POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering:
Technologies for eMobility

Master's Degree Thesis

Machine learning for performance forecast based on CPU/Memory heap data flow and backend usage for anomaly detection

Supervisors
Prof. Elena Maria BARALIS
Ing. Luca RAZZI

Candidate
Gabriele GAROFALO

October 2022

This thesis work deals with the development of a machine learning algorithm able to diagnose the applications state, carried out during my work experience in Sogei s.p.a.

Sogei s.p.a.

Sogei s.p.a. is an italian company operating in the ICT field. It is 100% controlled by the Ministry of Economy and Finance of which it is an in-house company. It carries out IT consultancy services for the public administration, in particular for the Ministry of Economy and Finance and for tax agencies on the basis of multi-year service contracts. Its activities are:

- Development and management of the tax information system, or the tax registry on behalf of the Ministry of Economy and Finance
- Development and management of the public accounting information system for the State General Accounting Office, of the information and accounting system of public debt
- Development and management of the public gaming information system on behalf of the Customs and Monopolies Agency
- Public expenditure monitoring, in particular health expenditure
- Software development and management for tax agencies (Revenue, Customs and Monopolies, State Property, eg. DOCFA)
- Supplier selection, production monitoring and shipment of the health card on behalf of the Revenue Agency
- Creation of the national registry of the resident population (ANPR) in collaboration with the Ministry of the Interior
- Creation of public invoice and payment services
- Creation and management of the DGC national platform for the issue and control of the COVID-19 Green Certification commonly known as the vaccine Green Pass

Our mission, as a strategic partner of the Economic and Financial Administration, is to contribute to the modernization of the country, actively participating in the digital transformation process of the Public Administration.
I belong to the organizational unit called **Application Performance Management** (APM). There are several applications running on virtual machines in Sogei,

and the task of the APM is to monitor and look over the state of the whole system and to cooperate with other units for debugging, benchmarks and many other activities. This thesis work was born with the goal to simplify the application state recognition using machine learning techniques, reducing the number of repetitive actions that are necessary to monitor effectively the system.

"Desidero ringraziare, innanzitutto, la professoressa Baralis e il mio tutor tecnico ingegnere Luca Razzi, perché senza il loro supporto e aiuto non sarebbe stato possibile realizzare questo lavoro. Un ringraziamento va anche ai miei colleghi e superiori in Sogei, in particolare a Sebastiano Luciano e Giuseppe Spoto, che mi hanno accolto nel migliore dei modi e permesso di crescere sotto molti punti di vista. Un grandissimo grazie va anche ai miei genitori, che mi sono stati vicino e che mi hanno sempre supportato, dando anche consigli, senza essere però vincolanti. Infine, vorrei ringraziare tutti gli amici e colleghi universitari che mi sono stati particolarmente vicini in questi anni, che mi hanno da sempre spronato e fatto andare avanti, anche nelle situazioni più avverse."

Table of Contents

List of Figures

Acronyms

AI

Artificial Intelligence

APM

Application Performance Management

ART

Average Response Time

AUC

Area Under the ROC Curve

BGSS

Between Group Sum of Squares

CH

Calinski Harabasz

CNN

Convolutional Neural Network

CPU

Central Processing Unit

CSV

Comma Separated Values

EVD

Eigen Value Decomposition

FN

False Negative

FNN

Feed-forward Neural Network

FP

False Positive

FPR

False Positive Rate

GAN

Generative Adversarial Nets

KNN

K-Nearest Neighborghs

MAP

Maximum A Posteriori

ML

Machine Learning

NB

Naive Bayes

PCA

Principal Component Analysis

ReLU

Rectified Linear Unit

ROC

Receiver Operating Characteristic

RPI

Responses Per Interval

RNN

Recurrent Neural Network

SVD

Singular Value Decomposition

TN

True Negative

TP

True Positive

TPR

True Positive Rate or Recall

WGSS

Within Group Sum of Squares

Chapter 1

Introduction

1.1 Bayes Theorem

To introduce properly the arguments present in this paper, it is necessary to introduce the *Bayes Theorem.*
The Bayes Theorem provides a principled way for calculating a conditional probability. Although it is a powerful tool in the field of probability, Bayes Theorem is also widely used in the field of machine learning, i.e. in developing models for classification predictive modeling problems such as the Bayes Optimal Classifier and Naive Bayes.
It is possible to distinguish between marginal, joint and conditional probability:

- **Marginal Probability**: Probability of an event irrespective of the outcomes of other random variables, e.g. $P(A)$
- **Joint Probability**: Probability of two (or more) simultaneous events, $P(A, B)$
- **Conditional Probability**: Probability of one (or more) event given the occurrence of another event, $P(A|B)$.

The Bayes Formula is usually expressed as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

It is also possible to represent the formula in an alternative way [1], that is:

$$P(B) = P(B|A) \cdot P(A) + P(B|notA) \cdot P(notA)$$

1.2 Machine Learning

Machine Learning (ML) is a subset of artificial intelligence (AI) that is concerned with creating systems that learn or improve performance based on the data they use. With artificial intelligence, which is however a generic term, are intended systems or machines that imitate human intelligence. Defining in a simple way the characteristics and applications of machine learning is not always possible, given that this branch is very vast and provides different methods, techniques and tools to be implemented. In addition, the different algorithm learning and development techniques give rise to as many possibilities of use that broaden the field of application of machine learning, making a specific definition difficult. However, it can be said that machine learning is about different mechanisms that allow an artificial machine to learn and expand its capabilities and performance over time. The machine, consequently, will be able to learn to fulfill certain assignments by rising, through training or experience, its abilities, responses and capabilities. At the basis of machine learning there are a series of different algorithms which, starting from primitive notions, will be able to make a specific decision rather than another or carry out actions learned over time.
The two main types of machine learning algorithms currently used are: **supervised learning** and **unsupervised learning**. They are categorized according how each algorithm learns the data to make predictions.

Chapter 2

Types of learning

2.1 Supervised Learning

Supervised learning is a type of machine learning in which a machine learns from known datasets (set of training examples), and then predicts the output. The dataset is composed of several features and a label for each instance. A supervised learning agent needs to find out the function that matches a given sample set. [2] Supervised learning further can be classified into two categories of algorithms: **classification** and **regression**.

2.1.1 Classification and Regression

The **classification** is a very common type of machine learning, and it deals with categorizing several instances in one of the predefined categories/classes. It is important to underline that target functions are discrete. This means that the class attribute, whose values should be determined, is a categorical attribute, that is to say that predicts categorical class labels based on the training set and the values (class labels). Every object is classified into one of the classes with certain accuracy. The task is that on the characteristics of objects whose classification is known in advance, make a model by which will be performed classification of new objects. In the problem of classification, the number of classes is known in advance and limited. There are several classification algorithms, each with their own pros and cons. In general this is the kind of learning that works best with supervised learning.
The **linear regression** problem is instead a mathematical approach used to perform predictive analysis with continuous/real or mathematical variables projections. It is a mathematical test used for evaluating and quantifying the relationship between the considered variables. Regression analyses are usually used for forecasting and prediction, in which their application has major overlaps with the area of machine

learning. Second, regression analysis can be used in some cases to determine causal relations between the independent and dependent variables. Importantly, regressions alone show only relations between a dependent variable and a fixed dataset collection of different variables. [3]

2.1.2 Naive Bayes

In machine learning, Naive Bayes (NB) classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem under strong (naive) conditional independence assumptions among the features. Using the naive conditional independence assumption that $P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y)$ since $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$\begin{aligned} P(y \mid x_1, \ldots, x_n) &\propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \\ &\Downarrow \\ \hat{y} &= \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y), \end{aligned} \tag{2.1}$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set. The assumptions regarding the distribution of $P(x_i \mid y)$ mainly influence the differences between the naive Bayes classifiers.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.

2.1.3 Linear Regression

In machine learning, linear regression is a technique of classifying the examples of a dataset (training set) to allow the machine to automatically learn a decision model. The algorithm assigns labels (categories) to instances using a continuous function. Each row of the dataset is an example consisting of:

- **The features (X)**, or attributes, which are the predictive variables that describe a category.
- **The result (Y)**, which is the target variable that tells the machine the correct result if the instance belonged to the Z label. It is the data that instructs the machine to decide in the right way

The machine learning algorithm must find a relationship between the variables X and Y through regression $y = (x)$. The final result is a straight line that minimizes the distance between the N examples in the training which belong to the same category. Once found, the classification function can be used to evaluate instances other than the training set. If the (x, y) coordinates of an instance approach the regression function f (x), the instance is classified with the label Z.

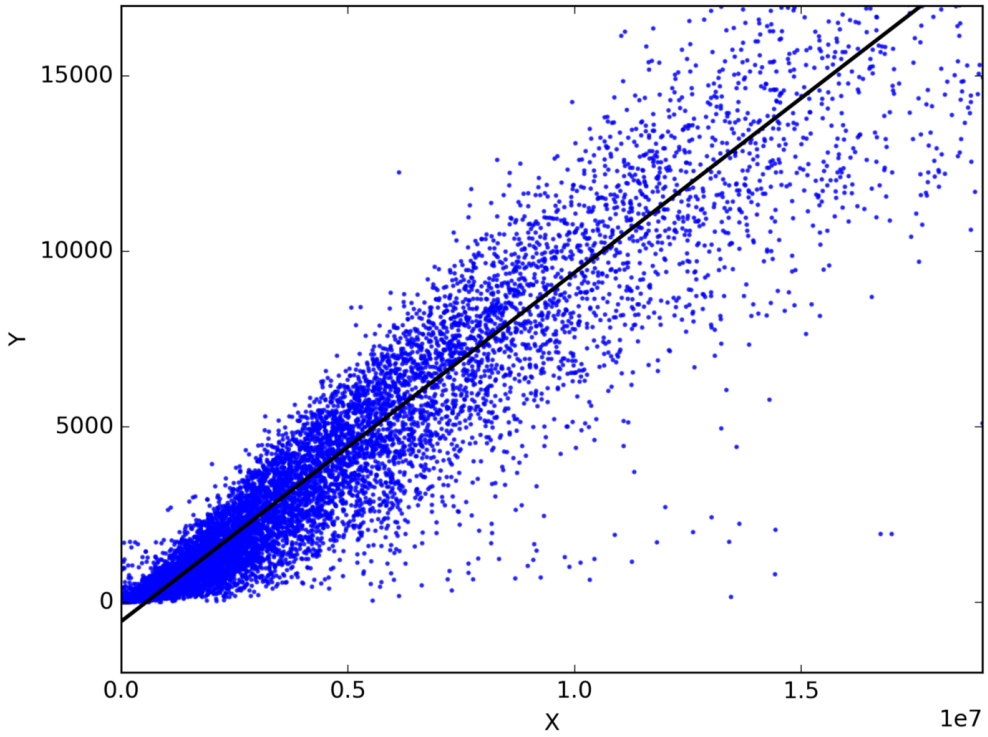

Figure 2.1: Linear Regression

2.1.4 Logistic Regression

Binary classification problems, thus with categorical targets, can be tackled using **Logistic regression**, another powerful supervised ML algorithm. It can be easily imagined as a linear regression but for classification problems. In this problem, a logistic function is used to model a binary output variable. The main difference between linear regression and logistic regression is that the latter's range starts from 0 and it is limited to 1. Furthermore, as opposed to linear regression, its input and output variables does not have to be necessarily linearly linked. This is thanks to the application of a nonlinear log transformation to the odds ratio.

$$\text{Logistic function} = \frac{1}{1+\exp(-x)}$$

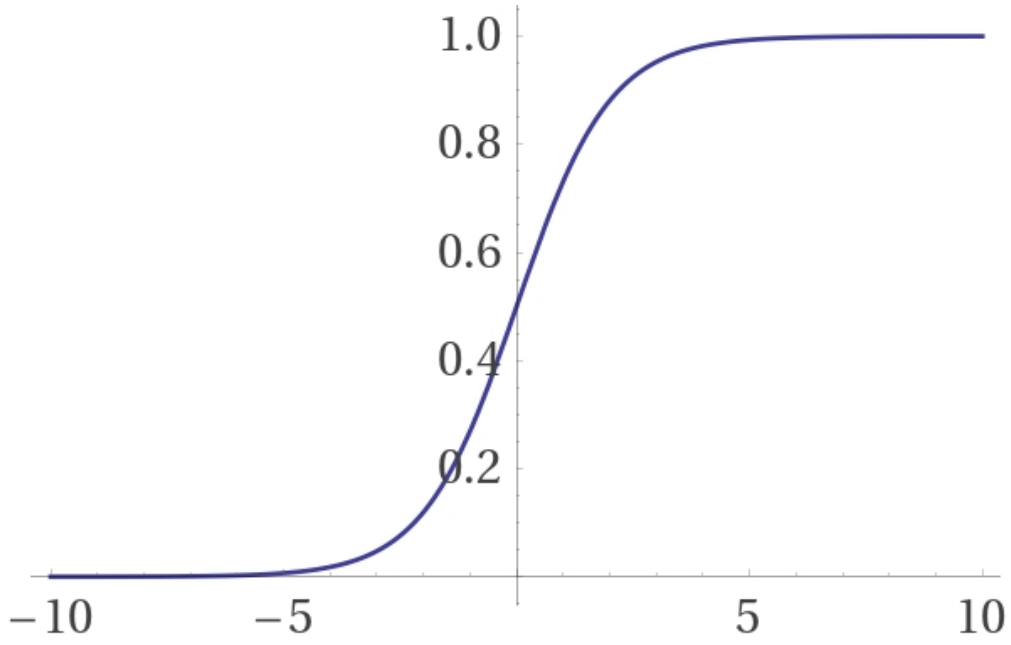

Figure 2.2: Logistic function graph

2.1.5 Neural Networks (Deep Learning)

Deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data. In a simple case, there could be two sets of neurons: ones that receive an input signal and ones that send an output signal. When the input layer acquires an input it sends a modified version of the input to the next layer. In a deep network, many layers can be present between the input and output, especially when it's about a deep network; this allows the algorithm to use multiple processing layers, usually made by many neurons and accomplish multiple linear and non-linear transformations. Deep learning refers to a rather wide class of machine learning techniques and architectures, with the hallmark of using many layers of non-linear information processing stages that are hierarchical in nature. According to how the architectures and methods are intended for use, i.e., classification/recognition or synthesis/generation, it is possible to categorize most of the elements in this area into three classes:

- **Generative Adversarial Nets**, or **GAN**, neural networks composed of up to two networks competing with each other. The two networks namely

generator — to generate data set and discriminator — to validate the data set. The objective is generating data points that are comparable to some of the data points in the training set. GANs are generative models that, through supervised learning, approximate an unmanageable cost function, which is a function that aims to minimize the error. Furthermore, it can be used to generate different cost functions, i.e. the maximum likelihood function.

- **Discriminative deep architectures**: the aim is to directly provide discriminative power for pattern classification, often by characterizing the posterior distributions of classes conditioned on the visible data
- **Hybrid deep architectures**: the aim is to combine the power of discrimination with the outputs of generative architectures via better optimization or/and regularization.

Despite the complex categorization of the deep learning architectures, the one's that are in practice are **deep feed-forward networks**, **Convolutional networks** and **Recurrent Networks**.

Feed-forward Neural Networks

A Feed-Forward Neural Network is a single layer **perceptron**. In machine learning,The perceptron is a simple linear binary classifier and therefore capable of effectively learning the rule necessary to recognize two different and linearly separable input classes. In other words, a binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. Data in succession can enter the layer and it is multiplied by the weights of the model. Then the sum of the weighted input values is computed to form a total. If the sum of the values is more than a predetermined threshold, which is normally set at zero, the output value is usually 1, and if the sum is less than the threshold, the output value is usually -1.

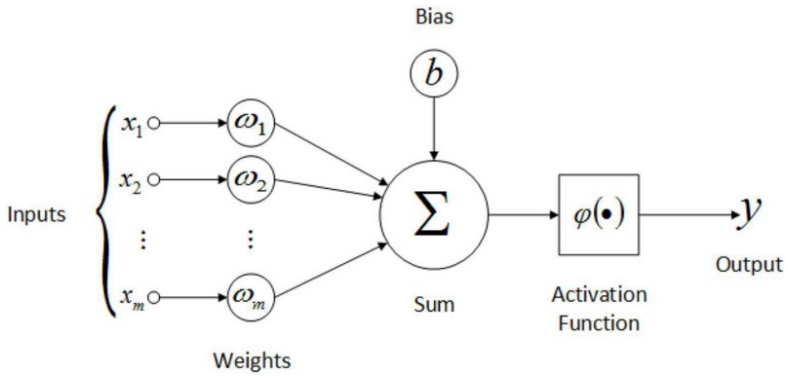

Figure 2.3: Neuron

The neural network can compare the outputs of its nodes with the desired values using a property known as the delta rule, allowing the network to alter its weights through training to create more accurate output values. This procedure made of training and learning is called **gradient descent**. The activity of updating weights in multi-layered perceptrons is theoretically equal, however, the process is defined *back*-propagation. In such conditions, the values returned as output by the final layer are used to modify each hidden layer inside the network. The goal of a feed-forward network is to approximate some function $f*$. For example, for a classifier, $y = f*(x)$ maps an input x to a category y. A feed-forward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

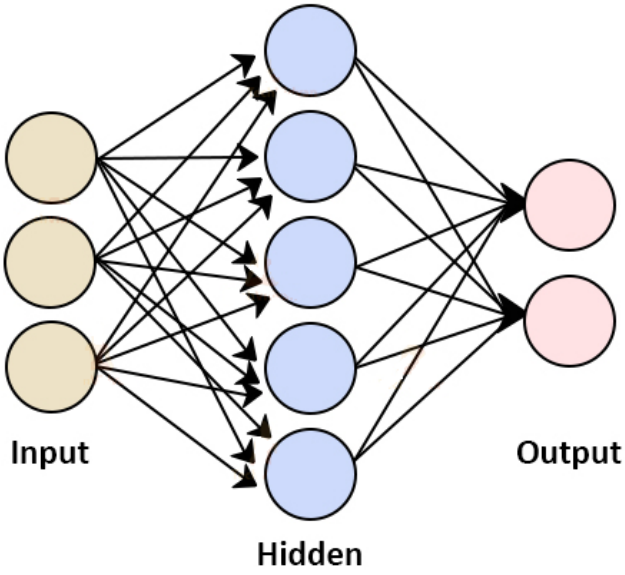

Figure 2.4: Architecture of artificial neural network

Convolutional Neural Networks

A convolutional neural network (CNN or ConvNet) is a network architecture for deep learning that learns directly from data, eliminating the need to manually extract features. In other words, **the pre-processing required in a ConvNet is much lower as compared to other classification algorithms**. Their name is based on the mathematical operation called convolution. A convolution is an operation between two functions of a variable which consists in integrating the product between the first and the second translated by a certain value. It therefore "blends" one function with another and indicates the overlap of the two functions when one is shifted on the other. CNNs are particularly useful for identifying

patterns in images for recognizing objects, faces and scenes. Furthermore, they can be effective for classifying non-image data such as audio data, time series and signals. Applications that require object recognition and computer vision, such as self-driving vehicles and facial recognition applications, rely heavily on CNNs. A key ingredient of ConvNets is the convolution layer. It takes input data from the previous layer and "averages" it locally via a filter, or weight pattern; the output is a locally averaged version of the input layer.

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are neural sequence models that achieve state of the art performance on task like speech recognition, language modeling and machine translation. They are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feed-forward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn.

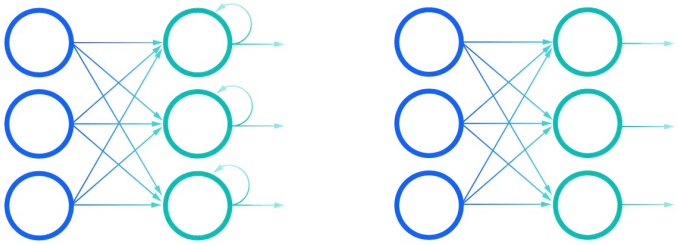

Figure 2.5: Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right)

Another distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feed-forward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. That said, these weights are still adjusted in the through the processes of back-propagation and gradient descent to facilitate reinforcement learning. Unlike feed forward neural networks, which map one input to one output, RNNs do not have this constraint. In fact, their inputs and outputs can vary in length, and different types of RNNs are used for different use cases; common structures are one to one, one to many, many to one and many to many. The activation functions are nearly the same as the other neural networks, the most commonly used are the sigmoid, hyperbolic tangent and ReLU. [4]

2.2 Unsupervised Learning

Unsupervised Learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. It is considered an optimal solution for exploratory data analysis, image recognition, cross selling strategies and costumer segmentation due to its ability to detect similarity and differences in information. In unsupervised learning, the algorithms are trained with data which is neither labeled nor classified. In unsupervised learning, the machine needs to learn from patterns without corresponding output values. The key points of unsupervised learning are that:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar to how a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which makes unsupervised learning more important.

In real world, input data does not always have in advance the corresponding output so, to solve such cases, unsupervised learning is needed. It is also commonly used in a wide spectrum of scenarios, such as: News Sections, Computer vision, Medical imaging, Anomaly detection, Customer personas, Recommendation Engines. There are two main types of unsupervised learning: **Clustering** and **Association**. [5]

2.2.1 Clustering

Clustering consists of a set of methods for grouping objects into homogeneous classes. A cluster is a set of objects that have similarities to each other, but which, on the other hand, are dissimilar with objects in other clusters. The input of a clustering algorithm consists of a sample of elements, while the output is given by a certain number of clusters in which the elements of the sample are divided according to a measure of similarity. It is a common technique for statistical data analysis, used in many fields besides machine learning, like pattern recognition, image analysis, information retrieval, bioinformatics, data compression and computer graphics. Usually, clustering means the partitioning of a given set of points of a certain metric space into subsets in such a way that close points fall into one group, and distant ones fall into different groups. Usually, clustering means the partitioning of a given set of points of a certain metric space into subsets in such a way that close points fall into one group, and distant ones fall into different groups.

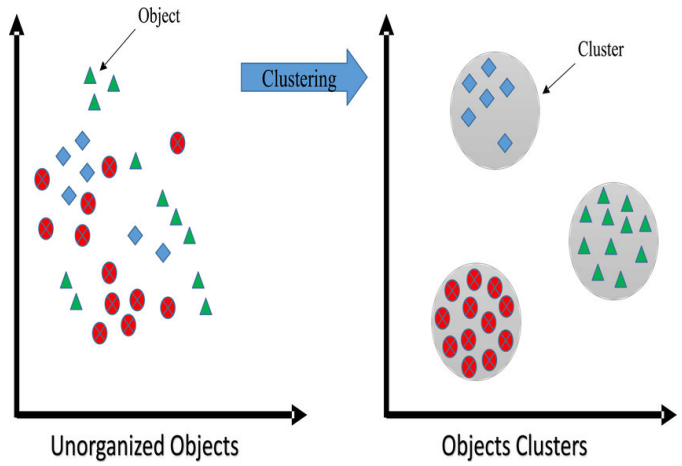

Figure 2.6: Clustering

Clustering Algorithm: K-means

K-means clustering is the most commonly used clustering algorithm. It's a centroid-based algorithm and the simplest unsupervised learning algorithm. The target of the algorithm is to minimize the data points variance within a cluster. The k-means problem can be formalized as a non-convex, mixed Continuous/Boolean problem. In this case study, it proved to be a good algorithm for the purpose, but other different algorithms for clustering do exist however. Let $C = [c_1, ..., c_k]$ be a $n \times k$ matrix that contains the centers.
Let U be a $N \prod k$ Boolean matrix that specifies which data point is assigned to which center, i.e.

$$u_{ij} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is in cluster } j \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

U can be intended as a Boolean matrix with constraints $U\mathbf{1}_k = \mathbf{1}_N$ meaning that each point is assigned to one and only one cluster.
The problem is formalized as:

$$\min_{C,U} \sum_{i=1}^{N} \left\| x^{(i)} - \sum_{j=1}^{k} u_{ij} c_j \right\|_2^2 : \sum_{j=1}^{k} u_{ij} = 1,\ 1 \leq i \leq N,\ U \in \{0,1\}^{N,k} \tag{2.3}$$

The k-means clustering algorithm uses iterative refinements to produce a final result. The algorithm inputs are the number of clusters k and the data set, then it starts with initial estimates for the k centroids, which can either be randomly generated or randomly selected from the data set. The algorithm then iterates between two steps:

- A **data assignment step**, in which each data point is assigned to its nearest centroid, based on the squared Euclidean distance;
- A **centroid update step**, in which optimal centroids are recomputed on the basis of the cluster imputations of the previous step.

Data assignment step: for each $x^{(i)}$, compute the distances from the current centroids $c_j, j = 1, ..., k$, and assign point i to the closest centroid. The set of points assigned to centroid c_j constitutes the cluster C_j.
Centroid update step: given the current clusters, we have that:

$$J^{clust} = \min_{c_1,...,c_k} \sum_{i=1}^{N} \min_{j=1,...,k} \left\| x^{(i)} - c_j \right\|_2^2 = \min_{c_1,...,c_k} \sum_{j=1}^{k} \sum_{i \in C_j} \left\| x^{(i)} - c_j \right\|_2^2 = \sum_{j=1}^{k} \min_{c_j} \sum_{i \in C_j} \left\| x^{(i)} - c_j \right\|_2^2 \tag{2.4}$$

The minimum of $\sum_{i \in C_j} \left\| x^{(i)} - c_j \right\|_2^2$ is simply attained by the centroid (barycenter) of the cluster C_j:

$$\boxed{c_j = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}} \tag{2.5}$$

Given a list of N vectors $x^{(1)}, ..., x^{(N)}$, and an initial list of k cluster representatives $c_1, ..., c_k$, repeat until convergence

1. *Data assignment*: assign each vector $x^{(i)}, i = 1, ..., N$ to its nearest representative.
2. *Update representatives*: for each group $j = 1, ..., k$, set c_j to be the mean of the vectors in group j.

Clustering Algorithm: Hierarchical Clustering

Hierarchical clustering, is an unsupervised learning algorithm that groups similar objects into groups called clusters.

There are ways to create cluster hierarchy:

- Agglomerative (bottom up approach)
- Divisive (top down approach)

In the **Agglomerative Hierarchical Clustering Technique**, initially each point is considered a cluster. Then the algorithm finds out which 2 clusters are the most similar and groups them to form a new cluster. It repeats this process till all the clusters are merged to form a single cluster.
Its fundamental steps are:

- In the initial step, the proximity of individual points is computed and all the data points are considered as individual.
- Step two: single clusters are formed by merging together similar clusters.
- Repeat: merge the two closest clusters and update the proximity matrix, until only a single cluster remains

The Hierarchical clustering Technique can be visualized using a dendrogram. A dendrogram is a type of tree diagram showing hierarchical clustering — relationships between similar sets of data. They can be column graphs (as in the image below) or a row graph.

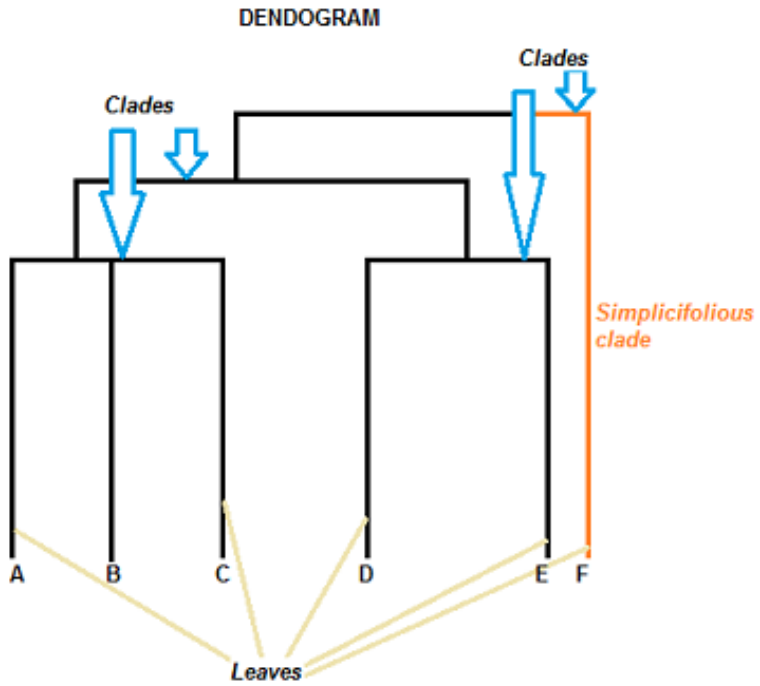

Figure 2.7: Dendrogram

The graph is composed by:

- The clade is the branch. Usually labeled with Greek letters from left to right (e.g. α, β, $\gamma \ldots$).
- Each clade has one or more leaves. The leaves in the above image are:
  -Single (simplicifolius): F
  -Double (bifolius): D E
  -Triple (trifolious): A B C

The maximum number of leaves is theoretically infinite, but the more is it, the more the graph will be hard to read. The clades arrangement is made according the similarity (or dissimilarity) of the clades; for example, two clades with close height are similar; clades with different heights are dissimilar — **the greater the difference in height, the more dissimilarity**.

The **Divisive Hierarchical Clustering Technique** is less used than the previous one, and works in the opposite way. In fact, all the data points are considered as a single cluster and in each iteration, the data points are separated from the cluster which are not similar. Individual clusters are made of also by each data point which is separated. In the end, n clusters will be left. [6] [7] [8]

Clustering Algorithm: Spectral Clustering

Spectral clustering is a technique with roots in graph theory, where the approach is used to identify communities of nodes in a graph based on the edges connecting them. It makes use of the spectrum (**eigenvalues**) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions.

A small recall about eigenvalues, eigenvectors and graphs
Let A be an $n \times n$ matrix.
A *non-zero* vector $\vec{x} \in \mathbb{R}^n$ is called an **eigenvector** of A if there exists some scalar $\lambda \in \mathbb{R}$ so that $A\vec{x} = \lambda\vec{x}$.
If $\vec{x}$ is an eigenvector of A, the corresponding value λ is called an **eigenvalue** of A, and λ is an eigenvalue of A with eigenvector $\vec{x}$.
While an eigenvector $\vec{x}$ must be non-zero (so that we are always excluding the trivial case $A\vec{0} = \vec{0}$), it is possible for the value λ to be zero.
$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$ has eigenvector $\begin{pmatrix} 2 \\ -1 \end{pmatrix}$ with eigenvalue 0. If λ is an eigenvalue for A, the eigenvectors for A corresponding to λ along with $\vec{0}$ form a subspace of $\mathbb{R}^n$.
Eigenvalues and eigenvectors of a matrix can be found in python using *numpy*, as shown below:

```
import numpy as np

# a 2x2 matrix
A = np.array([[0,1],[-2,-3]])

# find eigenvalues and eigenvectors
vals, vecs = np.linalg.eig(A)

# print results
```

```
    for i, value in enumerate(vals):
    print("Eigenvector:", vecs[:,i], ", Eigenvalue:", value)

    # Eigenvector: [ 0.70710678 -0.70710678] , Eigenvalue: -1.0
    # Eigenvector: [-0.4472136   0.89442719] , Eigenvalue: -2.0
```

A **graph** is a relational structure formed by a finite number V of vertices (or nodes) and a finite number E of segments (edges or links) that connect each node to the others. Let $G = (V,\ E)$ be a simple graph (no loops or multiple edges) with vertex set $V(G) = \{v_1, \cdots, v_n\}$ and edge set $E(G)$. It can be represented using an **adjacency matrix** $A(G)$: the elements of the matrix indicate whether pairs of vertices are adjacent (connected by an *edge*) or not in the graph. The elements have float values if the edges are weighted and boolean values if not.

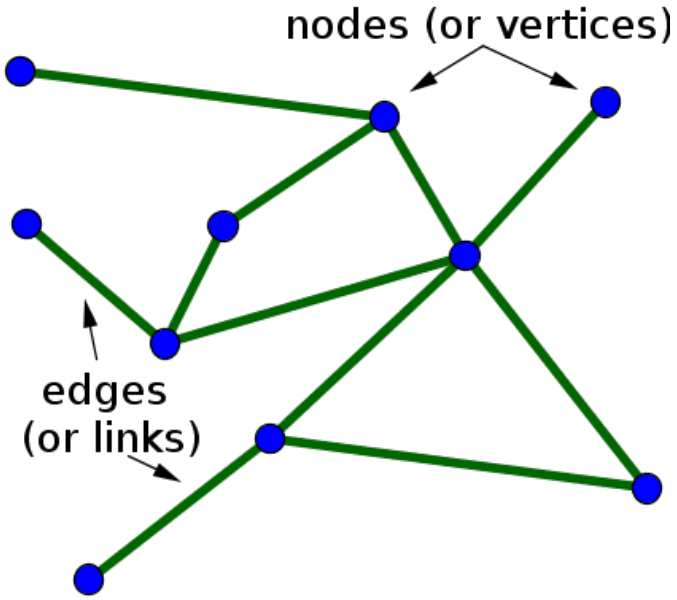

Figure 2.8: Graph example

Another important element in the graph theory is the **degree matrix**. The degree of a node expresses many edges connect to it; the *degree* of a vertex v_i is denoted by $d(v_i)$ or $d_G(v_i)$. The degree matrix is diagonal and each value at entry (i, i) is the degree of node i. The values can be computed summing each element of the relative row of the *unweighted adjacency matrix*.
Finally, the **Laplacian matrix** is another matrix representation of a graph, with several useful properties that can be taken advantage of also in spectral clustering. If $D(G) = diag(d(u), u \in V)$ is the diagonal matrix of vertex degrees of G and $A(G)$ is the (0,1) *adjacency matrix* of G, then the matrix $L(G) = D(G) - A(G)$ is called the *Laplacian matrix* of a graph G. It is obvious that $L(G)$ is positive

semi-definite and singular $M-$matrix.

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } (i,j) \in E \\ 0 & \text{if}(i,j) \notin E \end{cases} \tag{2.6}$$

Thus the all eigenvalues of $L(G)$ are called the *Laplacian eigenvalues* (or sometimes just eigenvalues) of G and arranged in non-increasing order:

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{n-1} \geq \lambda_n = 0 \tag{2.7}$$

The Laplacian's diagonal is the degree of the nodes, and the off diagonal is the negative edge weights. When a graph is completely disconnected, every Laplacian eigenvalue is 0. As the number of edges increases, some of the eigenvalues increase. In fact, the number of 0 eigenvalues corresponds to the number of connected components in the graph. The first nonzero eigenvalue is called the *spectral gap*. It gives some notion of the density of the graph. For example, in a 10 nodes graph densely connected (all pairs of the 10 nodes had an edge), the spectral gap is 10. The second eigenvalue is called the Fiedler value and indicates the minimum graph cut needed to separate the graph into two connected components. Thanks to that, the value can indicate how to separate the nodes into those approximately connected components.
Finally, spectral clustering involves 3 main steps:

1. Compute a similarity graph
2. Project the data onto a low-dimensional space
3. Create clusters

The graph can be created using ε**-neighborhood**, **K-Nearest Neighbors** (KNN) or can be **fully connected**.

- ε-neighborhood consists in fixing a ε and connecting each point to all the points which lie in it is ε-radius. Typically, the weight of the edges, i.e. the distance between the two points, are stored if all the distances between any two points are significantly different in scale, because they do provide additional information; if the difference is negligible, the weight are not stored. Thus, in this case, the graph built is an undirected and unweighted graph.
- K-Nearest Neighbors instead creates an edge for two vertices u and v only if v is among the k-nearest neighbors of u. Note that this leads to the formation of a weighted and directed graph because it is not always the case that for each u having v as one of the k-nearest neighbors, it will be the same case for v having u among its k-nearest neighbors.

- A fully connected graph is characterized by having all points connected with each other and edges weighted by the distance between the two points to every other point.

In order to **project the data onto a low-dimensional space**, the **graph Laplacian matrix** is computed, then normalized to improve math efficiency. This is done to avoid excessive distances, in the given dimensional space, between elements that could be of the same cluster. After that, k eigenvalues and eigenvectors are computed with k = number of clusters. [9] [10] [11] [12] Finally, the cluster are being defined:

1. Stack the eigenvectors vertically to form a matrix with the vectors as columns.
2. Every row of the new matrix represents the corresponding node. These rows form the feature vectors of the nodes.
3. Use a clustering technique (usually k-means) to now cluster these points into k clusters $\{C_1, \cdots, C_k\}$

Chapter 3

System description

One of the software used to monitor the systems is Broadcom DX Application Performance Management. CA Introscope is a product within the Application Performance Management solution that allows to:

- Monitor complex web applications in production environments 24 hours a day, 7 days a week
- Detect problems before they affect customers
- Resolve these issues quickly and collaboratively

CA Introscope performs end-to-end application transaction management and analysis. It is worth to briefly explain the purpose of the agents and other key elements in the monitoring process:

- The **Enterprise Manager** acts as the repository of Introscope performance metrics. The Enterprise Manager receives performance metrics from one or more Introscope agents. The agents users collect metrics centrally from many applications, application servers, and supporting systems. The Enterprise Managers can be deployed in different ways depending on the size and complexity of the enterprise system. The role of an Enterprise Manager depends on its deployment in a standalone or clustered APM environment.
- The **agent** is a data gathering component, collecting detailed performance information about applications and the computing environment as transactions are executed.
- The **Workstation** provides the Investigator, Console, and APM Status Console for viewing application health and data. With the Workstation, the CA APM administrator can perform these actions:

  - Set alerts for individual metrics or logical metric groups.
  - Customize views to represent their unique environment.
  - Set up reports for application health, Service Level Agreements, and capacity planning.

3.1 Dataset

The used dataset was extracted through the software described above. All the data points detected were extracted with a 15 seconds gap from each other. There are five main features:

- **ART** (Average Response Time). This value indicates the mean time in ms passed between one transaction and another.
- **Stalls**, which indicates the number of freezed transactions whit no response.
- **Concurrences**, which indicates the number transactions of the same type active on the application server
- **RPI** (Responses Per Interval)
- **CPU**, that reports the mean CPU usage in the interval

Other inevitably extracted data such as "Timestamp" or "app_name" was discarded because not relevant for the purposes of training. Initially, in the dataset, was also present a label indicating the application state, a boolean value specifying if the app was running correctly or not. This label was used for the supervised learning phase. The label expressing the malfunctioning of the app was manually inserted the data values in correspondence with the problems.

ART	Stalli	Concorrenze	RPI	CPU	stato_app
26	0	10	2367	29	1
28	0	7	2392	30	1
26	0	4	2259	28	1
27	0	8	2566	30	1
29	0	8	2471	32	1
27	0	5	2428	30	1
28	0	10	2451	32	1
28	0	9	2474	31	1
26	0	3	2530	31	1
28	0	3	2573	32	1
29	0	5	2501	30	1
26	0	1	2569	29	1
29	0	8	2476	31	1
26	0	680	2341	30	1
38	0	997	1851	32	1
6075	0	1005	3337	32	0
4464	1	1006	2147	30	0
7321	0	1006	2782	31	0
4912	1	1004	1975	31	0
7121	0	1005	2644	30	0
4179	2	1007	2712	35	0
5840	0	1003	2949	31	0
5892	0	1006	2198	30	0
6959	2	1002	2447	30	0
4688	2	1005	2801	33	0
5872	0	1002	2834	31	0
4459	3	1011	3035	36	0
6990	0	1003	2328	28	0
5774	1	1005	2403	30	0
5869	0	1007	2714	31	0
4954	0	1006	2846	34	0
6526	4	1011	2356	29	0
5122	4	1003	2887	33	0
5233	3	996	2844	32	0
6293	3	1002	2452	35	0
6392	2	1002	2255	29	0
5197	2	1008	2995	34	0
4557	3	999	3053	32	0
5614	3	1006	2895	33	0
4463	2	1008	3030	32	0
5556	2	6	3793	33	0
28	0	6	2632	34	1
28	0	2	2572	33	1
28	0	7	2630	34	1
29	0	5	2493	31	1
29	0	6	2574	33	1
28	0	11	2519	32	1
27	0	6	2720	34	1
28	0	5	2777	36	1
27	0	5	2760	33	1
28	0	5	2767	34	1

Table 3.1: Dataset example

In this excerpt from the dataset it is possible to see the app passing from an OK state (stato_app = 1) to a KO state (stato_app = 0)

Chapter 4

Supervised learning

4.1 Supervised Learning Performance Metrics

Performance metrics are a powerful tool required for comparing the different methods in machine learning. Below are reported some of the most used metrics. **Accuracy** score in supervised learning is a fundamental parameter to judge a model, but alone it is not a clear indicator of the performance. It is the number of correctly predicted data points out of all the data points. Mathematically, it is computed as the number of true negatives and true positives divided by the number of true negatives, true positives, false negatives, and false positives. A true positive (**TP**) or true negative (**TN**) is a data point that the algorithm correctly classified as true or false, respectively. A false positive (**FP**) or false negative (**FN**), on the other hand, is a data point that the algorithm incorrectly classified. Accuracy is calculated with the formula:

$$\frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

Another metric is the **Precision**, which indicates the percentage of positive instances out of the total predicted positive instances. In other words, it indicates *how much the model is right when it says it is right.*

$$\frac{TP}{TP + FP} \tag{4.2}$$

The **Recall**, or **True Positive Rate** express the percentage of positive values out of the **total actual positives** values. It can be explained with the sentence *how much right values the model missed showing the right values.*

$$\frac{TP}{TP + FN} \tag{4.3}$$

The **Specificity** is the percentage of negative instances out of the **total actual negative** instances. It is a measure of *how much false values the model missed showing the false values.*

$$\frac{TN}{FP + TN} \tag{4.4}$$

The **F1 score** is the harmonic mean of **precision** and **recall**, hence it takes the contribution of both values.

$$\frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * precision * recall}{precision + recall} \tag{4.5}$$

The parameters described before can be found in the **Confusion Matrix**, which is a $2 * 2$ matrix that suitably arranges the number of samples that fall in the four possibilities (TP, FP, FN, TN).

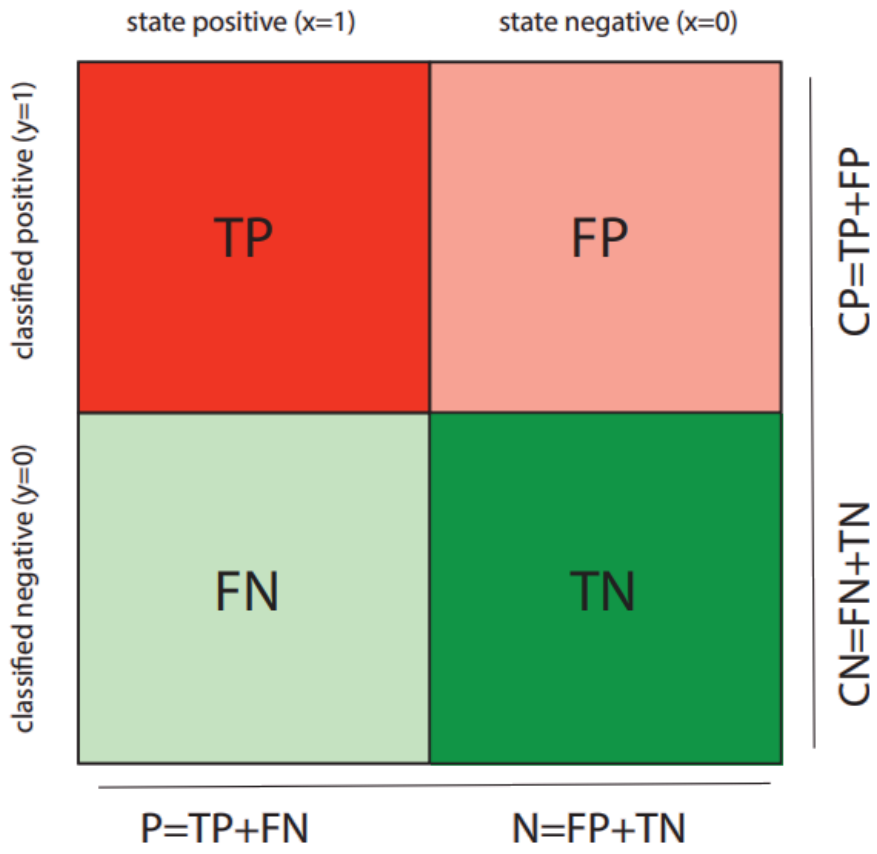

Figure 4.1: Confusion Matrix

Finally there is the **ROC curve**, that stands for *receiver operating characteristic* and the graph is plotted against True Positive Rate (**TPR** or **recall**) and False Positive Rate (**FPR** or $1 - specificity$) for various threshold values. Decreasing the threshold of the classification marks more items as positive, therefore increasing both False Positives and True Positives. A typical ROC curve is represented below. To compute the points in an ROC curve, it's possible to evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information, called AUC.

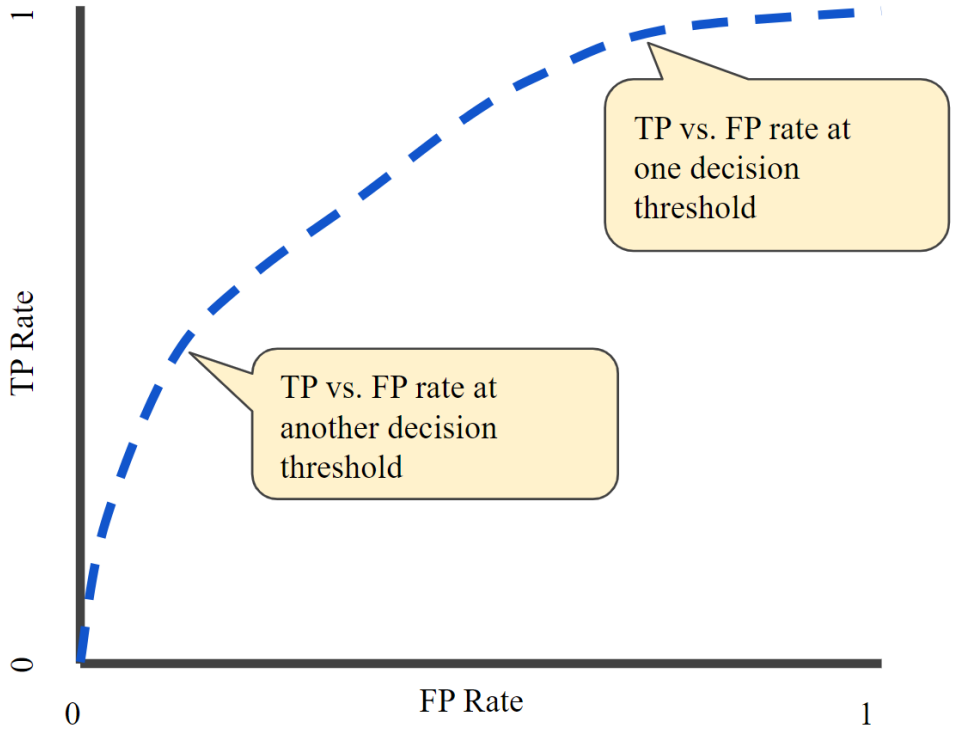

Figure 4.2: ROC curve: TPR vs FPR

AUC stands for "*Area under the ROC Curve.*" That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (integral calculus) from [0,0] to [1,1].

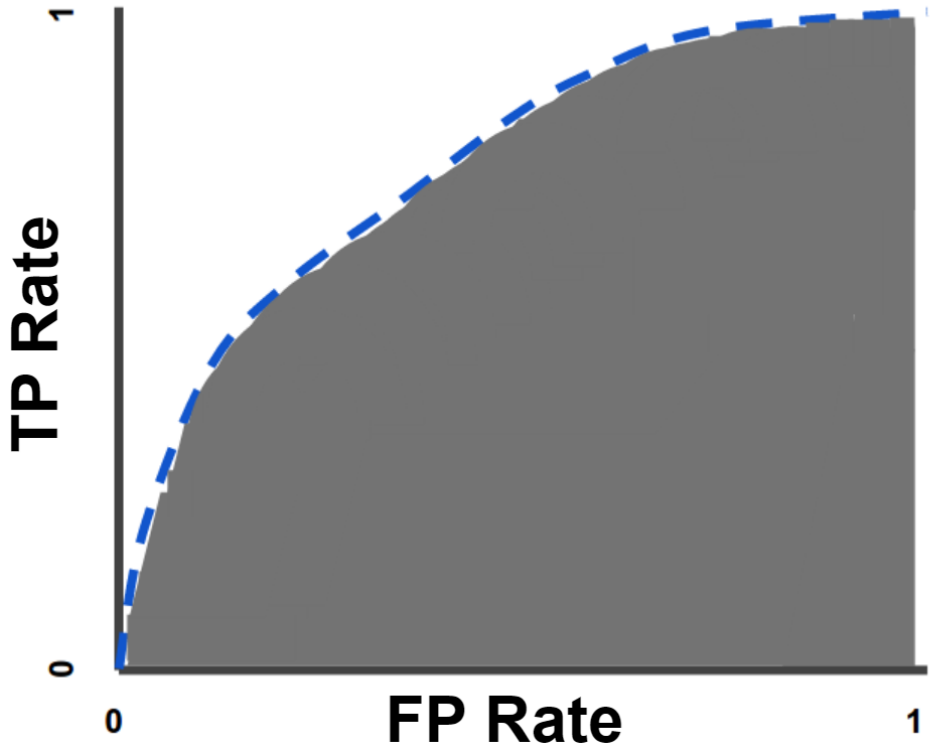

Figure 4.3: AUC curve

An excellent model has AUC near to the 1 which means it has a good measure of separability. A bad model has a value of AUC near 0 which indicates it has the worst measure of separability. Actually, it implies it is reciprocating the result, predicting exactly the opposite. When AUC is 0.5, it means the model has no class separation capacity whatsoever. [13] [14] [15] AUC is desirable for the following two reasons:

- AUC is scale invariant. It measures how well predictions are ranked, rather than their absolute values.

- AUC is classification threshold invariant. It indicates the goodness of the model's predictions no matter of what classification threshold is chosen.

In conclusion, there are several metrics for evaluating performance. However, the choice depends heavily on the task to accomplish. Some metrics are more useful than others in certain situations, depending on the type of data to predict.

4.2 Principal Component Analysis

Dataset are sometimes large and with many features and entries; this could be cause of hurdles in machine learning tasks. The Principal Component Analysis (**PCA**) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. This is made by creating new uncorrelated variables that consecutively maximize variance. PCA finds "principal components" (PCs), i.e. orthogonal directions of maximal variance. PCs are computed via Eigen Value Decomposition (**EVD**) of covariance matrix or alternatively, PCs can be found directly via Singular Value Decomposition (**SVD**) of (centered) data matrix.

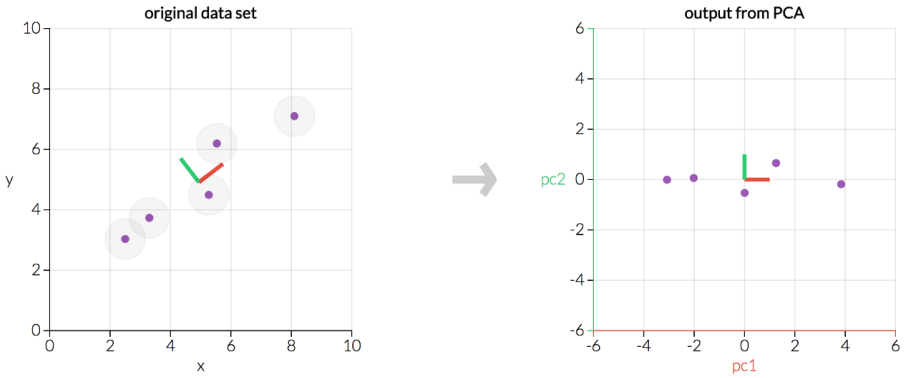

Figure 4.4: Example of a PCA transformation

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote a dataset with n samples and d attributes. For convenience, let also assume that $\mathbf{X}$ has zero mean (otherwise, one should center the data by extracting its mean). With the purpose of reducing the dimensionality of $\mathbf{X}$ from d to r-dimensional samples, the goal in PCA is to find a projection matrix $\mathbf{U} \in \mathbb{R}^{d \times r}$ such that the projected data $\mathbf{XU}$ are uncorrelated and retain as much information from $\mathbf{X}$ as possible. A typical solution of PCA is obtained by tackling the following optimization problem:

$$\begin{aligned} \min_{\mathbf{U}} \quad & \left\|\mathbf{X} - \mathbf{X}\mathbf{U}\mathbf{U}^T\right\|_F^2 , \\ \text{s.t.} \quad & \mathbf{U}^T\mathbf{U} = \mathbf{I} \end{aligned} \tag{4.6}$$

where $\mathcal{R}_{\mathbf{X}}(\mathbf{U}) = \left\|\mathbf{X} - \mathbf{X}\mathbf{U}\mathbf{U}^T\right\|_F^2$ represents the overall reconstruction error, $\|\cdot\|_F$ is the Frobenius norm and $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix, ensures that $\mathbf{U}$ is orthogonal. It is easy to show that minimizing $\left\|\mathbf{X} - \mathbf{X}\mathbf{U}\mathbf{U}^T\right\|_F^2$ is equivalent to maximize $\operatorname{tr}\left(\mathbf{X}\mathbf{X}^T\right) - \operatorname{tr}\left(\mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U}\right)$ (or maximize $\operatorname{tr}\left(\mathbf{U}^T\mathbf{C}_{\mathbf{X}}\mathbf{U}\right)$,

where $\mathbf{C_X} = \frac{1}{n}\mathbf{X}^T\mathbf{X}$ is the covariance matrix of $\mathbf{X}$, since $\text{tr}\left(\mathbf{X}\mathbf{X}^T\right)$ is a constant). Moreover, since $\text{tr}\left(\mathbf{U}^T\mathbf{X}^T\mathbf{X}\mathbf{U}\right) = \|\mathbf{X}\mathbf{U}\|_F^2 = \sum_{j=1}^{r} \mathbf{u}_j^T\mathbf{X}^T\mathbf{X}\mathbf{u}_j$, minimizing the reconstruction error is also equivalent to maximize the total variance of the projected data. It is also known in the literature that the solution of PCA is achieved by setting the columns of $\mathbf{U}$ as the eigenvectors of $\mathbf{C_X}$ associated with its highest eigenvalues. [16] [17] [18]
Finally, in simple terms, PCA is a method that brings together:

- A measure of how each variable is associated with one another. (Covariance matrix.)

- The directions in which our data are dispersed. (Eigenvectors.)

- The relative importance of these different directions. (Eigenvalues.)

4.3 Gaussian Naive Bayes Classifier

The first attempt to accomplish the classification was made realizing a *Gaussian Naive Bayes Classifier*. The code was implemented on a Jupiter Notebook using Google Colab. Initially all the necessary imports are performed, such as *pandas, numpy, sklearn, matplotlib*. Then the labeled dataset is loaded from a CSV file and converted into a dataframe. After that the dataset is randomly shuffled, divided in two parts, about 80% for the training and 20% for the testing, and then the dataframe values are *normalized*. Next a Principal Components Analysis is performed and the dimensionality is reduced to two. Finally the model is fitted on the dataset and next tested on the test dataframe.

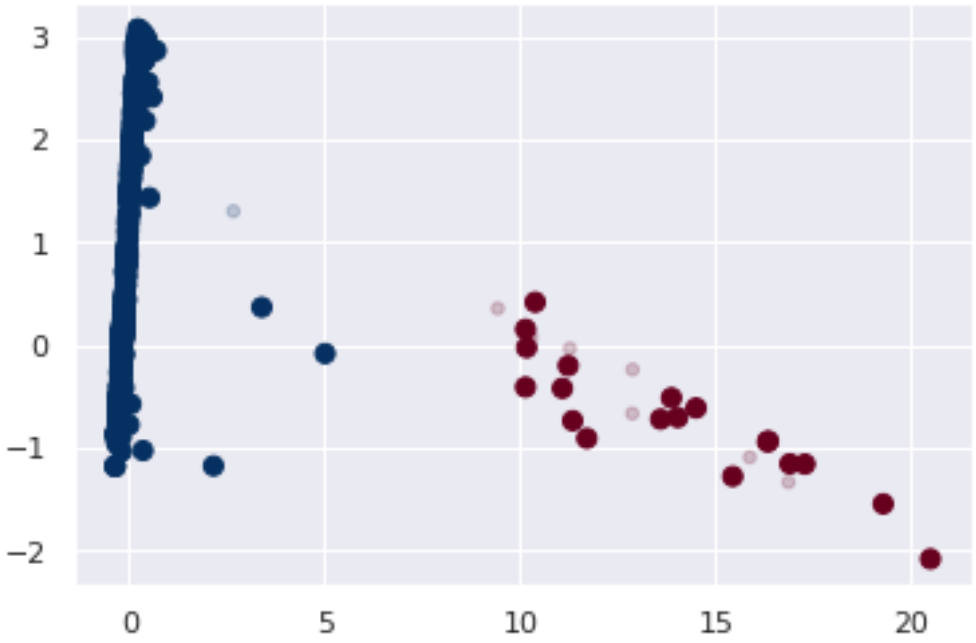

Figure 4.5: Gaussian Naive Bayes graph

In the graph above is represented the data in the test phase. The semi-transparent points are the ones used to test the algorithm, while the points used during the training are opaque. Then the mean accuracy on the given test data and labels returned was about 93%.

4.4 First version: PyTorch classifier

The first version of the classifier was implemented with PyTorch, one of the most used open source framework for machine learning. The initial part of the notebook imports the necessary modules for the process, i.e. *torch*, *matplotlib*, *csv*, *numpy* and *pandas*. Then the dataset, saved as a csv file, is read from Google Drive and opened with the pandas specific function. Then the features and the relative labels are split and loaded into two different torch tensors. After that, a linear classifier model class is being defined, with a layer which applies a linear transformation $y = xA^T + b$. Finally, a torch optimizer is introduced, which takes the parameters to update, the learning rate to use (and possibly many other parameters as well) and performs the updates through its `step()` method. The next phase is the training one, divided in 100 epochs while monitoring training and test accuracy. In the end the loss and accuracy plot are printed, as shown below.

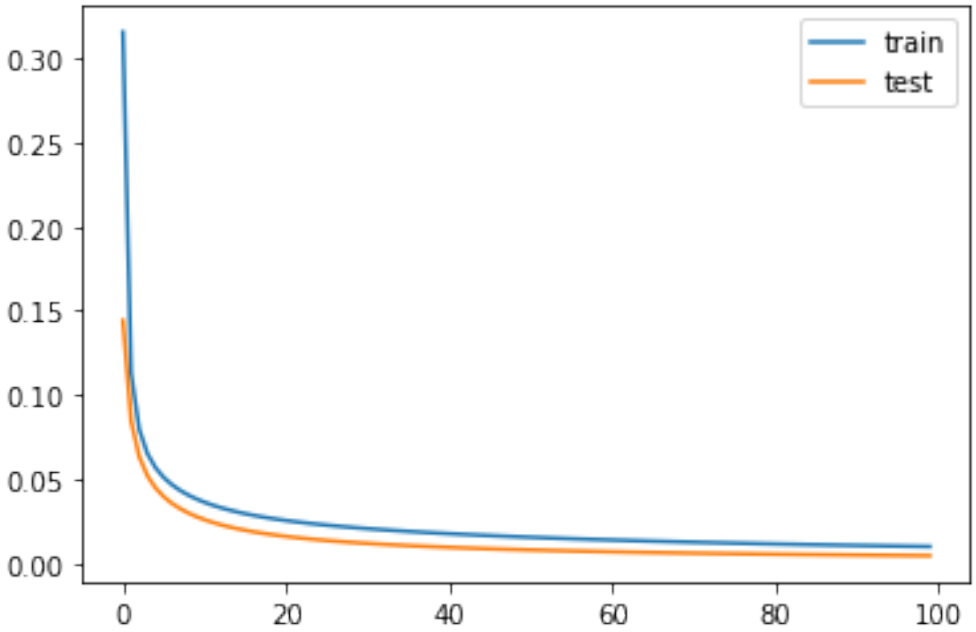

Figure 4.6: Pytorch loss

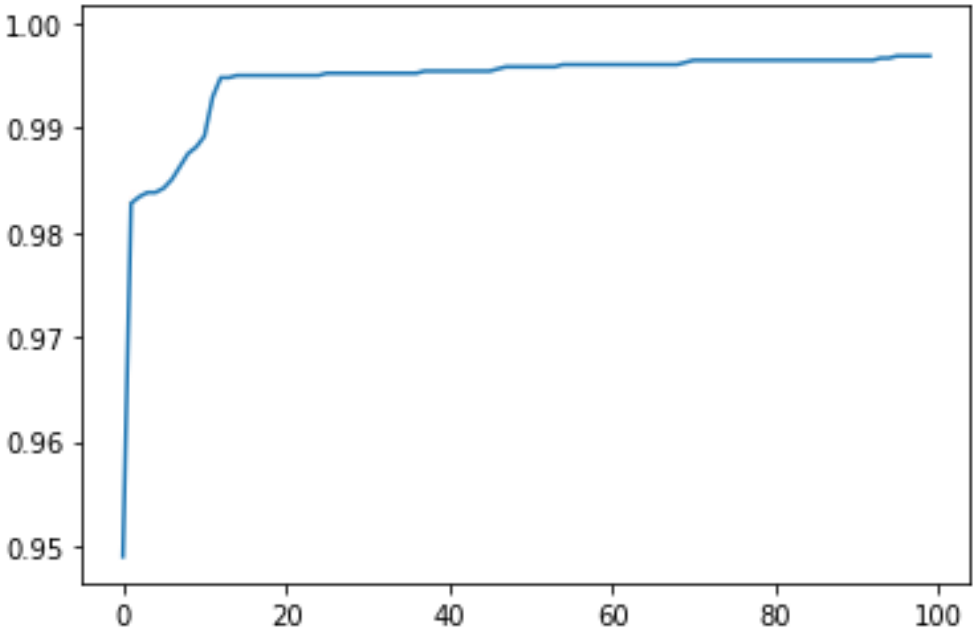

Figure 4.7: Pytorch accuracy

4.5 Second Version: Tensorflow classifier

A second version of the system was instead implemented with Tensorflow, which is another free and open-source software library for machine learning and artificial intelligence. Just as before, the first operations of the script are importing the necessary modules and opening the dataset to use, which is the same in order to do a right comparison between the two versions. Then the dataset values are normalized, this is the process of converting an actual range of values into a standard range of values, typically -1 to +1 or 0 to 1. The code normalizes datasets by converting each raw value. The code normalizes datasets by converting each raw value (including the label) to its Z-score. A Z-score is the number of standard deviations from the mean for a particular raw value to its Z-score. With this different configuration, the results achieved are more regular and overfit less than the previous version, as can be seen from the following images. [19]

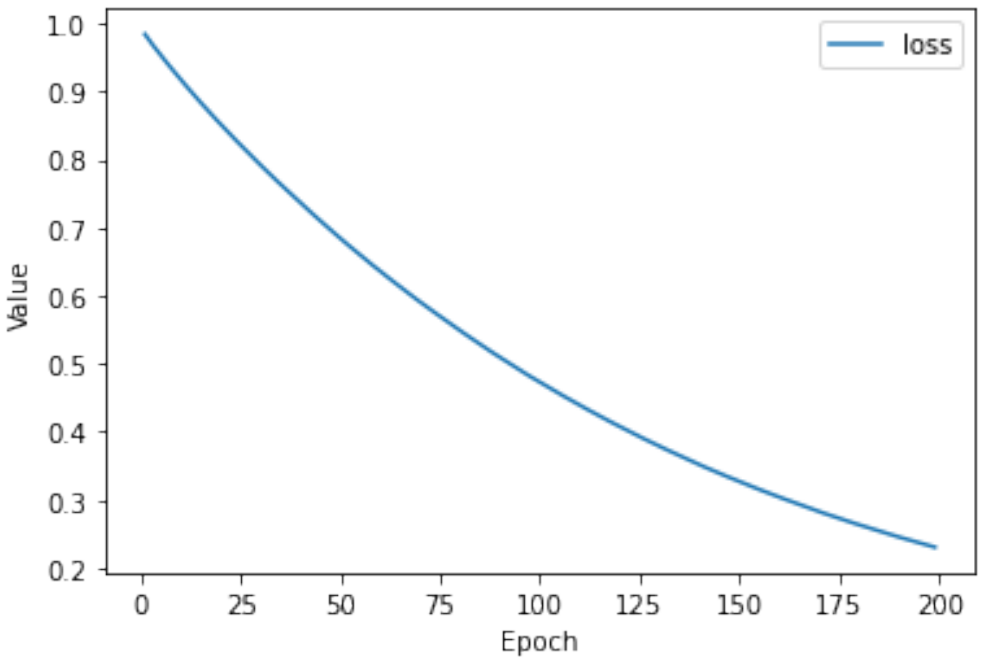

Figure 4.8: TensorFlow classifier loss

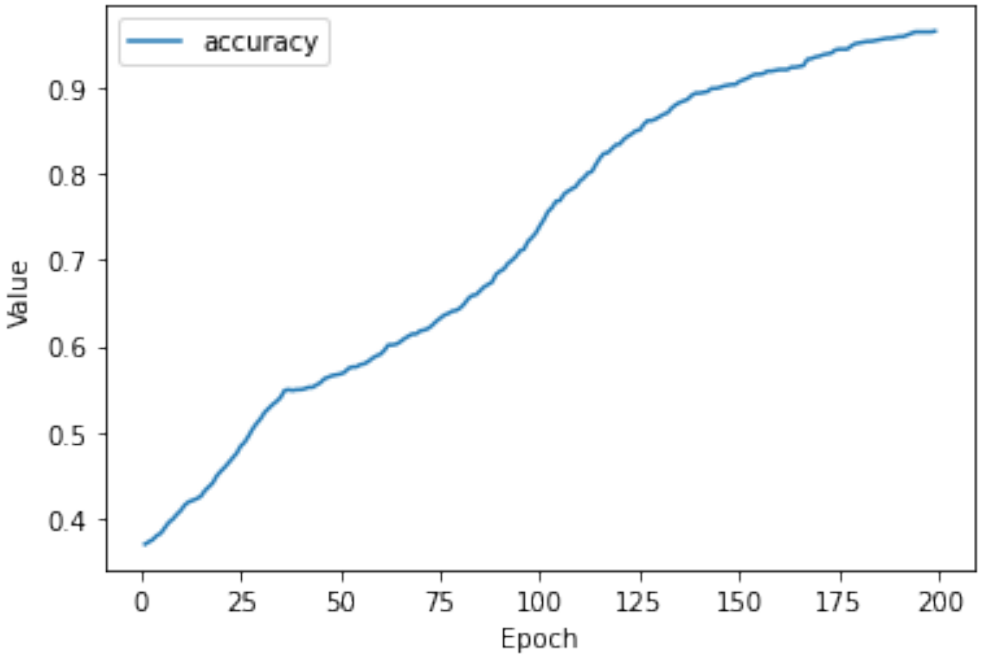

Figure 4.9: TensorFlow classifier accuracy

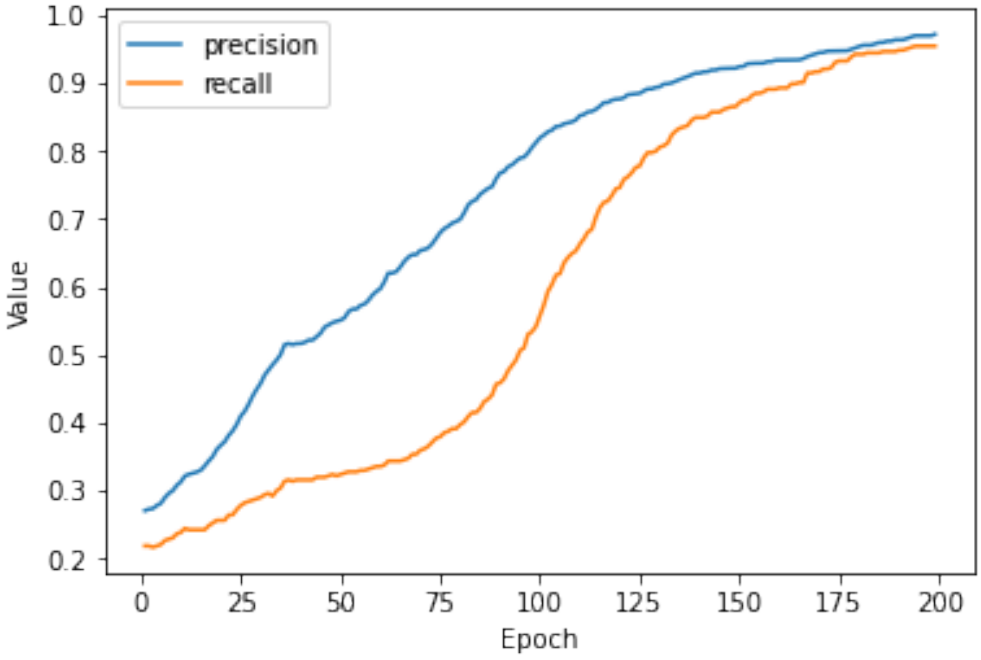

Figure 4.10: TensorFlow classifier precision and recall

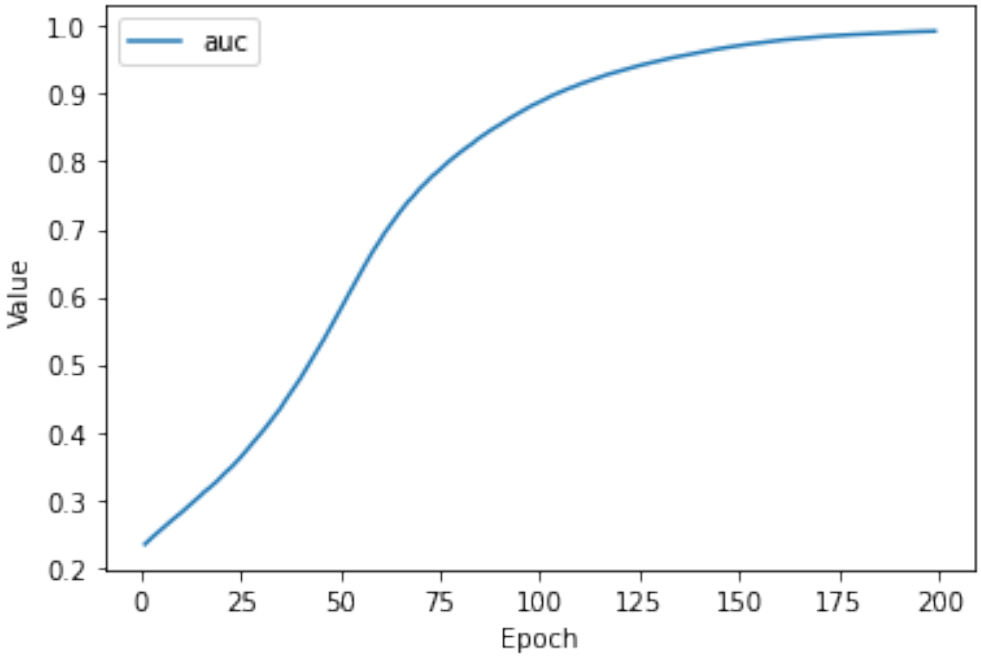

Figure 4.11: TensorFlow classifier AUC

4.6 Comments

The two versions give slightly different outputs; in this particular case, considering that the results are highly related to the dataset, it is possible to say that the TensorFlow version behavior is more appropriate. The training dataset used, as said before, was made specifying the application status in each detection interval. A further important improvement is making the classifier understand the application status without a manually labeled dataset, e.g. realizing an unsupervised learning, which will be discussed in the next chapter.

Chapter 5

Unsupervised learning

5.1 Clustering the data

A clustering approach is chosen in order to make the system recognize the app status autonomously. In particular, due to the nature of the values populating the dataset in the presence of an error, it is possible to implement and compare different clustering algorithms with good results. The dataset features are five, and the label to assign is a boolean value, that indicates the application state. The clustering algorithms are written into jupiter notebooks and developed using *scikit-learn*, which is a free software machine learning library for the Python programming language.

5.1.1 The clustering algorithms

K-means

The first algorithm chosen is based on *k-means*. The first section of the notebook deals with the import of the various modules, i.e. *pandas*, *numpy* and *scikit-learn*. Then the dataset (in csv format) is opened through *pandas* and the eventual null values are replaced with the mean value of the variable. After this step, the number of cluster is defined and in this case it is equal to two, for the possible application states: OK and KO. Finally, the **k-means** algorithm splits autonomously the data points in the two clusters, returning a label indicating the cluster in a csv file and the relative features.

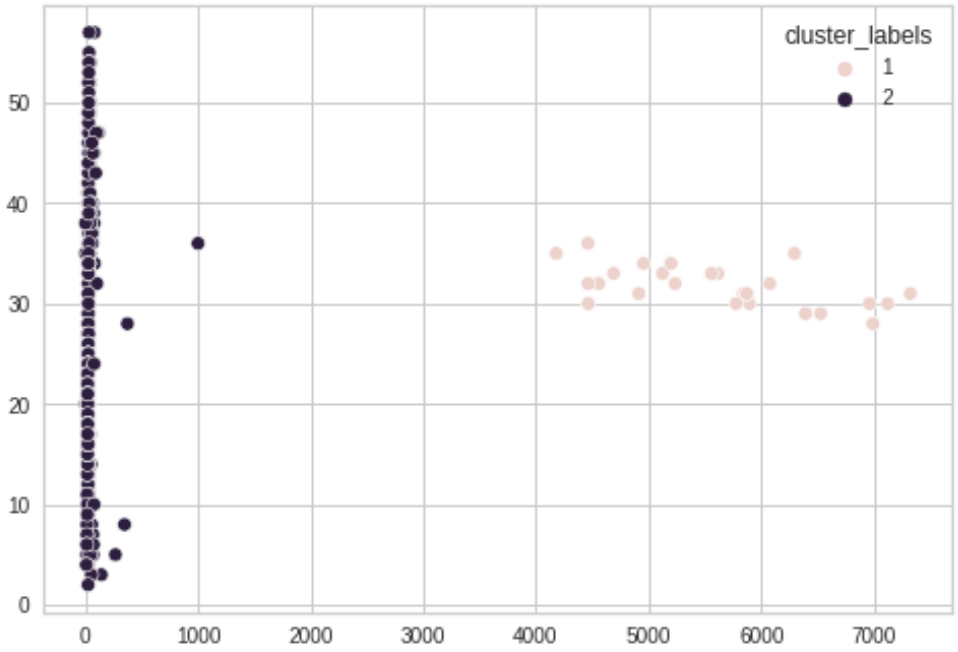

Figure 5.1: K-means clustering results

The *k-means clustering* returned the results represented in the graph above.

Hierarchical Clustering

The second algorithm is an **Agglomerative Hierarchical Clustering Technique**. It is performed using a library called *scipy* and using the same data in order to make a comparison. The procedure is pretty similar, first the notebook imports the various modules, then it reads the (obviously unlabeled) dataset, reduces its dimensionality to two and makes clusters. The results are then appended as labels in the original dataset in a csv file and plotted. The results are very similar to the previous ones: in fact the clusters detected are practically the same.

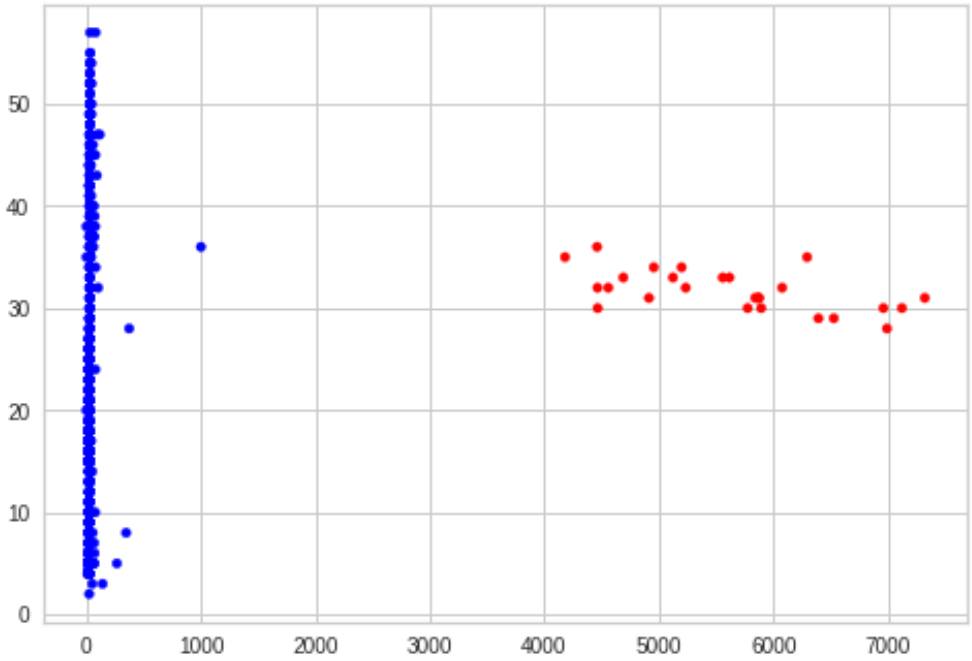

Figure 5.2: Hierarchical clustering results

Spectral clustering

Spectral clustering is the third type of clustering used in this scenario. It is implemented into a Jupiter notebook as the others and using the same datasets. It makes use of a library called *scipy*, and the procedure of the script is pretty similar to the others: first the modules are imported, then the unlabeled dataset is being read and its dimensionality is reduced using PCA. The results are then plotted and saved to a csv file.

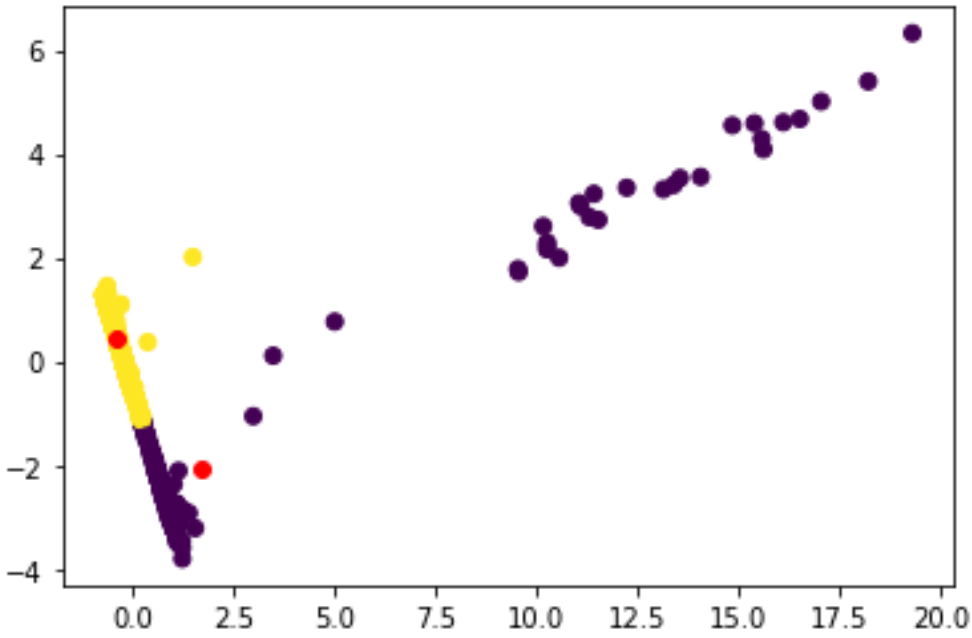

Figure 5.3: Spectral clustering results

At first glance, this type of clustering proved to be the worst of the three, showing very different clusters on the plot. In the next considerations the performances index of the three will be discussed more deeply.

5.1.2 Performance evaluation

Clustering is a type of *unsupervised learning*, so it is not possible to compare the results with labels like in *supervised learning* problems; hence other techniques are used to evaluate the results. Therefore evaluating the performance of a clustering algorithm is not as trivial as counting the number of errors or the precision and recall of a supervised classification algorithm. Clusters are evaluated on some similarity or dissimilarity measure such as the distance between cluster points. If the clustering algorithm separates dissimilar observations apart and similar observations together, then it has performed well. [20]

Silhouette Score

One of the most used performance metrics is the **Silhouette score**. The *Silhouette Score* and *Silhouette Plot* are used to measure the separation distance between clusters. It displays a measure of how close each point in a cluster is to points in

the neighboring clusters. This measure has a range of $[-1, 1]$ and is a great tool to visually inspect the similarities within clusters and differences across clusters. The coefficient can be calculated as:

$$\frac{n - i}{\max(i, n)} \tag{5.1}$$

Where:

- n is the distance between each sample and the nearest cluster that the sample is not belonging
- i is the mean distance within each cluster.

Source:

The higher the Silhouette Coefficients (the closer to +1), the further away the cluster's samples are from the neighboring clusters samples. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters. Negative values, instead, indicate that those samples might have been assigned to the wrong cluster. Averaging the Silhouette Coefficients, it's possible to get to a global Silhouette Score which can be used to describe the entire population's performance with a single value. [21]

Rand Index

Another metric taken into account for evaluating the results can be the **Rand Index**, a measure of the similarity between two data clusterings. It is equal to one if all the pairs fall into the same cluster of both the estimated and the theoretical grouping. The index decreases and tends to zero as the pairs are allocated in different clusters in the two groupings. Rand index R is:

$$R = \frac{a + b}{a + b + c + d} = \frac{a + b}{\binom{n}{2}} \tag{5.2}$$

Where, given a set of n elements in $S = \{o_1, \cdots, o_n\}$ and two *partitions* of S to compare, $X = \{X_1, \cdots, X_n\}$, a partition of S into r subsets, and $Y = \{Y_1, \cdots, Y_s\}$ a partition of S into s subsets and:

- a is the number of pairs of elements in S that are in the **same** subset in X and in the **same** subset in Y
- b is the number of pairs of elements in S that are in **different** subsets in X and in **different** subsets in Y

- c is the number of pairs of elements in S that are in the **same** subset in X and in in **different** subsets in Y
- d is the number of pairs of elements in S that are in **different** subsets in X and in the **same** subset in Y

Calinski-Harabasz Index

The **Calinski-Harabasz Index** is another clustering performance metrics, based on the degree of dispersion between clusters and clusters. It is also known as the *Variance Ratio Criterion*, and it is calculated as a ratio of the sum of inter-cluster dispersion and the sum of intra-cluster dispersion for all clusters (where the dispersion is the sum of squared distances). Theoretically, higher the CH index is, better the clustering is, since observations in each cluster are *denser* (closer), while clusters are further away from each other, and so well separated.
In order to calculate the CH Index, it is needed to calculate the inter-cluster dispersion or the between group sum of squares (**BGSS**), that measures the weighted sum of squared distances between the centroids of a clusters and the centroid of the whole dataset (barycenter):

$$BGSS = \sum_{k=1}^{K} n_k \times ||C_k \breve{} C||^2 \tag{5.3}$$

where:

- n_k is the number of observations in cluster k
- C_k is the centroid of cluster k
- C is the centroid of the dataset (barycenter)
- K is the number of clusters

Then the intra-cluster dispersion or the within group sum of squares (**WGSS**), which indicates the sum of squared distances between each observation and the centroid of the same cluster, is being calculated:

$$WGSS_k = \sum_{i=1}^{n_k} ||X_{ik} \breve{} C_k||^2 \tag{5.4}$$

where:

- n_k is the number of observations in cluster k
- C_k is the centroid of cluster k

- X_{ik} is the i-th observation of the cluster k

Then each individual within group sum of squares is summed:

$$WGSS = \sum_{k=1}^{K} WGSS_k \tag{5.5}$$

Finally, the Calinski-Harabasz Index is calculated as:

$$CH = \frac{\frac{BGSS}{K-1}}{\frac{WGSS}{N-K}} = \frac{BGSS}{WGSS} \times \frac{N-K}{K-1} \tag{5.6}$$

where:

- $BGSS$ is the between-group sum of squares (between-group dispersion)
- $WGSS$ is the within-group sum of squares (within-group dispersion)
- N is the total number of observations
- K is the total number of clusters

Hence, the large values of Calinski-Harabasz index represent better clustering.

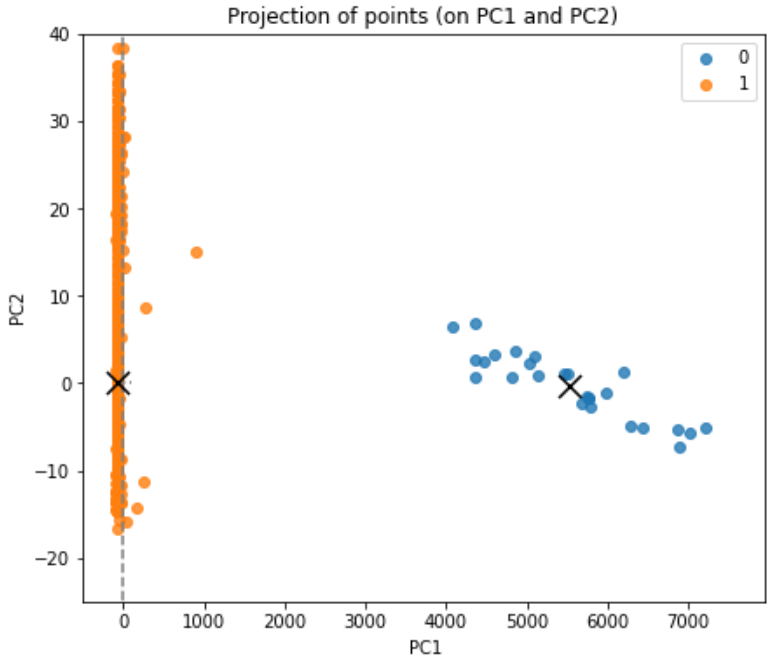

Figure 5.4: Cluster points

Manual Comparison

In this particular case, the original labeled datasets are available for comparison, so, to evaluate the performance of the clustering, several combinations of features and dataset can be compared. The comparisons are made between the "clustered" dataset and the original, manually labeled, dataset. Also the software to compare the two files is made with python into a Jupiter notebook. A very accurate model was obtained using k-means and only the features "ART" and "CPU", with a correspondence of $\approx 99\%$ for either the *k-means* and the *Agglomerative Hierarchical Clustering*. The files were compared with a python script specially created to mark the differences and evaluate the correspondence rate. Also the comparisons made using the two principal components in the clustering phase returned high correspondence, with a slightly smaller value.

5.2 Prediction with clustering

Despite **k-means** is an unsupervised learning algorithm, it offers a predict method. Given some input data, it predicts the closest cluster each sample in X belongs to. It is supposed to point out closest cluster for the unseen data, so it can be used once the model is trained. The dataset realized before with the clustering technique was then loaded in the classifier used for the supervised learning part. With no surprise, being the dataset nearly identical to the previous one, the results obtained were the same as the ones gotten with the manually labeled dataset.

5.3 Clustering final considerations

The employed clustering techniques, i.e. k-means, hierarchical and spectral clustering, returned different results, that were analyzed using the performance metrics described before. In particular, it is to say that k-means gave the best overall performance according to the silhouette score and and the Calinski-Harabasz Index, although not very different from the hierarchical results. The spectral clustering instead returned the worst results, as predictable from the plot, but the possibility to improve by doing some tuning it is not to exclude. These results highlights that there are several unsupervised algorithms and many performance metrics that can be used, but it is essential to know well the data in order to fit the adequate method. In fact, even if for this kind of datasets the best algorithms revealed to be those, for different types of data are probably others, maybe determined using different metrics in order to choose the most appropriate one.

Chapter 6

Conclusions

This thesis work tried to realize a Machine Learning algorithm capable of autonomously identify and detect the application status monitored by the APM organizational unit. To this end, two different approaches were adopted: **Supervised** and **Unsupervised Learning**. Both can be used in real use case scenarios, depending on the type of monitoring required. A mathematical and theoretical explanation of the various Machine Learning types was given, highlighting the necessary characteristics to know in order to understand the work carried out. The Supervised Learning consisted in the realization of a classifier (created in two different libraries) able to distinguish the application state after being trained on a manually exported dataset. Then the various performance metrics were measured and compared, in order to tune and improve the network. The export, filtering and choice of the dataset was not a trivial problem: in fact, this study highlighted how the number of features and their correlation influenced the results. The initial versions of the datasets brought to misleading results, when tried on classifiers or other algorithms that performed well on other reference dataset. The datasets quality was also reflected in the unsupervised learning phase, that, once the right features and dataset settings were chosen, returned good results. In particular three different types of clustering were tested: k-means, hierarchical and spectral. Also in this case, the performance metrics were described from a mathematical and theoretical point view, and then tested in a practical way. This system, after building a stand-alone, deployable version, is capable to read in real time the data delivered by the APM agents. The next evolution of the system could be also able to identify and detect the causes of errors in the applications, but, in order to do so, tens of months of data recording and labeling are required.

Bibliography

[1] Karl-Rudolf Koch. «Bayes' theorem». In: *Bayesian Inference with Geodetic Applications* (1990), pp. 4–8 (cit. on p. 1).

[2] IBM Cloud Education. *Supervised Learning*. URL: https://www.ibm.com/cloud/learn/supervised-learning/ (cit. on p. 3).

[3] Jasmina Dj Novaković, Alempije Veljović, Siniša S Ilić, Željko Papić, and Tomović Milica. «Evaluation of classification models in machine learning». In: *Theory and Applications of Mathematics & Computer Science* 7.1 (2017), pp. 39–46 (cit. on p. 4).

[4] IBM Cloud Education. *Machine Learning Deep Learning Architectures*. URL: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ (cit. on p. 9).

[5] IBM Cloud Education. *Unsupervised Learning*. URL: https://www.ibm.com/cloud/learn/unsupervised-learning (cit. on p. 10).

[6] Chaitanya Reddy Patlolla. *Understanding the concept of Hierarchical clustering Technique*. URL: https://towardsdatascience.com/understanding-the-concept-of-hierarchical-clustering-technique-c6e8243758ec (cit. on p. 14).

[7] Stephanie Glen. *Hierarchical Clustering / Dendrogram: Simple Definition, Examples*. URL: https://www.statisticshowto.com/hierarchical-clustering/ (cit. on p. 14).

[8] Fionn Murtagh and Pedro Contreras. «Algorithms for hierarchical clustering: an overview». In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), pp. 86–97 (cit. on p. 14).

[9] William Fleshman. *Spectral Clustering, Foundation and Application*. URL: https://towardsdatascience.com/spectral-clustering-aba2640c0d5b (cit. on p. 17).

[10] Alind Gupta. *Spectral Clustering*. URL: https://www.geeksforgeeks.org/ml-spectral-clustering/ (cit. on p. 17).

[11] Neerja Doshi. *Spectral clustering, the intuition and math behind how it works!* URL: https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7 (cit. on p. 17).

[12] Xiao-Dong Zhang. «The Laplacian eigenvalues of graphs: a survey». In: (2011). DOI: 10.48550/ARXIV.1111.2897. URL: https://arxiv.org/abs/1111.2897 (cit. on p. 17).

[13] Sarang Narkhede. *Understanding AUC - ROC Curve*. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (cit. on p. 24).

[14] Kartik Nighania. *Various ways to evaluate a machine learning model's performance*. URL: https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15 (cit. on p. 24).

[15] Google Developers. *Classification: ROC Curve and AUC*. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (cit. on p. 24).

[16] Ian T Jolliffe and Jorge Cadima. «Principal component analysis: a review and recent developments». In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202 (cit. on p. 26).

[17] Sebastian Raschka. *Implementing a Principal Component Analysis (PCA)*. URL: https://sebastianraschka.com/Articles/2014_pca_step_by_step.html (cit. on p. 26).

[18] Berman H.B. *Variance-Covariance Matrix*. URL: https://stattrek.com/matrix-algebra/covariance-matrix (cit. on p. 26).

[19] Sebastian Raschka. «An overview of general performance metrics of binary classifier systems». In: *arXiv preprint arXiv:1410.5330* (2014) (cit. on p. 29).

[20] Manish Pathak. *Evaluation Metrics for Supervised and Unsupervised Machine Learning*. URL: https://www.analyticsvidhya.com/blog/2020/10/quick-guide-to-evaluation-metrics-for-supervised-and-unsupervised-machine-learning/ (cit. on p. 36).

[21] Eugenio Zuccarelli. *Performance Metrics in Machine Learning: Clustering*. URL: https://towardsdatascience.com/performance-metrics-in-machine-learning-part-3-clustering-d69550662dc6 (cit. on p. 37).