Project 2
Big Data
Master Degree in Computer Engineering
Polytechnic University of Bari

Biagio Montaruli - b.montaruli@studenti.poliba.it

21 February 2018

1 Introduction

Because of the exponential growth in the size of computer networks and the number of web applications, the ability to improve defenses against the significant increasing of the potential threats that can be caused by network attacks is becoming a necessary security requirement [2]. In this context, Machine Learning (ML) could be a very useful way in order to develop intelligent Intrusion Detection Systems (IDSs) and Intrusion Prevention Systems (IPSs) which are one of the most important defense tools against the sophisticated and ever-growing network attacks. One of the key element to develop successful ML project is to have comprehensive and valid datasets. However, in most cases, datasets related to information security are characterized by some issues: many such datasets cannot be shared due to the privacy issues and those that become available are heavily anonymized, do not reflect the current trends and lack of traffic variety and attack diversity. In addition, the need to have updated datasets is related to malware evolution and the continuous changes in attack strategies [2].
This report describes the implementation and performance evaluation of a Big Data project whose goal is to use Apache Spark and MLlib in order to perform network traffic analysis on two specific datasets, KDD99 and CICIDS2017.
The rest of this report is organized as follows. Section 2 describes the structure of the KDD99 and CICIDS2017 datasets. Section 3 provides an overview on Apache Spark framework and MLlib along with the main ML models used in this project. The implementation of the project in Spark is detailed in Section 4, instead Section 5 shows the results of performance evaluation of each dataset and provides a description of the different evaluation parameters (accuracy, precision, recall and f1-score). Finally, Section

6 draws the conclusions making a summary on the obtained results and a comparison between the two datasets.

2 Overview of KDD99 and CICIDS2017 datasets

This section provides a description of KDD99 and CICIDS2017 datasets. They contain samples representing network connections labeled as either normal/benign, or as an attack, and in this case the attack type is specified. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address and use some well defined application protocol.

2.1 KDD99

KDD99 dataset (University of California, Irvine 1998-99) is an updated version of the DARPA98 and it has been realized by processing raw TCP dump data for a local-area network (LAN) simulating a typical U.S. Air Force LAN. It contains different types of attacks such as Denial of Service (DoS), port scanning and buffer-overflow along with benign samples that have been merged together in a single comma-separated values (CSV) file consisting of 4898431 samples characterized by 42 features. The derived features are divided into several categories:

- **Time-based traffic features**: these features are subdivided into two classes:
  - *same host features*: they examine only the connections in the past two seconds that have the same destination host as the current connection. Some examples of features that belong to this class are `same_srv_rate` (% of connections to the same continuous service) and `srv_count` (number of connections to the same service as the current continuous connection in the past two seconds).
  - *same service features* examine only the connections in the past two seconds that have the same service as the current connection. The features that fall into this class are `srv_serror_rate` (% of connections that have "SYN" errors), `srv_rerror_rate` (% of connections that have "REJ" errors) and `srv_diff_host_rate` (% of connections to different continuous hosts).
- **Content features**: they are features that are related to the same connection and are extracted using domain knowledge considering a window of 100 connections to the same host. In addition, these features are related to a special type of attacks such as Denial of Service (DoS) and probing attacks that scan the hosts or ports using a much larger time interval than two seconds and involve many connections

to some host(s) in a very short period of time. Some examples of features that belong to this category are `num_failed_logins` (number of failed login attempts) and `num_compromised` (number of "compromised" conditions).

2.2 CICIDS2017

The CICIDS2017 dataset, developed by the Canadian Institute for Cybersecurity, consists of over 2 millions of samples characterized by 79 features. It is intended for network security and intrusion detection purposes in fact it covers a diverse set of attack scenarios (Botnet, Brute Force, Web, Infiltration and DoS/DDoS attacks) and, as described in [2] has been obtained through a comprehensive testbed based on eleven characteristics, namely Attack Diversity, Anonymity, Available Protocols, Complete Capture, Complete Interaction, Complete Network Configuration, Complete Traffic, Feature Set, Heterogeneity, Labelling, and Metadata that are considered critical for a comprehensive and valid IDS dataset. The CICIDS2017 dataset consists of different files representing captured network traffic during 5 days, from 03/07/2017 (Monday) to 07/07/2017 (Friday):

- `Monday-WorkingHours.pcap_ISCX.csv`: it contains only benign samples that represent normal human activities.
- `Tuesday-WorkingHours.pcap_ISCX.csv`: it contains both benign samples and those representing Brute Force attacks performed against a File Transfer Protocol (FTP) and Secure SHell (SSH) server using the Patator tool [1].
- `Wednesday-workingHours.pcap_ISCX.csv`: among the benign samples included in this CSV file, there are those related to DoS attack performed in the morning and others that took place in the afternoon representing different Heartbleed attacks [2].
- `Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv`: this CSV file contains samples related to Cross-site Scripting (XSS) [3] and Brute Force attacks against a web application along with benign samples.
- `Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv`: network traffic samples captured on Thursday July 6 afternoon have been captured and stored in this CSV file. In particular, it contains both benign network connections and those related to infiltration attacks performed using Metasploit tool [4].

[1] `https://github.com/lanjelot/patator`
[2] `https://en.wikipedia.org/wiki/Heartbleed`
[3] `https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`
[4] `https://www.metasploit.com`

- `Friday-WorkingHours-Morning.pcap_ISCX.csv`: it contains samples related to botnet attacks mixed together with samples representing normal human activity.
- `Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv`: this CSV file includes samples that represents Distributed Denial of service (DDoS) attacks, performed using Low Orbit Ion Canon (LOIC) tool [5], along with benign samples.
- `Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv`: this is the second CSV file containing network traffic samples captured on Friday July 6 afternoon and it contains both benign samples and those related to Portscan attacks.

3 Apache Spark and MLlib

This project has been implemented using Apache Spark and its Machine Learning (ML) library called MLlib. After describing the main features of Spark and MLlib, this section also provides an overview on the main ML concepts and a brief description of the ML algorithms used in this project. Apache Spark is a cluster computing framework for large-scale data processing designed in order to be fast and general-purpose. In fact, it supports a wide set of efficient types of computations including batch applications, iterative algorithms, interactive queries and streaming that can be combined into pipelines [1]. Among all the main Spark components, since the goal of this project is the classification of network traffic samples, we decided to use MLlib which is based on Spark Core, the main Spark component that provides the low-level APIs to define and manage resilient distributed datasets (RDDs), which are Spark's main programming abstraction.

3.1 MLlib

MLlib is a package, built on and included in Spark, that takes advantage of the main features of Spark in order to deploy ML projects based on large datasets distributed on cluster of nodes. It provides operations (methods and classes) such as:

- *ML Algorithms*: common learning algorithms such as classification, regression, clustering, and collaborative filtering;
- *Featurization*: feature extraction, transformation, dimensionality reduction, and selection.

[5] `https://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon`

- *Pipelines*: tools for constructing, evaluating, and tuning ML Pipelines on large scale
- *Persistence*: saving and load algorithms, models, and Pipelines
- *Utilities*: linear algebra, basic statistics, etc.

MLlib actually consists of two packages that leverage different core data structures. The package org.apache.spark.ml is the primary Machine Learning package for Spark and provides DataFrame-based APIs. We will use this package in order to develop this project. On the other hand, Spark also provides the RDD-based APIs in the spark.mllib package.

Figure 1 illustrates the overall workflow that an user should follow when developing machine learning models in Spark along with the fundamental "structural" types of MLlib: Transformers, Estimators, Evaluators and Pipelines.

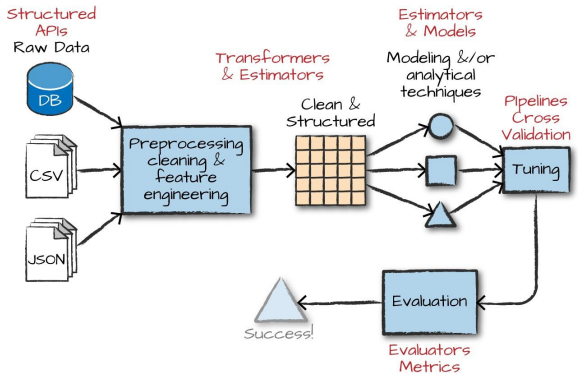


Figure 1: Workflow to develop a ML project in Spark.

Transformers are functions that takes a DataFrame as input and produce a new DataFrame as output. They are primarily used in preprocessing and feature engineering. An example of a Transformer is `StringIndexer` that converts string categorical variables into numerical values that can be used in MLlib.
An Estimator can be seen as a kind of Transformer that is initialized with data. It generally requires two passes over our data: the initial pass generates the initialization values and the second actually applies the generated function over the data. In the Spark's view, algorithms that allow users to train a model from data are also referred to as Estimators. So, all the ML algorithm can be considered as Estimators in Spark.
An Evaluator, as its name suggests, can be used to evaluate some performance metrics over a test set. We can also use an Evaluator to select the best model from the ones we tested.

Finally, a Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

3.2 Machine Learning basics

Machine Learning is one of the most popular trend in the last few years. It can be defined as a set of tools and methods that attempt to infer patterns and extract knowledge from data. ML problems can be subdivided into 3 main classes:

- **Supervised Learning**: Its main goal is to use a labeled dataset in order to train a model that we can use to make predictions on new unseen data. Supervised ML algorithms can be divided into 2 categories:
  - *Classification algorithms*: in classification we train an algorithm to predict a dependent variable that is categorical (it belongs to a discrete, finite set of values). In addition, we can have two main types of classification problems: binary classification, if each sample can belongs to one of two classes, or multiclass classification when we classify samples into more than just two categories.
  - *Regression algorithms*: unlike classification, in regression our dependent variable is continuous (a Real variable).
- **Unsupervised Learning**: It consists of trying to find patterns or discover the underlying structure in a given dataset. The main difference from supervised learning is the absence of a dependent variable to predict. So, in unsupervised learning, data samples are not labeled but we try to find similarities/patterns between samples in the dataset rather than making predictions. The main algorithms used in unsupervised learning are:
  - Clustering Algorithms (K-Means, Hierarchical, ...)
  - Dimensionality Reduction Techniques
  - Topic Modeling
  - Association Rule Mining
- **Reinforcement Learning**: It is a field of ML concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Reinforcement learning has many applications in different fields such as control theory, recommender systems, robotics and human computer interaction.

The problem to classify network attacks can be represented a multiclass classification problem and in order to classify the samples of KDD99 and CICIDS2017 datasets we have chosen the following classification algorithms:

- **Multinomial Logistic Regression**: It is a probabilistic discriminative model for classification that estimates directly the probability that a sample x belongs to a class Y_k by minimizing the maximum likelihood. In particular, the multinomial logistic regression algorithm produces a matrix of dimension $K \times J$ where K is the number of outcome classes and J is the number of features. The conditional probabilities of the outcome classes $k \in 1, 2, \ldots, K$ are modeled using the softmax function:

$$P(Y = k|\mathbf{X}, \boldsymbol{\beta}_k, \beta_{0k}) = \frac{e^{\boldsymbol{\beta}_k \cdot \mathbf{X} + \beta_{0k}}}{\sum_{k'=0}^{K-1} e^{\boldsymbol{\beta}_{k'} \cdot \mathbf{X} + \beta_{0k'}}}$$

  In order to compute the weights w_i, we minimize the weighted negative log-likelihood using a multinomial response model with elastic-net penalty to control for overfitting:

$$\min_{\beta, \beta_0} - \left[\sum_{i=1}^{L} w_i \cdot \log P(Y = y_i|\mathbf{x}_i) \right] + \lambda \left[\frac{1}{2} (1 - \alpha) \, ||\boldsymbol{\beta}||_2^2 + \alpha ||\boldsymbol{\beta}||_1 \right]$$

- **Decision Tree**: a Decision Tree (Figure 2) is a ML algorithm having a tree structure: internal nodes are represented by attributes (features), each branch denotes a possible value assumed by an attribute and leaf nodes denotes classification values (labels). The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision Trees can be used for both classification and regression since they support categorical as well as numerical attributes. One of the most popular Decision Tree algorithm is called ID3 (Iterative Dichotomiser 3). It is an iterative algorithm that creates a multiway tree by choosing at each step the attribute that maximizes the information gain, which measures how well a given attribute separates the training examples according to their target classification. In addition, a pruning step is usually applied to improve the ability of the tree to generalize to unseen data.

- **Random Forests**: Random Forests (Figure 3) is a ML method for combining decision trees by simply training different decision trees, combining their votes and then predicting the class with the majority of votes or taking the average in case of regression. In particular, each of these decision trees is trained by using a subset of the initial dataset as well as a subset of features. This approach is called *Bagging* and allows to prevent overfitting in fact, the prediction power of random forests is relatively higher compared to a single decision tree because the group's performance exceeds any individual.

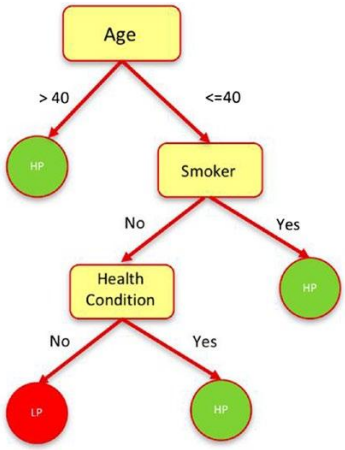


Figure 2: Example of a Decision Tree.

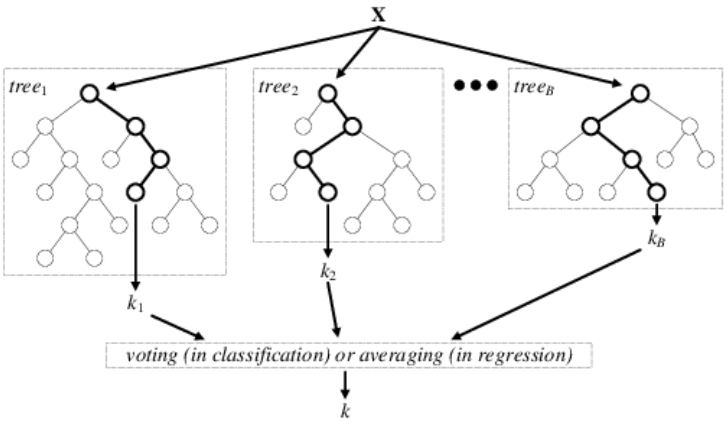


Figure 3: Example of Random Forests.

- **Naive Bayes**: Naive Bayes classifiers are a family of simple probabilistic classifiers based on the Bayes theorem with strong (naive) independence assumptions between every pair of features. This model is trained by computing the conditional probability distribution of each feature given each label and, for prediction, it applies Bayes theorem to compute the conditional probability distribution of each label given an observation. Naive Bayes classifiers are commonly used in text or document classification and, in this case, each observation (sample) is a document and each feature represents a term. A feature's value is the frequency of the term (in Multinomial Naive Bayes) or boolean value indicating whether or not the term was found in the document (in Bernoulli Naive Bayes).

4 Implementation

This section describes the design and the key ideas of the Spark driver programs implemented in Python in order to organize and analyze the samples of KDD99 and CICIDS2017 datasets.

4.1 Implementation for KDD99 dataset

It can be found in the `network_attacks_classifier_kdd99.py` Python script. First of all in lines 1-8 we import all the necessary modules and classes such as `LogisticRegression`, `DecisionTreeClassifier`, `NaiveBayes` and `RandomForestClassifier` to classify the samples, `MulticlassClassificationEvaluator` to evaluate the performance of the classifiers (models), but also the `StringIndexer`, used to encode a string column of labels to a column of label indices, and `VectorAssembler` which combines a given list of columns into a single vector column. The file containing the full KDD99 dataset is called `kddcup.data.corrected`. It is a CSV file and because it does not contain the list of attributes in the header, we define it in lines 15-23 and, after importing the dataset in line 25 (we also specify that the dataset does not provides the list of attributes in the header with `header=False`), we add the list of features to our `dataset` DataFrame with the method `toDF()`.
In lines 33-37 we create a Pipeline that uses multiple `StringIndexer` Transformers to encode the columns `protocol_type`, `service`, `flag` and `label` of type String into columns of indices of type Double (numerical columns) because `LogisticRegression` and `NaiveBayes` classifiers need to be trained on numerical values. Then, because of we need to assemble all of the input columns into a single vector that would act as the input feature for the classifiers, we use Spark's `VectorAssembler` in order to create a feature vector for each sample by considering only the numerical columns (lines 45-46).
In line 49 we select the columns `features` and `label_num` to create a new DataFrame and then we randomly split it into training set (`train_set`) and test set (`test_set`) so that the training set contains the 75% of samples in the dataset.
At this point we instantiate the classifiers (lines 57-67) and, for each model, we fit it using the training set (line 79), we make predictions on the test set (lines 82) and finally we evaluate and print in output the performance (lines 86-91) by considering accuracy, weighted precision, weighted recall and F1-score as performance metrics.

4.2 Implementation for CICIDS2017 dataset

The Spark program related to CICIDS2017 dataset has been implemented through the Python script `network_attacks_classifier_cicids17.py`.

As for the KDD99 dataset, we firstly import all the necessary modules and classes of MLlib to classify the samples (`LogisticRegression`, `NaiveBayes`, `DecisionTreeClassifier` and `RandomForestClassifier`), to evaluate the models' performance (`MulticlassClassificationEvaluator`) and to perform feature engineering (`StringIndexer` and `VectorAssembler` Transformers). As described in the previous section, the CICIDS2017 dataset consists of multiple CSV files and, unlike the KDD99 dataset, each CSV file contains the list of attributes in the header. In lines 19-20 we import the samples from all the CSV files and then we print some info about the dataset (lines 22-25). Instead in lines 30-35 we remove from the dataset all the samples that have negative values in the attributes `Flow Duration`, `Init_Win_bytes_forward`, `Init_Win_bytes_backward`, `Flow IAT Min`, `Fwd IAT Min` and `Fwd IAT Max` because the Naive Bayes classifier does not support negative values.
Then in lines 40-42 we use Spark's `VectorAssembler` in order to create a feature vector for each sample that would act as the input feature for the classifiers and, with lines 44-45, we encode the column `Label` that represent the target class into a numerical column (`Label_Idx`).
At this point the program selects the columns `features` and `Label_Idx` to create a new DataFrame (line 53) and then we randomly split it into training set (`train_set`) and test set (`test_set`) so that the training set contains the 75% of samples in the dataset and the remaining 25% of samples is assigned to the test set (line 56).
Now it's the turn to evaluate the performance with the different classifiers: we instantiate the classifiers (lines 61-71), we fit each model using the training set (line 83), then we make predictions using the test set (lines 86) and finally we evaluate and print in output the performance (lines 90-95) by considering accuracy, weighted precision, weighted recall and F1-score as performance metrics.
In addition, the Python script also plots, for each classifier, the confusion matrix (lines 98-111) that contains information about actual and predicted classifications done by the classifier.

5 Performance evaluation

For both the KDD99 and CICIDS2017 we measure how the four classifiers (Logistic Regression, Decision Trees, Random Forests and Multinomial Naive Bayes) are able to classify samples in the test set through four performance metrics: accuracy, weighted precision, weighted recall and F1-score. In particular, denoting the true positives (hits) as TP, true negatives (correct rejections) as TN, false positives (false alarms) as FP and false negatives (miss) as FN, in case of multiclass classification the adopted metrics are defined as:

- **Accuracy**: $ACC = \frac{TP+TN}{TP+TN+FP+FN} = \frac{1}{N}\sum_{i=0}^{N-1} \hat{\delta}\left(\hat{\mathbf{y}}_i - \mathbf{y}_i\right)$.
  Accuracy is the most basic metric for evaluating any classifier and is defined as the ratio between the number of correct classifications and the total number of samples. However, this is not the only indicator of the performance of the model that should be consider in a classification problem due to dependency on how the target classes are balanced and the presence of outliers.

- **Weighted Precision**: $PPV_w = \frac{1}{N}\sum_{\ell \in L} PPV(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$.
  It is obtained by computing the standard precision for each label ($PPV(\ell) = \frac{TP}{TP+FP}$), and finding their average weighted by support (the number of true instances for each label). This metrics represent the ability to avoid false positives.

- **Weighted Recall**: $TPR_w = \frac{1}{N}\sum_{\ell \in L} TPR(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$.
  It is obtained by computing the standard recall for each label ($TPR(\ell) = \frac{TP}{TP+FN}$), and finding their average weighted by support. This metrics represent the ability to avoid false negatives.

- **F1-score**: $F1_w = \frac{1}{N}\sum_{\ell \in L} F1(\ell) \cdot \sum_{i=0}^{N-1} \hat{\delta}(\mathbf{y}_i - \ell)$.
  It is obtained by computing the F1-score for each label ($F1 = 2 \cdot \frac{PPV \cdot TPR}{PPV+TPR}$), and finding their average weighted by support. The F1-score can be interpreted as a weighted average of the precision and recall.

5.1 Performance evaluation of KDD99

Results of performance evaluation of KDD99 dataset are shown in Figure 4 both in graphical and tabular way. As we can see, the best classifiers are Decision Tree and Random Forests which are characterized by an average score of 100% for all the evaluation metrics. On the other hands, also Naive Bayes and Logistic Regression have obtained quite good results. In fact, Logistic Regression provides, for all the metrics, results greater than 90%; instead regarding Naive Bayes, even if it has obtained the lowest results in accuracy, recall and F1-score, it still provides a very high precision (99%).

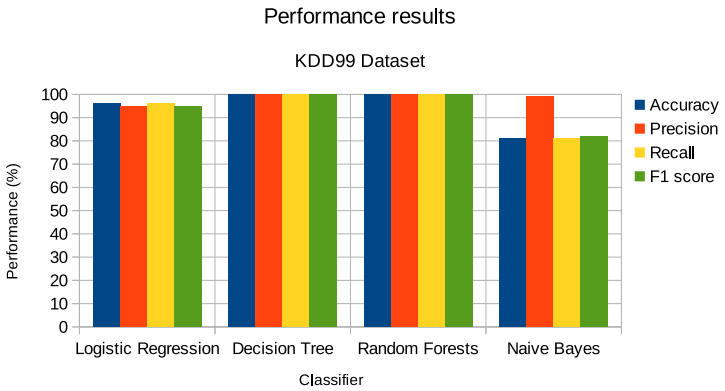
5.2 Performance evaluation of CICIDS2017

From the obtained results of performance evaluation detailed in Figure 5 and from the confusion matrices shown in Figures 6, 7, 8 and 9, we can see that, as for the KDD99 dataset, the best performance are achieved by Decision Tree and Random Forests classifiers which guarantee high performance scores for all the metrics. As for the other evaluation metrics, Logistic Regression is characterized by quite good results except for the weighted

precision which is rather low (59%), instead regarding Naive Bayes classifier, even though it still remains the worst classifier, it outperforms Logistic Regression with respect to the weighted precision (88%).

KDD99 - Performance evaluation (%)				
	Metric			
	Accuracy	Weighted Precision	Weighted Recall	F1-score
Logistic Regression	96	95	96	95
Decision Trees	100	100	100	100
Random Forests	100	100	100	100
Naive Bayes	81	99	81	82

(a) Tabular comparison

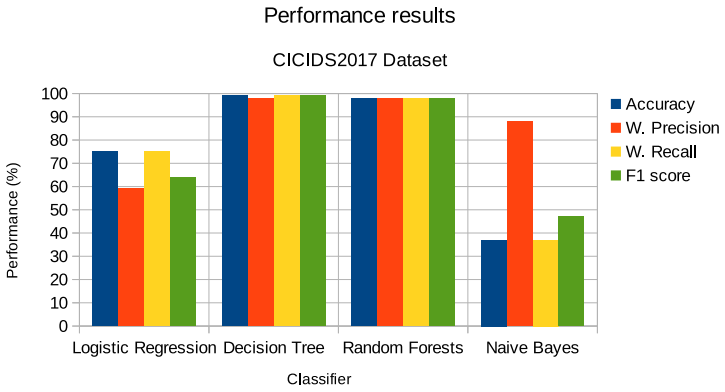


(b) Graphical comparison

Figure 4: Performance evaluation of KDD99 dataset

CICIDS2017 - Performance evaluation (%)				
	Metric			
	Accuracy	Weighted Precision	Weighted Recall	F1-score
Logistic Regression	75	59	75	64
Decision Trees	99	98	99	99
Random Forests	98	98	98	98
Naive Bayes	37	88	37	47

(a) Tabular comparison



(b) Graphical comparison

Figure 5: Performance evaluation of CICIDS2017 dataset

6 Conclusions

This report has described a new approach to network traffic analysis based on Apache Spark, one of the most promising Big Data technologies, and MLlib, Spark's scalable machine learning library. Starting from KDD99

and CICIDS2017 datasets, which contain over 2 millions of samples and more than 40 features, we designed two Python driver programs to classify the network traffic samples through four popular ML classifiers (Logistic Regression, Decision Tree, Random Forests and Naive Bayes) and we evaluated the performance of each classifier using four metrics, accuracy, weightes precision, weighted recall and F1-score. Considering the obtained results described in Section 5, we can state that, for both the datasets, Random Forests and Decision Tree outperform the other classifiers and provide excellent results. In addition, comparing the two datasets together, we have obtained better results for KDD99 dataset. This is because KDD99 provides more samples and less features (4898431 samples and 42 features of KDD99 vs 2830743 samples, which become 755774 after removing those that contain negative features that are not supported by Naive Bayes classifier, and 79 features of CICIDS2017), thus improving the generalization power and making more accurate the training and evaluation of the models.

References

[1] H. Karau et al. *Learning Spark*. O'Reilly Media, Inc, 2015.

[2] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,* INSTICC. SciTePress, 2018, pp. 108–116.

[3] M. Zaharia and B. Chambers. *Spark: The definitive guide*. O'Reilly Media, Inc, 2018.

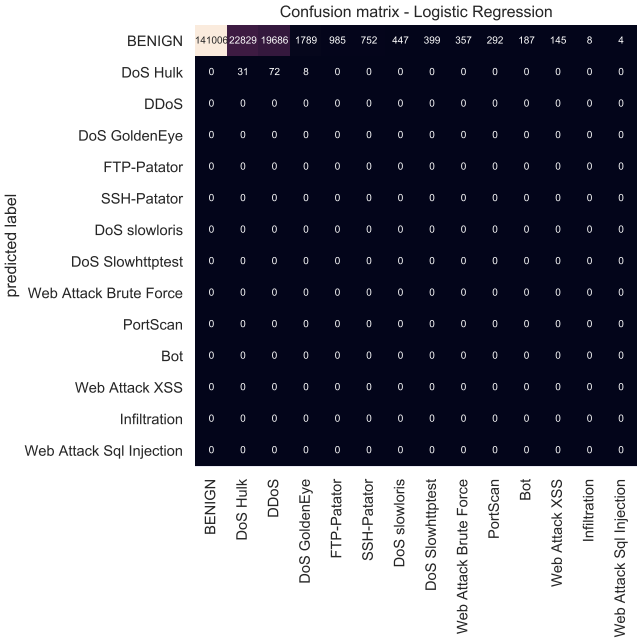


Figure 6: Confusion Matrix - Logistic Regression

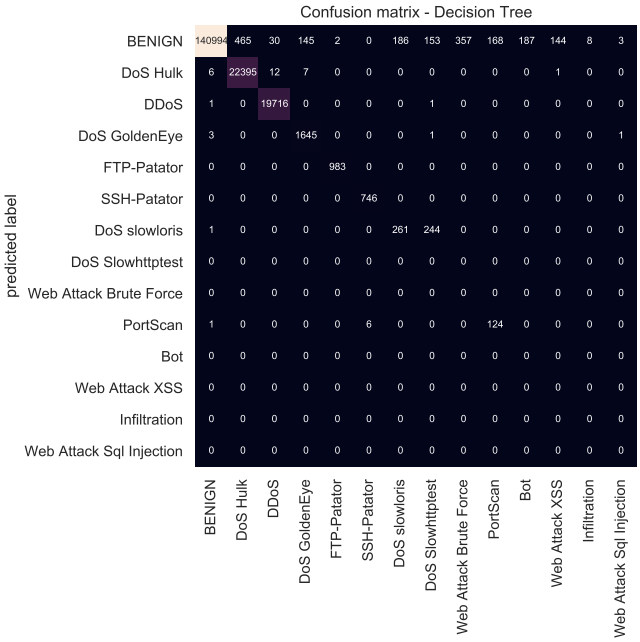


Figure 7: Confusion Matrix - Decision Tree

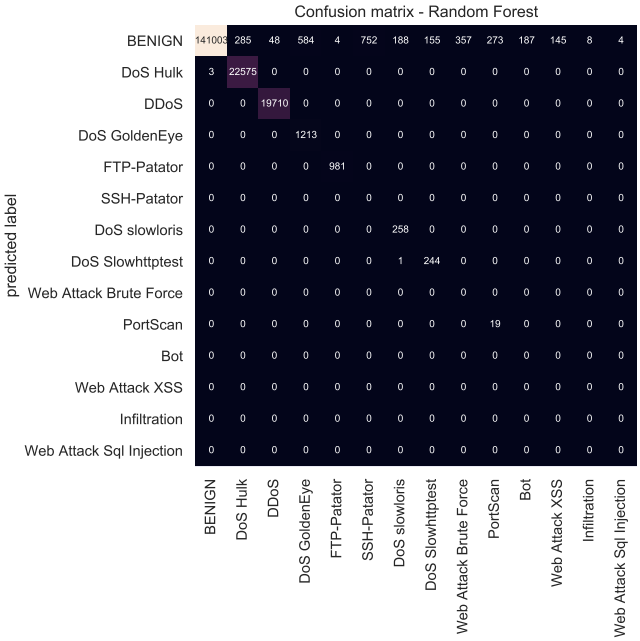


Figure 8: Confusion Matrix - Random Forests

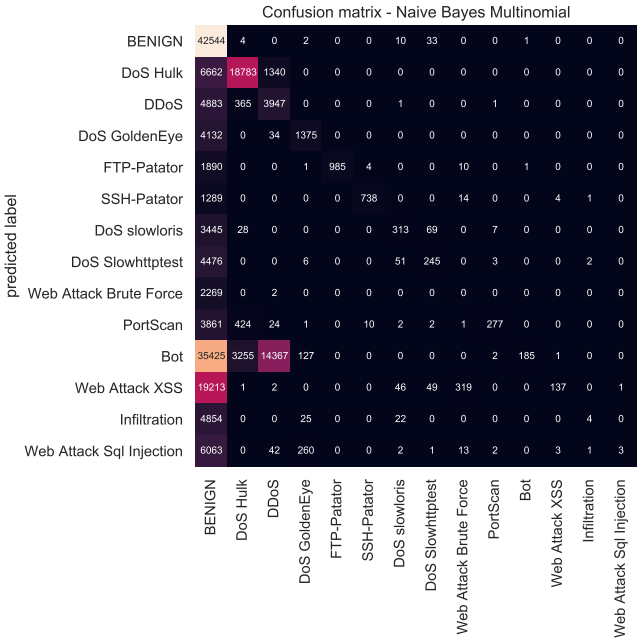


Figure 9: Confusion Matrix - Naive Bayes