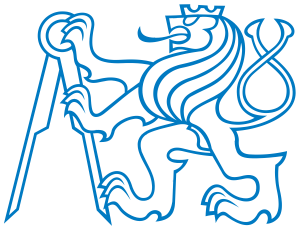
Handling Missing Values in Decision Forests in the Encrypted Network Traffic

Lukáš Sahula

Bachelor's Thesis



Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Supervisor: Ing. Jan Brabec
Program: Software engineering and technologies
May 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sahula** Jméno: **Lukáš** Osobní číslo: **435008**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Neúplná Data a Rozhodovací Lesy v Úloze Klasifikace Šifrovaného síťového provozu

Název bakalářské práce anglicky:

Handling Missing Values in Decision Forests in the Encrypted Network Traffic

Pokyny pro vypracování:

The thesis addresses the problem of training a classifier from data with missing values. More specifically, a Random Forest is to be trained to classify malware from proxy log data. The main focus is on implementing missing-valuehandling method(s) that would lead to improved classification performance, measured by precision and recall.
The concrete goals are:
1) Review prior art in the area of handling missing values in sparse datasets that are applicable to decision forest based classifiers and choose methods suitable for implementation in the thesis´ context.
2) Consider if a modification of existing tools or proposal of entirely new method would be beneficial for the specific case of proxy log classification.
(optional)
3) Implement the chosen methods in Python and connect them to a framework for model training and evaluation (developed prior as part of the software engineering project).
4. Evaluate and compare the classification performance (precision, recall) of implemented methods on a provided test set to the baseline method based on replacing missing values by a constant.

Seznam doporučené literatury:

[1] Criminisi, A., Shotton, J., & Konukoglu, E. (2011). Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114, 5(6), 12.
[2] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning 2nd edition.
[3] Stekhoven, D. J., & Bühlmann, P. (2011). MissForest?non-parametric missing value imputation for mixed-type data. Bioinformatics, 28(1), 112-118.
[4] Feelders, A. (1999). Handling missing data in trees: surrogate splits or statistical imputation?. Principles of Data Mining and Knowledge Discovery, 329-334.
[5] Lin, W. C., Ke, S. W., & Tsai, C. F. (2017). When Should We Ignore Examples with Missing Values?. International Journal of Data Warehousing and Mining (IJDWM), 13(4), 53-63.
[6] Saar-Tsechansky, M., & Provost, F. (2007). Handling missing values when applying classification models. Journal of machine learning research, 8(Jul), 1623-1657.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jan Brabec, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Jan Brabec podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgement

First of all, I want to thank my supervisor Ing. Jan Brabec for his guidance, support, and for offering me such an interesting (and exhausting) topic while back in 2017 I did not know anything about machine learning and the only thing I was really interested in was to find something "cool" to do with the Python programming language.

I also want to give my thanks to my family for their support and the trust they put in me. Especially to my brother Peter for regularly putting things in perspective.

An immeasurable amount of thanks also goes to my friend Tomáš, for he was ready to explain and talk about anything even remotely related to mathematics at all times.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

..... in Prague, on the 25th of May 2018

Abstract

This thesis examines the problem of malware classification using the random forest classifier trained on network traffic dataset. The dataset contains hundreds of millions of labeled objects, some of which are related to malware infection. However, roughly half of the datasets values are missing and these missing values have to be handled before or during the process of classification. The thesis discusses a number of existing approaches to missing data imputation and compares the results of those that are relevant to datasets of this scale. Furthermore, this work provides an analysis of the dataset itself in order to find the correlations between pairs of features and their missingness.

Keywords: malware, classification, random forests, supervised learning, missing values, imputation, feature, correlation

Tato práce zkoumá problém klasifikace malware za použití klasifikátoru náhodných lesů trénovaných na datasetu získaném ze síťového provozu. Tento dataset obsahuje stovky milionů kategorizovaných záznamů, z nichž některé jsou spojené s určitým druhem malwarové infekce. Avšak zhruba polovina hodnot v datasetu chybí a s těmito chybějícími hodnotami je nutno se vypořádat, ať už před procesem klasifikace, nebo během něj. Práce ukáže několik existujících způsobů imputace chybějících dat a porovná výsledky těch, které jsou vhodné k použití s datasetem těchto rozměrů. Dále práce poskytne analýzu použitých dat se záměrem nalezení míry korelace mezi páry jednotlivých atributů a toho, za jakých podmínek jednotlivé atributy chybí.

Klíčová slova: malware, klasifikace, náhodné lesy, učení s učitelem, chybějící hodnoty, imputace, feature, korelace

Table of contents

List of figures

List of tables

Introduction

Machine learning and cybersecurity are two highly significant subjects of today's tech industry. With machine learning being utilized in various fields at an increasing rate, it is no wonder that it has found its uses even in the field of cybersecurity. This thesis touches both of these subjects in the context of automated *malware classification*, with the main focus being dealing with *missing values* using the *random forest classifier*.

In order for a classification algorithm to work, it has to be trained on some set of data. In this case the datasets are encrypted network traffic and as such they contain a moderate amount of *missing data*. Missing data, also referred to as missing values mark the absence of a value somewhere inside the dataset. Most classification algorithms are not designed with an implicit way of dealing with missing values and thus they have to be handled before or during the classification process.

Over the past two decades [1][2][3], a moderate amount of methods for dealing with missing data has been introduced, but none has yet shown superior results that would put it above the others. Plenty of methods are somewhat situational and work only with specific cases of missing data. Thus it is not clear which method to choose at which time. This thesis studies these methods along with analysing the network traffic datasets in order to find out the most efficient one.

The dataset used in this thesis contains real and huge data spanning across more than a hundred of enterprises. As such, one of the concerns in choosing the methods for handling missing data are its computational speed and memory demands.

The thesis consists of several sections. At the beginning, there is a brief introduction to malware classification, followed by an explanation of the classification algorithm often connected to the missing data problem - the *random forest classifier*. The next chapter provides an analysis of the network datasets. The chapter after that concerns itself with missing data mechanisms and gives a summary of known algorithms for missing data *imputation*. The last section is dedicated to the conducted experiments and results of this thesis.

1 Malware and classification

This chapter serves both as a general introduction to the term *malware* and as an introduction to *machine learning*, specifically to one of its subsets, *classification*. After explaining both terms, there is a short section dedicated to connecting the two terms.

1.1 Malware

Malware is an abbreviated form of the term malicious software. It is often used when referring to viruses, spyware, ransomware, and other software designed to cause harm to a computer, server, network or a mobile device. [4]

Viruses are computer programs with the goal of spreading from one file to another across one or multiple devices through a network undetected and without consent of the user. A common misinterpretation of viruses is that they are programs designed to delete or move data, however, this damage is often a side effect. The definition of a virus is that it spreads itself. [4]

Spyware is a form of computer program that runs on the infected computer and tracks its user's habits to form a pattern that can be then used for advertisements or sent to the spyware's creator. [4]

Ransomware is a potentialy very dangerous software that threatens to delete, block or publish the target's data if the program's author's requirements are not met. [5]

When malware spreads through the network, or when it communicates with their command and control servers, it leaves a detectable trace in the network traffic. What more, these traces can often be found much sooner (ranging from weeks to months) than researchers are able to capture a sample of the invading malware. [6] This only emphasizes the importance of utilizing machine learning in the context of malware detection, manual systems and blacklists are not enough today.

According to the five years long study by Georgia Institute of Technology, the endpoints with which specific malware communicates do not change in long time periods (years). This means that once a node in the network suspicious of being infected is found, it is possible to look for traffic going in

and out, which in turn can help with identifying other infected devices. [6]

1.2 Classification

In statistics and machine learning, classification is the process of assigning a specific category, or a *class*, to which a new object or an observation belongs. [7] This assignment, or *prediction*, is done after processing a set of previously categorized data, often called the *training dataset*. [7] This is done by a classification algorithm, known as a *classifier*. [7] This classifier learns from a set of data, for example emails labeled as spam or non-spam, in order to predict the category of new incoming emails. That way an email client can move unwanted spam to the spam folder and keep the relevant mail in the inbox. Another is predicting whether a patient's tumor is benign or malign, based on the hospital records with data from other patients. Or, as will be the case in this thesis, whether an instance of network traffic is the related to (and which) malware, based on data gained from network proxy logs.

Within the terminology of machine learning, classification is in the category of *supervised learning*. [8] Supervised learning means that the classifier learns from the training dataset with labeled data. This dataset is often processed into the form of a matrix $\mathbf{X}$ in order to make the work with it more convenient. The rows in this matrix contain the observed attributes, called *features*, of each individual object. The rows are sometimes called *feature vectors*, labeled $\mathbf{x} = (x_1, x_2, x_3, ..., x_i)$ where i denotes the number of features. Feature vector sometimes refers to the column, containing all values of a single feature across all objects. However, in this thesis, feature vectors refer to the rows. These features are quantifiable and can be categorical (smoker or non-smoker), integer-valued (age of the patient in years), or real-valued (patient's body temperature). Since the learning is supervised, the dataset also includes a column with the class labels of each observed object labeled $\mathbf{Y}$.

When the classifier is trained, it can predict the classes of new objects based on their features. The mathematical function implemented by the classifier maps input feature vectors $\mathbf{x}$ of a matrix $\mathbf{X}$ to a class $\mathbf{c}$ from the set of classes $\mathbf{C}$. The performance of the classifier is usually tested on another set of data called the *testing dataset*. That is done to avoid *overfitting* - "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". [9] According to this definition, an overfitted

classifier could be able to predict the classes of the objects in its training dataset accurately, yet completely miss the classes of new objects. The opposite of overfitting is *underfitting*, which can also be avoided by measuring the classifier's performance on the testing dataset. In machine learning, overfitting and underfitting is sometimes also called "overtraining"and "undertraining"respectively.

Classification does not always have to be binary, meaning the data can be of more than two categories. This is called *multiclass classification*. Returning to the example of e-mail classification, some classifiers could be trained to predict whether an incoming e-mail belongs to *work*, *school*, or *personal* categories.

There are also other forms of classification, like the *multilabel classification*. In multilabel classification, multiple classes can be assigned to an observation. This makes sense in cases where the categories are not mutually exclusive. However, this thesis will be focusing only on the problem of multiclass classification.

1.3 Malware classification

As was mentioned in the malware section of this chapter, malware can often be detected very early by monitoring the incoming and outgoing network traffic of the observed device. However, this is not something that every computer user can do, and even then it can prove to be a relatively tedious task. In this day and age there can be thousands of individual network requests performed by a computer and going through them manually is impossible. As such, the reasonable thing to do would be to take all the data and feed it to a machine learning algorithm to do the work instead. Another point to be made is that current solutions such as static rules defined by domain or server blacklists do not work very well as they do not work dynamically and new entries have to be added manually.

This thesis is based on the work done on labeled network datasets containing observations of more than two hundred different classes of malware. The goal is not only to be able to predict whether a computer is infected with malware, but also to decide which class of malware infection it is. It is, therefore, a case of multiclass classification.

2 Random forest classifier

Random forests or *random forest classifiers* are the central focus of this thesis because they are most suited for the malware classification problem. They can handle *multiclass classification*, evaluate data relatively fast and they are not heavily affected by *imbalanced datasets*. [10] They run efficiently on large amounts of data and give estimates of what feature variables are important for classification. [11]

This chapter explains in detail how random forests work and how they can be implemented. As the name implies, random forest is an ensemble of learning algorithms called *decision trees*. In order to understand random forests, decision trees are the first thing that needs to be explained.

2.1 Decision trees

2.1.1 Overview

Decision tree classifier is a classification algorithm named because of its structure resembling a tree known from graph theory. Another variation of a decision tree is the regression tree, which only differs in that the predicted outcome is not a class but a real value. One of the most significant publications about decision trees is *Classification and Regression Trees* by L. Breiman. It introduces the *CART algorithm* for growing both classification and regression trees. [12] Other algorithms for growing decision trees also exist, for example the *ID3 algorithm* or the *C4.5 algorithm*.

Prediction using a decision tree can be interpreted as a set of questions asking about the attributes of the observed object. For example, predicting whether a picture is taken outdoors or indoors could work as follows:

- Is the top of the picture blue?
  - If yes, the picture was taken outdoors.
  - If no, is the bottom of the picture green?
    - If yes, the picture was taken outdoors.
    - If no, the picture was taken indoors.

As can be seen in the example, a grown tree is built from a root node, decision nodes and terminal nodes, also called *leaves*. All leaves are associated with the resulting prediction class. Each decision node contains information

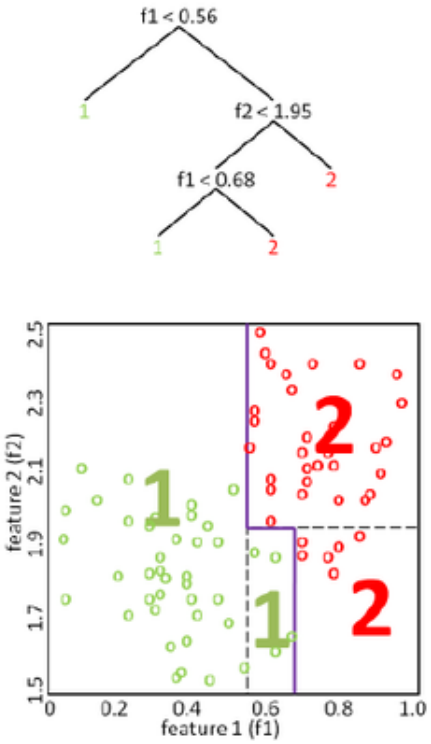


Figure 1: Picture of a decision tree with the training data used for its growth [13]

about its split, which consists of the selected feature and its value. When an observed object is analysed to predict its class, the value of the feature in question is compared with the value stored at the decision node. Depending on the result of this comparison, the object is then sent to the left child or the right child of the node. When the object reaches a leaf node, it is given the class that is associated with it. This explanation is visualised in Figure [1].

2.1.2 Strengths

In comparison with other classification algorithms, decision trees have various strengths and advantages, like:

- They require little data preparation, for example data normalisation is not needed like in other techniques. [14]

- They work with both numerical and categorical data. [14]
- They perform well with large datasets within reasonable time. [11]
- They are easily interpretable by most people after a brief explanation. The graphical representation of the tree is also very easy to follow and the decision making process is similar to how people generally make decisions in real life. [14]

2.1.3 Weaknesses

However, decision trees are not perfect as classifiers go because they also have some weaknesses, for example:

- They are not as accurate as other approaches. [14]
- They are not very robust, a small change in training data can lead to a big change in the decision tree, leading to different results. This is called *overfitting*. [14]

All these disadvantages are going to be addresed again in the random forests part of this chapter.

2.1.4 Growing a decision tree

Training a decision tree classifier is done by recursively partitioning the input data according to the best calculated split. The split specifies the feature and value at which the data is partitioned into two parts. Once the data is partitioned, a new node t_i is created and the process continues at its children, see algorithm [1]. If it is for some reasons impossible or not needed to split the input dataset, a leaf node is created and the recursion does not continue. [10]

To expand on the notation, the nodes are numbered in breadth-first order, starting with the root node t_0. Since all the trees encountered in this thesis are binary, the left child of node t_i can be denoted as t_i^L and the right child can be denoted as t_i^R. Since the dataset is being recursively partitioned, the subset of samples belonging to node t_i can be denoted as S_i. The subset of samples belonging to the left child of node t_i is denoted as S_i^L and the one belonging to the right child as S_i^R. It follows that $S_i = S_i^L \cup S_i^R$ and that $S_i^L \cap S_i^R = \theta$ for every inner node. [10]

Algorithm 1 This algorithm shows how the decision tree classifier is grown recursively. It uses the dataset S_i as the input and returns the root node t_0 of the tree. [10]

```
1: function GrowTree(S_i)
2:     if ShouldCreateLeafNode(S_i) then
3:         return CreateLeafNode(S_i)
4:     end if
5:     θ ← FindBestSplitParameters(S_i)
6:     S_i^L, S_i^R ← SplitDataset(S_i, θ)
7:     t_i^L ← GrowTree(S_i^L)
8:     t_i^R ← GrowTree(S_i^R)
9:     return t_i
10: end function
```

2.1.5 When to create a leaf node

Multiple conditions are checked before the dataset $\mathbf{S_i}$ is split, depending upon which the split may not even take place and a leaf node gets created. The conditions may differ among different implementations of the tree. The ones used for the purposes of this thesis are the following:

- Check the number of unique classes in dataset $\mathbf{S_i}$. If all the samples are of the same class, there is no need to split further and a leaf node can be created. [10]
- Check if it is possible to split the dataset $\mathbf{S_i}$ at least once. It is possible, that all the feature vectors in $\mathbf{S_i}$ are the same, but their classes differ. If that is the case, a leaf node is created. [10]
- One of the hyperparameters of a decision tree is usually the minimum number of samples required to split the dataset ($minSamplesToSplit$). If $|\mathbf{S_i}| < minSamplesToSplit$ then a leaf node is created. This parameter is often set low in random forests because it allows the trees to grow deeper. [10]

Other implementations also include the $maxDepth$ hyperparameter, limiting the maximum depth of a tree, or the change in *impurity measure* (information gain) as the stopping condition during node splitting. [10]

2.1.6 How to create a leaf node

The leaf node contains the subset of sampled associated with it. Upon arriving in a leaf node during prediction, the most common choice is to return the class with maximum frequency in the samples. Different implementations can return the whole normalized histogram of classes in the samples, where each value represents the probability of the object belonging to the given class.

2.1.7 Finding the best split

The best split out of all possible splits of the dataset is defined as the one that provides the largest *information gain*. Information gain corresponds to the decrease in impurity in the node's children. *Impurity measure* of node's subset $\mathbf{S_i}$ is a function $i(\mathbf{S_i}) \in \mathbb{R}$. The purer the data (the lower the impurity measure), the more confidence can be given in the node's prediction. Therefore, the split θ_i that reduces the impurity in the node's children the most, is also the best split. To find it, all possible splits have to evaluated. The decrease in impurity is defined as follows:

$$\Delta i(\mathbf{S_i}, \theta_i) = i(\mathbf{S_i}) - \frac{|\mathbf{S_i^L}| i(\mathbf{S_i^L}) + |\mathbf{S_i^R}| i(\mathbf{S_i^R})}{|\mathbf{S_i}|} \tag{1}$$

There are several impurity measures, most common of which are *Gini impurity* and *entropy*. Studies show that there is not a significant difference between the two and therefore can be used interchangebly. [15] Only entropy is used for the purposes of this thesis. Entropy is defined as:

$$H(\mathbf{X}) = -\sum_{c \in \mathbf{C}}^{n} p(c) \log p(c) \tag{2}$$

Where $\mathbf{C}$ is the set of classes and $p(c)$ is the frequency of class c in the node's subset $\mathbf{S_i}$. [10] If the class is not in the subset at all, it does not add to the sum anything either. The entropy is at its maximum if the classes in the subset $\mathbf{S_i}$ are equally distributed. If all the objects belong to the same class, the entropy equals to zero. As was stated earlier, information gain corresponds to the decrease in impurity $\Delta i(\mathbf{S_i}, \theta_i)$. [10]

To select the best split, all splits along every dimension have to be evaluated. [10]. This can be optimised, as is shown in algorithm [2]. The data points are grouped together to reduce duplicities. This means that out of

50 objects having the same coordinate in the evaluated dimension, we only remember the class counts of those 50 objects and the coordinate itself. Further work in finding the split is done only with the class counts. [10] Since

Algorithm 2 This algorithm shows how to find the split with the highest information gain. [10]

```
1: function FindBestSplitParameters(S_i)
2:     maxGain_Δ ← −∞
3:     for alldimensions do
4:         counts^L ← (0, 0, ..., 0)
5:         counts^R ← GetClassCounts(S_i)
6:         coordinates ← find unique coordinates in dimension
7:         Sort(coordinates)
8:         for point ∈ coordinates do
9:             counts^L ← add class counts present on point
10:            counts^R ← remove class counts present on point
11:            currentGain_Δ = ComputeGain_Δ(counts^L, counts^R)
12:            if currentGain_Δ > maxGain_Δ then
13:                maxGain_Δ ← currentGain_Δ
14:                θ_i ← create split parameters from point and neighbor
15:            end if
16:        end for
17:    end for
18:    return θ_i
19: end function
20: function ComputeGain_Δ(counts^L, counts^R)
21:    size^L ← Sum(counts^L)
22:    size^R ← Sum(counts^R)
23:    return −(size^L × H(counts^L) + size^R × H(counts^R)) / (size^L + size^R)
24: end function
```

many features can have only a very limited set of values, this optimisation can give a significant performance boost to the algorithm. [10] By sorting the coordinates, the class counts can be added to $counts^L$ and removed from $counts^R$ in an organised way. That is done by iterating over all of the sorted coordinates throughout the dimension. This approach also makes the time of each entropy calculation constant because it is dependent only on the number of classes, instead of the number of objects. [10]

After finding the best possible split θ_i, the dataset $\mathbf{S_i}$ is then partitioned

into $\mathbf{S_i^L}$ and $\mathbf{S_i^R}$ by a threshold set by the splits feature and value. Considering the number of dimensions and classes are constant, the worst case time complexity of this algorithm is $\mathcal{O}(|\mathbf{S_i}| \log \mathbf{S_i})$. The worst case is the one where there are no data points with duplicate coordinates and the sorting of the coordinates will equal to sorting the whole dataset. In practice, however, it can be expected that the performance will be better, but still highly dependant on the given data. [10]

2.2 Random forests

Since single decision forests tend to overfit to the training data [14], are not making very accurate predictions, and tend to change significantly with small alterations to the datasets, they do not look very good compared to other approaches. However, an *ensemble method* called random forests deals with these issues. It groups multiple randomized decision trees together and builds a much stronger classifier. There are two categories of ensemble methods, **boosting methods** and **averaging methods**. Boosting methods create the weak classifiers sequentially and every classifier is an improvement of the previous one. [10] Boosting methods are not in the scope of this thesis, so only averaging methods are going to be discussed.

Averaging methods build the classifiers in parallel and average their predictions. This averaging decreases the potentially high variance of the weak classifiers. Random forests belong to this category. [10] Depending on the implementation, the predictions are aggregated either by *majority voting* or by *soft voting*. Majority voting chooses the most frequent class among the predictions of all the trees, while soft voting averages the class probability results from each tree. [16] For random forests to bring an improvement over decision trees, each tree in the forest has to be different. This is where the "random"in the name comes from. Randomness is injected into each tree in different ways, depending on the implementation. It is usually done by introducing randomness to the node splitting process, or by selecting a random subsample from the training dataset. [10] *Breiman forests* [11], by far the most commonly used random forest algorithm, combine *bootstrap aggregating* and *random feature subsets* at each node to inject randomness into the forest. This is done in this thesis as well.

2.2.1 Bootstrap aggregating

Bootstrap aggregating, also called *bagging*, was first introduced in [17]. For a forest of size m and training dataset of size n, bagging assigns a dataset of size n' to each of the m trees. These datasets are created by selecting random samples with replacement from the training dataset. This means that some objects are not present in the sampled dataset and some objects might be repeated. The trees are then trained on their corresponding sample. The size of the bootstrapped datasets is usually parametrized in the forest, but the most common choice is to select $n' = n$.

2.2.2 Random feature subsets

Random feature subsets inject randomness directly into the growing process of the decision tree. When looking for the best possible split, a standard decision tree looks across all dimensions of the training data. [2] However, trees in Breiman forests only use a random subset of the dimensions, sampled for each split node individually. The number of dimensions selected by this sampling is another hyperparameter of the random forest, denoted in this thesis as $maxFeatures$. The value of this parameter affects the correlation between individual trees. [10] The least corellation would be achieved by setting $maxFeatures = 1$. Selecting $maxFeatures = numberOfDimensions$ removes all randomness from the individual trees' splitting process, leaving only the randomness induced by bagging. That, however, is usually not enough. More so when the training dataset contains a big amount of objects. In this particular case, the forest would not bring many advantages over a decision tree. An empirically tested rule of thumb says that $splitFeaturesCount = \sqrt{numberOfDimensions}$ is a reasonable default for classification tasks. [18] Curiously enough, the default for regression classes seems to be equal to the number of dimensions. [18]

3 Network dataset

This chapter centers on the network datasets used for malware classification. The following subsections provide a description of the dataset as well as an analysis done to gather more information on the correlation among pairs of features. The last part of this chapter tries to answer why most of the studied algorithms do not provide satisfying results.

3.1 Dataset description

The data comes from network traffic of more than a hundred large enterprises taken on five days of early 2017. The dataset contains roughly 600 million of individual records. Out of these, only 4 million are labeled as positive, making the dataset heavily imbalanced. Imbalanced dataset means that "at least one of the classes constitutes only a very small minority of the data."[19] In this case, each of the positive classes are a minority while the negative class is an overwhelming (more than 99%) majority. However, a random sampling of the negative objects was done in order to allow faster computation, which also deals with this issue. Out of all the negative objects, only 1.5 million of objects was sampled to make the proportion between classes more balanced.

Each object in the dataset consists of 55 columns. The first column represents the class to which the object belongs. The next 3 columns contain metadata information of the object and are not used for training and prediction. Those are the timestamp, user and host and they are used in aggregated evaluation of the classifier, described later in Chapter [5]. The remaining 51 columns are the features. Because of the sensitivity of the data, we are unable to publish their exact meaning. However, some examples of the features could be the number of bytes transmitted or received, duration of the network event, direction of the packet, the amount of incoming and outgoing packets, the IP address of the destination, etc. The data comes from three different sources. One of them is the network flow, one is the output of *Cisco CTA* engine, which detects the periodicity of connections as an example. The third source is computed globally, for instance the number of users connecting to the server or the information from DNS. The individual values of these features are represented by integers and real numbers.

The objects in the dataset all come from a set of 115 classes in total.

An important thing that needs to be taken into consideration when working

with a dataset like this is the fact that the samples are taken from various days. This could potentially mean that a classifier can perform better on data taken from the same day or from the day after, but worse with data from the next month, or even a year. The samples analysed in this thesis are from days week apart in January, with one day from the beginning of March. In this small scale, however, no significant differences in performance that could be attributed to this were found.

3.2 Analysis

This subchapter examines the dataset and attempts to analyse the nature of missingness of its missing data. The first step of the analysis was to look at the features and see how often and when are they missing. After that, a *correlation matrix* was constructed to see the correlation among pairs of the features. Another correlation matrix was constructed from an altered dataset where each value was replaced by 1 and each missing value was replaced by 0 to see the how correlated the features' missingness is. A matrix of conditional probabilities was also created to provide information on how often is one feature not missing when the other one is missing.

3.2.1 Missingness

In the whole dataset, there are only 4 features that are not missing from any of the objects. 12 features, on the other hand, are missing in more than 90% of the time. The average missingness throughout the dataset is roughly 51%. Some features are closely tied together and when one of them is missing, the others are missing as well. Therefore the assumption is that some of the features are not missing at random. [4.2] Other features, however, do not fulfill this assumption and could possibly be used when others are missing if they are somehow correlated.

3.2.2 Correlation matrix

The second step in this analysis was to figure out how are the features correlated to each other. For that, the *Pearson correlation coefficient* was used, defined as: [20]

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3}$$

In this equation X and Y are features and the result is the amount of correlation between them. Correlation is bound between -1 and 1: [20]

$$-1 \leq \rho_{X,Y} \leq 1 \tag{4}$$

The closer the correlation is to 1, the stronger the positive linear dependence between the features is and vice versa, the closer the correlation is to -1, the stronger the negative linear dependence between the features is. For the purposes of this thesis it is not required to differentiate between positive and negative correlation because both are equally important. Therefore an absolute value of correlation is used in the following analysis.

Computing this correlation for each existing pair of all the features in the dataset we get a 51 x 51 symmetric matrix with the number 1 on the main diagonal. Every element in this matrix symbolizes the correlation between the feature represented by the element's row, and the feature represented by the element's column. The matrix is symmetric because the pair of features (X, Y) has the same correlation as the pair (Y, X). Furthermore, the main diagonal is full of 1s because a feature is completely correlated with itself. However, in some cases it is impossible to compute the correlation, and that is when the feature is always missing. In this thesis, its correlation value is replaced by 0. With this matrix, it is possible to create a heatmap in order to provide a more visual representation, seen in Figure [2]. It is possible to see that some pairs of features in fact are highly correlated.

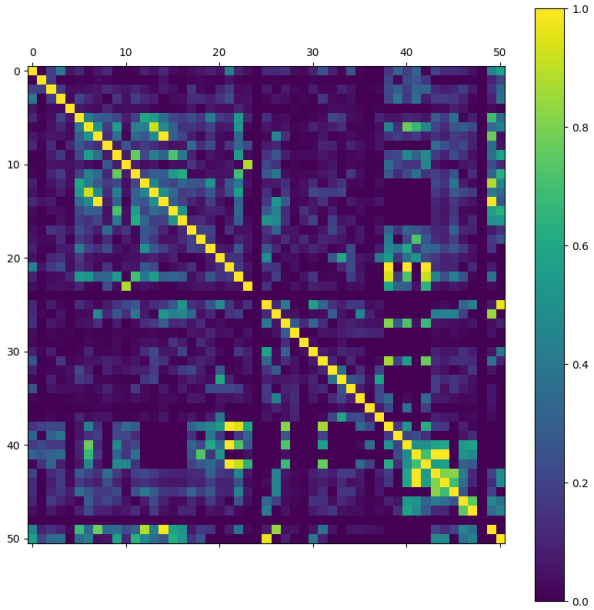


Figure 2: Correlation matrix showing the amount of correlation among pairs of features

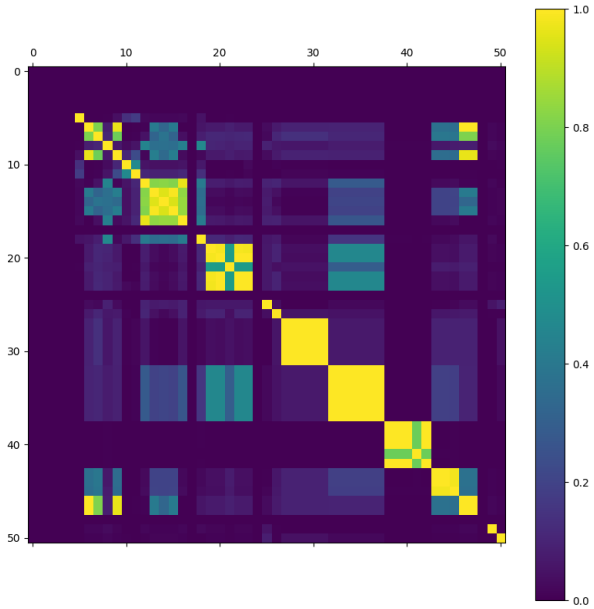


Figure 3: Correlation matrix showing the amount of correlation among pairs of features based on them being missing or not

3.2.3 Missingness correlation matrix

The same way as before, it is possible to create a correlation matrix from the dataset where every not missing feature is replaced by number 1 and every missing feature is replaced by number 0. This correlation matrix tells us how correlated the missingness of pairs of features is. To put it in other words, it shows how dependent the fact that a feature is missing on another feature being missing or not missing. A heatmap of this matrix is in Figure [3]. However, the features that are never missing, or the ones that are always missing, have no variance because their value is either always 1 or always 0. Their correlation therefore could not be computed and set to 0. It is possible to see that the most correlated are the features that are next to each other, meaning that they come from the same source or that they are connected to each other. This ties together with the assumption that those features are not missing at random [4.2].

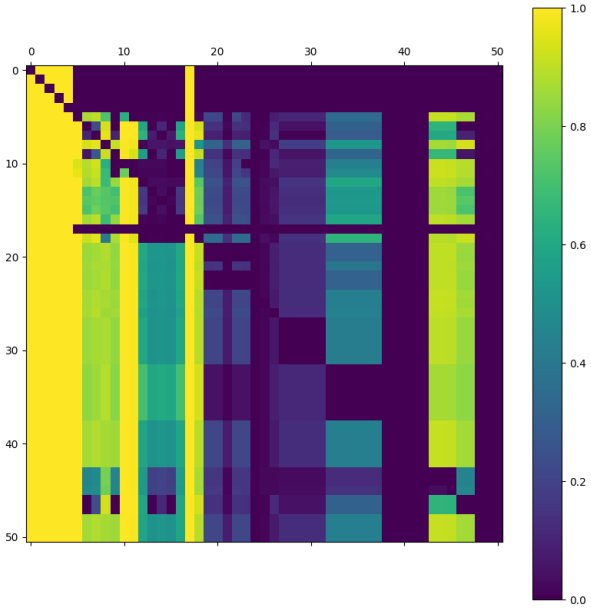


Figure 4: Conditional probabilities matrix showing each element's (i,j) probability that feature j is not missing when feature i is missing

3.2.4 Conditional probabilities matrix

The conditional probabilities matrix represents the probability that feature X is not missing when feature Y is missing [5]. This matrix is not symmetric because the probabilities differ for pairs (X, Y) and (Y, X). [4] The yellow belt going from the top to bottom symbolizes the features there are never missing. The probability that the first four features are missing is zero, therefore the conditional probability that other features are not missing when the first four are is also zero. The value on the i-th row in the j-th column gives the probability that feature j is not missing when feature i is missing.

$$P_{notmissing|missing}(X, Y) \tag{5}$$

3.2.5 Feature substitution

Combining the conditional probabilities matrix with the correlation matrix, it is possible to set a correlation threshold and pick only those pairs of features that are correlated above that threshold. With this list of pairs it is then possible to look at the conditional probabilities matrix and see how often is one of the features not missing when the other one is. That way, an algorithm like the surrogate splits [22] could take these highly correlated pairs into account. If there were a feature that is missing everytime another feature is not missing, the algorithm could base the decision on the value of the other feature.

In other words, let us for example say that feature A is highly correlated with feature B. Feature A is missing 70% of times while feature B is missing only 20% of the time. Since they are highly correlated, the classifier can base its decision on feature B instead of on feature A when feature A is missing. The next part of this analysis looks at each feature and studies how many different features can be used in the classification instead of them when they are missing.

Setting the correlation thresholds to 0.3, 0.5 and 0.8 showed that a small amount of features have some correlated features that could substitute for them. With the threshold set to 0.3, there are 20 features that have at least one candidate. [5][6][7] The figures show each feature and the number of their correlated counterparts. This count of correlated features depends on the conditional probability of them being not missing when the feature they are correlated to is missing. The numbers on the y axis are the conditional probability thresholds. As expected, when this threshold increases, the amount of correlated features decreases.

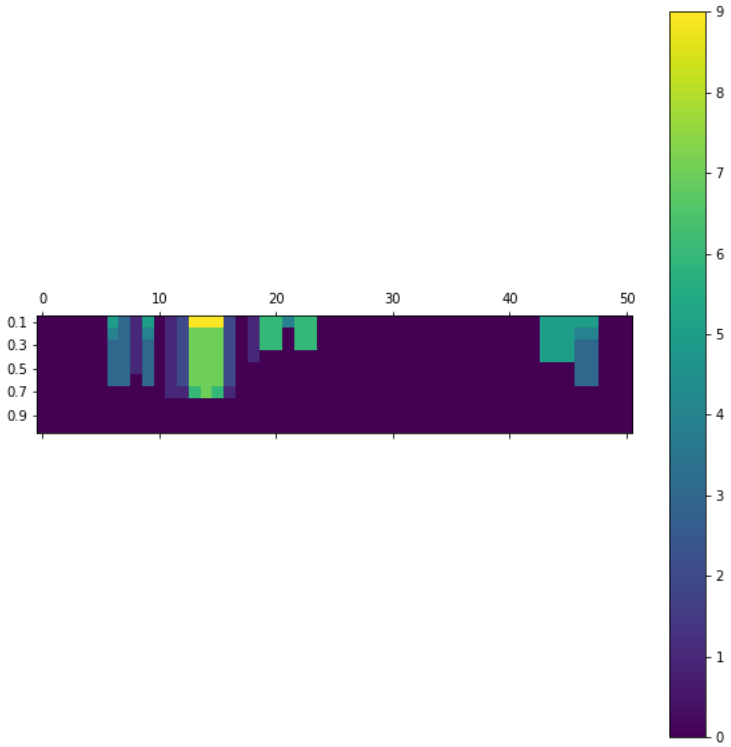


Figure 5: Counts of potential substitions based on their non-missingness probability for each feature with correlation threshold 0.3

In Figure [5] it is possible to see that the most prominent features are features 13, 14 and 15. When they are missing, there is at least a 10% probability that 9 other features are present that could take their place. As the required probability increases, up until 60% there are 6 of those features.

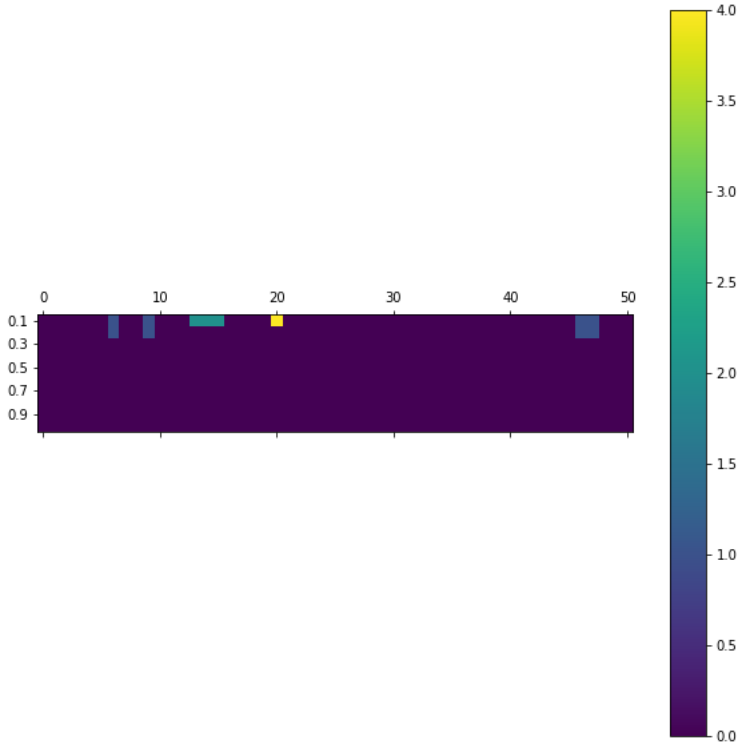


Figure 6: Counts of potential substitions based on their non-missingness probability for each feature with correlation threshold 0.5

Figure [6] shows that with correlation threshold set to 0.5, only 8 relevant features remain, with the maximum of 4 correlated features for feature 20. The probability threshold is also much lower than in the previous case.

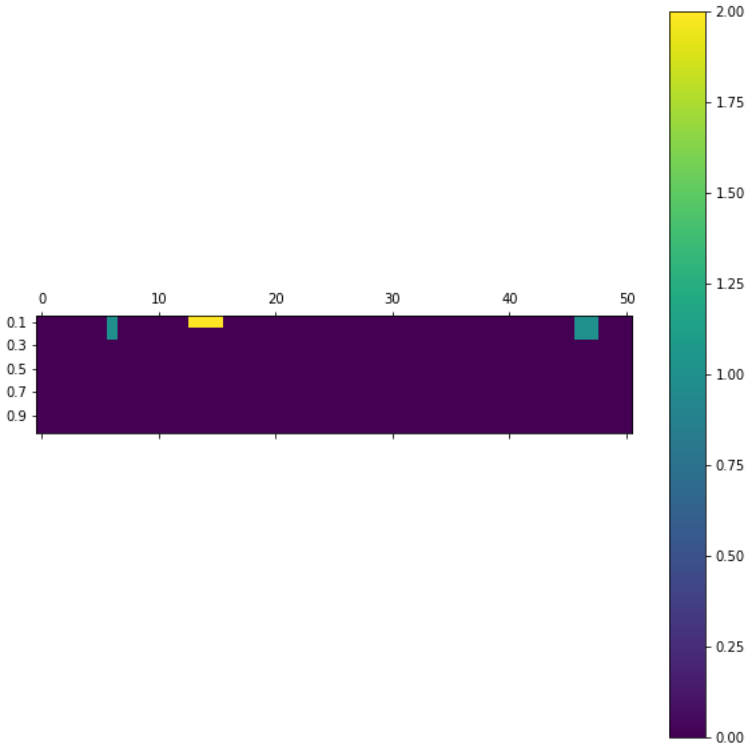


Figure 7: Counts of potential substitions based on their non-missingness probability for each feature with correlation threshold 0.8

Correlation threshold of 0.8 further reduces the number of relevant features to 6 [7]. The maximum of correlated features in this case is only 2.

A conclusion can be made, that although there are some features that can be substituted by other features when they are missing, the probability of the substitution features not being missing is not very high. Also, as the demand for correlation between these features increases, the amount of potential candidates decreases rapidly.

4 Handling missing values

In statistics, *missing values* or *missing data* mark the absence of value in a feature variable of an observed sample. Missing data within a dataset make it impossible for conventional machine learning algorithms to properly learn from it. Many of these algorithms require complete data and do not have an implicit way of handling missing values. In order for the statistical analysis to work, the missing data have to be somehow dealt with beforehand. [1]

There are various ways of handling missing data, some as simple as dropping the samples containing any missing values altogether, leaving only those samples with all data present. [2] This however, cannot be done when there are missing data somewhere in most of the observations. Another simple method, the strawman imputation [1] would be replacing missing data with the mean or median of all the non-missing values of the feature in question. The process of replacing the missing data is called *imputation*. Only the imputation methods related to random forests are examined in this thesis because of the nature of the analysed data. Also, most of the interesting imputation methods already are in this subset.

More sophisticated methods of imputation also exist and they are discussed more thoroughly in this thesis. Some methods work better with smaller datasets, or with datasets with low missingness ratio. The amount of correlation between feature variables can also be important to some methods and not that important to others. [1] Some imputation algorithms are very slow and using them in experiments with big datasets can get seriously time-consuming. In other words, choosing a relevant method of imputation for a given dataset is not a simple task and it can prove useful to do an analysis of the data as well as an analysis of the imputation algorithm itself beforehand.

4.1 Related work

Various works comparing different imputation methods were published in the last years. Multiple imputation methods meant for decision trees are well explained in [3].

Another work worth mentioning that focuses on random forests and missing data imputation is [21]. It introduces a new type of random forests along with the *adaptive tree imputation* method. However, this method can be used with regular random forests as well.

Perhaps the most recent work on this topic is [1] which is a continuation of the previous work by one of the same authors. In this paper, multiple approaches to imputation using random forests are discussed in detail and tested on multiple datasets with different attributes. A comparison of their imputation accuracy is made as well. The datasets used in this work are both real and synthetic. However, the missingness is induced manually [1] and thus the results of the experiments can be biased.

4.2 Missing data mechanisms

Missing data are usually divided into three categories, depending on the characteristics of their missingness. These categories are also called missing data mechanisms. [2] It is important to understand these mechanisms in order to make an educated assumption about the dataset in question. If an assumption is made that the values are missing completely at random while they are missing systematically, an analysis based on this assumption may be biased.

- If the samples with missing data are a random subset of all observed samples, they have a similar distribution. There is no relationship between whether a value is missing and any other value in the data set, be it missing or observed. These values are *missing completely at random* (MCAR). [2] When a dataset has missing values that are MCAR, it can still be analysed with unbiased results, provided that there are enough observed samples. However, data that are truly MCAR are not encountered often.

- When the missing data is somehow dependent or related to other non-missing values in the observation, the data is *missing at random* (MAR). [2] There is no definitive way of distinguishing between MCAR and MAR data. The assumption of it being one or the other is only as good as the knowledge of the data and the field of the one proposing the assumption. A good example of MAR data would be that males are less likely to fill a depression survey, although it does not have any connection to their level of depression.

- The last missing data mechanism is *missing not at random* (MNAR). It means that there is a relationship between value of the variable

that is missing and the reason why it is missing in the first place. For example a male suffering from a strong depression can decide not to fill a depression survey because of said depression.

4.3 Selected imputation algorithms

This section contains a list of imputation algorithms that were taken in consideration when deciding which ones to implement.

4.3.1 Baseline imputation

This method will be used throughout the thesis as the baseline reference value. It replaces the missing data with a constant value outside of the features' interval before growing the tree. Thus, when an object is evaluated and the relevant feature is missing, the object is always sent either to the left or to the right, based on the value that was chosen at the start.

4.3.2 Strawman imputation

Strawman imputation [1] is a simple method for handling missing values. It works very fast compared to other methods and its implementation is very simple. The missing values are imputed before the forest is grown by calculating the median of non-missing values in the feature column. An altered variation of strawman imputation uses the mean instead of the median.

4.3.3 On-the-fly-imputation

Adaptive tree imputation [21] later named as *On-the-fly-imputation* [1] (OTFI) is a method of imputing missing data at the time of growing the tree. That is also where its name comes from. It draws a random value from the non-missing in-bag dataset within the current node to impute the missing values. The algorithm works as follows:

1. First, the best split is calculated as usual, using only non-missing data.
2. After finding the best split, random values from the non-missing in-bag data within the current node are used to impute the missing values.
3. Once the data are imputed, the node is split into the left and right children nodes and the imputed data are reset back to missing.

The values the algorithm uses to impute the missing data are stored within the decision node while growing the tree along with the respective frequencies in which they occur within the node's dataset. These are then used again to impute the missing values in the testing dataset at the time of prediction to decide whether the observation belongs to the left or to the right child. After being sent to the child node, the imputed data is reset to missing again and the process repeats.

The implementation of OTFI algorithm used for experiments in this thesis had to be improved in order to increase imputation speed and reduce the memory used. Instead of storing the non-missing values and their frequencies, only the probability of the node going left or right is saved at the node. That way, when a new object with a missing value at the relevant feature is examined, the missing value is not imputed at all.

4.3.4 Missingness incorporated in attributes

Missingness incorporated in attributes [3] (MIA) works similarly to OTFI in that it also imputes missing data during the forest growing process. It searches for the best split in three different approaches to the missing data. Let X be a numeric feature used to split a node and s a possible split value of X. Over all split values, the method looks at the following:

- Split A: $\{ X \leq s$ or $X =$ missing $\}$ versus $\{ X > s \}$.
- Split B: $\{ X \leq s \}$ versus $\{ X > s$ or $X =$ missing $\}$.
- Split C: $\{ X =$ missing $\}$ versus $\{ X =$ not missing $\}$.

After finding the best split in each approach separately, the algorithm then chooses which one of them provides a better information gain and then it splits the dataset accordingly. As in the OTFI method, the MIA implementation used in this thesis makes the decision node remember which split type it had used in order to decide how the testing data should be propagated during prediction.

This algorithm is in practice an extension of the method that was used as a baseline. The baseline in fact does the same thing as split A in here. It follows that this method takes approximately three times more time than the baseline.

4.3.5 Other algorithms

Other notable imputation algorithms were also considered, such as *surrogate splits* [22] or *missForest* [1]. However, surrogate splits perform less reliably as the amount of missing data increases [22] and are slow when used with larger datasets. [21] The low computational speed is a problem for missForest as well, making it "100's of times"slower than methods like OTFI or MIA. [1] Since the network traffic dataset is very big and a big portion of its data is missing [3], these methods are not very relevant and they are not in the scope of this thesis.

5 Experiments

This chapter focuses on the experiments done with network data regarding the performance of several imputation methods. The first part describes the measures used for evaluation and why they were used. The second part looks at the results of the experiments and their comparison. All of the experiments were done on the same data, that is the network traffic logged on three days of January 2017, each of them a week after the previous one, was used as the training dataset, whereas the testing dataset was a single day of March of the same year. Each experiment was evaluated in three different ways - *unaggregated*, *aggregated*, and *aggregated relaxed*. Unaggregated means that the performance is evaluated in a regular per-row manner. Aggregated (by one of the metadata column, for example a user) evaluation is done in order to remove duplicities (multiple records marking the same user as a suspect of malware infection) and removes negative records if there is at least one positive one. This makes sense because if a user is infected, the network traffic he produces can be either regular or point to some malware infection. This aggregation removes the arguably unimportant, clean data. It is done to reflect the seen performance by the customers who are not interested in flows, but in users. Going a level further, the aggregated relaxed evaluation clears the differences between different classes of malware and reduces the classification problem into the binary case - infected, or not infected. Yet the classifier itself still remains multiclass.

5.1 Evaluation metrics

There is a plenty of measures that can be used to determine the classifier's performance. The most commonly observed is perhaps the *classification error* [10]. The problem with classification error is that it does not say much when the dataset is imbalanced, as is the situation with the network data. For example if we had a dataset where there are 10,000 samples and only 1 of them belonged to a positive class, the classifier could classify all samples as negative and it would achieve a classification error of 0.01%. In this project we are interested in two different measures, *precision* and *recall*. In order to compute them, we also need the *confusion matrix*.

5.1.1 Confusion matrix

In binary classification, we can label the two classes as positive and negative. *Confusion matrix* [23] divides the classification results into the following categories:

TP: True positives. The number of positive objects the classifier labeled as positive.

FP: False positives. The number of negative objects the classifier labeled as positive.

TN: True negatives. The number of negative objects the classifier labeled as negative.

FN: False negatives. The number of positive objects the classifier labeled as negative.

These categories are then used to compute other, more interesting measures.

In multiclass classification, the *confusion matrix* gets more complicated. Consider a dataset with 3 different classes A, B and C. If the classifier takes a sample that belongs to class A and labels it as B, then it counts as FN for class A, but as FP for class B.

This gets even more confusing with the network datasets. The samples are either negative and thus belonging to one specific class, or positive, which means one of multiple positive classes. For that reason, samples belonging to the negative classes that are classified as negative do not count as TP. Similarly, when a positive sample is labeled as positive but a different class, it should not be counted as FN, rather as TPish. As for TNs, it does not make much sense to distinguish them from TPs, since they basically are TPs for the negative class. This is a way of reducing the multiclass problem to a binary problem. In the context of this project, a *one-vs-all* confusion matrix, which evaluates one class at a time, is computed. Furthermore, TNs do not get counted and the TPs of the negative class are optionally filtered out in the process.

5.1.2 Precision

Precision, or *positive predictive value* [23] is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Precision is the fraction of positive objects among all objects that the classifier labeled as positive. It can also be interpreted as the probability that a positively labeled object is truly positive. A precision score of 1.0 means

that every object labeled as positive by the classifier is truly positive. It does not say, however, anything about the classifier's ability to recognize all truly positive instances. In multiclass environment, precision of every class varies because it is computed separately.

5.1.3 Recall

Recall, or *sensitivity* [23] is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Recall is the fraction of positive objects that the classifier labeled as positive among all truly positive objects. It can be interpreted as the probability that a truly positive object is labeled as positive. A recall score of 1.0 means that every truly positive object is labeled as positive by the classifier. This could be achieved simply by labeling all objects as positive, recall does not say anything about the number of false positives. Recall of every class varies among all classes in multiclass environment.

For a better comprehension of precision and recall, Figure [8] shows what they represent.

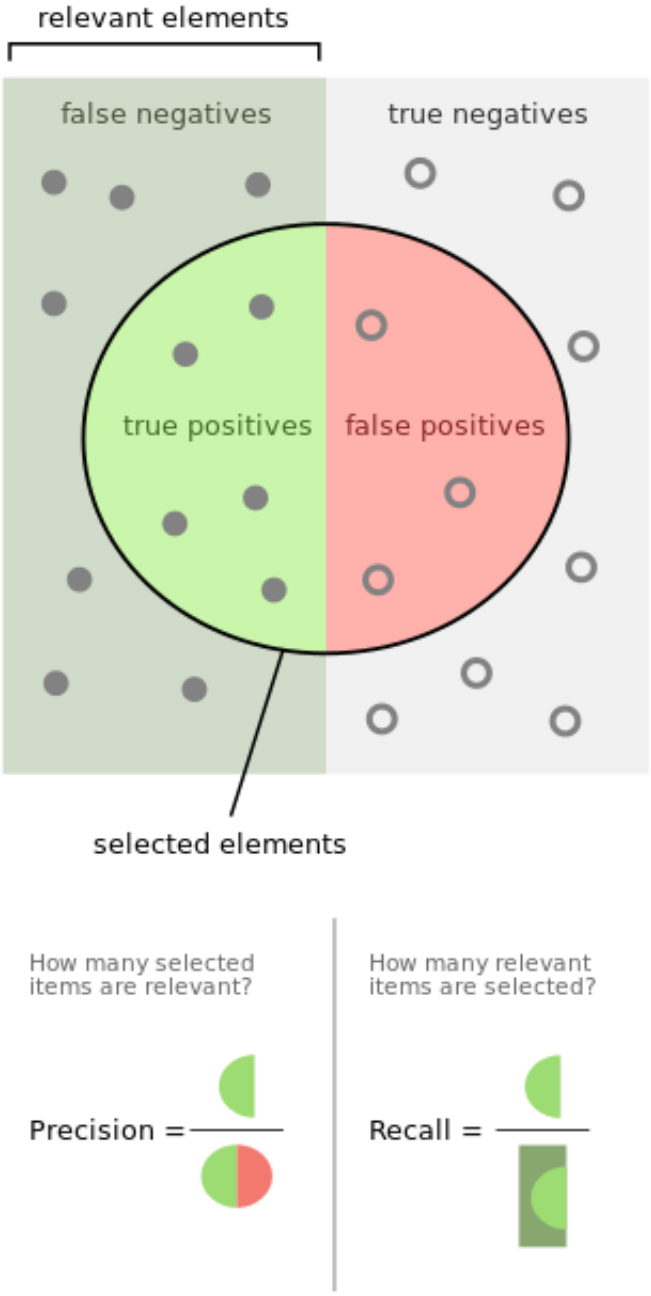


Figure 8: Visual representation of precision and recall [24]

5.2 Results

This subsection presents the results of the individual experiments. The main focus is the average prediction precision, recall and also the number of classes that were predicted with precision above specific thresholds. The following tables will always show all three evaluation approaches (unaggregated, aggregated, relaxed) for every tested method. The tested methods are the baseline [4.3.1] method, the Strawman [4.3.2] imputation method both with mean and median values, the OTFI [4.3.3] method and the MIA [4.3.4] method. The two variations of the Strawman method are further referred to simply as mean and median.

The randomness factor of random forests gives the resulting precision and recall a variance of about 1%. So if one method has a precision of 10% and another one only 8%, we can assume that their performance could be the same.

5.2.1 Average overall precision

The first metric to look at is the overall precision averaged across all classes. However, in this case only the positive classes are relevant, so the negative class is excluded. Looking at Figure [9], it can be seen that the OTFI method does not bring any improvement compared to the baseline and its results are actually even weaker. This could be happening because the OTFI method computes the split statistic from the not missing data and therefore does not have enough data to grow the trees sufficiently. The MIA method performs slightly better than the baseline while the Strawman imputation using mean value performs equally. The Strawman using median proved to be worse.

Figure 9: Precision of the tested methods averaged across all positive classes

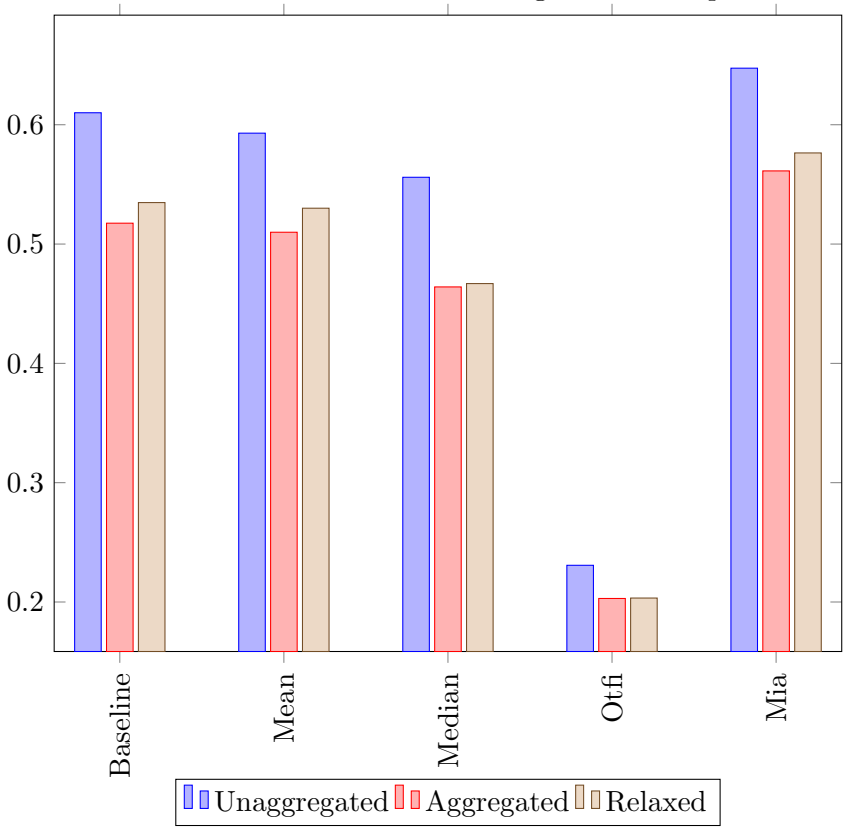


Table 1: The number of classes predicted with precision above a specified threshold

Method U / A / R	Baseline			Mean			Median			OTFI			MIA		
# of classes prec > 0.5	70	61	63	70	59	63	65	52	52	25	22	22	74	67	68
avg recall %	74	83	84	76	82	82	70	80	80	27	25	25	74	85	85
# of classes prec > 0.8	54	37	39	54	38	41	45	32	32	25	21	21	60	40	41
avg recall %	74	87	88	77	87	87	71	85	85	27	26	26	74	86	86
# of classes prec > 0.9	44	28	31	48	31	33	39	26	26	25	19	19	53	34	35
avg recall %	73	86	87	78	85	86	71	84	84	27	25	25	74	86	86
# of classes prec > 0.95	37	25	29	38	26	28	36	23	23	24	19	19	45	33	35
avg recall %	71	85	86	75	84	85	70	83	83	26	25	25	71	87	86
# of classes prec = 1.00	22	22	27	21	23	26	19	20	20	18	18	18	28	28	31
avg recall %	56	83	85	63	83	84	57	81	81	16	26	26	62	85	86

5.2.2 Number of classes with precision above a certain threshold

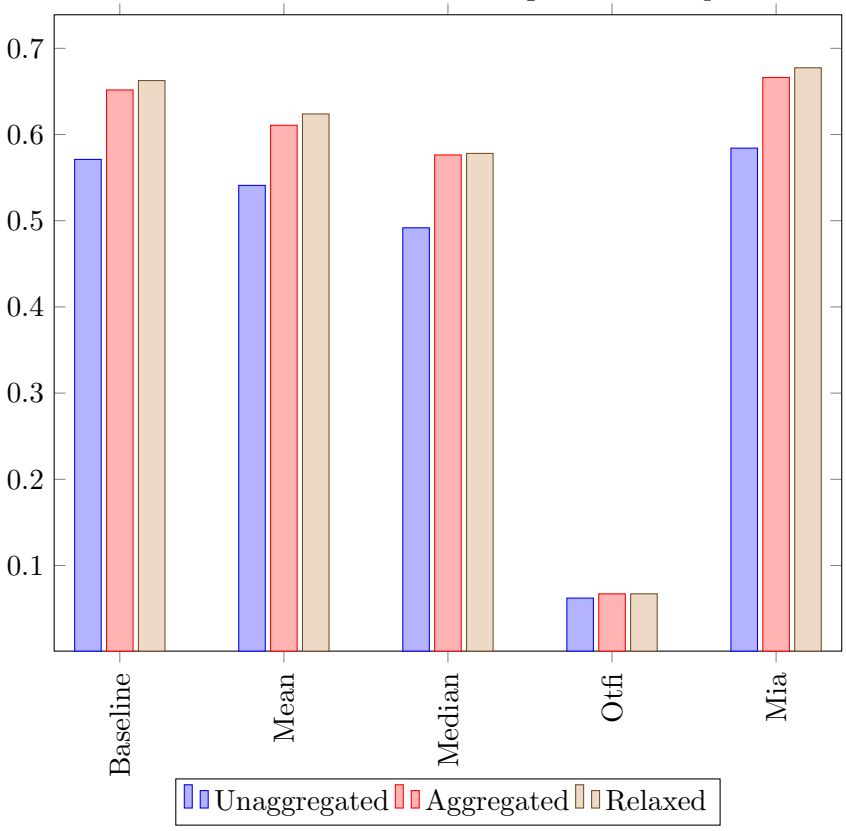
Average overall precision works as a metric to give a solid first view of the classifier's performance. However, it does not say much about the balance of the classifier. If the testing dataset had for example 90 objects of class 1 and the remaining 10 objects were of 10 different classes, the average overall precision could be 0.9. Sadly, that does not say if the classifier recognized all the classes, or only the one with 90 samples.

Because of that, it is good to look at the number of classes that were predicted with precision above some chosen threshold. The thresholds chosen for this experiment are 0.5, 0.8, 0.9, 0.95 and 1.00. Table [1] shows the number of classes with precision above these threshold for each method and each evaluation aproach along with the average recall of the classes in question underneath them.

5.2.3 Average overall recall

Apart from precision, the second studied measure was recall. As with the precision, the recall is averaged across all classes except for the negative one. OTFI method's very small recall again shows how ineffective the method is

Figure 10: Recall of the tested methods averaged across all positive classes



when it is used on the network data. [10]

From the results it can be clearly seen that the Strawman imputation, both with the mean value or the median value, work similar to the baseline imputation, if not a little worse. As could have been expected, the MIA imputation method on the other hand works a little better than the baseline method, which is because it contains the baseline in its own implementation, and then looks a little longer to see if it can find a better split. The OTFI method, however, did not work so well. The reason for that is that the network dataset has more than a half [3.2] of its values missing and the algorithm only uses the data that is not missing when finding the best split.

Conclusion

The focus of this thesis was to examine the problem of handling missing values in network traffic datasets. The thesis introduced a number of existing methods for imputation of missing data, but not all of them are relevant to the network datasets. Most of the algorithms do not scale well with bigger amounts of data missing or they run very slow on big datasets, so only the relevant were implemented.

The implemented methods were tested and compared, showing that the MIA imputation method provides a slight improvement in both prediction precision and prediction recall over the method used as the baseline. While the baseline's unaggregated averaged precision was 61%, the MIA method's averaged precision was about 65%. In the aggregated approach, baseline's score is 52% and MIA's 56%. The relaxed precision is 53% for the baseline and 58% for MIA. As for recall, baseline's average is 57% unaggregated, 65% aggregated and 66% relaxed, while MIA scored about one percent higher with 58% unaggregated, 66% aggregated and 68% relaxed.

The OTFI method on the other hand provided much worse results than the baseline, showing that it also belongs to the methods that do not scale well with heavy missingness because it uses only the non-missing data in computing the split statistics.

The Strawman imputation method ended up close to the baseline, but did not surpass it. The Strawman variation using mean of the features as the imputation provided better results than the median variation.

Apart from that, an analysis on the dataset was done to find the correlations between the features' values and their missingness. Some of the features proved to be missing exclusively together, but not all of them. Some features' values are significantly correlated and in some cases it was found out that when one the features is missing, then its correlated pair has a reasonable probability of being not missing. This can be used in the future to further enhance the classifier's performance.

Reference

[1] F. Tang and H. Ishwaran, "Random Forest Missing Data Algorithms," *Stat Anal Data Min: The ASA Data Sci Journal*, vol. 10, pp. 363–377, 2017, https://doi.org/10.1002/sam.11348.

[2] P. Allison, "Missing Data," *Quantitative Applications in the Social Sciences*, vol. 136, 2001.

[3] B. Twala, M. Jones, and D. Hand, "Good methods for coping with missing data in decision trees," *Pattern Recognition Letters*, vol. 29(7), pp. 950–956, 2008.

[4] R. Moir, "Defining Malware: FAQ," *Microsoft TechNet*, 2003, https://technet.microsoft.com/en-us/library/dd632948.aspx, visited 2018-05-06.

[5] A. Young and M. Yung, "Cryptovirology: Extortion-based security threats and countermeasures," 09 1996.

[6] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A Lustrum of malware network communication: Evolution and insights," *S&P 2017, 37th IEEE Symposium on Security and Privacy, May 23-25, 2017, San Jose, USA*, 05 2017, http://www.eurecom.fr/publication/5177.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, ser. Springer Series in Statistics. Springer New York, 2009, https://books.google.cz/books?id=tVIjmNS3Ob8C.

[8] E. Alpaydin and F. Bach, *Introduction to Machine Learning*. MIT Press, 2014, https://books.google.cz/books?id=7f5bBAAAQBAJ.

[9] A. Stevenson, *Oxford Dictionary of English*, ser. Oxford Dictionary of English. OUP Oxford, 2010, https://books.google.cz/books?id=anecAQAAQBAJ.

[10] J. Brabec, "Decision Forests in the Task of Semi-Supervised Learning," *Czech Technical University in Prague. Computing and Information Centre*, 2017.

[11] L. Breiman, "Random Forests," *Machine Learning*, vol. 45(1), pp. 5–32, 2001.

[12] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984, https://books.google.cz/books?id=JwQx-WOmSyQC.

[13] M. Hanselmann, U. Köthe, M. Kirchner, B. Y Renard, E. Amstalden van Hove, K. Glunde, R. Heeren, F. A Hamprecht, and R. H Morgan, "Towards digital staining using imaging mass spectrometry and random forests-technical report," 04 2009.

[14] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, ser. Springer Texts in Statistics. Springer New York, 2013, https://books.google.cz/books?id=qcI_AAAAQBAJ.

[15] L. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," vol. 41, pp. 77–93, 05 2004.

[16] Z. Zhou, *Ensemble Methods: Foundations and Algorithms*, ser. CHAPMAN & HALL/CRC MACHINE LEA. Taylor & Francis, 2012, https://books.google.cz/books?id=BDB50Ev2ur4C.

[17] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, Aug 1996, https://doi.org/10.1007/BF00058655.

[18] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006, https://doi.org/10.1007/s10994-006-6226-1.

[19] C. Chen and L. Breiman, "Using random forest to learn imbalanced data," 01 2004.

[20] M. Taboga, "Lectures on probability and statistics," 2010, https://www.statlect.com.

[21] H. Ishwaran, U. Kogalur, E. Blackstone, and M. Laure, "Random survival forests," *The Annals of Applied Statistics*, vol. 2, pp. 841–860, 2008.

[22] A. Feelders, "Handling missing data in trees: Surrogate splits or statistical imputation?" 03 2000.

[23] D. Powers, "Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation," vol. 2, 01 2008.

[24] Walber, "Precision and recall," 2014, https://en.wikipedia.org/wiki/Precision_and_recall#/media/File:Precisionrecall.svg, visited 2018-05-22, licence: CC BY-SA 4.0.

A Enclosed CD contents

The root directory on the enclosed CD contains the following items:

thesis.pdf: The PDF file with the thesis.

src: The source directory with all implemented code written in Python.