Project 2: Network Traffic Analysis with Spark and MLlib

Project goals

Perform network traffic analysis and classification of network attacks using Apache Spark and MLlib

Target datasets: KDD99 and CICIDS2017

Use different Supervised Learning models to classify network samples:

- Logistic Regression
- Decision Tree
- Random Forests
- Multinomial Naive Bayes

Evaluate performance both in tabular and graphical way using four performance metrics:

- Accuracy
- Weighted Precision
- Weighted Recall
- F1-score

Datasets

KDD99:

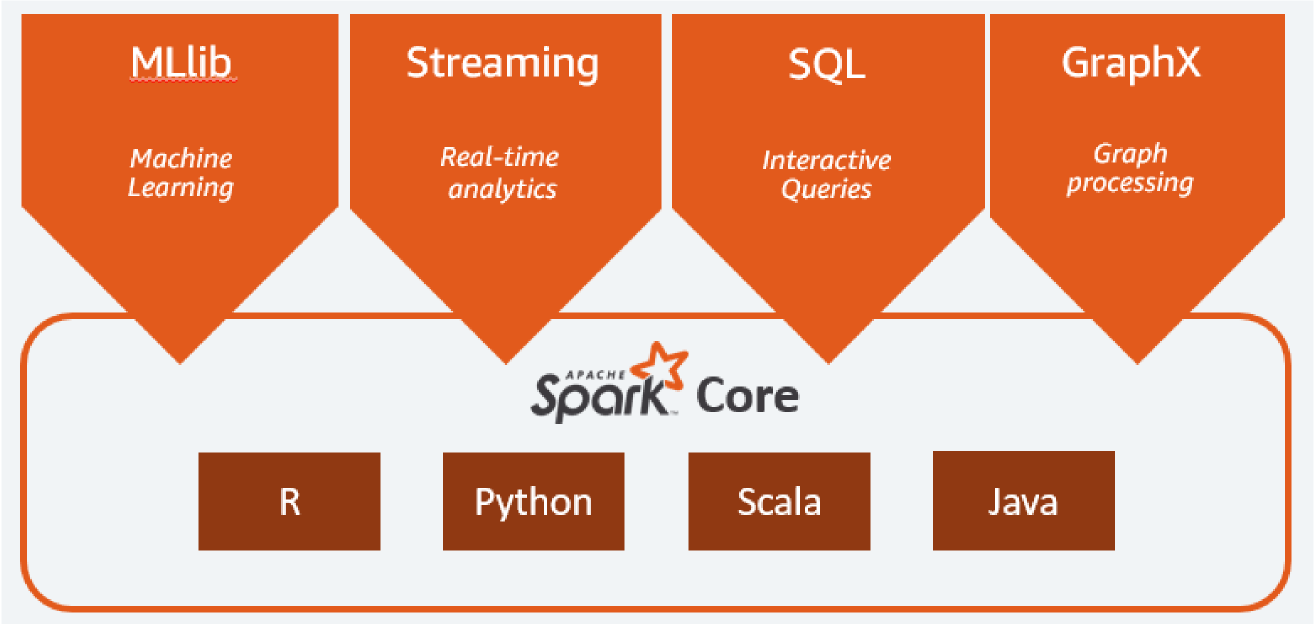
- Developed by University of California, Irvine 1998-99
- It consists of a single CSV file containing different types of network attacks (DoS, port scanning, buffer-overflow, …) along with benign samples
- Dataset sizes: 4898431 samples characterized by 42 features
- The derived features are divided into two main categories:
  - Content-based features
  - Time-based traffic features

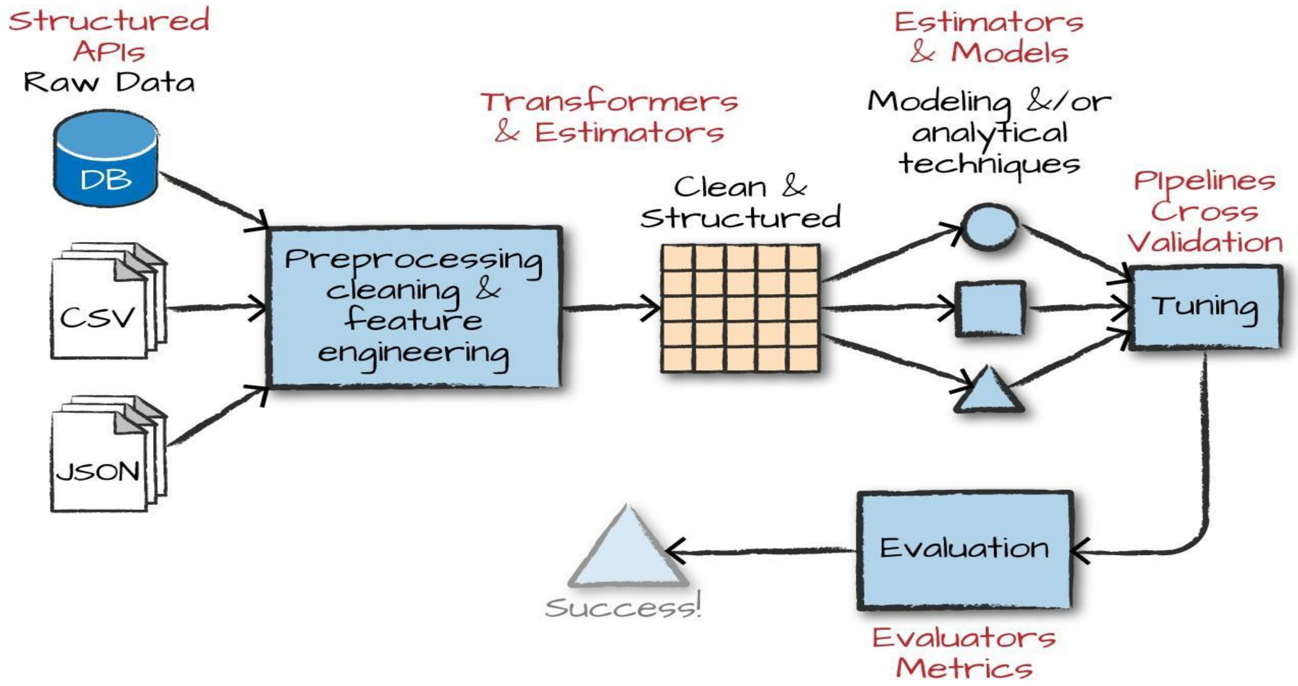
CICIDS2017:

- Realized by the Canadian Institute for Cybersecurity
- Dataset sizes: 2830743 samples and 79 features
- It consists of different files representing captured network traffic during 5 days, from 03/07/2017 (Monday) to 07/07/2017 (Friday)
- The implemented attacks include Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, Botnet and DDoS

Spark and MLlib

- Apache Spark:
  - It is a cluster computing framework for large-scale data processing designed to be fast and general-purpose.
  - Characterized by different SW components: Spark Streaming, MLlib, SparkSQL and GraphX
- MLlib:
  - It is a package, built on and included in Spark, for developing and deploying ML projects on a distributed cluster of nodes.
  - It provides a wide range of operations to manage and analyze large datasets:
    - ML Algorithms: classification, clustering, regression and collaborative filtering
    - Feature Engineering: feature extraction, transformation, dimensionality reduction
    - It provide a uniform set of high-level APIs built on top of DataFrames for constructing, evaluating, and tuning ML Pipelines using the fundamental structural types of MLlib: Transformers, Estimators, Evaluators and Pipelines





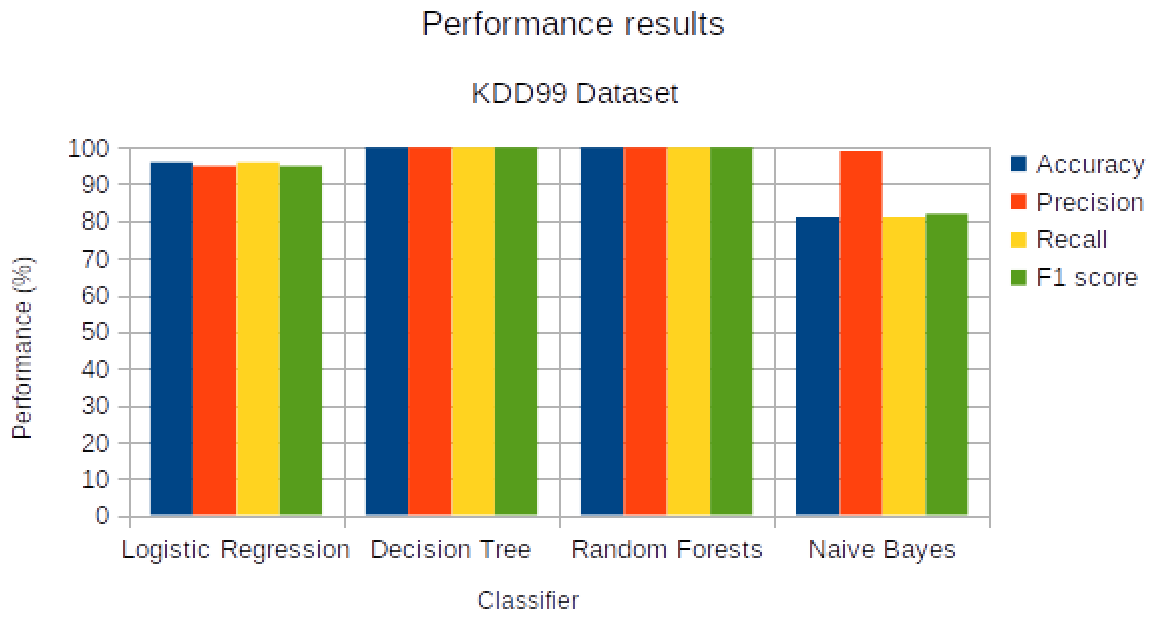
Project Implementation for KDD99

- The Spark driver program has been implemented through the *network_attacks_classifier_kdd99.py* Python script
- Import all the necessary modules and classes of MLlib:
    - to classify the samples (*LogisticRegression, NaiveBayes, DecisionTreeClassifier* and *RandomForestClassifier*)
    - to evaluate the models' performance (*MulticlassClassificationEvaluator*)
    - to perform feature engineering (*StringIndexer* and *VectorAssembler*)
- Load the dataset (kddcup.data.corrected - CSV file withouth the list of attributes in the header)
- Create a Pipeline that uses multiple *StringIndexer* to encode the String columns *protocol_type, service, flag* and *label* into numerical columns
- Use Spark's *VectorAssembler* in order to create a feature vector for each sample (needed for training the classiifers)
- Create the final DataFrame having the columns *features* and *label_num*
- Split the dataset into Training Set (75 %) and Test Set (25 %)
- Train the four classifiers using the Training Set
- Make predictions on the Test Set
- Evaluate performace and print classification results in output

Project Implementation for CICIDS2017

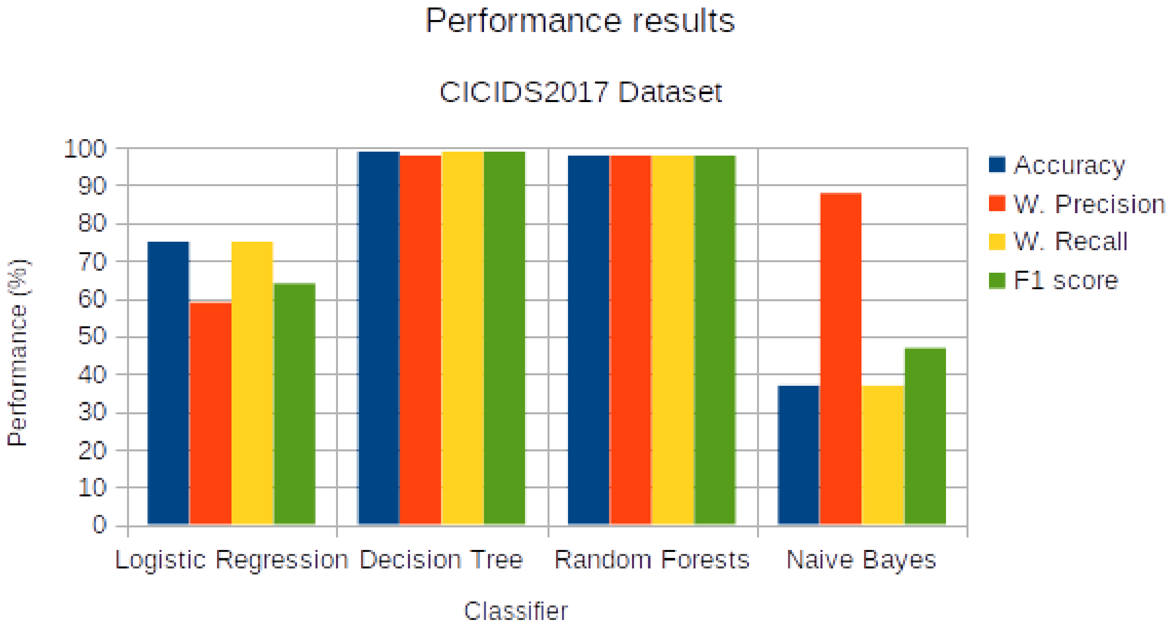
- The Spark driver program has been implemented through the *network_attacks_classifier_cicids17.py* Python script
- Import all the necessary modules and classes of MLlib:
  - to classify the samples (*LogisticRegression, NaiveBayes, DecisionTreeClassifier* and *RandomForestClassifier*)
  - to evaluate the models' performance (*MulticlassClassificationEvaluator*)
  - to perform feature engineering (*StringIndexer* and *VectorAssembler*)
- Load the CSV files that make up the dataset
- 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* (the Naive Bayes classifier does not support negative values).
- Use Spark's *VectorAssembler* in order to create a feature vector for each sample (needed for training the classiifers)
- Encode the column *Label* that represent the target class into a numerical column (*Label_Idx*) using *StringIndexer* Transformer
- Create the final DataFrame having the columns *features* and *label_num* and split it into Training Set (75 %) and Test Set (25 %)
- Train the four classifiers using the Training Set
- Make predictions on the Test Set
- Evaluate performace, print classification results in output and plot, for each classifier, the confusion matrix

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



Performance Evaluation - KDD99

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



Performance Evaluation - CICIDS2017

Conclusion

- For both the datasets, Random Forests and Decision Tree outperform the other classifiers and provide excellent results in all the performance metrics.
- Quite good results have been also obtained by Logistic Regression especially with the KDD99 dataset. The lowest results have been achieved in the weighted precision metric.
- The lowest performance have been achieved by Naïve Bayes classifier. Among all the performance, precision is the metrics in which it provides the best results.
- Comparing the two datasets together, we have obtained better results for KDD99 dataset: this is because KDD99 provides more samples and less features, thus improving the generalization power and making more accurate the training and evaluation of the models.