Chapter 1

Introduction

Cyber Physical Systems (CPS) are nowadays widely used in different application domains, such as smart-homes, smart-cities, hospitals, etc... They are mainly composed of two entities: a cyber part consisting in a computing and networking component, and a physical part consisting in different controllers and sensors. The existence of a connected cyber part implies its susceptibility to multiple cyber threats. The malfunctioning of these systems, due to a cyber threat, can cause severe impacts on the real life and the safety of the community, for example a blackout or water contamination. That is why many algorithms have been designed for the security monitoring of those systems, in particular the anomaly and attack detection.

Nowadays, machine and deep learning algorithms are used to detect those anomalies and intrusions. But, in majority, they rely only on the cyber part of the systems and on the data describing their behaviour, ignoring their physical models. The idea behind this work is to employ a hybrid machine learning algorithm, in particular neural networks, to detect anomalies and attacks in CPS considering its physical model.

1.1 Physic guided machine learning in literature

As mentioned before, the aim of the work is to fuse the black-box and theory-based models together to get better predictions. However this is not the first time such a fusion is examined. In the literature various approaches of the fusion of neural networks with theory-based models were presented. Those approaches can be divided into two types given what aspect of the algorithm they're changing: those that modify in first place the input to take into consideration the physical constraints, and those that modify the structure of the neural network.

–>here comes some more explanations–<

1.2 Case study

In order to focus on the implementation of the hybrid machine learning algorithm, a CPS, with ready to use datasets, was chosen from a list provided in [1]: the **power system** [2], which network diagram was represented on figure 1-1. The system is composed of two power generators who are alimenting the whole system. Intelligent Electronic Devices (IEDs) R1 to R4 and the breakers BR1 to BR4 can be found connected directly to those generators. Each IED switches its corresponding breaker when a fault is detected, valid or fake. The communication between the IEDs and the Substation Switch is done wirelessly. On the other hand the Substation Switch is connected with the Primary Domain Controller (PDC) and the Control Room.

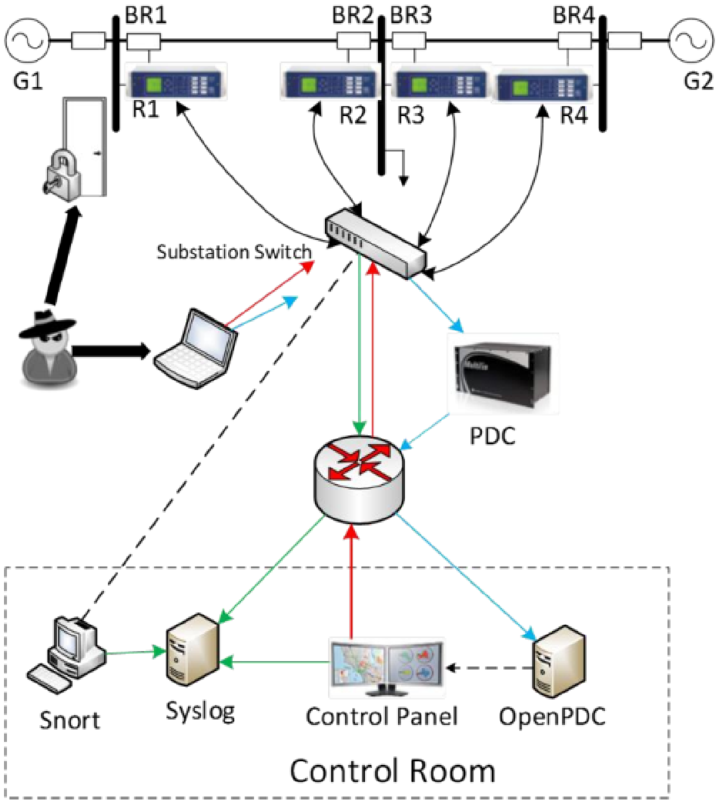


Figure 1-1: Power system network diagram [2]

The operation of this power system can be described following 6 main scenarios:

- normal behaviour,
- short-circuit,
- line maintenance,
- remotely opening the breakers (attack),
- disruption of fault protection system (attack),

- fault imitation (attack).

Each of those scenarios can be divided into several sub-scenarios concerning different entities of the system or/and the failure range. Every scenario was labelled with a number between 1 and 41. In this way **37 scenarios** are obtained, divided and numbered as follows:

- 1 no events scenario, its number it is 41,
- 8 natural fault scenarios, its number ranges are 1-6 (short-circuit) and 13-14 (line maintenance),
- 28 attack scenarios, its number ranges are 7-12 (fault imitation), 15-20 (remotely opening the breakers), 20-30 and 35-40 (disruption of fault protection system).

The reason for dropping the numbers between 31 and 34 in the naming process of scenarios is not known.

The datasets provided in [1] represent **78377 events**, in which one of those scenarios was reproduced in the system. They have been grouped by scenario into 3 datasets: binary (attack or normal operation), three-class (attack, normal fault and no events) and multiclass (differentiating all 37 scenarios). Each of these 3 datasets is composed of 15 .arff or .csv files comporting in average 141 events for each of 37 scenarios. The exact number of events per file for each scheme is illustrated on figure 1-2. For the 3 class dataset **55663 attack**, **18309 natural fault** and **4405 normal operation** events were found. The distribution of these schemes throughout the files is shown on figure 1-3.

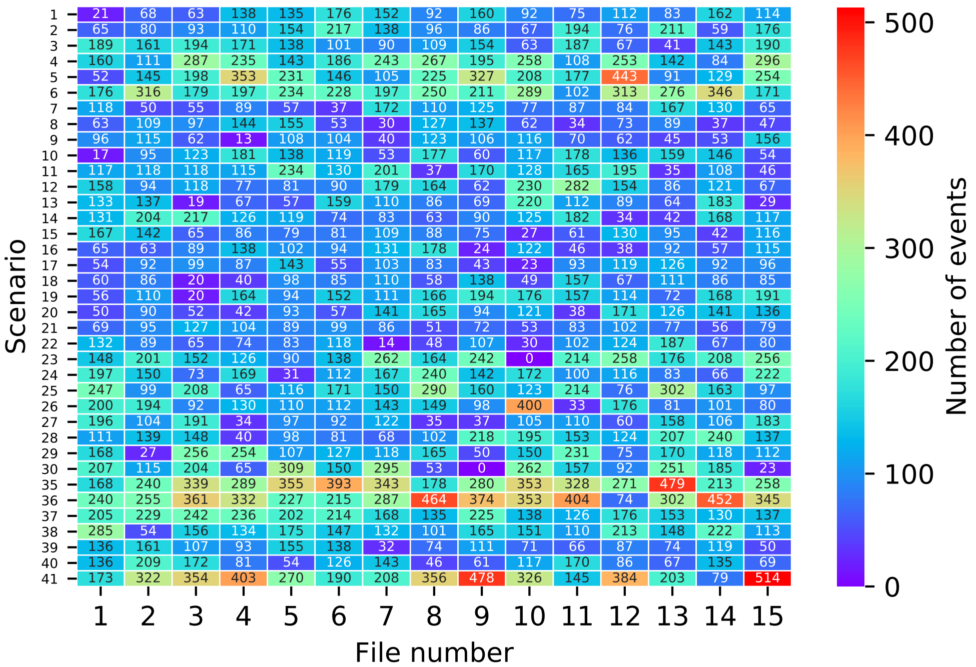


Figure 1-2: Scenarios distribution throughout all 15 files

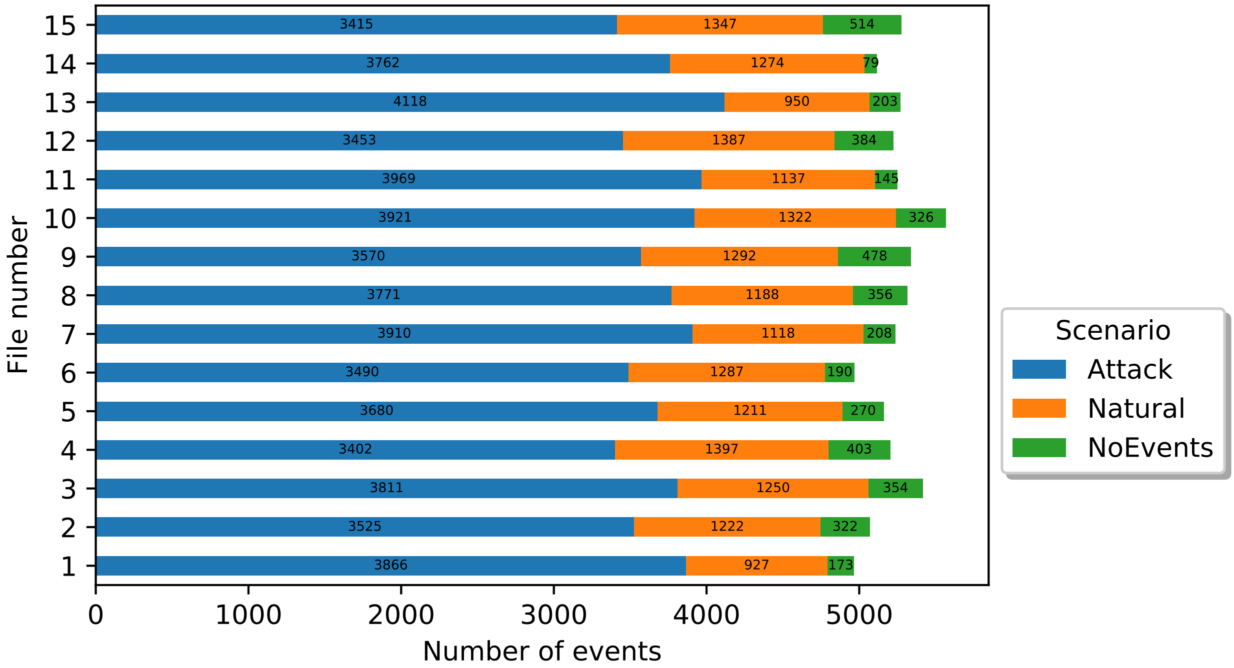


Figure 1-3: Scenarios distribution throughout the 3-class dataset files

Figure 1-3 shows also this distribution for the binary datasets. It is sufficient to add the number of natural (orange) and normal operation (green) events.

The scenarios are not equally distributed in the case of the 37 schemes dataset, it is especially shown by the standard deviation of 61, which is an important value compared to some scenarios counting less than 100 events. On the other hand, in the case of 3-class scenarios, the distribution is even more not equal compared to the 37 schemes dataset. The **mean standard deviation among all files is equal to 1767**, which is an enormous result given that some scenarios count only around 100 events.

Every electrical grid around the world uses a **3-phased** electric power. Such a grid is composed of three alternating current generators combined. Those generators pass the current in three conductors. That way three conductors are obtained, and each of them conducts a phase of current named A, B and C respectively. The current phases have the same frequency, but a difference of phase of 1/3 of a cycle between each of them. In addition to that, each current has a corresponding voltage, with the same frequency and phases differences [3].

In order to simplify the analysis of three-phase power systems, symmetrical components transformation is used, for both voltage and current. This transformation is defined as:

$$\begin{bmatrix} V_0 \\ V_1 \\ V_2 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & a & a^2 \\ 1 & a^2 & a \end{bmatrix} \begin{bmatrix} V_A \\ V_B \\ V_C \end{bmatrix}, \tag{1.1}$$

where $\mathbf{V_0}$, $\mathbf{V_1}$, $\mathbf{V_2}$ are called respectively **zero sequence**, **positive sequence** and **negative sequence**, $a = e^{i\frac{2\pi}{3}}$ and V_A, V_B, V_C the A-C voltage phases [4]. As each sequence is a weighted sum of sinusoidal functions (A-C phases), it can be on its own written as one sinusoidal function after mathematical transformations.

Each phase is a **sinusoidal** function. Its equation form is $y = A \cdot \sin(\omega t + \theta)$, where A is the amplitude, ω the angular frequency and θ the initial phase. Two terms will be used in what follows: the **magnitude** which is the absolute value of the amplitude and the **angle** which refers to initial phase. These two variables were illustrated on figure 1-4.

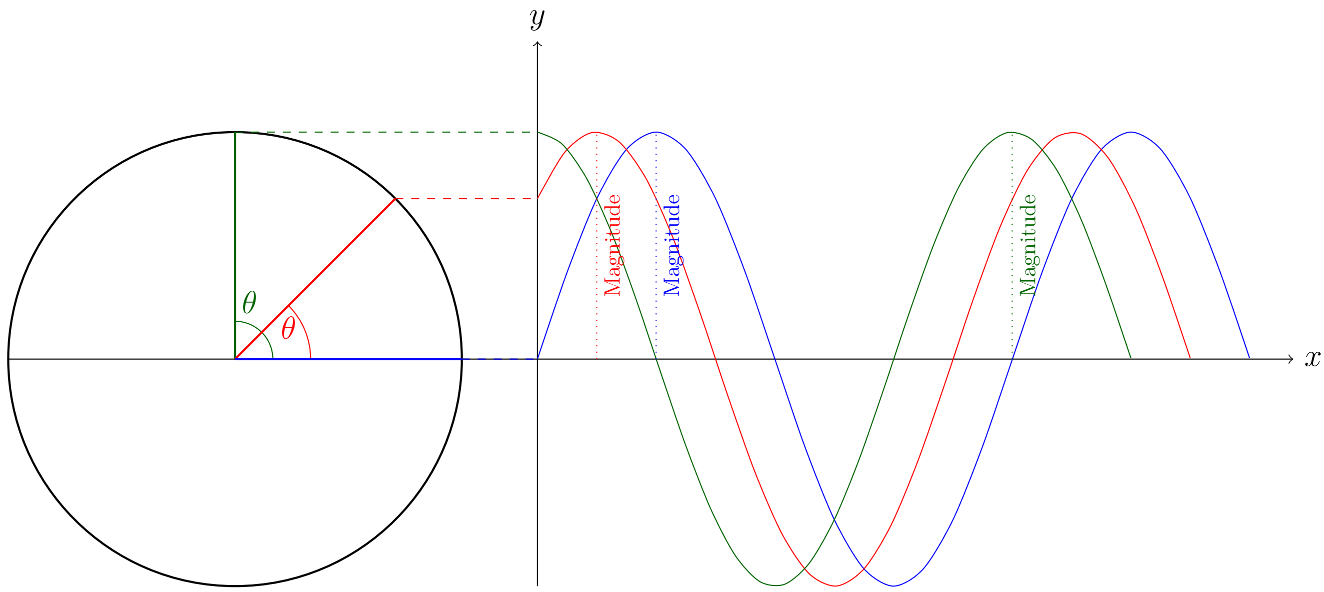


Figure 1-4: Magnitude and angle (θ) of sinusoidal functions

Every previously mentioned event is described by **128 features**: 116 provided by four IEDs (each one provides 29 types of measurements) and 12 other features are reserved for control panel logs, snort alerts, relay logs of 4 IEDs. The mentioned 116 features, each has a label formed by **concatenation** of the **source IED reference** (it can be R1, R2, R3, R4) and the **measurement name**, as provided in table 1.1. For example R4-PM5:I stands for phase B current phase magnitude measured by R4.

Table 1.1: IED measurements [2]

Feature	Description
PA1:VH – PA3:VH	Phase A-C Voltage Phase Angle
PM1:V – PM3:V	Phase A-C Voltage Phase Magnitude
PA4:IH – PA6:IH	Phase A-C Current Phase Angle
PM4:I – PM6:I	Phase A-C Current Phase Magnitude
PA7:VH – PA9:VH	Pos.–Neg.– Zero Voltage Phase Angle
PM7:V – PM9:V	Pos.–Neg.–Zero Voltage Phase Magnitude
PA10:VH - PA12:VH	Pos.–Neg.–Zero Current Phase Angle
PM10:V - PM12:V	Pos.–Neg.–Zero Current Phase Magnitude
F	Frequency for relays
DF	Frequency Delta (dF/dt) for relays
PA:Z	Appearance Impedance for relays
PA:ZH	Appearance Impedance Angle for relays
S	Status Flag for relays

Those datasets have been used in several works related to CPS cyber-attack classification, one of which is [5], where the author try to find the most accurate algorithm to predict the status of the power system. The following chapter shows an attempt to partially reproduce the results obtained by them.

Chapter 2

Machine learning algorithms comparison

Before going further and analysing neural networks, a deeper look at classical machine learning algorithms will be taken, in particular Random Forest and Support Vector Machine (SVM) in the context of anomaly detection in the CPS presented in chapter 1. However this was done before in [5] using the black-box model algorithms only. In their approach they used Weka [6] in order to find the most performant algorithm among 7 they have chosen (OneR, NNge, Random Forest, Naïve Bayes, SVM, JRipper, Adaboost).

This chapter shows an attempt to reproduce the results provided in [5], using two different machine learning toolkits (Weka and scikit-learn [7]) in order to confirm the obtained results. That is why, first, these two toolkits will be presented, then the obtained results will be discussed.

2.1 Weka

Weka, or more exactly **W**aikato **E**nvironment for **K**nowledge **A**nalysis, is an open source machine learning software developed at The University of Waikato in Hamilton, New Zealand and based on Java programming language. It is well known especially in academic environments and a lot of machine learning researches were conducted using it, one of them the mentioned before [5]. It incorporates various machine learning tools: classifiers, regressors, visualizers, data pre-processor etc...

Weka is characterised by 3 main operating schemes. First, it can be run using a **graphical user interface** (GUI), enabling the user, even without deep knowledge in programming, to make machine learning experiments and analyse available classifiers. Second, more advanced users have the option to run all the available tools using a **command line**. Finally, Weka's tools can be **integrated directly into code in several programming languages (Java, Python, R, Spark)**, which enables even larger versatility.

2.2 scikit-learn

scikit-learn is an open-source machine learning **Python library** developed originally by David Cournapeau as a Google Summer of Code project and now it is maintained by a team of volunteers. It is well known in both academic and commercial environments, since it is used by many enterprises such as Spotify, Evernote, Booking.com, and research facilities like Inria or Télécom ParisTech [8]. It includes, just like Weka, various machine learning tools such as classifiers, regressors, data preprocessor etc... Its visualisation capabilities are limited, however there exist many additional Python packages for data visualisation such as YellowBrick, Eli5 and others... They will be further discussed in next chapter.

scikit-learn can be used only as an extension of Python language, what makes a bit harder for non experts to start working with it. However, since Python is an user friendly programming language and scikit-learn has a well made documentation, this toolkit is easy to use. Along with other Python packages, and especially pandas, scikit-learn is a very powerful, ergonomic and versatile solution for machine learning problems.

It might be also interesting to mention that scikit-learn can be used within Weka after installing the appropriate add-on. This enables Weka users to use scikit-learn classifiers and regressors and run Python code just inside Weka's GUI.

2.3 Metrics for classifiers comparison

After the presentation of the used toolkits, in order to be able to compare machine learning algorithms the following metrics are introduced:

- **accuracy**: ratio of correct classifications over the total number of samples,
- **precision**: ratio of correct classifications for a particular class over all classifications that indicated that class,
- **recall**: ratio of correct classifications for a particular class, over all samples corresponding for this class,
- **f-measure**: weighted average of precision and recall given by the equation:

$$\text{f-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

In addition to that, precision, recall and f-measure metrics, can be calculated in three different ways:

- **micro**: the metrics are determined globally by calculating true positives, false negatives and false positives,
- **macro**: the metrics are calculated for each class, then it gives their unweighted mean value,
- **weighted**: the metrics are calculated for each class, then it gives their weighted average value by the number of true instances for each class.

2.4 Experimental setup

In order to reproduce the results presented in [5], Weka version 3.8.4 and scikit-learn version 0.23.1, running on Anaconda 3.18.11 with Python 3.7.6.final.0, were used. Given the availability of classifiers between these two toolkits, only 3 classifiers were chosen: SVM, NaïveBayes and Random Forest. SVM in Weka comes from a package untitled libsvm available for this toolkit. The used parameters are presented for both scikit-learn and Weka in tables 2.1 and 2.2 respectively in order to help in the reproduction of this work in the features. For more information about particular parameters the references to their explanation are given in square-brackets next to each classifier.

Table 2.1: Scikit-learn classifiers parameters

(a) Random Forest [9]

Parameter	Value
n_estimators	100
criterion	"gini"
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	"log2"
max_leaf_nodes	None
min_impurity_decrease	0.0
min_impurity_split	None
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None

(b) SVM [10]

Parameter	Value
C	1.0
kernel	"rbf"
degree	3
gamma	"scale"
coef0	0.0
shrinking	True
probability	True
tol	1e-3
cache_size	7000
class_weight	None
verbose	False
max_iter	1000
decision_function_shape	"ovr"
break_ties	False
random_state	None

(c) NaïveBayes [11]

Parameter	Value
priors	None
var_smoothing	1e-9

Table 2.2: Weka classifiers parameters

(a) Random Forest [12]

Parameter	Value
bagSizePercent	100
batchSize	100
breakTiesRandomly	False
calcOutOfBag	False
computeAttributeImportance	False
debug	False
doNotCHeckCapabilities	False
maxDepth	0
numDecimalPlaces	2
numExecutionSlots	1
numFeatures	0
numIterations	100
outputOutOfBagComplexityStatistics	False
printClassifiers	False
seed	1
storeOutOfBagPredictions	False

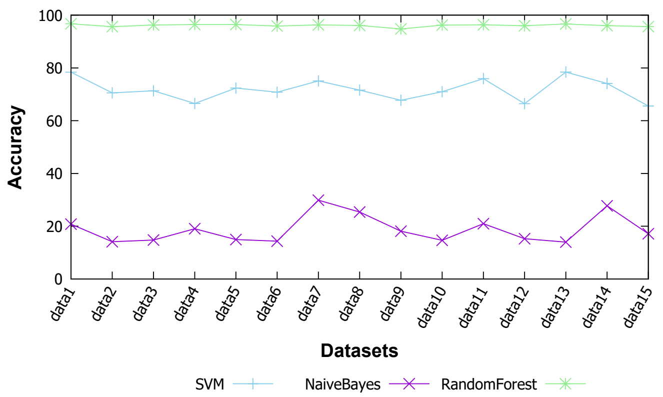
(b) SVM [13]

Parameter	Value
SVMType	C-SVC (classification)
batchSize	100
cacheSize	40.0
coef0	0.0
cost	1.0
debug	False
degree	3
doNotCHeckCapabilities	False
doNotReplaceMissingValues	False
eps	0.001
gamma	0.0
kernelType	radial basis function
loss	0.1
modelFile	Weka-3-8-4
normalize	False
nu	0.5
numDecimalPlaces	2
probabilityEstimates	False
seed	1
shrinking	True
weights	

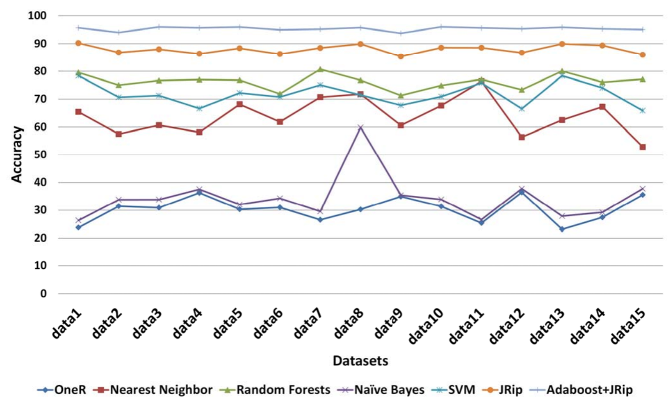
(c) NaïveBayes [14]

Parameter	Value
batchSize	100
debug	False
displayModelInOldFormat	False
doNotCHeckCapabilities	False
numDecimalPlaces	2
useKernelEstimator	False
useSupervisedDiscretization	False

2.5 Comparison results

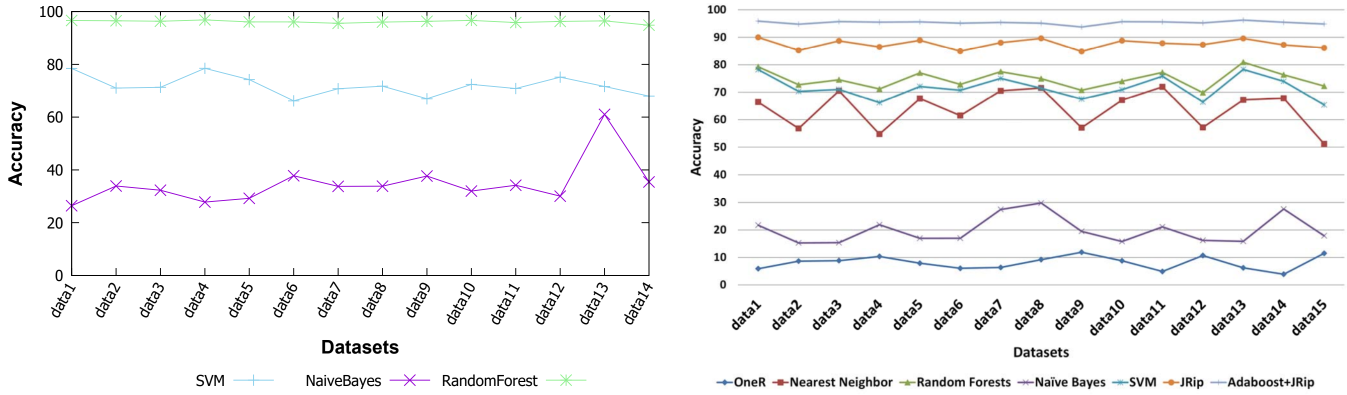


(a) Our attempt



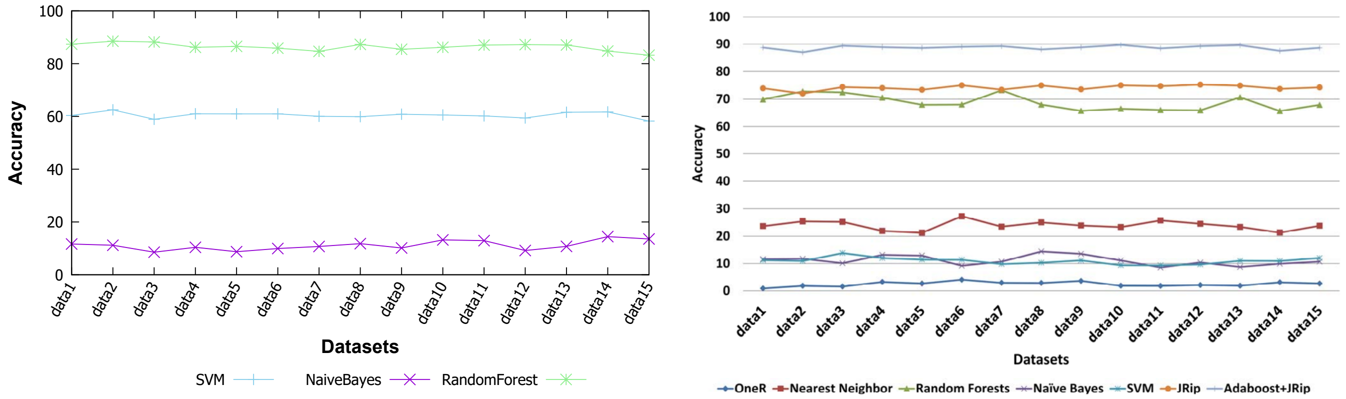
(b) Original results [5]

Figure 2-1: Accuracy for three-class datasets



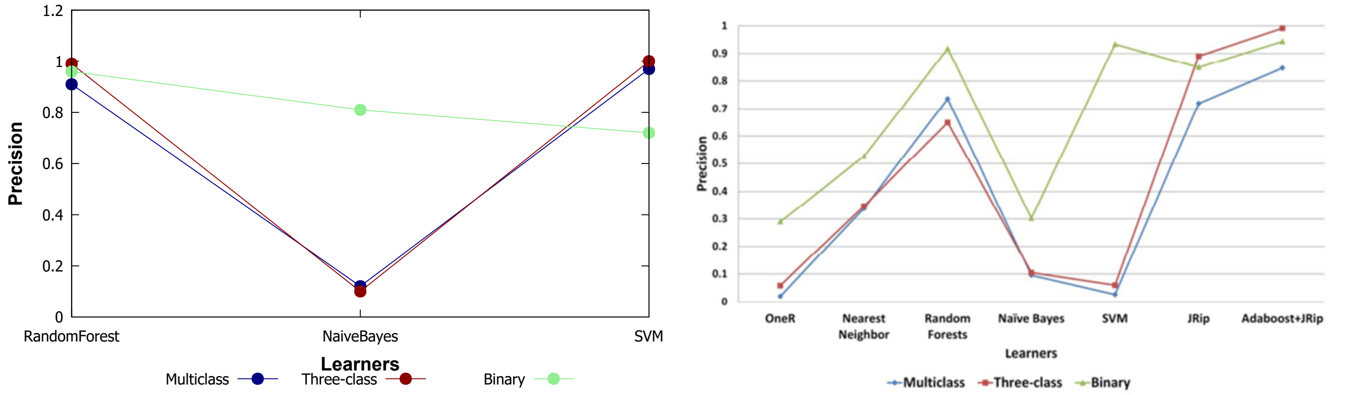
(a) Our attempt (b) Original results [5]

Figure 2-2: Accuracy for binary datasets



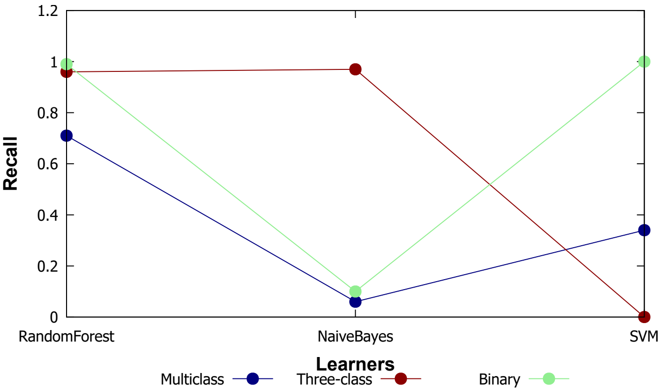
(a) Our attempt (b) Original results [5]

Figure 2-3: Accuracy for multiclass datasets

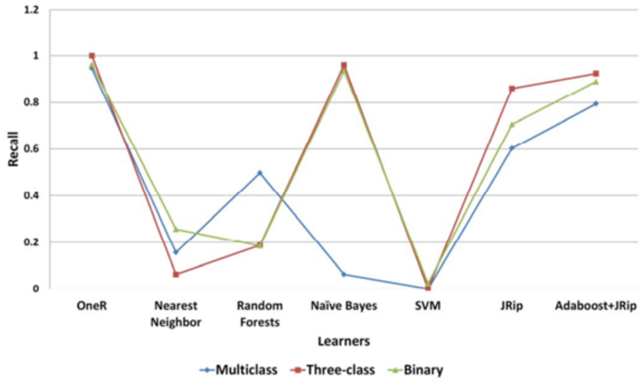


(a) Our attempt (b) Original results [5]

Figure 2-4: Precision

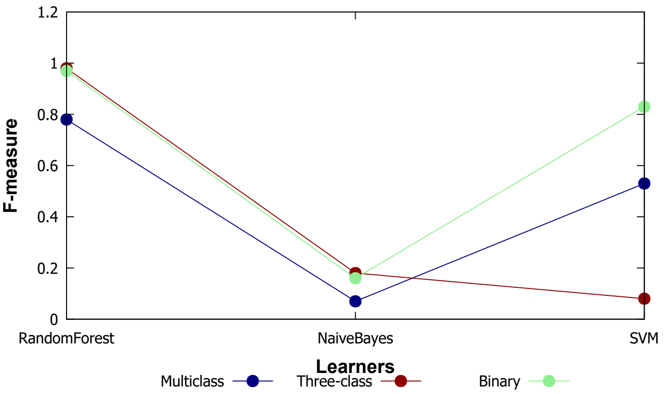


(a) Our attempt

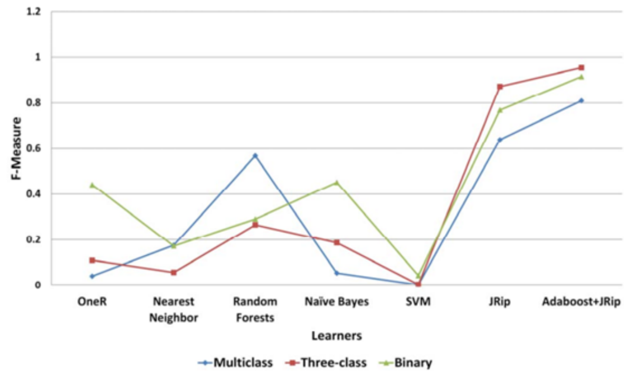


(b) Original results [5]

Figure 2-5: Recall



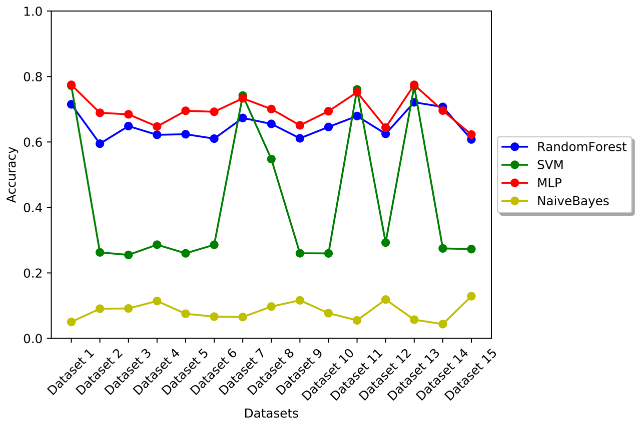
(a) Our attempt



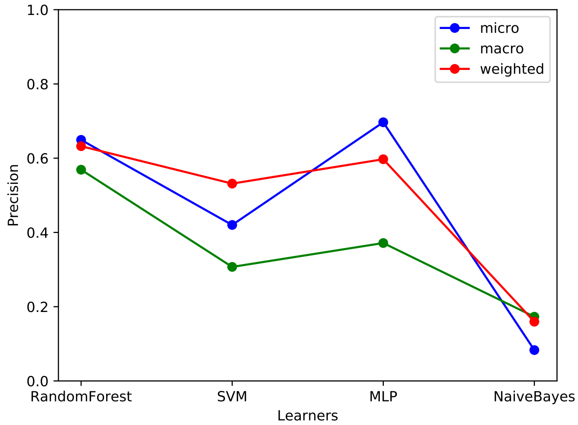
(b) Original results [5]

Figure 2-6: F-measure

The obtained results indicate clearly that Random Forest algorithm is the more accurate and gives clearly the best results, with Adaboost+JRIP with slightly worse performance. On the other hand, the results presented in [5] shows better results for Adaboost+JRIP. For SVM and Naïve Bayes the results are comparable, expect for precision value for SVM. The origin of this difference is so far unknown.

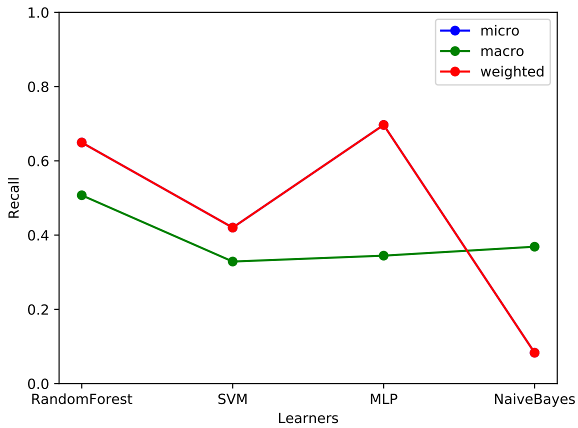


(a) Accuracy

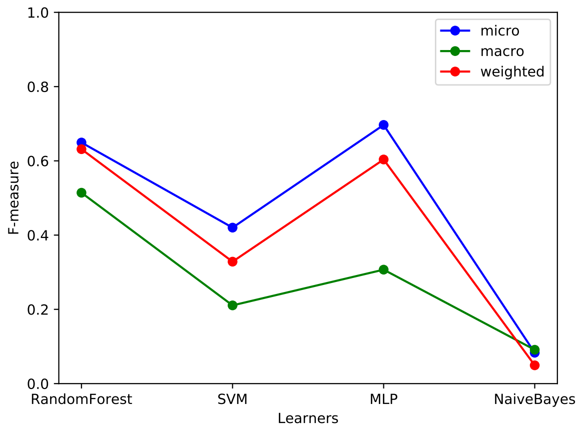


(b) Precision

Figure 2-7: scikit-learn results



(c) Recall[a]



(d) F-measure

[a]micro and weighted values are the same in this case.

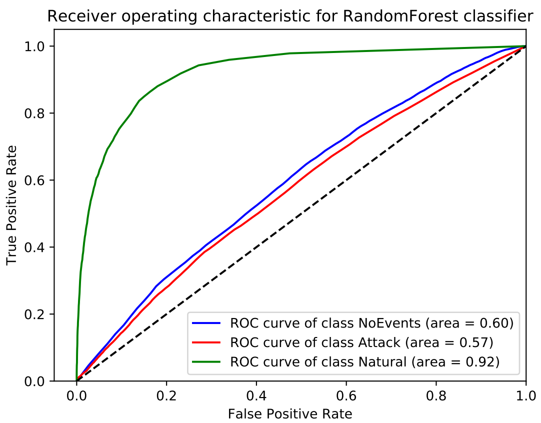
Figure 2-7: scikit-learn results

It can be observed that the obtained results are partially different from those obtained using Weka. The results for MLP and Naïve Bayes are comparable to those from Weka, but on the other hand, the results for Random Forest and SVM differ considerably. This made MLP the most reliable classifier compared to others in this comparison.

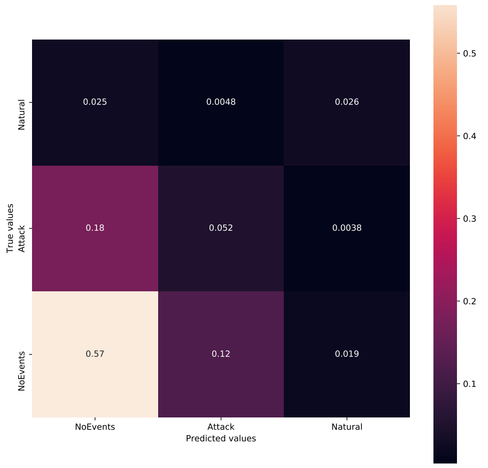
It can be also deducted that Weka is calculating metrics globally (corresponds to micro in scikit-learn).

2.6 scikit-learn further methods' analysis

In addition to all that, scikit-learn enables the user to plot the receiver operating characteristic (ROC) curves for each class and the confusion matrix. The ROC curve represents the plot od true positive rate when the false positive rate changes. The confusion matrix on the other hand shows the normalized number (over the total number of samples) of predicted values of each class for each class. The results are illustrated on figures 2-8, 2-9, 2-10 and 2-11.

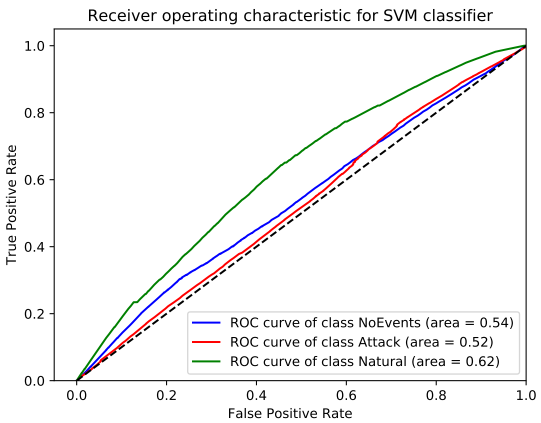


(a) ROC Curve

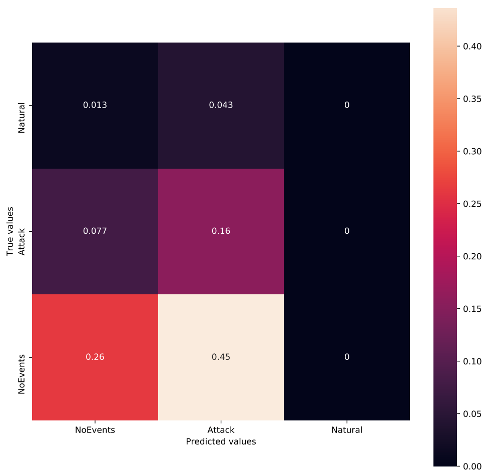


(b) Confusion Matrix

Figure 2-8: Random Forest ROC curve and confusion matrix

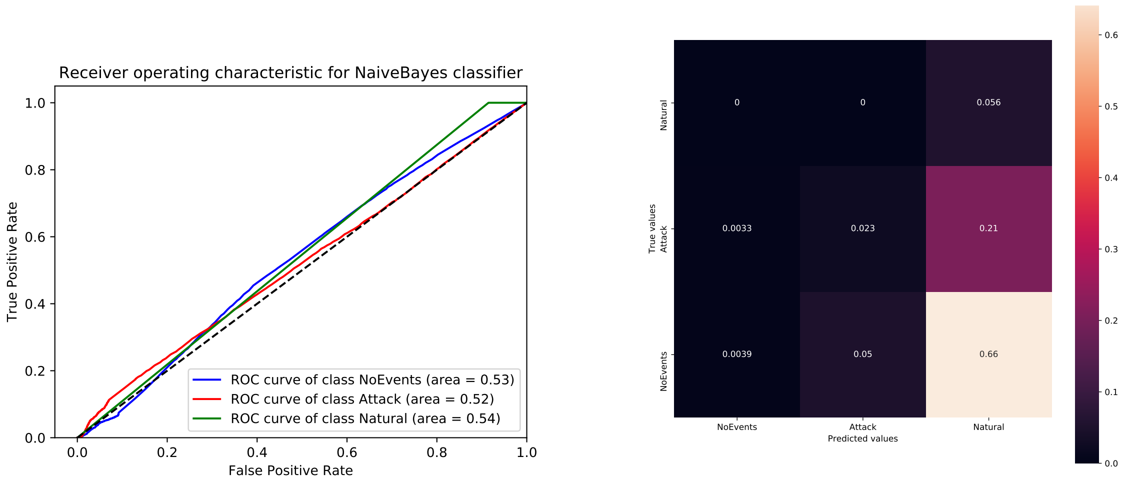


(a) ROC Curve



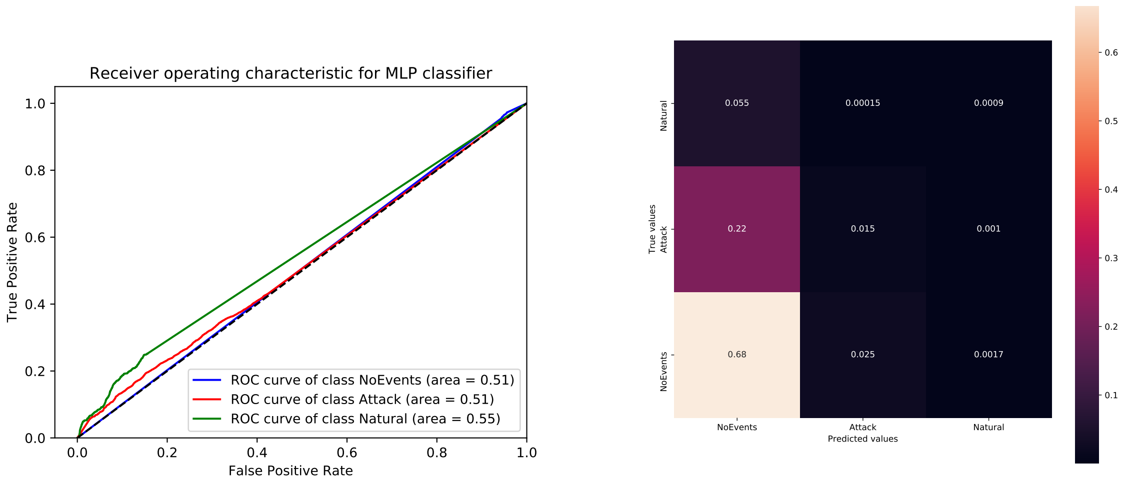
(b) Confusion Matrix

Figure 2-9: SVM ROC curve and confusion matrix



(a) ROC Curve (b) Confusion Matrix

Figure 2-10: Naïve Bayes ROC curve and confusion matrix



(a) ROC Curve (b) Confusion Matrix

Figure 2-11: MLP ROC curve and confusion matrix

The previous figures show that Random Forest classifier has the higher capacities to distinguish between the occurrence of each class, or it's absence. It's visible on both ROC curve and the confusion matrix, where the highest number of predictions is shown for the true positives for each class. SVM tends to predict only NoEvents and Attacks but does not really succeed in distinguishing between them. Naïve Bayes fails to make true predictions, it considers everything of class natural. Finally MLP, it succeeds in determining the class NoEvents, but does not distinguish over classes almost at all, despite the high accuracy (it is due because of the huge number of samples of class NoEvents).

Given this analysis, it can be deducted that Random Forest algorithm acts the best, and that is why it will be adapted in next chapters, in which, at first, an analysis of features and their importance will be made. However a deeper look at the amelioration of MLP will be also made later.

Chapter 3

Features' importance

After having determined the most performant algorithm, which is Random Forest, it is time to go further and analyse which features impact the results of classification the most. The focus will be especially on false predictions. In order to do that, six tools will be compared: LIME [15], ELI5 [16], YellowBrick [17], Treeinterpreter [18], dtreeviz [19] and export_graphviz tool from scikit-learn, where the last three ones are designed for Decision Tree and Random Forest classifiers.

3.1 Result's interpreters' comparison

3.1.1 LIME

LIME (Local Interpretable Model-agnostic Explanations) is a tool that is used to explain the behaviour of machine learning classifiers. It supports, as for this day, only the explanation of individual predictions for any scikit-learn classifier or regressor. This explanation consist in a list of features ordered by their relative importance for a particular prediction. This list can be shown is a raw mode (as a python list) or in a visual form (pyplot figure, jupyter notebook or html file).

In order to class the features according to their importance, LIME approximates the model by an interpretable one, created based on perturbing the features of the examined instance. More the perturbed instances are similar to the examined instance, higher is the weight of the perturbed feature.

3.1.2 ELI5

ELI5 (Explain like I am a 5-year old) is a tool, in form of a Python package, used to debug machine learning classifiers and explain their predictions. It supports multiple machine learning frameworks, including scikit-learn. It can be used to explain how the model works both locally for one prediction and globally for the whole model. The output can take several forms just like in LIME case.

For white-box models, ELI5 works as an extension of scikit-learn and it's capable to extract the weights of model's features for different classes. In addition to that it can show the weights that contributed in a particular prediction. On the other hand, for black-box models, this tool integrates a modified version of LIME, supporting more machine learning frameworks, and a permutation importance method, which checks how the model's accuracy decreases when removing one of the features and on this basis determines the importance of the features.

3.1.3 YellowBrick

YellowBrick is another Python package, which is an extension of scikit-learn framework. It is meant to give global interpretation of the analysed model on different levels. It is possible not only to visualize features importances calculated directly by scikit-learn, but also to give a classification report (accuracy, recall, precision, f-measure), plot a confusion matrix, a ROC curve and much more. In addition to all that, YellowBrick comes with a tool for determination of correlation between features in the dataset.

3.1.4 Treeinterpreter

Treeinterpreter is a simple Python package that works with scikit-learn trees and random forest classifiers. It's only usage consists in decomposing the obtained prediction into bias and contributions of different features. The output is given in the form of a numpy array.

3.1.5 scikit-learn export_graphviz

export_graphviz is a scikit-learn embedded function that enables the user to visualise a decision tree with all the branches and save it into Graphviz[1] format, that can be converted into a vector graphic. It is possible also to visualise decision trees composing random forest model in scikit-learn, since the possibility to extract particular decision trees when using this framework. However the interpretability of results for random forest classifier can be hard.

3.1.6 dtreeviz

dtreeviz is a more advanced version of export_graphviz available in scikit-learn. For every leaf in the tree it can show a histogram indicating the influence of feature value on class selection. In addition

[1]a set of tools for diagram creation using graphs.

to that, dtreeviz enables the user to show the path of a particular prediction. The result is saved in the form of a svg vector graphic.

3.1.7 Summary

It can be concluded that ELI5 is the most versatile package compared to others, especially because it enables both global and local interpretations and does not limit its support to scikit-learn, plus it has LIME integrated in it. YellowBrick, on other hand, adds the possibility to analyse from a statistical view the features available in the dataset. Finally comes Treeinterpreter and dtreeviz that are interesting tools when analysing especially Decision Trees. A summary of the most import features of all 5 packages is shown in table 3.1, where 6 comparison metrics where taken into consideration:

- global interpretation: capacity of the tool to interpret the whole model,
- local interpretation: capacity of the tool to interpret a particular sample from the dataset,
- black-box models support: the fact if the tool supports only black-box models (models that can not be simply interpreted),
- features' statistical analysis: the fact if the tool supports statistical analysis of features in the dataset, without taking into consideration the model,
- works only with scikit-learn,
- decision trees graphical visualisation.

Table 3.1: Comparison of tools for model analysis

	LIME	ELI5	YellowBrick	Treeinterpreter	dtreeviz
Global interpretation	✗	✓	✓	✗	✗
Local interpretation	✓	✓	✗	✓	✓
Black-box models support	✓	✓	✓	✗	✗
Features' statistical analysis	✗	✗	✓	✗	✗
Works only with scikit-learn	✓	✗	✓	✓	✓
Decision Trees graphical visualisation	✗	✗	✗	✗	✓

3.2 Features' importance determination

For the rest of the chapter LIME was chosen to determine the features' importance because of it working with all black-box models available in scikit-learn. The capabilities of LIME were sufficient and that is why ELI5 was not used in his place.

Since in this case the explanations of single samples are not really interesting, an attempt to generalize the results was made: Lime explainer was run on 100 false predictions of a chosen class. The results are concatenated together, and for all the features that are duplicated, the importance is calculated as the mean value of the importances and only one entry is kept with the calculated average importance. This algorithm was also run omitting the differentiation between classes. The results, reduced to 10 entries each, are shown below.

For NoEvents class:

```
                                    importance
feature
R2-PM1:V > 130872.03                -0.013013
R3-PA2:VH <= -93.75                 -0.011134
R2-PA7:VH <= -101.20                -0.010218
R3-PM2:V > 130431.15                -0.009583
R2-PM7:V > 130857.40                -0.009377
...                                       ...
R3-PM5:I <= 330.70                   0.005762
0.00 < R1-PA12:IH <= 32.04           0.007514
128762.21 < R2-PM1:V <= 129859.49    0.007776
R3-PM2:V <= 128425.29                0.007998
R4-PA5:IH > 115.38                   0.008534

[361 rows x 1 columns]
```

For Attack class:

```
                                 importance
feature
R3-PA2:VH > 113.97               -0.012837
-97.10 < R4-PA7:VH <= -35.66     -0.010994
-97.43 < R1-PA7:VH <= -35.85     -0.010306
R2-PA2:VH > 114.00               -0.009730
```

```
R3-PM2:V <= 128425.29          -0.006914
...                                  ...
R3-PA2:VH <= -93.75             0.008307
R3-PA7:VH <= -101.22            0.008676
R2-PM1:V > 130872.03            0.009242
R3:S > 0.00                     0.010717
R2-PA7:VH <= -101.20            0.014018

[380 rows x 1 columns]
```

For Natural class:

```
                        importance
feature
R2-PA5:IH <= -74.26     -0.003613
R4-PM3:V > 132484.78    -0.002835
R4-PM2:V > 132187.18    -0.002437
R2-PM7:V <= 128751.24   -0.002290
R2-PM1:V <= 128762.21   -0.002151
...                           ...
R1-PA1:VH > 71.28        0.003518
R2-PM7:V > 130857.40     0.003693
R2-PA5:IH > 63.30        0.004103
R3:F > 60.00             0.004873
R2:F > 60.00             0.005168

[331 rows x 1 columns]
```

For all classes:

```
                             importance
feature
R3-PM2:V <= 128425.29         -0.006715
R2-PA7:VH > 65.91             -0.006353
R3-PM5:I <= 330.70            -0.006009
-40.44 < R3-PA1:VH <= 65.76   -0.005808
R3-PA4:IH <= -65.22           -0.005764
```

```
...                                  ...
R2-PA3:VH <= -75.28             0.005588
R3-PA4:IH > 102.49              0.005723
R4-PA1:VH > 71.50               0.007481
R3-PM2:V > 130431.15            0.008501
R2-PM1:V > 130872.03            0.009070

[331 rows x 1 columns]
```

Bibliography

[1] T. Morris, “Industrial Control System (ICS) Cyber Attack Datasets - Tommy Morris.” https://sites.google.com/a/uah.edu/tommy-morris-uah/ics-data-sets.

[2] U. Adhikari, S. Pan, and T. Morris, “Power System Attack Datasets,” Apr. 2014.

[3] “Three-phase electric power,” *Wikipedia*, May 2020.

[4] J.L. Kirtley Jr., “Introduction To Symmetrical Components,” *6.061 Introduction to Power Systems*, vol. Class Notes Chapter 4.

[5] R. C. Borges Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, “Machine learning for power system disturbance and cyber-attack discrimination,” in *2014 7th International Symposium on Resilient Control Systems (ISRCS)*, (Denver, CO, USA), pp. 1–8, IEEE, Aug. 2014.

[6] “Appendix B - The WEKA workbench,” in *Data Mining (Fourth Edition)* (I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, eds.), pp. 553–571, Morgan Kaufmann, fourth edition ed., 2017.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] “Who is using scikit-learn? — scikit-learn 0.23.1 documentation.” https://scikit-learn.org/stable/testimonials/testimonials.html.

[9] “Sklearn.ensemble.RandomForestClassifier — scikit-learn 0.23.1 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[10] “Sklearn.svm.SVC — scikit-learn 0.23.1 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[11] "Sklearn.naive_bayes.GaussianNB — scikit-learn 0.23.1 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html.

[12] "RandomForest." https://weka.sourceforge.io/doc.dev/weka/classifiers/trees/RandomForest.html.

[13] "LibSVM." https://javadoc.scijava.org/Weka/weka/classifiers/functions/LibSVM.html.

[14] "NaiveBayes." https://weka.sourceforge.io/doc.dev/weka/classifiers/bayes/NaiveBayes.html.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144, 2016.

[16] Mikhail Korobov and Konstantin Lopuhin, "ELI5." https://github.com/TeamHG-Memex/eli5.

[17] B. Bengfort, R. Bilbro, N. Danielsen, L. Gray, K. McIntyre, P. Roman, Z. Poh, *et al.*, "Yellowbrick," 2018-11-14, 2018.

[18] Ando Saabas, "TreeInterpreter." https://github.com/andosa/treeinterpreter.

[19] Terence Parr and Prince Grover, "Dtreeviz." https://github.com/parrt/dtreeviz.