decision function that is tendentially more accurate, i.e., reduces the error. As a consequence, we could define a simple and general learning approach based on merging decision functions. This merge-based learning approach would work as an online algorithm, as a new decision function can be learned howsoever (any learning algorithm would do) from new training data and merged with an existing decision function which thereby gets more accurate. Moreover, as merging is linear in the size of the decision functions involved, merge-based learning (dividing training data and learning individual decision functions that get merged) is faster than any hyper-linear learning (learning on the whole training data set). This way, we can save learning time and guarantee scalability even for large training data sets.

Unfortunately, we cannot guarantee that the merger of two decision functions gives a more accurate one in general. This is intuitively clear if we think of merging a very accurate decision function with another one that makes random decisions, which clearly cannot improve the already accurate decision function. We need to assume somewhat accurate input decision functions in order to expect an even more accurate output. We formalize the notion of "somewhat accurate" as probably accurate:

Definition 12. A decision functions $df : \mathbf{A} \rightarrow D(C)$ is **probably accurate** with respect to an accurate decision function *oracle* iff $\forall \mathbf{a} \in \mathbf{A} : df(\mathbf{a})$ is a random sample of $oracle_{\mathbf{a}}$.

The merger of probably accurate decision functions leads to a more accurate one.

Theorem 13. *Let $df_1, \ldots, df_n$ be a series of independently learned decision functions $df : \mathbf{A} \rightarrow D(C)$ that are probably accurate with respect to an accurate decision function $oracle : \mathbf{A} \rightarrow D(C)$. For large n, the merged decision function $df_1 \sqcup \ldots \sqcup df_n$ converges in probability to the oracle.*

Proof. Let $d_{a,1} = df_1(\mathbf{a}), \ldots, d_{a,n} = df_n(\mathbf{a})$ be a series of distributions of decision functions $df_1, \ldots, df_n$ in a concrete context $\mathbf{a}$, each a random sample of $oracle_{\mathbf{a}}$. As we prove the theorem independently for each concrete conext $\mathbf{a} \in \mathbf{A}$, we drop the index $\mathbf{a}$ from now on and assume an arbitrary but fixed actual context. For the merger $\sqcup$ of the series of distributions it holds

$$\begin{aligned} \forall c \in C : (d_1 \sqcup \ldots \sqcup d_n)(c) &= d_1(c) + \ldots + d_n(c) \text{ and} \\ |d_1 \sqcup \ldots \sqcup d_n| &= |d_1| + \ldots + |d_n| \end{aligned}$$

with $|d_i| = \sum_{c \in C} d_i(c)$ the size of a distribution, i.e., the sum of all frequencies $d_i(c)$ of all possible classes C. As each d_i is a random sample of *oracle*, it can be understood as a result of a series of $|d_i|$ random decisions:

$$\begin{aligned} d_i(c) &= [D_1 = c] + \ldots + [D_{|d_i|} = c] \text{ or} \\ \frac{d_i(c)}{|d_i|} &= \frac{[D_1 = c] + \ldots + [D_{|d_i|} = c]}{|d_i|} \end{aligned}$$

which is the unweighted average of a series of random decisions for which Corollary 16 applies, cf. Appendix A. Understanding $X_i = \frac{d_i(c)}{|d_i|}$ as random variables, it therefore holds that $E(X_i) = p_c$ and $Var(X_i) \leq p_c(1 - p_c)$.

Moreover, for the merged distribution it holds:

$$\begin{aligned}(d_1 \sqcup \ldots \sqcup d_n)(c) =& d_1(c) + \ldots + d_n(c)\\ =& \frac{d_1(c)}{|d_1|}|d_1| + \ldots + \frac{d_n(c)}{|d_n|}|d_n|\\ =& X_1|d_1| + \ldots + X_n|d_n|.\end{aligned}$$

Instead of the merged distribution $(d_1 \sqcup \ldots \sqcup d_n)$, we consider its corresponding probability distribution $(d_1 \sqcup \ldots \sqcup d_n)_N$ which is normalized with $N = \sum_{i=1}^{n} |d_i|$.[7]

$$(d_1 \sqcup \ldots \sqcup d_n)_N(c) = \frac{\sum_{i=1}^{n} X_i \times |d_i|}{\sum_{i=1}^{n} |d_i|}$$

This is the weighted average of a series of random variables X_i with weights $N_i = |d_i|$ for which Lemma 15 applies. Hence

$$\forall c \in C : \lim_{n\to\infty} P(|(d_1 \sqcup \ldots \sqcup d_n)_N(c) - p_c| \geq \epsilon) = 0 \text{ for any } \epsilon > 0.$$

This holds for any actual context $\mathbf{a} \in \mathbf{A}$ which concludes the proof. ∎

Recall that $decide(df) = apply(mode, df)$, i.e., applying *mode* to the underlying distributions. The result of *mode* is uneffected by normalization with any positive constant N, especially $N = |d|$, the size of the distribution:

$$\begin{aligned}mode(d) = c \Leftrightarrow & d(c) = \max_{c' \in C} d(c')\\ \Leftrightarrow & d_N(c) = \frac{d(c)}{N} = \max_{c' \in C} \frac{d(c')}{N}, N > 0.\end{aligned}$$

4.2 Accuracy of Approximation by Merging Decision Functions

The simple approximation operations *approx* and $k - approx$ are based on merge ($\sqcup$) like the simple learning approach discussed in [Section 3.4.1]. However, it should be intuitively clear that, although based on merge, these approximations do not converge (in probability) to an accurate decision function. This is not a contradiction to Theorem 13 as, in general, the theorem's prerequisites are not fulfilled by the merge operations used in approximations. It is sufficient to observe this for *approx* since $k - approx$ is just a repeated application of *approx*.

Recall the definition of $approx(i, df) = \bigsqcup_{a \in A_i} df|_{i:idx_i(a)}$. This means that we merge distributions $d_{\mathbf{a}} = df(\mathbf{a})$ and $d_{\mathbf{a}'} = df(\mathbf{a}')$ of *different* actual contexts $\mathbf{a}, \mathbf{a}' \in \mathbf{A}$. The actual context vectors $\mathbf{a}, \mathbf{a}'$ differ in position i and take different values $a, a' \in A_i$.

[7] Normalization: $d(c) = N \times p_c \Leftrightarrow d_N(c) = p_c$

In general, the formal context attribute A_i has an impact on the decision. Then, however, $oracle_{\mathbf{a}}$ and $oracle_{\mathbf{a}'}$ are *different* classification probability distributions. Still $d_{\mathbf{a}}$ and $d_{\mathbf{a}'}$ could be random samples of—hence probably accurate with respect to—the same accurate distribution, either $oracle_{\mathbf{a}}$ or $oracle_{\mathbf{a}'}$. But, this could only happen by chance and must not be assumed in general. Therefore, we expect to loose accuracy by merge based approximation operations like *approx* and $k - approx$. In general, these operations trade accuracy for memory space.

4.3 Scalability of Merging Decision Functions

Decision functions $df : \mathbf{A} \to C$ require memory $\Omega(|\mathbf{A}| \times |C|)$, i.e., they need to capture, for each actual context $\mathbf{a} \in \mathbf{A}$ and each class $c \in C$, the corresponding frequency $d_a(c)$. Hence, this is obviously a *lower bound* for the time complexity of any learning algorithm creating such decision functions.

Assuming that accessing the distribution of a decision function in an actual context $d_a = df(\mathbf{a})$ takes constant time, the time complexity of merged based learning is exactly this lower bound. Hence, other learning algorithms are at least as complex as merged based learning. Therefore, we can save learning time of more complex learning algorithms and guarantee scalability even for large training data sets by learning on smaller data sets and merge the resulting decision functions.

Moreover, learning on the smaller data sets can trivially be parallelized; using p processors it takes time $O(l(n/p))$ to learn p decision functions given a data set of size n and a complexity of $O(l(n))$ of the learning algorithm. Using a parallel sum technique, the merge operation of the resulting decision functions can be parallelized as well, requiring $O(\log(p))$ sequential merge operations of decision functions on p processors. Merging distributions takes time $O(|C|)$ on one processor or $O(1)$ on $|C|$ processors. Merging decision functions takes $O(|\mathbf{A}| \times |C|)$ one processor or $O(1)$ on $|\mathbf{A}| \times |C|$ processors. Altogether simple learning can be done in time $O(l(n/p) + \log(p \times |\mathbf{A}| \times |C|))$ on p processors or $O(l(n/p) + \log(p))$ on $p \times |\mathbf{A}| \times |C|$ processors.

5 Implementation

In this section we introduce an instantiation of DA towards tree-based decision models, referred to as Decision Graph Algbera (DGA). Decision functions of DGA is, thus, defined as decision graph functions (DFG). The implementation of DGA, referred to as *decision graph* (DG), avoids redundancies in the representation of a decision function. It serves as a concrete example of an instance of our DA framework and will be compared to decision trees, another implementation of DGA, in our experiments in [Section 6]. DGs combine Ordered Binary Decision Diagrams (OBDDs) [Bryant 1992, Bryant 1986] used to represent boolean functions, and χ-terms used to handle context-sensitive information in static program analysis [Trapp 1999].

As stated in [Section 3], every decision term representation of df can be seen as a tree $G = (N, E, r)$. The root node $r \in N$ corresponds to the selection operator x^1 of attribute A_1, returning the child $idx_1(a)$ for a given argument $a \in A_1$. The child is a sub-term representing the $(n-1)$-ary decision function $df(a)$. 0-ary decision functions df^0 are leaves labeled with elements of Y. Thus, the only one core operation for DGA is an operation of getting a child of the node:

$$child\ :\ DFG[\mathbf{A}, Y] \times A \rightarrow DFG[\mathbf{A}', Y] \quad \mathbf{A}' = A_2 \times \cdots \times A_n$$

While the rest of discussed operations are derived:

$$\begin{aligned} restrict(i, dfg, a) \;&:=\; cons(a_1, restrict(i, child(dfg, a_1), a), \ldots \\ &\quad\;\; \ldots, a_{|A_1|}, restrict(i, child(dfg, a_{|A_1|}), a)) \\ &\quad\;\; a \in A_i, \quad a_1, \cdots, a_{|A_1|} \in A_1, \quad i \in [2 \ldots n] \\ restrict(1, dfg, a) \;&:=\; child(dfg, a) \quad a \in A_1 \end{aligned}$$

We introduce DGs in [Section 5.1]; we discuss the *restrict* operation in [Section 5.1.1], *apply* in [Section 5.1.2] and the *k-approx* operation in [Section 5.1.3]. Moreover, in the last [Section 5.2] we discuss how our DA can be instantiated towards probability-based decision model such as Naïve Bayes classifier. Notice, that instantiations towards further decision models are possible too but are not in the scope of this paper.

5.1 Decision Graphs

The co-domain in DGs is represented as a class distribution $D(C)$ discussed in Section 3.3. Our implementation uses a *repository* that captures decision term representation of *df*. However, the repository guarantees that each term corresponding to a unique decision (sub-)function, reduced by redundancy elimination, cf. Section 3.2.1, is captured only once and gets a unique identifier. The children in the decision tree of *df* only refer to the unique identifier of the corresponding sub-terms. As a consequence, we never store two *equivalent*, cf. Section 3.2.1, decision (sub-) functions in the repository and we never make use of two equivalent sub-terms in the same decision function. Since two selection operators x and x' may point to the same children, our terms are represented by rooted directed acyclic graphs (instead of trees). Figure 7 shows the tree and graph representations of the simple decision function introduced in Section 3.

To help recognizing equivalent sub-terms we maintain a "normal" order of all nodes: If attribute A_i is chosen for splitting by a classification algorithm

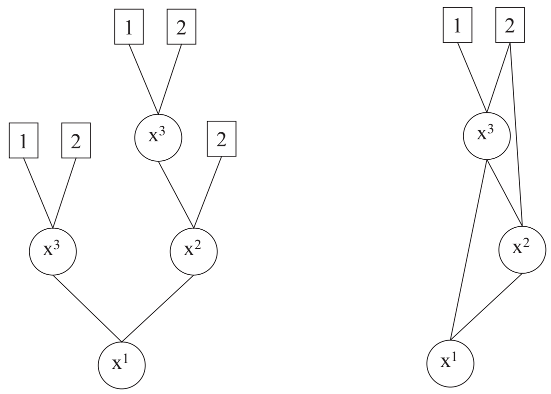


Figure 7: *Left:* A non-redundant decision tree representation of $df^3 = x^1(x^2(x^3(1,2), x^3(1,2)), x^2(x^3(1,2), x^3(2,2)))$. *Right:* a non-redundant decision graph of the same decision function.

before attribute A_j, then $i < j$ and $x^i < x^j$, and operator x^i occurs before operator x^j on all paths from root to the leaves in the corresponding decision graph.

All DA operations discussed in [Section 3] are implemented on DGs; DGs inherit all decision function properties, e.g., those discussed in [Section 4]. We discuss the implementation of *restric*, *apply* and *k-approx* in the next sections; other operations are implemented in a straightforward way.

5.1.1 Implementation of Restrict

In the previous [Section 3.6] we showed that many DA operations can be defined using core operation *restrict*. For DGA, *restrict* becomes a derived operation based on core operation *child*. Our implementation of *restrict* on DGs is presented in [Algorithm 2]. Basically, it recursively applies *restrict* operation over the children of the given graph node and returns a new decision term constructed from the results of the operation applied. Lines 1–6 handle the base cases where the decision terms are leaves (lines 1–3), or the attribute, which we restrict to, corresponds to the given decision term (lines 4–6). In the former case the algorithm returns the leaf. In the latter case the restriction is applied by getting a-th child $child(x^k, a)$ of the current decision term x^k, and then *restrict* is pushed further towards the leaves (since there can exist more then one decision term with the same corresponding attribute in the path due to the properties of the learning algorithm).

Lines 7–9 recursively apply *restrict* to the children of the decision term x^k. Note that $children(x^k)$ refer to the sub-terms of x^k, and $|children(x^k)|$ to the number

Algorithm 2 restrict($i \in Integer, x^k \in DG(D), a \in AttributeValue) \rightarrow DG(D)$

1: **if** $x^k \in D$ **then**
2:     **return** x^k
3: **end if**
4: **if** $k == i$ **then**
5:     **return** $restrict(i, child(x^k, a), a)$
6: **end if**
7: **for all** $x^j \in children(x^k)$ **do**
8:     $x^* = restrict(i, x^j, a)$
9: **end for**
10: **return** $x^k(x^*{}_1, \ldots, x^*{}_{|children(x^k)|})$

of sub-terms of x^k. Finally, in line 10, a new decision term is created if and only if it has not been created before.

5.1.2 Implementation of Apply

In [Section 3] we showed that many DA operations can be defined as special cases of the higher order function *apply*. Our implementation of *apply* on DGs is outlined in [Algorithm 3]. It pushes an *operation* to the leaves and applies it on them. Hence, the operations must be defined on the leaf values $D(C)$. The result is a new decision term recursively constructed from the results of the operation applied to the leaves.

Lines 1–3 handle the base case where decision terms are leaves and the operation is applicable directly. Lines 4–17 handle the case where one of the decision terms is a leaf. Lines 18–23 handle the case where the root selection operators, i.e., the first selection attributes, are identical. In all three cases (lines 9, 15, 22), a new decision term is created if and only if it has not been created before. Line 24 handles the case when the selection operators differ and both decision terms are not leaves. In this case we *evert* the second term x^j to k. An *apply* of *op* to the result leads to the case where the root selection operators are identical.

5.1.3 Implementation of Approximate

To save space or to avoid overfitting in a decision functions, pruning is applied and different pruning variants can easily be implemented using the *approx* operation. It was defined in [Section 3.4.1] and allows to ignore any attribute of a decision function. A simple pruning method, the *k-approximation* (a special implementation of approximation), is efficiently implemented with Algorithm 4.

Algorithm 3 apply($op \in D \times D \to D, x^k, x^j \in DG(D)) \to DG(D)$

if $x^k, x^j \in D$ **then**
    return $op(x^k, x^j)$
end if
if $x^k \in D \,||\, x^j \in D$ **then**
    if $x^k \in D$ **then**
        for all $x' \in children(x^j)$ **do**
            $x^* = apply(op, x^k, x')$
        end for
        return $x^j(x^*{}_1, \ldots, x^*{}_{|children(x^j)|})$
    end if
    if $x^j \in D$ **then**
        for all $x' \in children(x^k)$ **do**
            $x^* = apply(op, x^j, x')$
        end for
        return $x^k(x^*{}_1, \ldots, x^*{}_{|children(x^k)|})$
    end if
end if
if $k == j$ **then**
    for all $i \in [1 \ldots |children(x^k)|]$ **do**
        $x^* = apply(op, children(x^k)_i, children(x^j)_i)$
    end for
    return $x^j(x^*{}_1, \ldots, x^*{}_{|children(x^k)|})$
end if
return $apply(op, x^k, evert(x^j, k))$

The k-approximation of decision functions is easy to understand based on the tree representation: all subtrees of depth $\geq k$ from the root are replaced with the merger $\sqcup$ of their leaves. In particular it means that the distributions of the same classes is summarized. Recall from the standard tree representation that every leaf keeps the distribution for each class. Therefore, the result is a new decision function with depth $\leq k$. [Algorithm 4] finds the subtrees at depth k, [Algorithm. 5] then merges their respective leaves.

5.2 Naïve Bayes Classifier

As an example of how DA can be instantiated with decision models other than trees and graphs, we discuss Naïve Bayes classifiers. Naïve Bayes is a simple probabilistic classifier based on Bayesian statistics [Keogh and Pazzani 1999]. It

Algorithm 4 k-approx($k \in \mathrm{N}, x \in DG(D)) \rightarrow DG(D)$

if $k = 0$ **then**
  return new $collapse(x)$
end if
for all $x^i \in children(x)$ **do**
  $x_i^* = k\text{-}approx(k-1, x^i)$
end for
return new $x(x_1^*, \ldots, x^*_{|children(x)|})$

Algorithm 5 collapse($x \in DG(D)) \rightarrow D$

if $x \in D$ **then**
  return x
end if
$v \leftarrow \perp$
for all $x' \in children(x)$ **do**
  $v \leftarrow add(v, collapse(x'))$
end for
return new v

calculates conditional probabilities

$$\forall \mathbf{a} \in A, c \in C : P(C = c|\mathbf{A} = \mathbf{a}),$$

the probability of an actual class c given actual attribute values $\mathbf{a} \in \mathbf{A} = (A_1 \times \cdots \times A_n)$. Thereby, it naively assumes a conditional independence of the attributes from each other. Then class c is most probable for actual attribute values $\mathbf{a} = (a_1, \ldots, a_n)$ if

$$P(C = c) \times P(A_1 = a_1|C = c) \times \cdots \times P(A_n = a_n|C = c)$$

is maximum among all classes in C [Mitchell 1997, Keogh and Pazzani 1999]. Therefore, a decision model representing Naïve Bayes obviously needs to capture (or compute) the absolute probability distribution of each actual class $D(C)$, and the conditional probabilities of each actual attribute given each actual class:

$$\forall c \in C, \forall A_{i\in[1,n]}, \forall a \in A_i : P(A_i = a|C = c).$$

The constructor for decision function over Naïve Bayes classifier (DFNB), takes a $D(C)$ that represents the absolute probability over classes C and probability distribution functions ($PDF_i \widehat{=} PDF(A_i)$) for each attribute $A_i \in \mathbf{A}$, that represents the conditional class probability for each class value $c \in C$:

$$cons : (D(C) \times \underbrace{PDF_1 \times \cdots \times PDF_n \times \ldots \times PDF_1 \times \cdots \times PDF_n}_{k \;\; \text{times}})$$

where $k = |C|$. The constructor of such a probability distribution for each attribute A_i takes a $y = |A_i|$ number of pairs of a probability value $p \in P$ and a corresponding attribute value $a \in A_i$, and returns a PDF:

$$cons_{PDF} : (A_i \times P)^y \to PDF_i$$

where

$$\sum_{p_i \in P} p_i = 1 \quad \text{and} \quad p \in [0 \ldots 1]$$

Alternatively, PDF_i can be constructed from a parametric distribution, e.g., Normal distribution, with the respective parameters, e.g., μ and σ. The operations over PDF are following:

$$\begin{aligned} freq &: PDF \times A_i \to P \\ freq(cons_{PDF_i}((a_1, p_1), \ldots, (a_i, p_i), \ldots (a_y, p_y)), a_i) &\equiv p_i \end{aligned}$$

The core operations of DFNB are: getting the class distribution $D(C)$, and getting the PDF for a certain class value $c \in C$ of ith attribute, respectively:

$$\begin{aligned} df_c &: DFNB \to D(C) \\ pdf(i) &: DFNB \times C \to PDF_i \end{aligned}$$

The *restrict* operation multiplies the class frequencies with the conditional probabilities of an attribute value $a \in A_i$ of ith attribute of a respective class $c \in C$. It returns a new Naïve Bayes:

$$\begin{aligned} restrict(i) &: DF[\mathbf{A}, D] \times A_i \to DF[\mathbf{A}', D], \\ &\quad \mathbf{A}' = A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\ restrict(i, dfnb, a) &:= cons(cons_d(c_1, freq(df_c(dfnb), c_1) * freq(pdf(i, dfnb, c_1), a), \ldots \\ &\quad c_k, freq(df_c(dfnb), c_k) * freq(pdf(i, dfnb, c_k), a)), \\ &\quad pdf(1, dfnb, c_1), \ldots, pdf(i-1, dfnb, c_1), pdf(i+1, dfnb, c_1), \ldots \\ &\quad pdf(n, dfnb, c_1), \\ &\quad \ldots \\ &\quad pdf(1, dfnb, c_k), \ldots, pdf(i-1, dfnb, c_k), pdf(i+1, dfnb, c_k), \ldots, \\ &\quad pdf(n, dfnb, c_k)) \\ &\quad a \in A_i \end{aligned}$$

The *evert* operation simply changes the order of the PDFs of attribute values while putting the set of the i-th attribute first in the constructor:

$$\begin{aligned} evert(i) \ &: \ DF[\mathbf{A}, D] \to DF[\mathbf{A}', D] \quad \mathbf{A}' = A_i \times A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\ evert(i, dfnb) &:= cons(df_c(dfnb), pdf(i, dfnb, c_1), pdf(1, dfnb, c_1) \ldots, pdf(i-1, dfnb, c_1), \\ &\quad pdf(i+1, dfnb, c_1), \ldots, pdf(n, dfnb, c_1) \\ &\quad \ldots \\ &\quad pdf(i, dfnb, c_k), pdf(1, dfnb, c_k), \ldots, pdf(i-1, dfnb, c_k), \\ &\quad pdf(i+1, dfnb, c_k), \ldots, pdf(n, dfnb, c_k)) \end{aligned}$$

The *approx* operation constructs a decision function that ignores attribute A_i by simply "forgetting" the PDFs of this attribute:

$$\begin{aligned} approx(i) \ &: \ DF[\mathbf{A}, D] \to DF[\mathbf{A}', D] \quad \mathbf{A}' = A_1 \times \cdots \times A_{i-1} \times A_{i+1} \times \cdots \times A_n \\ &\quad i \in [1 \ldots n] \\ approx(i, dfnb) &:= cons(df_c(dfnb), pdf(1, dfnb, c_1), \ldots, pdf(i-1, dfnb, c_1), \\ &\quad pdf(i+1, dfnb, c_1), \ldots, pdf(n, dfnb, c_1), \\ &\quad \ldots \\ &\quad pdf(1, dfnb, c_k), \ldots, pdf(i-1, dfnb, c_k), \\ &\quad pdf(i+1, dfnb, c_k), \ldots, pdf(n, dfnb, c_k)) \end{aligned}$$

In order to apply a function g on a set of k DFNBs, this function has to be defined over $D(C)$. The default definition of *apply* as a derived operation was shown in [Section 3.6.1]. However, the *merge* operation, which is a special case of *apply*, can be defined directly as a special core operation. The prerequisite of this operations is that both DFNBs are defined over the same domain and co-domain:

$$dfnb_1, dfnb_2 : \mathbf{A} \to D(C)$$

and a special operation $\sqcup_{PDF} : PDF(A) \times PDF(A) \to PDF(A_i)$ is implemented over $PDF(A)$ domain, i.e., over the PDFs of each attribute A_i given a class $c \in C$. The $\sqcup_{PDF}$ can be implemented as a sum of discrete random variables, convolutions, etc. [Bertsekas and Tsitsiklis 2002] (the particular implementation of this operation is outside the scope of this paper). The *merge* operation applies an operator $\sqcup_D$ over two Naïve Bayes':

$$\begin{aligned} merge \ &: \ DF[\mathbf{A}, D] \times DF[\mathbf{A}, D] \to DF[\mathbf{A}, D] \\ merge(dfnb_1, dfnb_2) &:= apply(\sqcup_D, dfnb_1, dfnb_2) \end{aligned}$$

The merge operator $\sqcup_D$ is defined over $D(C)$ and is applied on $df_c(dfnb_1), df_c(dfnb_2)$.

Finally, a merger of two decision functions is defined as:

$$
\begin{aligned}
apply\ &:\ (D \times D \to D) \times DF[\mathbf{A}, D] \times DF[\mathbf{A}, D] \to DF[\mathbf{A}, D] \\
apply(\sqcup_D, dfnb_1, dfnb_2) &:= cons(\sqcup_D(df_c(dfnb_1), df_c(dfnb_2)), \\
&\quad \sqcup_{PDF}(pdf(1, dfnb_1, c_1), pdf(1, dfnb_2, c_1)), \ldots, \\
&\quad \sqcup_{PDF}(pdf(n, dfnb_1, c_1), pdf(n, dfnb_2, c_1)), \\
&\quad \ldots \\
&\quad \sqcup_{PDF}(pdf(1, dfnb_1, c_k), pdf(1, dfnb_2, c_k)), \ldots, \\
&\quad \sqcup_{PDF}(pdf(n, dfnb_1, c_k), pdf(n, dfnb_2, c_k)))
\end{aligned}
$$

The *decide* operation can be implemented based on the implementation of a *class* operation in $D(C)$ or in a more straightforward way:

$$
\begin{aligned}
decide\ &:\ DF[\mathbf{A}, D] \times \mathbf{A} \to C \\
decide(dfnb, \mathbf{a}) &:= \arg\max_{c \in C} P(C = c) \times P(A_1 = a_1 | C = c) \times \cdots \times P(A_n = a_n | C = c)
\end{aligned}
$$

In addition to the general DA operations, certain operations can be common to some but not all decision models. This can be modeled with common subtypes of DA. For instance, decision trees and graphs share operations of a Decision Graph Algebra, Naïve Bayes classifiers and other probability-based models share operations of a Naïve Bayes Algebra, etc.

6 Experiments

In [Section 6.1], we compare our decision graph representation with decision trees. It shows that different DA implementations are comparable. In [Section 6.2], we assess the accuracy of learning by merging decision functions. It experimentally validates [Theorem 13].

6.1 Comparison of Decision Trees and Decision Graphs

In the previous section, we presented decision graphs as an DA implementation alternative to decision trees. Here we compare decision graphs generated by the C4.5 learning algorithm with the corresponding decision tree implementation C4.5.

6.1.1 Data Selection

Our experiments are performed on 16 different benchmark datasets from the UCI Machine Learning Repository [Frank and Asuncion 2010]. We were interested in a classification problems and, therefore, selected the 14 largest with both

Table 1: Dataset Characteristics

Id	Dataset	Training Instances	Test Instances	Tree Size/Depth
1	ionosphere	309	42	21/7
2	cancer-wisconsin	500	199	125/4
3	australian	552	138	143/9
4	crx	600	119	174/9
5	dibetes	688	100	27/7
6	anneal	798	133	151/14
7	german	800	200	416/11
8	hypothyroid	2527	636	19/8
9	ad	3057	420	153/55
10	waveform	4000	1000	515/18
11	nursery	11664	1294	905/7
12	chess	28042	4886	10001/5
13	adult	32561	16281	8124/20
14	connect-4	67557	13994	15940/22
15	census-income	159617	39906	46363/26
16	covtype	409985	171027	28389/63

categorical and continuous attributes. We also added two large datasets with only continuous attributes to show the applicability of our theory framework even in these cases.

The used datasets are presented in [Tab. 1] in ascending order of the number of training instances. In addition to the dataset names, the table also reports on the number of training and test instances, and the number of nodes/the depths of the generated decision trees. The final column (Attributes) describes what type of attributes each dataset is using. In all bar charts we present our experimental results for these datasets in the same order as in this table.

6.1.2 Implementation Details

We used the decision trees generated by the FC4.5 learning algorithm [Ping He and Xu 2007, Ruggieri 2002] as a baseline to which we compare our graph-based implementation. FC4.5 is a fast implementation of the C4.5 learning algorithm outlined in [Section 3.3.2]. We adopted the FC4.5 algorithm to directly learn *both* decision trees and decision graphs. As a result, both representations have exactly the same classification accuracy when no additional pruning is applied[8]. In order to make a fair comparison between the two representations, we had to make a few minor adjustments though:

[8] This is confirmed experimentally as well.

1. Each internal node of the decision tree constructed by FC4.5 keeps a training weight, a distribution, and a possible classification, information that is later used for decision making. In our decision graph implementation, the internal nodes do not contain any such information. They just contain information about the attribute they represent. Only the leaves keep a classification distribution $d(C)$. Both representations use the same type of distributions, the frequency based distribution presented in [Section 3]. Additionally, we take into account unknown attribute values by using counts less than one in the distributions [Ruggieri 2002]. Due to these simplifications, the repository is able to identify (and reuse) equivalent sub-graphs without loosing any information.

2. As a consequence of (1), the *decide* operation used for both decision graphs and decision trees is a simplified version of *decide* as implemented in the FC4.5 algorithm. For example, when a test attribute value is missing, the test data is passed to all the children of the current node without taking into account the partitioned weight of the children. Because of this simplification, we expected to lose in classification accuracy when comparing the results with the pure unmodified FC4.5 implementation. However, the accuracy remained the same in all experiments. This observation justifies the simplifications presented in (1) where we neglect additional information kept in the internal nodes.

3. A continuous attribute A_i can be partitioned into different intervals in different branches of the tree. In these cases, we consider each new partitioning as a new categorical attribute and also coming with a new selection operator. Having the same partitions for the same continuous attributes yields to the same categorical attributes. Therefore, the merge of two continuous attributes is reduced to the merge of two categorical attributes (recall to the [Section 3.4]). As we will see when we discuss our experimental results, this drastically reduces the chance for the repository to identify redundancy due to equivalent sub-graphs in datasets where many continuous attributes are used.

6.1.3 Assessment Sizes and Times of Decision Trees and Graphs

In this section, we present the result of the first experiment. Our decision graph implementation recognizes identical subtrees and makes sure that we only keep one such instance. Thus, we expect our graphs to contain fewer nodes than the corresponding decision trees.

[Fig. 8] displays two bars for each dataset for comparing the number of nodes in the decision graph (right) with the tree size (left, always scaled to 100%). For example, Bars 1 (ionosphere) show that our decision graph has the same size as

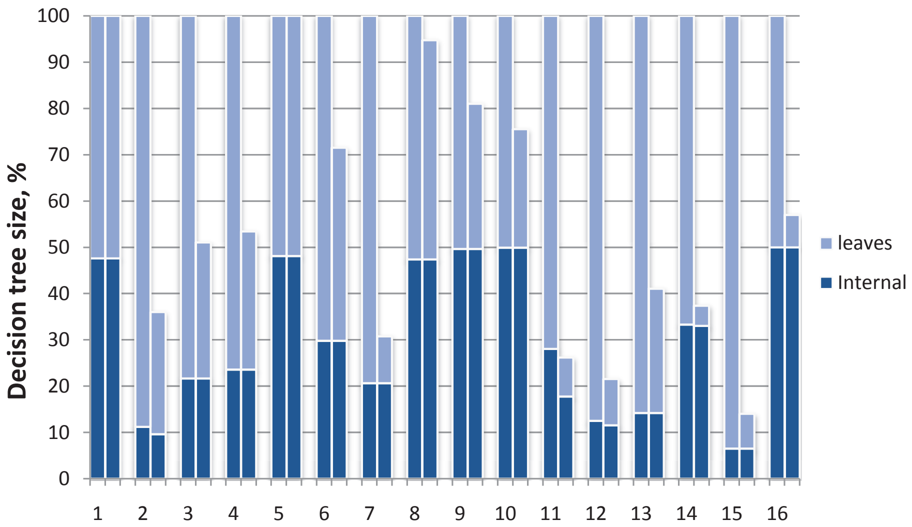


Figure 8: The percentage of reduced nodes and leaves compared to the total tree size (100%)

a corresponding decision tree whereas Bars 2 (cancer-wisconsin) show that our decision graph contains only 36% of the nodes of the corresponding decision tree, indicating a 64% node reduction when using graphs rather than trees. The overall result, an average node reduction of 44%, indicates that much memory can be saved by using our decision graph representation.

Each bar in the chart is also divided into two parts separating internal nodes from leaves. When comparing the number of internal nodes in the tree bars with the corresponding graph bars, we see that, in most cases, the numbers are almost the same, indicating that a majority (98%) of the reduced nodes are leaves. The result has two reasons:

1. A large part of the removed leaf nodes are *bottom leaves* representing attribute value combinations not covered by any instances in the dataset. Although associated with different weights, a majority of these leaves could be removed. The remaining part of the removed leaves are due to non-empty, but identical, distributions.

2. The internal node reduction is quite small (4%). The major reason is that we treat different interval partitions of continuous attributes as entirely different attributes. Hence, the possibility of identifying identical subtrees in these cases is very low. This explanation is supported by the observation that in

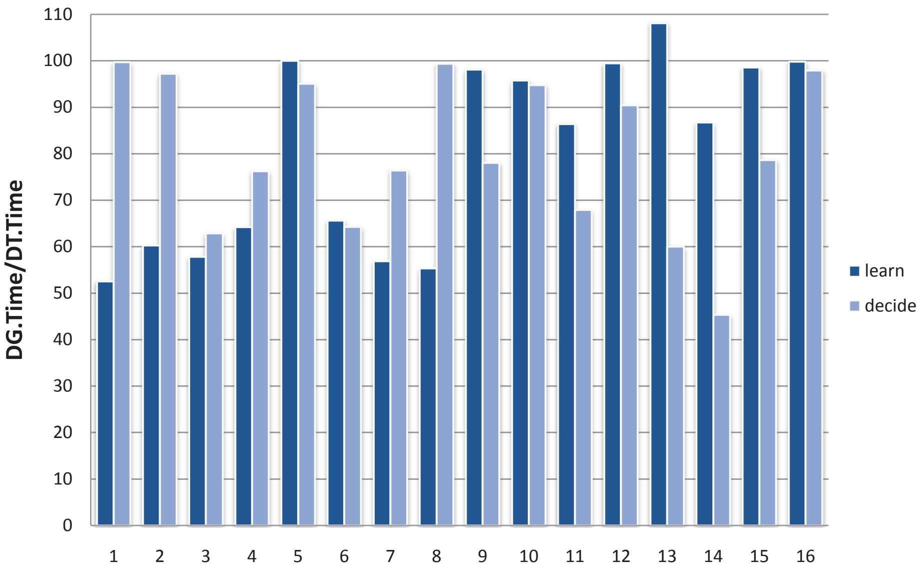
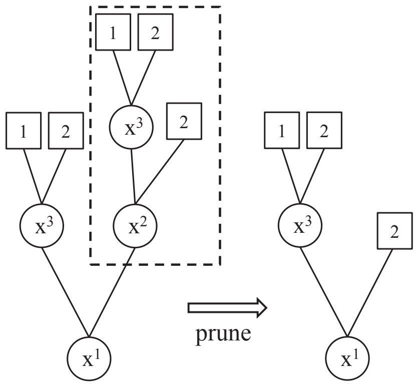


Figure 9: Times of learning and deciding based on a decision graph as % of decision tree (100%)

datasets where we are using mainly categorical attributes (2,11,12,14) , we have a larger internal node reduction. For example, case 11 (nursery) has a reduction of 37%.

Additionally, we measured the time for construction the decision trees (DT) and decision graphs (DG) and also the time used for classifying (*decide*) a fixed number of instances for each dataset. The time was measured in milliseconds but, we have used a relative measure (*DG.Time/DT.Time*) in order to simplify the comparison. The result presented in [Fig. 9] shows that the decision graph implementation is faster in almost all cases. The only exception is the graph construction in the case 13. The average construction and classification time for decision graphs is about 19 and 20% less than for decision trees, respectively. The reduced classification time for decision graphs is at first glance a bit surprising given that the number of selections in both cases are the same. However, this is likely due to a reduced strain on disk cashes and the hierarchy of memory cashes due to the reduced memory usage in the smaller graphs.



(a) df^3 (b) pruned df^3

Figure 10: Pruning of the decision function df^3

6.1.4 Assessment of Accuracy and Times of Approximated Decision Trees and Graphs

In a second experiment, we compare the accuracy of *k*-approximated decision graphs with post-pruned decision trees. We also compare the time required for learning followed by pruning of decision trees with the time required to learn directly to *k*-approximated decision graphs.

The post-pruning in the FC4.5 implementation uses a so-called reduced error pruning strategy [Quinlan 1987b], a rather complicated process where internal nodes of a fully grown tree are removed one at a time as long as the error is decreasing. In fact, the algorithm selects particular subtrees and replaces them with single leaves assigned the most common classification value corresponding to the highest class distribution on the roots of these subtrees [Witten and Frank 2005]. [Fig. 10] shows an example of the reduced error pruning of the previously introduced decision function df^3. The right subtree rooted in x^2 is pruned and replaced by the most common classification value 2.

Our *k*-approximation is, in contrast, a very simple process, where we merge the leaves of all subtrees below a certain depth *k*, cf. [Section 5.1.3]. Notice that although the approximation takes place during the learning process, it is some kind of post-pruning approach since we merge the leaves of fully grown branches, unlike pre-pruning which can suffer from premature termination of a tree-growing process. In fact, by carefully observing *k*-approximation and a reduced error pruning strategy the pruning basically performs a *k*-approximation over particular subtrees chosen by the pruning algorithm. In our experiments, we used the depth of the pruned decision trees to decide *k* used in the *k*-

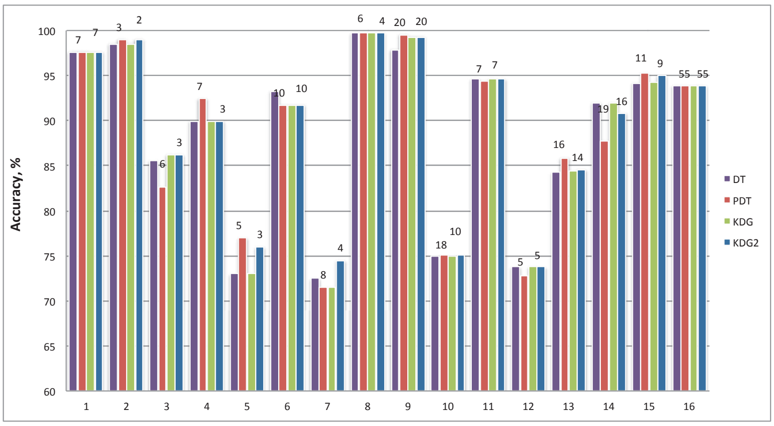


Figure 11: The accuracy gained by pruning decision tree and using k-approximation

approximations. After the pruning over the 16 decision trees was performed by FC4.5 algorithms, we measured the depths of the pruned decision trees and took them as an input $k_1, \ldots k_{16}$ parameters to our k-approximation procedure over 16 decision graphs. [Fig. 11] shows the results of the accuracy comparison.

For each dataset (1–16) we have four bars. The first three show: 1) the accuracy (%) of the decision tree before pruning (denoted DT), 2) the accuracy of the pruned decision trees (denoted PDT), and 3) the accuracy of the k-approximated decision graphs (denoted KDG). On top of each PDT bar, we show the depth k of the pruned decision tree; this is also the depth of the corresponding KDG. Finally, the fourth bar shows the results of an improved k-approximation referred to as KDG2. Here we decrease the depth k step-by-step as long as the accuracy increases (the classification error decreases). The fourth bar shows the KDG2 accuracy along with the final depth as the number on top of that bar.

Firstly, the results for the two approaches (PDT and KDG) are quite similar. On average, the pruned decision trees outperform the k-approximated graphs by only 0.04%. This is a bit surprising given the difference in complexity of the two approaches.

Secondly, in a number of datasets (1,10,11,12) the depth of the pruned trees remains unchanged. Consequently, no k-approximation is applied and the results for DT and KDG are the same. At least for datasets 10 (waveform), the second approximation strategy leads then to improvements: it reduces the size of the decision graph considerably and reaches the accuracy of pruned decision tree, cf. PDT vs. KDG2.

Thirdly, there are case where KDG and KDG2 are more accurate that PDT

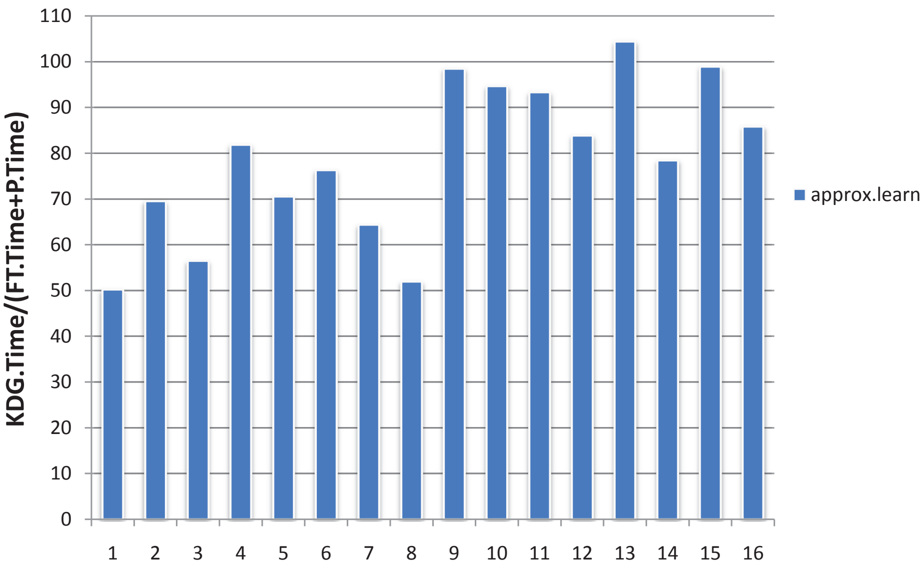


Figure 12: Learning and approximation times of decision graphs as % of decision tree (100%)

(3, 12, 14). For example, dataset 14 (connect-4) indicates that the accuracy of the decision tree before pruning was 92%, after pruning - 88%, k-approximated accuracy - 92% with k=19, and k-approximated accuracy - 91% with $k = 16$. There are also other cases where pruning is not improving accuracy at all and cases where PDT is more accurate than KDG and KDG2.

Regarding the learning and pruning/approximation times, the results are non-ambiguous again. In the time measurements, we have used the same relative metric as in [Section 6.1.3]. [Fig. 12] shows that the learning of k-approximated decision graphs clearly outperforms the joint procedure of learning and pruning the decision trees by up to 50%. On average, the k-approximated approach requires about 21% less time than the tree pruning approach.

6.2 Accuracy of Learning by Merging Decision Functions

In [Section 4.1] we showed that merging of a series of probably accurate decision functions gives a new decision function that is tendentiously more accurate. In this section we present an experiment of merging a series of decision graphs which confirms our theoretical observations.

Table 2: Dataset Characteristics

Id	Dataset	Training Instances	Attributes	Accuracy(%)
1	audiology	200	69	81
2	monks	415	6	61
3	balance-scale	438	4	71
4	tic-tac	749	9	83
5	car	1728	6	95
6	mushroom	6494	22	100
7	nursery	11164	8	95
8	chess	28042	6	74

6.2.1 Data Selection

This experiment is performed on 8 different benchmark datasets from the UCI Machine Learning Repository [Frank and Asuncion 2010]. Notice, that the datasets chosen for this experiment differ from those we selected before. This is because we were only interested in datasets with categorical attributes. Therefore, we took two datasets from previous experiments (nursery and chess) and added six new datasets. We only selected datasets with a number of learning instances ≥ 200.

The selected datasets are presented in [Tab. 2] sorted in ascending order of the number of training instances. In addition to the dataset names, the table also gives the number of training instances and the number of attributes. The last column (Accuracy %) shows the accuracy gained by learning over a complete dataset using FC4.5 algorithm for decision graphs.

6.2.2 Assessment of Accuracy of Merged Decision Graphs

For this experiment, each training set was divided into eight smaller sets. For each small set (1/8 of a complete dataset), a decision graph was learned, refer to as *regular decision graph*. The accuracy of all regular decision graphs was measured with regard to the same common test set.

For each training set, we merged the regular decision graphs step by step using the apply algorithm discussed in [Section 5.1.2] which, in turn, uses the merge operator $\sqcup$ over the distributions on the leaves explained in [Section 3.4.1].

The accuracy of the merged decision function depends on the order in which the regular decision graphs are merged. For each training set, we computed all permutations of its 8 regular decision graphs (40 320 permutations for each training set) and regular decision graphs in the order of their occurance in the

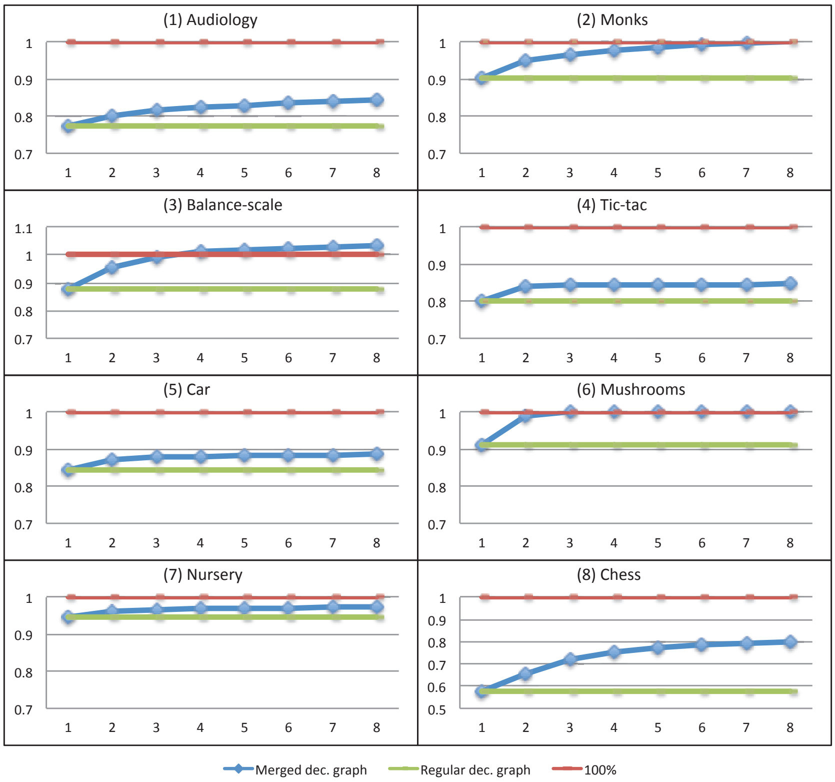


Figure 13: Average accuracy (%) of the merged decision graph, decision graph learned over 1/8 of a dataset (Regular dec. graph) and a line of 100% accuracy

permutations. Measured the accuracy for each step and computed an average accuracy after each merging step.

[Fig. 13] displays for each dataset the average accuracy growth of the merged decision graphs (blue plot) after each step and the average accuracy of the regular decision graph (green line) with regard to the accuracy gained by learning over the complete dataset. That is the merged accuracy and regular accuracy calculated at each step is divided by the complete set accuracy. As predicted by [Theorem 13], all charts show that on each step the accuracy of the merged decision graph tendentiously grows. For example, the accuracy growth for "audiology" is 5.5% (from 62.5% till 68%), while the accuracy of the regular decision graph is around 62.5%. The highest accuracy growth is 16% for the "chess" dataset, and the lowest is 2.5% for "nursery" dataset. Moreover, for all datasets we can see that the merged accuracy slowly growth towards 1̈(red line), i.e. towards the result that a decision model gives when learned over a complete

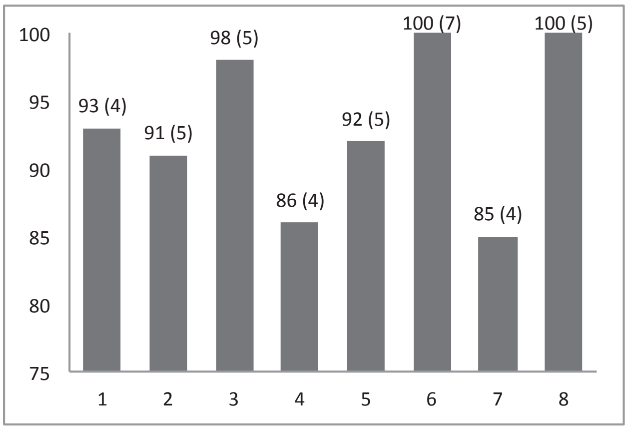


Figure 14: Positive permutations and the number of positive results per permutation.

dataset.

For each dataset, we calculated the average number of *accuracy growth steps* per permutation (the number of steps per permutation when the accuracy of the merged decision function grows by merging) and the *permutations with accuracy growth* (percent of permutations with more than halv of the steps are accuracy growth steps).[9]

[Fig. 14] shows the results for each dataset, cf. dataset identifiers in [Table 2]. Numbers above each bar give the permutations with accuracy growth and the accuracy growth steps (in phases). For instance, Bar 8 ("chess") show that 100% of the permutations have at least 4 merging steps leading to accuracy growth; on avarage about 5 merging steps per permutation lead to accuracy growth.

The overall result indicates that merging decision functions gives a new decision function that is tendentiously more accurate and, therefore, we could define a general online learning approach based on merging decision functions.

6.3 Complexity Bound of the Decision Models

Depending on the strategy and the problem to solve, resulting decision models differ in speed of learning and deciding, their memory consumption, and their decision accuracy. Therefore, the decision models to be applied are usually chosen based on the problem domain, sometimes even on the sample data [P.-N. Tan and Kumar 2005, Nilsson 1996]. Our previous work practically compared the memory and decision overheads of different decision models

[9] Notice, that the total amount of merging steps is seven, since on the first step we do not merge the decision graph.

along with the decision accuracy in a specific problem taken from the context-aware composition domain [Danylenko et al. 2011] .

However, upper bounds on memory and decision overheads of the decision model can be determined regardless of the specific problem domain. Below we present such bounds for the decision models discussed in this paper: dispatch tables, decision trees, decision graphs, and Naïve Bayes classifier.

6.3.1 Decision Tables

Decision tables are implemented as n-dimensional arrays, n the number of context attributes. Each dimension i contains entries corresponding to the sample values of the context attribute A_i. Thus, the memory consumption M of the Dispatch Table can be approximated from below by

$$M = size \times m^n$$

where *size* bytes are necessary to encode all variants in C, and m is the minimum number of samples of any of the context attributes.

Each access to an n-dimension array is basically an access to a 1-dimensional array requiring some offset calculations:

$$offset = base\_address + (((d_1 \times |A_1| + d_2) \times \ldots |A_{n-1}| + d_{n-1}) \times |A_n| + d_n) \times size,$$

where d_i is the index and $|A_i|$ the sample size of the context attribute A_i. Therefore, a decision time for an n-dimensional Dispatch Table can be estimated as

$$T(n) = (log \times k + n) \times T_{flop} + (n - k + 1) \times T_{aa} + c,$$

where k is the number of continuous attributes, log is the number of floating point operations for calculating the logarithm[10], T_{flop} is the time for a flop, T_{aa} is the array access time, and c is a constant time used for small operations.

6.3.2 Decision Trees

Decision trees encode context attributes in the inner nodes. Each outgoing edge of such a node corresponds to a value (or value range) of the context attribute. Each path from the root node to a leaf in the Decision Tree represents actual context values leading to a classification result.

In the worst case, the memory required for capturing the Decision Tree is even larger than for the corresponding table: k leaves if the table has k entries, and (almost) k inner nodes. This size reduces when the decisions are approximated.

[10] Many processors provide the integer $log2$ in a single instruction in hardware; in our Java implementation we need 21 flops.

It can also be reduced if all paths from an inner node lead to the same decision (making this whole subtree redundant). Hence, the memory consumption is

$$M = size \times edges,$$

where *edges* is number of edges in the tree (assuming that *size* bytes are even sufficient to encode all different nodes).

The number of tests necessary to reach a leaf is equal to the *depth* of the Decision Tree. This *depth* varies around the number n of context attributes: for *discrete* context attributes it is at most n; *continuous* attributes can even occur several times on the path due to data partitioning. So, generally, the decision time is $depth \times T_{aa}$ and we approximate

$$T(n) \approx n \times T_{aa} + c.$$

6.3.3 Decision Graphs

Decision graphs represent decision trees in a compact way by eliminating redundant subtrees. In particular, graphs are a generalization of Ordered Binary Decision Diagrams (OBDDs) [Bryant 1992, Bryant 1986], known as a compact representation of Boolean functions. In practice, they reduce the exponential memory consumption of table representations of these functions to acceptable sizes. Generally, decision time and worst case memory size of graphs are the same as for Decision Trees. Thus, the worst case memory and look-up overhead for decision trees and decision graphs are equal, but, due to the elimination of redundancies, the size is expected to be considerably smaller in practice. This assumption is evaluated in our experiments in the next section.

6.3.4 A Naïve Bayes Classifier

Naïve Bayes naively assumes conditional independence of the context attributes from each other using a simple classification method which classifies an item by determining a probability of its belonging to a certain class $c \in C$ [Mitchell 1997].

Naïve Bayes can be specified as set of probabilities that are accessed during classification for computing the most probable variant. For a discrete attribute A_i, the probability is stored in an array with $|C| \times |A_i|$ elements; for a continuous attribute, a *mean* and a *variance* are computed and stored in two arrays of size $|C|$. So the memory consumption with k continuous attributes is

$$M = |C| \times size \times (2k + 1 + (n - k) \sum_{i=1}^{(n-k)} |A_i|).$$

A decision using a Naïve Bayes classifier takes quite some time: it requires 4 flops for each discrete attribute and 88 flops for each continuous attribute (including mathematical operations computing Gaussian) for each possible class. Thus, the decision time is estimated as

$$T(n) = (4n + 84k) \times T_{flops} + (2n + k) \times T_{aa} + c.$$

Based on these theoretical bounds, we cannot decide which decision model to prefer. It depends on the bias between acceptable decision time and memory overhead, and on the concrete problem, i.e., the number of attributes, sample points etc. However, once the number of attributes and the sample data points are decided, the above approximations can be used to (pre-)select a preferred decision model using a common DA interface.

7 Related Work

Best to our knowledge, there is no common unifying theory that specifies the common abstract decision model that allow knowledge combination and reuse between different application domains. Although there exists a variety of systems that provide the analysis engines for capturing and processing large volumes of decision information. They are usually available as stand-alone applications for data analysis and as data mining or machine learning engines which can be integrated to the third-party applications. The examples of such tools are Weka Toolbox [Hall et al. 2009] and Rapid Miner [Rapid Miner 2007]. Weka is a widespread collection of machine learning algorithms for data mining analysis. Rapid Miner is the open-source system for data and text mining that contains methods for web mining, opinion mining, sentiment analysis, etc. The main scientific benefit of such systems is the clean, object-oriented class hierarchy that provides the common general interface from which different decision models can be instantiated. Such interface specifies operations for data processing including routines generating a decision model from decision information and testing it on an unseen dataset. Even thought each tool introduces a general implementation interface, it still varies on the set of operations which is usually limited to construction and evaluation of decision model. This limitation does not allow to combine or modify decision models.

At the same time, there exists a great variety of algorithms and data structures (most commonly modifications of decision trees and decision tables) for learning and capturing classification information. Generally, a modification of these data structures comes with a modifications of a corresponding learning algorithm.

Several variants approach the so-called fragmentation problem, a result of replications as discussed in [Nilsson 1996]. One suggested approach uses decision tree nodes switching on *combinations* of attributes. For instance, Lam and Lee [Lam and Lee 2004] present a method for building classification models by using correlation analysis of attributes (identifying so-called functionally dependent attributes). Similar ideas are presented in [Pagallo and Haussler 1990, John 1994]. Vilalta *et al.* [Vilalta et al. 1997] investigate top-down decision tree construction and prove theoretically and empirically the significance of the fragmentation problem in the learning process. To overcome this problem they choose the best out of a number of possible attribute orderings by assessing their results against all training examples, and thereby avoiding a misclassification of examples for which only little support is found.

Friedman *et al.* [Friedman et al. 1996] present lazy learning, an algorithm which tries to construct the best decision tree for a given decision domain by basically keeping the information of each training instance. However, this algorithm requires a lot of memory when using the classical decision tree structure. Keeping the information in our decision graphs might be more memory efficient.

Oliver presents decision graphs similar to ours as a modification of decision trees [Oliver 1993]. However, Oliver had to invent a completely new learning algorithm. In contrast to Oliver's approach, our graphs are learning algorithm independent and can substitute decision trees as used by any tree construction algorithm.

Quinlan [Quinlan 1987a] merges different decision trees and extracts proposition rules from an already generated decision model in order to eliminate unused conditions replicated in different paths of the tree. Sets of decision rules for the same data domain are merged in order to increase accuracy of a classifier. However, the rules have to be extracted from decision trees and their merger has to be implemented somehow, e.g., in a decision tree again. In contrast, our merge operator can be applied directly to decision trees or graphs. On the other hand, it is not guaranteeing improved accuracy.

Perner [Perner 2011] addresses the issue of comparison of decision trees that represent the classification models of the same problem domain. It arises when two different data sets for one problem are available or when the data set is collected in temporal sequence. In order to compare different decision trees the author proposes an approach of decision rules extraction followed by computing the similarity measure between several sets of rules. In fact, decision function properties such as equivalence and redundancy complemented by evert operation discussed in our paper can allow us to identify similarity in different classification models. In contrast to our work focusing on unification of operations , Perner puts more effort to identifying the specific steps for classifiers having a concrete tree representation.

Bonatec *et al.* [Bohanec and Bratko 1994] addressed the problem of simplifying decision trees, possibly at the expense of accuracy, so that the simplified decision tree still represents the problem domain "sufficiently" well. The chosen simplification method is tree pruning, where the approach is to find the smallest pruned tree with some specified accuracy rate. This was achieved by generating a dense sequence of the pruned trees, decreasing in size, where each tree had the highest accuracy among the pruned trees with the same size. Similar to our $k - approx$ Bonatec et al. have traded the accuracy for the simplicity of a classification model (represented in their case as decision tree). However, they have to develop a new algorithm, which had to be an add-on to the current pruning algorithm, and specify a certain accuracy threshold. In contrast, our $k - approx$ approach does not require any adopted pruning algorithm since it is executed during the learning process and the only parameter it needs is the required depth of the resulting tree.

A theoretical framework for system model checking non-finite aspects of a system is presented by Mokhtari *et al.* [Mokhtari et al. 2008]. Similar to ours, it is based on higher order functions for defining Multiway Decision Graphs (MDG). The goal is to overcome the OBDD binary representation limitations for a certain class of many-sorted first-order logic formulae. Essentially, it is a BDD generalization with signatures for MDG construction, evert, merge, and pruning operations, however, tailored to applications in system model checking.

In addition, there are many approaches suggesting different modifications of decision trees and tables data structures ([Kargupta and Dutta 2004,]).

Finally, our decision graphs are a generalization of χ-terms [Trapp 1999], capturing context-sensitive program analysis results, and Ordered Binary Decision Diagrams (OBDDs) [Bryant 1992, Bryant 1986] representing propositional logics formula in a compact way. χ-terms define *merge* and *approx* but not *evert*, while OBDDs (and their generalizations to multi-valued and multi-target decision diagrams) lack a natural definition of *merge* and *approx*.

7.1 R3 Rational for Choosing Decision Model

In [Section 2] we presented the rationales for applying specific decision models in particular problem domains. Among four different rationales, we particularly mentioned rational of non-functional properties of a decision model (performance or representation properties), which, unfortunately, is the least frequently used. In this section, we present a short overview over the papers which use this particular rational to choose a decision model for solving Computer Science research problems.

In [Bonnel et al. 2011] the authors proposed an Information Retrieval (IR) Interface (IRI) evaluation framework aimed at evaluating the suitability of any IRI to different IR scenarios. In this work the authors used decision trees as a decision model to identifying scenarios in which the particular IRI is effective. The decision trees model was chosen based on its simplicity in representation, interpretation, and rules extraction.

In [Zulkernain et al. 2010] the authors propose an architecture to the system that automatically administrates personal unavailability with regard to the cell phones in order to manage cell phone disruptions. Decision making process is based on the decision trees model in order to process a data from phone's sensors and activate a corresponding correct action. The rational for choosing a decision tree is its cheap computational complexity at runtime, and its suitability to a discrete set of a small number of outcomes.

[Chamlertwat et al. 2012] propose a system which automatically analyze customer opinions from Twitter micro-blog service based on sentiment analysis. As a decision model that classifies each tweet into "opinion" or "non-opinion" the authors used Support Vector Machines (SVM). The reason is the results gained in their previous study that showed that SVM give the best performance in terms of accuracy for filtering opinion tweets.

In [Min Lee et al. 2011] a span detection model which enables parameter optimization and optimal feature selection in order to improve an accuracy of detection has been proposed. In order to maximaze the detection rates the authors used Random Forests decision model. This algorithm is chosen based on its characteristics in terms of execution speed for high-dimension data.

[Garcia Rosa and Adan-Coello 2010] propose a symbolic-connectionist hybrid system that predicts the thematic roles assigned to the word in a sentence context. The authors use a symbolic connection hybrid decision model that is constructed based on Neural Networks. The main reasons are a short training time and a possibility of simple extraction of symbolic knowledge.

Finally, [Dvořśk and Mikušek 2010] present a computer-aided technique for design of digital systems that can produce representations of arbiters and allocators in a form of a Multi-Terminal Binary Decision Diagram (MTBDD). The representation in terms of MTBDD has been chosen based on its compact and non-redundant representation characteristics for Boolean functions.

Its interesting that only 11% of studies refer to the non-functional requirements of decision models. As we discussed before, this may be caused by the fact that benchmarking and adopting decision models for a specific problem domain is a non-trivial task. Therefore, such solution as a generalized DA can benefit the way of choosing a specific decision model.

8 Conclusions and Future Work

In this paper, we define *Decision Algebra*, a theoretical framework for learning and capturing decision information, which is applicable in different fields of Computer Science including (but not limited to) Data Mining and Program Analysis. This unification allows comparing results from different domains and benefiting from the improvements across domain boundaries.

It shows that DA can be used to formalize classical approaches in Data Mining addressing the typical classification problems of fragmentation, replication, and model overfitting. In fact, classical data structures used in classification (e.g., decision trees and tables) and variants thereof as exploited in special learning and classification algorithms can be understood as DA instances by varying the implementations of the DA operations.

To demonstrate this, the paper presents DA generalization specification along with two DA implementations: *decision trees* and *decision graphs*. The latter was inspired by similar data structures from other fields of Computer Science (compiler optimization) demonstrating the possibility of inheriting improvements from other domains due to our unifying theory.

We could map DA operations to decision graphs and decision trees in a straight-forward way. As most DA operations could be defined on a general decision function level (regardless of their implementations in trees and graphs), the comparison of the two variants is rather fair showing advantages and disadvantages of the data structures instead of advantages and disadvantages in different implementations thereof.

On the practical side, our experiments show benefits of decision graphs compared to decision trees regarding memory resource and time utilization as detailed below.

First, we show that, without losing accuracy, decision graphs reduce the memory consumptions of decision trees by 44% (on average over a number of standard datasets). The reduction is largely caused by the elimination of redundant leaves, but also replicated and redundant decision (sub-)trees contribute to the memory reduction. Redundancy increases (and with it the potential saving of our redundancy elimination) as decision graphs avoid keeping additional information in internal nodes. Such internal information can be different for different (otherwise identical) subtrees. The absence of such information does not influence the classification accuracy as proved by our experiments. Also worth mentioning is that the reduction appears to grow with the number of categorical attributes and with the size of (number of instances in) the dataset.

Second, k-approximated decision graphs and pruned decision trees have almost equivalent accuracy. As opposed to, e.g., error based pruning, the k-approximated decision graphs do not apply any complex statistics calculations in the leaves and simply merge classes in a fully grown tree branch. This means that using k-approximated decision graphs allows avoiding additional costly post-pruning.

Third, the time for decision graph construction shows a decrease by 19% compared to the time for decision tree construction. Furthermore, the time measured for classification using decision graphs was 20% less than in the corresponding decision trees. This is the result of less strain on cashes due to the memory reduction in decision graphs. The time for learning followed by pruning the decision tree compared to learning directly to the k-approximated decision graph decreased by around 21%. This result supports our statement about avoiding post-pruning operations mentioned above.

Moreover, theoretically and experimentally we show that merging of a series of probably accurate decision functions gives a new decision function that is tendentiously more accurate. The experiment result of merging decision graphs showed lowest accuracy growth of 2.5% and highest of 16%.

DA is only a first attempt towards a unifying theory in classification and our decision graph based implementation is still a prototype. Quite a few theoretical and implementation aspects should be considered in future work. This includes theoretical and practical modifications of the learning process to enable efficient online learning with high accuracy. More specifically, we seek improving the implementation of the *add* operator of DA, which, in turn, allows joining different classification models from the same data domain and thereby performing an iterative learning. Also, we will investigate how the reordering of attributes (using the *evert* operation) performs and influences the size of the graph. Moreover, future work should specify other than tree-based and probability-based decision models as instances of decision functions including support vector machines, neural networks, etc.

References

[Andersson et al. 2008] Andersson, J., Ericsson, M., Kessler, C., and Löwe, W.: "Profile-Guided Composition"; Software Composition, vol. 4954, pages 157–164, Springer Berlin/Heidelberg, (2008).

[Bertsekas and Tsitsiklis 2002] Bertsekas D.P. and Tsitsiklis J.N.: "Introduction to Probability"; Athena Scientific, (2002).

[Bohanec and Bratko 1994] Bohanec, M. and Bratko, I.: "Trading Accuracy for Simplicity in Decision Trees"; Machine Learning, 15, pp. 223–250, (1994).

[Bonnel et al. 2011] Bonnel, N., Chevalier, M., Chrisment, C., and Hubert, G.: "A Framework to Evaluate Interface Suitability for a Given Scenario of Textual Information Retrieval"; volume 17, number 6, pages 831–858. Journal of Universal Computer Science, (2011).

[Brian and Priestley 2002] Brian, B. A. and Priestley, H. A.: "Introduction to Lattices and Order, Second Edition"; Cambridge University Press, (2002).

[Bryant 1986] Bryant, R. E.: "Graph-Based Algorithms for Boolean Function Manipulation"; IEEE Transactions on Computers, 35, pp. 677–691, (1986).

[Bryant 1992] Bryant, R. E.: "Symbolic Boolean manipulation with ordered binary-decision diagrams"; ACM Computing Surveys, 24, pp. 293–318, (1992).

[Ceci et al. 2007] Ceci, M., Appice, A., Barile, N., and Malerba, D.: "Transductive Learning from Relational Data"; Proceedings of the 5th international conference on Machine Learning and Data Mining in Pattern Recognition, MLDM '07, pages 324–338, Berlin, Heidelberg, Springer-Verlag, (2007).

[Chamlertwat et al. 2012] Chamlertwat, W., Bhattarakosol, P., Rungkasiri, T., and Haruechaiyasak, C.: "Discovering Consumer Insight from Twitter via Sentiment Analysis"; volume 18, number 8, pages 973–992. Journal of Universal Computer Science, (2012).

[Chang and Lin, 2001] Chang, C.-C. and Lin, C.-J.: "Libsvm: a library for support vector machines"; National Taiwan University, Department of Computer Science and Information Engineering [Online], (2001). Available: http://www.csie.ntu.edu.tw/cjlin/libsvm

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V.: "Support-vector networks"; Machine Learning, volume 20, pages 273-297, (1995).

[Danylenko 2011] Danylenko, A.: "Decisions: Algebra and Implementation"; Licentiate Thesis, Linnaeus University, Växjö, Sweden, (2011).

[Danylenko et al. 2011] Danylenko, A., Kessler, C., Löwe, W.: "Comparing Machine Learning Approaches for Context-Aware Composition"; 10th International Conference on Software Composition, pages 18–33. Springer-Verlag, (2011).

[Dvořšk and Mikušek 2010] DvoÅk, V. and MikuÅek, P.: "Design of Arbiters and Allocators Based on Multi-Terminal BDDs"; volume 16, number 14, pages 1826–1852. Journal of Universal Computer Science, (2010).

[Feng et al. 2010] Feng, J.; Attig, A.; Schwarz, M.; and Perner, P.: "Incremental Learning of Statistical Models from a Temporal Data Stream . An MML-Based Appoach."; Trans. MLDM, pages 1–17, 2010.

[Fernandez del Pozo et al. 2005] Fernandez del Pozo, J. A., Bielza, C., and Gomez, M.: "A list-based compact representation for large decision tables management"; European Journal of Operational Research, 160(3), pp. 638–662, (2005).

[Frank and Asuncion 2010] Frank, A. and Asuncion, A.: "UCI Machine Learning Repository" (2010).

[Friedman et al. 1996] Friedman, J. H.; Kohavi, R.; and Yun, Y.: "Lazy Decision Trees" (1996).

[Garcia Rosa and Adan-Coello 2010] Luis Garcia Rosa, L. J. and Adan-Coello, J. M.: "Biologically Plausible Connectionist Prediction of Natural Language Thematic Relations"; volume 16, number 21, pages 3245–3277. Journal of Universal Computer Science, (2010).

[Hafez et al. 1999] Hafez, A.; Deogun, J.; and Raghavan, V. V.: "The Item-Set Tree: A Data Structure for Data Mining"; Data Warehousing and Knowledge Discovery, First International Conference, DaWaK'99, pages 183–192, Springer, (1999).

[Han and Kamber 2000] Han, J. and Kamber, M.: "Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)"; 2nd ed. Morgan Kaufmann, (2000).

[Hall et al. 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H.: "The WEKA Data Mining Software: an Update"; volume 11, number 1, pages 10–18. SIGKDD Explorations, (2010).

[John 1994] John, G. H. (1994): "Robust Linear Discriminant Trees"; AI& Statistics-95, pages 285–291. Springer-Verlag, (2005).

[Kargupta and Dutta 2004] Kargupta, H. and Dutta, H.: "Orthogonal Decision Trees"; Proceedings of The Fourth IEEE International Conference on Data Mining (ICDM'04), pages 1028–1042, (2004).

[Keogh and Pazzani 1999] Keogh, E. J. and Pazzani, M. J. : "Learning Augmented Bayesian Classifiers: A Comparison of Distribution-based and Classification-based Approaches"; (1999).

[King 1967] King, P.: "Decision tables"; The Computer Journal, 10(2), pp. 135–142, (1967).

[Kitchenham and Charters 2007] Kitchenham, B. and Charters, S.: "Guidelines for performing systematic literature reviews in software engineering". Keele and Durham University, Tech. Rep. EBSE 2007-001, (2007).

[Lam and Lee 2004] Lam, K.-W. and Lee, V. C. S.: "Building Decision Trees Using Functional Dependencies"; ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2, page 470, Washington, DC, USA. IEEE Computer Society, (2004).

[Min Lee et al. 2011] Min Lee, S., Seong Kim, D., and Sou Park, J.: "Cost-Sensitive Spam Detection Using Parameters Optimization and Feature Selection"; volume 17, number 6, pages 944–960. Journal of Universal Computer Science, (2011).

[Lawrence Marple 1987] Marple, S. Laurence Jr.: "Digital Spectral Analysis"; Prentice-Hall, Inc., Englewood Cliffs, NJ., (1987).
[Mitchell 1997] Mitchell, T. M.: "Machine Learning". McGraw-Hill, New York, USA, (1997).
[Mokhtari et al. 2008] Mokhtari, Y., Abed, S., Ait Mohamed, O., Tahar, S., and Song, X.: "A New Approach for the Construction of Multiway Decision Graphs"; Proceedings of the 5th international colloquium on Theoretical Aspects of Computing, pages 228–242, Berlin, Heidelberg. Springer-Verlag, (2008).
[Moshkov 1997] Moshkov, M.: "Algorithms for Constructing of Decision Trees"; PKDD '97: Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery, pages 335–342, London, UK. Springer-Verlag, (1997).
[Murphy and McCraw 1991] Murphy, O. J. and McCraw, R. L.: "Designing Storage Efficient Decision Trees"; IEEE Trans. Comput., 40(3), pp. 315–320, (1991).
[Nilsson 1996] Nilsson, N. J.: "Introduction to Machine Learning: An early draft of proposed text book", Stanford University, Stanford, [Online], (1996). Available: http://robotics.stanford.edu/people/nilsson/mlbook.html
[Oliver 1993] Oliver, J. J.: "Decision Graphs - An Extension of Decision Trees", (1993).
[P.-N. Tan and Kumar 2005] P.-N. Tan, M. S. and Kumar, V.: Introduction to Data Mining; Addison Wesley, (2005).
[Pagallo and Haussler 1990] Pagallo, G. and Haussler, D.: "Boolean Feature Discovery in Empirical Learning"; Mach. Learn., 5(1), pp. 71–99, (1990).
[Perner 2011] Perner, P.: "How to Interpret Decision Trees?"; ICDM, pages 40–55, (2011).
[Ping He and Xu 2007] Ping He, L. C. and Xu, X.-H.: "Fast C4.5"; Proc Int Conf Machine Learning Cybernetics, ICMLC 2007, volume 5, pages 2841–2846, Hong Kong, China, (2007).
[Rapid Miner 2007] http://mloss.org/software/view/27/ (2007).
[Quinlan 1987a] Quinlan, J. R.: "Generating production rules from decision trees"; IJCAI'87: Proceedings of the 10th international joint conference on Artificial intelligence, pages 304–307, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc (1987a).
[Quinlan 1987b] Quinlan, J. R.: "Simplifying decision trees"; Int. J. Man-Mach. Stud., 27(3), pp. 221–234, (1987b).
[Quinlan 1993] Quinlan, J. R.: "C4.5: programs for machine learning"; Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (1993).
[Rokach and Maimon 2008] Rokach, L. and Maimon, O.: "Data Mining with Decision Trees: Theory and Applications"; World Scientific, Singapore, (2008).
[Ruggieri 2002] Ruggieri, S.: "Efficient C4.5"; IEEE Transactions on Knowledge and Data Engineering, 14, pp. 438–444, (2002).
[Seredin et al. 2009] Seredin, O., Kopylov, A., and Mottl, V.: "Selection of Subsets of Ordered Features in Machine Learning"; Proceedings of the 6th International Conference on Machine Learning and Data Mining in Pattern Recognition, MLDM'09, pages 16–28, Berlin, Heidelberg. Springer-Verlag, (2009).
[Tang and Meersman 2007] Tang, Y. and Meersman, R.: "On Constructing Semantic Decision Tables"; volume 4653, pages 34–44, Springer, Berlin/Heidelberg, (2007).
[Trapp 1999] Trapp, M.: "Optimierung Objektorientierter Programme"; PhD thesis, Universität Karlsruhe, Karlsruhe, (1999).
[Vilalta et al. 1997] Vilalta, R., Blix, G., and Rendell, L.: "Global Data Analysis and the Fragmentation Problem in Decision Tree Induction"; 9th European Conference on Machine Learning, pages 312–326. Springer-Verlag, (1997).
[Witten and Frank 2005] Witten, I. H. and Frank, E.: "Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)"; Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (2005).
[Zulkernain et al. 2010] Zulkernain, S., Madiraju, P., Iqbal Ahamed, S., and Stamm, K.: "A Mobile Intelligent Interruption Management System"; volume 16, number 15, pages 2060–2080. Journal of Universal Computer Science, (2010).

A Generalized Weak Law of Large Numbers

For our main result, the convergence of mergers of decision functions towards the accurate one, we need to establish the following

Lemma 14. *Let $X_1, \ldots, X_n$ be a series of independent, identically distributed random variables with $E(X_1) = \mu$ and finite variance $Var(X_1) \leq \sigma^2$. Define the weighted average of the X_i:*

$$A_n = \frac{\sum_{i=1}^{n} X_i \times N_i}{\sum_{i=1}^{n} N_i}, N_i > 0.$$

It holds for the expected value and the variance, resp., of these weighted averages:

$$E(A_n) = \mu \tag{1}$$

$$Var(A_n) \leq \sigma^2 \tag{2}$$

$$\lim_{n \to \infty} Var(A_n) = 0. \tag{3}$$

Proof.

$$\begin{aligned}
E(A_n) &= E\left(\frac{\sum_{i=1}^{n} X_i \times N_i}{\sum_{i=1}^{n} N_i}\right) \\
&= \frac{1}{\sum_{i=1}^{n} N_i} E\left(\sum_{i=1}^{n} X_i \times N_i\right) \\
&= \frac{1}{\sum_{i=1}^{n} N_i} \sum_{i=1}^{n} E(X_i) \times N_i \\
&= \frac{1}{\sum_{i=1}^{n} N_i} \sum_{i=1}^{n} \mu \times N_i \\
&= \frac{\mu}{\sum_{i=1}^{n} N_i} \sum_{i=1}^{n} N_i \\
&= \mu
\end{aligned}$$

which proves [Equation 1]. Further

$$\begin{aligned}
Var(A_n) &= Var\left(\frac{\sum_{i=1}^{n} X_i \times N_i}{\sum_{i=1}^{n} N_i}\right) \\
&= \frac{1}{\left(\sum_{i=1}^{n} N_i\right)^2} Var\left(\sum_{i=1}^{n} X_i \times N_i\right) \\
&= \frac{1}{\left(\sum_{i=1}^{n} N_i\right)^2} \sum_{i=1}^{n} Var(X_i) \times N_i^2 \\
&\leq \frac{1}{\left(\sum_{i=1}^{n} N_i\right)^2} \sum_{i=1}^{n} \sigma^2 \times N_i^2 \\
&\leq \frac{\sum_{i=1}^{n} N_i^2}{\left(\sum_{i=1}^{n} N_i\right)^2} \sigma^2
\end{aligned}$$

To see that this term is less or equal σ^2 [see Equation 2], and approaches zero for large n, [see Equation 3], we rewrite its first factor:

$$\frac{\sum_{i=1}^{n} N_i^2}{\left(\sum_{i=1}^{n} N_i\right)^2} = \sum_{k=1}^{n} \frac{N_k^2}{\left(N_k + \sum_{i=1, i\neq k}^{n} N_i\right)^2} = \sum_{k=1}^{n} \frac{N_k^2}{N_k^2 + 2N_k \sum_{i=1,i\neq k}^{n} N_i + \left(\sum_{i=1,i\neq k}^{n} N_i\right)^2}$$

and note that for each k, it holds $\sum_{i=1,i\neq k}^{n} N_i > 0$ for $N_i > 0$, proving [Equation 2], and $\sum_{i=1,i\neq k}^{n} N_i$ approaches infinity for large n, proving [Equation 3]. ∎

From Lemma 14, it immediately follows that the weighted averages A_n converge in probability to their expected values μ:

Lemma 15. *Let A_n be weighted average of a series $X_1, \ldots, X_n$ of independent, identically distributed random variables with $E(X_1) = \mu$ and finite variance $Var(X_1) \leq \sigma^2$. Then for any $\epsilon > 0$*

$$\lim_{n\to\infty} P(|A_n - \mu| \geq \epsilon) = 0.$$

Proof. Due to Chebyshev's inequality, $P(|X - \mu| \geq k\sigma) \leq 1/k^2$, or $P(|X - \mu| \geq \epsilon) \leq 1/\epsilon^2\sigma^2$ (when choosing $\epsilon = k\sigma$), we have:

$$\lim_{n\to\infty} P(|A_{n,c} - \mu| \geq \epsilon) \leq \lim_{n\to\infty} \frac{1}{\epsilon^2} Var(A_n),$$

when choosing $X = A_n, \sigma^2 = Var(A_n)$. This converges to zero for large n as $Var(A_n)$ does according to Lemma 14. ∎

Lemmata 14, 15 and their proofs are similar to the weak law of large numbers stating that under the same conditions the unweighted sample average of real valued random variables converges in probability towards the expected value. In fact, it is a special case with weights $N_i = 1$, which we use in the following (obvious, hence unproved) corollary.

Corollary 16. *Let $D_1, \ldots, D_n$ be a series of independent decisions, identically distributed from a classification distribution $d : C \to \mathbb{R}$ with expected values $E[D_1 = c] = p_c$ and variances $Var[D_1 = c] = p_c(1 - p_c)$ for any decision $c \in C$.*[11] *Let $A_{n,c}$ be the (un-)weighted average of $D_1, \ldots, D_n$ of these decisions with equal weights $N_1 = 1$. Then it holds:*

$$
\begin{aligned}
E(A_{n,c}) &= p_c \\
Var(A_n) &\leq p_c(1 - p_c) \\
\lim_{n\to\infty} Var(A_{n,c}) &= 0 \\
\lim_{n\to\infty} P(|A_{n,c} - p_c| \geq \epsilon) &= 0 \ \textit{ for any } \epsilon > 0.
\end{aligned}
$$

Overall, this Generalized Week Law of Large Numbers can be used as a supplementary material to the text presented in [Section 4] regarding the accuracy of learning by merging decision functions.

[11] Expectation and variance of standard classification distributions; $[\cdot]$ the Iverson bracket with $[cond] = \begin{cases} 1 & \text{if } cond \\ 0 & \text{otherwise} \end{cases}$