Cost-sensitive optimal decision trees dealing with delays

Sven Butzelaar[1]
Supervisor(s): Emir Demirović[1], Koos van der Linden[1]
[1]EEMCS, Delft University of Technology, The Netherlands

January 29, 2023

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Name of the student: Sven Butzelaar
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Frans Oliehoek

Abstract

Machine learning can be used to classify patients in a hospital. Here, the classifier has to minimize the cost of misclassifying the patient and minimize the costs of the tests. Unfortunately, obtaining features may be costly, e.g., taking blood tests or doing an x-ray scan. Furthermore, it is possible that acquiring those test results may take a few days. To train such a classifier, several machine learning algorithms exist. Decision trees appear as favourites since, unlike other classifiers, decision trees do not need all feature values to classify an instance. Current approaches, however, only use heuristics to find a local optimum. Although heuristics are relatively fast, they are not optimal and therefore may not capture well the underlying characteristics of the given dataset. We propose an optimal approach to train cost-sensitive decision trees while also considering these delayed tests. Here we show, smaller trees with higher accuracy and a lower cost can be constructed, compared to a heuristic approach. We use dynamic programming to allow us to skip many calculations, which speeds up the programming time to seconds.

1 Introduction

Training a classifier, using machine learning, is a widely used technique to create a way of making decisions without people having to be involved. The objective of such a classifier can be to minimize the number of times a wrong decision is made. Another objective applied in cost-sensitive classification is when obtaining data may come with a cost. Here the objective is to minimize the misclassifications and the costs of testing. The prototypical example of the problem of cost-sensitive classification is medical diagnosis[12]. There are many other use cases, e.g, classifying software errors[4], however, the rest of the paper uses medical diagnosis to explain the problem. Both the cost of misclassifying a patient and the cost of performing tests must be minimized. Acquiring feature values, like blood tests or x-ray scans can come with costs. An additional condition to the problem is to minimise the time required to obtain those feature values.

Many studies like Davis[4], Turney[12], Freitas[6] and Breiman[3] show that decision trees are the best suitable classifier for this application. Decision trees use step-by-step decisions where the next test is based on the results of the previous test. This results in the classifier not needing to do all possible tests. This is more cost-efficient than other classifiers where all test results are needed upfront. In addition, decision trees are more interpretable as they show exactly what decisions the classifier is based on[1], whereas other classifiers can be hard to interpret.

Optimal decision trees are trees that globally optimize an objective for a given maximum size [2]. So far, not many solutions considered optimal decision tree classifiers, due to their big runtime complexity. Calculating an optimal decision tree is proven to be NP-hard [7]. However, not only have optimal trees a better accuracy. They can also be trained, having a much smaller maximal depth which makes them even more interpretable. On top of that, optimal trees have the property to better capture the underlying characteristics of the dataset[2]. This is because the tree is directly optimised according to the globally defined objective [5].

Currently, optimal trees have been studied using mixed integer programming [2] or dynamic programming[5]. However, the mixed integer programming approach is not scalable. The dynamic programming approach overcomes this by cashing previous calculations and pruning using upper and lower bounds. However, they do not consider costs and delays. Likewise, only trees dealing with costs and delays have been studied without considering

optimal solutions [8]. This research will combine both optimal trees and cost-sensitive trees with delays.

Combining those ideas creates a problem. When it takes time to acquire test outcomes, it is inefficient to wait for every individual test before continuing with the subsequent tests. Therefore, when encountering a delayed test, the approach we use immediately pays the costs of all features that we possibly need further on. Regardles whether the results will be used or not. An example is Figure 1, where the left tree, if node 1 is a delayed test, the costs of features 1, 2 and 3 have to be paid. Now when classifying a patient we traverse through the decision tree and when finding a delayed test we pay the cost of each test in that subtree. By doing this we cannot calculate the optimal tree by independently calculating the left and right subtree. This means the solution in Demirović's [5] cannot be applied to calculate the optimal tree. This research solves this problem by extending Demirović's algorithm with a specialized step dealing with delayed features.

TODO: sumarize results.

The research question is: Can optimal cost-sensitive decision trees, dealing with delayed costs using dynamic programming, be trained and how does this compare to the state-of-the-art? To answer this question the paper consist of six more sections. In section 2 we show in further detail what research in this field is done. We elaborate on the pre-knowledge used in this research in section 3. While in section 4 we explain the implementation of the algorithm used in our research. In section 5 we describe how the experiments are done and what results these experiments gave The ethics and reproducibility of the research is covered in section 6. And lastly, we discuss all conclusions and related work in section 7.

2 Related work

Cost-sensitive decision trees: Several researches, such as ICET[12], TMDP[10] and ID3[4] show algorithms for training cost-sensitive classifiers.

ICET uses a genetic approach to hyper-tune the biases used for deciding on the trade-off between information gain and feature costs. Subsequently, they use these biases to evaluate trees using an induction algorithm. Although ICET does take delays into account, they consider misclassification costs indirectly.

Tree-based Markov Decision Processes (TMDP) takes advantage of the Markov Decision Processes (MDP) as a modelling tool. MDPs allow a classifier to choose which attribute to measure next taking into consideration the expected future sum of costs [9]. They generate decision trees for all subsets of features. With those decision trees, they use MDP to construct a decision tree as the final classifier. The algorithm does show promising results in accuracy and minimizing cost, however, with more than 10 features they consider only using MDP for the ten most expensive features and assuming they will use the cheapest features anyways. This is due to the exponential space complexity of constructing trees for all subsets of the features. On top of that, they do not consider delays.

Cost-sensitive ID3 uses a greedy approach to constructing a decision tree to maximize accuracy while minimizing the test costs. It assumes that the cost of all tests located in the whole tree have to be paid. This greedy approach has a much lower time complexity, but the assumption for paying all test costs gives a much lower accuracy compared to other methods.

Optimal decision trees: Several algorithms, like MIO[2] and Murtree[5] show methods for training optimal decision trees.

Mixed integer optimization (MIO) uses mixed integer programming to calculate the optimal decision tree. This algorithm, however, showed to lack scalability when the number

of features increases.

Murtree uses dynamic programming to compute the optimal tree. It shows way faster runtimes than MIO. The algorithm, however, only works for binary feature values. Section 3.2 explains how the algorithm works in further detail. Since MIO and Murtree have limitations, they don't solve the problem indicated in the introduction. They both are inefficient when dealing with delayed tests.

3 Preliminaries

Our research explores the combination of two concepts. Firstly, a cost-sensitive approach in decision trees, including a technique taking delays into account. Secondly, a method that allows optimal decision trees to be constructed using a dynamic programming approach. We will cover this in sections 3.1 and 3.2 respectively. In section 3.3 we will explore why dealing with delays creates a problem for using dynamic programming to construct optimal trees.

3.1 Cost-sensitive trees with delays

Several papers, like Davis[4], Shani[9], Turney[12] and Freitas[6] study the training of decision trees, minimising both misclassification costs and costs per feature. Costs per feature are common in real-world applications such as training a decision tree to classify whether a patient is sick. A test such as a blood test or an x-ray scan costs more money than asking a patient their age. It is likely that doing the more expensive tests later in the decision tree is cheaper. Another case to consider is when information gained from a test is low while it is expensive. In such a case it may be wiser not to perform the test.

Another common condition that often occurs in real-life applications is delayed tests. It can happen that performing a test or receiving its results, takes time. In the case of having to use multiple delayed tests, it is more time efficient to do them all at once to reduce the waiting time. So far we found only one paper including this in training the trees[12]. They proposed that when encountering a delayed test you immediately pay the costs of all tests in that subtree. This involves the possibility that costs are made for a test without the result being used

The goal of training the decision tree in our research is to minimize the average cost per data point in the dataset. To calculate the cost per data point the sum is taken of the cost of all tests found when traversing through the tree. In the exception for a delayed test, where the costs of all tests in the subtree are added to the total cost. Then, if the data point is misclassified, we add the misclassification cost. When classified correctly no cost is added.

One thing to observe is that the ratio between the feature costs and the misclassification costs has a big impact on the results of the decision trees. When having a relatively low misclassification cost the tree will end up in fewer nodes. On the other hand, when classification costs are relatively high, the costs of the tests will not influence the result much and the tree will mostly be optimised based upon the least misclassifications.

In some cases, it is possible that two features share a common cost. An example is when two different blood tests are needed. Here the tests can be done on the same blood sample. Therefore, the cost of obtaining the blood sample only has to be paid once. We call this a discount.

3.2 Optimal decision trees using the Murtree algorithm

Demirović [5] studied the possibilities of using dynamic programming to reduce the programming time for training optimal decision trees. By using 2 properties of a decision tree it is possible to take advantage of reusing previous calculations. The two properties as found in the research [5] are :

Property 1: *(Independence) Given a dataset $\mathcal{D}$, a feature node partitions the dataset $\mathcal{D}$ into its left and right subtree, such that $\mathcal{D}_{left} \cap \mathcal{D}_{right} = \emptyset$ and $\mathcal{D} = \mathcal{D}_{left} \cup \mathcal{D}_{right}$*

Property 2: *(Overlap) Given a classification node, a set of features encountered on the path from the root node to the classification node, and an instance, the order in which the features are used to evaluate the instance does not change the classification result.*

With property 1 we can now calculate the left and right subtree independently and know that the optimal left subtree and the optimal right subtree together must form an optimal tree. Now we can use property 2 to use memoization. Since the order of features encountered does not matter, we have many repeated calculations which we can store in memory and reuse later. Note that without property 1 memoization would be impossible, due to dependencies of the other subtree.

Eq: (1) shows a high-level recursive overview of the Murtree algorithm taken from (Demirović) [5]. It calculates the minimum possible misclassification, only considering binary features. Given a dataset $\mathcal{D}$, a maximal depth d and a maximal number of nodes n of the tree, Murtree calculates the optimal tree as $T(\mathcal{D}, d, n)$. Furthermore, $\mathcal{F}$ represents the set of possible features. For every feature, f in $\mathcal{F}$ we can split the dataset in $\mathcal{D}(f)$ and $\mathcal{D}(\overline{f})$ for every instance in $\mathcal{D}$ including and excluding feature f respectively.

The first two lines of the equation are there just to place a natural limit on the number of feature nodes and depth to avoid redundancy. The third line is the base case where $\min\{|\mathcal{D}^+|, |\mathcal{D}^-|\}$ is the number of misclassifications. Here, $|\mathcal{D}^+|$ and $|\mathcal{D}^-|$ are the number of positive and negative instances in $\mathcal{D}$ respectively. The fourth line is the recursive step where for every feature the dataset will be split. Then for every possible amount of nodes, the left and right subtrees are calculated. The minimum of these calculations is taken at the end. f is then the optimal splitting feature for where the split was the minimum.

$$T'(\mathcal{D}, d, n) = \begin{cases} T'(\mathcal{D}, d, 2^d - 1) & n > 2^d - 1 \\ T'(\mathcal{D}, n, n) & d > n \\ \min\{|\mathcal{D}^+|, |\mathcal{D}^-|\} & n = 0 \vee d = 0 \\ \min\{T(\mathcal{D}(\overline{f}), d-1, n-i-1) + & n > 0 \wedge d > 0 \\ \quad T(\mathcal{D}(f), d-1, i) & \\ \quad : f \in \mathcal{F}, i \in [0, n-1]\} & \end{cases} \tag{1}$$

To further optimise, the Murtree algorithm uses a specialized method for computing optimal trees with depth of at most two [5]. It precomputes the data in a 2d array where the axis are the features and the data is the number of instances having both features. This can be used to lower the computational complexity.

3.3 Dealing with delays

The algorithm we have so far assumes that subtrees are independent. However, when splitting on a delayed feature we say the costs of all features in the subtree must be paid, even if instances will never use the result of that test. To explain why this violates the property of independence we can look at the example in Figure 1.

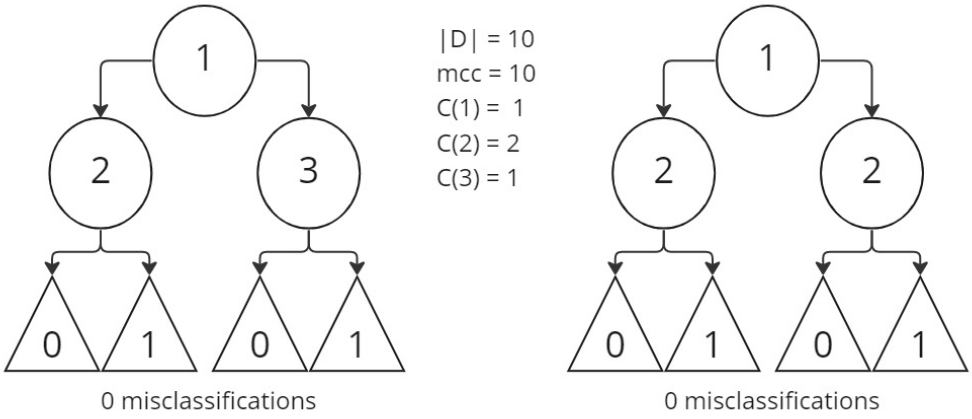

Figure 1: Example of a delayed case

Assume that feature 1 is a delayed feature and $|\mathcal{D}(1)| = |\mathcal{D}(\bar{1})| = 5$ e.i. feature one splits the dataset in half. Recall that to calculate the costs of these trees we take the number of misclassifications times the misclassification cost plus $|\mathcal{D}|$ times the sum of all used features. The cost for taking the left tree would be $0 \cdot 10 + 10 \cdot (1 + 2 + 1) = 40$. The cost of taking the right tree would be $0 \cdot 10 + 10 \cdot (1 + 2) = 30$. Thus, the right tree would be a better choice. However, a dynamic programming approach would take the left subtree, since it will only look at the trees left and right sides independently. This means the algorithm cannot know what is optimal on the left when calculating the right. For the right side, it will think feature 3 is a better choice than feature 2 because it does not know that if we would use feature 2 on the right side too, we only pay the cost of feature 2 once and the total tree would be cheaper.

4 Algorithm for cost-sensitive optimal classification trees with delays

The basic idea of our algorithm is to recursively try every possible tree and to obtain the optimal one. Naively doing this will lead to an infeasible computing time. Therefore, we use optimalisations like pruning and memoization to speed up the process considerably.

In section 4.1 we discuss the basic concept of our method. We extend the method from the Murtree algorithm [5] to also deal with costs and delays. Furthermore, in 4.2 we go into more detail about what techniques are used in our method.

4.1 High-level idea

Our algorithm uses the same approach as Murtree [5], including costs. When a misclassification occurs we add the misclassification cost. Additionally, when splitting on a feature we add this cost. We do this as long as the features we split on are immediate. When splitting

on a delayed test we use our special step where for every possible set of features we calculate its optimal tree using Murtree [5]. Then we add the costs of all features in the set.

We extend the Murtree algorithm T shown in Eq (1)[5] to T' shown in Eq: (2). Like in T we only consider binary features. Additionally to T, every f has its assigned cost $\mathcal{C}(f)$ where if a previously used feature shares a common cost a discount is applied. Furthermore, $\mathcal{I}(f)$ or $\overline{\mathcal{I}(f)}$ return whether the feature is immediate or delayed respectively. On top of that, the constant number *mcc* equals the cost per misclassification.

Eq: (2) shows a recursive function of the high-level idea. The first two lines of the equation are the same as in T and are there just to place a natural limit on the number of feature nodes and depth to avoid redundancy. The third line is the base case where $\min\{|\mathcal{D}^+|, |\mathcal{D}^-|\}$ is the number of misclassifications, which is multiplied by the cost per misclassification *mcc*. In this research, only symmetrical misclassification costs are considered. Including asymmetrical costs is outside the scope of this research. The fourth line is the recursive call. It takes the minimum of the evaluation function E for every possible splitting feature in $\mathcal{F}$ and every possible n.

$$T'(\mathcal{D}, d, n) = \begin{cases} T'(\mathcal{D}, d, 2^d - 1) & n > 2^d - 1 \\ T'(\mathcal{D}, n, n) & d > n \\ \text{mcc} \cdot \min\{|\mathcal{D}^+|, |\mathcal{D}^-|\} & n = 0 \vee d = 0 \\ \min\{E(\mathcal{D}, d, n, f, i) & n > 0 \wedge d > 0 \\ \qquad : f \in \mathcal{F}, i \in [0, n-1]\} & \end{cases} \tag{2}$$

The new function E Eq: (3) is the evaluation function used in T' that has an indirect recursive link, that is, both functions call each other. Its additional inputs are the splitting feature f and the number of nodes used in the right side of the subtree n. The function is divided into two cases, where the first case is for immediate splitting features and the second for delayed features. The first case is almost the same as Murtree where the subtrees are split on f and both costs are added. Now, the cost of splitting the data on f is added. This is the number of instances times the cost of the feature: $|\mathcal{D}| \cdot \mathcal{C}(f)$. In the second case, the costs of all features in the subtree must be paid. Therefore, we take the powerset of all possible features that are not used so far $\mathcal{P}(\mathcal{F})$. For each subset of features in this powerset $\mathcal{F}'$, the minimum misclassifications are calculated using the original Murtree algorithm and multiplied by the misclassification cost *mcc*. Then the cost of splitting $\mathcal{D}$ on $f \cup \mathcal{F}'$ is added. Lastly, the function returns the minimum cost of all subsets.

$$E(\mathcal{D}, d, n, f, i) = \begin{cases} T'(\mathcal{D}(\overline{f}), d-1, n-i-1) + & \mathcal{I}(f) \\ \quad T'(\mathcal{D}(f), d-1, i) + |\mathcal{D}| \cdot \mathcal{C}(f) & \\ \min\{(T(\mathcal{D}(\overline{f}), d-1, n-i-1) + & \overline{\mathcal{I}(f)} \\ \quad T(\mathcal{D}(f), d-1, i)) \cdot \text{mcc} + & \\ \quad |\mathcal{D}| \cdot (\mathcal{C}(f) + \sum_{f_i}^{\mathcal{F}'} \mathcal{C}(f_i)\ ) & \\ \qquad : \mathcal{F}' \in \mathcal{P}(\mathcal{F})\} & \end{cases} \tag{3}$$

4.2 Main algorithm description

We use the Murtree algorithm[5] and extend this to also consider costs for features and when splitting on a delayed feature, paying the costs of all the features in the subset. This subsection is split by first explaining how the algorithm is modified to deal with costs in section 4.2.1. After that section 4.2.2 explains how we introduced delays.

4.2.1 Cost-sensitive optimal trees

The Murtree algorithm only looks at calculating an optimal tree for the least number of misclassifications. To add a cost per misclassification and costs per feature we changed the recursive case and the base case. For the recursive case when combining the two subtrees the cost of both subtrees is added and alse the cost for splitting the particular feature times the size of the dataset is included. Then the best nodes' cost is compared with the current nodes' accuracy and if the current node is better, the current node is assigned to the best node.

For the base case (max depth is smaller or equal to 2) the Murtree algorithm uses its specialised algorithm. Here, updating the best subtrees will be calculated by multiplying the misclassification score (MS) with the misclassification cost and adding the cost of using the child feature. This is the cost of the feature times the number of instances reaching that child node. Then in the end the cost for splitting on the parrent feature is the cost of the left plus the cost of the right plus the cost of using the parrent feature. The best tree is then the tree where the total cost is the lowest. This is illustrated in Algorithm 1. Here Algorithm 1 is part of Murtree [5], where lines 2 to 11 are for immediate features and are almost the same as in Murtree. Only calculating the penalty is different. A further explanation to lines 12 to 25 is given in section 4.2.2

Lastly, the Murtree algorithm uses an extra optimization described in Section 4.4 of the paper[5]. It uses a lower bounding technique that is unusable since we have introduced costs per feature. This technique is therefore not included.

4.2.2 dealing with delays

To solve the problem when introducing delays, we introduce a special step in the method used in our final algorithm. When encountering a delayed test, we create a powerset of all possible leftover features. Then the original Murtree algorithm is used for every possible subset of features. This algorithm uses its own fresh cache. To then calculate the cost we use the returned value from Murtree as the number of misclassifications. This step is shown in Algorithm 2.

Lastly, where Algorithm 1 shows the base case for depth of max two, a special step is needed for when the root feature is delayed. Lines 12 to 25 deal whit this case and calculates the best tree for every combination of subtrees.

5 Experimental Setup and Results

In this section our algorithm is evaluated. To start off, a comparison is made for its average costs and accuracy against the state-of-the-art, using several datasets. Additionally, the scalability of the algorithm by testing its run-time against different numbers of features and instances is evaluated.

We use the extended version of the Murtree[5] algorithm described in section 4. In the experiments the maximal depth of the tree is 4 and the maximal number of nodes is 15. We use most settings where Murtree performed best. The cache uses the dataset variant and the features are dynamically picked in order. However, when the original Murtree algorithm for the delayed case is applied the cache uses the branch variant to minimise the initialization. We do use incremental frequency computation with the exclusion of similarity-based lower bounds. Only when we are executing the delayed step we do use a similarity-based lower

Algorithm 1: Specialised algorithm for computing optimal classification trees of depth two with three nodes extended from Murtree [5]

/* We loop over every combination of two features. f_i as root feature and f_j as child feature */
for $f_i \in \mathcal{F}$ **do**
  if $\mathcal{I}(f_i)$ **then**
    // We can only calculate left and right subtree independently when f_i is immediate
    for $f_j \in \mathcal{F}$ *s.t.* $i \neq j$ **do**
      /* We calculate the penalty and if the penalty is lower we update $BestLeft(f_i)$. Here CS e.i. the number of misclassifications, is calculated the same way as in Murtree */
      $MS_{left} \leftarrow CS(\overline{f_i}, \overline{f_j}) + CS(\overline{f_i}, f_j)$
      $Penalty_{left} \leftarrow MS_{left} * mcc + \mathcal{C}(f_j) \cdot |\mathcal{D}(\overline{f_i})|$
      if $BestLeft(f_i).penalty > Penalty_{left}$ **then**
        $BestLeft(f_i).misclassifications \leftarrow MS_{left}$
        $BestLeft(f_i).penalty \leftarrow Penalty_{left}$
      end
      the same is done for the right subtree
    end
  else
    /* We calculate the penalty for all combinations of the left and right tree when f_i is delayed */
    for $f_{left} \in \mathcal{F}$ *s.t.* $i \neq left$ **do**
      for $f_{right} \in \mathcal{F}$ *s.t.* $i \neq right$ **do**
        $MS_{left} \leftarrow CS(\overline{f_i}, \overline{f_{left}}) + CS(\overline{f_i}, f_{left})$
        $MS_{right} \leftarrow CS(f_i, \overline{f_{right}}) + CS(f_i, f_{right})$
        $Penalty \leftarrow (MS_{left} + MS_{right}) * mcc + (\mathcal{C}(f_i) + \mathcal{C}(f_{left}) + \mathcal{C}(f_{right})) \cdot |\mathcal{D}|$
        if $BestDelayed(f_i).penalty > Penalty$ **then**
          $BestDelayed(f_i).misclassifications \leftarrow (MS_{left} + MS_{right})$
          $BestDelayed(f_i).penalty \leftarrow Penalty$
        end
      end
    end
  end
end
// Now we take the best root feature
for $f_i \in \mathcal{F}$ **do**
  if $\mathcal{I}(f_i)$ **then**
    $Best\_tree \leftarrow$
    $\min\{best\_tree, BestLeft(f_i).penalty + BestRight(f_i).penaty + \mathcal{C}(f_i) \cdot |\mathcal{D}|\}$
  else
    $Best\_tree \leftarrow \min\{best\_tree, BestDelayed(f_i).penalty\}$
  end
end
return $best\_tree$

Algorithm 2: SolveSubtreeDelayedCase($\mathcal{D}, \mathcal{B}, \mathcal{C}, d, n, UB, f, best$), the special step for delayed features

input : Dataset $\mathcal{D}$, branch of the subtree $\mathcal{B}$, feature costs $\mathcal{C}$, depth d, number of feature nodes n, upper bound on the costs UB, the delayed feature we are splitting on f and the best node sofar in this branch $best$

output : Optimal classification tree within the input specification that minimises the costs on D.

```
// Compute allowed number of nodes for child subtrees
n_max ← min{2^(d−1) − 1, n − 1}
n_min ← (n − 1 − n_max)
// We create a loop for every possible subset of the leftover features.
featuresLeftToCheck ← getFeaturesLeftToCheck(D, F)
powersetFeatures ← powerset(featuresLeftToCheck)
for featureSubset ∈ powersetFeatures do
    costForSplitting ← (C(f) + sum{C(f_i) for every f_i in featureSubset}) · |D|
    // if the cost for splitting is bigger than the upper bound we can skip this subset
       of features
    if min{best − 1, UB} < costForSplitting then
        continue
    end
    for n_L ∈ [n_min, n_max] do
        n_R ← n − 1 − n_L
        // Impose an upper bound UB', that ensures that a feasible tree will have
           fewer misclassifications than allowed
        UB' ← (min{best − 1, UB} − costForSplitting)/mcc
        subtree_L ← T(D(f̄), B, C, d, n_L, UB', featureSubset)
        UB' ← UB' − subtree_L
        // If ub' is smaller then zero it means that subtree_L must have been too
           expensive
        if UB' ≤ 0 then
            continue
        end
        subtree_R ← T(D(f), B, C, d, n_R, UB', featureSubset)
        current ← subtree_L + subtree_R + costForSplitting
        if current < best then
            best ← current
        end
    end
end
return best
```

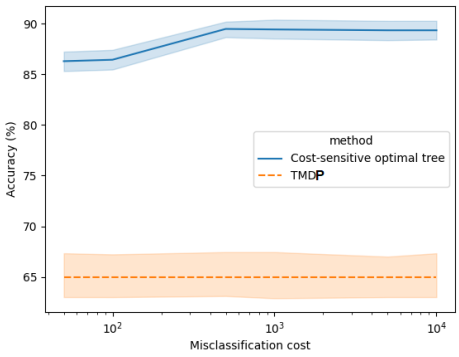

Figure 2: Diabetes

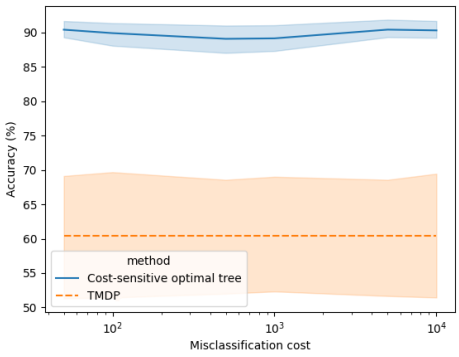

Figure 3: Heart

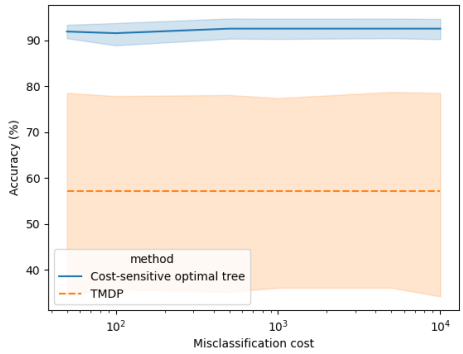

Figure 4: Hepatitis

bound for each individual subset of features. The code is written in C++ and compiled using GCC.

The results are compared with TMDP [9]. Even though this algorithm does not consider delayed, the method shows The code used is open source and written in C# [11]. We consider, for datasets heart and hepatitis, only subsets over the 10 most costly features together with all cheaper features together.

The evaluation of the model is rewritten to also consider delayed tests. All experiments are run on a Windows computer with an I7 core, with 16GB of RAM.

The datasets used, are the same as the ones used in TMDP and can also be found on their GitHub [11]. To use them for our own algorithm the features of the datasets are binarized. This is done by adding a feature for every possible value of the features with more than two unique values. A discount is then added to make using multiple features created from one feature have the same cost as using the original feature. The training/test split is like in TMDP a 80/20% split. We take 9 different splits and both algorithms run on these same splits.

Figure 2, 3, 4 show both methods' accuracy for different values for the misclassification cost. In all datasets, our algorithm scores better on accuracy compared to TMDP. Figure 5, 6, 7 show the average cost per instance divided by the misclassification cost. The objective of

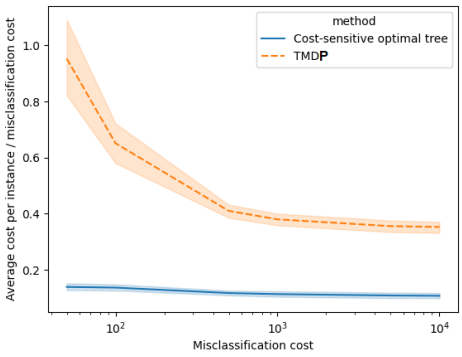

Figure 5: Diabetes

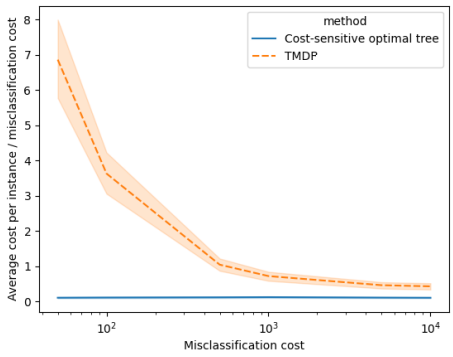

Figure 6: Heart

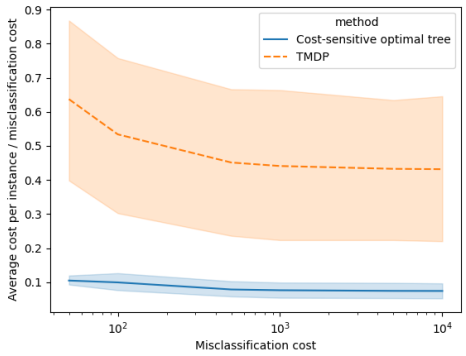

Figure 7: Hepatitis

this research is to minimize these values. The TMDP results have a bigger variance compared to our algorithm since the algorithm does not consider delays. Therefore, delayed tests can appear anywhere in the tree which influences the cost. Our algorithm does a better job of minimizing costs.

The runtime of the algorithm scales exponentionally on the number of feature and on the maximum number of nodes and depth. The misclassification cost and the ratio between delayed and immidiate features influence the runtime, but this is negligible. Here, the runtime may differ around 10%. Figure 8 shows the runtime for different number of features. The hepatitis dataset was used for this figure.

6 Responsible research and reproducibility

To cover all ethical issues that may arise using this optimal cost-sensitive classifier are described in section 6.1. The legal use of all data and information is clarified is also covered in the same section. We cover the reproducibility of the research in section 6.2.

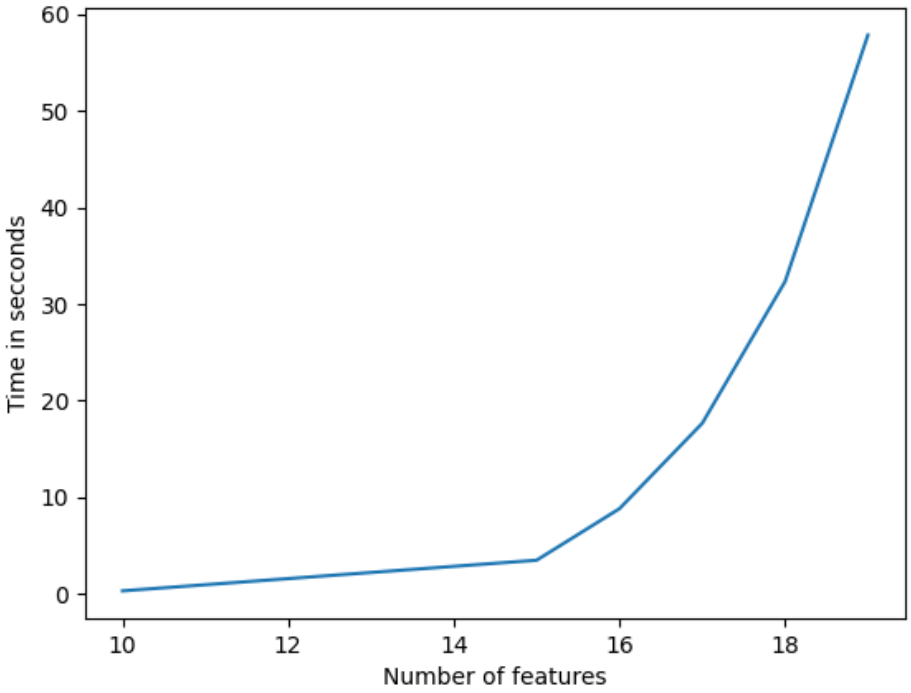

Figure 8: Scalability on the number of features

6.1 Ethics

Applying the methods described for real-world cases e.g. for patient diagnosis, the following few ethical remarks should be considered. Firstly, wrongly classifying could cause a big impact on people's lifes. Therfore, in our opinion for critical cases, there should always be a human expert double-checking the decisions made by these classifiers. Secondly, the cost of misclassifying an instance has a huge impact on how the decision tree is trained. Choosing a bad misclassification cost may result in a undesirable classifier. Lastly, the trained classifier dependents highly on the training data. This makes it important that there are no biases in the data. Furthermore when there is a drift in the data itself, the trained classifier should be reevaluated or retrained.

The related research is cited and all datasets are referred to and open source. Additionally, the code used for the experiments is open-source and cited.

6.2 Reproducibility

All considerations and decisions made are explained to assure full reproducibility. The way the algorithm is extended from Murtree [5] is elaborated on and all code will be made available. Furthermore, all experiments are explained in detail including all variables. The code to the TMDP algorithm is cited and all alternations explained. The data used is open

source and we explain what changes are made in this research.

7 Conclusions and future work

In this research, we show a method to train optimal cost-sensitive decision trees also incorporating an approach to deal with delayed tests. We explain how dynamic programming is used to reduce the runtime considerably. Explanation is given on how we handle delayed features and how these are implemented. Our algorithm was tested against the state-of-the-art and showed a considerable decrease in cost with still a feasible run-time.

Some limitations to this research have to be considered. When we refer to a tree as an optimal tree we imply that trees are only optimal given the features are binarized and the tree has a certain maximum depth. When the data has continuous features we do lose information when binarizing them making them suboptimal. Furthermore, it is possible that a bigger depth is necessary to get an optimal tree.

Multiple elements can be valuable for future researched. Firstly, in our research we only look at symmetrical misclassification costs, while in real-world cases false positives may be more costly than false negatives or the other way around. Secondly, we currently assume that when doing a delayed feature we have to pay the cost of all features in the subtree. It is also possible to perform only all cheap tests of the subtree but leave out the most expensive tests, implying that the expensive test are only conducted after the outcome of the delayed feature. Thirdly, in our implementation, performing the delayed step, we could reuse the cache of other subsets to give upper or lower bounds depending on the subset of features. We believe this could improve the runtime considerably. Fourthly, extending the similarity-based lower bound to also be cost-sensitive is out of the scope of this project. This can also lead to a faster runtime. Lastly, the experiments only used datasets with two labels. It would be possible to extend this to multi-label datasets.

References

[1] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[2] Dunn Bertsimas and Jack Dimitris. Optimal classification trees. 106:1039–1082, 2017.

[3] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and regression trees*. Wadsworth and Brooks, Monterey, CA, 2017.

[4] Jason V. Davis, Jungwoo Ha, Christopher J. Rossbach, Hany E. Ramadan, and Emmett Witchel. Cost-sensitive decision tree learning for forensic classification. In *Machine Learning: ECML 2006*, pages 622–629. Springer Berlin Heidelberg, 2006.

[5] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

[6] Alberto Freitas, Altamiro Costa-Pereira, and Pavel Brazdil. Cost-sensitive decision trees applied to medical data. In *Data Warehousing and Knowledge Discovery*, pages 303–312. Springer Berlin Heidelberg, 2007.

[7] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[8] Susan Lomax and Sunil Vadera. A survey of cost-sensitive decision tree induction algorithms. *ACM Comput. Surv.*, 45(2), mar 2013.

[9] Shlomi Maliah and Guy Shani. Mdp-based cost sensitive classification using decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[10] Shlomi Maliah and Guy Shani. Using pomdps for learning cost sensitive decision trees. *Artificial Intelligence*, 292:103400, 2021.

[11] Guy Shani. Implementation for cost sensitive classification, based on an mdp formalization of the problem. `https://github.com/shanigu/CostSensitive`, 2022.

[12] Peter D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. 2(1):369–409, apr 1995.