FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Malware classification on time series data through machine learning

Diogo Moutinho de Almeida

Mestrado Integrado em Engenharia Informática e Computação

FEUP Supervisor: Eugénio Oliveira

Company Supervisor: Jürgen Eckel

July 15, 2016

Malware classification on time series data through machine learning

Diogo Moutinho de Almeida

Mestrado Integrado em Engenharia Informática e Computação

July 15, 2016

Abstract

Malware classification can be a challenge considering the great amount of variety and increasing emergence of malware, as well as, available classification methods. For this reason, it is not unusual for a file to be considered a different type of malicious file by different classifiers. In fact, an assignment made by a single classifier might change through time, as a consequence of methods refinements or new discoveries. When using multiple independent classifiers, past classifications of a certain file might help on deciding on which one to trust. This dissertation aims at finding a way to facilitate this analysis by collecting historical data on files that already have assigned their final and last classification, and determine which machine learning algorithm can better predict a new file classification given this very same data. Besides the historical data, other characteristics shall be taken into account like: source of the file, filetype and filesize. The machine learning algorithms we have used are: C4.5, Random Forests, Multi-Layer Perceptron (MLP) and Long short-term memory (LSTM). It was possible with this approach to find an alternative way in finding the correct malware classification of a file, given a multiple number of classifiers, taking into account its classification history.

Resumo

A classificação de malware pode ser um desafio considerando o seu número actual e crescimento, bem como, os seus métodos de classificação. Por esta razão, é comum para um ficheiro ter diferentes classificações perante múltiplos classificadores. Para além disso, uma atribuição feita por apenas um classificador pode mudar ao longo do tempo, consequência do melhoramente dos métodos de classificação ou de novas descobertas. Quando em posse de diferentes e independentes classificadores, classificações passadas de um determinado ficheiro podem influenciar em quem confiar. Esta dissertação tem como objectivo encontrar uma forma de ajudar esta análise juntando dados históricos de ficheiros cuja classificação é final, e determinar qual o algoritmo de machine learning consegue prever melhor a classificação de um novo ficheiro. Para além dos dados históricos, outras características serão tidas em conta: a origem, tipo e tamanho do ficheiro. Os algoritmos de machine learning utilizados são: C4.5, Random Forests, Multi-Layer Perceptron (MLP) e Long short-term memory (LSTM). Espera-se que esta abordagem ofereça um método alternativo ou ajude na atribuição correcta de malware de um ficheiro, recorrendo a múltiplos classificadores e ao histórico de classificações.

Acknowledgements

I want to thank everyone who supported and contributed to the development of this dissertation. I would like to give a special thanks to Jürgen Eckel who accepted me into IKARUS Security Software and Prof. Eugénio Oliveira who was always available whenever I required help.

I am fully glad to have the friends and family beside and supporting me all these months.

In the end, a special thanks to my mother, father, brother and my beloved dog.

Diogo Moutinho de Almeida

If we knew exactly the laws of nature and the situation of the universe at the initial moment, we could predict exactly the situation of the same universe at a succeeding moment.

Henri Poincaré

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation and Goals . . . 1
- 1.3 Structure . . . 2

2 Literature Review **3**
- 2.1 Machine Learning . . . 3
- 2.2 Time series . . . 4
- 2.3 Neural Networks . . . 4
  - 2.3.1 Activation Functions . . . 6
  - 2.3.2 Recurrent Neural-Networks . . . 6
- 2.4 Decision Trees . . . 8
  - 2.4.1 ID3 . . . 9
  - 2.4.2 C4.5 . . . 10
  - 2.4.3 Random Forests . . . 10
- 2.5 Genetic Algorithms . . . 11
  - 2.5.1 Fitness function . . . 11
  - 2.5.2 Representation and Initialization . . . 11
  - 2.5.3 Selection . . . 11
  - 2.5.4 Reproduction . . . 12

3 Methods **15**
- 3.1 Data Acquisition . . . 15
- 3.2 Data Preprocessing . . . 16
  - 3.2.1 Decision Trees and MLP . . . 18
  - 3.2.2 Neural Networks . . . 18
- 3.3 Models . . . 19
  - 3.3.1 Decision Trees . . . 20
  - 3.3.2 Neural Networks . . . 20

4 Implementation **23**
- 4.1 Equipment, tools and libraries . . . 23
- 4.2 Data Preprocessing . . . 24
- 4.3 Models . . . 24
  - 4.3.1 Decision Trees: J48 and Random Forest . . . 24
  - 4.3.2 Neural Networks . . . 25

5 Results and Discussion **29**
5.1 Dataset 29
5.2 Genetic Algorithm and Neural Networks 30
5.2.1 MLP 30
5.2.2 LSTM 30
5.3 Comparisons 31

6 Conclusion **33**
6.1 Goals 33
6.2 Future Work 33

References **35**

List of Figures

2.1 Neuron[Kar16a] . . . 5
2.2 Model of a neuron[Kar16a] . . . 5
2.3 Neural-network architecture example[Nie16] . . . 5
2.4 Sigmoid Function . . . 7
2.5 Tanh Function . . . 7
2.6 RNN unfolded. . . . 7
2.7 LSTM block.[GSK+15] . . . 8
2.8 Simple decision tree . . . 9
2.9 Roulette wheel parent selection, figure from[Jon90] . . . 12
2.10 Cross-over. . . . 13
2.11 Mutation. . . . 13

3.1 General and simplified version of the data aquiring process. . . . 17
3.2 Model from (a) databases. . . . 18
3.3 Model from (c) databases. . . . 18
3.4 Model from (e) database. . . . 19

5.1 Number of classification modifications per classifier. . . . 29
5.2 Accuracy function over batches (LSTM) . . . 31
5.3 Loss function over batches(LSTM) . . . 32

LIST OF FIGURES

List of Tables

3.1 File classifications in time. 15
3.2 File attributes. 15
3.3 File classifications in time. 18
3.4 One Hot Encoding. 19
3.5 Genome's domains 20

5.1 Loss and accuracy results from the algorithms. 31

Abbreviations

MLP	Multi Layer Perceptron
LSTM	Long short term memory

Chapter 1

Introduction

1.1 Context

Today's world is witnessing a significant growth of technology. Both people and companies take advantage of the latest advancements to their own benefit. A great amount of those advancements contribute to the expansion of the cyberspace. Cyberspace can be defined as:

> "the complex environment resulting from the interaction of people, software and services on the Internet by means of technology devices and networks connected to it, which does not exist in any physical form." [Ltd12]

In an environment where everything is connected, it is mandatory to preserve confidentiality, integrity and availability for both systems and sensitive information. [Ltd12] Considering this, security is an evolving field of the utmost importance although it has been continually neglected. [Hum15]

Machine learning is another evolving field with great relevance. It aids in the development of model creations for businesses and recognize important patterns, either to understand the environment, or to predict the future.

This dissertation, offered by the company IKARUS Security Software, aims to unite both these fields, machine learning and security, in a way where detection of threats is more easily accomplished.

1.2 Motivation and Goals

The company IKARUS Security Software, located in Blechturmgasse 11 1050 Vienna, Austria, works on finding new security solutions for both private and public clients.

Motivated to improve the detection of threats in systems, it is the company and dissertation's goal to research and implement a system capable of deciding if a file is a threat or not, considering multiple classifications given by a set of classifiers over time.

Four machine learning methods were considered to analyse the data: C4.5, Random Forest, Multi-Layer Perceptron and Long short term memory.

1. Is it possible to predict a classification based on past variations of classifications across different classifiers?
2. Is it possible to improve certain algorithms by searching for optimal parameters?
3. What algorithm can best predict the classification?

1.3 Structure

This dissertation is divided in the following chapters.

In chapter 2 a literature review over the main topics related to the development of this thesis will be shown.

In chapter 3 the methodology and planning is presented.

In chapter 4 it's possible to read about the implementations of the most relevant methodologies.

In chapter 5 the results are demonstrated and discussed.

In chapter 6 a conclusion over the whole work is presented, as well as, future work.

Chapter 2

Literature Review

In Literature Review, an overview over the main topics related to the development of this thesis will be shown. Apart from giving a foundation on the subject, state of the art work will also be presented, which served as inspiration for this work. This chapter is structured in the following way:

- **Machine Learning** and **Time series**: an introduction on what these are and why are they important in the context of this thesis.
- **Neural Networks**, **Decision Trees** and **Genetic Algorithms**: an explanation of these subjects, as well, specific algorithms that were used in the project.

2.1 Machine Learning

Today's world is witnessing an increase in data traffic, with no sign of stopping[Cis16]. In 2013, the International Data Corporation predicted that 80% of customer data would be wasted[IDC16].And this is where machine learning can help with the ability to discover patterns in all this amount of data[BM01].

Regarding machine learning, [Mit06] says:

> [...] we say that a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E. Depending on how we specify T, P, and E, the learning task might also be called by names such as data mining, autonomous discovery, database updating, programming by example, etc.

[LS95] also states:

> Machine learning is the study of computational methods for improving performance by mechanizing the acquisition of knowledge from experience.

Machine learning tasks can be separated into three different classes[RN09]:

- **Supervised learning**: the machine has at its disposal both the inputs and outputs to learn.
- **Unsupervised learning**: the learning process is executed without any correct output available.
- **Reinforcement learning**: given a certain input and consequent action, the latter is evaluated without the correct action being disclosed.

2.2 Time series

A time series is a discrete or continuous sequence of discrete time points spaced at uniform time intervals. It can be used in different fields such as pattern recognition, finance, statistics, weather casting or many others[Cha04].

A time series analysis of a sequence of data points over time tells us what causal effects one or more variables change might have on other variables over time. Time series can be both continuous, when observations are made continuously on time, and discrete when observations are taken at specific time and at most cases equally spaced. Time series forecasting uses a model to predict future values based on previously observed values.

2.3 Neural Networks

Machine learning algorithms based on neural networks have been gaining popularity in such a way, that big companies such as Google[1] and Amazon[2] are developing their own libraries. These algorithms have shown great success when faced with problems regarding speech recognition[AAB+15], image recognition[SNY15] or recommendation systems[SNY15]. A neural network, to say in the words of Dr. Robert Hecht-Nielson[Cau89], is "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.". These networks are modelled after the architecture of animal brains, where neurons transmit and process chemical and electrical signals to other neurons[Bel14].

The Figure 2.2 represents the mathematical model of a single neuron as shown in Figure 2.1. The biological neuron receives input signals from its dendrites and outputs signals from its axons onto other neuron's dendrites through synapses[Kar16a].

The basic computational model works in a similar fashion: it receives several inputs ($\mathbf{x_0}$,$\mathbf{x_1}$,$\mathbf{x_2}$), multiplies each one of them by a certain weight[3] ($\mathbf{w_0}$,$\mathbf{w_1}$,$\mathbf{w_2}$)adding an offset (bias) and sums all these. If the result is above a certain threshold, it fires the neuron through an activation function[Nie16][Kar16a]. In the Figure 2.3, we can see an example of a neural network architecture, Multi-Layer Perceptron (MLP). It is a feedforward neural network which consists of an input layer, output layer and one or more hidden layers of nodes (neurons). The goal is to find a

[1] https://www.tensorflow.org/
[2] https://github.com/amznlabs/amazon-dsstne
[3] Weights quantify an input influence

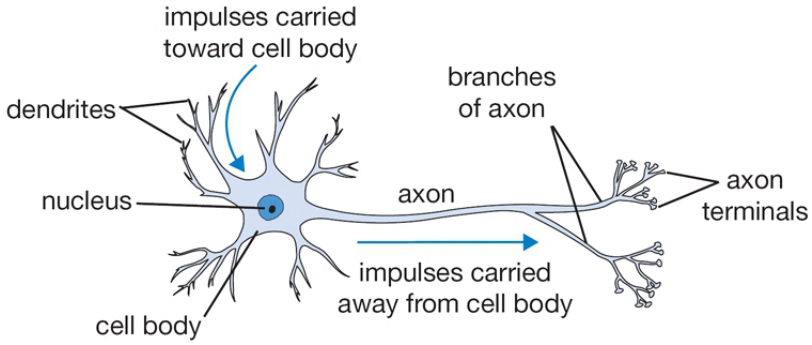

Figure 2.1: Neuron[Kar16a]

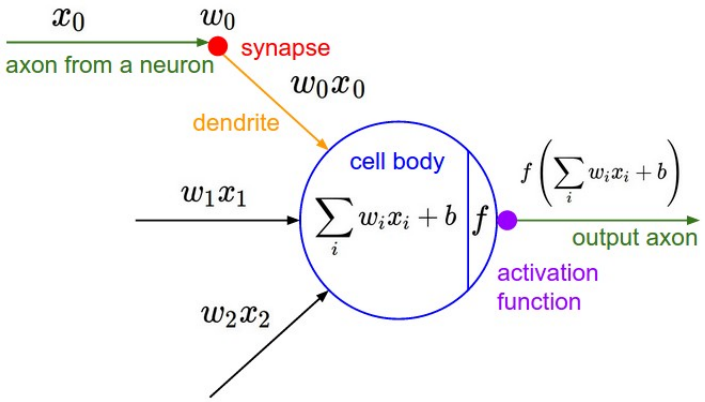

Figure 2.2: Model of a neuron[Kar16a]

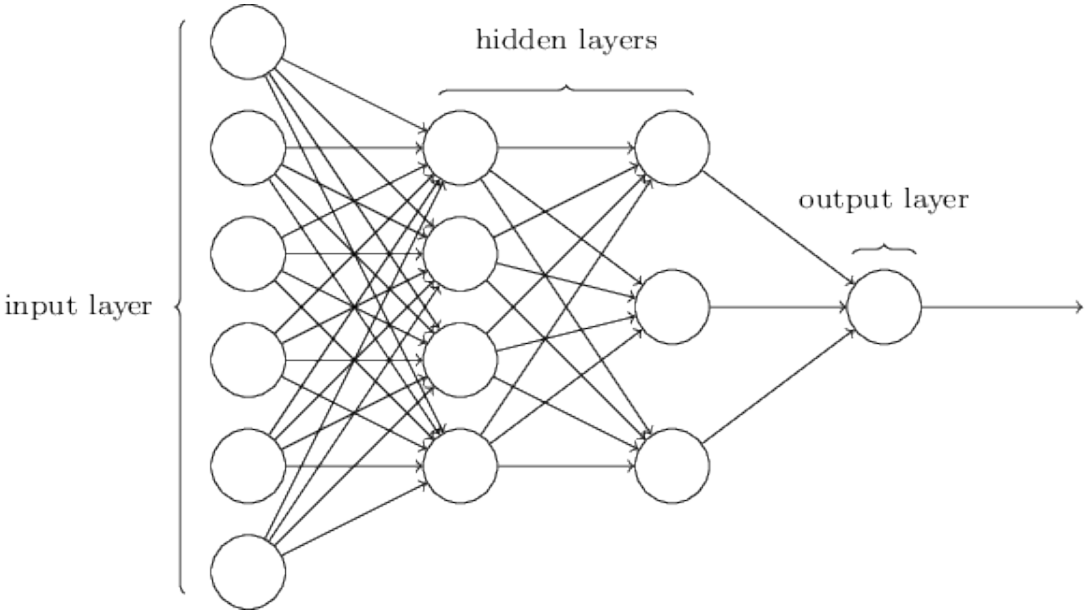

Figure 2.3: Neural-network architecture example[Nie16]

combination of weights and biases so that the output approximates the real output function for all the training inputs. In order to quantify how close the network is to find the optimal aproximate function, a loss functions needs to be defined. One example is the Cross-entropy[NBJ02] which was the one used in this dissertation. The way a neural network usually learns is through a method called Backpropagation, which consists in computing the gradient of a loss function with respect to any weight or bias in the network[RHW88]. It shows how quickly the changes in the weights and biases influence the loss. Along with backpropagation an optimization function needs to be used to update the weights. The optimization function used in this dissertation is the Stochastic gradient descent (SGD)[Bot12].

2.3.1 Activation Functions

- **Sigmoid** The sigmoid function has the form:

$$\sigma(x) \equiv \frac{1}{1+e^{-x}} \tag{2.1}$$

  It perfoms an operation over a real-valued number and returns a number into range between 0 and 1. Large negative numbers become 0, while large positive numbers become 1. This function has been used for a long time, because it gives a clear interpretation of the firing rate of a neuron, but has suffered a decline in use because of its easiness to saturate and kill gradients. Another undesirable feature of this function, although not as severe, is its non-zero centered outputs, which result in a slower convergence.

- **Tanh** The sigmoid function has the form:

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.2}$$

  It perfoms an operation over a real-valued number and returns a number into range between 1 and -1. Although saturation also happens, the zero centered values give this function an edge over the sigmoid because of its convergence.

2.3.2 Recurrent Neural-Networks

Recurrent neural networks are a feedforward neural network architecture with the capability of processing a sequence of values[BC16]. This is possible by including and connecting edges that span adjacent sequence or time steps. In a specific step t, nodes get input both from the current data point $x(t)$, as well as, from the hidden node values on the previous step *t-1* [Lip15]. These interactions are represented in the Figure 2.9.

If we take a look at the Figure 2.9, we can see the similarities between RNN and feedforward neural networks, and so, these networks can take advantage of backpropagation, specifically one called *backpropagation through time* (BPTT), which computes the gradient across the many time steps[Wer90]. These networks can suffer from a significant problem. When learning long-term

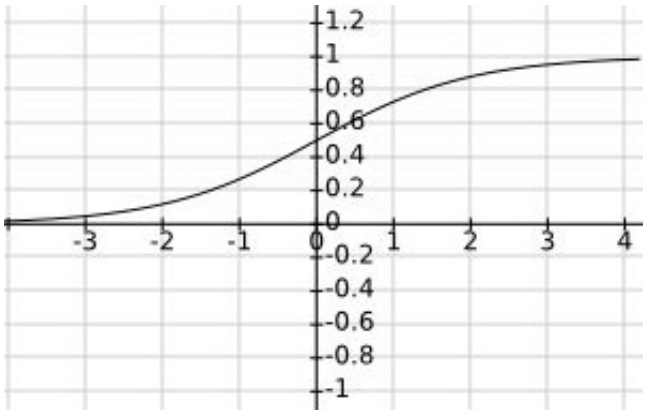

Figure 2.4: Sigmoid Function

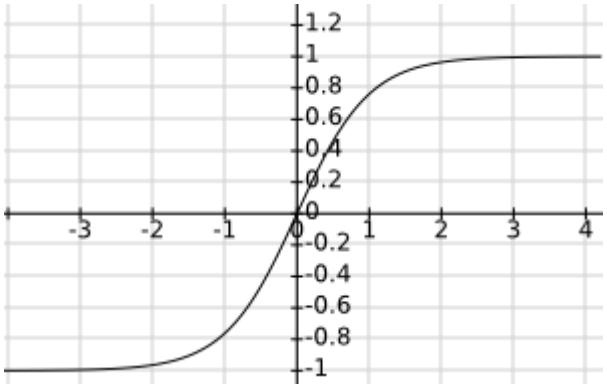

Figure 2.5: Tanh Function

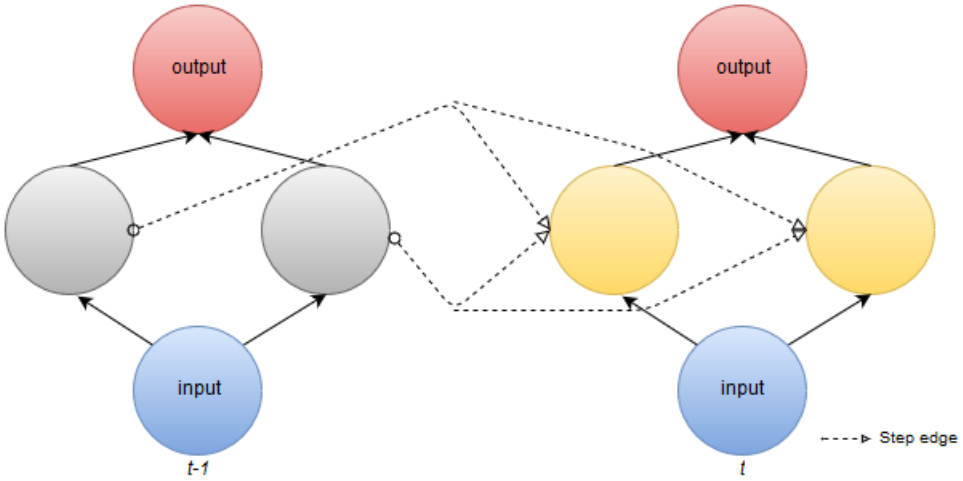

Figure 2.6: RNN unfolded.

dependencies, as result of an unstable relationship between the parameters and the dynamics of the RNN, a vanishing or exploding gradient can occur[BSF94].

2.3.2.1 Long short-term memory

Long short-term memory (LSTM), initially introduced by [HS97] and improved further by other people, is a special kind of RNN designed to fight the long-term dependency problem. This architecture is able to preserve the error backpropagated through time and layers, allowing it to learn over many time steps.[GSK+15] This architecture introduces a new structure, and one variant can be seen in the Figure 2.7.

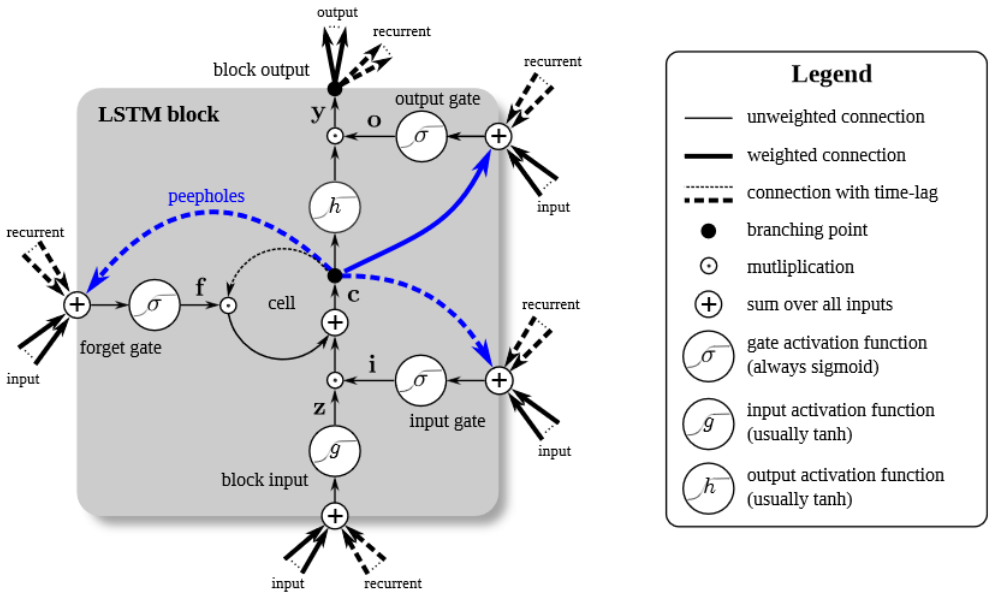

Figure 2.7: LSTM block.[GSK+15]

This is an improved version of the one proposed by [HS97]. The most relevant new features are:

- **Forget Gate**: ability to reset its own state giving the ability to forget past knowledge[GSC00].
- **Peephole Connections**: ability for the gates to look at the cell state in order to learn precise timings[GS00].

2.4 Decision Trees

The process which allows to create general models from analyzing a set of instances to classifiy objects is called inductive inference[PKSR02]. Decision trees are one of the most used methods for this process[Mit06]. Given a set of instances, where each one has a group of features and corresponding labels, a decision tree algorithm generates a sequential model composed by a sequence of tests, where the result will decide what branch of the tree to follow as in the Figure 2.8 [Kot13][Sal94].

There are two main types of decision trees:

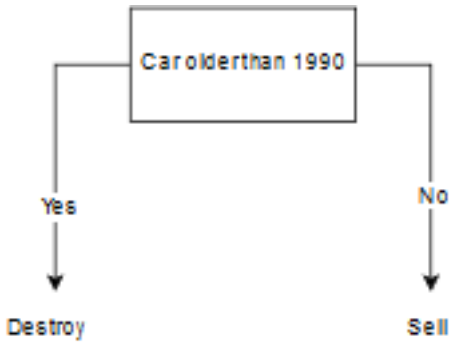

Figure 2.8: Simple decision tree

- **Classification Tree**: the output is a class.
- **Regression Tree**: the output is a real number.

Famous examples of decision trees are:

- **ID.3**[Qui86]: Iterative Dichotomiser 3, classification algorithm.
- **C4.5**[Qui93]: extension of ID3 algorithm, also classification algorithm.
- **CART**[BFOS84]: Classification And Regression Tree.
- **Random Forests**[Ho95][Bre01]: both classification and regression algorithm.

For this thesis, only C4.5 and Random Forests were considered.

2.4.1 ID3

In order to generate a decision tree we need to choose which attribute to test at each node of the tree. The classification algorithm ID3[Qui86], originally developed by J. R. Quinlan, uses a statistical property called information gain which is obtained from a measure called entropy.

- **Entropy**:

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2(p_i)$$
$$Entropy(S,A) = \sum_{c \in A} P(c)E(c)$$

- **Information Gain**:

$$Gain(S,A) = Entropy(S) - Entropy(S,A)$$

Steps:

- For the set S, calculate the entropy of every attribute.

- The attribute with largest information gain becomes the decision node.
- If a branch has an entropy bigger than 0, it needs to be split further, otherwise it becomes a leaf node.
- Recursively apply the algorithm on the non-leaf branches with the remaining attributes.

2.4.2 C4.5

C4.5[Qui93] is a classification algorithm developed by J. R. Quinlan, as an extension to his earlier work on ID3[Qui86]. Some improvements over his previous algorithm are:

- Capable of handling continuous values.
- C4.5 allows building a decision tree with unknown attribute values in their instances, by evaluating the gain only with the known values.
- After the tree creation, the algorithm tries to replace branches with leaf nodes, if it results in a lower expected error rate.
- After a decision tree is generated, it is possible to classify new instances with unknown attribute values by estimating probabilities of the multiple outcomes.

2.4.3 Random Forests

Random Forests[Ho95][Bre01] is an ensemble learning method for classification and regression. An ensemble is often more accurate than the single classifiers composing the ensemble[MO11]. This algorithm is characterized by the following steps:

- N subsets are sampled from the initial set S randomly with replacement, creating what it is called bootstrap samples[BB96]. This process is called bagging.
- N trees are created from the bootstraped samples..
  - During creation, at each node m features are selected at random from all the features.
  - The feature which provides the best split shall split the node.

When the forest is complete and a new instance appears, the prediction depends on what type of problem we are dealing with:

- **Classification:** every tree gives its prediction and the chosen one is found by a voting majority.
- **Regression:** an average or weighted average of all the predictions.

2.5 Genetic Algorithms

Genetic Algorithms first developed by John H. Holland[Hol75], weren't developed to solve specific problems but to import the mechanisms of natural adaptation into computer systems[Mit96]. Nowadays, genetic algorithms are used as probabilistic search procedures which work on large spaces[GH88].

According to [RN09], a genetic algorithm:

> [...] starts with a set of one or more individuals and applies selection and reproduction operators to "evolve" an individual that is successful, as measured by a fitness function.

[RN09] also states that before we apply this algorithm to a problem four questions need to be answered first:

- **Q1**: What is the fitness function?
- **Q2**: How is an individual represented?
- **Q3**: How are individuals selected?
- **Q4**: How do individuals reproduce?

2.5.1 Fitness function

The fitness function depending on the problem, takes an individual as input and returns a real number.

2.5.2 Representation and Initialization

The set of parameters of an individual are called chromosome[Mit96] and they are typically represented by strings of bits. They are discretized on some power of 2, resulting in a specific number of bits per parameter[Whi94]. When a parameter is continuous it is still possible to discretized it, given that the chosen intervals give a level of precision to the output[Whi94]. After having an established representation, it is important to have a good initial population because of its impactful role, both in finding a quality solution and in the time taken to accomplish it[KM14]. However, most of the cases it is generated randomly, unless there is some knowledge about the solution.

2.5.3 Selection

Given a certain population, it is necessary to select individuals to reproduce and create new offspring. As a general strategy, it is attributed a fitness value to each individual, and the most fit individual has the most probability of survival and reproduction[LL11]. There are many selection methods, and just a few relevant ones to this thesis will be introduced.

2.5.3.1 Roulette-wheel selection

In this method, individuals are assigned sectors of the roulette wheel proportionally to their fitness, making the random selection biased towards fitter individuals[Jon90]

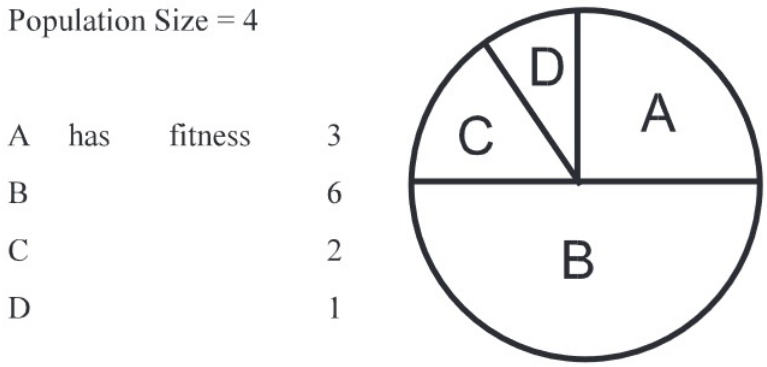

Figure 2.9: Roulette wheel parent selection, figure from[Jon90]

The selection probability of the i-th individual with a fitness value of w_i is given by:

$$p_i = \frac{w_i}{\sum_{i=1}^{N} w_i} (i = 1, 2, ..., N) \tag{2.3}$$

2.5.3.2 Rank Selection

In the previous method, if an individual has a fitness value too different from the rest of the population, it will overwhelm the chances of getting selected over the rest. In a way to to prevent that, the rank selection method was introduced. This method sorts all the individuals from best to worst according to their fitness level. These ranks will correspond to their new fitness values, where fitness with value 1 is the worst and the best will have a fitness value equal to the number of individuals in the population.

$$p_i = \frac{rank(i)}{\sum_{i=1}^{N} w_i} (i = 1, 2, ..., N) \tag{2.4}$$

2.5.3.3 Elitism Selection

When building a new population this type of selection allows the best individuals from the old population to survive and join the new population without any modification.

2.5.4 Reproduction

After selecting all the individuals, it is time to generate a new population. The chosen chromosomes are randomly paired for reproduction, which is accomplished by two processes[RN09]:

- **Cross-over**: For each pair of individuals, a cross-over point is randomly chosen. One of the offsprings shall have the first genes (or bits) until the cross-over point from one parent and

the last genes from the cross-over point from the other parent. The second offspring shall have the opposite. This process is represented in the Figure 2.10 with the red line indicating the cross-over point.

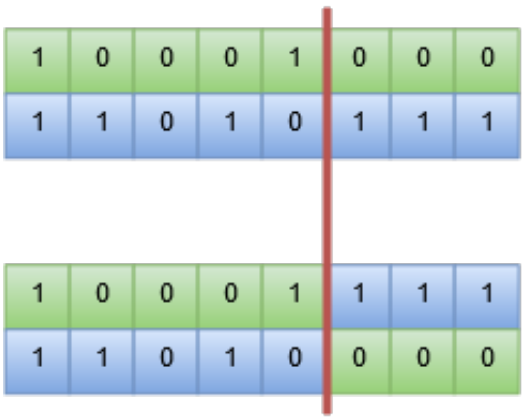

Figure 2.10: Cross-over.

- **Mutation.**: With an inpendent probability it is possible for each gene from the offspring to be flipped, as in the Figure 2.11.

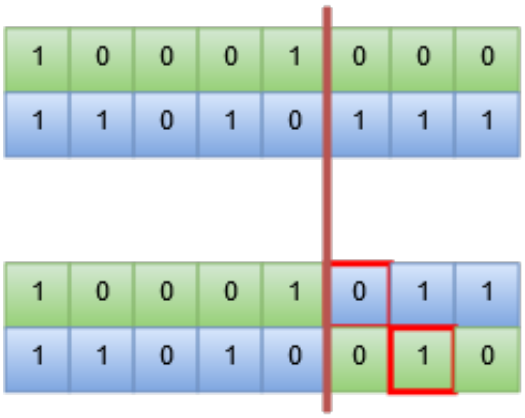

Figure 2.11: Mutation.

Chapter 3

Methods

In this chapter, it is presented the methodology and planning.

However, before we got into the methods themselves, we had to define what we required so we could solve the problem.

Ultimately, there was a need for data on a great amount of files. For each file we needed different classifications assigned by different classifiers in seperate points in time, as well as, its attributes and final classification.

The table 3.1 and 3.2 are a representation of what we wanted our data to be.

Table 3.1: File classifications in time.

File	Classifier	Classification	Date
1	C_1	None	2016-01-09 15:19:31
1	C_1	Trojan.Win.123	2016-01-19 15:19:31
1	C_2	Trojan.HON	2016-01-09 15:19:31
1	C_2	Win.Trojan123	2016-01-19 15:19:31
1	C_3	VirusOSX.Generic	2016-01-09 15:19:31
1	C_3	None	2016-01-19 15:19:31

Table 3.2: File attributes.

File	Type	Size	Final Classification
1	exe	9999	Trojan123

0

3.1 Data Acquisition

The first action towards solving the problem was acquiring the data needed for preprocessing and analyse:

- File size: size of the file in bytes

- File type: type of the file, according to its extension
- File source: from where the file originated
- Classifier
- Classification
- Date of classification
- Final Classification...

However, to get this kind of data we needed to go through multiple preliminary steps.

The Figure 3.1 shows sequentially from (a) to (e), the stages required to obtain the data in the format above. The files used for this thesis came from different external sources into our environment. An explanation of the flow of the Figure 3.1 is given:

1. **(a)** represents multiple databases which save the details of the new incoming files to be analysed. New files come every second. Databases already existing prior to the dissertation. Figure 3.2.
2. Machine **(b)** monitors regularly the databases in (a) for new incoming files and saves their identifier to the database (c). The monitoring stops when the count of files reaches 100000.
3. Database **(c)** saves the files' identifier. It also stores its entry date, as well as a *state*. This state helps in figuring out if it is time for the file to be classified again. A file has to be classified again if: $current\_date >= entrydate + days(2^{state})$ The max value of state is 4 and is incremented each time a file's details is given. Figure 3.3.
4. **(d)**: Multiple parallel machines with 12 different classifiers which regularly check (c) for files eligible to be classified.After each classifier has a result, it gets saved in the database (e) (Figure 3.4) with the desired format, ready to be preprocessed and analysed by the multiple machine learning algorithms in (f).

3.2 Data Preprocessing

After acquiring all the necessary data, the next step is to transform the data so each of our algorithms can read it.

The first problem is shared by all algorithms and it regards the different classifiers results. Each classifier gives a result in a different naming standard. For example, classifier *A* might call a trojan from the family *DOG* "TROJ.DOG", while classifier *B* might call it "dog_trojan". Although they mean the same, their representation also has to be the same. Looking over the classifications given, it is possible to notice that most possess: type, platform and one or two families. These features can be extracted using regular expressions for each different classifier.

Other problems are specific to one or more algorithms.

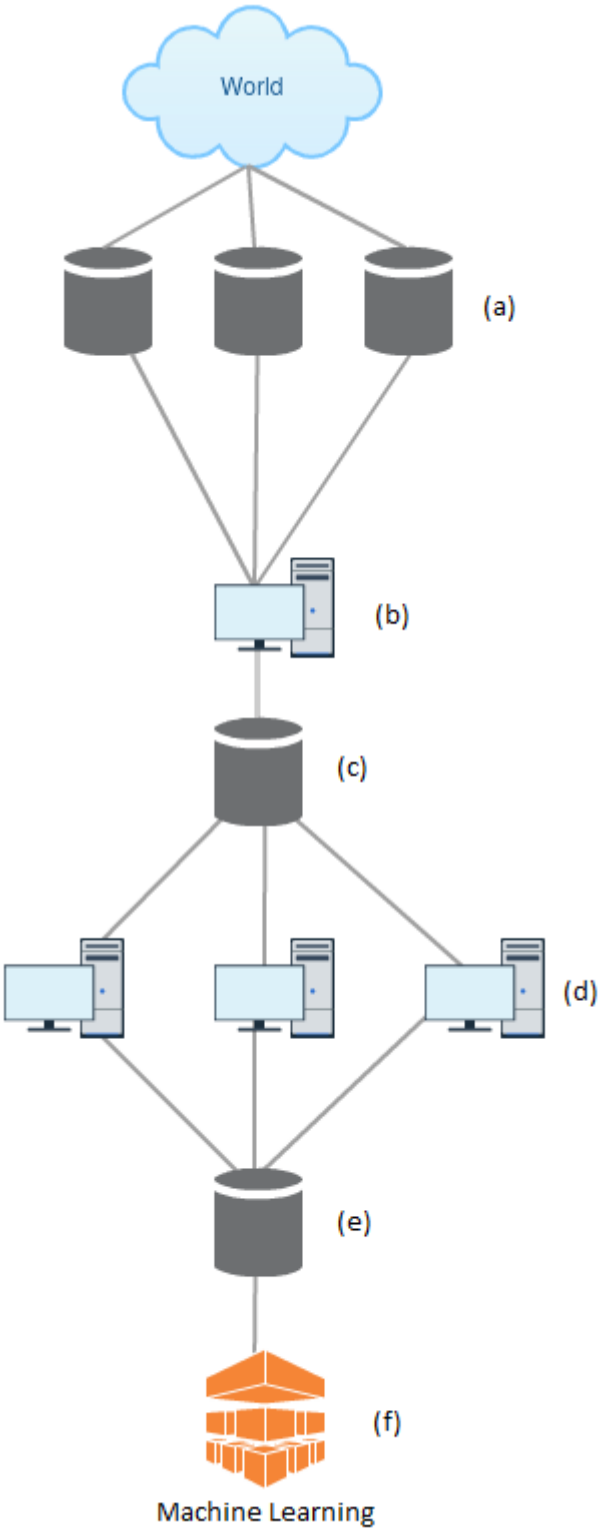

Figure 3.1: General and simplified version of the data aquiring process.

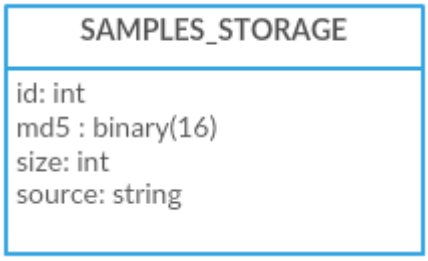

Figure 3.2: Model from (a) databases.

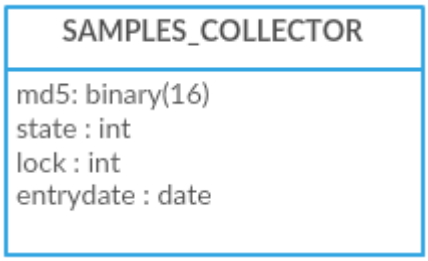

Figure 3.3: Model from (c) databases.

3.2.1 Decision Trees and MLP

Something that all these algorithms lack in common is the core ability to analyse data over time. One way to overcome this is by manipulating the data and flatten it. Using the tables 3.1 and 3.2 as an example, a sample after getting flattened would be in the following format ready for the decision trees(C*N*TS*Z* = Classifier N Time Step Z):

Table 3.3: File classifications in time.

Type	**Source**	**Size**	**C1TS1**	**C1TS2**
exe	source_sky	9999	None	Trojan.Win.123

C2TS1	**C2TS2**	**C3TS1**	**C3TS2**	**Final Classification**
Trojan.HON	Win.Trojan123	VirusOSX.Generic	None	Trojan123

3.2.2 Neural Networks

Only categorical values will be considered.

In order to feed a neural network categorical values, as the ones we are working with, these need to be encoded as numerical variables. One typical method is to use one hot encoding as in Table 3.4.

Another problem is the unexistence of some values[1] which were represented by the number of 0 equal to the size of each encoded categorical variable. An unknown value in the table 3.4 would

[1]For example a classifier failed to give a result from a file in certain day.

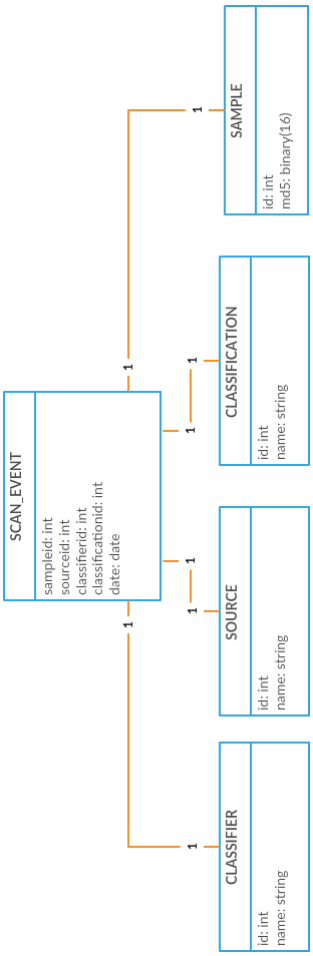

Figure 3.4: Model from (e) database.

Table 3.4: One Hot Encoding.

Categorical value	**Encoded**
source_sky	0001
Virus	0010
Classifier1	0100
Trojan	1000

be represented as *0000*.

3.3 Models

In this section, it will be introduced what kind of model specifications were chosen.

3.3.1 Decision Trees

For each algorithm, there are some parameters that can tuned in order to change the result accuracy and loss.

1. C4.5:
   (a) Pruning confidence threshold: 0.25
   (b) Number of instances per leaf: 2 (default)

3.3.2 Neural Networks

Both MLP and LSTM will have access to a training set and a testing set to find out how well they can perform on an independent set of data. Also, both architectures have a lot of parameters that need to be defined:

- Number of hidden Layers
- Number of nodes per layer
- Dropout[SHK+14]
- Batch size
- Activation
- Learning rate
- Decay[Kar16b]

Finding the optimal combination of parameters can become a very difficult task, and for this reason, a genetic algorithm was designed to help finding the best values for each parameter.

The parameter domain was defined in the following manner:

Table 3.5: Genome's domains

Number of hidden Layers	**Number of nodes per layer**	**Dropout**
[1,3]	[0,511]	[0.3,0.8]

Batch Size	**Activation**	**Learning Rate**	**Decay**
{8,16,32,64,128,256}	{'tanh','sigmoid'}	$[10^{-1},10^{-4}]$	$[10^{-3},10^{-4}]$

The fitness value is influenced by both the loss of the training set, as well as, the number of hidden layers in the architecture. $fitness\_value = rre * serr + rrs * ssize$

- **rre**: reversed rank error
- **serr**: error scaling
- **rrs**: reversed rank size

- **ssize**: size scaling

Both scalings were defined to 0.5. The selection method decided is a hybrid of ranked and elitism selection. After a rank selection is made and their new population is generated and their fitness function calculated, the top 5% of the old population replace the worst 5% of the current one if they are better. And then, a new rank selection process is made. The initial population is 6 with a number of 100 generations. The mutation rate is 0.03, and a cross rate of 0.5 was also added to increase randomness.

Chapter 4

Implementation

In this chapter, it is presented the most relevant implementations, as well as, the equipment, tools and libraries used.

4.1 Equipment, tools and libraries

The programming language used for this dissertation was Python 2.7 under the Anaconda[1] platform on Windows 10. This platform is a high performance distribution of Python designed to help with data science projects. It was used in all stages: data acquisition, data preprocessing, model creation and analysis.

Weka[2] was the tool chosen to analyse the data with the decision trees algorithms: C4.5 and Random Forest. Weka is a collection of machine learning algorithms for data mining tasks developed in Java. A python library called python-weka-wrapper[3] was used to communicate with the java virtual machine in order to use Weka functionalities.

The development of MLP[4] and LSTM[5] were made possible through a minimalist and highly modular neural network library called Keras[6]. This library runs on top of TensorFlow or Theano[7], libraries designed tp perform mathematical operations involving multi-dimensional arrays efficiently. Theano was the only option available, because TensorFlow is not available for Windows. Keras supports CUDA for efficient computations when using an NVIDIA graphic card. The one used was GeForce GTX 960[8].

[1] https://www.continuum.io/
[2] http://www.cs.waikato.ac.nz/ml/weka/
[3] https://pypi.python.org/pypi/python-weka-wrapper
[4] Multi Layer Perceptron
[5] Long Short Term Memory
[6] https://github.com/tensorflow/tensorflow
[7] https://github.com/Theano/Theano
[8] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-960/specifications

4.2 Data Preprocessing

In the data preprocessing stage the most relevant process is the one-hot-encoding of the categorical values which used the pandas[9] library.

An example of its usage:

```
1 import pandas as pd
2 (...)
3 encoded = pd.get_dummies(list_classifications)
4 (...)
5 type = encoded['trojan']
```

Listing 4.1: One-hot-encoding

The function **get_dummies()** returns a dictionary where the categorical value is a key to an array with size equal to the size of *list_classifications*, similar to the Table 3.4.

4.3 Models

In this section, it is presented the most relevant implementations features of the models.

4.3.1 Decision Trees: J48 and Random Forest

Both decision trees share the same implementation, because both use the same functions used by python-weka-wrapper, only differing in the parameters called:

The dataset file is a csv file where each line is in the format given by the Table 3.3.

```
1 evaluate_cross("j48","weka.classifiers.trees.J48",["-C", "0.25","-M","2"],
    dataset_path,10,verbose)
```

Listing 4.2: C4.5

```
1 evaluate_cross("randomforest","weka.classifiers.trees.RandomForest",["-I", "50","-K
    ","0","-S","1"],dataset_path,10,verbose)
```

Listing 4.3: Random Forest

The function *evaluate_cross()* is defined by:

```
1 import weka.core.jvm as jvm
2 from weka.core.converters import Loader
```

[9] http://pandas.pydata.org/

```
3  from weka.classifiers import Classifier, Evaluation
4  from weka.core.classes import Random
5  import weka.core.serialization as serialization
6
7  def evaluate_cross(type_,classname_,options_,dataset_path,num,verbose=None):
8      try:
9          if verbose:
10             print "\n########\nInitiating "+type_+" classifier:"
11         jvm.start(max_heap_size="4096m")
12         data = dataset_loader(dataset_path,verbose)
13         cls = Classifier(classname=classname_,)
14         cls.options = options_
15         evaluation = Evaluation(data)
16         evaluation.crossvalidate_model(cls, data, num, Random(412))
17         if verbose:
18             print(evaluation.summary())
19         jvm.stop()
20     except:
21         print sys.exc_info()
```

Listing 4.4: evaluate_cross

4.3.2 Neural Networks

4.3.2.1 Input shape

Each architecture requires a different input data shape.

- **LSTM**:(number_of_samples,timesteps,timestep_dimension)
- **MLP**:(number_of_samples,dimension)

4.3.2.2 Genetic Algorithm for Hyperparameter Search

The search for the optimal parameteres was executed using the following genetic algorithm:

```
1  population = generate_initial_population(POPULATION_SIZE)
2  population = simulate(population)
3  pop_size = len(population)
4  old = population
5  for j in xrange(NUM_GENERATIONS):
6    new_population = []
7    population = sort_rank(population)
8    for i in xrange(pop_size/2):
9      ind1, ind2 = select(population)
10     new1, new2 = cross_mutate_genes(population[ind1][GENOME],population[ind2][
           GENOME])
11     new_population.append(new1)
```

```
    new_population.append(new2)
  new_population = simulate(new_population)
  new_population = sort_rank(new_population)
  new_population = (sort_rank(new_population+population[-int(elite_rate*len(
      population)+0.5):]))[-len(population):]
  population = new_population
  old = population
  elite = population[-int(0.05*len(population)+0.5):]
```

The function *simulates()* runs the neural network with the defined parameters of the individual genome returning the loss of the training set:

```
def simulate(population):
  population_ = []
  idx = 0
  for individual in population:
    q = Queue()
    p = Process(target=golive, args=(individual,q))
    p.start()
    p.join()
    val_loss = q.get()
    population_.append([individual,val_loss,gene_val(individual[0]),0,0,0])
  return population_
```

4.3.2.3 Model Creation

The following code is responsible for creating the LSTM mode:

```
from keras.models import Sequential
from keras.layers.core import Activation, Dense, Dropout
from keras.optimizers import SGD
from keras.layers.recurrent import LSTM

def run(output_dim,input_shape,num_layers,hidden_sizes,dropout,batch_size_,
    activation_,lr_,decay_,ftrain,ftest,datapath,num_samples):

    f = h5py.File(datapath, "r")
    rsequences = True
    if num_layers == 1:
        rsequences = False

    model = Sequential()
    model.add(LSTM(hidden_sizes[0],input_shape=input_shape, init='glorot_uniform',
        inner_init='orthogonal',
                   activation=activation_, inner_activation='hard_sigmoid',
                   return_sequences=rsequences))
```

```
    model.add(Dropout(dropout))
    for i in xrange(num_layers-1):
        if i+2 == num_layers:
            rsequences = False
        model.add(LSTM(hidden_sizes[i+1],return_sequences=rsequences))
        model.add(Dropout(dropout))
    model.add(Dense(output_dim))
    model.add(Activation('softmax'))
```

The MLP model is coded the same way, but instead of: *model.add(LSTM(...)* it becomes *model.add(Dense(...)*.

4.3.2.4 Model compilation and training

Both models have the same compilation and training process. This block compiles the network with the chosen loss function and optimizer function.

```
    sgd = SGD(lr=lr_, decay=decay_, momentum=0.9, nesterov=True)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=["
        accuracy"])
```

The following block sets a splitting size of the batches. Each time the model trains a new batch it evaluates the training set and calculates its loss. If it hasn't improved for 2 batches, it stops earlier and returns the current loss function. It trains all batches *nb_epoch* times.

```
    test_samples = int(num_samples * 0.1)
    num_samples = num_samples - test_samples
    split = batch_size_

    early_stop = 0
    val_loss = 1000

    for e in range(nb_epoch):
        print("\n#######\n#######\nEpoch %d Early_stop %d" % (e ,early_stop))
        start = 0
        end = 0
        for i in range(int(num_samples/split)):

            end = end + split
            if end > num_samples:
                end = num_samples
            if start >= end:
                break

            print str(start)+"--"+str(end)+"/"+str(num_samples)
            sc = model.fit(f['arr'][start:end], f['sol'][start:end],
                batch_size=batch_size_,
```

```
23                nb_epoch=1,
24                shuffle=True)
25            end  += 1
26            start = end
27
28        score = model.evaluate(f['arr'][num_samples+1:], f['sol'][num_samples+1:],
              batch_size=batch_size_, verbose=1)
29
30        if score[0] < val_loss:
31            val_loss = score[0]
32            early_stop = 0
33        else:
34            early_stop += 1
35            if early_stop == 2:
36                print("Val_loss didn't improve for 5 times, leaving.")
37                return score[0]
38    return val_loss
```

Chapter 5

Results and Discussion

In this chapter an evaluation of the work done and results shall be presented.

5.1 Dataset

A dataset file was collected in the end of the 16 days (2^{state}), but also in a random day inbetween.

The Figure 5.1 has a chart representing how many times a classifier (identifiers from 115 to 126) changed its classificaton for a given file.

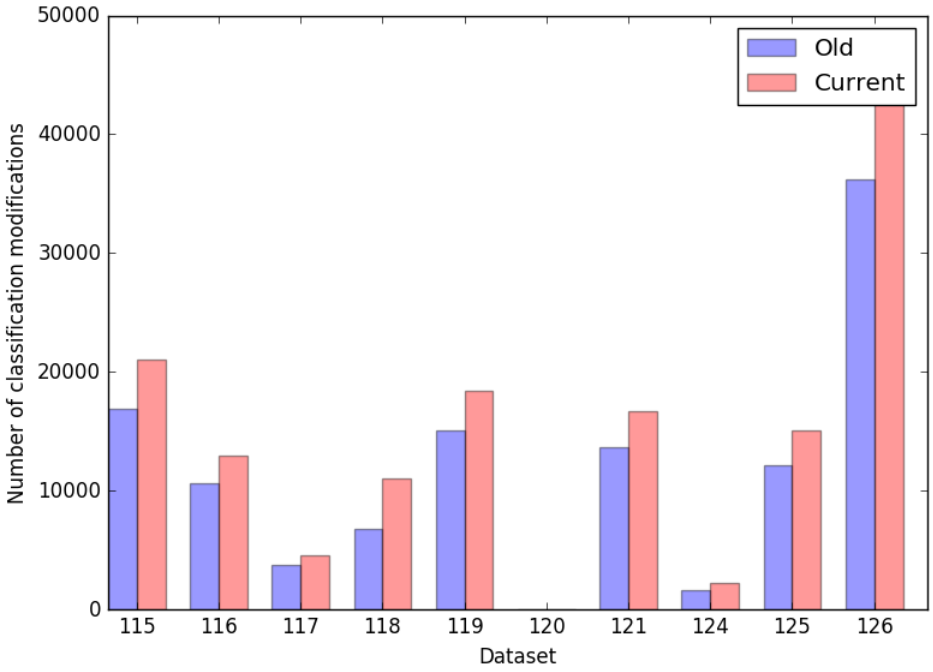

Figure 5.1: Number of classification modifications per classifier.

The *old* dataset had a total of 4625656 instances, while the *current* dataset had a total of 5308415 instances.

Looking at the total number of instances, the number of modifications is quite underwhelming. This means that in many cases a classification did not vary.

One explanation might be the short time frame of 16 days.

5.2 Genetic Algorithm and Neural Networks

5.2.1 MLP

When applying the genetic algorithm, the most fit individual was the one with the following parameters:

- **Number of hidden layers**: 1
- **Number of nodes layers**: 973
- **Dropout**: 0.6
- **Batch Size**: 64
- **Activation**: tahn
- **Learning rate**: 0.002043652998
- **Decay**: 5.54231178799e-06

When running the testing set, the network was able to have a loss of 14% with accuracy of 91%. Only one epoch was needed.

5.2.2 LSTM

When applying the genetic algorithm, the most fit individual was the one with the following parameters:

- **Number of hidden layers**: 1
- **Number of nodes layers**: 625
- **Dropout**: 0.5
- **Batch Size**: 64
- **Activation**: sigmoid
- **Learning rate**: 0.014042720195
- **Decay**: 3.43065325719e-06

When running the testing set, the network was able to have a loss of 9% with accuracy of 98%. Only one epoch was needed.

5.3 Comparisons

Table 5.1: Loss and accuracy results from the algorithms.

Algorithm	**Accuracy**	**Loss**
C4.5	95%	29%
Random Forest	94%	26%
MLP	91%	14%
LSTM	98%	9%

Looking at the Table 5.1, LSTM is clearly the better algorithm with an accuracy of 98% and only 9%. Both decision trees algorithms showed a similar result. And MLP had the worst result.

The Figure 5.2 and 5.3 show both the accuracy and loss evolution along with the batches from LSTM. Until approximately the batch 1025, both functions present an apparent random behaviour and then convert very rapidly.

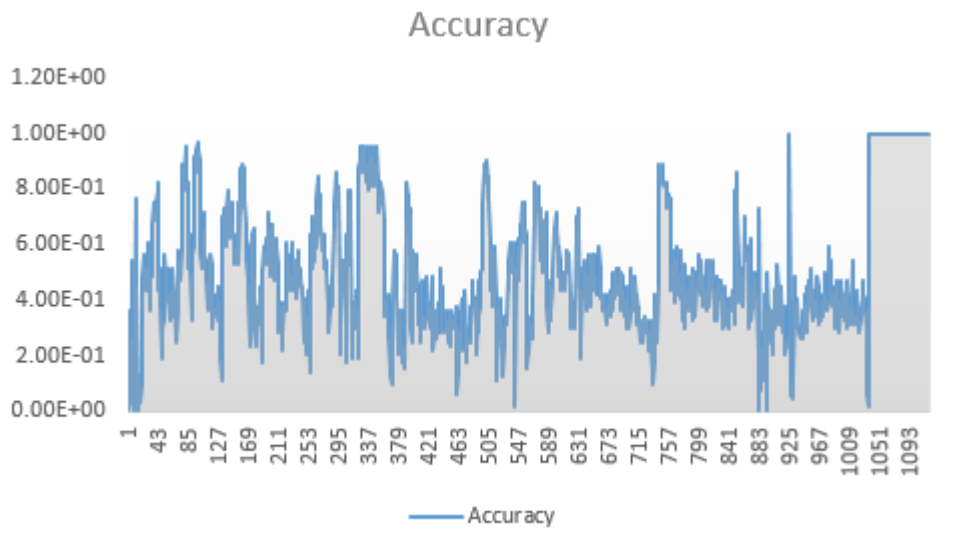

Figure 5.2: Accuracy function over batches (LSTM)

These overall results were better than expected across all the architectures. One explanation can be found in the Figure 5.1. Because there aren't many modifications, the need for analysis over time is reduced improving the loss and accuracy values of every algorithm.

Because there is still a relevant number of classification modifications over time, LSTM comprove that its capabilities for this type of data makes it better than the other architectures where they don't have the features to analyse time series data, needing further data preprocessing.

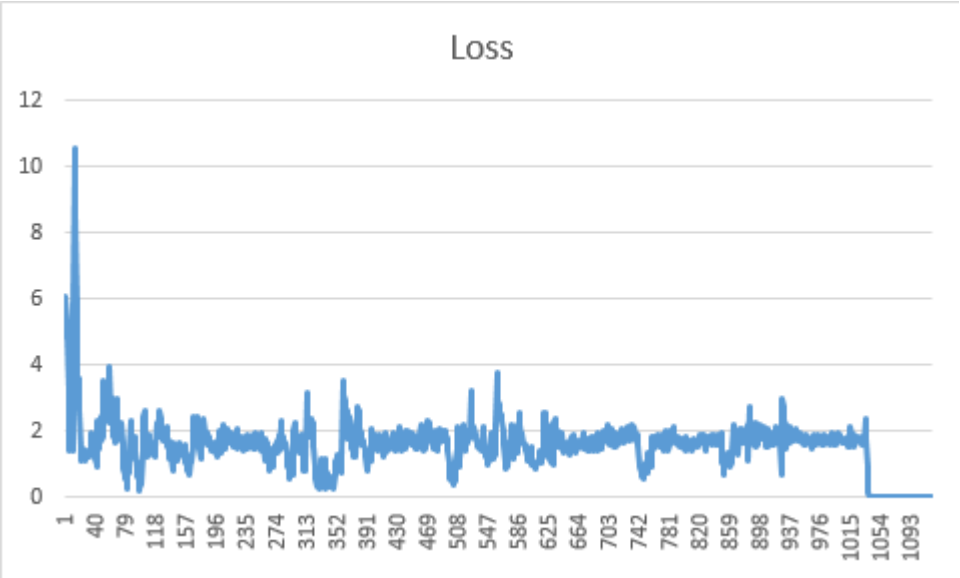

Figure 5.3: Loss function over batches(LSTM)

Chapter 6

Conclusion

6.1 Goals

All goals proposed in the introduction of this thesis were accomplished.

1. *Is it possible to predict a classification based on past variations of classifications across different classifiers?*

   All four machine learning methods C4.5, Random Forest, Multi-Layer Perceptron and Long short term memory, were able to predict a classification with great accuracy: 95%, 94%, 91%, 98% respectively. Furthermore, it shows that algorithms (C4.5, Random Forest and MLP) not ready to analyse time series data, are capable of that as long as data is modified for that effect. Even though, an architecture specified for this task is still better.

2. *Is it possible to improve certain algorithms by searching for optimal parameters?*

   Yes. Through genetic algorithms we were able to find the optimal structure and parameters for both neural networks, MLP and LSTM.

3. *What algorithm can best predict the classification?*

   The best algorithm was LSTM with an accuracy of 98% and loss of 9%.

6.2 Future Work

Although the results are good and proved it is possible to predict a classification based on past classifications by multiple classifiers, further research is needed. The classification variations were low and the time frame might be too low as well.

Another interesting possibility is to include some technical details of the files (for example function calls sequences) into the input data, because this dissertation was mostly over statistical data of the multiple classifiers.

Conclusion

References

[AAB+15] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan C. Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[BB96] Leo Breiman and Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[BC16] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[Bel14] Jason Bell. *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley & Sons, 1 edition, 11 2014.

[BFOS84] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

[BM01] Indranil Bose and Radha K. Mahapatra. Business data mining - a machine learning perspective. *Information & Management*, 39(3):211–225, 2001.

[Bot12] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

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

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[Cau89] Maureen Caudill. Neural networks primer, part i. *AI Expert*, 1989.

[Cha04] Chris Chatfield. *The analysis of time series: an introduction*. CRC Press, Florida, US, 6th edition, 2004.

[Cis16] Cisco. Cisco visual networking index: Forecast and methodology, 2015–2020. Available at http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html, June 2016.

REFERENCES

[GH88] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[GS00] Felix A. Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *IJCNN (3)*, pages 189–194, 2000.

[GSC00] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000.

[GSK+15] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[Ho95] Tin Kam Ho. Random decision forests. In *Third International Conference on Document Analysis and Recognition, ICDAR 1995, August 14 - 15, 1995, Montreal, Canada. Volume I*, pages 278–282, 1995.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[Hum15] Fay Humphries. Cyber security awareness 'severely neglected' | itweb. http://www.itweb.co.za/index.php?option=com_content&view=article&id=142733, visited on 05/02/2016, 2015.

[IDC16] IDC. Idc releases 2014 predictions for chief marketing officers. Available at http://web.archive.org/web/20140701052425/http://www.idc.com/getdoc.jsp?containerId=prUS24541813, June 2016.

[Jon90] Gareth Jones. Genetic and evolutionary algorithms, 1990.

[Kar16a] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. Available at http://cs231n.github.io/neural-networks-1/, June 2016.

[Kar16b] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. Available at http://cs231n.github.io/neural-networks-3/#anneal, June 2016.

[KM14] Erfan Khaji and Amin Satlikh Mohammadi. A heuristic method to generate better initial population for evolutionary methods. *CoRR*, abs/1406.4518, 2014.

[Kot13] S. B. Kotsiantis. Decision trees: a recent overview. *Artificial Intelligence Review*, 39(4):261–283, 2013.

[Lip15] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[LL11] Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *CoRR*, abs/1109.3627, 2011.

[LS95] Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Commun. ACM*, 38(11):54–64, November 1995.

REFERENCES

[Ltd12] IsecT Ltd. Iso/iec 27032 cybersecurity guideline. http://www.iso27001security.com/html/27032.html, visited on 12/01/2016, 2012.

[Mit96] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[Mit06] Tom Mitchell. The discipline of machine learning. Technical Report CMU ML-06 108, 2006.

[MO11] Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011.

[NBJ02] George E. Nasr, E. A. Badr, and C. Joun. Cross entropy error function in neural networks: Forecasting gasoline demand. In Susan M. Haller and Gene Simmons, editors, *FLAIRS Conference*, pages 381–384. AAAI Press, 2002.

[Nie16] Michael Nielsen. Neural networks and deep learning. Available at http://neuralnetworksanddeeplearning.com/chap1.html, June 2016.

[PKSR02] Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. Decision trees: An overview and their use in medicine. *Journal of Medical Systems*, 26(5):445–463, 2002.

[Qui86] J. R. Quinlan. Induction of decision trees. *MACH. LEARN*, 1:81–106, 1986.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[Sal94] Steven L. Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240, 1994.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SNY15] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. APAC: augmented pattern classification with neural networks. *CoRR*, abs/1505.03229, 2015.

[Wer90] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.