University of Tartu
Faculty of Science and Technology
Institute of Mathematics and Statistics

Priyush Protim Sharma

Tree-based methods in supervised learning with Estonian Health Insurance Fund data

Actuarial and Financial Engineering

Master's Thesis (30 ECTS)

Supervisor: Prof. Jüri Lember and Mark Gimbutas

Tartu 2021

TREE-BASED METHODS IN SUPERVISED LEARNING WITH ESTONIAN HEALTH INSURANCE FUND DATA

Master's thesis

Priyush Protim Sharma

Abstract

The main aim of this master's thesis work is to provide an overview of some tree-based models and to test the suitability of these models in finding the incorrectly submitted invoices received by the Estonian Health Insurance Fund. C4.5, CART and bagged CART are the three algorithms that are used to train the models and to apply binary classification with these models in order to reduce the number of invoices that must be checked manually.

CERCS research specialisation: P160 Statistics, operations research, programming, actuarial mathematics.

Key Words: Estonian Health Insurance Fund, Tree based models, Machine Learning, Statistical Learning, R.

PUUPÕHISED JUHENDATUD ÕPPE MEETODID EESTI HAIGEKASSA ANDMETE NÄITEL

Magistritöö

Priyush Protim Sharma

Lühikokkuvõte

Käesoleva magistritöö põhieesmärk on anda ülevaade mõnedest puupõhistest masinõppe meetoditest ja testida nende sobivust Eesti Haigekassa andmestikus alusetult esitatud raviarvete tuvastamiseks. Valitud meetoditeks on C4.5, CART ja bagged CART, mida kasutatakse binaarseks klassifitseerimiseks, et vähendada käsitsikontrolli suunatavate raviarvete arvu.

CERCS teaduseriala: P160 Statistika, operatsioonanalüüs, programmeerimine, finants- ja kindlustusmatemaatika.

Märksõnad: Eesti Haigekassa, puupõhised mudelid, masinõpe, tehisõpe, R.

Contents

1 INTRODUCTION

Every time an insured person visits a medical specialist, the hospital invoices the Estonian Health Insurance Fund (EHIF) for that visit. The treatment invoice contains all health services and diagnoses of the patient relevant to that visit. The EHIF checks the treatment invoices and pays the required amount for each genuine invoice. This includes routine checks for a certain type of suspicious treatment invoices submitted during the visit. In this way, the pre-selected invoices are reviewed manually, and if necessary, the hospitals are asked for further explanations and additional documents. Most of these invoices turn out to be justified, while a tiny percentage of them turn out to be faulty. The practical aim of the thesis is to apply binary classification models to reduce the number of invoices that must be checked manually.

Tree-based methods are the choice of algorithms for this master thesis to model the classification. In the first chapter, there is an introduction to pattern recognition and the mathematics behind the intuition of the classification problem is given.

There is a detailed description of the decision tree and two of the main tree algorithms namely CART and C4.5 are discussed in the second chapter. In the third chapter, the topic of discussion moved to ensemble learning which is basically converting a weak learner into a strong learner. Bagging, an ensemble learner is discussed in this chapter. The description of the data, the analysis, and the classification based on the above-mentioned algorithm can be seen in the fourth chapter.

The practical tool used for this master's thesis includes R studio, Python and

MS Excel. This master's thesis is written in LATEX using overleaf software.

The author would like to thank Estonian Health Insurance Fund for providing the data. The author would also like to thank both supervisors, Prof Jüri Lember and Mark Gimbutas for their guidance and advice.

2 PATTERN RECOGNITION

We live in a data-centric world and this data comes in a wide variety of forms: numeric, textual (structured or unstructured), audio and video signals. Understanding and making sense of this vast and diverse collection of data (identifying patterns, trends, anomalies, providing summaries) requires some automated procedure to assist, and this is where the role of pattern recognition comes into play.

In short, pattern recognition or classification is the process of predicting the class of given data points. Classes are also known as targets or labels or categories. The process of pattern classification involves building classifiers capable of automatically constructing methods for distinguishing between different exemplars based on their differentiating patterns.

The uses of a pattern classifier are to provide:

- A descriptive model explaining the difference between patterns of different classes in terms of features and their measurements.
- A predictive model that predicts the class of an unlabelled pattern.

There are two main approaches to statistical pattern recognition: supervised classification and unsupervised classification. Labelled data is used in the supervised classification, while in the later, the data are not labelled, and the goal is to find groups in the data and the features that distinguish one group from another. This master thesis will focus on the supervised learning approaches.

Some of the common examples of classification are:

- Detection of spam emails

- Prediction loan default
- Medical diagnosis
- Facial recognition

Given there is a set of measurements obtained through observation and is represented as a d-dimensional pattern vector $x \in \mathbb{R}^d$. Each element in the vector is known as a **feature** and the vector itself as feature vector. The unknown nature of the observation is called a **class**. We can identify every class with a number from a set Y which can take any value range from 0 to $k-1$, altogether k classes, depending on the nature of the problem.

$$Y = \{0, 1, \ldots, k-1\}$$

For example: An email spam classifier will have two classes: spam and not spam. Similarly, we can have 3 classes for a credit risk classifier: low risk, medium risk, and high risk.

We restrict our attention to binary classification for our simplicity and $Y = \{0, 1\}$ in our case.

In pattern recognition, one creates a function $g(x) : \mathbb{R}^d \to \{0, 1\}$ which represents one's guess of y given x. Here, the mapping g is called a classifier.

Formally, a classifier is a function:

$$g(x) : \mathbb{R}^d \to Y. \tag{1}$$

It can be rewritten as

$$g = \sum_{i=0}^{k-1} i \, I_{C_i}(x), \tag{2}$$

where I_C is the indicator of set C, i.e

$$I_{C_i}(x) = \begin{cases} 1 & \text{if } x \in C; \\ 0 & \text{if } x \notin C. \end{cases}$$

Hence g defines a partition $\{C_0, C_1, \ldots, C_k - 1\}$ of the set $\mathbb{R}^d$ where class i will be assigned to an object if and only if the feature vector belongs to the set C_i (Lember, 2012).

2.1 Loss and Symmetric Loss

The classifier g depend on the loss function. Loss functions play an important role in any statistical modelling. They define an objective which the performance of the classifier is evaluated against and the parameters learned by the classifier are determined by minimizing a chosen loss function. The loss function can be defined as :

Definition 2.1. ***The loss function***

$$L : Y \times Y \to \mathbb{R}^+$$

assigns to every pair (i, j) loss that occur when the object belonging to class i is classified as belonging to class j.

A good classifier is such that the loss $L(y, g(x))$ is small (Lember, 2012).

2.1.1 Symmetric Loss

The loss function that can be most commonly seen in a classifier is symmetric or 0-1 loss. We have $L(i, i) = 0$ for symmetric loss and the loss of misclassification is always the same. Hence, the symmetric loss is

$$L(i,j) = \begin{cases} 0 & \text{when } i = j, \\ 1 & \text{when } i \neq j. \end{cases}$$

2.2 Risk and Empirical Risk

Since the input vectors x are not known for sure, they are considered as random. Every random vector has a distribution that is uniquely characterized by its distribution function. In the following, we are considering F be the distribution function of feature vector.

Definition 2.2. ***The risk*** *of classifier g is the average loss over the joint distribution $F(x,y)$:*

$$R(g) = \int L(y,\ g(x))dF(x,y). \tag{3}$$

If the distribution of $F(y,x)$ is known, we choose the classifier that minimizes $R(g)$ over a class of classifers $\mathcal{G}$. However, in the real world, it is not possible to know the distribution $F(y,x)$ and we have the dataset D_n instead. The data set $D_n = \{(x_i, y_i), i = 1, ..., n\}$ where x_i are the data samples and y_i the corresponding class labels. Now, every sample from D_n can be considered as the empirical distribution with empirical distribution function F_n.

$$F_n(x,y) = \frac{1}{n}\sum_{i=1}^{n} I_{\{x_i \leq x, y_i \leq y\}}$$

The empirical distribution function F_n is an estimate to unknown distribution function F. It can be shown that F_n is a good estimate of F with the help

of Glivenko-Cantelli theorem.

Theorem 1. ***The Glivenko-Cantelli Theorem***

Let $X_1, \ldots, X_n$ be a collection of i.i.d. random variables with distribution function F, and let F_n denote the empirical distribution function. Then as $n \to \infty$,

$$P\left[\sup_x |F(x) - F_n(x)| \to 0\right] = 1$$

or equivalently

$$P\left[\lim_{x\to\infty} \sup_x |F(x) - F_n(x)| \to 0\right] = 1$$

that is, the convergence is uniform in x.

The same can be written for $F(x, y)$ and $F_n(x, y)$ as :

$$P\left[\sup_{x,y} |F(x,y) - F_n(x,y)| \to 0\right] = 1$$

If the unknown distribution $F(x, y)$ is replaced by its empirical version $F_n(x, y)$ while calculating the risk $R(g)$, we can obtain the empirical risk of the classifier.

$$R_n(g) \;=\; \int L(y, g(x)) dF_n(y, x) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, g(x_i)). \tag{4}$$

When L is symmetric, then the empirical risk or the empirical error is given by

$$R_n(g) \;=\; \frac{1}{n}\sum_{i=1}^{n} I_{\{y_i \neq g(x_i)\}}.$$

Now, the classifier which minimizes the empirical risk over $\mathcal{G}$ can be found with the help of empirical risk minimization (ERM) principle. When loss is symmetric, then the empirical risk minimization principle chooses such a classifier that minimizes the number of misclassified objects (empirical error) in training data set (Lember, 2012).

2.3 The Problem

The pattern recognition problem that is approached in this thesis is based on Estonian Health Insurance Fund(EHIF) data. There are 7903 observations with 46 features and a target variable in the dataset which are dummy coded into zeros and ones. EHIF receive millions of invoices every year and it is quite impossible to manually check each one of them. As a result, there are some discrepancies that often goes unnoticed and incorrect invoices are remitted by Haigekaissa. The main goal of this thesis is to train a classifier so as to find the few incorrect invoices among the plethora of correct invoices and reduce the number of invoices that must be checked manually. Decision trees, which includes C4.5 and CART are the initial choice of preference for this thesis. Bagging, an ensemble learning method, is also used so as to improve the performance of the base learners.

2.4 Model Assessment

Model assessment is a very important step in selecting the best possible classifier. The idea is to search for a good classifier from a set of classifiers $\mathcal{G}$. The set $\mathcal{G}$ is often referred as the model. We have the confusion matrix as the main building block for the assessment in a classification problem.

Definition 2.3. *A **confusion matrix** summarizes the classification performance of a classifier with respect to some test data. It is a two-dimensional matrix, indexed in one dimension by the true class of an object and in the other by the class that the classifier assigns.*

		prediction outcome	
		1	**0**
actual value	**1**	True positive	False negative
	0	False positive	True negative

There will be 2 x 2 matrices for binary classes where each cell gives a different insights.

- True Positive(TP): Number of observations that correctly classified as "1" or "success"
- True Negative(TN): Number of observations that correctly classified as "0" or "failure"
- False Positive(FP): Number of observations that incorrectly classified as "1" or "success"
- False Negative(FN): Number of observations that incorrectly classified as "0" or "failure"

From the confusion matrix, we can calculate different performance measures to get interesting insights about our classifiers. The mostly commonly used measures of performance are Accuracy and Error rate.

Accuracy is calculated as the number of all correct predictions divided by the total number of the dataset.

$$Accurary \ = \ \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{5}$$

On the other hand, the error is the proportion of incorrectly classified observations which is calculated using:

$$Error \ = \ \frac{(FP + FN)}{(TP + TN + FP + FN)} \ = \ 1 - Accuracy \tag{6}$$

This is similar to the empirical risk for a particular classifier in the equation (4).

These standard metrics (i.e. Accuracy and Error) work well on most problems, which is why they are widely adopted. But often it can be seen that the standard metrics become unreliable or even misleading when classes are imbalanced, or severely imbalanced. The standard metrics treats all classes equally important and as a result, it is quite difficult to get reliable measure for the dataset with class imbalance. Hence, we define two groups of metrics that may be useful for imbalanced classification because they focus on one class: Sensitivity-Specificity and Precision-Recall.

2.4.1 Sensitivity-Specificity

Sensitivity refers to the true positive rate and summarizes how well the positive class was predicted.

$$Sensitivity \ = \ \frac{TP}{(TP \ + \ FN)}$$

Specificity is the complement to sensitivity, or the true negative rate, and summarises how well the negative class was predicted.

$$Specificity = \frac{TN}{(FP + TN)}$$

2.4.2 Precision-Recall

Precision summarizes the fraction of examples assigned the positive class that belong to the positive class.

$$Precision = \frac{TP}{(TP + FP)}$$

Recall summarizes how well the positive class was predicted and is the same calculation as sensitivity.

$$Recall = \frac{TP}{(TP + FN)}$$

Precision and recall can be combined into a single score that seeks to balance both concerns, called the F-score or the F-measure.

$$F - Measure = \frac{(2 \cdot Precision \cdot Recall)}{(Precision + Recall)}$$

3 DECISION TREES

A decision tree is an classifier in the form of a hierarchical tree structure which uses divide and conquer strategy. Decision trees are one of the effective methods of supervised learning. The procedure of growing a decision tree includes dividing the data set into groups as homogeneous as possible in terms of the variable to be predicted. The growing method involves taking the set of classified data as an input and yields a tree where each end node(leaf) is a decision and each non-final node represent a test. Each leaf represents the decision of belonging to a class of data verifying all tests path from the root to the leaf.

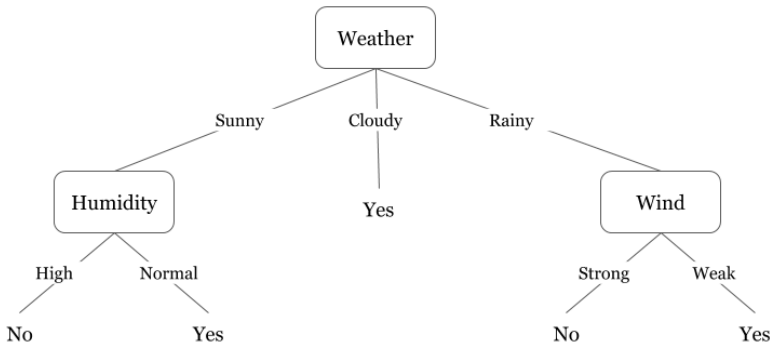

Figure 1: Illustration of decision tree

A hierarchical decision tree can be built with numerous nodes and directed edges/ branches. There are three types of node that we can find in a tree :

- **Root node:** This is the node that has no incoming edges. It is usually at the top of the tree.

- **Internal nodes:** These nodes are with one incoming edge and two or more outgoing edges.
- **Leaf or terminal nodes:** The nodes with one incoming edge.

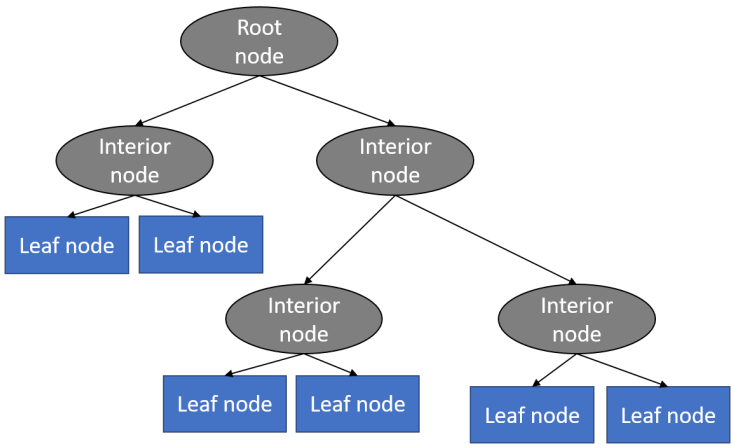

Figure 2: Illustration of nodes in a decision Tree

3.1 Construction of a decision tree

A classification tree is constructed using a labelled data set, $D = \{(x_i, y_i), i = 1, ..., n\}$ where x_i are the data samples and y_i the corresponding class labels. The tree construction begins at the root node and goes on to successively partitioning the feature space.

The construction involves three steps :

- **Selecting a splitting rule for each internal node.**
  The procedure of growing a decision tree follows a greedy top-down approach also known as recursive binary splitting to stratify the features. The recursive binary splitting approach is top-down because it

begins at the top of the tree and then successively splits the features. It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead . So, starting at the top, it tries to analyze all features and all threshold values for each feature to choose the optimal set of features and threshold values that will have the least misclassification **error** for a classifier with a symmetric loss function. This involves determining the features, together with a method for partitioning the values that those features take. The idea of splitting results in partitioning the data into successively purer subsets, although it may not be possible to continue splitting until all leaf nodes are pure. The pure nodes result in low misclassification error for a particular loss function and hence they are preferred (Webb, 2003). The different methods used to select the best features are discussed below.

- **Determining which nodes are terminal nodes.**

  This means that for each node, we must decide whether to continue splitting or to make the node a terminal node and assign to it a class label. If we continue splitting until every terminal node has pure class membership, then we are likely to end up with a large tree that overfits the data and gives a high misclassification error on an unseen test set. On the other hand, if we stop the tree too soon, it may result in a small tree and may underfit the data. There are several stopping criteria that can be considered but the idea of pruning the tree can be really efficient in this regard (Webb, 2003).

- **Assigning class labels to terminal nodes.**

  Labels are usually assigned based on **majority vote** in most of the

cases. This is because majority vote helps in selecting the class that minimizes the empirical risk over the training set. Consider in a dataset D, there are 9 sample points with class 0 and 5 sample points with class 1. Let $n_0 = 9$ and $n_1 = 5$. Also, let $g_1(x) = 1$ and $g_0(x) = 0$
then we can write : $R_n(g_1) = \frac{n_0}{(n_0+n_1)} = \frac{9}{14}$ and $R_n(g_0) = \frac{n_1}{(n_0+n_1)} = \frac{5}{14}$. We get $R_n(g_1) > R_n(g_0)$ for $n_0 > n_1$. Hence we can conclude that taking majority vote help in reducing of the empirical risk over the dataset.

Decision trees are very simple and can be easily interpretable. That is why they are widely used in classification. We will focus on two very commonly used decision tree classifier: C4.5 and CART in this chapter. Both of these classifier belong to CART-procedure.

3.2 C4.5

The C4.5 classifier was proposed by Ross Quinlan in 1993. It was developed to overcome the limitations of ID3, its predecessor. C4.5 uses *Gain Ratio* as a impurity measure to select the best possible feature and also the corresponding split on that feature from the dataset. Before going into the definition of *Gain Ratio*, we try to define a measure commonly used in information theory, called *Entropy*, that characterizes the impurity of an arbitrary collection of data.

The following section is based on Thomas M Cover, 1999. Let X be a discrete random variable in space $\mathcal{X}$ and the probability mass function $p(x) = Pr\{X = x\}, x \in \mathcal{X}$

Definition 3.1. *The Entropy $H(X)$ of a discrete random variable X is de-*

fined by

$$H\left(X\right) = -\sum_{x\in\mathcal{X}} p\left(x\right) log\ p\left(x\right). \tag{7}$$

The log is to the base 2 and *Entropy* is expressed in bits, which is the smallest unit of storage. We use $0 \log\left(0\right) = 0$ as adding terms of zero probability does not change the *Entropy.*

Consider (X, Y) to be a single vector-valued random variable, we define *Joint Entropy* (X, Y) as

Definition 3.2. *The Joint Entropy* $H(X, Y)$ *of a pair of discrete random variables* (X, Y) *with a joint distribution* $p(x, y)$ *is defined as*

$$H(X,Y)\ = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y) log\ p(x,y), \tag{8}$$

which can also be expressed as

$$H\left(X,Y\right)\ = -E \log\ p(X,Y). \tag{9}$$

We also define the *Conditional Entropy* of a random variable given another as the expected value of the entropies of the conditional distributions, averaged over the conditioning random variable.

Definition 3.3. *If* $(X,\ Y\ ) \sim p\left(x,\ y\right)$*, the Conditional Entropy* $H(Y|X)$ *is*

defined as

$$\begin{aligned} H\left(Y|X\right) &= \sum_{x\in\mathcal{X}} p\left(x\right) H\left(Y|X=x\right) \\ &= -\sum_{x\in\mathcal{X}} p\left(x\right) \sum_{y\in\mathcal{Y}} p\left(y|x\right) \ log\ p\left(y|x\right) \\ &= -\sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p\left(x,y\right) \ log\ p\left(y|x\right) \\ &= -E\ log\ p(Y|X). \end{aligned} \tag{10}$$

3.2.1 Relative Entropy

The *RelativeEntropy* is a measure of the distance between two distributions. In statistics, it arises as an expected logarithm of the likelihood ratio. The *Relative Entropy* $D(p||q)$ is a measure of the inefficiency of assuming that the distribution is q when the true distribution is p. *Relatvive Entropy* is also known as Kullback-Leibler divergence.

Definition 3.4. *Let X be a discrete random variable. The Relative Entropy between two probability mass functions p(x) and q(x) is defined as*

$$D(p||q) \ = \ \sum_{x\in\mathcal{X}} p(x)\ log\ \frac{p(x)}{q(x)} = E_p\ log\ \frac{p(X)}{q(X)}. \tag{11}$$

3.2.2 Mutual Information

Mutual Information is a measure of the amount of information that one random variable contains about another random variable. It is the reduction in the uncertainty of one random variable due to the knowledge of the other.

Definition 3.5. *Consider two discrete random variables X and Y with a*

joint probability mass function $p(x, y)$ *and marginal probability mass functions* $p(x)$ *and* $p(y)$*. The Mutual Information I (X; Y) is the Relative Entropy between the joint distribution and the product distribution p(x)p(y):*

$$\begin{aligned} I(X;Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p\,(x,y)\ \ log\ \frac{p\,(x,y)}{p\,(x)\,p\,(y)} \\ &= D(p(x,y)||p(x)p(y)) \\ &= E_{p(x,y)}\ log\ \frac{p(X,Y)}{p(X)p(Y)} \end{aligned} \tag{12}$$

3.2.3 Relationship between Entropy and Mutual Information

We can write from equation(8), $Mutual\ Information$ as

$$\begin{aligned} I\,(X;Y) &= \sum_{x,y} p(x,y)\ log\ \frac{p(x,y)}{p(x)p(y)} \\ &= \sum_{x,y} p(x,y)\ log\ \frac{p(x|y)}{p(x)} \\ &= -\sum_{x,y} p(x,y)\ log\ p(x) + \sum_{x,y} p(x,y)\ log\ p(x|y) \\ &= -\sum_{x,y} p(x,y)\ log\ p(x) - \ (-\sum_{x,y} p(x,y)\ log\ p(x|y)\ ) \\ &= H(X)\ -\ H(X|Y). \end{aligned} \tag{13}$$

3.2.4 Selection of best feature/split in C4.5

C4.5 classifier uses *Gain Ratio* as the measure to select the best feature (Quinlan, 2014). The *Gain Ratio* can be defined in terms of the *Entropy* and *Mutual Information* which we have discussed above.

We have defined *Entropy* for a random variable in the above section. *Entropy* of the set Y can be given by

$$H(Y) = -\sum_{i \in \{0,1\}} p_i \ log\,(p_i),$$

where p_i is the proportion of the data belong to class i.

Mutual Information, that is defined above is also known as *Information Gain* or simply *Gain*. *Mutual information* is defined for random variables with known distribution. It gives the dependence between 2 variables.

$$I(X;Y) \ = \ H(X) - H(X|Y)$$

We have $X = (X_1, \ldots, X_d)$ as the feature vector with different features in it. The *Mutual Information* between Y and any of the features in X can be calculated and out of all these features, the one that maximizes *Mutual Information* is selected. This is because this feature is closely related to Y and putting it in the root node will result in best splitting.

When we try to calculate the same in a decision tree, for a particular feature, say X_1, *Mutual Information* is referred as *Information Gain*.

$$Information\ Gain(Y, X_1) \ = \ H(Y) \ - \ H(Y|X_1)$$

If, for example, $Information\ Gain(Y, X_1) = 0$ (or close to 0), then X_1 and Y are (nearly) independent, meaning that there is no relation between them. That is why the feature with maximum *Mutual Information* or *Information Gain* is used.

Consider the following table :

Table : 1

X_1	X_2	Y
T	T	1
T	F	1
T	T	1
T	F	1
F	T	1
F	F	0

The Entropy, H(Y) can be given by

$$H(Y) \;=\; -\sum_{i=1}^{2} P(Y=y_i)\log_2 P(Y=y_i)$$

We have,

$$P(Y=1)=\frac{5}{6}$$
$$P(Y=0)=\frac{1}{6}$$

$$H(Y) \;=\; -(\frac{5}{6}\log_2\frac{5}{6}+\frac{1}{6}\log_2\frac{1}{6})=0.65$$

From the same table, we can calculate the *conditional Entropy* $H(Y|X_1)$ as

$$H\left(Y|\ X_1\right)=-\sum_{j=1}^{2}P\left(X_1=x_j\right)\sum_{i=1}^{2}P\left(Y=y_i\mid X_1=x_j\right)\ \log_2 P\left(Y=y_i\mid X_1=x_j\right)$$

$$
\begin{aligned}
P(X_1 = T) &= \frac{4}{6} \\
P(X_1 = F) &= \frac{2}{6} \\
P\left(Y = 1 |\ X_1 = \ T\right) &= \frac{4}{4} = 1 \\
P\left(Y = 0 |\ X_1 = \ T\right) &= \frac{0}{4} = 0 \\
P\left(Y = 1 |\ X_1 = \ F\right) &= \frac{1}{2} \\
P\left(Y = 0 |\ X_1 = \ F\right) &= \frac{1}{2}
\end{aligned}
$$

$$
H(Y|X_1) \ = \ -\frac{4}{6}(1\log_2 1 + 0\log_2 0) - \frac{2}{6}(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) = 0.33
$$

Using the above formula, the *Information Gain(Y, X_1)* can be calculated as

$$
Information\ Gain\left(Y, X_1\right) \ = \ H\left(Y\right) \ - \ H\left(Y |\ X_1\right) = \ 0.65 - 0.33 \ = \ 0.32
$$

Although being very useful, the *Information Gain* has an undesired characteristic, which is to favor the feature variables with a large number of values. Those highly branching features are likely to split the data into subsets with low Entropy values. This may result in overfitting and the number of nodes in the tree may be very large.

For example : If we have a feature ID, in our feature vector, the *Information Gain(Y,ID)* will be the highest as splitting according to each ID will result in a pure node since it has only one case per ID. This will result in overfitting in the training data and further lead to bad performance in the test data.

To address this issue, an adjusted version of *Information Gain* was introduced, called *Gain Ratio* which is used by the C4.5 classifier. *Gain Ratio* attempts to lessen the bias of *Information Gain* on highly branched features

by introducing a normalizing term called the *Intrinsic Information*. The *Intrinsic Information* is same as the *Entropy*.

$$IntI(X_1) = H(X_1)$$

Thus, we define the *Gain Ratio* as the ratio of *Information Gain* to the *Intrinsic Information*.

$$Gain\ Ratio(Y, X_1) = \frac{Information\ Gain(Y, X_1)}{IntI(X_1)} \tag{14}$$

In other words, *Gain Ratio* is the ratio of *Information Gain* to the *Entropy* of the that feature. The ratio has a range from 0 to 1. For all the feature variables, the one that gives the highest *Gain Ratio* is chosen for the split.

To recall, the *Information Gain(Y,X)* can be defined as :

$$Information\ Gain(Y, X) = H(Y) - H(Y|X)$$

Case 1 : When X and Y are independent, we have

$$H(Y|X) = H(Y)$$
$$H(X|Y) = H(X)$$

Therefore, *Information Gain(Y,X)* is zero when X and Y are independent and as a result *Gain Ratio(Y,X)* is also zero.

Case 2 : When X and Y are not independent, we can write *Gain Ratio(Y,X)*

can be written as

$$Gain\ Ratio(Y,X) \ = \ \frac{H(Y) - H(Y|X)}{H(X)} = \frac{H(X) - H(X|Y)}{H(X)} \ = 1 - \frac{H(X|Y)}{H(X)}$$

which shows *Gain Ratio(Y,X)* ≤ 1 as *Entropy* is non-negative.

The main advantage of *Gain Ratio* over *Information Gain* is that it biases the decision tree against considering features with a large number of distinct values. In short, *Gain Ratio* tend to favour the features with less categories. At each stage of the classification tree procedure, a decision of which variable to split and how to make that split is made. For a binary variable, there is only one way to split, so no search over splits is required. Nominal or ordinal variables may produce a binary or multi-way split. For example, the variable 'income' with three levels {low, medium, high} could be split in a binary manner as : {low, medium} and {high} or {low} and {medium, high} or a three-way split – {low}, {medium} and {high}.

In case of C4.5, *Gain Ratio* is calculated for each split and after combining them by taking the weighted average of each split, the one with the highest *Gain Ratio* is selected. For continuous features, all the values are sorted and the mid-value is picked for which partition is created and *Gain Ratio* for that split is calculated and recorded. This is done for all the possible splits and the partition that maximizes the *Gain Ratio* is selected.

3.2.5 Pruning in C4.5

One of the important steps to avoid overfitting the data is tree pruning. C4.5 uses a pruning technique called error based pruning which depends on statistical confidence estimates. The heart of this statistical pruning technique is the calculation of a confidence interval for the error (Quinlan, 2014).

We start by measuring the observed error f over the set of N training data points. We define f as the number of samples incorrectly classified divided by the total sample points. It is same as the missclassification error defined earlier. So, if we have E number of points that are incorrectly classified, then we can say the error $f = \frac{E}{N}$. The error based pruning assumes the error to follow the binomial distribution with default confidence level of 25%. The probability of "success" here is f i.e $\frac{E}{N}$.

Assuming that N is large, the normal approximation of the binomial confidence interval for the error f can be estimated by :

$$CI \;=\; f \pm z_{1-\alpha/2}\sqrt{\frac{f(1-f)}{N}} \tag{15}$$

Here, α is the desired confidence and $z_{1-\alpha/2}$ is the z-score for desired level of confidence (obtained from normal table).

The classifier C4.5 uses the upper bound of the confidence interval to estimate the error. C4.5 compares the error confidence intervals for the two trees in order to decide whether to replace a near-leaf node and its child leaves by a single leaf node. The tree with the highest error is pruned. It is important to notice that the probability of pruning increases when the confidence level decreases,and vice-versa. For the unpruned sub-tree, the error is calculated as a weighted average over its child leaves.

Consider the example below :

There is an unpruned tree with three different nodes. We target the health plan node near the bottom of the tree for pruning. We are trying to calculate if it will be a good idea to prune this node. The default confidence interval for C4.5 is 25%, which gives the value of $z = 0.67$ (from the Standard Normal

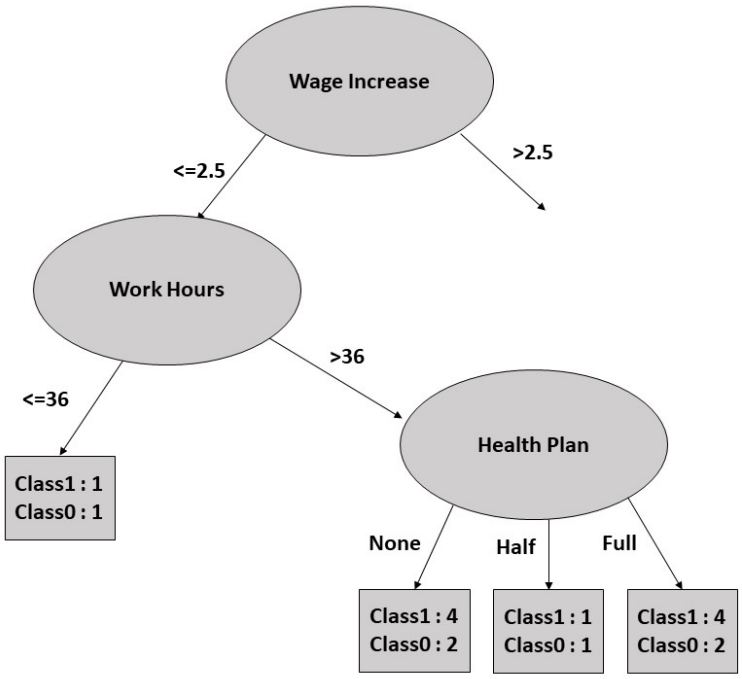

Figure 3: Illustration of nodes in a decision Tree

Distribution Table: Right-Tailed). We can calculate the error estimate by taking the upper bound from the formula in the above equation.

$$e = f + z\sqrt{\frac{f(1-f)}{N}}$$

- **Case 1** : health plan = None

  $f = 2/6$, $N = 6$, $z = 0.67$

  => error estimate = .46

- **Case 2** : health plan = Half

  $f = 1/2$, $N = 2$, $z = 0.67$

  => error estimate = .73

- **Case 3** : health plan = Full

$f = 2/6$, $N = 6$, $z = 0.67$

=> error estimate = .46

Combining these estimates, gives average error estimate = 0.50

That is,

$$\frac{(6 \cdot\ 0.46) + (2 \cdot 0.73) + (6 \cdot\ 0.46)}{6 + 2 + 6} = \frac{7}{14} = 0.50$$

On the other hand, if the tree were pruned by replacing the health plan node by a leaf $(9, 5)$, the error estimate calculation would be as follows: $f = 5/14$, $N = 14$, $z = 0.67$ => error estimate = .44

Since the pruned tree results in a lower estimate for the error, the leaves are indeed pruned and we replace "Health plan" with a leaf. The pruning start from the bottom of the tree and goes up as the pruning process continues. Similarly we can calculate the error estimate for "Work hours" node after the pruning of "health plan". If the sample size is small, an alternate method is used to the confidence interval for the error. This interval is called the Wilson score interval (Wilson, 1927) and the formula can be given by :

$$CI = \left(f + \frac{z^2_{1-\alpha/2}}{2N} \pm z_{1-\alpha/2} \sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2_{1-\alpha/2}}{4N^2}} \right) / \left(1 + \frac{z^2_{1-\alpha/2}}{N} \right) \quad (16)$$

3.2.6 Assignment of class labels in C4.5

The majority class of the instances assigned to the leaf is taken to be the class prediction of that subbranch of the tree.

3.3 CART

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively. It was first introduced by Leo Breiman et al. in 1984.

The classifier works repeatedly in three steps:

Step 1: Find each feature's best split. For each feature with K different values there exist K-1 possible splits. Find the split, which maximizes the splitting criterion. The resulting set of splits contains best splits (one for each feature).

Step 2: Find the node's best split. Among the best splits from Step 1 find the one, which maximizes the splitting criterion.

Step 3: Split the node using best node split from Step 2 and repeat from Step 1 until stopping criterion or pruning criterion is satisfied.

3.3.1 Selection criterion in CART

As splitting criterion, there is a wide variety of impurity function that can be implemented to pick the best feature. In our case, *Gini index* or *Gini Impurity* is used for building up the CART trees. *Gini impurity* calculates the of probability of a specific feature that is classified incorrectly when selected randomly. The *Gini index* varies between values 0 and 0.5, where 0 indicates the classification to be pure, i.e. all the elements belong to a specified class or only one class that exists there and 0.5 indicates the observations is evenly

distributed in all classes.

$$Gini\ Index\ =\ \sum_{i=0}^{k} p_i(1-p_i)\ =\ 1-\ \sum_{i=0}^{k} {p_i}^2. \tag{17}$$

where p_i is the proportion of patterns belonging to class i and k is the total number of available classes.

As we have mentioned above, we have limited the scope of this master thesis to two classes i.e $Y\ =\ \{0,1\}$

Thus, *Gini Index* for two classes can be calculated as

$$Gini\ Index\ =\ p_0\,(1-p_0)\ +\ p_1\,(1-p_1) \tag{18}$$

Now we can write above equation as :

$$\begin{aligned} Gini\ Index\ &=\ (1-p_1)\,(1-1+p_1)\ +\ p_1\,(1-p_1) \\ &=\ p_1\,(1-p_1)+p_1\,(1-p_1)=2p_1\,(1-p_1) \end{aligned}$$

Similarly we can write $Gini\ Index = 2p_0\,(1-p_0)$. Therfore, if there are only two classes, then these formulas can be simplified as

$$Gini\ Index = 2p_*\,(1-p_*)\,, \tag{19}$$

where p_* is the proportion of belonging to one of the classes.

For all the predictors, the one that generates the lowest Gini split is chosen. The combined *Gini index* for a split is the weighted sum of the all Gini Indexes in that split.

Example: Consider the following table:

Here "Return" is the response variable taking two values **Up** and **Down**.

Past Trend	Open Interest	Return
Positive	Low	Up
Negative	High	Down
Positive	Low	Up
Positive	High	Up
Negative	Low	Down
Positive	Low	Down
Negative	High	Down
Negative	Low	Down
Positive	Low	Down
Positive	High	Up

Let's calculate the *Gini Index* for the "Open Interest" feature.

Now we have,

$$P(Open\ Interest = High) = \frac{4}{10}$$
$$P\left(Return = Up \mid Open\ Interest = High\right) = \frac{2}{4}$$
$$P\left(Return = Down \mid Open\ Interest = High\right) = \frac{2}{4}$$

Gini index for Open interest = high can be given by :

$$Gini\ Index = 1 - (\frac{2}{4})^2 - (\frac{2}{4})^2 = 0.5$$

And,

$$P(Open\ Interest = Low) = \frac{6}{10}$$
$$P\left(Return = Up \mid Open\ Interest = Low\right) = \frac{2}{6}$$
$$P\left(Return = Down \mid Open\ Interest = Low\right) = \frac{4}{6}$$

Gini index for Open interest = Low can be given by :

$$Gini\ Index = 1 - \left(\frac{2}{6}\right)^2 - \left(\frac{4}{6}\right)^2 = 0.45$$

Now the weighted sum of Gini indexes for the Open interest feature is given by :

$$Gini\ Index = \frac{4}{10} \times 0.5 + \frac{6}{10} \times 0.45 = 0.47$$

After growing the tree, the next big challenge is termination of the splitting procedure. As mentioned earlier, growing the tree until every node is pure is not feasible as it might lead to overfitting on the other hand stopping early may result in a very small tree, which results in underfitting.

3.3.2 Pruning in CART

Like C4.5, pruning is used in CART to prevent the tree from fully grown. This is done so as to prevent overfitting. The most common pruning method that is used is known as cost-complexity pruning.

Let's associate a real number $R(t)$ with each node t in a given tree T_0. The

quantity R(t) can be written as

$$R(t) = r(t)p(t), \tag{20}$$

where, r(t) is the misclassification error at this node t and p(t) is the proportion of data points that reached node t. $p(t)$ can be calculated as $p(t) = \frac{N(t)}{n}$, $N(t)$ is the number of data points in node t.

Thus we define $R(T_0)$ as

$$R(T_0) \ = \ \sum_{t \in T_0} R(t), \tag{21}$$

where, $R(T_0)$ is the estimated misclassification error for the entire tree T_0

Let $\alpha \geq 0$ be a real number called the complexity parameter. We define the cost-complexity measure $R_\alpha(T_0)$ as

$$R_\alpha(T_0) = R(T_0) + \ \alpha \left|T_0\right| \tag{22}$$

Here, $|T_0|$ is the number of leaves in tree T_0. We want to find, for each α, the subtree T_α that will minimize $R_\alpha(T_0)$. We use the weakest link pruning to find the T_α.

For any node t_0, let T_{t_0} be the branch of tree T_0 with root t_0. We define $g_1(t_0)$ as

$$g_1(t_0) = \frac{R\left(t_0\right) - R\left(T_{t_0}\right)}{\left|T_{t_0}\right| - 1},$$

The weakest link t_0 in T_0 is defined such that

$$t_0 = \arg\min_{(t_0)} g_1(t_0)$$

and then we prune T_{t_0} and define $\alpha_0 = g_1(t_0)$.

Let $T_1 = T_0 - T_{t_0}$ be the pruned true. We recalculate again and find the weakest link t_1 in T_1.

$$g_2(t_1) = \frac{R(t_1) - R(T_{t_1})}{|T_{t_1}| - 1},$$

$$t_1 = \arg\min_{(t_1)} g_2(t_1) \; and \; \alpha_1 = g_2(t_1)$$

We proceed the steps until there is only the root node is left. This will result in nested sequence of subtrees.

$$T_0 \supset T_1 \supset \cdots \supset T_n = \{root\}$$

and

$$\alpha_0 \; < \alpha_1 \; < \ldots \; < \alpha_n$$

There is a small penalty for having a large number of nodes when α is small. As α increases, the subtree has fewer terminal nodes. If $\alpha = 0$ then the biggest tree will be chosen because the complexity penalty term is essentially dropped. As α approaches infinity, the tree of size 1, i.e., a single root node, will be selected.

Thus, we need to find a subtree of T_0, say T_α that minimizes R_α

$$T_\alpha = \arg\min_{(T \in \; subtrees \; of \; T_0)} R_\alpha(T)$$

The following theorem by L. Breiman shows that the sequence above contains T_α for any α. Let $\alpha_i = g_i(t_i)$, where $i = 1, \ldots, n$, $\alpha_0 = 0$ and $\alpha_{n+1} = \infty$

Theorem 2. *It holds* $\alpha_0 < \alpha_1 < \ldots < \alpha_n$. *When* $\alpha \in [\alpha_l, \alpha_{l+1})$, *then* $T_\alpha = T_{\alpha_l} = T_l$, *for all* $l = 1, \ldots, n$.

The above theorem shows that there exist an index l such that $\alpha \in [\alpha_l, \alpha_{l+1})$ and $T_\alpha = T_l$ (Lember, 2012).

3.3.3 Assignment of class labels in CART

The assigning of labels to the leaf node is similar with C4.5. Here, majority vote is used to assign the final class to the node.

4 BAGGING

The name bagging came from the acronym of Bootstrap Aggregating. Leo Breiman (Breiman, 1996), in the year 1996, coined this term and developed the concept of bagging to improve classifications by combining classifications of randomly generated training sets. Bagging is a part of ensemble learning models where the goal is to build a prediction model by combining the strengths of a collection of simple base models. CART trees are used as weak learners for bagging in this master thesis.

Given, for a set of n independent observations $x_1, \ldots, x_n$, each with variance σ^2, the variance of mean $\overline{x}$ of the observations is given by $\frac{\sigma^2}{n}$. In short, averaging a set of observations reduces variance. Hence a natural way to reduce the variance and thus increase the prediction accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions. But in practice, it is not possible to get access to multiple sets of training data. Therefore, we can bootstrap, by taking repeated samples from the training data set. In this approach we generate B different bootstrapped training data sets. We then train our method on these different bootstrapped training sets and combine the results to get the final outcome. The majority vote is used to get the final outcome.

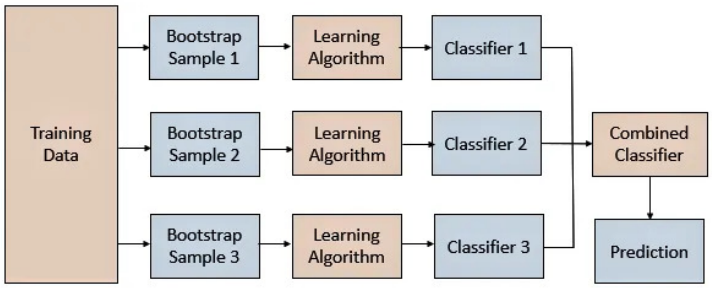

Figure 4: Bagging

There is a description of bagging algorithm below. Here, we have a dataset D and D_{bs} which is the bootstrap sample with replacement. Let g_b be a weak learner. The weak learners are also often refered as base learners or base classifiers. After B rounds, the bagging algorithm combine these base classifiers $g_1, \ldots, g_B$ by taking a majority vote to form the final $G(x)$ classifier (Zhou, 2019).

Algorithm 1: Bagging

Input: Dataset D $= \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$;

Base learning algorithm $\mathcal{L}$;

Number of base learners B

Process:

for *b=1 to B* **do**

$g_b = \mathcal{L}(D_{bs})$, D_{bs} is the bootstrap distribution.

end

Output:

$G(x) = argmax_{y \in Y} \sum_{b=1}^{B} I(g_b(x) = y)$

Some of the important features of bagging are as follows :

- **Out-of-Bag Error Estimation :** The key feature of bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. Breiman(1996d) mentioned in his paper, for a given n observations, the probability that the ith training example is selected 0, 1, 2,...times is approximately Poisson distributed with $\lambda = 1$, and thus, the probability that the ith example will occur at least once is $1 - \left(\frac{1}{e}\right) \approx 0.632$.

  It means there are about 36.8 % original training examples which have not been used in its training process for each base learner in bagging. As a result, there is no need to perform cross-validation or the validation set approach in order to estimate the test error of a bagged model. These remaining one-third of the observations that are not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.

  We can predict the response for the ith observation using each of the trees (base learner) in which that observation was OOB. This will yield around B/3 predictions for the ith observation. In order to obtain a single prediction for the ith observation, we can average these predicted responses (for numeric response variable) or can take a majority vote (if classification is the goal). This leads to a single OOB prediction for the ith observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB MSE (for numeric response variable) or classification error (for a classification problem) can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation (James et al., 2013).

- **Variable Importance :** One of the advantages of decision trees is easy interpretation. Bagging typically results in improved accuracy over prediction using a single tree. However, it can be difficult to interpret the resulting model. When we bag a large number of trees, it is no longer possible to represent the resulting statistical learning procedure using a single tree, and it is no longer clear which variables are most important to the procedure. Thus, bagging improves prediction accuracy at the expense of interpretability. But we can obtain an overall summary of the importance of each variable using the RSS (for numeric response variable) or the Gini index (for nominal response variable). We can add up the total amount that the Gini index is decreased by splits over a given predictor and averaged over all B trees in the bagged classification trees (James et al., 2013).

5 ANALYSIS

In this section, a C4.5 model, a CART model, and a Bagged CART model will be fitted to the real-life Estonian Health Insurance Fund data to classify the invoices. Estonian Health Insurance Fund or Eesti Haigekassa is the state-funded health insurance that is provided for the vast majority of Estonian residents on various grounds (e.g. employee, student, pensioner, monk, or nun, etc.). As mentioned earlier, there are 7903 rows with 46 features and a target variable in the dataset.

All the categorical variables are altered by using one-hot encoding. One-hot encoding creates new variables for all the levels of all categorical variables with values 1 or 0, based on whether the original variable value is equal to that level or not. The response variable, *OutpReclaimStatus* is the reclaim status of the invoice where 0 means it's not reclaimed and 1 means the invoice has been reclaimed. $OutpReclaimStatus = 0$ is taken as the positive class which will be later used in creating the confusion matrix.

The data is almost equally distributed between both genders with 41.52% being male while 58.48% being female. The average age of a patient is 66 years.

We have used R to implement the different classifiers. There are a wide variety of packages available in Rstudio to make the modelling process easier and easily interpretable. We have used the famous "caret" package for fitting the models to the data. The "dpylr" package is used to manipulate the data points while the plots are generated using the "ggplot2" package.

The frequency of each class for the *OutpReclaimStatus* variable is shown in the figure below. It can be observed that the majority of the points, that is

more than 99% are assigned to the class 0.

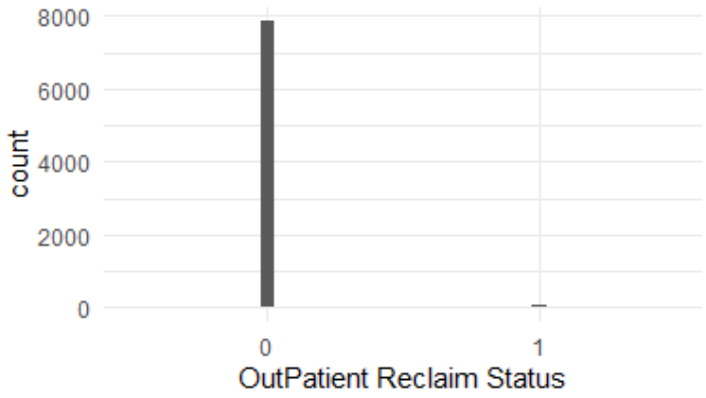

Figure 5: Plot showing the frequency of each class in the response variable

As the figure 5 stats, the variable *OutpReclaimStatus* is not balanced as one class is dominated by the other and as a result, it is not feasible to apply the classifier directly to the dataset. In other words, the class distribution is not equal or close and it is skewed into one particular class (class = 0 in our case). So, the classifier will be accurate for the majority class but if we want to predict the minority class, it will not be that efficient. Therefore, we use an oversampling technique called SMOTE, to generate some synthetic data points. The general idea is of SMOTE oversampling technique is to create synthetic data points in the minority class so that we can reduce the class skewness and have some balance in the dataset.

5.1 SMOTE

SMOTE stands for Synthetic Minority Oversampling Technique. SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem

posed by random oversampling. It focuses on the feature space to generate new data points with the help of interpolation between the ones that lie together.

SMOTE is significantly better than random oversampling as the latter involves randomly duplicating examples from the minority class and adding them to the dataset. As a result, there is no new information gathered by the classifier. On the other hand, SMOTE increases the variety in the dataset, giving new information which would not be possible in duplicating the data.

The SMOTE algorithm has two necessary parameters: k and N, where k represents the number of nearest neighbors it will consider, and N gives the number of times the minority class will be amplified.

The algorithm proceeds as follows :

Step 1: Choose a point say A, randomly from the minority class.

Step 2: Find its k nearest neighbor. The value of $k = 5$ by default.

Step 3: Calculate the difference in distance between the point and its neighbor.

Step 4: This difference is multiplied by any random value in [0,1] and is added to the previous point, A to get the new data point. For k nearest neighbor, k new points will be generated.

Step 5: The process is repeated until the desired number of new points is generated.

Example

Consider a sample point (6,4) and let (4,3) be one of its nearest neighbor. Let $x_1 = 6$, $x_2 = 4$, $x_2 - x_1 = -2$

$y_1 = 4$, $y_2 = 3$, $y_2 - y_1 = -1$

The new sample point will be generated as

$(x^*, y^*) = (6, 4) + rand(0, 1) \times (-2, -1)$, where *rand(0,1)* generates a random number between 0 and 1.

The above SMOTE technique is used when the dataset has continuous features only. The dataset that is being used in this thesis comprises of both continuous and categorical variables. Therefore, an extended approach of SMOTE called SMOTE-NC (Synthetic Minority Over-sampling TEchnique-Nominal Continuous) is used to generate the new data points. Chawla et al. described the SMOTE-NC algorithm as :

Step 1: Median computation: Compute the median of standard deviations of all continuous features for the minority class. If the nominal features differ between a sample point and its potential nearest neighbors, then this median is included in the Euclidean distance computation as a penalty.

Step 2: Nearest neighbor computation: Compute the Euclidean distance between the feature vector for which k-nearest neighbors are being identified (minority class samples) and the other feature vectors (minority class samples) using the continuous feature space. For every differing nominal feature between the considered feature vector and its potential nearest-neighbor, include the median of the standard deviations previously computed, in the Euclidean distance computation.

Step 3: Populate the synthetic sample: The continuous features of the new synthetic minority class sample are created using the same approach of SMOTE as described earlier. The nominal feature is given the value occuring in the majority of the k-nearest neighbors.

Consider the example below :

F1 = 1 2 3 A B C

F2 = 4 6 5 A D E

The Euclidean Distance between F2 and F1 would be :

$Distance = \sqrt{(4-1)^2 + (6-2)^2 + (5-3)^2 + \text{Med}^2 + \text{Med}^2}$.

Here , Med = the median of standard deviations of all continuous features for the minority class.

The median value is included twice in the distance calculation due to the fact that two of the nominal feature are not the same in F1 and F2 and as a result, the distance is penalized by adding the median. This can be interpreted as the distance between two points increases if they have less common nominal values.

This new dataset with additional sample points in the minority class is used to fit the data to the classifier. A detailed description of SMOTE and SMOTE-NC can be found in the original paper published in 2002 by Nitesh V. Chawla et al. The paper concludes the result that shows that the SMOTE approach can improve the accuracy of classifiers for a minority class. Chawla et al. wrote in the conclusion that out of a total of 48 experiments performed, SMOTE-classifier(classifier that used SMOTE) does not perform the best only for 4 experiments (Chawla et al., 2002) .

5.2 Model Building

The dataset is divided into two sets : training set and test set. The training dataset is the sample of data that is used to fit the model. On the other hand, the test set is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. The result based on the training

data can be misleading and hence using an additional dataset can be really effective to understand the performance of a model. The test set is also known as validation set in some cases.

Both the training set and the test set are created by random sampling from the dataset. The "sample" function, that is available in base R is used to generate it. The training set has 70% of the total data points while the remaining is stored as the test set.

The training dataset, **T-ORG** has 5532 (70% of 7903) observations and

class 0	class 1
5489	43

the test set has 2371 (20% of 7903) observations.

class 0	class 1
2353	18

Now the training set is modified and additional synthetic data points are created in the minority class in order to handle class imbalance. This is done with the help of the SMOTE-NC technique. We have used Python software to generate the new data points with SMOTE-NC. The new dataset, **T-SMOTE** has **10978** observations with each class having **5489** observations. As we are using the symmetric loss function, the data points are generated until both classes have the same number of observations.

The classifiers are fitted using *train* function in the *caret* package in R. There were 3 separate 10-fold cross-validations used for resampling. Cross-validation is a resampling method that involves fitting the same classifier

multiple times using different subsets of the data. This is used to make the best out of a limited sample and it generally results in a less biased or less optimistic estimate of the model's prediction than other methods. We have used k-fold cross-validation in our model where the value of k is set to 10. In k-fold cross-validation, the original sample is randomly partitioned into k equal-sized subsamples. Of the k subsamples, a single subsample is retained as the validation dataset for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation dataset. The k results can then be combined (majority vote) to produce a single estimation. As a result, after fitting a classifier to the training data, a model is generated. All the **46** features were used to train the classifiers.

The metric for evaluation is set to be "ROC". A ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate and False Positive Rate. True Positive Rate (TPR) is a synonym for recall/sensitivity while False Positive Rate (FPR) can be defined as 1-specificity. The ROC score that has been measured here is the Area under the ROC curve (AUC).

- **C4.5** : method = "J4.8" where J4.8 is the name of C4.5 classifier in the *caret* package.

  The *train* function returns various models with different combinations of C and m for C4.5. The value of C gives the set confidence threshold for pruning while m denotes the minimum number of data points per leaf. The best combination of C and m results in highest "ROC" score and that model is selected. The parameters of the best tune model for

the **T-ORG** dataset are $C = 0.5$ and $m = 3$ and the results are as follows :

Table : 2

AUC	Sensitivity	Specificity
0.778	0.997	0.143

On the other hand, the parameters of the best tune model for the **T-SMOTE** dataset are $C = 0.0255$ and $m = 3$ and the results are as follows :

Table : 3

AUC	Sensitivity	Specificity
0.993	0.983	0.995

- **CART** : method = "rpart" where rpart is the name of CART classifier in the *caret* package.

  The best model in CART is based on the complexity parameter α. The value of α for which the model gives highest "ROC" score is selected. The value of $\alpha = 0$ yields the best model in the **T-ORG** dataset as the dataset is highly imbalance and the results can be seen below :

Table : 4

AUC	Sensitivity	Specificity
0.678	0.998	0.091

When the CART classifier is applied to the **T-SMOTE** dataset, the α = 0.080 results in highest ROC and yield the following table :

Table : 5

AUC	Sensitivity	Specificity
0.853	0.800	0.834

- **Bagged CART** : method = "treebag" where treebag is the name of Bagging CART classifier in the *caret* package.
  This method allows to use CART trees for bagging. The number of trees is set to 100 for bagging. There are no other specific tuning parameter for CART in *caret* package. The result of the bagged CART model for the **T-ORG** dataset is :

Table : 6

AUC	Sensitivity	Specificity
0.793	0.999	0

and the results can be given by the following table for the **T-SMOTE** dataset.

Table : 7

AUC	Sensitivity	Specificity
0.999	0.991	0.996

The training set is fitted to all three models and based on the ROC score, the bagged CART model outperforms the other two single tree models. But the goodness of fit for the model can be misleading as the assessment is based on the training data. In the next section, the model is assessed on the test set to get an unbiased performance measure.

5.2.1 Model assessment with test set

In order to test the performance of the classifiers, we have three different models which maximize the ROC score based on the training data. Now, the best-tuned model is fitted in the test data to get the performance measure. The performance of the classifiers is measured using a list of six different metrics namely: *Sensitivity, Specificity, Precision, Recall, F1 score, Balanced Accuracy*. The *Balanced Accuracy* is a new measure which is defined as the arithmetic mean of *Sensitivity* and *Specificity*. The following tables show the performance metrics for C4.5, CART, and Bagged CART for both **T-ORG** and **T-SMOTE** dataset.

The table below shows the performance measure of the classifiers on the test set trained on **T-ORG** dataset:

Table : 8

Classifier	Sensitivity	Specificity	Precision	Recall	F1 score
C4.5	0.993	0.111	0.992	0.993	0.993
CART	1	0	0.932	1	0.996
Bagged CART	1	0	0.992	1	0.996

In the **T-ORG** dataset, the classes are highly imbalanced and as a result, when the classifiers trained on that dataset are applied to the test set, most of the classes are assigned to the majority class. The specificity is very low for all of the three models and it can be interpreted as the classifiers having a very high false-positive rate. As mentioned above, to address this issue, SMOTE-NC oversampling technique is used.

The table below shows the performance measure of the classifiers on the test set trained on **T-SMOTE** dataset:

Table : 9

Classifier	Sensitivity	Specificity	Precision	Recall	F1 score
C4.5	0.988	0.277	0.994	0.989	0.992
CART	0.787	0.555	0.995	0.787	0.879
Bagged CART	0.992	0.500	0.996	0.992	0.994

The C4.5 model has shown better performance in *senstivity* as compared to the CART model while the later has higher *specficity* score.

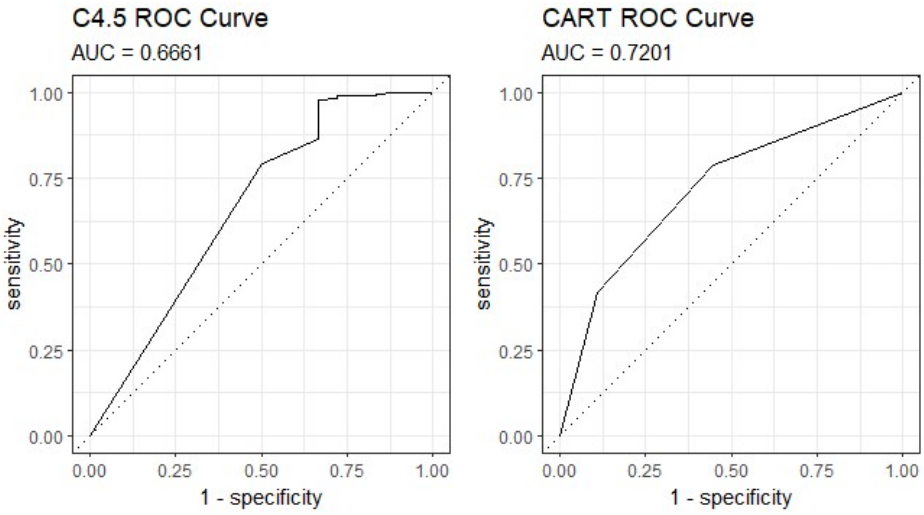

Figure 6: ROC curve of C4.5 and CART

The ROC-AUC curves generated on the test set data show that the CART model yields better results than the C4.5 model. If we look back at Table : 3 and Table : 4 and compare the training AUC score with the test AUC score

for both of these classifiers, it can be seen that the C4.5 model has overfitted the **T-SMOTE** dataset and as a result in spite of good training score, the model failed to perform well in the test set. On the other hand, the CART model has a similar *sensitivity* between the training set, **T-SMOTE** and the test set but failed to capture a good *specificity* score.

When we look at the size of the trees generated by each model, it can be noted that the CART model results in a significantly small tree and as a result, there is a strong chance that the model underfits the training set, while on the other hand C4.5 model results in a very large tree. This difference can be explained by the different pruning techniques used by both of these classifiers.

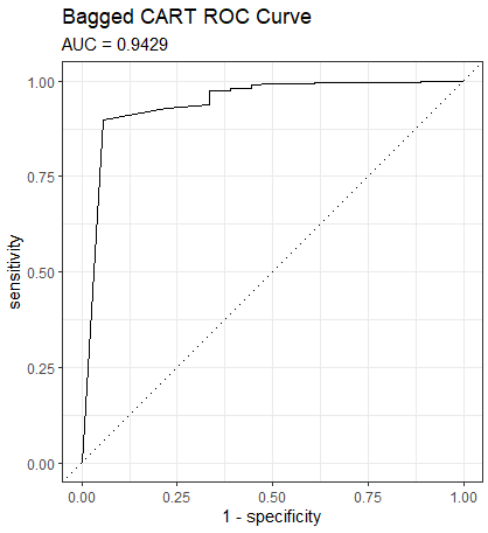

Figure 7: ROC curve of Bagging

The Bagged CART has the best overall performance. This is because bagging is an ensemble learning method and applying the CART classifier to 100 bagged trees results in much better performance.

Finally, if we look at the *Balanced Accuracy* for all of the above models, it can be said that the base classifiers are not very much different from each other based on the score.

Classifier	Balanced Accuracy
C4.5	0.633
CART	0.671
Bagged CART	0.746

6 CONCLUSION

The purpose of the thesis was to give an overview of the different tree-based models and their application in binary classification. There were three different classifiers that were fitted to the data provided by the Estonian Health Insurance Fund to classify incorrect invoices from a large pile of invoices generated every year. This is done to reduce the human effort of manual checking as the number of invoices can go up to millions. The generated models were assessed based on the performance measures that were introduced in the first chapter.

It can be observed from the analysis that the Bagged CART classifier outperforms the other two single tree classifiers. The Bagged CART trained model has an overall accuracy of 98% and can successfully differentiate half of the incorrect invoices in the test set. When we compared the single tree classifiers, CART has much better *specificity* than its counterpart, C4.5. It is also seen that the size of the tree grown by each of these base classifiers largely varies due to the difference in pruning methods. The thesis also considers the class imbalance in the data and tries to solve the issue with an oversampling technique, SMOTE-NC.

The scope of future studies includes analysis and understanding alternate tree-based classifiers and other ensemble learning methods like boosting and random forest for model classification.

REFERENCES

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

Breiman, Leo, Jerome H Friedman, Richard A Olshen, and Charles J Stone (2017). *Classification and regression trees*. Routledge.

Chawla, Nitesh V, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer (2002). "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16, pp. 321–357.

Cover, Thomas M (1999). *Elements of information theory*. John Wiley & Sons.

Dinov, Ivo D (2018). *Data Science and Predictive Analytics*. Springer.

Flach, Peter (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

Friedman, Jerome, Trevor Hastie, Robert Tibshirani, et al. (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

Hssina, Badr, Abdelkarim Merbouha, Hanane Ezzikouri, and Mohammed Erritali (2014). "A comparative study of decision tree ID3 and C4. 5". In: *International Journal of Advanced Computer Science and Applications* 4.2, pp. 13–19.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Lember, Jüri (2012). "Introduction to statistical learning (lecture notes)". In:

Mitchell, Tom M et al. (1997). "Machine learning". In:

Quinlan, J Ross (1996). "Improved use of continuous attributes in C4. 5". In: *Journal of artificial intelligence research* 4, pp. 77–90.

Quinlan, J Ross (2014). *C4. 5: programs for machine learning*. Elsevier.

Webb, Andrew R (2003). *Statistical pattern recognition*. John Wiley & Sons.

Wu, Xindong and Vipin Kumar (2009). *The top ten algorithms in data mining*. CRC press.

Zhou, Zhi-Hua (2019). *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC.

7 APPENDIX 1 : R CODE

```
#######################################################################
library(tidyverse)
library(ROSE)
library(caret)
library(pROC)
library(ROCR)
library(smotefamily)
library(rpart)
library(rpart.plot)
library(PRROC)
library(MLeval)
library(MLmetrics)
library(precrec)
library(yardstick)
library(Metrics)
library(writexl)
#######################################################################
# Getting the data into the system
data <- readRDS("data.rds")
summary(data)
# Descriptive analysis #
prop.table(table(data$GenderN))
# Mean age #
mean(data$AgeFromRegistry)
# Response variable #
```

```
table(data$OutpReclaimStatus)
ggplot(data, aes(x=as.factor(OutpReclaimStatus)))+geom_bar(fill='lightblue',
col='black',width = 0.05)+theme_minimal()+
xlab("OutPatient Reclaim Status")

#####################################################################

#converting features into factors

data$GenderN <- as.factor(data$GenderN)
data$InpEmergencyCareJ <- as.factor(data$InpEmergencyCareJ)
data$OutpEmergencyCareJ <-as.factor(data$OutpEmergencyCareJ)

data$InpSpecialtyReferredDoctor2 <- as.factor(data$InpSpecialtyReferredDoctor2)
data$InpSpecialtyReferredDoctor3 <- as.factor(data$InpSpecialtyReferredDoctor3)
data$InpSpecialtyReferredDoctor9 <- as.factor(data$InpSpecialtyReferredDoctor9)

data$OutpSpecialtyReferredDoctor2 <- as.factor(data$OutpSpecialtyReferredDoctor2
data$OutpSpecialtyReferredDoctor3 <- as.factor(data$OutpSpecialtyReferredDoctor3
data$OutpSpecialtyReferredDoctor9 <- as.factor(data$OutpSpecialtyReferredDoctor9

data$InpDischargeStatus2 <- as.factor(data$InpDischargeStatus2)
data$InpDischargeStatus3 <- as.factor(data$InpDischargeStatus3)

data$OutpDischargeStatus2 <- as.factor(data$OutpDischargeStatus2)
data$OutpDischargeStatus3 <- as.factor(data$OutpDischargeStatus3)
```

```r
data$SameMainDiagnosis <- as.factor(data$SameMainDiagnosis)
data$SameMainDiagnosisCat <-as.factor(data$SameMainDiagnosisCat)
data$SameMainDiagnosisSChp <- as.factor(data$SameMainDiagnosisSChp)

data$SameIssuedDoctor <- as.factor(data$SameIssuedDoctor)
data$SameReferredDoctor <- as.factor(data$SameReferredDoctor)
data$InpIssuedOutpReferred <- as.factor(data$InpIssuedOutpReferred)
data$SameFiscalAccount <- as.factor(data$SameFiscalAccount)

data$OutpReclaimStatus <- as.factor(data$OutpReclaimStatus)
levels(data$OutpReclaimStatus)=c("No","Yes")
table(data$OutpReclaimStatus)
#####################################################################
#spliting the data into training and test set
data$OutpReclaimStatus <- as.factor(data$OutpReclaimStatus)
levels(data$OutpReclaimStatus)=c("No","Yes")
table(data$OutpReclaimStatus)

set.seed(123)
t <-(sample(1:nrow(data),0.7*nrow(data)))
train_ORG <- data[t,]
test  <- data[-t,]

# Getting the SMOTE train dataset
train_SMOTE <- read.csv("train_smote.csv")
```

```
#########################################################################
# FOR THE TRAIN_ORG DATASET
#########################################################################

# Define training control
set.seed(1996)
train.control <- trainControl(method = "repeatedcv", savePredictions = T,
                              number = 10, repeats = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)
# Train the model
C4.5_1<- train(OutpReclaimStatus ~., data = train_ORG, method = "J48",
               trControl = train.control, metric = "ROC")


CART_1 <- train(OutpReclaimStatus ~., data = train_ORG, method = "rpart",
                trControl = train.control, metric = "ROC")

BaggedCART_1 <- train(OutpReclaimStatus ~., data = train_ORG,
                method = "treebag", trControl = train.control,
                nbagg= 100, metric = "ROC")
#################################################################
# RESULTS FOR MODELS TRAINED ON TRAIN_ORG DATASET #
C4.5_1$results
C4.5_1$bestTune

CART_1$results
CART_1$bestTune
```

```
BaggedCART_1$results
#######################################################################
## ASSESSING THE MODELS ON THE TEST DATASET ##
#C4.5
C4.5_values <- bind_cols(
  predict(C4.5_1, newdata = test, type = "prob"),
  Predicted = predict(C4.5_1, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)
C4.5_CM_1 <- confusionMatrix(C4.5_values$Predicted,
                reference = C4.5_values$Actual)
C4.5_CM_1

#CART
CART_values <- bind_cols(
  predict(CART_1, newdata = test, type = "prob"),
  Predicted = predict(CART_1, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)
CART_CM_1 <- confusionMatrix(CART_values$Predicted,
                reference = CART_values$Actual)
CART_CM_1

#Bagged CART
BaggedCART_values <- bind_cols(
  predict(BaggedCART_1, newdata = test, type = "prob"),
```

```
  Predicted = predict(BaggedCART_1, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)
BaggedCART_CM_1 <- confusionMatrix(BaggedCART_values$Predicted,
                    reference = BaggedCART_values$Actual)
BaggedCART_CM_1

## COMBINING THE RESULTS ##

TestCM_1 <- rbind(C4.5_CM_1$byClass,CART_CM_1$byClass,BaggedCART_CM_1$byClass)
TestCM_1

row.names(TestCM_1) <- c("C4.5", "CART", "Bagging(CART)")
TestCM_1
#############################################################################
# FOR THE TRAIN_SMOTE DATASET
#############################################################################

# Define training control
set.seed(1996)
train.control <- trainControl(method = "repeatedcv", savePredictions = T,
                              number = 10, repeats = 3, classProbs = TRUE,
                              summaryFunction = twoClassSummary)
# Train the model
C4.5<- train(OutpReclaimStatus ~., data = train_SMOTE, method = "J48",
               trControl = train.control, metric = "ROC")
```

```
CART <- train(OutpReclaimStatus ~., data = train_SMOTE, method = "rpart",
                trControl = train.control, metric = "ROC")

BaggedCART <- train(OutpReclaimStatus ~., data = train_SMOTE,
                    method = "treebag", trControl = train.control,
                    nbagg= 100, metric = "ROC")
#####################################################################
# RESULTS FOR MODELS TRAINED ON TRAIN_ORG DATASET #

C4.5$results
C4.5$bestTune

CART$results
CART$bestTune

BaggedCART$results
#####################################################################
## ASSESSING THE MODELS ON THE TEST DATASET ##

#C4.5
C4.5_values <- bind_cols(
  predict(C4.5, newdata = test, type = "prob"),
  Predicted = predict(C4.5, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)
```

```
C4.5_CM <- confusionMatrix(C4.5_values$Predicted,
                reference = C4.5_values$Actual)
C4.5_CM

#CART
CART_values <- bind_cols(
  predict(CART, newdata = test, type = "prob"),
  Predicted = predict(CART, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)

CART_CM <- confusionMatrix(CART_values$Predicted,
            reference = CART_values$Actual)
CART_CM

#Bagged CART
BaggedCART_values <- bind_cols(
  predict(BaggedCART, newdata = test, type = "prob"),
  Predicted = predict(BaggedCART, newdata = test, type = "raw"),
  Actual = test$OutpReclaimStatus
)

BaggedCART_CM <- confusionMatrix(BaggedCART_values$Predicted,
                    reference = BaggedCART_values$Actual)
BaggedCART_CM

## COMBINING THE RESULTS ##
```

```r
TestCM <- rbind(C4.5_CM$byClass,CART_CM$byClass,BaggedCART_CM$byClass)
TestCM

row.names(TestCM) <- c("C4.5", "CART", "Bagging(CART)")
TestCM

######################################################################

## VISUALIZATION : ROC_AUC SCORE IN THE TEST SET FOR THE MODEL
## TRAINED IN TRAIN_SMOTE DATASET ##

## ROC_AUC ##

## C4.5 ##

C4.5_auc <- auc(C4.5_values$Actual == "No", C4.5_values$No)
R3<-roc_curve(C4.5_values, Actual, No) %>%
  autoplot() +
  labs(
    title = "C4.5 ROC Curve",
    subtitle = paste0("AUC = ", round(C4.5_auc, 4))
  )
## CART ##

CART_auc <- auc(CART_values$Actual == "No", CART_values$No)
R2 <- roc_curve(CART_values, Actual, No) %>%
  autoplot() +
```

```
  labs(
    title = "CART ROC Curve",
    subtitle = paste0("AUC = ", round(CART_auc, 4))
  )
## BAGGED CART ##

BaggedCART_auc <- auc(BaggedCART_values$Actual == "No", BaggedCART_values$No)
R1 <- roc_curve(BaggedCART_values, Actual, No) %>%
  autoplot() +
  labs(
    title = "Bagged CART ROC Curve",
    subtitle = paste0("AUC = ", round(BaggedCART_auc, 4))
  )
```

8 APPENDIX 2 : FEATURES

Name of the feature	Description
GenderN	Is the patient female? (male is the reference)
InpEmergencyCareJ	Indicator of whether the inpatient invoice is an emergency care invoice
OutpEmergencyCareJ	Indicator of whether the outpatient invoice is an emergency care invoice
InpSpecialtyReferredDoctor2	How the patient arrived to inpatient care? (0 - came by himself without referral ; 1 - by ambulance)
InpSpecialtyReferredDoctor3	How the patient arrived to inpatient care? (0 - came with referral ; 1 - by ambulance)"
InpSpecialtyReferredDoctor9	How the patient arrived to inpatient care? (0 - other ; 1 - by ambulance)"
OutpSpecialtyReferredDoctor2	How the patient arrived to outpatient care? (0 - came by himself without referral ; 1 - by ambulance)
OutpSpecialtyReferredDoctor3	How the patient arrived to outpatient care? (0 - came with referral ; 1 - by ambulance)"

Name of the feature	Description
OutpSpecialtyReferredDoctor9	How the patient arrived to outpatient care? (0 - other ; 1 - by ambulance)"
InpDischargeStatus2	Status of the patient at the end of the inpatient invoice: (0- transferred to another hospital ; 1 - patient left)
InpDischargeStatus3	Status of the patient at the end of the inpatient invoice: (0 - dead ; 1 - patient left)
OutpDischargeStatus2	Status of the patient at the end of the outpatient invoice: (0- transferred to another hospital ; 1 - patient left)
OutpDischargeStatus3	Status of the patient at the end of the outpatient invoice: (0 - dead ; 1 - patient left)
ServiceProviderDistance	Distance (in km) between reported places where the inpatient service was provided and the outpatient service was provided.
SameMainDiagnosis	Is the ICD-10 main diagnosis of inpatient invoice and outpatient invoice exactly the same?

Name of the feature	Description
SameMainDiagnosisCat	Is the ICD-10 category of main diagnosis of inpatient invoice and outpatient invoice the same?
SameMainDiagnosisSChp	Is the ICD-10 sub-chapter of main diagnosis of inpatient invoice and outpatient invoice the same?
SameIssuedDoctor	Has the same doctor issued both the inpatient and outpatient invoice?
SameReferredDoctor	Is the referring doctor of the inpatient invoice and outpatient invoice the same?
SameSpecialtyIssuedDoctor	Is the specialty of the doctors who issued the invoices the same?
InpIssuedOutpReferred	Has the doctor who issued the outpatient invoice also referred the patient on the inpatient invoice, or vice-versa.
SameFiscalAccount	Is the fiscal account of the inpatient invoice and outpatient invoice the same?
SameMainDiagnosisCat	Is the ICD-10 category of main diagnosis of inpatient invoice and outpatient invoice the same?

Name of the feature	Description
OutpMainDiagnosisNum	Main ICD-10 diagnosis of the inpatient bill, transliterated into continuous numeric variable (preserving the ordering of ICD-10)
InpSpecialtyIssuedDoctorNum	Specialty of the doctor/nurse who issued the inpatient invoice, transliterated into continuous numeric variable (preserving the ordering of http://pub.e-tervis.ee/classifications/Erialad)
OutpSpecialtyIssuedDoctorNum	Specialty of the doctor/nurse who issued the inpatient invoice, transliterated into continuous numeric variable (preserving the ordering of http://pub.e-tervis.ee/classifications/Erialad)
AgeFromRegistry	Age of the patient, as derived from personal code
OutpReclaimStatus	Is the outpatient invoice reclaimed? 0 - no, 1 - yes
OutpServDuringInp6000	Amount of lab tests on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp1000	Amount of appointments on outpatient bill which are performed during the inpatient bill.

Name of the feature	Description
OutpServDuringInp1700	Amount of pharmaceutical products on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp7000	Amount of examinations and procedures on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp5000_et_al	Amount of dental care services on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp1400	Amount of transportation services on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp1500	Amount of blood and blood products on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp9000	Amount of general practitioners services, which are covered by capitation, on outpatient bill which are performed during the inpatient bill.
OutpServDuringInp4000_et_al	Amount of surgical procedures on outpatient bill which are performed during the inpatient bill.

Name of the feature	Description
OutpServDuringInpDRG_et_al	Amount of Diagnosis-Related Group codes on outpatient bill which are marked during the inpatient bill.
OutpServDuringInpProp	What proportion of procedures of the outpatient invoice were performed during the inpatient invoice?
OutpServStrictlyDuringInpProp	What proportion of procedures of the outpatient invoice were performed during the inpatient invoice, excluding the first and last day of the inpatient invoice?
PreviousBills	Number of all other bills during one month before the end of the inpatient bill.
PreviousHospitalizations	Number of other inpatient bills during one month before the end of the inpatient bill.
PreviousImmediateHospitalizations	Number of other inpatient bills which ended exactly the day before the start of the inpatient bill.
PreviousBillsSameInpMainDiagnosis	Number of bills with the same main diagnosis as the inpatient bill during one month before the end of the inpatient bill.

Name of the feature	Description
PreviousBillsSameOutpMain Diagnosis	Number of bills with the same main diagnosis as the outpatient bill during one month before the end of the inpatient bill.
PreviousBillsSameInpService Provider	Number of bills from the same service provider as the inpatient bill during one month before the end of the inpatient bill.
PreviousBillsSameOutpService Provider	Number of bills from the same service provider as the outpatient bill during one month before the end of the inpatient bill.
InpMainDiagnosisNum	Main ICD-10 diagnosis of the inpatient bill, transliterated into continuous numeric variable (preserving the ordering of ICD-10)

9 APPENDIX 3 : SMOTE CODE

```
#Getting the data for SMOTE-NC
import pandas as pd
df = pd.read_csv("train.csv")

#Putting all categorical value in one column
cols =['GenderN', 'OutpEmergencyCareJ',
'InpEmergencyCareJ','InpSpecialtyReferredDoctor2'
,'InpSpecialtyReferredDoctor3','InpSpecialtyReferredDoctor9',
'OutpSpecialtyReferredDoctor2','OutpSpecialtyReferredDoctor3',
'OutpSpecialtyReferredDoctor9','InpDischargeStatus2',
'InpDischargeStatus3','OutpDischargeStatus2','OutpDischargeStatus3',
'SameMainDiagnosis','SameMainDiagnosisCat',
'SameMainDiagnosisSChp','SameIssuedDoctor','SameReferredDoctor'
,'SameSpecialtyIssuedDoctor','InpIssuedOutpReferred',
'OutpReclaimStatus']

#converting into categories
df[cols] = df[cols].astype('category')
df.dtypes

#Package for SMOTE NC
from imblearn.over_sampling import SMOTENC

X_train = df.drop(['OutpReclaimStatus'], axis = 1)
y_train = df["OutpReclaimStatus"]
```

```
sm = SMOTENC(categorical_features= [0,1], random_state=0)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

DF = pd.concat([X_train_res.reset_index(drop=True), y_train_res], axis=1)

y_train_res.value_counts()

DF.to_csv("data.csv")
```

Non-exclusive licence to reproduce thesis and make thesis public

I, Priyush Protim Sharma,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, "Tree-based methods in supervised learning with Estonian Health Insurance Fund data", supervised by Prof. Jüri Lember and Mark Gimbutas.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Priyush Protim Sharma
16/08/2021