TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

PERIODO:

AGOSTO-DICIEMBRE 2020

ING. SISTEMAS COMPUTACIONALES

Materia:
Datos masivos

Bloque 3

Proyecto.

Docente:
José Christian Romero Hernández

Alumno:
Jonathan Hernández Iglesias

Index

Introduction

Machine learning algorithms (Machine learning) are very useful tools for solving problems that can be used by all types of companies that generate data and that these are useful to them, in everyday life there are many examples of the generation of this type of data, for example when you take a survey or when you connect for the first time to your streaming service and ask your preferences, the use of these algorithms has become fundamental pillars for companies to know and predict their public in many ways; In the following document, a comparison between 4 algorithms will be made, the comparison will consist of evaluating their precision in the model when evaluating the data of a dataset of the marketing department of a Portuguese bank in which calls are made, the information of the dataset is as follows:

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5 - default: has credit in default? (categorical: 'no','yes','unknown')
6 - housing: has housing loan? (categorical: 'no','yes','unknown')
7 - loan: has personal loan? (categorical: 'no','yes','unknown')
# related with the last contact of the current campaign:
8 - contact: contact communication type (categorical: 'cellular','telephone')
9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10 - day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
# other attributes:
12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
# social and economic context attributes
16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)
17 - cons.price.idx: consumer price index - monthly indicator (numeric)
18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19 - euribor3m: euribor 3 month rate - daily indicator (numeric)
20 - nr.employed: number of employees - quarterly indicator (numeric)

Con esto debemos obtener la predicción en cuanto a si el cliente estará suscrito a un depósito de término

21 - y - has the client subscribed a term deposit? (binary: 'yes','no')

A brief explanation of what each algorithm does will be given throughout the document.

Theoretical framework

SVM

The support vector machine classification-regression algorithm was developed in the 1990s, within the field of computational science. Although it was originally developed as a binary classification method, its application has been extended to multiple classification and regression problems. SVMs have turned out to be one of the best classifiers for a wide range of situations, which is why it is considered one of the benchmarks within the field of statistical learning and machine learning [1]
Support Vector Machines allow finding the optimal way to classify between various classes. Optimal classification is done by maximizing the separation margin between classes. The vectors that define the edge of this separation are the support vectors [2], the use of SVM extends to many areas but these are some examples in which we can use SVM:

- Optical character recognition
- Face detection for digital cameras to focus correctly
- Spam filters for email
- Image recognition on board satellites (knowing which parts of an image have clouds, land, water, ice, etc.)

Using SVM has many advantages:

- Effective in high-dimension spaces.
- It is still effective in cases where the number of dimensions is greater than the number of samples.
- It uses a subset of training points as a decision function (called support vectors), so it is also memory efficient.
- Versatile - Different kernel functions can be specified for the decision function. Common kernels are provided, but custom kernels can also be specified [3].

But some disadvantages are:

- If the number of entities is much larger than the number of samples, avoid over-tuning in choosing the core functions and the regularization term is crucial.
- SVMs do not directly provide probability estimates, they are calculated using an expensive five-fold cross-validation (see Scores and Probabilities, below) [3].

To use SVM this is based on the maximum margin hyperplane.

Equation of a hyperplane [4]

- **x=w0 + w1a1+w2a2+… +wkak =w·a**

Maximum margin hyperplane equation in terms of support vectors (dual formulation) [5]

- **x=b +Σi vector soporte αi yi a(i)·a**

- **x=b +Σi vector soporte αi yi a(i)·a**

With:

- **b,αi** , numerical parameters to be determined
- **a,** instance (to be classified)
- **a(i)**, i-th bracket vector.
- **yi** , class of a (i) (which is also an instance, of training), with values +1, -1
- obtaining of b,αi :
    - Quadratic optimization problem with constraints
    - There are no local minimums
    - Optimization tools
    - Specific algorithms for more efficient SVM training
- Assuming linearly separable classes

Decision tree

The creators of the classification tree methodology with application to machine learning, also called CART methodology, were Leo Breiman, Jerome Friedman, Richard Olshen and Charles Stone. Its application in the field of Statistics began in 1984.
Machine learning algorithms are classified into two types:

- Supervised.
- Not Supervised.

A decision tree is a supervised machine learning algorithm because for it to learn the model we need a dependent variable in the training set.
The structure of a decision tree is as follows:
Decision trees are made up of nodes and are read from top to bottom.

Within a decision tree we distinguish different types of nodes:

- First node or root node: in it the first division occurs based on the most important variable.
- Internal or intermediate nodes: after the first division we find these nodes, which divide the data set again according to the variables.
- Terminal nodes or leaves: they are located in the lower part of the diagram and their function is to indicate the final classification.

Another concept that you should be clear about is the depth of a tree, which is determined by the maximum number of nodes in a branch.

Advantage

- They are easy to construct, interpret and visualize.
- Select the most important variables and not always use all predictors in their creation.
- If data is missing, we will not be able to traverse the tree to a terminal node, but we can make predictions by averaging the leaves of the sub-tree that we reach.

- It is not necessary that a series of assumptions are fulfilled as in linear regression (linearity, normality of the residuals, homogeneity of the variance, etc.).
- They serve both for qualitative and quantitative dependent variables, as well as for numerical and categorical predictive or independent variables. Also, you don't need dummy variables, although they sometimes improve the model.
- They allow non-linear relationships between the explanatory variables and the dependent variable.
- We can use them to categorize numerical variables.

Disadvantages

- They tend to overfitting the data, so the model when predicting new cases does not estimate with the same success rate.
- They are influenced by outliers, creating trees with very deep branches that do not predict well for new cases. These outliers must be eliminated.
- They are not usually very efficient with regression models.
- Creating trees that are too complex can lead to poor adaptation to new data. The complexity reduces the capacity for interpretation.
- Skewed trees can be created if one of the classes is more numerous than the other.
- Information is lost when they are used to categorize a continuous numeric variable.

The creation of a decision tree for a classification problem is carried out by applying Hunt's algorithm, which is based on the division into sub-sets that seek an optimal separation. Given a set of training records of a node, if they belong to the same class it is considered a terminal node, but if they belong to several classes, the data is divided into smaller sub-sets based on a variable and the process is repeated.

To select which variable to choose to obtain the best division, the Classification Error, the Gini index (rpart) or the Entropy (C50) can be considered.

The Gini index measures the degree of purity of a node. It measures the probability of not removing two records of the same class from the node. The higher the Gini index, the lower the purity, so we will select the variable with the lowest weighted Gini. It usually selects unbalanced divisions, where it normally isolates a majority class in a node and classifies the rest of the classes in other nodes. [6]

Logistic Regression

Logistic regression is a statistical tool developed in the 60's, it is a simple but very useful algorithm and it can be adapted for a multiclass classification, the basis of regression systems is to predict qualitative characteristics of an object based on The data that we could have, which must be known variables, these can be qualitative or qualitative values that must act as dependent variables. An example of this could be classifying the sex of a black widow by her size.

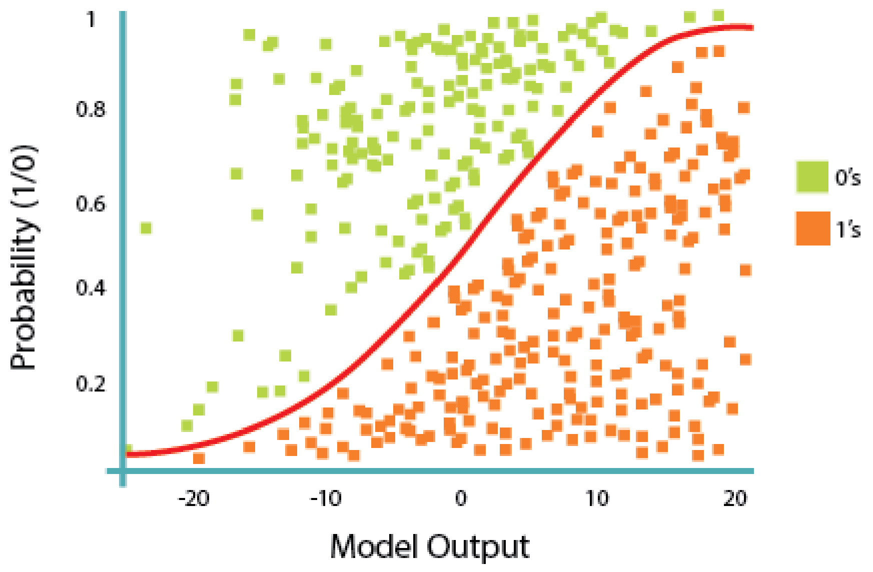


Imagen1.-Regresión logística

The logistic regression is divided into 2 types:

- Binary logistic regression.
  - This makes use of known variables, whether qualitative or quantitative, to act as the variables that will be used to determine the characteristics that we need to predict a third variable that can only have an answer within the parameters 0 or 1.
- Multinomial logistic regression.
  - Multinomial regression also makes use of qualitative or quantitative variables to predict data, but in this case we not only have 2 classes 0 or 1, in this case we can have more of these classes.

Through logistic regression, the aim is to obtain the probability that an event occurs based on variables that we determine are important to predict the new one, these must be relevant to be able, for example, if we have a human being and we want to classify it in a biological gender, We can do it if we have information about the reproductive system, its complexion, etc; with these data we could predict their sex [7].

The binary logistic regression algorithm is a subcategory of the logistic regression algorithms, it refers to binary because it is one in which you can only have two final values, 0 or 1, false or true, its use can be given when we are interested in knowing the influence of a set of variables to predict an output variable, its use is very useful when we want to know a third variable. [8]

If we have a variable in which we only have two possible events in which a person lives or dies, these will be our dependent variables and we want to know the effect of other independent variables such as age or smoking, the logistic regression model will help us to calculate the next:

- Given the values of the independent variables, estimate the probability that the event of interest occurs (for example, becoming ill).
- We can evaluate the influence that each independent variable has on the response, in the form of OR.
- An OR greater than one indicates an increase in the probability of the event and an OR less than one indicates a decrease.

But to build a binary logistic regression model we need:

- A set of independent variables.
- A response variable that only has two values 0 or 1. Here it differs from the multiple regression model, where the response variable is numeric.

The multinomial logistic regression algorithm is used in models with dependent variables of the numerical type that can be divided into more than two categories, while the binary can only be done in 2 categories, being polyatomic, it is an extension of the binary version of logistic regression , the variables can be predictive with which we will help us to predict the outcome. [8]

Multilayer perceptron

Later work with multilayer perceptrons has shown that they are able to approximate an XOR operator, as well as many other non-linear functions.
Just as Rosenblatt based the perceptron on a McCulloch-Pitts neuron, conceived in 1943, perceptrons themselves are building blocks that only prove useful in functions as large as multilayer perceptrons; The multilayer perceptron is the hello world of deep learning: a good place to start when you're learning about deep learning. [9]
A multilayer perceptron (MLP) is a deep artificial neural network. It is made up of more than one perceptron. They are composed of an input layer to receive the signal, an output layer that makes a decision or prediction about the input, and between those two, an arbitrary number of hidden layers that are the true computational engine of the MLP. MLPs with a hidden layer are able to approximate any continuous function.
Feedback networks like MLPs are like tennis or ping pong. They are mainly involved in two movements, a constant coming and going. You can think of this guess-and-answer ping pong as a kind of fast-paced science, since every guess is a test of what we think we know, and every answer is feedback that lets us know how wrong we are.
The architecture of the multilayer Perceptron is characterized by having its neurons grouped in layers of different levels. Each of the layers is made up of a set of neurons and there are three different types of layers: the input layer, the hidden layers and the output layer.
The neurons of the input layer do not act as neurons themselves, but are only responsible for receiving signals or patterns from outside and propagating these signals to all neurons in the next layer. The last layer acts as the network's output, providing the network's response to each of the input patterns to the outside. Neurons in hidden layers perform nonlinear processing of received patterns. [10]
The multilayer perceptron evolves the simple perceptron and for this it incorporates layers of hidden neurons, with this it manages to represent non-linear functions.

The multilayer perceptron is made up of an input layer, an output layer, and n hidden layers in between; It is characterized by having disjoint but related outputs, in such a way that the output of one neuron is the input of the next.
In the multilayer perceptron, about 2 phases can be differentiated:

- Spread.
    - In which the output result of the network is calculated from the input values forward.
- Learning.
    - In which the errors obtained at the output of the perceptron are propagated backwards (backpropagation) in order to modify the weights of the connections so that the estimated value of the network increasingly resembles the real one, this approximation is carried out by the gradient function of the error.

Implementation

For the implementation of the algorithms we used scala and together with the spark framework tools, the Scala programming language was used.
Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and securely typed way. Easily integrate features of functional and object-oriented languages.
Scala is a purely object-oriented language in the sense that everything is an object. Object types and behaviors are described by classes and traits (which could be translated as a "trait"). Classes can be extended through subclasses and a flexible compounding mechanism that provides a clear replacement for multiple inheritance.
Scala is also a functional language in the sense that every function is a value. Scala provides a lightweight syntax for defining anonymous functions. It supports higher-order functions, allows nested functions, and supports currying. Scala's Case classes and constructs built into the pattern recognition language model algebraic types used in many functional programming languages.
In practice, developing domain-specific applications generally requires “Domain Specific Languages” (DSL). Scala provides a unique combination of language mechanisms that simplify the creation of language-specific constructs in the form of libraries:
any method can be used as an infix or postfix operator
closures are built automatically depending on the expected type (target types).
The joint use of both characteristics facilitates the definition of new sentences without having to extend the syntax and without using meta-programming features such as macros. [11]
All these advantages help us to work on big data.
Along with the implementation of spark that provides us with great tools to be able to carry out great jobs in big data.
Apache Spark is today one of the most influential and important technologies in the world of Big Data. It is a computational system of open clusters, unified, ultra-fast analysis engine for Big Data and Machine Learning, these are advantages that spark offers us.

Speed

Spark can be 100 times faster than Hadoop for large-scale data processing by exploiting in-memory computing and other optimizations. It is also fast when data is stored to disk, and currently holds the world record for large-scale disk sorting.

Easy to use

Spark has easy-to-use APIs for operating on large data sets. This includes a collection of more than 100 operators for transforming data and familiar data frame APIs for manipulating semi-structured data. APIs such as Java, Scala, Python and R. It is also known for its ease of use when creating algorithms that acquire all the knowledge of very complex data.

A unified engine

Spark comes bundled with top-level libraries, including support for SQL queries, data streaming, machine learning, and graphics processing. These standard libraries increase developer productivity and can be seamlessly combined to create complex workflows.

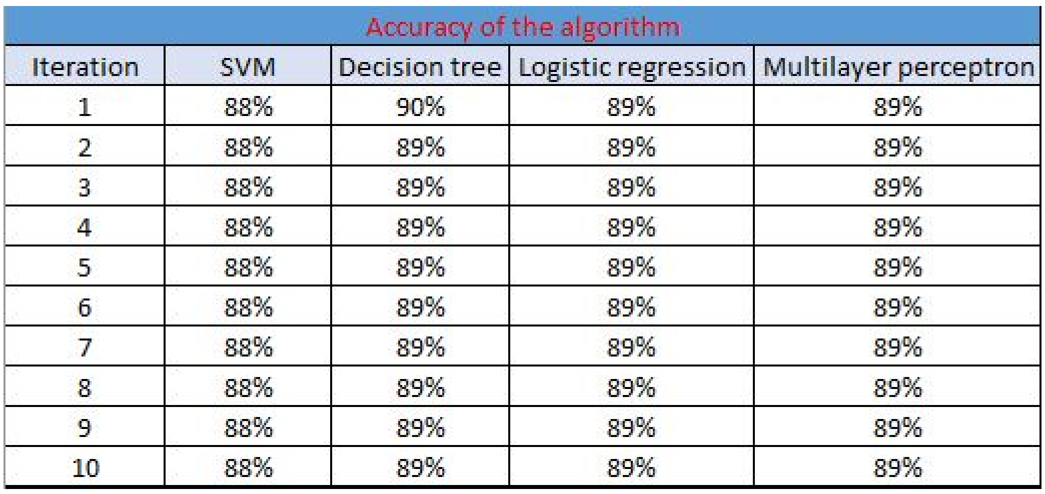
Apache Spark consists of:

- **Spark SQL**: Structured and semi-structured data processing module. With this, it will be possible to transform and perform operations on RDDs or dataframes. Special for the treatment of data.
- **Spark Core**: Core of the framework. It is the base of libraries where the rest of the modules are supported.
- **Spark MLLib**: It is a very complete library that contains numerous Machine Learning algorithms, both for clustering, classification, regression, etc. It allows us, in a friendly way, to be able to use Machine Learning algorithms.
- **Spark Streaming**: It is the one that allows the ingestion of data in real time. If we have a source, for example Kafka or Twitter, with this module we can enter the data from that source and dump it to a destination. Between the data ingestion and its subsequent dump, we can have a series of transformations.
- **Spark Graph**: Allows graph processing (DAG). It does not allow you to paint graphs, but rather it allows you to create operations with graphs, with their nodes and edges, and carry out operations. [12]

for all these advantages is that scala and spark are used for this development.

Results

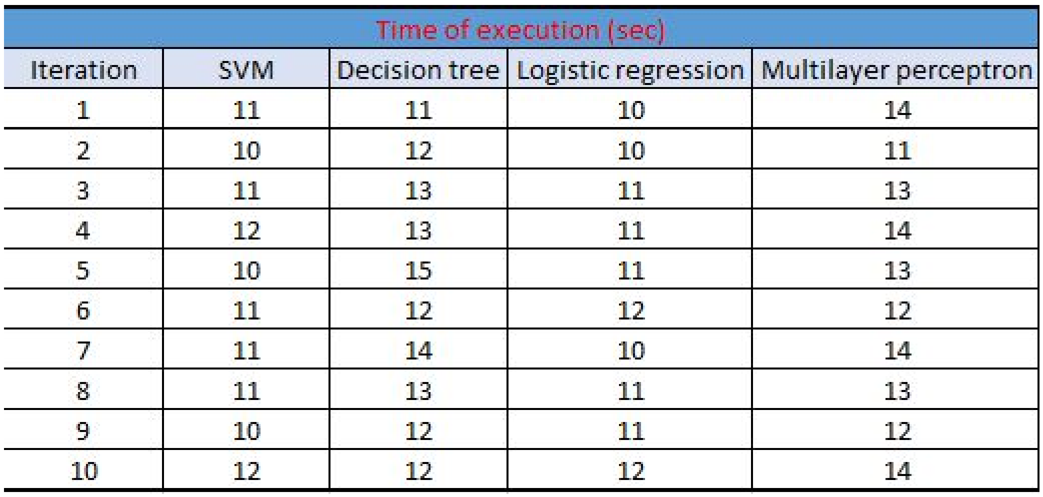
Below are 2 tables which show the performance of each algorithm in the project, the first table shows the precision of the algorithm with the data from the bank-full.csv dataset, as we can see in the table, the 4 algorithms have a Very similar estimated precision, in this case we can infer that the decision tree is one of the best techniques given that it has a maximum precision of 90% in the first iteration but within the next 9 iterations it drops to 89% as the techniques of logistic regression and multilayer perceptron having a tie with these techniques, lastly we have the SVM which has a precision of 88% which does not vary in any iteration, the 4 algorithms are very stable in terms of the precision of the model since varies with time but the one that had a point above it was the decision tree but with this we must also check the execution time with which we will have a better idea of which s the best algorithm.



Accuracy of the algorithm				
Iteration	SVM	Decision tree	Logistic regression	Multilayer perceptron
1	88%	90%	89%	89%
2	88%	89%	89%	89%
3	88%	89%	89%	89%
4	88%	89%	89%	89%
5	88%	89%	89%	89%
6	88%	89%	89%	89%
7	88%	89%	89%	89%
8	88%	89%	89%	89%
9	88%	89%	89%	89%
10	88%	89%	89%	89%

The following table shows the execution times of each algorithm with a maximum iteration rate of 10, according to the data in the table at a glance we can infer that the decision tree algorithm takes the longest time in an iteration. It takes 15 seconds but if we review the table in detail, the multilayer perceptron algorithm is the one that takes the longest since its average time is 13 seconds while those with a lower average time is the SVM and the logistic regression algorithm which takes 10.9 seconds. , we can make a rank of these 4 being as follows.

1.Logistic regression: 10.9 seconds.
2.SVM: 10.9 seconds.
3.Decision tree: 12.7 seconds.
4.Multilayer perceptron: 13 seconds.



Time of execution (sec)				
Iteration	SVM	Decision tree	Logistic regression	Multilayer perceptron
1	11	11	10	14
2	10	12	10	11
3	11	13	11	13
4	12	13	11	14
5	10	15	11	13
6	11	12	12	12
7	11	14	10	14
8	11	13	11	13
9	10	12	11	12
10	12	12	12	14

According to the previous data tables, they all have a similar performance but according to the execution time and its level of precision, they would be as follows,

1.Logistic regression.
2.SVM.
3.Decision tree
4.Multilayer perceptron.

The first place is the logistic regression algorithm that has a shorter execution time and its level of precision is one of the highest and in these times of speed, time is everything.
In second place was the SVM because its average execution time is the lowest but its precision is one point lower than all.
Third, we have the decision tree as it has good precision but its average execution time is very high.
In last place we have the multilayer perceptron, this had the average precision to the others but it is the one that had the highest average execution time.

Conclusions

The classification and regression algorithms that we compare in this document are very useful in many areas of daily life, but carrying out this project helps us to know which ones are easier to use and which are better, in this case they all fulfill their function but some have a better performance than others in my case I can say that the easiest to elaborate is the logistic regression because it is not complicated to understand or implement, while the multilayer perceptron was one of the most complicated but once you understand them, all are be simple to use and implement.
To implement any of these algorithms it is necessary to evaluate what we are looking for to know which is the best that we can implement.

References

[1] Moro, S., Cortez, P., & Rita, P. (2014, junio). UCI Machine Learning Repository: Bank Marketing Data Set. UCI. https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
[2]Heras, J. M. (2019b, mayo 28). Máquinas de Vectores de Soporte (SVM). IArtificial.net. https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/
[3]Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, págs. 2825-2830, 2011.
[4]N. Cristianini and J. Shawe-Taylor. An introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.
[5]Ben-Hur A, Ong CS, Sonnenburg S, Schölkopf B, Rätsch G, 2008 Support Vector Machines and Kernels for Computational Biology. PLoS Comput Biol 4(10): e1000173. doi:10.1371/journal.pcbi.1000173
[6]Merayo, P. (2020, 26 mayo). Qué son los árboles de decisión y para qué sirven. Máxima Formación.
https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/
[7]Perez, A. J., Kizys, R., & Manzanedo Del Hoyo, L. M. (2010, enero). Regresion logistica Binaria. MECD. https://econometriai.files.wordpress.com/2010/01/reg-logistica.pdf

[8]Rodrigo, J. A. (2016, agosto). Regresión logística simple y múltiple. cienciadedatos. https://www.cienciadedatos.net/documentos/27_regresion_logistica_simple_y_multiple.html
[9]Honkela, A. (2001, 30 mayo). Multilayer perceptrons. aalto. http://users.ics.aalto.fi/ahonkela/dippa/node41.html
[10]Nicholson, C. (s. f.). A Beginner's Guide to Multilayer Perceptrons (MLP). Pathmind. Recuperado 19 de diciembre de 2020, de https://wiki.pathmind.com/multilayer-perceptron
[11]Scala. (s. f.). Introducción. Scala Documentation. Recuperado 3 de enero de 2021, de https://docs.scala-lang.org/es/tour/tour-of-scala.html
[12]Ilabaca, S. (2019,Febrero). ¿Qué es apache spark?. Recuperado 3 de enero de 2020. https://www.analytics10.com/que-es-apache-spark/

Código

/------- S V M -------//

```
// we import the libraries to be able to perform the SVM
import org.apache.spark.sql.SparkSession
import org.apache.log4j._
import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.Pipeline
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.ml.classification.LinearSVC

// Error log
Logger.getLogger("org").setLevel(Level.ERROR)

// We create our spark session to start using spark
val spark = SparkSession.builder().getOrCreate()

// We must load the data from the dataset "bank-full.csv"
val data =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("
csv").load("bank-full.csv")

// We make a vector to store the columns that will be used.
val assembler = new
VectorAssembler().setInputCols(Array("age","balance","day","duration","campaign","pdays","
previous")).setOutputCol("features")

// We will use stringindexer to make the data in column Y binary.
val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")

// We divide the data into the training and test sets leaving only 30% of the data for testing
val Array(training, test) = data.randomSplit(Array(0.7, 0.3), seed = 11L)

// We are going to create the linearSVM model with the label and features columns of the
dataset, as well as for the prediction, with 10 iterations
val lsvc = new LinearSVC()
.setLabelCol("label")
.setFeaturesCol("features")
.setPredictionCol("prediction")
.setMaxIter(10)
.setRegParam(0.1)
```

```
// With the help of Pipeline, the estimator will be called in the input data set to fit a model, this
through the indices and thus have the metadata that we require for the operation of the
algorithm.
val pipeline = new Pipeline().setStages(Array(labelIndexer, assembler, lsvc))

// we adjust the model
val model = pipeline.fit(training)

// The results of the test set are taken with transform
val result = model.transform(test)

// Result of the test data set in an RDD for the prediction data
val predictionAndLabelsrdd = result.select($"prediction", $"label").as[(Double, Double)].rdd

// a MulticlassMetrics object is initialized with the previous results for some metrics
val metrics = new MulticlassMetrics(predictionAndLabelsrdd)

println("-SVM-")

// We use the metrics data to print the precision of the algorithm.
println(s"Presicion = ${(metrics.accuracy)}")

// We calculate the execution time of the algorithm.
val time = System.nanoTime
val duration = (System.nanoTime - time) / 1e9d
println("Tiempo de ejecución: " + duration)


//-------  D E C I S I O N   T R E E  -------//

// we import the libraries to be able to make the decision tree
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.IndexToString
import org.apache.log4j._
import org.apache.spark.ml.PipelineStage
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SQLContext
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.VectorAssembler

// Error log
Logger.getLogger("org").setLevel(Level.ERROR)
```

```
// We create our spark session to start using spark
val spark = SparkSession.builder().getOrCreate()

// We must load the data from the dataset "bank-full.csv"
val data =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("
csv").load("bank-full.csv")

// We create the label indexer to assign column Y to the indexes.
val labelIndexer = new
StringIndexer().setInputCol("y").setOutputCol("indexedLabel").fit(data)

// We make a vector to store the columns that will be used.
val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")).setOut
putCol("features")
val features = assembler.transform(data)

// the dataset is identified by feature in a vector to be able to use it.
val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(
4).fit(features)

// We divide the data into the training and test sets leaving only 30% of the data for testing
val Array(trainingD, testD) = features.randomSplit(Array(0.7, 0.3))

// We create a DecisionTree object
val ds = new
DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures")

// we create the branch to create the prediction.
val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(labelInde
xer.labels)

// With the help of Pipeline, the estimator will be called in the input data set to fit a model, this
through the indices and thus have the metadata that we require for the operation of the
algorithm.
val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, ds,
labelConverter))

// we create a training model with the training data.
val model = pipeline.fit(trainingD)

// Data transformation in the model with the test data
val predictions = model.transform(testD)
```

```
// we print the predictions.
predictions.select("predictedLabel", "y", "features").show(5)

println("- Decision Tree -")

// the tree model is generated
val treeModel = model.stages(2).asInstanceOf[DecisionTreeClassificationModel]
println(s"tree model:\n ${treeModel.toDebugString}")

// we calculate the precision of the model with the evaluator data
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredictionCol("prediction"
).setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
println(s"Precision = ${(accuracy)}")

//Execution time
val time1 = System.nanoTime
val duration = (System.nanoTime - time1) / 1e9d
println("Tiempo de ejecución: " + duration)


//-------  L O G I S T I C   R E G R E SS I O N  -------//



// we import the libraries to be able to perform the logistic regression
import org.apache.spark.sql.SparkSession
import org.apache.log4j._
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer,
VectorAssembler}
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.ml.Pipeline

// Error log
Logger.getLogger("org").setLevel(Level.ERROR)

// We create our spark session to start using spark
val spark = SparkSession.builder().getOrCreate()

// We must load the data from the "bank-full.csv" dataset and make it dataframe
val df =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("
csv").load("bank-full.csv")
```

```
// We make a vector to store the columns that will be used.
val assembler = new
VectorAssembler().setInputCols(Array("age","balance","day","duration","campaign","pdays","
previous")).setOutputCol("features")

// We will use stringindexer to make the data in column Y binary.
val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")
val dataIndexed = labelIndexer.fit(df).transform(df)

// We divide the data into the training and test sets leaving only 30% of the data for testing
val Array(training, test) = dataIndexed.randomSplit(Array(0.7, 0.3), seed = 12345)

// we instantiate the new logistic regression
val lr = new LogisticRegression()

// With the help of Pipeline, the estimator will be called in the input data set to fit a model, this
through the indices and thus have the metadata that we require for the operation of the
algorithm.
val pipeline = new Pipeline().setStages(Array(assembler,lr))

// we fit the model with the training data.
val model = pipeline.fit(training)

// we generate the results
val results = model.transform(test)

// we calculate the predictions of the model.
val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd

// We create a metric variable to have the data of the prediction evaluator
val metrics = new MulticlassMetrics(predictionAndLabels)

println("-Logistic Regression -")

// We create the confusion matrix with the predictions of the columns.
println(metrics.confusionMatrix)

// We calculate the precision of the model.
println(s"Precision = ${(metrics.accuracy)}")

// We calculate the execution time of the model.
val time2 = System.nanoTime
val duration = (System.nanoTime - time2) / 1e9d
println("Tiempo de ejecución: " + duration)



//-------  M U L T I L A Y E R   P E R C E P T R O N -------//
```

```
// we import the libraries to be able to perform the perceptron multilayer
import org.apache.spark.sql.SparkSession
import org.apache.log4j._
import org.apache.spark.ml.feature.IndexToString
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.linalg.Vectors

// Error log
Logger.getLogger("org").setLevel(Level.ERROR)

// We create our spark session to start using spark
val spark = SparkSession.builder().getOrCreate()

// We must load the data from the "bank-full.csv" dataset and make it a dataframe.
val df =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("
csv").load("bank-full.csv")

// We make a vector to store the columns that are used.
val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")).setOut
putCol("features")
val features = assembler.transform(df)

// We will use stringindexer to make the data in column Y binary.
val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")
val dataIndexed = labelIndexer.fit(features).transform(features)

// We divide the data into the training and test sets leaving only 30% of the data for testing
val split(training,test) = dataIndexed.randomSplit(Array(0.7, 0.3), seed = 1234L)

// We assign the value to the layers of our model
val layers = Array[Int](5, 2, 3, 2)

// we create the model with the given parameters.
val trainer = new
MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L).setMa
xIter(100)

// We fit the model with the training data.
val model = trainer.fit(training)
```

```
// we create a variable to insert the results of our model
val result = model.transform(test)

// we create predictions with prediction and label columns
val predictionAndLabels = result.select("prediction", "label")
//mostramos los datos calculados
predictionAndLabels.show(10)

println("- Multilayer perceptron -")

// We calculate the precision of our model.
val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
println(s"Accuracy test = ${evaluator.evaluate(predictionAndLabels)}")

// we calculate the execution time
val time3 = System.nanoTime
val duration = (System.nanoTime - time3) / 1e9d
println("Tiempo de ejecución: " + duration)
```