Bogazici University

Department of Electrical and Electronics Engineering

EE 492
Senior Design Project Final Report

Malicious Software Detection Using Machine Learning Algorithms

Furkan Karakaş
With Student Number: 2014401198

Principal Investigator:
Prof. Dr. Emin Anarım

A report submitted for EE492 senior design project class in partial fulfillment of the requirements for the degree of Bachelor of Science (Department of Electrical and Electronics Engineering) in Boğaziçi University

May 31st, 2019

Abstract

The time has changed, and it is now more important than ever to be able to make sense of data with enormous size, a.k.a. "big data analysis". It does not actually matter what your profession is. Knowing how to develop a machine learning model in its simplest form is an essential skill of the 21st century. It is even a greater necessity in the field of information security - cyber crimes are today a reality and hackers want to influence as many devices as possible. The range of devices varies today, i.e. it does not only include PCs. The Internet of Things (IoT) has been advancing very rapidly lately, so it is also of utmost importance to guarantee the safety of the smart devices in our home. For example, we need to be in a position to detect a malware if a hacker infects our device with his own malicious code.

This was the motivation for me to have a bachelor project about this topic. My goal was to learn the ways how machine learning algorithms work, what kind of methods there are, and how you can apply them to the field of information security. From a relative perspective, I am able to tell that I have been successful in these points. I have learned how to use the most modern machine learning tools, i.e. the programming language Python, and most importantly, its libraries. Python has very powerful libraries for machine learning and data manipulation, and throughout the semester, I have been trying to learn them, and properly use them for my goal.

Acknowledgements

I would like to deliver special thanks to my principal inspector *Prof. Dr. Emin Anarım* for letting me have a bachelor project in this field. I was already interested in information security, and I can safely say that I learned a lot of things in this work. Furthermore, I would like to thank *Erhan Davarcı* for his guidance. He is a doctoral student of Prof. Anarım. I could consult him about the things that I was not sure about, and he shared his valuable knowledge with me.

Most importantly, I would like to thank the awesome data science community at `https://www.kaggle.com/`. It is a place to do data science projects, and people usually share their codes with the community in an open-source manner so that everybody can learn something, and they can also contribute to the original work to help it getting improved. It has also free educational information about the programming language Python and machine learning, which really helped me a lot!

Contents

List of Figures

1 Introduction

In this senior design project, I have developed several machine learning algorithms which decide whether a given Windows machine is infected by malicious software or not. The data for training the machine learning model and testing it are provided by Microsoft. As a matter of fact, the topic of my project is an open source, online competition [1] which is organized by Microsoft. Machine learning, as a subset of the more general term "Artificial Intelligence" is one of the hottest learning areas at the moment. In my opinion, it is one of the essential skills that every engineer must study, regardless of the study path that one takes. After all, the current world that we live in is built on big data, and machine learning is all about making sense of enormous size of data.

There is the famous Moore's law which states that "the number of transistors in a dense integrated circuit doubles about every two years". There is also a similar relationship in terms of the size of the data which are being processed everyday. Figure 1 shows the code which is used for the aircraft in the "Apollo Program" and its writer Margaret Hamilton. The program took place in 1969 and back in these days, the programming languages were very low-level, i.e. you had to work directly on the hardware level. They are harder to program and usually not user friendly. You have to write more lines of code in order to achieve the same desired result. However, even though one has to write such long lines of code, the data that he or she is processing must be 4 MB maximum. As the technology advanced, new high-level languages such as C, C++ and in the nearer past, a much higher-level computer language Python appeared, which I have also used in my project. This way, programmers did not have to worry too much about the hardware and they could simply abstract it in order to achieve a more specific programming goal. In Figure 2, we see Katie Bouman and the hard drives that she used in order to store the information, which is necessary to construct the image of the most recent discovery of the first-observed black hole. These hard drives contain data of some *petabytes*, i.e., we are talking about the order of 10^{15} bytes! These two pictures illustrate the evolution of the data and the programming languages, more specifically, "information technology", throughout the years.

Although it is crucial to be able to process the huge data in this information age, being able to protect them from theft and modification is as important as that. This general concept is called as "Information Security", and this concept is more important than ever in today's world. Governments, as well as equally-powerful international companies are spending millions of dollars in this field, but there is still no guarantee that their information is kept secret from a potential hacker. There is no perfectly secure system, only *more secure* systems. There are already information security expert outages in the industry - it is really hard to find a qualified person in this particular area, and this shortage is expected not to be filled anytime in the near future.

My motivation for starting this term project was to combine these two areas, information security and machine learning. At first glance, they do not seem to be related to each other. However, machine learning algorithms can be used to response faster. What we achieve by machine learning is that we learn by experience - if we encounter a similar instance in the future, then we will be able to make a call much faster and it will free up the resources that an organization uses. Our resources are scarce, and the organizations that are able to use them as efficiently as possible will keep being part in the game.

Figure 1: The computer programmer Margaret Hamilton stands next to the printed version of the code, which she wrote for the "Apollo Program".[2]

Figure 2: Katie Bouman and the data that she used and which are stored inside many hard disks for the purpose of construction of the first ever picture of a black hole.[3]

2 Machine Learning Terminology

In this section, I will explain what machine learning is, what kind of steps it consists of, and important terminology for it. I would like to point out to the fact that the machine learning, as a science, has too much mathematical depth in it. It uses fundamental concepts from the sub-domains of mathematics such as probability theory, statistics, and a lot of linear algebra. In this report, however, I will try not to dive into too much of mathematical details. Instead, I will focus more on the practical aspect of the machine learning, where you deal with processing the initial data, picking a good model depending on the problem that you are facing, and evaluating how well the model that you have developed performed on predicting future values.

2.1 What is machine learning?

"Machine Learning (ML) is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions." [4]

According to the above definition, ML takes inspiration from the natural way of learning for humans - we learn by observation and experience. For each additional datum in our data set, we try to see its similarities with the previous data. If this new additional datum is somewhat comparable to the previous data, then it should have a similar value with these previous data. Hence, we make a prediction about the value of its target variable in favour of these similar previous data. Note that in the general sense of machine learning, a target variable need not exist. As I will be mentioning different types of machine learning, there are actually no target variables in the class of unsupervised learning. Nevertheless, for getting a good intuition, thinking of the ML as predicting the values of the target variables is a good starting point.

2.2 Types of ML

Most generally, we can divide ML algorithms into two groups: supervised and unsupervised learning. The majority of ML algorithms use supervised learning, in which we have input variables $\mathbf{X}$, also known as *features*, and we have an output variable, y, a.k.a. *the target variable*. It is worth mentioning that there may be more than one target variable, but we will not consider that case here since it adds a substantial amount of sophistication to our model. We use an algorithm, the ML model, which maps these features to the target variable:

$$f(\mathbf{X}) = y. \tag{1}$$

In the supervised learning, the goal is to basically represent the mapping f as accurately as possible so that given a new datum with features $\boldsymbol{X_{new}}$, we can predict the target outcome y_{new} as accurately as possible. As the name suggests, we *supervise* the ML algorithm like a teacher by first feeding a training data set into the algorithm so that it can observe the properties of the mapping f for each pair of features $\mathbf{X}$ and target variable y. After the model gets enough samples, it has sufficient experience to predict the target variable of a new input. In my project, I used supervised algorithms since I need to decide whether a given Windows machine with specific features is infected by malware or not.

In the unsupervised learning, unlike in the supervised one, we do not have a target variable y. The goal of the unsupervised learning is to explore the data at hand by observing the distribution of objects in the data, how the data are organized, etc. It is called *unsupervised*, because there is no teacher, nor is there a correct answer for the problem. Think of it as a

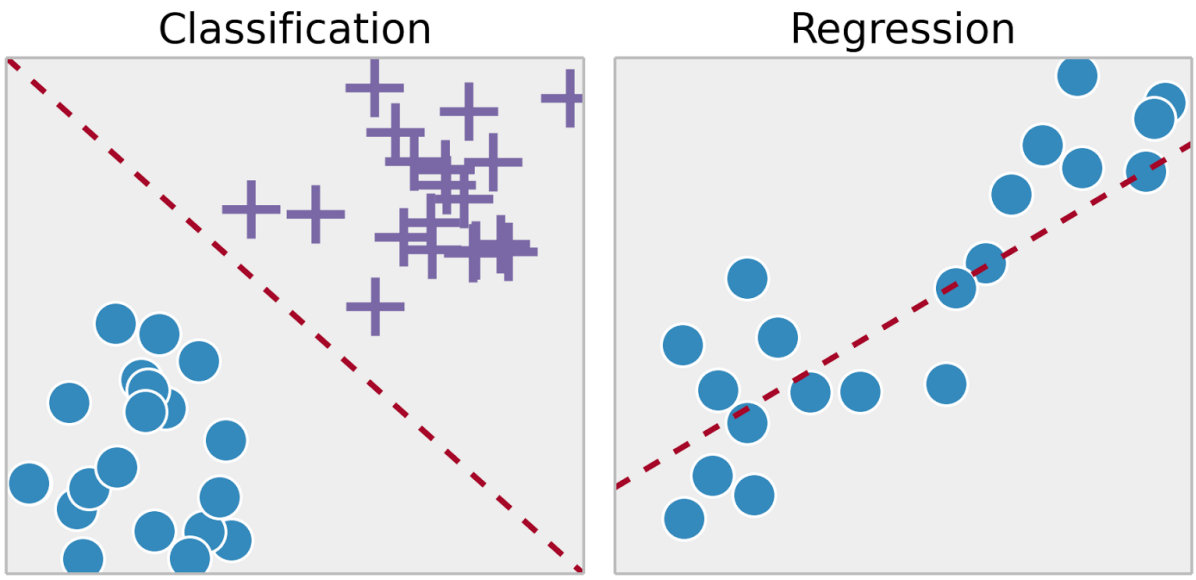


Figure 3: A simple example for the classification and regression tasks in a machine learning algorithm.[5]

kind of discovery - the data set is our world, and we are set to discover as much as possible in this world.

2.3 Classification vs Regression

Regression and classification are two different types of supervised learning methods, that is, in both methods, we have a target variable to predict. This is where these two methods differ from each other - if the target variable is continuous, then the method that we employ is a regression algorithm. On the other hand, if the target variable is a categorical variable, that is, if there is only a finite number of states that the target variable can take, then the method that we employ is a classification algorithm.

Figure 3 shows a simple example for classification and regression tasks in ML. In a classification problem (in this case, *a binary classification problem*), we want to somehow find a curve, not necessarily straight line, which partitions the space. Based on the location of a point in this space, we make a prediction that this particular point belongs to a particular class. On the other hand, in a regression task, we would like to fit the n-dimensional data onto a curve, called *the best fitting curve*, which is again not necessarily a straight line. Based on the features that a data point has, we make a prediction about the value of its target variable based on the curve.

In this project, my task is to predict whether a given Windows machine is infected by a malware or not. Hence, my task is a binary classification problem.

2.4 Machine Learning Algorithms for Classification Tasks

There are several useful ML algorithms for classification problems. I would like to mention some of them in this section.

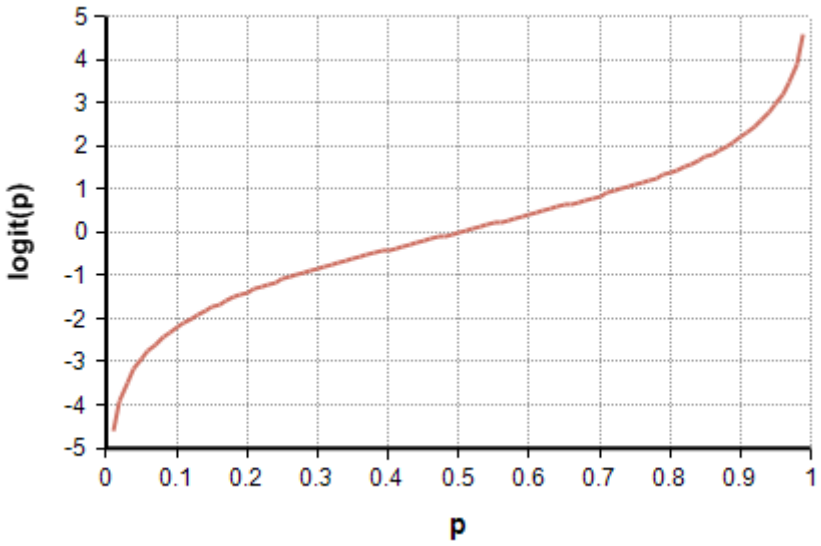


Figure 4: The graph of logit(p)

2.4.1 Logistic Regression

Logistic regression is the appropriate method to use if the target variable is binary. It is used to explain the relationship between the features and the dependent variable.

The data set has to satisfy some properties if we want to employ the logistic regression:

- The target variable should be dichotomous (binary).
- There should not be any outliers in the data set, that is, data points which have extreme values and deviate significantly from the mean value.
- There should not be high correlations between the features.

I will address the high correlation between the feature columns in the data-cleaning stage later.

At the center of the logistic regression analysis lies the log-odds. Mathematically, it is defined as follows:

$$logit(y) = \ln \frac{\Pr(y=1)}{\Pr(y=0)} = \ln \frac{P}{1-P} = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k \tag{2}$$

In our model, we basically have to estimate the values of β_ks, and if the new data point lies above this line, we predict it as 1, and 0 otherwise. Figure 4 shows the graph of $logit(P)$ in the interval [0,1].

2.4.2 K-Nearest Neighbors (KNN)

KNN is one of the most widely used classification algorithm. It is *non-parametric*, meaning that the algorithm does not make any assumptions about the underlying data set. So, the algorithm has the ability to adapt itself to any kind of data distribution. This is where the algorithm's strength comes from.

Figure 5 illustrates the basic working principle of the K-nearest neighbors algorithm. As the value of k gets higher, we have a better chance of predicting the value of the data point. In this example, if we choose $k = 1$, then according to the algorithm, the new data point is classified as being in Class 1. On the other hand, if we choose $k = 3$, then it will be classified

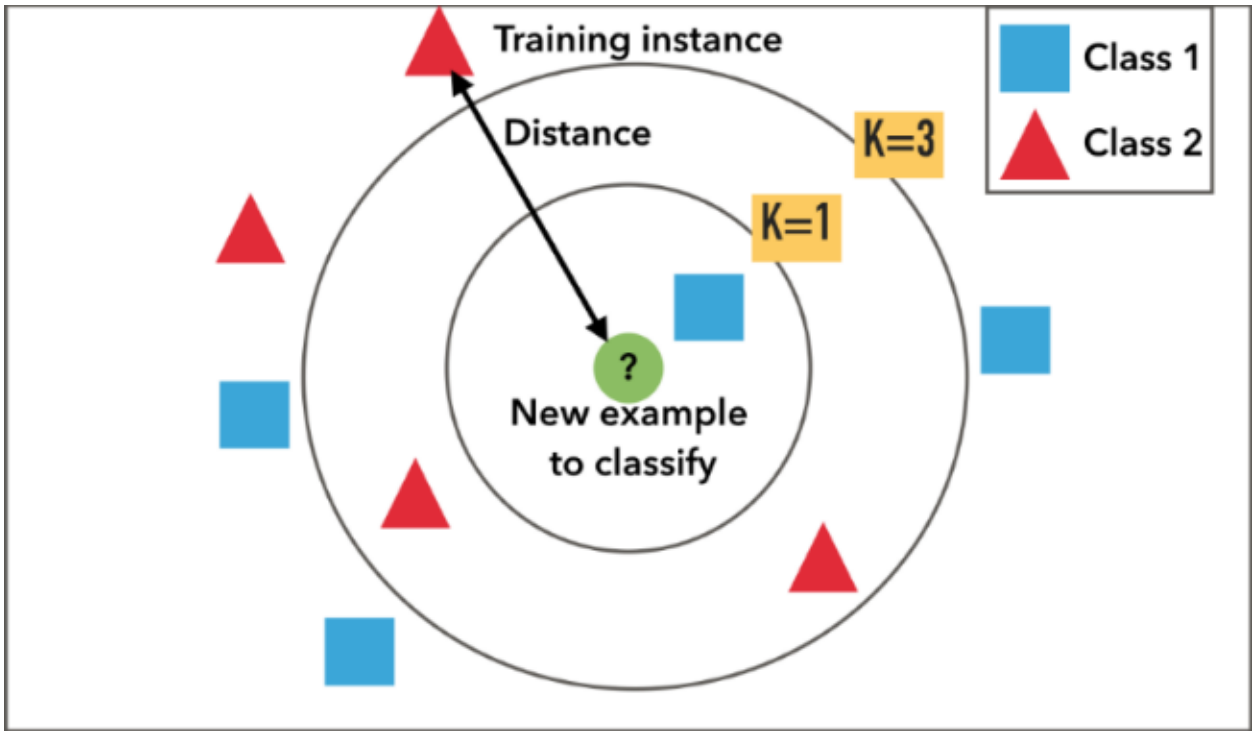


Figure 5: KNN algorithm for $k = 1$ and $k = 3$. For an unknown data point, if $k = 1$, then the data point is classified the same as its nearest neighbor. On the other hand, if $k = 3$, then the data point is classified according to its other two neighbors. [6]

as being in Class 2 since the number of elements belonging to the Class 2 is greater than the number of elements belonging to the Class 1 in the circle.

As you might have guessed, we need to calculate the distance of the new data point to other data points in our data set in order to arrive at the decision for classification, and this process can be quite costly for large data sets, which consist of several millions of rows and around 50 distinct features, which is the case for the data set in this project.

2.4.3 Support Vector Machines

Support vector machine (SVM) is another classification method for supervised learning. In this one, we want to construct a hyper plane in the n-dimensional space, where n is the number of features in the data set. Once the hyper plane is constructed, it divides the n-dimensional space into distinct regions, where each region corresponds to each specific class. In the case of a binary classification problem, the hyper plane partitions the n-dimensional space into two disjoint regions. For each additional data point in the testing set, we find its location in the n-dimensional space, and if it belongs to the region 1, then we classify the new data point as a member of class 1. On the other hand, if it belongs to the region 2, then we classify it as a member of class 2. Figure 6 illustrates the basic working principle of the support vector machine algorithm. It is worth noting that the hyper plane need not be linear. It depends on the problem, what kind of a hyper plane you should use. Usually, one turns the initial problem into another problem by applying a suitable transformation to the data, and in this new transformed space, you can construct a linear hyper plane. Now, you just have to apply the inverse transformation on that hyper plane, and you will get another surface in the original space, possibly a non-linear one. Such transformations are called *kernels*.

One very important characteristic of a good SVM classifier is that it tries to achieve a good *margin* on the data. The margin is the separation between the hyper plane and

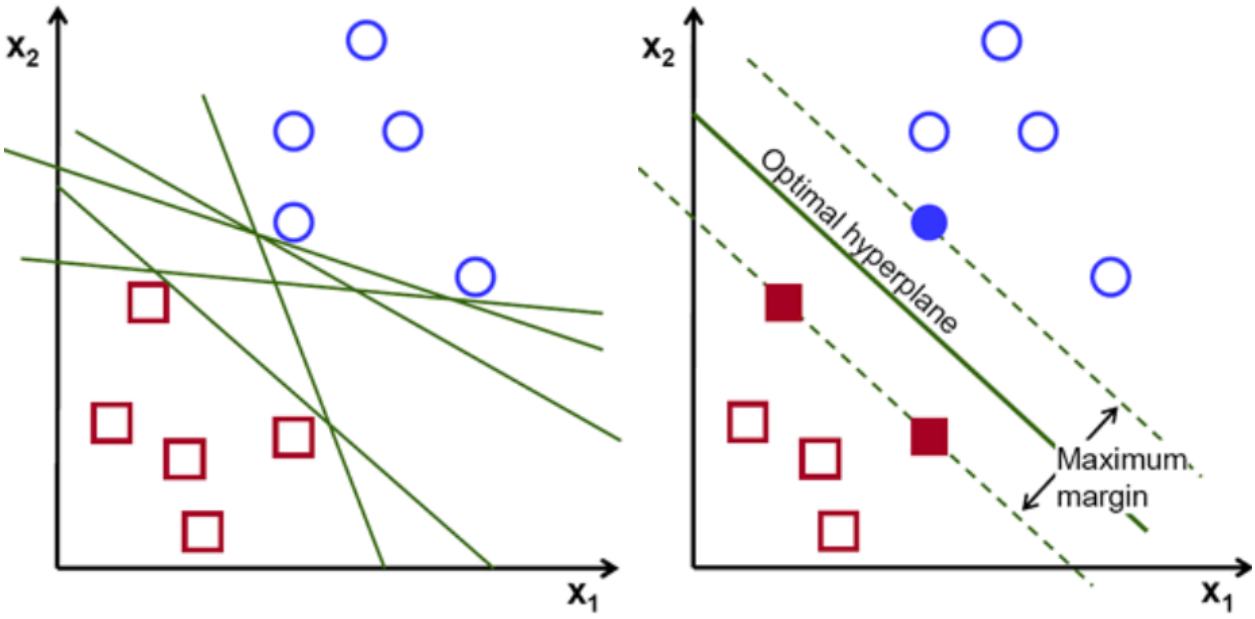


Figure 6: A basic example how the support vector machine algorithm works. Among the other candidates, we select the best hyper plane, which maximizes the margin between two different classes of data points. [7]

each of the data point in distinct classes. The hyper plane with a good margin tries to stay equidistant towards all distinct classes of data points, i.e., it should not have a more favorable position towards one class than the others.

2.4.4 Decision Trees

The decision trees are another useful machine learning algorithms, which are suitable for classification, as well as regression tasks. A decision tree is a flow-chart-like structure, which splits at each node, taking another path, according to a specific collection of the criteria about the features in a data point. It is one of the most commonly used method for solving machine learning problems, and it is also being used as the building block of more complex machine learning problems such as *Random Forests*, which we shall explore in the next subsection.

Figure 7 shows a basic flow chart for a decision tree, which handles the classification problem: "As an investor, is it risky for me to invest in a particular commodity?". It does a splitting based on one or several criteria about the candidate's profile. We start at the *root node* by checking the savings of the candidate. If the candidate has low savings, then he or she goes to the leftmost node in the middle, and it is said that a *splitting* took place. Now, we are at a *child node* of the root node, and the root node is the *parent node* of the current node that we are in. At this node, we test the candidate's assets. If the assets that he or she possesses are low, then we arrive at the decision that the investment is risky, in a bad way. At this current node, we cannot go down any further. Such nodes are called *leaf* or *terminal* nodes.

How does the algorithm determine the criterion for splitting? One of the most common method for this task is called ID3, and it is based on the *information gain*. The information gain is based on the decrease in entropy after the data set is split on an attribute. Recall

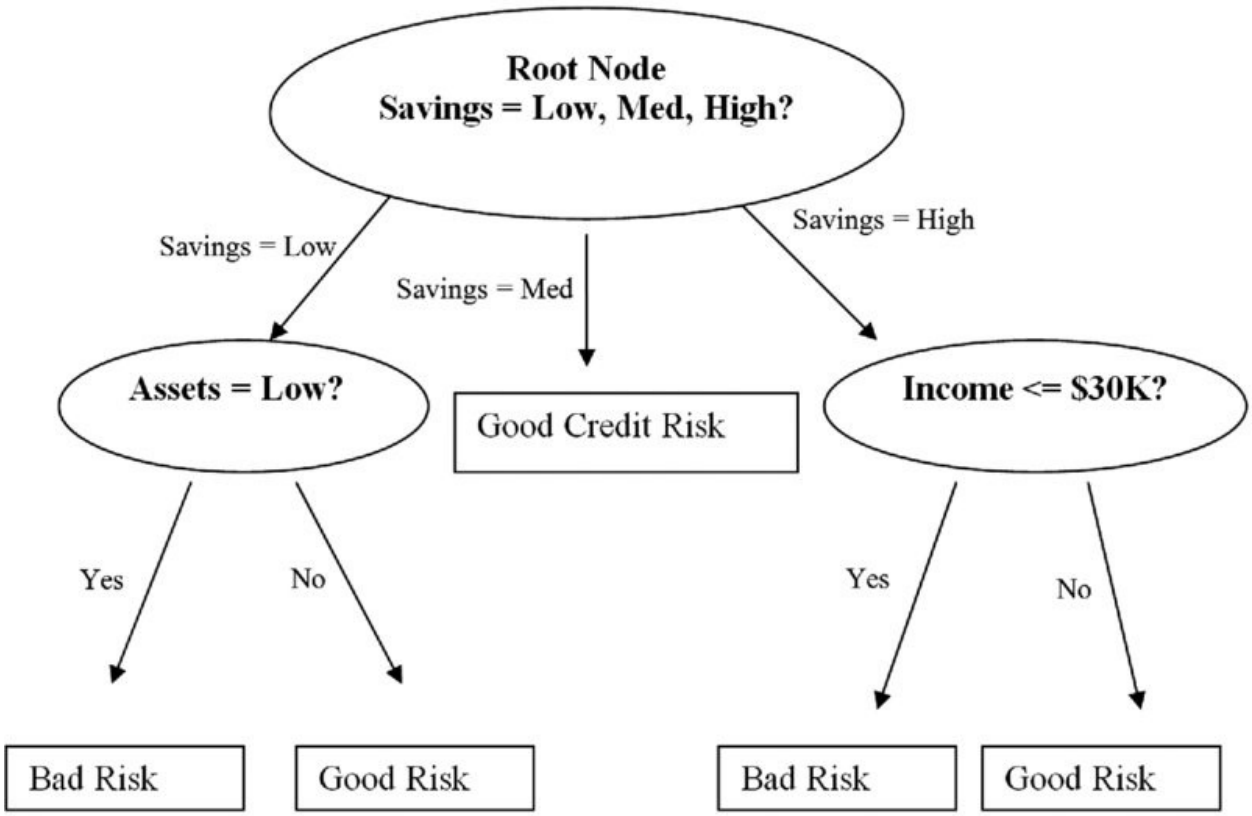


Figure 7: A simple example for the flow chart of a decision tree. [8]

that the entropy of a discrete random variable X with c distinct values is defined as:

$$E(X) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{3}$$

where p_i is the probability that the random variable X is equal to x_i. Given two random variables X and T, we define the mutual entropy as:

$$E(T, X) = \sum_{c \in X} P(c)E(c) \tag{4}$$

Taking T as the target variable, we calculate the value of $E(T, X)$ for each feature X.

The *information gain* is based on the decrease in entropy after a splitting takes place. Constructing the decision tree is all about finding the feature X, which yields the maximum information gain, which is defines as:

$$Gain(T, X) = E(X) - E(T, X) \tag{5}$$

So, we basically search for the attribute, which yields the highest information gain, and thence we split the decision tree on the current node according to that particular feature. After that, we do the same process recursively for the child nodes until we find a branch with zero entropy, which is basically a leaf node.

There are other methods in order to determine the criterion for splitting such as *Gini-Index, Chi-Square* and *reduction in variance*, which I will avoid explaining in this report.

2.4.5 Random Forests

The random forest is a supervised learning algorithm. It can be used for both regression and classification tasks. As the name suggests, it creates a forest and somehow it makes this

forest random. A forest consists of trees, meaning that the random forest algorithm uses many decision trees with different learning models. After using different learning models, the overall result increases and we get more accurate results when we try to test the model on the testing set. In Layman's terms: *A random forest builds multiple decision trees and merges them together to get a more stable and accurate prediction.* [9]

A Random Forest nearly has the same hyper parameters as a decision tree does. Instead of searching for the most important node while splitting at a node, it searches for the best feature among a random subset of features. By adding randomness to the model, it usually performs better than a regular decision tree does. There are some important hyper parameters for a Random Forest:

n_estimators: This parameter indicates the number of trees that the algorithm builds before taking the maximum voting or taking averages of predictions. A higher number of this hyper parameter increases the stability of the machine learning model, but at the cost of a slower computational speed.

max_features: This is the maximum number of features that the algorithm considers in order to split at a given node.

min_sample_leaf: This hyper parameter indicates the minimum number of samples, which need to be at a leaf node. This may have the effect of smoothing the model.

There are some other parameters, which I will avoid explaining them here. You can have access to all of the parameters at the documentation website: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

2.4.6 Light Gradient Boosting Machine (LightGBM)

In order to understand what LightGBM is, we need to clarify what *boosting* and *gradient boosting* are.

By boosting, we mean that we turn some weak learners into a single strong learner by combining them. Suppose that we have several weak learners, which can predict a target variable with an accuracy of slightly higher than 0.5, i.e. we have several learners that perform better than tossing a fair coin and taking the result of this experiment as the value of the target variable. What we do by boosting is that we look at the results of these several weak learners, and we cast a voting among them in order to determine the final result of our prediction. By doing this, we combined several weak learners and we constructed a strong learner from them. The boosting is this particular task in its essence.

Like other boosting methods, *gradient boosting* combines weak learners into a strong learner in an iterative fashion. However, they differ in how they handle in the shortcomings of the weak learners. The gradient boosting algorithm deals with these shortcomings by using the gradients in the loss function. The loss function is a measure indicating how good are model's coefficients are at fitting the underlying data. [10]

That being said, LightGBM is a gradient boosting algorithm. According to its original repository [11], it has the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.

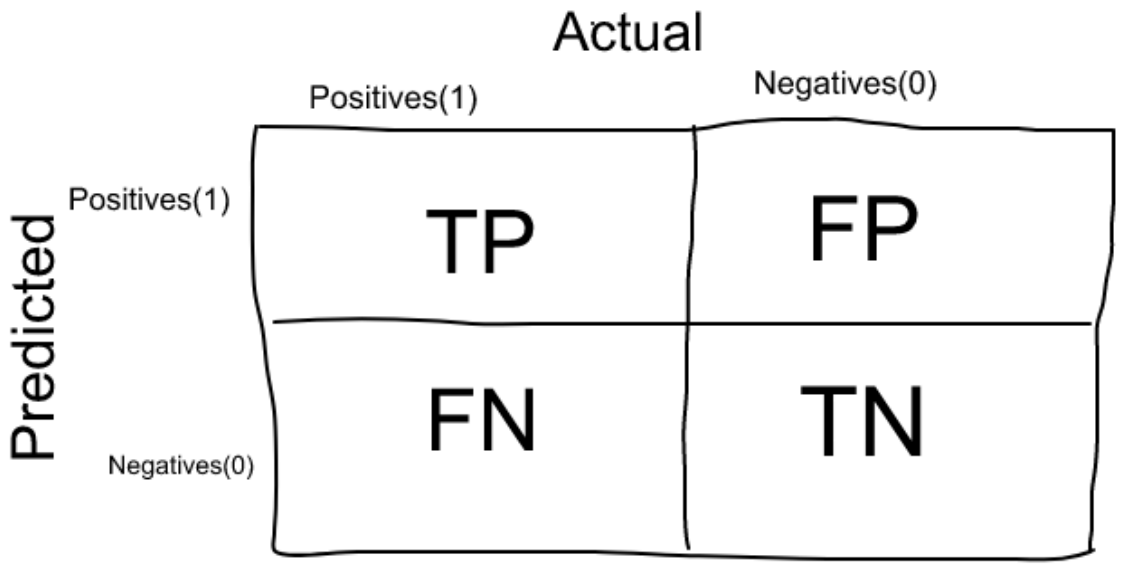


Figure 8: Elements in a confusion matrix: TP, TN, FP, and FN. [12]

- Support of parallel and GPU learning.
- Capable of handling large-scale data.

LightGBM is also widely used in many winning solutions of machine learning competitions, including the one that I am investigating in this term project.

2.5 Evaluating Machine Learning Models

Once we prepared an appropriate machine learning model, we have to test it on the testing data set that we have created before. There are several metrics to be considered while evaluating the performance of machine learning models for binary classification. The most important ones are:

- confusion matrix,
- accuracy,
- precision,
- recall or sensitivity,
- specificity,
- F1 score,
- area under the curve (AUC).

There are some important names associated with the binary classification problem before delving into the above-mentioned metrics. Refer to the Figure 8. We have actual vs. predicted values for the target variable. If it is actually positive and we predict it as positive correctly, then this instance is called *true positive (TP)*. If it is actually positive and we predict it as negative wrongly, then this instance is called *false negative (FN)*. If it is actually negative and we predict it as positive wrongly, then this instance is called *false positive*

(FP). If it is actually negative and we predict it as negative correctly, then this instance is called *true negative (TN)*. All of the above metrics are defined in terms of these quantities, so it is rather important to understand them well.

2.5.1 Confusion Matrix

A confusion matrix is shown in Figure 8. It consists of four entries called *true positives (TP)*, *false negatives (FN)*, *false positives (FP)*, and *true negatives (TN)*. It is one of the easiest and most widely used performance metrics among data scientists for classification problems. It is worth mentioning that the confusion matrix is not a stand-alone performance measure, but almost all of the performance metrics are based on the elements in the confusion matrix.

2.5.2 Accuracy

The name is self-explanatory: Accuracy is the number of correct predictions made, divided by all predictions. Mathematically, it can be shown as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{6}$$

Accuracy is a good measure if the target variable in our data set is balanced, meaning that the number of entries with positive and negative values should be close to each other. It is worth mentioning that you should never use accuracy as an evaluation metric if the target variable in the data set is the majority of one class, i.e., the data set is not balanced. For the sake of example, consider a machine learning algorithm, which predicts whether a given patient suffers from cancer or not. Given 100 people, say 5 of them really suffers from cancer. If we predict all of the patients as "not having cancer", then we get an accuracy of 95%, but does it mean that our model works as intended? Definitely not! In this particular example, accuracy is not a sound metric for performance evaluation of the model.

2.5.3 Precision

Precision is a metric for measuring the success rate for the positively predicted data. Mathematically, it can be defined as follows:

$$Precision = \frac{TP}{TP + FP}. \tag{7}$$

Like in the previous example, assume that the property of having cancer denotes "positive" (1), and having no cancer denotes "negative" (0). If we predict that everybody has cancer, then we have 5 true positives since indeed 5 people are really cancerous out of 100. Also, we get 95 false positives since we actually predicted 95 as cancerous although they were not. In this case, our model has a precision of 5%.

2.5.4 Recall or Sensitivity

Recall is a measure how well our model can distinguish between true positives and false negatives. Mathematically, it is defined as:

$$Recall = \frac{TP}{TP + FN}. \tag{8}$$

Like in the previous example, if we predict that everybody has cancer, then we have 5 true positives, and for the false negatives, we have none. Hence, we get a recall of 100%.

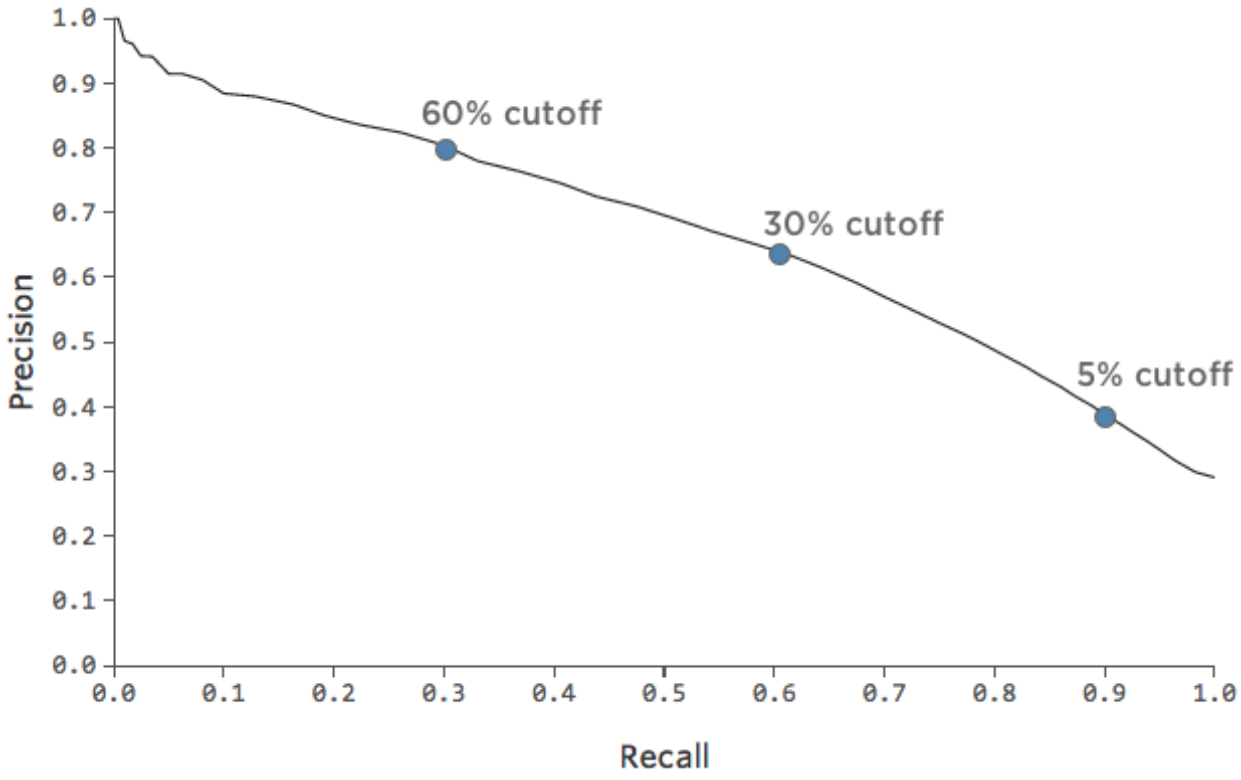


Figure 9: Precision vs. Recall curve for different thresholding values. [13]

Note that in this prediction model, we maximized the value of recall, but we somehow neglected the value of precision. Usually, this is the trade-off that a machine learning model faces, which is known as *precision-recall trade-off*. Figure 9 shows this trade-off for different thresholds. In the example above, our threshold value if 0%, which means that we predict every patient as having cancer.

A good data scientist must find a good compromise between precision and recall. Depending on the problem that is being investigated, he or she can put more emphasis on one and neglect the other. For example, in the case of diagnosing a patient as having cancer, it is of utmost importance that a patient that is actually cancerous must not go undiagnosed. So, he or she must make sure that the number of false negatives (FN) should be as small as possible, meaning that the value of recall should be as high as possible, even if it means a small value of precision, and hence more incorrectly-as-cancerous-diagnosed patients.

2.5.5 Specificity

Specificity is the exact opposite of recall. Mathematically, it is defined as:

$$Specificity = \frac{TN}{TN + FP}. \tag{9}$$

For the same example above, we have zero true negatives since we predicted everyone as having cancer. So, we have a specificity of 0%.

2.5.6 F1 Score

We mentioned above that precision and recall have some kind of a trade-off in a machine learning model. So, investigating solely a single one of them is not a good idea while

evaluating the machine learning model. Instead, you might want to combine them together to produce a single performance metric that represents both of them.

A naive approach might suggest that we should take the arithmetic average of these two quantities. However, this is not a good idea. For example, consider the example that we gave above. We had a recall of 100% and a precision of 5%. Note that we classify every patient as cancerous, so it is clear that our model is terrible, and a simple arithmetic mean is no good for such a model. Instead, we use the *harmonic mean*. The F1 score is defined as:

$$F1Score = HarmonicMean(Precision, Recall) = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{10}$$

The harmonic mean of 100% and 5% is around 9.5%, and for a model as bad as this one, it is a much more suitable F1 score than the case with the arithmetic mean.

2.5.7 Area Under Curve (AUC)

Area Under the Curve (AUC) is another very useful metric, specially designed for binary classification problems. However, it can be extended to any classification problems with arbitrary number of outcomes.

We are talking about the area under some curve. What is this curve?

This curve is called *Receiver Operating Characteristic (ROC)*. Figure 10 shows an example of such a curve. The dashed line in the middle shows the worst-case scenario for a model, which is no different than tossing a fair coin and taking its result as the value of the target variable, and that curve has an area of 0.5 unit squared under its graph, which is the lowest value for AUC. On the other hand, our model could have been perfect and it could be a "square", meaning that it could have an area of 1 unit squared under its graph, which is the highest value for AUC. Hence, the value of AUC is bounded by these two numbers:

$$0.5 \leq AUC \leq 1.0 \tag{11}$$

For a good model, we want AUC to be as close to 1.0 as possible. The basic working principle is shown in Figure 11. Setting the threshold value to 85%, we see that TPR increases since $TPR = \frac{TP}{TP+FN}$, and at the current stage of the threshold we predict a portion of the positive quantities correctly. The value of the TPR will keep increasing until no more positive quantities remain left of the threshold, which corresponds to the threshold value around 40%. Similarly, the value of FPR will keep increasing until no more negative quantities remain left of the threshold, which corresponds to the threshold value around 10%.

By setting the threshold value to different values iteratively, say, threshold running from 0 to 1 with a step size of 0.001, we construct our ROC curve, and the quantity AUC is nothing more than the area under this particular curve.

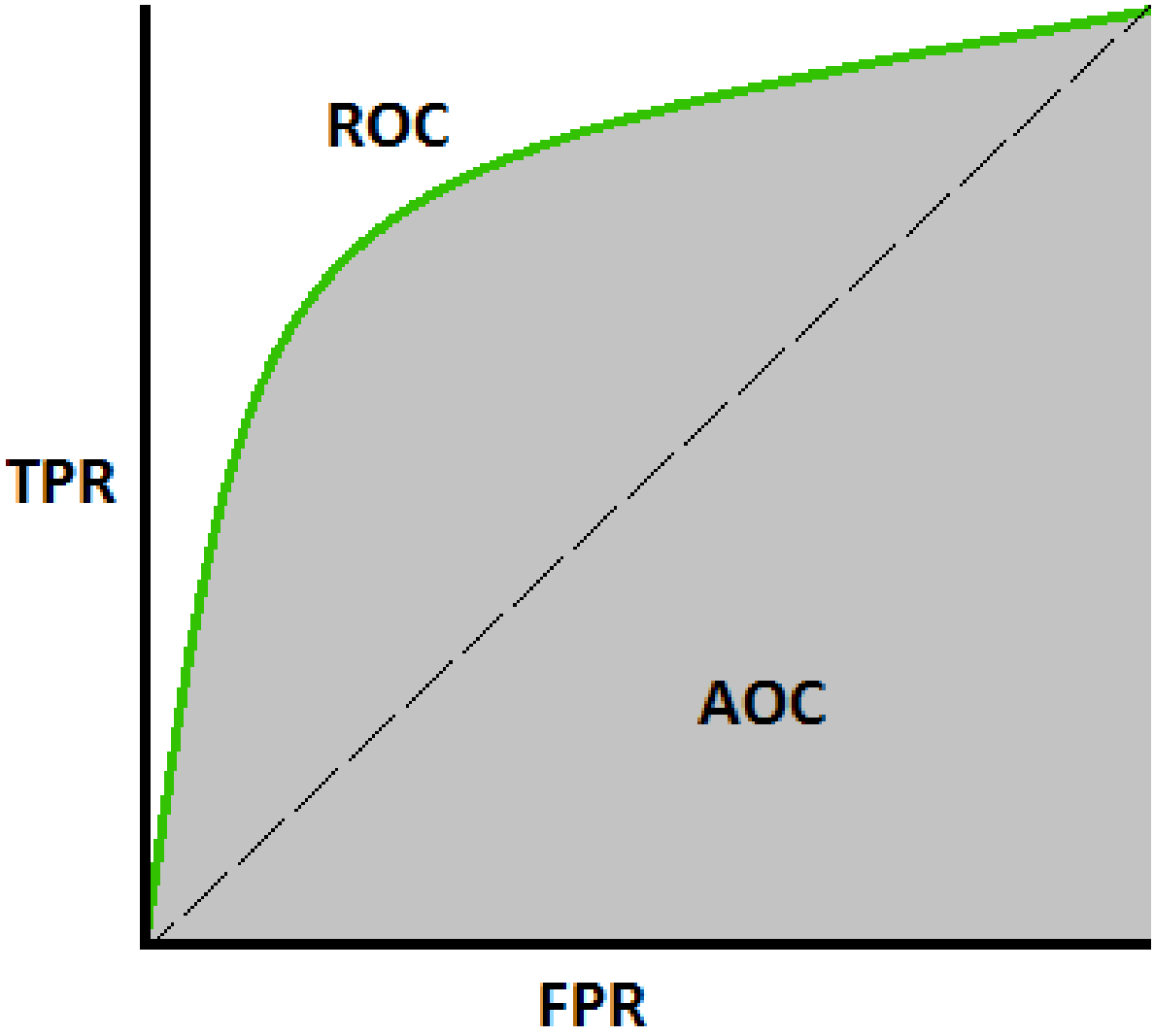
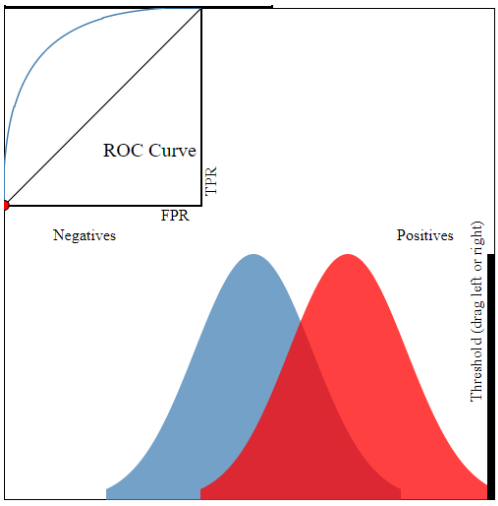
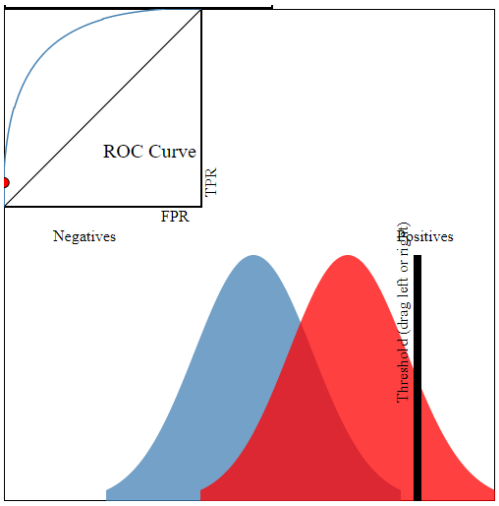


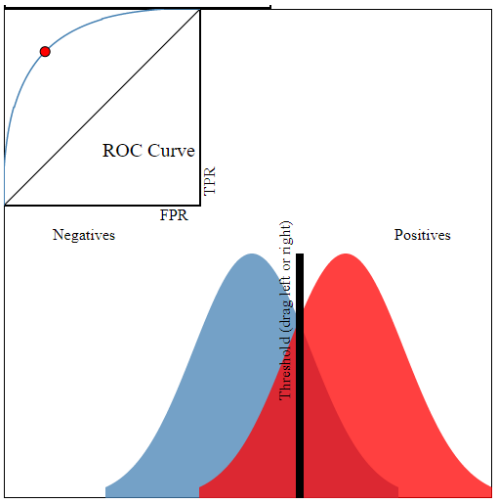
Figure 10: An example of a ROC curve. The x-axis represents the "False Positive Rate (FPR)", which is the same as 1-Specificity. The y-axis represents the "True Positive Rate (TPR)", which is the same as Recall or Specificity. [14]



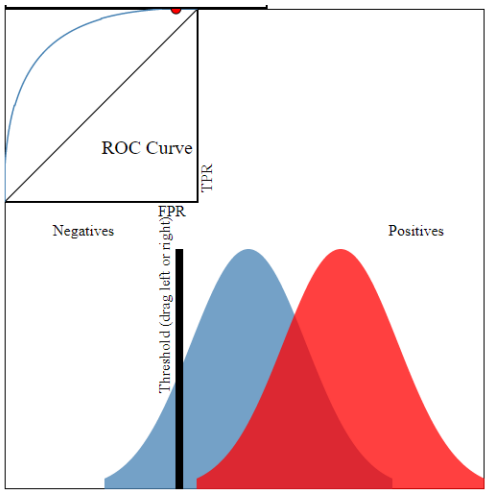
(a) Threshold set to 100%.



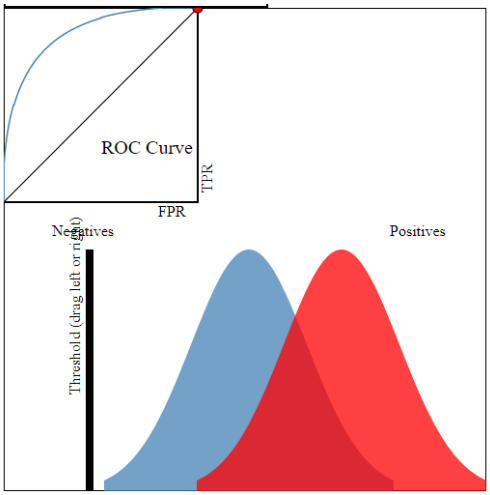
(b) Threshold set to 85%.



(c) Threshold set to 60%.



(d) Threshold set to 40%.



(e) Threshold set to 10%.

Figure 11: ROC curve and points on that curve for different values of thresholds for a binary data, in the shape of the Gaussian distribution. Notice that the threshold 100% corresponds to the point (0,0) on the ROC curve whereas the threshold 0% (and possibly more) corresponds to the point (1,1) on the ROC curve. [15]

3 Work That I Have Done

After talking about some terminology about machine learning, without delving too much into the mathematics behind it, now it is time for me to explain the work that I have completed in the project.

My project mainly consists of two parts. First part is the data preprocessing part, in which I cleaned the data set that is given to us, the online competitors, by Microsoft. In the second part, I applied the machine learning models that I have mentioned in the section above. Some have been successful, and some did not produce desirable results.

Concerning the machine learning tasks, I used the programming language "Python", which is the *de facto* standard for machine learning today, thanks to its extensive libraries for data processing, data visualization, and machine learning.

You can have access to all of my work in my GitHub repository, which I have created for the term project specifically. You can find the Jupyter notebooks with the file extension ".ipynb" in the URL `https://github.com/CoffeeGuy95/Malware-Prediction`. These notebooks are very powerful tools that help you to distribute Python code, as well as its output in a compact way.

3.1 Data Preprocessing

The data set at our disposal is pretty dirty, so we need to have some steps before we can use it for building machine learning models.

We start the task by defining the feature types in our data set. We will be using pandas library for reading data frames, and using them in the working environment.

The Missing Data We cannot work with the missing data, denoted as NA, in our machine learning models. So, we need to get rid of them somehow. There are different ways to deal with them, but I simply wanted to remove them in this task. To illustrate the percentage of the missing values in each column, we use the following code snippet:

```
# checking missing data
total = initial_data.isnull().sum().sort_values(ascending = False)
percent =
    (initial_data.isnull().sum()/initial_data.isnull().count()*100).sort_values(ascending
    = False)
missing_initial_data = pd.concat([total, percent], axis=1, keys=['Total',
    'Percent'])
missing_initial_data.head(50)
```

The result of this code snippet is given below:

```
                                        Total   Percent
PuaMode                                 8919174 99.974119
Census_ProcessorClass                   8884852 99.589407
DefaultBrowsersIdentifier               8488045 95.141637
Census_IsFlightingInternal              7408759 83.044030
Census_InternalBatteryType              6338429 71.046809
Census_ThresholdOptIn                   5667325 63.524472
Census_IsWIMBootEnabled                 5659703 63.439038
SmartScreen                             3177011 35.610795
OrganizationIdentifier                  2751518 30.841487
SMode                                   537759 6.027686
```

```
CityIdentifier                                    325409  3.647477
Wdft_IsGamer                                      303451  3.401352
Wdft_RegionIdentifier                              303451 3.401352
Census_InternalBatteryNumberOfCharges             268755  3.012448
Census_FirmwareManufacturerIdentifier             183257  2.054109
Census_IsFlightsDisabled                          160523  1.799286
Census_FirmwareVersionIdentifier                  160133  1.794915
Census_OEMModelIdentifier                         102233  1.145919
Census_OEMNameIdentifier                           95478 1.070203
Firewall                                          91350   1.023933
Census_TotalPhysicalRAM                            80533  0.902686
Census_IsAlwaysOnAlwaysConnectedCapable             71343 0.799676
Census_OSInstallLanguageIdentifier                 60084 0.673475
IeVerIdentifier                                     58894 0.660137
Census_PrimaryDiskTotalCapacity                    53016  0.594251
Census_SystemVolumeTotalCapacity                  53002   0.594094
Census_InternalPrimaryDiagonalDisplaySizeInInches 47134 0.528320
Census_InternalPrimaryDisplayResolutionHorizontal 46986 0.526661
Census_InternalPrimaryDisplayResolutionVertical 46986 0.526661
Census_ProcessorModelIdentifier                     41343 0.463410
Census_ProcessorManufacturerIdentifier             41313 0.463073
Census_ProcessorCoreCount                          41306 0.462995
AVProductsEnabled                                 36221  0.405998
AVProductsInstalled                                 36221 0.405998
AVProductStatesIdentifier                          36221 0.405998
IsProtected                                         36044  0.404014
RtpStateBitfield                                  32318  0.362249
Census_IsVirtualDevice                              15953  0.178816
Census_PrimaryDiskTypeName                         12844 0.143967
UacLuaenable                                       10838 0.121482
Census_ChassisTypeName                             623   0.006983
GeoNameIdentifier                                  213   0.002387
Census_PowerPlatformRoleName                       55     0.000616
OsBuildLab                                         21    0.000235
LocaleEnglishNameIdentifier                           0     0.000000
AvSigVersion                                        0    0.000000
OsPlatformSubRelease                               0     0.000000
Processor                                          0     0.000000
OsVer                                              0     0.000000
AppVersion                                          0    0.000000
```

We see that there are columns, which lack data in more than 90% rows.

I decided to drop all of the columns, which lack more than 6% of data. On top of that, Microsoft specifically mentioned that some columns are irrelevant for the malware detection task. One more important thing to mention is that the column `MachineIdentifier` is irrelevant for the purpose of machine learning. So, combining all of them, we drop them from the original data frame:

```
#Dropping irrelevant columns
initial_data.drop(['PuaMode', 'Census_ProcessorClass',
    'DefaultBrowsersIdentifier', 'Census_IsFlightingInternal',
                'Census_InternalBatteryType', 'Census_ThresholdOptIn',
                   'Census_IsWIMBootEnabled', 'SmartScreen',
```

```
                'OrganizationIdentifier', 'SMode', #Getting rid of too many NaN
                    columns
                'RtpStateBitfield', 'IsSxsPassiveMode', 'AVProductsInstalled',
                    'AVProductsEnabled',
                'IeVerIdentifier', 'Census_OEMNameIdentifier',
                    'Census_OEMModelIdentifier',
                'Census_ProcessorManufacturerIdentifier',
                    'Census_ProcessorModelIdentifier',
                'Census_InternalBatteryNumberOfCharges',
                    'Census_OSInstallLanguageIdentifier',
                'Census_OSUILocaleIdentifier',
                'Census_FirmwareManufacturerIdentifier',
                    'Census_FirmwareVersionIdentifier',
                'Wdft_RegionIdentifier'], #Getting rid of irrelevant columns
               axis=1, inplace=True)
#The machine identifier column is irrelevant for the purpose of ML
initial_data.drop(['MachineIdentifier'], axis=1, inplace=True)
```

Now, we got rid of NA values from most of the columns. We do the same for rows:

```
initial_data[initial_data.isnull().any(axis=1)] #axis 0: index, axis 1: columns
```

We see that there are still 921,983 rows that contain NA values. We drop them:

```
#Dropping rows which contain NaN
initial_data.dropna(axis=0, how='any', inplace=True)
initial_data.reset_index(drop=True, inplace=True) #Resetting the index after
    dropping NaN rows
initial_data
```

Now, our data frame is free of NA values.

Skewness Skewness means that in a data frame, columns consist of entirely a single element. For example, in a data frame, if column A consists of the value 0 99% times and of the value 1 only 1% times, then this column A is not very useful to us for the machine learning task, and we can simply drop it.

We investigate the number of unique entries and skewness in the columns by the following code snippet:

```
#Detect features where almost all entries are a single element
pd.options.display.float_format = '{:,.4f}'.format
sk_df = pd.DataFrame([{'Features': c, 'Number of Unique Entries':
    initial_data[c].nunique(),
                       'Skewness':
                           initial_data[c].value_counts(normalize=True).values[0] *
                           100} for c in initial_data.columns])
sk_df = sk_df.sort_values('Skewness', ascending=False)
sk_df
```

This code shows that there are more than 10 columns, which have a skewness of more than 90%. I drop these particular columns by the following code snippet:

```
#I will drop columns whose skewness is greater than 90%
drop_cols = ['IsBeta', 'Census_DeviceFamily', 'Census_IsFlightsDisabled',
```

```
    'AutoSampleOptIn', 'Census_IsPortableOperatingSystem',
            'ProductName', 'HasTpm', 'UacLuaenable', 'Census_IsVirtualDevice',
               'Platform', 'OsVer', 'Firewall', 'Census_IsPenCapable',
            'IsProtected', 'Census_IsAlwaysOnAlwaysConnectedCapable',
               'Census_FlightRing', 'Census_HasOpticalDiskDrive', 'Processor',
            'Census_OSArchitecture']
initial_data.drop(drop_cols, axis=1, inplace=True)
```

Coding Categorical Variables A categorical variable is a variable, which consists of a string. For example, the color feature of a toy is a categorical variable. The machine learning algorithms do not like categorical variables in the data sets, so we have to convert them into digits by an appropriate encoding.

One way of encoding is called *label encoding*, where we simply assign an integer from 0 to $n-1$, where n is the number of unique entries in the categorical variable. Label encoding works well if there exists some kind of ordering between categories. For example, if the categories consist of strings such as "first", "second", "third", etc. then it makes sense to use label encoding. However, as in the previous example, if the category that we are dealing is the color of an object, such an encoding does not make sense, and it usually misleads our machine learning model.

Another way of encoding is called *one-hot encoding*, which works as follows. Assume that a categorical variable consists of n unique categories. What we do it that we add n additional column to the data frame, which indicate the existence of these n categories in that particular row. For example, assume that we want to encode the color of a toy, which takes the values "red", "green", and "blue", using one-hot encoding. We create 3 different columns with the names `Is_Red`, `Is_Green`, and `Is_Blue`. If a toy is red, then the column `Is_Red` takes the value 1, and the rest of the columns take the value 0. After that, we concatenate these columns with the original data frame, and finally we delete the categorical column.

Let us investigate the categorical variables in our data frame:

```
#We need to get rid of categorical variables in order to use sklearn learning
    methods
#Let's see number of unique entries in our data set
unique_entries = pd.DataFrame([{'Features': c,
                                'Type': initial_data[c].dtype,
                                'Number of unique entries':
                                   initial_data[c].nunique(),
                                'Unique entries':
                                   initial_data[c].unique().tolist()} for c in
                                   initial_data.columns])
unique_entries.sort_values('Number of unique entries', ascending=False,
    inplace=True)
unique_entries.reset_index(inplace=True, drop=True)
unique_entries
```

After running this column, we can investigate the number of unique categories in the categorical columns. Note that there are columns, which contain more than 700 categories within. It is not practical to add 700 new columns to the data frame after using one-hot encoding. So, we simply delete columns, which contain more than 12 unique categories within:

```
#Let us drop categorical features that contain more than 12 unique entries
columns_to_drop = ['AvSigVersion', 'OsBuildLab', 'Census_OSVersion',
    'AppVersion', 'EngineVersion',
                   'Census_ChassisTypeName', 'Census_OSEdition',
                       'Census_OSBranch', 'Census_OSSkuName',
                   'Census_MDC2FormFactor']
initial_data.drop(columns_to_drop, axis=1, inplace=True)
```

After that, we use one-hot encoding to encode the categorical features into integers:

```
#Use one-hot encoding to encode the categorical variables
categorical_columns = ['Census_PrimaryDiskTypeName', 'Census_GenuineStateName',
    'Census_OSWUAutoUpdateOptionsName',
                       'Census_ActivationChannel', 'SkuEdition',
                           'Census_OSInstallTypeName', 'OsPlatformSubRelease',
                       'Census_PowerPlatformRoleName']
prefixes = ['Census_PrimaryDiskTypeName_Is', 'Census_GenuineStateName_Is',
    'Census_OSWUAutoUpdateOptionsName_Is',
            'Census_ActivationChannel_Is', 'SkuEdition_Is',
               'Census_OSInstallTypeName_Is', 'OsPlatformSubRelease_Is',
            'Census_PowerPlatformRoleName_Is']
one_hot = pd.get_dummies(initial_data[categorical_columns], prefix=prefixes)
one_hot
```

And finally, we concatenate it with the original data frame, and remove the categorical features:

```
#Now, drop the categorical columns
initial_data.drop(categorical_columns, axis=1, inplace=True)
initial_data = initial_data.join(one_hot)
initial_data
```

Correlation Between Columns We can investigate the pairwise correlation between the columns. The Pearson correlation coefficient between two random variables is defined as:

$$\rho_{X,Y} = \frac{\text{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{12}$$

where $\text{Cov}(X,Y)$ is the covariance of the random variables X and Y, and σ_X, σ_Y are the standard deviations of X, and Y, respectively. One can mathematically show that the Pearson coefficient is bounded above and below by:

$$-1 \leq \rho_{X,Y} \leq 1 \tag{13}$$

and we say that two random variables are *uncorrelated* if $\rho_{X,Y} = 0$.

If the correlation between two columns is close to 1 or -1, then we can deduce some information from one column about the other one. So, we can make a very good guess about the outcome of the second column by solely looking at the first column. In machine learning, we do not want columns to be correlated.

We check the pairwise correlation between the columns in our data frame by using the code snippet below:

```
corr_df = initial_data.corr()
corr_df
```

The result of this code snippet is a data frame, which shows the pairwise correlation of the columns. I searched entries that are close to 1 or -1, then I deleted one of the columns by using the code snippet below:

```
#OsSuite and SkuEdition_Is_Home have a correlation of 0.99
#Census_OSBuildRevision and OsPlatformSubRelease_Is_th1 have a correlation of
    0.98
#SkuEdition_Is_Invalid and OsPlatformSubRelease_Is_windows7 have a correlation
    of 0.89
#Census_PowerPlatformRoleName_Is_Mobile and
    Census_PowerPlatformRoleName_Is_Desktop have a correlation of -0.82
#SkuEdition_Is_Pro and OsSuite have a correlation of -0.97
#SkuEdition_Is_Pro and SkuEdition_Is_Home have a correlation of -0.97
#Census_ActivationChannel_Is_Retail and Census_ActivationChannel_Is_OEM:DM have
    a correlation of -0.84
#Census_GenuineStateName_Is_IS_GENUINE and
    Census_GenuineStateName_Is_INVALID_LICENSE have a correlation of -0.90
#Census_PrimaryDiskTypeName_Is_SSD and Census_PrimaryDiskTypeName_Is_HDD have a
    correlation of -0.85
columns_to_drop = ['OsSuite', 'Census_OSBuildRevision', 'SkuEdition_Is_Invalid',
    'Census_PowerPlatformRoleName_Is_Mobile',
                  'SkuEdition_Is_Pro', 'Census_ActivationChannel_Is_Retail',
                     'Census_GenuineStateName_Is_IS_GENUINE',
                  'Census_PrimaryDiskTypeName_Is_SSD']
initial_data.drop(columns_to_drop, axis=1, inplace=True)
```

Now, our data set is relatively cleaner than the first data set that is handed to the online competitors by Microsoft. Now, we can begin with the machine learning tasks.

3.2 Machine Learning and Evaluation

Before feeding the data into the model, first we need to split the initial data frame so that we can test the performance of the model later. I used the following code snippet in order to split the initial data frame into testing and training data sets:

```
#Let us split our initial data into train and test data frames
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(initial_data.drop('HasDetections', axis=1),
                                                    initial_data['HasDetections'],
                                                       test_size=0.125,
                                                       random_state=42)
```

I divided the data set as 1:7, which is reasonable. Up to 1:3 is fine. One has to find the balance between the sizes of train and test data sets, otherwise some problems such as overfitting or underfitting may occur.

Balanced training set In a binary classification problem, it is important to have a balanced training set. By balanced we mean that the number of positives and negatives should be close to each other. Let us check the value counts in the training data set:

```
y_train.value_counts()
```

where `y_train` is the target variable in the training data set. It might be tricky to memorize the methods such as "value_counts()", but whenever in doubt, one can always consult the official documentation online. The result of this code snippet is given below:

```
1    3500187
0    3499375
Name: HasDetections, dtype: int64
```

We see that there are 3,500,187 rows with a positive value, and 3,499,375 rows with a negative value. Hence, the training set is balanced and we do not need to worry about balancing the training set before applying machine learning algorithms.

Algorithms with no success I tried to train my model using KNN, logistic regression and support vector machines, which I have mentioned in the previous section. Unfortunately, the training time takes way too long. I left the machine for 1 day alone, but the algorithm was still running. It can be either the case that the algorithm never halts, i.e. it does not converge, or it takes way too long to finish the process. From the theory of computation, we know that the time complexity for the KNN algorithm is $O(nd + kn)$, where n is the cardinality of the training set, d is the dimension of the training set, and k is the number of nearest neighbors. In our data set, n is around 7,000,000 and d is the number of columns, which is 62. Hence, it makes sense that it takes forever to train these models.

Decision tree I used the following code snippet in order to train my model, using the decision tree classifier:

```
#Let us use decision trees for the machine learning model
from sklearn import tree
clf = tree.DecisionTreeClassifier()
%time clf = clf.fit(X_train, y_train)
```

It took 6 minutes and 14 seconds to train the model.

After the training is finished, I tested the accuracy of the model with:

```
#Let us now predict the values
from sklearn.metrics import accuracy_score
y_pred = clf.predict(X_test)
y_pred_prob = clf.predict_proba(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred))
```

which yielded an accuracy of 55.06%. Using additional codes, I found the AUC equal to 0.551. The ROC curve of the decision tree classifier is shown in Figure 12. It was expected for the decision tree not to perform very well on this complex data set. The task itself is already intricate, i.e. we are trying to detect malware in a Windows machine solely from the data set that is provided to us. Normally, a malware detection is inherently a time-series problem, i.e. you cannot deal with malware codes independent of the time, nor do we investigate any executable files by means of special software tools in order to detect malicious scripts, cross-site scripting, etc.

LightGBM Next, I trained my model, using LightGBM algorithm. In this case, I partitioned my initial data set into three subsets, which are

- training set, as in the previous case,

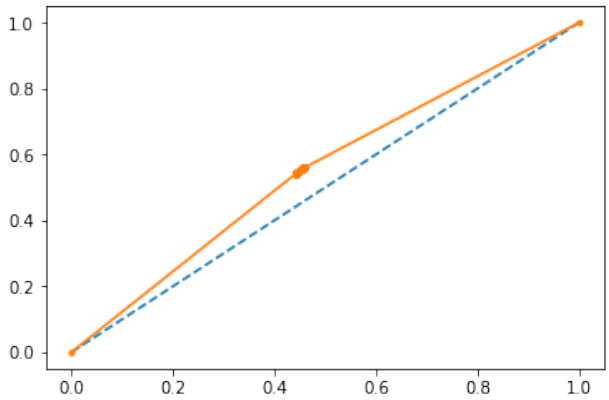


Figure 12: ROC curve of the decision tree classifier

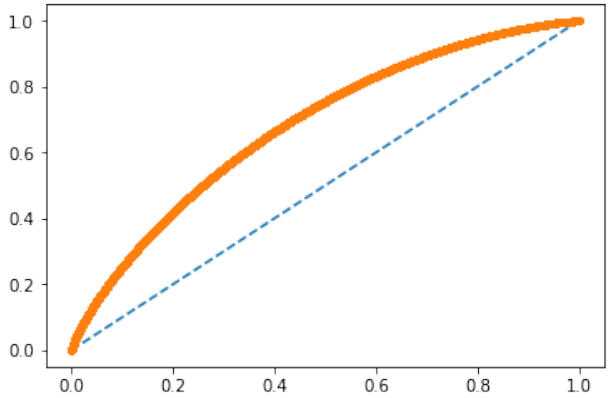


Figure 13: ROC curve of the LightGBM classifier.

- testing set, as in the previous case,
- and *validation set*, a different one.

The validation set is like the testing set, but we use the validation set more frequently, and we use it while training the model. This data set is used in order to fine-tune the hyperparameters for a machine learning algorithm. Hyperparameters are, for example, the number of estimators, the depth of the tree, the maximum number of leaves in the tree, etc. So, the model never learns from this data, but it uses this data set just for *validation* during training.

I create 3 subsets from the initial data set using the following code snippet:

```
#Let us split our initial data into train, test and cross validation data frames
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(initial_data.drop('HasDetections', axis=1),
                                                initial_data['HasDetections'],
                                                   test_size=0.25, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
    test_size=0.25, random_state=1)
```

I split the initial data set in the proportion $1 : 3/4 : 9/4$ for test, validation, and training, respectively. The training time took 17 minutes and 27 seconds. Testing the model on the testing data set, we find an accuracy of 63.17%, and AUC is equal to 0.682. The ROC curve of the model is shown in Figure 13.

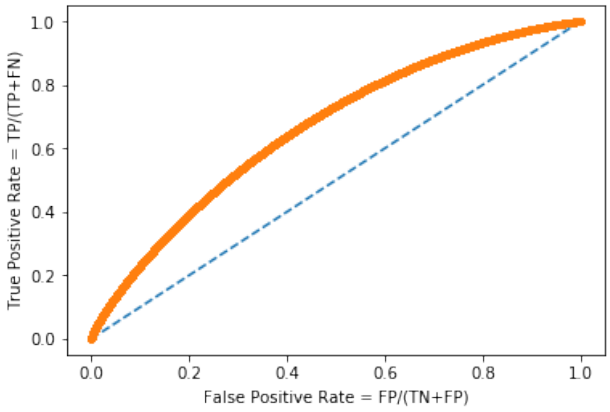


Figure 14: ROC curve of the random forest classifier

This algorithm gave very satisfactory results in such a short training time. Indeed, it can handle very large data efficiently.

Random forests Unlike in the LightGBM case, we do not need to specifically create a validation set for the random forest since it will automatically do it itself. So, we partition the original data set into two sets, with proportions 1 : 3.

Now, we define our model `m` as follows:

```
from sklearn.ensemble import RandomForestClassifier
m = RandomForestClassifier(n_estimators=100, min_samples_leaf=5,
    max_features=0.5, n_jobs=-1, verbose=100)
%time m.fit(X_train, y_train)
```

Verbosity means that during training and testing, it outputs interactive things. Without that argument, we see the output only at the end of the training. The other variable number of jobs is the number of processors being used during training, and the argument -1 means that there is no limit. The training time took 57 minutes and 22 seconds, which was the longest among all of them.

Now, we use same procedure as with the other models to evaluate the performance of the algorithm. It turns out that the accuracy is 61.92%, and AUC is 0.665. The ROC curve is given in Figure 14.

4 Realistic Constraints

In this section, I would like to point out the essential limitations of using some algorithms. Then, I would like to mention the standards that I have followed throughout the project.

4.1 Cost Analysis

There have been no physical costs such as labor costs, laboratory costs, etc. for this project. However, there was an important cost, which is called *the computational costs*.

First is the *memory cost*. I had to read the huge data set into the computer memory (RAM), and as I checked the task manager, I could see that around 8 gigabytes of computer memory was being used by the program Python. So, it is an important requirement to have a computer, which has a lot of free random access memory to spare. Mine was 16 gigabytes, so I did not face many difficulties during reading the data set, and training the model.

Second is the *time cost*. Algorithms like SVM, KNN, and logistic regression simply did not work out for me. As I mentioned in the previous sections, the time complexity of the KNN and other algorithms is huge, and it simply cannot finish training the model in a reasonable time. I simply tried using all of the possible algorithms, more like a trial-and-error method.

4.2 Standards

This is an open-source competition. Even though I did not be participate in the competition officially, I published my findings on code-sharing websites such as GitHub. This is a great way of contribution since everybody has access to what you have done in that particular area, and they can put additional materials on top of it. I also used Python since it is one of the best programs out there for machine learning and data science. The best thing about Python is that it is open source, so everybody can contribute to its development. Furthermore, there are many useful packages in Python for machine learning and data science, the most important ones being **Pandas, Scikit-learn** and **Tensorflow**. Jupyter Notebooks are one of the cleanest working environments for Python. Anaconda distribution for Python [16] provides these notebooks, as well as many useful packages preinstalled. For example, Pandas is already preinstalled there. That is why I used Anaconda throughout my project.

Such open source software codes use the *MIT License*, open source initiative approved license. [17]

5 Conclusion

This has been a very good term project for me, where I had the opportunity to practice with a hands-on task in machine learning and data science. I learned how to use the programming language Python and its libraries for the machine learning tasks. I had the opportunity to train various machine learning models, and evaluate their performances for the given data set. Overall, I think I had a fruitful term project, and I am happy that I chose such a topic like this since I want to specialize in the field of information security, and this was the perfect project to combine machine learning and information security topics under one single roof.

References

[1] https://www.kaggle.com/c/microsoft-malware-prediction

[2] http://qeprize.org/createthefuture/margaret-hamilton-apollos-code/

[3] https://www.pbs.org/newshour/science/katie-bouman-hardly-knew-what-a-black-hole-was-her-algorithm-helped-us-see-one

[4] https://emerj.com/ai-glossary-terms/what-is-machine-learning/

[5] https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[6] https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7

[7] https://towardsdatascience.com/support-vector-machine-vs-logistic-regression-94cc2975433f

[8] https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

[9] https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

[10] https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab

[11] https://github.com/microsoft/LightGBM

[12] https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

[13] The image is taken from: http://docs.statwing.com/the-confusion-matrix-and-the-precision-recall-tradeoff/

[14] The image is taken from: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

[15] The images are taken from the interactive web application: http://www.navan.name/roc/

[16] https://www.anaconda.com/

[17] https://opensource.org/licenses/MIT