FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	The decision tree on a stream of features
Student:	Daniel Schmidt
Supervisor:	Ing. Jan Motl
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2020/21

Instructions

Implement a classification decision tree, which can learn on a stream of features. This learning task is known as a subtype of online learning.

Requirements:
1) The implementation must produce results identical to the results obtained with an offline algorithm. Both, the online and offline algorithms must be implemented.
2) The implementations must be sufficient to evaluate the speed up of the online implementation vs. the offline implementation during the model update. Beyond that, the implementations can be minimalistic. For example, handling of heterogeneous data or data with missing values is not required.

Optimizations:
1) Presort each feature *x* based on the tuple (*x*,*y*), where *y* is the label.
2) Maximize reuse of the found splits from the previous run, if available.

Perform:
1) Literature review.
2) Describe the implemented algorithms.
3) Describe the performed experiments.
4) Discuss the experimental results.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 5, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Decision tree on stream of features

Daniel Schmidt

Department of Applied Mathematics
Supervisor: Ing. Jan Motl

July 29, 2020

Acknowledgements

I would like to thank my supervisor, Ing. Ján Motl for always being be able to help and provide valuable feedback even though recent world events made meetings quite challenging. I would also like to thank my family and friends for endless support they were giving me throughout this year.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on July 29, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Schmidt, Daniel. *Decision tree on stream of features.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Táto práca sa venuje rozhodovacím stromom a ich použitiu na alternatívny spôsob dávkového učenia prostredníctvom prúdu príznakov. Predovšetkým sa sústreďuje na algoritmy stavby stromu ich jednotlivé silné a slabé stránky. Praktická časť obsahuje implementáciu dvoch zo spomínaných algoritmov a porovnanie ich trénovacej rýchlosti na prúde príznakov.

Klíčová slova rozhodovacie stromy, strojové učenie, veda o dátach, klasifikácia, dávkové učenie

Abstract

This thesis researches decision trees and their usage for alternate way of online learning through stream of features. Mainly focuses on different algorithms of building the tree, their individual strengths and weaknesses. Practical part includes implementations of two of the explained algorithms and comparison of their training speed on stream of features.

Keywords decision tree, machine learning, data science, classification, online learning

Contents

List of Figures

List of Tables

Introduction

Learning from experience has been part of human lives for almost forever. Invention of computers, however brought us a possibility to use data in larger scale. As many people can nowadays share various information about their life on the internet, what they like and what they don't, what countries they visit, they create huge amount of data that can be processed.

Data can be used to find patterns of behaviour in things that are being surveyed. Common examples are similarities between products on e-shop, music features in one genre of music, symptoms of various diseases. Many times learnt patterns become obsolete, sometimes irrelevant to their case as times change, which indicates that new data must be surveyed and new patterns learnt. Another interesting idea is that the same data can still be viable, but must be seen from another perspective.

This new view on data comes from medical branch where doctors while trying to address rare illness often start with the most probable cause using the information they get, like blood cultures, temperature, blood pressure etc. When the proposed medicine doesn't work they don't necessarily try to compare them to other similar patients which would represent additional data, but they try to do more exams like magnetic resonance imaging to know more information about the one patient they have. More information represents more features of the same thing. Trying to always see different perspective when looking at the data means to be looking at the stream of its characteristic features.

This work focuses on decision trees as models that learn patterns from data and are able to predict some characteristics about them based on these patterns. Specifically is trying to test for time it takes to learn patterns in data between variations of decision trees using different algorithms.

Aim of the thesis

The aim of the thesis is to implement two algorithms of decision trees and compare them in manner of learning on a stream of features.

Firstly in literature review basic idea behind machine learning concept and explanation of different decision tree algorithms will be performed with focus on the most promising algorithm that is suitable to handle input of a stream of features as a subtype of online learning.

The practical part consists of experiments regarding training speed of two implemented algorithms, one being common algorithm for decision tree and the latter being the most promising one, providing comparison between these implementations for various data sets. Later on future work and an possible improvement to the practical part of the work will be discussed.

Chapter 1

Theoretical background

The first chapter describes theory behind experiments in this work, starting from machine learning basics through decision trees, techniques and optimization associated with decision trees. This chapter ends with description of algorithm proposed by various authors as an improvement to decision trees mainly in its training phase which can increase its time efficiency and memory management.

1.1 Machine learning

According to Alpaydin, machine learning is programming computers to optimize a performance criterion using example data or past experience [1]. Model for pattern recognition is defined and its parameters are optimized using training data. Resulting model should be able to predict future behaviour based on similar patterns between data from the past experience and new uncategorized data.

Models prediction can be resembled to linear equation where model's parameters is the structure of equation itself, argument X is all data input used either for learning or testing and solution Y called target variable, prediction or output of the model. Optimization process simply tries to create best equation, as for each given training data input X, the correct Y is given back as an output. It is difficult to create such an equation that satisfies all the data as it can be varied greatly, some of data values can be missing or data not following expected pattern. Therefore training results in model that only approximates data, trying to maximize accuracy of predicting target variable.

1.1.1 Application

It is believed that first attempt of machine learning was used for medical diagnosis, where training data are patient's symptoms with target variable being the recognised disease [2]. After training process, based on symptoms

new patient experiences, model can predict what disease this new patient has. However with missing or misinterpreted symptoms comes greater risk of model predicting wrongly, therefore it is still questionable whether to leave final decision for such model instead of human experts.

Nowadays machine learning has many use-cases where predicting future data or finding patterns can be beneficial either for better outcomes produced by earlier response thanks to quick evaluation and prediction of a model or less consumption of human resources. Medicine is still one of the fields where machine learning shows great potential. One of the examples is gene expression showing promise to be able to distinguish diseases by learning patterns of their biomarkers as well as DNA sequence recognition for identification of genes. [3].

Other common applications:

- Image Recognition
- Speech Recognition
- Text Recognition
- Document classification
- Product advertising

1.1.2 Machine learning models

there are three types of learning in machine learning [4].

1. supervised learning
    - classification
    - regression
2. unsupervised learning
    - clustering
3. reinforced learning

Alpaydin defines supervised learning as mapping data input to output where correct output values are provided by supervisor [1]. Output value can either be continuous in which case learning method is called regression or distinct, classification.

Common supervised learning algorithms according to Kotsiantis are [5]:

Decision trees
Intuitive approach to classification and regression using tree like decision modelling [6]. Each node represents value that divides data points into branches trying to group only data points with same class. Leaf nodes produce target variable prediction.

Support vector machines
Represents data as points in space, examples of the unique categories are separated into different domains.

Artificial Neural Networks
multi-layer system consisting of neurons and synapses that resembles function of the brain where each of the network's neuron computes output based on input from all neurons in the previous layer and sends output using weighted synapses to all neurons in the next. Output layer predicts target variable. Error of prediction is propagated back and weights of synapses are updated.

K-nearest neighbours
Data points are represented as points in space. their position is based on values of its attributes using metric system and prediction is calculated as most occurrent.

Linear regression
Creates hyper-plane that has the smallest sum of squared error distances to individual training data points.

Usage of these algorithms depends on use-case. K-nearest neighbours takes little training time, as the training data represents trained model, as well as retraining which means adding new points into space, but predicting new data takes much longer as distance between new data point and all of training points is measured. On the other hand Decision trees take longer to train and are more difficult to retrain but predicting means crawling through tree to leaf node which takes substantially less time [7].

In unsupervised learning there are no output values, model itself learns regularities and similar patterns in the data. One of the methods is clustering where data is grouped based on its similar attributes, like grouping customers based on previous purchases [8]. In the example of retail there is also a possibility to potential customers for certain products. If many customers buy product A and also product B, customers which bought product A but not B are potential B product customers. This is called association rule.

In the reinforcement-learning model, an agent learns by interacting with environment.Agent can be defined as anything that is capable of observing its surrounding using its sensors and is taking action using its effectors. As the output of this models is sequence of actions, agent chooses next action based

on policy. Policy is some kind of ranking system for agent's actions. Agents learns from past sequences and create new better policy [9]. Good example is game of chess where it's more important to be able to do series of good moves rather than single one. Move is good if it's part of good gaming policy.

Next section describes classification into details.

1.2 Classification

In previous section two types of machine learning approaches we introduced, supervised and unsupervised learning. The difference between them is that in supervised learning labeled training data is used. Labeled data can be explained as data where target variable or output is known, so each model, while in training phase, learns different mappings from input to output.

ID	Age	Rich	Sex	**Survived**
1	64	Yes	M	Yes
2	38	No	M	No
3	25	No	F	Yes
4	52	No	F	Yes

Table 1.1: Example dataset

The table 1.1 is an example of data in machine learning where each row is individual data point and target variable is column 'Survived'. Use-case can be interpreted as learning what people survived sinking of Titanic. In this example whilst having very few data both females survived. Correct output of model learnt on this data using new input data would be that female passengers survived catastrophe because out of all training data points it couldn't map single female to passing away.

Now the question falls to how many different values can target variable gain. Example from table 1.1 shows two values for it's target variable, Yes and No. which means it's a case of classification.

1.2.1 Definition

Classification is by some authors defined as prediction of target variable or class given input data, where class is one of definite set of values. Most common is binary classification where target variable can be only one of two values usually Yes or No, True or False etc. If target variable has continuous set of values it's called regression [10].

In the next section supervised learning algorithm - Decision tree is introduced and explained either for classification or regression.

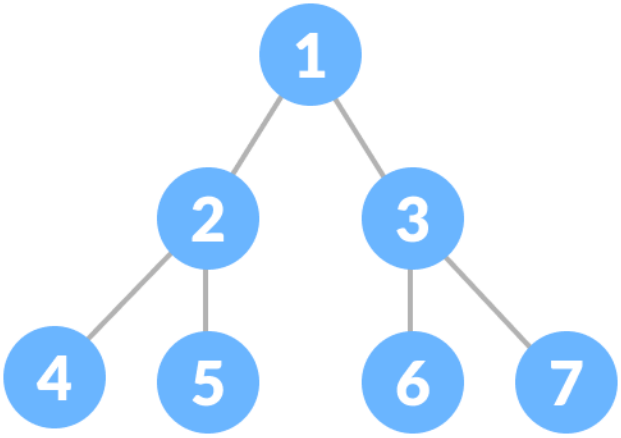
1.3 Decision tree classifiers

To explain decision tree learning algorithms, firstly it's important to describe tree as data structure itself.

1.3.1 Tree data structure

Tree is defined as connected graph without cycle. Graph consists of two things called edges and nodes. Node can be connected to other node via edge. connected graph means that for each pair of nodes exists a way in the graph [11].

Figure 1.1: Example of a tree graph



In figure 1.1 numbered circles are nodes of a graph and gray lines are edges. This is an example of connected graph because from each node there is a way using nodes and edges to every other node. the last part of definition mentioned that tree must be without cycles. This can be easily explained as having only one way between each pair of nodes. The example is already a tree graph there are no cycles, but extra edge between node 4 and 5 would create another choice for creating way between nodes 1 and 5. The first would be 1-2-5 and second would be 1-2-4-5. Nodes 4, 5, 6, 7 have only one connection therefore they are called leaves.

Tree in programming is abstract data type where its edges are oriented. In example 1.1 root node is numbered 1 from which two oriented edges connect it to nodes 2 and 3. These nodes are children in relation to node 1 and node 1 is their parent. Depth of the tree is the longest way from the root node to the lowest leaf node [10].

Useful tree applications are binary search trees where each node contain number in a way that left sub-tree (tree created from left child as a root) contain numbers lower than the parent node and right sub-tree higher ones [10]. Numbers are added by comparing to root node and then continuing left or right whether the number to be added is lower or higher. This repeats with children up to the point where no further comparison is possible and new node is created as a child of previous leaf node either as left or right child. Using this algorithm efficient way of sorting numbers as well as faster searching method can be achieved.

1.3.2 Decision tree

It is very intuitive learning about subject or pattern using series of questions. People form these questions to get to their answer the fastest way asking about attributes of searched answer. In that case it may be efficient to ask a question that eliminates most of undesired options.

Popular game that represents this is called Akinator which is a website that lets people choose anything scaling from things through animals up to fictional characters and then start guessing it. The harder it is for Akinator to guess it the more points are rewarded. After defeating Akinator, if it can't figure it out, there is a option to write what the answer was, which helps improve the system. The guessing method uses series of questions about various attributes to which player answer yes/no. If Akinator is certain what the one thing player thinks about is, then tries to guess it.

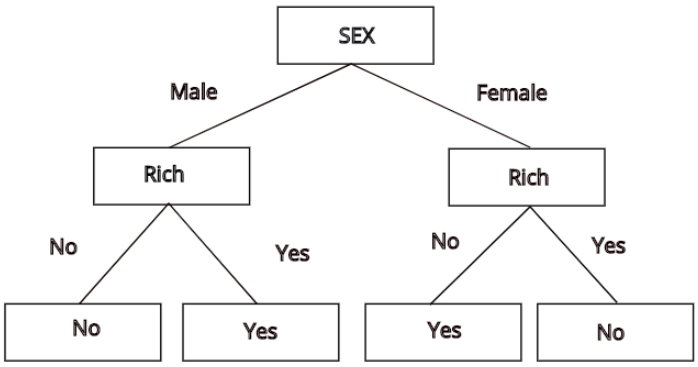
This is effectively represented as a tree where in each node, except leaf ones, is a question about attributes of desired variable with each branch splitting set of possible answers based on answer to the question. This continues until there are no more than one most probable option. Leaf nodes hold this option as the answer what the desired variable is.

Decision tree is a tree based supervised learning model where each non-leaf node represents logic question testing data point's features. Each branch represents outcome of that test. Leaf nodes hold prediction of target variable [7].

1.3.3 Learning process

In Table 1.2 attributes or features except for Survived which is target variable are candidates for tests. These tests then split data set into groups depending on the outcome of that test. For example if node uses attribute *Sex* as test for split data points with *M* value are branched into one child while points with *F* to other. This is case of categorical data where there is discrete number of possible values. In many cases trees these attributes split into distinguished groups and create as many children nodes as is the number of unique values in such attribute. However it is still possible for tree to do binary cut if test

Figure 1.2: Example of a Decision tree model



ID	Age	Rich	Sex	**Survived**
1	64	Yes	M	Yes
2	38	No	M	No
3	25	No	F	Yes
4	52	No	F	Yes
3	25	Yes	F	No
4	52	Yes	F	Yes
3	25	No	F	No
4	52	Yes	F	Yes

Table 1.2: Example data set

compares to one of the unique values and puts every data point sharing this value into one child and all the other into other one.

Handling continuous attribute(see attribute *Age* in Table 1.2 is different. Binary cuts are intuitive as the threshold value is chosen. All the points having value of the attribute bellow this threshold are separated from the ones above this threshold. Choosing such threshold can be time expensive process if every unique value of the attribute found in data set is considered for the test. In case of many data points with lots of different values, intervals are chosen as distinguished classes. Like in case of categorical data, data points are separated into their respective unique intervals.

Another method is to select only few candidates for test for example every twentieth value if there are thousands of them. Using this method faster learning phase can be achieved at cost of probably less accurate model as it can easily miss splitting values with high information gain.

The most crucial part of the decision's tree learning process is to determine how to grow the tree meaning what is the order of questions to ask, so it takes least effort making it from the root note to prediction in leaf nodes. Figure 1.3.2 demonstrates decision tree as simple visualisation which is why decision

tree's are well known in the world. thanks to their high interpretability and non-metric approach it is easy to understand decision tree structure as well as its predictions.

Important part of training decision tree is a way to determine what attribute is best for splitting the data trying to isolate individual classes in these splits. Example would be if someone wants to know whether there is a high chance of rain, asking if there is cloudy sky may give better information to base prediction of weather situation on.

There are two common ways of ranking quality of information gained out of splitting data on question:

- Information gain using entropy
- Gini-impurity

1.3.3.1 Entropy and Information gain

Entropy in information technology shows measurement of chaos in information. Entropy is expressed in following equation where S is list of values, s is number of unique values and p_i is probability of value p_i. Entropy gains values reaching from 0 to 1 where list has 0 entropy if there is only one value and 1 if all values(at least two) are equally shared.

$$H(S) = \sum_{i=1}^{s} -p_i * log_2 p_i \tag{1.1}$$

Information gain shows improvement of entropy in children nodes. By definition it is entropy of parent node subtracted by weighted entropies of each of it's children nodes as show in equation 1.2 where a is attribute used on split.

$$IG(T, a) = H(T) - H(T|a) \tag{1.2}$$

1.3.3.2 Gini-impurity

Another criteria which is used for determining better split is gini-impurity. Idea of this approach is to ask question: "what is the probability to classify data point incorrectly ?". Equation below calculates sum of probabilities of incorrectly predicted labels p_i. The lowest value for gini-impurity is 0 in case of every data point falling in the same category, while 0.5 as the highest in case there is an even distribution of categories.

$$E(S) = \sum_{i=1}^{s} p_i * (1 - p_i) \tag{1.3}$$

1.3.3.3 ID3

ID3 is algorithm used for building decision trees invented by Quinlan in 1983. Idea is to find unused attribute that gives highest information gain. Building comes in greedy fashion, where best possible attribute in each node is selected, not accounting for alternate paths and is used to split data based on testing value. Number of splits equals number of unique values in the attribute [12].

Building in the node ends when any of these called base cases occurs:

1. There are no more attributes on which to produce split
2. All of the data points share the same target class
3. There are no more data points in data set

The first point illustrated greediness of algorithm as once it uses one attribute that attribute can't be used for split again. In the second one as the node is pure, there is no need for further splitting.

The third point can be explained as generalisation of a model. For example one attribute having two unique values, but subset in the current node has only one unique value as other data points were split previously. Trying to split on that attribute produces two children one being empty. Therefore a step back is taken to parent, which is transformed to leaf with class label of majority of points. Generalisation shows in this case that, if new data comes through branch with the missing value of that attribute, it will still be predicted in that parent node that was made leaf.

Advantages of ID3:

- Fast training
- Shallow trees
- Easy to understand

Disadvantages of ID3:

- Tree may be over-fitted
- Does not handle numeric values in attributes

1.3.3.4 C4.5

Presented by the same author Quinlan in 1993 C4.5 is introduced to improve on ID3 weak points. It uses depth-first strategy, where for each data C4.5 considers split and selects the one with highest information gain. Categorical data are split on unique values into multiple branches, whereas numeric data are split based on threshold which is usually one of the values in numeric

attribute [12]. Two branches are created for numeric data where data are split based on comparison to the threshold. Base cases when to stop building are same as in the ID3 algorithm 1.3.3.3.

C4.5 Allows pruning which can lead to higher training error rate, but more importantly reduced unseen testing data error rate. Pruning may resolve problems with over-fitting that is common for ID3 algorithm.

Advantages of C4.5:

- Handles both numeric and categorical data
- Allows attribute to be marked as missing which is then omitted from building phase
- Prunes the tree for better testing accuracy

Disadvantages of C4.5:

- Still may be subject to over-fitting
- Creates more empty branches giving no information to prediction

1.3.3.5 CART

Classification and regression trees algorithm created by Breiman in 1984 is algorithm Which creates binary trees (trees that always have binary cuts). As a criteria to determine better splits in regression trees, it uses mean squared error which is defined in equation below.1.4 Algorithm uses cost-complexity pruning to remove branches with little information gain towards target variable prediction. [12].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1.4}$$

n in the equation is number of data points in data set, vector Y represents observed values from testing data with $\hat{Y}$ being values predicted by model.

Advantages of CART:

- CART can handle outliers
- Usable for both categorical and numerical attributes
- Uses pruning to remove insignificant but costly information

Disadvantages of CART:

- Can be unstable, small modification to data or removing some data points leads to change in splits.

1.3.4 Pruning

Many times its is not worth having training phase to split data until complete purity is achieved, as it can easily lead to over-fitting of a model. Pruning of a tree means to remove sections of tree, which provide only little information to classify data.

According to alpaydin there are two methods of pruning:

- pre-pruning
- post-pruning

Pre-pruning is used in limiting suitable subset length for split which prevents creation of nodes with little classification value. However, more frequent approach is to first create tree where leaf nodes are pure, calculate classification value for sub-trees and remove least significant ones afterwards.

Scientists describe few methods of post-pruning [13]:

1. rule pruning
2. error-complexity pruning
3. reduced error pruning

Rule pruning is based on idea that node in decision tree can be described as conjunction of decisions from root node to leaf. Decision tree resembles series of rules to each leaf node. Nodes are simplified to improve validation accuracy.

Error complexity pruning calculates error cost at each node using formula 1.5, where $r(t)$ is defined as number of misclassified examples divided by number of all examples in a node. $p(t)$ stands for probability occurrence of a node.

$$R(t) = r(t) * p(t) \tag{1.5}$$

Last of the methods of pruning mentioned above is reduced error pruning where each of the tree's nodes are candidates fro removal. Tree uses validation set to measure error rate across different versions of the tree, where nodes are pruned. Tree with lowest error rate after these iterations is selected. It uses error cost to determine error complexity at each node defined below in equation 1.6. $R(T)_t$ is error cost of a sub-tree rooted at node t. Zstands for number of leaves.

$$a(t) = \frac{R(t) - R(T)_t}{Z - 1} \tag{1.6}$$

1.3.5 Parameters

Each model comes with it's own set of parameters. These parameters are optimized for best accuracy, time efficiency etc. There are 2 types of parameters regarding machine learning models:

- model parameters
- model hyper-parameters

Main difference between parameters and hyper-parameters is that parameters are learnt during training of the model whereas hyper-parameters are supplied to the model at the beginning of the training.

In decision trees order of the tests in nodes is considered to be parameter as the order is learnt by model based on information gain of splits, but maximum tree depth is hyper-parameter supplied to model that can't be exceeded. Hyper-parameters are changed during many training iterations each one having different setup for later optimization while parameters are based on data input.

Decision tree hyper-parameters:

Criterion Criterion to determine better split. Either gini-impurity or entropy.

Maximum tree depth Maximum depth of the tree after which prediction must be made even if subset still isn't considered pure.

Threshold As the information gain or gini-impurity is real number, threshold also being rather small real number, is used to stop splitting. If the best splitting index is below this threshold, node is considered pure and no more split evaluations are done.

1.3.6 Prediction

Prediction of trained decision tree tests features in new data to values stored in nodes and traversing to bottom through tree, until leaf node is reached. Value stored in this node is prediction of target variable in new data.

1.3.6.1 Accuracy

In supervised learning algorithms it's important to estimate model's prediction accuracy either for future predictions or selecting the best model for use-case at hand. Many times based on accuracy not only model is retrained optimizing parameters, but also several different models are trained the same way, usually the most accurate model being selected. Other criteria for choosing different model would be interpretability or time expense learning and predicting.

1.3.6.2 Bias-variance trade-off

R. Kohavi believes that it's important to estimate accuracy with low variance and low bias. [14] Bias is difference between model's average prediction and

correct prediction. Very high bias usually means that model hasn't been able to recognise underlying pattern in the data, often called model is under-fitted, therefore it's predictions are often wrong in training as well as in testing data. Complexity of a model influences bias. In Decision trees high bias can be cause with low maximum depth as the model is not able to learn pattern in data.

On the other hand if maximum depth on the tree is not limited it can cause opposite problem which is high variance or over-fitting. High variance means that model learns training data too much, which causes noise in data and errors to be included in learning phase. Even if results on training data bring excellent accuracy thanks to detail high variance model paid to it, results on new test data have high error rates, which makes such model inferior. In decision trees having very deep trees causes high variance.

Bias-variance trade-off is finding the balance between bias and variance so the accuracy of model is at it's peak. It can be seen from example with decision tree that finding the right depth for the tree is crucial, as either very deep or very shallow tree contributes to high error rates.

1.3.6.3 K-fold cross-validation

R. Kohavi defines k-fold cross-validation of data set as random split into k approximately equally sized subsets. Training is happening k times with each time with different one subset as testing data other being training data.

Figure 1.3 illustrates k-fold cross-validation. After training each time, accuracy of model is estimated, with final accuracy being average accuracy of all iterations. Special case of k-fold cross validation is leave-one-out which is a particular case where k in k-fold is equal to 1. If there are n data points in data set, estimating accuracy in such model means training model n times having all data points but one as training data, while having the one left as testing data.

1.3.7 Online learning

In many cases power of the trained model to predict correct future values may fade over time as trends tend to change, which could lead to changes for correct predictions. Online learning methods respond to the issue where models are able to retrain on new sequences of data. Models analyze error of prediction after each input and adjusts model's parameters to improve accuracy. [15]

Some models are more sensitive to changes in data than the others. In k-nearest neighbours adding new to data means to add new points into the space with other points unaffected by this process, whereas decision trees creates splits based on values of features. With new data points tree cannot simply put new data as new splitting value at leafs as these new data may well be important to consider as splitting values in higher branches. Retraining

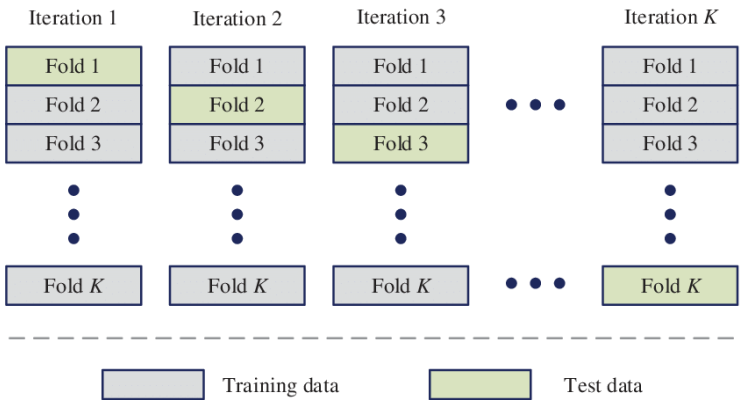


Figure 1.3: k-fold cross-validation

whole tree with new and all older data can lead to high time consumption for training tree model.

If optimization of model's parameters comes after small set of data rather than after each one, such method is called batch learning. Being less thorough than online learning but much more time efficient makes batch learning more appealing in many scenarios.

1.3.8 SLIQ

1.3.8.1 Overview

SLIQ, standing for Supervised learning in Quest, is another method of building decision tree introduced at IBM research Center by Manish Melta, Rakesh Agrawal and Jorma Rissanen in 1996 during data mining project called Quest. This different approach differentiates from the common algorithms such as C4.5 and CART by eliminating redundant sorting of attribute values at each node, while also being able to use breadth-first strategy for best split cost evaluation at each node in the current tree depth during building phase of a tree. As for pruning phase SLIQ approach uses Minimum Description Length principle [16].

1.3.8.2 Scalability

In the previous sections are described common algorithms for building a tree such as CART 1.3.3.5 and C4.5 1.3.3.4. These algorithms while handling numeric attributes have one weak spot in common. At each node while trying to evaluate splits they need to have attribute values sorted to traverse them and test for value that creates the best split.

This creates an issue where for each node, each attribute column needs to be sorted thus creating issue with having large data-sets consisting of high

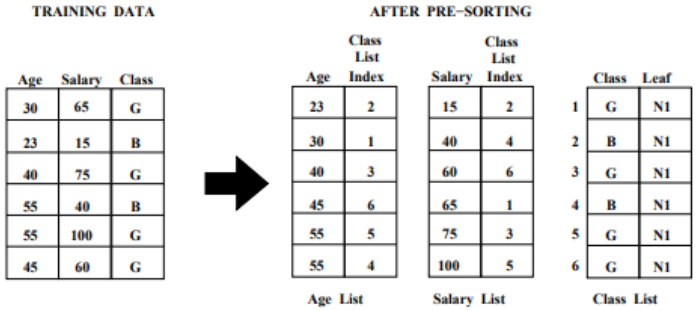


Figure 1.4: Pre-sort of attribute columns

number of attributes and individual values within them [16].

1.3.8.3 Building phase

SLIQ approach uses pre-sorting technique where each attribute column is sorted only once at the start of building phase. By using pre-sort SLIQ greatly reduces cost for creating sorted attribute columns and improves scalability of such Decision tree for large data-sets.

The Pre-sorting technique creates sorted list for each attribute column where entries are tuples. Each tuple consists of attribute value and class list index. Individual tuples are sorted based on attribute value. Class list index references to a separate list of target variables so each attribute value can be linked to its target variable of the training examples.

Class list entries also consist of two fields. First is the very class of the target variable and second being reference to the leaf node of the decision tree. Each ith class list entry represents the ith example of the training data-set. Training examples with common leaf node index in their respective class list entries belong to the same partition of the data much similar to the subsets of data created by splits using CART or C4.5. Attribute list can be written to disks if memory needs to be saved but class-list must be present for the whole building phase [16].

At the start of the building phase all leaf node references in the class list are initialized as root node. In Figure 1.4 training data is processed into sorted attribute lists and class list with leaf node references pointed to root node in the figure shown as $N1$ [16]. After process of pre-sorting and creation of a class list, splits are evaluated and tree-growth begins.

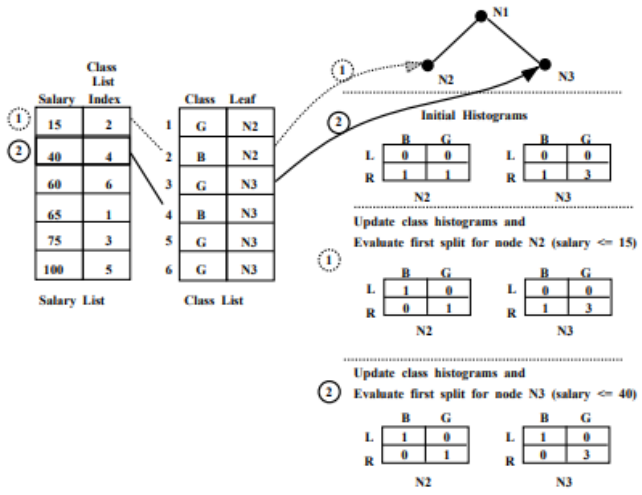


Figure 1.5: split evaluation

1.3.8.4 Split evaluation

Split evaluation at each node can be separated into two phases:

- Split processing
- Class list update

Splits are processed during one time traversal through each of the attribute lists created and sorted in the previous phase. Each of attribute value can be linked to its class and partition of the leaf node. For each node class histogram is created, which is a data structure that holds frequency of classes belonging to data in the leaf node's partition.

Histograms in leaf nodes are updated after each entry v linked to the leaf node. At the same time splitting index is evaluated for the test $A \leq v$. After one traversal of attribute list best split for the attribute is known for every leaf node.

Figure 1.5 shows evaluation of a split on a salary attribute [16]. L in leaf node's histograms represents data that satisfy the test $A \leq v$ where v is the splitting value and R represents partition that doesn't satisfy the test. Numbers in the histogram table count data points after test and their class distribution. After the split of $N3$ on salary value 15 it's histogram contains 0 data points in L section for either B or G class and 3 for G with 1 for B in R section. This means that there are 4 data points in partition $N3$ where all of the have salary value higher than 15 and their distribution is 1 : 3 for

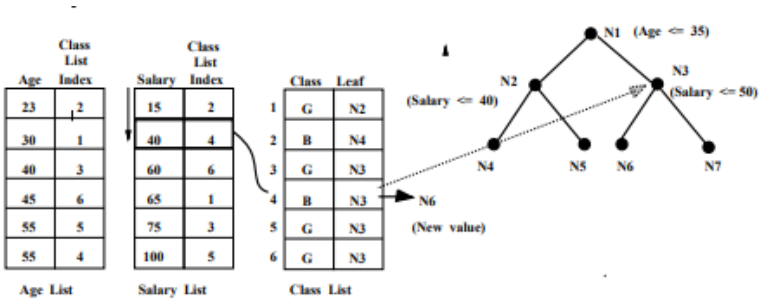


Figure 1.6: class labels update

$B : G$ class. This can be easily checked in the salary attribute list linked to class list searching through entries with leaf node index $N3$.

After finding the best split value new children nodes are created and all entries belonging to the partition of a parent node need to have their leaf indices in class list updated based on test to the splitting value of a parent node.

In figure 1.6 leaf indices are being updated after split in $N3$ based on attribute salary [16]. New value of fourth entry in class list is being updated to leaf index $N6$ after test to the splitting value.

1.3.8.5 Tree pruning

Authors of SLIQ use Minimum Description Length strategy for the purpose of tree pruning. this strategy states that the best model for encoding data is the one which has the minimum sum of a cost of describing the data using a model and the cost of describing model itself [16]. Let M be the model and D the data that model encodes then the total cost of encoding, $cost(M, D)$ is:

$$cost(M, D) = cost(D|M) + cost(M) \tag{1.7}$$

Where $cost(D|M)$ is the number of bits as cost of data encoding using model M and $cost(M)$ is the cost of model description.

Chapter 2

Implementation

As the aim of the thesis states, the main focus is on comparison between two implementations of a decision tree and their use for alternative online learning meaning not learning on additional data but rather using new features.

In this chapter two implementations of decision tree algorithms are introduced.

- Baseline model using C4.5 algorithm
- SLIQ model + memory saving

Each of these models have their respective parts as offline and online implementations, where offline stands for whole tree built on a data set and online, having an additional method to add a new feature and retrain the tree with data set containing this new feature to resemble online training on a stream of features.

The only hyper-parameter of these models is maximum depth of a tree as among decision trees is the most common one and for basic purpose of this thesis to compare training time of these algorithms is essential how well they fare against both shallow and deep trees. It also ensures that if high number for depth is given as input, that one implementation doesn't create tree of a different depth. The requirement of these trees is that they are identical.

Other hyper-parameters such as splitting threshold and criterion would need to be strictly identical therefore are constants for both implementations and are

The very concept is implemented using python programming language, although such implementation can be done easily using various programming languages python was used as it is one of the most popular programming languages regarding machine learning.

2.1 Offline tree

2.1.1 Baseline

Offline representation of a Baseline model is implemented using *DecisionTree* class, where its methods are based on a popular implementation from python sci-kit library. These methods include:

1. Fit
2. Predict
3. Score

2.1.1.1 Fit

Fit method builds the tree from scratch using training data and maximum depth as a input. Training data input is split into two arguments as *train_x* and *train_y*, where *train_x* are data points without the column that is a target variable and *train_y being* the target variable.

Nodes in this regard are objects of a special class called *TreeNode* which has a default purpose of a node in a decision tree to hold information about the splitting value and attribute which this value is linked to, it's left and right children and also information, whether it is a leaf node or not for simple test whether to try to crawl through that node further or not.

This method is a starting point that initializes depth of the tree either maximum or current, creates the root node and calls recursive method *built_tree* which has node, in the case of *fit* method the root node, as the input.

The *built_tree* method represents standard C4.5 algorithm mentioned in section 1.3.3.4. Algorithm is used only for numeric features, as all the data used in this thesis have only numeric features. Implementations can be upgraded for nominal features in future for more detailed approach. For each column and for each of theirs value this method evaluates and determines better split based on information gain using entropy. Best split information is stored with all the data that has been split based on test into two partitions. If the gain exceeds threshold new children nodes are created and *build_node* is called on each child node with its partition. This simulates depth-first exploration of nodes as it always calls method on most left child to expand.

2.1.1.2 Predict

Simple method that creates list of prediction of target variable based on input data *train_x* similar as in section 2.1.1.1. For each data point tree is traversed based on test of data point's attribute values compared to splitting values in a node. List of predictions of target variable in unlabeled data set is given as the output of the method.

2.1.1.3 Score

Score method compares predictions of given input *train_x* to input of the target variable and calculates simple accuracy based on correct predictions. The formula of precision is as follows:

$$Precision = \frac{C}{D} * 100 \tag{2.1}$$

In equation 2.1 C stands for number of correctly predicted data points divided by the number of all the data points, D. Output is given as a percentage, therefore is multiplied by 100.

2.1.2 SLIQ

SLIQ implementation follows similar structure to the baseline counterpart methods to build tree and predict data. Class implementing SLIQ algorithm is called *SliqTree* which uses objects *SliqNodes* as bulding blocks of the tree. *SliqTree* is represented as an array of nodes where each node has the information about parent and children indices in the array.

These *SliqNodes* hold information about splitting index, value and attribute which the value belongs to as well as information about left and right child, the same as in the *TreeNodes* used for Baseline model. For the goals of the thesis to reduce the number of unnecessary splits, *SliqNodes* need to hold information about their update status. This update status will be further explained in the section 2.2 where online learning feature of both implementations will be introduced. Each node holds indices to data points that belong to its partition.

SLIQ algorithms described in 1.3.8 traverse values in sorted list where these values often belong to different nodes and after traversal of the attribute list each leaf node is updated with the best possible split out of the attribute. With nodes being updated randomly, *SliqNodes* also hold information about best current split and its partitions. *TreeNodes* in Baseline model store partitions locally in a built_tree method as only one node is processed at a time.

The main methods of the SliqTree are:

1. Fit
    - build_tree
    - best_split_per_attribute
    - update_histogram
    - check_growth
2. Predict
3. Score

```
for key in self.left_count:
            self.left_count[key] = 0
            self.right_count[key] = 0
for l, r  in itertools.zip_longest(left,right):
            if(l):
                self.left_count[class_list.iloc[l[1],1]] += 1

            if(r):
                self.right_count[class_list.iloc[r[1],1]] += 1
```

Listing 1: Iteration optimization of histogram

Predict and Score methods work the same as in Baseline implementation in section 2.1.1 with the only difference being traversing through the tree as the nodes of the tree are stored in an array not as the path from root to leaf.

The main difference is in the Fit method that creates and sorts attribute lists and initializes class list as described in section 1.3.8. After initialisation *build_tree* method is called only once which controls the whole build of the tree. in this method children of nodes are created tree is checked whether more node expansion is needed. *Check_growth* method has purpose to determine whether all nodes are either expanded or are considered leaf nodes that no longer need expansion.

The base of the algorithm is implemented in *best_split_per_attribute* and *update_histogram* methods, where the first one implements traversal through attribute lists and the latter evaluation of a split and its splitting index using information gain and entropy, while also update of a histogram in each leaf node. Updating histogram is optimized in a way that testing values and distributing classes uses two iterators one going in attribute list for values lower than testing value and one for higher values.

In listing 1 class attributes *left_count* and *right_count* are dictionaries with keys being unique values of target variable or class and as a pair they represent histogram at each node. *left* and *right* stand for left and right subset of values that has been split by the splitting value in the current evaluation which means that left subset consist of all value in attribute list that are lower than splitting value and right subset of higher values as the attribute lists are sorted. Attribute lists also store information about indices of individual values to their respective position in class list. Once class has been accessed, it is used as a key into histogram and updates distribution of the class.

2.2 Online tree

To modify these implementations for the purpose of online learning on a stream of features, new method called *add_feature* is introduced. The goal

```
def add_feature(self, data, column_name):
        self.root = TreeNode()
        self.tree_depth = 1
        self.train_x[column_name] = data
        self.fit(self.train_x, self.train_y, self.max_depth)
```

Listing 2: add_feature for Baseline model

is to relearn trees with new features added one by one, to be later compared in regard to their individual training time.

2.2.1 Baseline model

Baseline implementation uses *add_feature* method with input being new column of a data set as a initialization of a new model and then starting a new *fit* method with a data set where new column is added.

The implementation of the method can be seen in listing 2 where *data* is the list of values that has the exact length as the number of rows of a input data set and a *column_name* as the name of the attribute of these values.

Root of the tree is set to a new node which makes traversing to the previously created nodes during training unavailable, simulating creation of a new tree. As the data set given previously in a training is already split into *train_x* and *train_y* described in section 2.1.1.1 data in a new feature is added to the *train_x* data set and new *fit* method is started.

2.2.2 SLIQ

For the SLIQ implementation of *add_feature* method a new approach is introduced. The baseline approach created a new tree with the data set that was updated with a new feature. The SLIQ implementation follow this idea only slightly. The difference is that in SLIQ implementation only tree branches, where the new feature's values would take lead as the splitting values are changed.

Example can be shown in figure 2.1, where in node B, new feature could provide better information for more accurate split but not in the other nodes. Improvement would be not to build whole tree from scratch but identify nodes that need to be updated start building new paths from them. Once one node is updated it is essential that the whole subtree of descendants beneath this node, in the example nodes E and D, is built anew, as the recent split using new feature can change partitions continuing in node's children. The idea is to preserve as much of the original tree as possible. For that purpose each node holds information about need of an update mentioned in the offline section 2.1.2 .

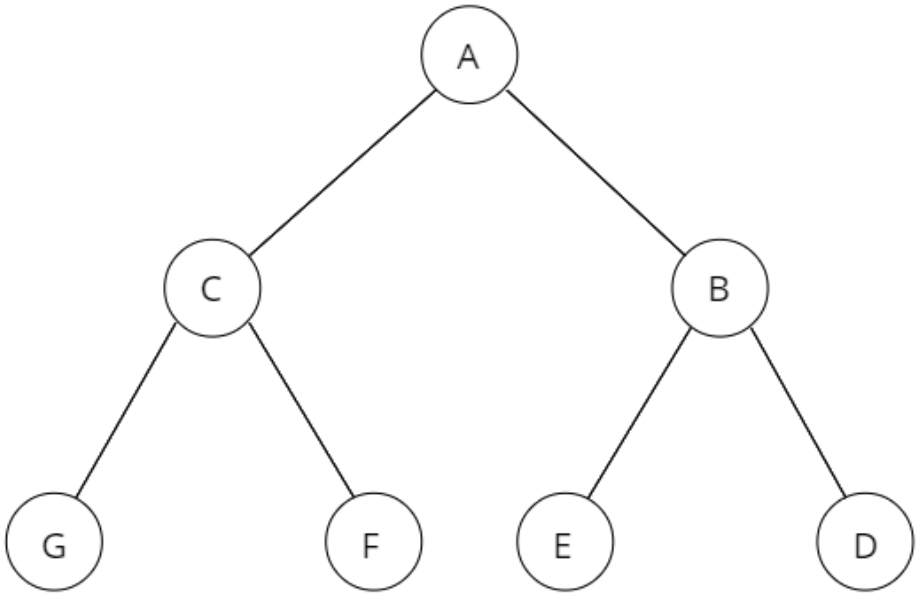


Figure 2.1: SLIQ Optimisation example

2.2.2.1 Realisation of a add_feature method

Realisation of the method can be separated into several steps:

1. Creation of a new attribute list
2. Checking information gain of possible new splits
3. Finding descendant nodes of a node to be updated
4. Removing descendant nodes
5. Starting new build_tree method

First step works as a initialisation for new building phase by creating and sorting new attribute list, similar to *fit* method in offline part of the SLIQ tree implementation. In addition each node's update status needs to be set

```
if(node.update):
    self.find_all_descendants(nodes_to_update_per_node, node)
    for ind in nodes_to_update_per_node:
        self.class_list.loc[self.class_list['leaf'] == ind, 'leaf'] = node.id
    nodes_to_update.update(nodes_to_update_per_node)
    node.evaluated = False
    node.leaf= False
```

Listing 3: Class list update after finding descendants

to *False*. This is required as each time method is called all nodes must be considered for an update.

Second step describes traversal of node list and evaluation of new splits using new attribute and its values. As each node is keeping information about its partition for the purpose of switching between nodes during attribute list traversal, this step only calculates splitting index using information gain formula 1.2 and updates nodes if an improvement is found. The difference is that each node updated during these evaluations have their update status changed.

After traversing the tree The closest to root nodes that need update will be split on new splitting value creating new partitions. Class list holds information about data points their respective partition in a form of leaf node index, therefore these values must be reverted to the node that is being updated, if the indices belong to the children of such node. Listing 4 shows the process of finding descendants of node that has its update status changed. In this process two sets are used:

1. nodes_to_update_per_node

2. nodes_to_update

The first set stores descendant nodes of a currently evaluated node while the second one has information about all descendant nodes in the tree. After this process all nodes that need their split updated are again considered for split evaluation and are no longer considered leaves if it is the case that leaf node gained new split with the addition of new attribute list.

Listing 3 describes finding the descendants as recursive method which has stopping rule set to the leaf node. If node has no children then finding of descendants of such branch is stopped. The input set is *nodes_to_update_per_node*, which shows that after method, whole sub-tree of a input node is added to the set.

Next to last step removes all nodes from the tree that has been descendant to updated nodes. All of these indices are in the *nodes_to_update* set and afterwards depth of the tree is set to the depth of the most deep remaining node and last step, new *build_tree* method is called. This is possible thanks

```
def find_all_descendants(self, nodes_to_update, node):
    if((not node.left_child)  and (not node.right_child)):
        return
    left = self.tree[node.left_child]
    right = self.tree[node.right_child]
    nodes_to_update.add(node.left_child)
    nodes_to_update.add(node.right_child)
    self.find_all_descendants(nodes_to_update, left)
    self.find_all_descendants(nodes_to_update, right)
```

Listing 4: find_all_descendants method

to *check_growth* method as each of the nodes that has been updated with new attribute will be considered by the method for new split evaluations.

Chapter 3

Experiments

In this chapter testing of implemented models is described with results of time consumption as well as limitation to the optimisation of decision tree building leading to limitations in testing itself. The main focus is to show how well can modified decision tree algorithm fare against one of the common algorithms in offline learning style but also in different approach to online learning. As there are always ways to optimise these models and also how to approach the use case of training decision trees on new features for the same data, these results should be viewed as a guideline where to start and to expect from these models used for specific use-case.

The main focus of this thesis is that hypothesis, that retraining tree with added features using using common algorithm like C4.5 is as good as using SLIQ tree method with simple tree saving management. In previous chapter both implementations used for testing in this chapter were introduced.

The first requirement of this approach is that time must not be affected by another variable only by the algorithms themselves. Therefore all the hyper-parameters are constant except for the maximum depth of the tree that is used as mark that both implementations end in the same depth. In Both implementations information gain and entropy were used as a criterion for determining better split and pruning of the tree as a way of optimisation was omitted. The reason is that optimisation of a trained tree is not the goal of this thesis but rather finding a way how to deal with slow speed of retraining trees over and over again.

The second requirement is that The accuracy of both implementations is identical in the meaning that for the same data both grant same prediction results. By achieving such goal the variables left to be tested are time of training and memory usage of these models. Memory usage is not the goal of this thesis therefore won't be included in results.

3.1 Protocol

In this section machine learning techniques used for testing will be described with mention to optional approach and the reason using either of them. The main focus will be on these four aspects:

- Maximum depth
- Time function
- Feature management
- Data management

3.1.1 Maximum depth

Maximum depth of the tree during experiments which is given as a input to the tree was set to 6 as deeper trees tend to over-fit easily. Rather than having deeper trees usually more trees individually trained trees on the same data set take a vote for the target variable. Also having very shallow trees makes difference between models less noticeable.

3.1.2 Time Function

For the purpose of training time measurement of implementations *time* python module is used with its function also called *time()*. This function returns a floating point number in seconds that has passed since the **epoch**. The epoch is a specific date where counting the time starts and it is platform dependent. For example for unix systems it is January 1, 1970, 00:00:00 (UTC). The return of the function on Unix system would be number of seconds, type float, that has passed since that date.

In this work this feature was used as it can be simply used for measuring time spent training models where time of training is gained by subtracting time after training by time before training. This measurement is optimized using system calls to get precise time, therefore the only hindrance can be the subtraction between two floats which might not give the most precise answer, but for the purpose of this work is enough to justify differences between training speeds of trained models.

3.1.3 Feature management

The addition of features has been managed manually as there is far less work with addition of a new features one by one than by adding new data points the same way. Implementations support only online version without option for batch learning therefore tree must be retrained after each new feature. The reason is that batch learning usually works better with new data points

where there is huge amount of rows to be added so it is more time efficient to add a small subset of new data called batch rather than optimising tree after each new row. Data sets usually don't have many features and importance of each feature added is best recognised while testing model after each individual addition, therefore it would seem to be the right course of action.

In testing environment for every data set two of its features are cut off and then individually added for retraining these choices of features are made randomly and are managed manually through editing of input into training function as it resembles data pre-processing that is often made by human analysis. The importance of a feature can be based on information value it gives to the model or number of missing values in the feature.

3.1.4 Data management

Data used in training are not processed in any way except for the features that are cut just to be added later as a part of online training feature. After download data is randomly split into their training and testing parts in the ratio of 75:25, 75 for training and 25 for testing using script included in CD enclosed with this work.

After the split these data sets are saved into individual files and are used for their role in the testing process. The reason for this approach is simplicity of presentation of the training process. the main focus is on the implementations of training algorithms and their test results in simply set environment.

3.2 Algorithms

The only classifiers compared are Decision tree classifiers one using C4.5 algorithm and the second one using SLIQ algorithm enhanced with memorization of already better splits. Both implementations are used only for numeric features. In the early development there was an idea of using popular sci-kit implementation of decision tree as a benchmark, but as it is highly optimised using pruning and other features the comparison would be affected by features other than implementation of a building phase only.

3.3 Data sets

Data sates used in this work are selected based on these specifications:

- Data type of features
- Number of features
- Number of instances
- Number of classes of target variable

- Missing values in instances

The data type of all features has to be numeric as the implementations described in previous chapter handle only numeric values. The modification for nominal features is beyond the scope of this work but data pre-processing using one-hot encoding which is a technique to change nominal feature into numeric one can be used for purpose of testing models even with nominal features.

Number of features is set to be lower than 15 even if it isn't high restriction having more than 15 features in a data set used for decision trees is not very common. Even if the data set has more features usually only few of them are selected.

The number of instances in all data sets doesn't exceed 1000 due to implementations having each individual unique value to be considered for a split makes computation time much higher than it usually is. As the features are continuous they have many unique values. This problem is solved by having only few values to be considered for a split, for example every tenth value or random values of one attribute to avoid time inefficiency. These implementations use every value, therefore training takes too much time even for 1000 instances. This doesn't affect the purpose of this work as the goal is to show that one implementation is faster than the other not the optimal one but an improvement to the other method.

All tree models assume only binary target variable, that makes the number of classes for the target variable to be set to number 2.

Even if the algorithms in their descriptions are able to handle missing values, implementations focus on training phase in set environment not their ability to be adaptive to complications during training. Data sets with missing values are not included in this experiment.

3.3.1 Open-ML

Open-ML is an open service where people add various data sets for the public use. These data sets range from car specifications, housing to patient care. People are able to create their own data sets or find data sets that best suit their use-case based on filters available to find data sets with certain specifications. Specifications of data sets used for the experiment mentioned earlier are set as a filter into the service.

3.3.2 Data sets

The statistic description of data sets is show in table 3.1. All except two of the data sets, namely *first* and *sample_data_num_2*, are downloaded from the Open-ML service. These two were created during implementation of algorithms for initial testing purpose whether the models work as intended. *first* functions as random data generator, where there are random values spread

across attribute and random class distribution of target variable with no intended pattern to see, if models work correctly even on random data. The other one is created with pattern involvement, which models should find and build tree around this information if they work properly. The creation of these data sets is based on a script *DataCreator* enclosed with this work.

Data set	**Number of features**	**Number of rows**
bodyfat	15	252
sample1	10	462
sample2	9	320
fri_c0_250_10	11	250
fri_c3_500_10	11	500
diggle_table_a1	5	48
diggle_table_a2	9	310
vowel	11	990
housing	14	506
prvy	6	100
sample_data_num_2	6	30
autoPrice	16	159
rmftsa_ladata	11	508
chatfield_4	13	235
dataset_37_diabetes	9	768
strikes	7	625
chscase_census4	8	400

Table 3.1: Data sets

3.4 Results

Training time results of Baseline and SLIQ implementations are show in tables 3.2 and 3.3 respectively. These tests are based on two randomly chosen features that were first left out and later added into the data sets.

Though there are many more combinations of which features are part of this experiment, the results were mostly the same with exceptions being that feature has so much value to data that predictions are made basically thank to the feature alone. This was true probably in one or two out of many combinations that were tried during experiments. These tables therefore can be viewed as sample result of the testing.

With baseline model the trend was that each added feature increased training time, which is highly expected because baseline model is retrained from scratch with every new feature and having more values to consider for split always increase computation time.

With SLIQ model this was not the case in many trails. SLIQ with feature of changing only worse splits compared to new feature often meant, that the training time of adding new feature is much lower than the first building phase, as much of the tree is preserved and only few branches are changed. This scenario can be seen using example data set *sample1*, where in baseline model results starting iteration took almost 87 seconds, first addition 98 and second 114 seconds. In the meantime SLIQ model of the same dataset took 7, 4 and 2 seconds for the respective iterations.

For the training time of these models itself SLIQ was in most of cases constantly about 8-10 times faster than the baseline model, mainly thanks to less time spent on sorting values of attribute which in SLIQ implementation is only done once.

As for the accuracy of predictions in individual implementations, these are not optimized in a sense of pruning and with maximum depth being the only hyper-parameter to be as input into creating a tree, accuracy on testing data is between 60 and 90% on average throughout all data sets. The positive result is that trees that were offline trained on 10 features have the same structure therefore same prediction as trees that were trained on 8 features and later online retrained using the 2 that remained, even when these left out features brought much value and tree had to be changed.

During evaluation of results for accuracy between the implementations, most feature combinations gave the same accuracy results for both baseline and SLIQ implementation, which is what is expected, but there were some cases where accuracy was slightly off with accuracy difference no higher than 5%.

Charts 3.1, 3.2 and 3.3 show how number of instances is affecting training time in both implementations used for from scratch offline training in the first chart and then for each addition of new feature individually. From the trendline for baseline implementation compared in these three charts, it can be seen, that each addition of new feature increases average training time, whereas SLIQ version in many cases actually has its training time decreased as the tree saves the branches that doesn't need to be retrained.

From the behaviour of SLIQ line in consecutive charts, adding features one by one optimizes training time in a way where eventually adding new features doesn't change the tree at all. In baseline version as the tree is always retrained, there may be time waste on recreating the same tree even if new feature is added.

Number of instances in offline as well as online simulation using addition of two features affects much more heavily baseline implementation in increased training time, than it does for SLIQ version, this is mostly thanks to SLIQ algorithm sorting all values only once. With these results it can be seen, that SLIQ implementation handles more data much better than the baseline version.

Data set	Base training time (s)	add_feature_1 training time	add_feature_2 training time
bodyfat	34.90	37.94	40.33
sample1	86.96	97.96	113.70
sample2	59.73	71.59	82.13
fri_c0_250_10	28.01	31.54	34.28
fri_c3_500_10	108.71	123.59	135.76
diggle_table_a1	0.31	0.46	0.58
diggle_table_a2	29.01	33.93	40.35
vowel	301.63	342.83	378.37
housing	160.35	174.13	189.57
prvy	5.86	5.88	7.41
sample_data_num_2	0.43	0.59	0.52
autoPrice	24.74	26.49	27.83
rmftsa_ladata	119.28	129.92	147.63
chatfield_4	28.68	32.27	35.09
dataset_37_diabetes	213.56	246.92	290.86
strikes	94.79	118.98	143.58
chscase_census4	51.40	61.67	71.43

Table 3.2: Baseline model results

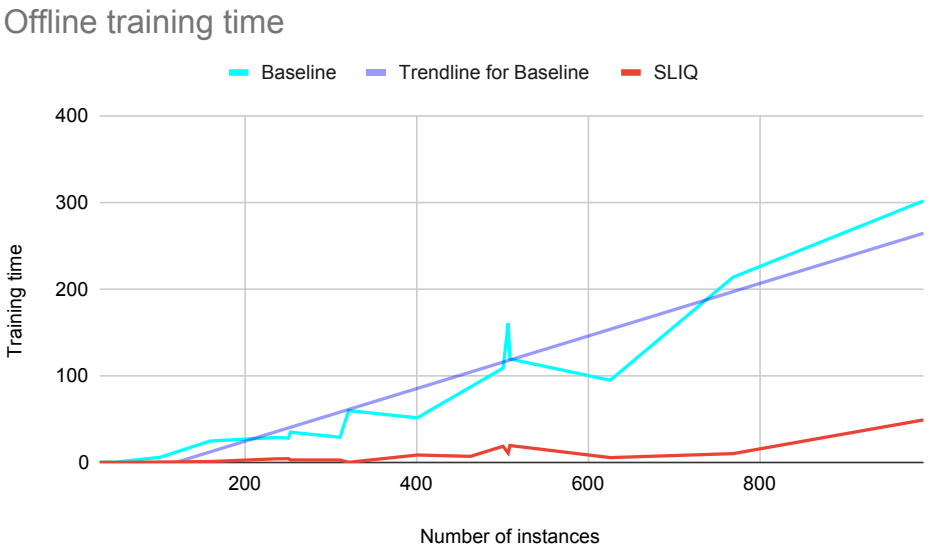


Figure 3.1: Offline training time comparison

Data set	Base training time (s)	add_feature_1 training time	add_feature_2 training time
bodyfat	2.86	1.60	1.66
sample1	6.98	3.64	1.99
sample2	0.12	0.07	0.06
fri_c0_250_10	4.31	2.80	1.48
fri_c3_500_10	18.60	6.37	7.47
diggle_table_a1	0.03	0.02	0.08
diggle_table_a2	2.75	1.37	1.49
vowel	49.02	40.54	25.04
housing	10.84	7.54	4.88
prvy	0.42	0.99	0.27
sample_data_num_2	0.03	0.02	0.03
autoPrice	1.01	0.65	0.34
rmftsa_ladata	19.49	27.77	8.28
chatfield_4	4.09	1.33	6.29
dataset_37_diabetes	10.13	2.94	10.14
strikes	5.52	1.95	3.65
chscase_census4	8.53	6.31	4.25

Table 3.3: SLIQ model results

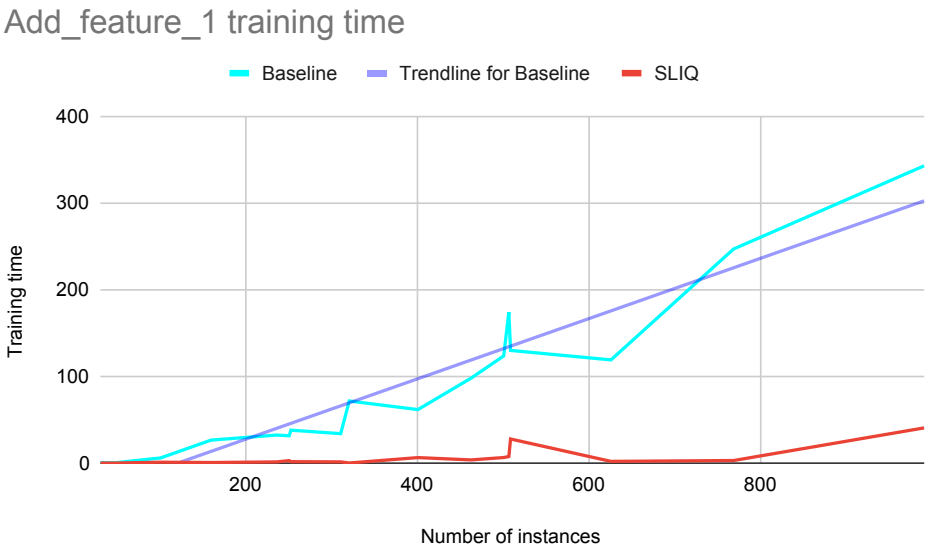


Figure 3.2: Add_feature_1 training time comparison

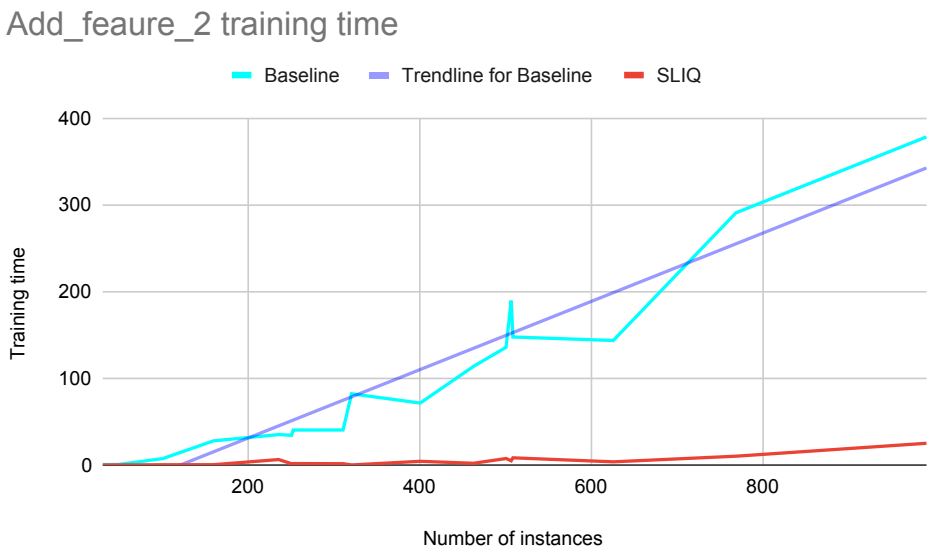


Figure 3.3: Add_feature_2 training time comparison

3.5 Statistical evaluation

Using every data set and many combinations of features to consider for adding SLIQ tree performed much better than baseline version therefore in this statistical evaluation it is assumed that the hypothesis, that using baseline version is equally time-efficient than using SLIQ version is likely to be rejected as SLIQ version performed much better in regard to time spent training.

Using sign-test where in all 70 different combinations of data sets SLIQ version performed quicker, null hypothesis, that the baseline version is as fast as SLIQ version can be rejected in favor of alternate hypothesis that SLIQ version performs better.

3.6 Discussion and future work

This section refers to possible direction of future work regarding the topic of implementing decision trees for the task of training on stream of features. The first idea that comes to mind would be creating a more detailed and varied environment for tests, especially using techniques such as k-fold validation.

The goal of this work was to test speed of the models but there is more to consider for model to be more suitable. For example there may be case where speed of the model is not as important as memory used in process, therefore it may be viable to test for memory usage of the algorithms. The

tests themselves were created with simplicity in mind, using few features of data set that were firstly manually cut off and then later added into data set. It may be very efficient if going for a large scale test to be able to automatize this process which can create even more possibilities to compare more decision tree implementations and even compare the best of decision trees to other classifiers in this manner.

Next possibility is optimization of models. The focus of this work was to test only training phase of each model trying to leave out optimizations such as pruning to not create more place for human error which would affect training times of models. The future steps could definitely include pruning trees and inclusion of more hyper-parameters variation like gini-impurity and different splitting index thresholds, to see the behaviour of models in more realistic environment.

The optimization of models also include handling values especially categorical values and missing values. In this work all values are numeric with missing values left out and each individual value is considered for a split. As this can be provide more detailed training time results, it is not used in scenarios where data set is large. These times being able to process huge amount of data is important, therefore evaluation of a split using every unique value is very likely to be counterproductive.

Conclusion

This work focus was on decision trees and a different approach to online learning specifically to learning on a stream of new features rather than new data with implementation of a algorithm that would provide higher training speed than more common algorithms as the trees needed to be constantly retrained with new features.

The first chapter explained common terms used in data science such as machine learning, classification and common classifiers, that were described briefly, with more detailed description to decision tree its hyper-parameters and common algorithms for building a decision tree, which ended with the explanation and theory behind proposed algorithm used for higher training speed called SLIQ.

The second chapter focused on the individual implementations of baseline model using popular tree building algorithm C4.5 and improved model using SLIQ algorithm. Both these implementations were upgraded with a method that is capable of retraining the tree in case new feature to existing data is added. In baseline tree retraining is made from scratch with the same data but with new feature already included in it, while in SLIQ version new modification is introduced, where as much as possible of original tree is saved and only parts of the tree, where new feature gives more information, are changed.

The third chapter described experiments that were performed on these models mainly concerning training speed as the goal of this work was to prove that new modified model using SLIQ algorithm is much more time efficient than the baseline one while preserving the same accuracy for both of these algorithms. Results concluded that SLIQ version, even though both implementations are equally non-optimised, performed much better in terms of training speed using variety of training and testing data sets, while both implementations preserved same testing accuracy and tree structure in most cases.

Results of experiments shown in this work can be used primarily for deeper

analysis where this can be used as directional help of where to start and to expect to optimize creation of variety of decision trees used for specific use-cases.

Bibliography

[1] Alpaydin, E. *Introduction to machine learning*. Adaptive computation and machine learning, MIT Press, 2004, ISBN 978-0-262-01211-9.

[2] Kononenko, I. Machine learning for medical diagnosis: history, state of the art and perspective. *Artif. Intell. Medicine*, volume 23, no. 1, 2001: pp. 89–109, doi:10.1016/S0933-3657(01)00077-X. Available from: `https://doi.org/10.1016/S0933-3657(01)00077-X`

[3] Libbrecht, M. W.; Noble, W. S. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, volume 16, no. 6, 2015: pp. 321–332.

[4] Pant, A. Introduction to Machine Learning for Beginners. Jan 2019, [Cited 2020-4-29]. Available from: `https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08`

[5] Kotsiantis, S. B. Supervised Machine Learning: A Review of Classification Techniques. *Informatica (Slovenia)*, volume 31, no. 3, 2007: pp. 249–268. Available from: `http://www.informatica.si/index.php/informatica/article/view/148`

[6] Myles, A. J.; Feudale, R. N.; et al. An introduction to decision tree modeling. *Journal of Chemometrics: A Journal of the Chemometrics Society*, volume 18, no. 6, 2004: pp. 275–285.

[7] Gupta, P. Decision Trees in Machine Learning [online]. May 2017, [Cited 2020-2-13]. Available from: `https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052`

[8] Ghahramani, Z. Unsupervised learning. In *Summer School on Machine Learning*, Springer, 2003, pp. 72–112.

[9] Kaelbling, L. P.; Littman, M. L.; et al. Reinforcement learning: A survey. *Journal of artificial intelligence research*, volume 4, 1996: pp. 237–285.

[10] Duda, R. O.; Hart, P. E.; et al. *Pattern classification, 2nd Edition*. Wiley, 2001, ISBN 9780471056690. Available from: https://www.worldcat.org/oclc/41347061

[11] Mehlhorn, K.; Sanders, P. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008, ISBN 978-3-540-77977-3, doi:10.1007/978-3-540-77978-0. Available from: https://doi.org/10.1007/978-3-540-77978-0

[12] Priyam, A.; Abhijeeta, G.; et al. Comparative analysis of decision tree classification algorithms. *International Journal of current engineering and technology*, volume 3, no. 2, 2013: pp. 334–337.

[13] Patel, N.; Upadhyay, S. Study of various decision tree pruning methods with their empirical comparison in WEKA. *International journal of computer applications*, volume 60, no. 12, 2012.

[14] Kohavi, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, Morgan Kaufmann, 1995, pp. 1137–1145. Available from: http://ijcai.org/Proceedings/95-2/Papers/016.pdf

[15] Shalev-Shwartz, S. Online Learning and Online Convex Optimization. *Found. Trends Mach. Learn.*, volume 4, no. 2, 2012: pp. 107–194, doi:10.1561/2200000018. Available from: https://doi.org/10.1561/2200000018

[16] Mehta, M.; Agrawal, R.; et al. SLIQ: A Fast Scalable Classifier for Data Mining. In *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings*, *Lecture Notes in Computer Science*, volume 1057, edited by P. M. G. Apers; M. Bouzeghoub; G. Gardarin, Springer, 1996, pp. 18–32, doi:10.1007/BFb0014141. Available from: https://doi.org/10.1007/BFb0014141

Appendix A

Acronyms

SLIQ Supervised learning in Quest

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source scripts
- `datasets` ..... the directory of used data sets
- `thesis.pdf` ..... the thesis text in PDF format