KU LEUVEN

Complete decision tree induction functionality in scikit-learn

Ir. Sven Van Hove

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisors:
Prof. dr. Jesse Davis
Prof. dr. ir. Hendrik Blockeel

Assessor:
Dr. ir. Marc Claesen

Mentor:
Elia Van Wolputte

Academic year 2017 – 2018

Preface

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my wive and the rest of my family.

Ir. Sven Van Hove

Contents

Abstract

The `abstract` environment contains a more extensive overview of the work. But it should be limited to one page. [EHWP16]

Chapter 1

Introduction

1.1 Context

Decision tree induction is one of the most well-known tools in the machine learning community. Most of the theoretical groundwork was laid in the last three decades of the previous century. Researchers Leo Breiman and Ross Quinlan have been particularly influential in this space. Some well know algorithms include the Concept Learning System [HMS66], ID3 [Qui79, Qui83, Qui86] by Quinlan and Classificatiom And Regression Trees (CART) [BFSO84] by Breiman.

Contemporary AI researchers focus most of their attentions on neural networks and in particular deep learning — the recent hype around DeepMind's AlphaGo [SSS+17] victories comes to mind — but decision tree research is not dead. Researchers still continue to propose new or improved algorithms and analyses.

Theory is one thing, but the algorithms need to be implemented as computer programs to actually be useful. Sci-kit learn [PVG+11] is a very popular machine learning library written in Python. As such, it also contains implementations of various decision tree induction algorithms. Before sci-kit learn became popular, a Java-based library called Weka [EHWP16] (or "Waikato Environment for Knowledge Analysis" in full) was often used instead. Even today, the implementations of decision tree algorithms in Weka are still in many respects superior to those in scikit-learn. Other libraries that implement similar algorithms exist (e.g., Apache Spark [ZXW+16]), but those are beyond the scope of this text.

1.2 Goal

The goal of this thesis is to alleviate the discrepancies between sci-kit learn and Weka concerning decision tree induction. Mind that decision tree induction tools can never be truly "complete" as stated in the title because the field is immensely broad and still continues to grow. Nevertheless, an effort can be made to improve feature parity between these two popular tools.

One such discrepancy was found when comparing the performance of Weka and sci-kit learn on an activity dataset [KWM11]. The difference between classification accuracies in this case was considerable at about 25% in favor of Weka.

1.3 Motivation

Some would perhaps question the relevance of such *outdated* techniques anno 2018. This feeling is misguided. The advantages of decision tree induction algorithms are still hard to compete with, even for more modern algorithms [PVG+11, Mur98, KZP07]:

1. Comprehensible: makes intuitive sense even for the uninitiated.
2. Transparent, as opposed to for example artificial neural networks
3. Easy to visualize tree (if number of nodes remains small)
4. Non-parametric, makes very few assumptions about data
5. No data normalization required
6. Handles both categorical and numeric data
7. Handles missing data elegantly
8. Handles multiclass, multilabel and multioutput problems natively
9. Fast training
10. Fast inference

Of course decision tree induction algorithms are not perfect:

1. Unstable: small modifications in training data can result in a completely different tree
2. Learning optimal trees is an NP-Complete problem [HR76], so heuristics are used to find approximations
3. Prone to overfitting if not actively countered by adding early stopping criteria or an extra pruning step
4. Prone to bias when one class appears more much frequently in the training set than others.

1.4 Thesis structure

The structure of the remainder of this text is as follows. First, an overview of the literature study concerning decision tree induction will be presented. In particular the link between an implementation and its underlying algorithm will be clarified, including the effects of that choice on the capabilities of the tool.

Chapter 2

Literature review

The relevant literature for this thesis mostly consists of papers concerning decision tree induction. These go back many decades, but fortunately there are some review and survey papers that provide a convenient overview [Mur98, RM05, KZP07]. On top of the academic literature, the source code and accompanying documentation of scikit-learn and Weka has also been a rich source of information.

2.1 Prerequisites

The reader ought to be familiar with basic machine learning concepts such as supervised learning, classification, regression, bias-variance trade-offs, model validation and ensemble learning. Furthermore, elementary knowledge of decision tree induction is expected. The most important basic concepts will be discussed briefly. Topics that are particularly important for the next chapters will be elaborated on.

2.2 Scope

A wide variety of decision tree induction algorithms exists. Here, only the *top down induction of decision trees (TDIDT)* family is considered. It is the most common approach and it is particularly relevant to the software tools under scrutiny.

Furthermore, only classification trees are considered. With little effort, most TDIDT classification algorithms can be converted to regression algorithms. Yet, these are far less popular and better alternatives such as XGBoost [CG16] exist.

Ensemble methods are also out of scope. Recent decision tree algorithms rarely work with a single tree, but rather with an ensemble of trees. Random forests [Bre01] is a very popular example of bootstrap aggregating or *bagging*. Regardless, the scope of this thesis concerns the fundamentals of decision trees, and not their derivatives. Implementation improvements suggested in this thesis could still potentially benefit related ensemble methods.

The algorithms in scope are all offline learning methods invented before the big data era. This implies that computation is done locally and that all data has to fit in memory. As such, online learning methods or distributed algorithms are out of scope.

Finally, only univariate tests are in scope. The test performed in each internal node must only evaluate one attribute of the observation. For categorical attributes, this typically implies checking whether or not the input is equal to a fixed category. For numeric (and thus ordered) attributes, the input value is compared against a fixed threshold using the less than or equal and greater than operators. Consequently, the input space is partitioned recursively using axis-aligned hyperplanes. This scope limitation precludes well-known but seldom used extensions such as oblique trees.

2.3 Terminology

Throughout the relevant literature, there is a lack of ubiquitous vocabulary shared by all researchers. Decision trees are used in various scientific fields, each with its own jargon. Specifically, there is a big divide between researchers that approach the problem from a machine learning perspective compared to those who come from a statistics background. To avoid confusion, some basic terms are reviewed. A *decision tree* consists of *(internal) nodes* which are connected to other nodes via a one-to-many *parent-child* relation on one hand, and *leaves* which have no children on the other hand. The *root node* is the only node without parent. In a *binary tree*, all internal nodes have two children.

Induction algorithms typically receive a *training set* as input data to construct a decision tree while a *test set* is used afterwards for model validation. These sets are tables of data where each row represents an *observation*. All observation are fully described by a common set of *attributes*. Some attributes are *categorical*, others may be *numeric*. Because decision tree induction is a part of supervised learning, one or more *class labels* are also associated with each observation. If the total number of distinct class values equals two, the task is called *binary classification*. Otherwise it is called *multiclass classification*. *Multilabel classification* occurs when one observation can be tagged with a variable number of class values at once. *Multioutput classification* on the other hand occurs when multiple distinct classes, each with their own set of values, have to be derived from the same set of attributes. This can be accomplished trivially by creating multiple trees, each handling one class. Regardless, combining them in one tree might offer performance benefits. In a way, multilabel classification is a special case of multioutput classification. Decision trees are one of the few machine learning algorithms that can handle all these modes of operation natively.

During *training*, first one root node is created and all observations in the training set are stored in this node. When a node is *split* using some test function, that test partitions the observations in subsets and then creates a child node for each subset. This process is repeated recursively until some stopping citerium is reached. The *purity* of a node is defined as the percentage of observations in that node that belong to the majority class. A *pure node* is a node with 100% purity.

2.4 A generic TDIDT algorithm

A typical TDIDT algorithm for classification consists of two phases: a grow phase and an optional prune phase. The grow phase requires three functions with fixed signatures: a test generation function, a splitting function and a stopping function. Historically, researchers presented their TDIDT algorithms with fixed functions. Because of the common interface it is now common to choose these functions *à la carte*. One could try to evaluate the performance of each function separately, but choosing the best of each function does not guarantee a global optimum. Holistic tests must be performed to ensure the best configuration is chosen. Also note that the efficacy of each combination seems to depend on the domain in which it is applied [Min89].

2.4.1 Univariate test generation

Based on the observations in a node, tests can be devised that spit those observations in a number of subsets. The goal of this step is to generate a finite number of tests $\tau_i \in \mathcal{T}$ based on one given attribute. Recall the tests based on multiple attributes exist but are out of scope. In the next step, one specific test is chosen from this set of possible tests.

Generating tests for categorical attributes is trivial. For binary trees the value of the attribute is compared against one specific category. If it matches, it belongs to the first subset, else to the second. This results in as many tests as there are possible categories for the attribute. For non-binary trees, one test suffices that maps each distinct category to a specific subset.

In the case of numeric, ordered attributes, threshold are introduced to partition the observations based on that ordering. That way, an infinite number of tests can be generated, which is of course undesireable. However, at least for the training data, not all tests will result in a different partitioning. A clever choice of thresholds should bring the number of subsets back to a manageable level.

2.4.2 Splitting

Classic TDIDT algorithms work by recursively splitting nodes based on some optimal test $\tau \in \mathcal{T}$, the set of all possible tests. A heuristic called the splitting criterion is required to determine this τ. A few such criteria have stood the test of time.

Purity

The perfect test τ^* creates a partition $\mathcal{S}_{\tau^*} = \{S_1, \ldots, S_k\}$ wherein each subset is pure, so optimizing for weighted average partition purity is a sensible first criterion.

$$p(\mathcal{S}_\tau) = \sum_i \frac{|S_i|}{|S|} p(S_i) \tag{2.1}$$

Here, $S = S_1 \cup \ldots \cup S_k$ and $p(S)$ is the set purity as described above.

Entropy and information gain

In practice purity does not appear to work very well. That is why researchers came up with an alternative based on Shannon's information theory [Sha48]. Quinlan used such metrics in many of his prominent algorithms such as ID3 and C4.5 [Qui86, Qui93], but it was already invented earlier for the Concept Learning System (CLS) [HMS66]. Define entropy (or missing information) of a variable V with possible values v_i and associated probabilities p_i as follows:

$$s(V) = -\sum_i p_i \log_2(p_i) \tag{2.2}$$

The same concept can be applied to the class variable. Define the class entropy $s_C(S)$:

$$s_C(S) = -\sum_c p(c) \log_2(p(c)) \tag{2.3}$$

where $p(c)$ is the probability that a random observation in S belongs to class c. This value can be defined for any node, independent of any specific partition.

For a given test τ, a similar definition can be given for each subset S_i of the induced partition on S:

$$s_C(S_i) = -\sum_c p_i(c) \log_2(p_i(c)) \tag{2.4}$$

For the entropy of the whole partition $\mathcal{S}_\tau$, again use the weighted average entropy of its subsets:

$$s_C(\mathcal{S}_\tau) = \sum_i \frac{|S_i|}{|S|} s_C(S_i) \tag{2.5}$$

Finally, calculate the information gain $h_{IG}(\tau, S)$ of the split that resulted from test τ:

$$h_{IG}(\tau, S) = s_C(S) - s_C(\mathcal{S}_\tau) \tag{2.6}$$

where $\mathcal{S}_\tau$ is the partition resulting from test τ.

Gain ratio

The information gain criterion is biased towards tests with many possible outcomes. This could be a problem in non-binary trees. The gain ratio alleviates this problem. First define split information $SI(\tau, S)$ — the maximum possible information gain — as follows:

$$SI(\tau, S) = -\sum_i \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \tag{2.7}$$

Finally, define the gain ratio:

$$h_{GR}(\tau, S) = \frac{h_{IG}(\tau, S)}{SI(\tau, S)} \tag{2.8}$$

In binary trees, this heuristic typically causes a less balanced tree compared to the information gain criterion [Qui93].

Gini

Distance metrics such as the Gini impurity index can be used instead of heuristics based on information theory [BFSO84]. The definitions follow the same pattern as those of the information gain:

$$g(S) = \sum_c p(c)(1 - p(c)) \tag{2.9}$$

$$g(S_i) = \sum_c p_i(c)(1 - p_i(c)) \tag{2.10}$$

$$g(\mathcal{S}_\tau) = \sum_i \frac{|S_i|}{|S|} g(S_i) \tag{2.11}$$

$$h_G(\tau, S) = g(S) - g(\mathcal{S}_\tau) \tag{2.12}$$

2.4.3 Stopping

Breiman argues in his CART book [BFSO84] that choosing good stopping criteria is far more important than choosing good splitting criteria. If early stopping was not applied or no pruning (see below) was performed afterwards, trees would grow excessively large on real world data sets. This is a classic case of overfitting. It negatively impacts many factors that make decision trees attractive in the first place, such as their comprehensibility and their fast training and inference. It is also detrimental to the performance of the model on unseen data since the model fails to generalize properly.

Some simple stopping criteria are based on the depth of the tree, the purity of a node or the number of observations belonging to a node.

More complex stopping criteria are based on the Minimum Discription Length (MDL) of a tree [Ris78] or on statistical techniques such as a Ξ^2-test. Quilan proposed to use the latter in his ID3 algorithm but decided not to include it in the successor (C4.5) [Qui86, Qui93].

2.4.4 Pruning

A better alternative to early stopping criteria is to let the tree grow freely, and to prune it afterwards in a bottom-up fashion. Typically, the current error of the subtree rooted at the given node is compared to what the performance would be if this node would be converted to a leaf by pruning its children. If it would perform better as a leaf, the children are effectively pruned away. Many different pruning algorithms exists. What follows is a non-exhaustive list of common pruning approaches.

Reduced Error Pruning (REP)

Reduced Error Pruning is one of the most straightforward and statistically sound methods of pruning a tree [Qui87, EK01a, EK01b]. Instead of using the whole training set to grow the tree, some randomly chosen observations are withheld in a seperate validation set. By using this validation set after the growth phase is completed, an unbiased estimate of the error of each node in the tree can be calculated. Nodes at the bottom of the tree are converted into leaves if the estimated error of the leaf is equal to or less than the estimated error of the subtree rooted at the given node. This process is repeated recursively until the smallest possible tree is obtained with the minimum estimated error based on the validation set.

The disadvantage of this method is that less data is available for growing the tree, potentially negatively impacting this process. This is not a concern if training data is available in abundance.

Error Based Pruning (EBP)

Error Based Pruning is a technique used in C4.5 [Qui93]. It does not require a separate validation set, so the full training set can be used to grow the tree. The downside of this is that this method is less statistically sound. An upper bound is calculated based on the training error and that upper bound is used instead of the original error in comparisons. Generally speaking: if a node is associated with fewer observations, then there is less certainty about the error and the upper bound will be further away from the original value.

Cost Complexity Pruning

Cost Complexity Pruning, used in the CART algorithm [BFSO84], takes another approach akin to regularization in classic optimization problems. First, it generates

a series of pruned trees based on the original. Then it considers both the total training error and a cost factor proportional to the size of each tree to make a first selection. If the training error increases due to the pruning, but it is compensated for by a much smaller tree, the operation as a whole can still be considered positive depending on a trade-off factor. The final tree is chosen from this first selection using a seperate validation set. As such, the same drawbacks apply here as for Reduced Error Pruning.

Others

Many other pruning algorithms exist [Min89, BA97, Elo99, EMSK97]. The reader can find some inspiration in the following list:

1. Minimum error pruning [NB86]
2. Pessimistic pruning [Man97, Qui87, Qui93]
3. MDL-based pruning [MRA+95, QR89]
4. Critical Value Pruning [Min87]
5. Pruning using back propagation [KC01]

Alternative: Rule-based Pruning

An outlier in this list is Rule-based Pruning. Decision trees can be converted to a series of if-then statements where the condition is a conjunctive clause. These statements can be further simplified to if-then-else statements and then optimized, which can be seen as an alternative form of pruning. The resulting model is no long a tree, but it can still approximate the underlying concept that the tree used to represent.

2.5 Conclusion

TDIDT algorithms incorporate different components, for each of which a number of alternatives are available. This makes them a very flexible tool with uses in a variety of settings. Popular algorithms such as C4.5 and CART are opiniated in the sense that they each propose a small number of specific configurations of components. Fortunately, that does not stop algorithm implementers from offering more choice to their users, as shown in the next chapter. Note also that there is no single precise definition of ID3, C4.5 or CART. New insights were aquired over time and added to the solution, but the algorithm name rarely changed.

Chapter 3

Software for decision tree induction

Chapter 2 briefly discussed the theoretical basics of decision trees. In this chapter, we consider applications of this theory in the form of two popular software libraries: Weka [EHWP16] and scikit-learn [PVG+11].

3.1 Algorithmic perspective

The most important difference between the two libraries regarding decision trees is in the base algorithm they started from. The J48 algorithm in Weka is based on Quinlan's C4.5 [Qui93], while the `DecisionTreeClassifier` in scikit-learn used CART by Breiman [BFSO84] as the foundation. This has a profound impact on the capabilities of both implementations. These capabilities are divided in categories and discussed one by one in the following subsections.

3.1.1 Structural capabilities

CART only supports binary trees, and the same applies for scikit's implementation. It is an option in J48, but the default settings generate non-binary trees. Binary trees are typically deeper than their non-binary counterparts, making the inference phase more computationally expensive. On the other hand, Elomaa et al. claim that the use of binary discretization with C4.5 needs about the half training time of using C4.5 multisplitting [ER99]. Note that this only impacts splits on categorical attributes; splits of numeric attributes are always binary.

3.1.2 Input capabilities

C4.5 and CART can both handle categorical and numeric attributes. Only the predecessor of C4.5 (i.e., ID3) was not capable of dealing with numeric values. J48 can also handle both just like its theoretical counterpart C4.5.

Scikit-learn's implementation did not inherit the categorical input capabilities of CART. This is understandable from a software engineering perspective: scikit-learn is built on top of numpy [Oli06] which itself is a numeric, scientific computing toolkit. The user can work around this issue by preprocessing the categorical data, using for example a `LabelEncoder`[1] to map each category onto a unique ID or a `OneHotEncoder`[2] that introduces as many boolean dummy variables as there are categories where only one is active at any given category.

Consider an example where an attribute Color contains three categories: Red, Green and Blue. By default Weka would generate a single test out of this attribute. Red would be mapped to the first child, green to the second and blue to the third. This is not possible in scikit-learn because it is based on CART which only generates binary trees. CART — which can handle categorical attributes in theory — would generate three tests instead for attribute x: $(x == \text{Red})$, $(x == \text{Green})$ and $(x == \text{Blue})$. Each test has a boolean output which decides whether to jump to the left or the right child.

In scikit-learn, the one-hot encoder would replace the Color attribute with three numeric dummy attributes: ColorRed, ColorGreen and ColorBlue. In this case, Red is represented as [1, 0, 0], Green as [0, 1, 0] and Blue as [0, 0, 1]. Because these are numeric attributes, the test generation function will try to find thresholds to partition the space. In this case, only one threshold somewhere between 0 and 1 (e.g., 0.5) is needed. As such, each dummy variable will cause the generation of one test: $(x \leqslant 0.5)$. Although the implementation is slightly different, there is a trivial isomorphism between these three tests and the three CART tests. Consequently, the semantics are preserved even though they are slightly obscured.

Alternatively, the label encoder would replace this categorical attribute with a single numeric attribute with possible values 0, 1 and 2. This implies that there is an order among the colors, which is not supposed to be the case. Tests such as $(x \leqslant 1.5)$ could be generated. It is equivalent to $(x == \text{Red} \vee x == \text{Green})$. This rule is more expressive then what was possible so far. That looks positive at first, but not all logical combinations can be represention this way. Even worse, what can be represented depends entirely on the non-existent order of the original categorical variable. In short, this technique does not adhere to the original semantics and should not be used.

In conclusion, the suggested workaround with the one-hot encoder is acceptable but that does not change the fact that it negatively impact two of the decision tree advantages we listed earlier in chapter 1: no data preparation required and excellent comprehensibility.

[1] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

[2] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

3.1.3 Output capabilities

First, regression is not supported by C4.5 and consequently not a capability of Weka's J48. For this thesis regression is out of scope, but it is still an important point. Note however that Weka contains other decision tree algorithms, some of which do accomodate regression.

3.1.4 Splitting criteria

Another difference lies in the choice of splitting criteria. C4.5 and J48 support the typical criteria based on information theory such as information gain and gain ratio. CART and scikit-learn on the other hand both support information gain and the gini impurity index. Gain ratio is not included since it makes little sense for binary trees. The purity criterion is not implemented in any of the algorithms due to its poor performance.

3.1.5 Early stopping and pruning

As mentioned at the end of chapter 2, both C4.5 and CART seemingly have their favourite pruning algorithms. The former explicitly supports Error Based Pruning and Rule-based Pruning, while the latter favours Cost Complexity Pruning. J48, like C4.5, supports Error Based Pruning by default. Quinlan's other proposal, Reduced Error Pruning, is also an option. Rule based pruning is not supported, but Weka contains other rule based algorithms as an alternative. Of course the option not to prune at all is also available. In that case user can fall back on some very simple early stopping criteria such as the minimum number of samples per leaf before a split can happen.

Surprisingly, scikit-learn did not follow Breiman's suggestion of implementing Cost Complexity Pruning. Instead, it opted for the inferior practice of using simplistic early stopping criteria such as a maximum tree depth or a minimum number of observations per node required to split it. This is by far the most significant difference between the two implementations.

3.2 Software engineering perspective

From a practical point of view, the most obvious difference is that the former is written in Java and the latter in Python. This has implications that go beyond mere syntax. Weka is written in a traditional object-orient style of software engineering, while scikit-learn uses a more procedural style with some intermittent object oriented aspects spaced throughout the code. This choice of the scikit-learn developers was motivated by performance reasons, but on the other hand this makes the code harder to understand, maintain and extend. This effect is amplified by the fact that the code is not always pure Python; it also contains some Cython and C code.

3.3 Capabilities

3.4 Conclusion

Chapter 4

Methodology

Intro

4.1 Conclusion

Chapter 5

Results and discussion

Intro

5.1 Pruning

5.2 Categorical attributes

5.3 Conclusion

Chapter 6

Conclusion

Intro

6.1 Contributions

6.2 Retrospective

6.3 Future work

Appendices

Appendix A

The First Appendix

Appendices hold useful data which is not essential to understand the work done in the master's thesis. An example is a (program) source. An appendix can also have sections as well as figures and references.

Bibliography

[BA97] Leonard A Breslow and David W Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(1):1–40, 1997.

[BFSO84] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[EHWP16] F Eibe, MA Hall, IH Witten, and JC Pal. The weka workbench. *Online appendix for "data mining: practical machine learning tools and techniques.": Fourth Morgan Kaufmann*, 2016.

[EK01a] Tapio Elomaa and Matti Kääriäinen. An analysis of reduced error pruning. *Journal of Artificial Intelligence Research*, 15:163–187, 2001.

[EK01b] Tapio Elomaa and Matti Kaariainen. An analysis of reduced error pruning. *Journal of Artificial Intelligence Research*, 15:163–187, 2001.

[Elo99] Tapio Elomaa. The biases of decision tree pruning strategies. In *International Symposium on Intelligent Data Analysis*, pages 63–74. Springer, 1999.

[EMSK97] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence*, 19(5):476–491, 1997.

[ER99] Tapio Elomaa and Juho Rousu. General and efficient multisplitting of numerical attributes. *Machine learning*, 36(3):201–244, 1999.

[HMS66] Earl B Hunt, Janet Marin, and Philip J Stone. Experiments in induction. 1966.

[HR76] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[KC01] Boonserm Kijsirikul and Kongsak Chongkasemwongse. Decision tree pruning using backpropagation neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1876–1880. IEEE, 2001.

[KWM11] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[Man97] Yishay Mansour. Pessimistic decision tree pruning based on tree size. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 195–201. Citeseer, 1997.

[Min87] John Mingers. Rule induction with statistical data - a comparison with multiple regression. *Journal of the operational research Society*, 38(4):347–351, 1987.

[Min89] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[MRA+95] Manish Mehta, Jorma Rissanen, Rakesh Agrawal, et al. Mdl-based decision tree pruning. In *KDD*, volume 95, pages 216–221, 1995.

[Mur98] Sreerama K Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, 2(4):345–389, 1998.

[NB86] T Niblett and I Bratko. Learning decision rules in noisy domains, 1986.

[Oli06] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[QR89] J Ross Quinlan and Ronald L Rivest. Inferring decision trees using the minimum description lenght principle. *Information and computation*, 80(3):227–248, 1989.

[Qui79] J Ross Quinlan. Discovering rules by induction from large collections of examples. *Expert systems in the micro electronics age*, 1979.

[Qui83] J Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning, Volume I*, pages 463–482. Elsevier, 1983.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui87] J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

[Qui93] J Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[RM05] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005.

[Sha48] Claude E Shannon. A mathematical theory of communication (parts i and ii). *Bell System Tech. J.*, 27:379–423, 1948.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[ZXW+16] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.

Master's thesis filing card

Student: Ir. Sven Van Hove

Title: Complete decision tree induction functionality in scikit-learn

Dutch title: Complete beslissingsboom inductie functionaliteit in scikit-learn

UDC: 681.3*I20

Abstract:

500 word abstract

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisors: Prof. dr. Jesse Davis
Prof. dr. ir. Hendrik Blockeel

Assessor: Dr. ir. Marc Claesen

Mentor: Elia Van Wolputte