TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

PERIODO:

Enero-junio 2020

Ing. Informática

MATERIA
Datos Masivos

Proyecto Final

ALUMNOS Y NO. DE CONTROL
Anahi Estrada Casillas 15212154
Fernando Ordaz Zamora 17210037

DOCENTE
Christian Romero Hernandez

ENTREGA
Junio 2020

Index

Introduction

This document proposes the use of different machine learning algorithms (machine learning) as they are very common in solving a problem. A brief explanation of what each algorithm consists of as well as application examples will be given.

When comparing the results generated by these 4 machine learning algorithms, we will be able to know with which one is more effective when analyzing the data.

On the other hand we will be able to evaluate the measurement of the precision of the algorithms and the comparison of this precision.

When more than two algorithms are compared, it is necessary to take into account the problems of our datasets so that in each algorithm certain parameters are adjusted to obtain results.

In this document we will take the information from the data that is related to direct marketing campaigns of a Portuguese banking institution.

Marketing campaigns were based on phone calls. Often, more than one contact with the same client was required, to access whether the product (term bank deposit) would be (or not) subscribed [12].

Input variables:

Bank client data:

1 - age (numeric)

2 - job: type of job (categorical: "administrator", "unknown", "unemployed", "management", "domestic worker", "entrepreneur", "student",
"worker", "self-employed", "retired", "technician", "services")

3 - marital: marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)

4 - education (categorical: "unknown", "secondary", "primary", "tertiary")

5 - default: do you have credit in default? (binary: "yes", "no")

6 - balance: average annual balance, in euros (numeric)

7 - home: do you have a home loan? (binary: "yes", "no")

8 - loan: do you have a personal loan? (binary: "yes", "no")

# related to the last contact of the current campaign:

9 - contact: type of contact communication (categorical: "unknown", "telephone", "mobile")

10 days: last contact day of the month (numeric)

11 months: last contact of the year of the year (categorical: "Jan", "Feb", "Mar", ..., "Nov", "Dec")

12 - duration: duration of the last contact, in seconds (numeric)

# other attributes:

13 - campaign: number of contacts made during this campaign and for this customer (numeric, includes the last contact)
14 days: number of days that passed after the customer was contacted for the last time since a previous campaign (numeric, -1 means that the client was not previously contacted)
15 - previous: number of contacts made before this campaign and for this client (numerical)
16 - poutcome: result of the marketing campaign previous (categorical: "unknown", "other", "failure", "success")
Output variable (desired goal):
17 - and - does the client Have you signed a term deposit? (binary: "yes", "no")
The data we use is found in [13].

The algorithms used for this work are the following:
Support Vector Machine (SVM)
Decision Tree
Logistic Regression
Multilayer Perceptron,

Theoretical framework of algorithms

SVM

A support vector machine (SVM) is a supervised learning algorithm that can be used for binary classification or regression. The SVM theory is based on the idea of structural risk minimization in [7] is a brief explanation of this. An SVM first maps the entry points to a feature space of a larger dimension and finds a hyperplane that separates them and maximizes the margin m between classes in this space. The SVM finds the optimal hyperplane using the dot product with functions in the feature space that are called kernels. The optimal hyperplane solution can be written as the combination of a few entry points that are called support vectors. [6] An application example is found in [8], other examples are applications such as natural language processing, speech, image recognition, and machine vision.

Decision Tree

The decision tree classifier is one of the possible approaches to multi-stage decision making; table search rules, conversion of decision table to optimal decision trees and sequential approaches are others. The basic idea involved in any multi-stage approach is to divide a complex decision into a union of several simpler decisions, waiting for the final solution obtained in this way would resemble the desired desired solution. A complete multi-stage review. The recognition schemes are given by Dattatreya and Kanal [6]. Hierarchical classifiers allow the rejection of class labels at intermediate stages.
The steps can be summarized as follows:

1. Dividimos the data in two or more sets based homogeneous differentiator most significant input variables.
2. The decision tree identifies the most significant variable and its value that provides the best homogeneous population sets.
3. All input variables and all possible division points are evaluated and the one with the best result is chosen.

Logistic Regression Logistic Regression

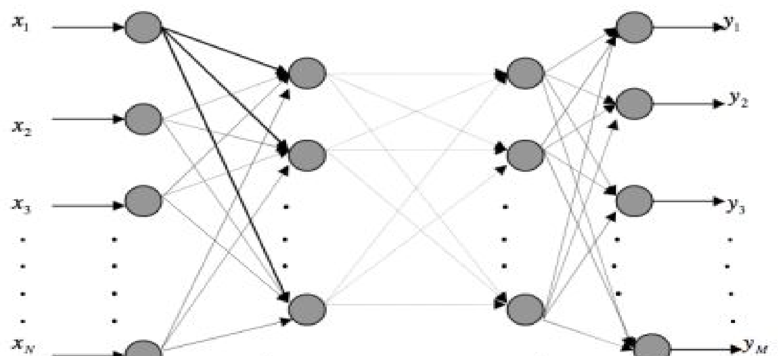
or Logistic Regression is a classification algorithm that is used to predict the probability of a categorical dependent variable (a predictive analysis). In logistic regression, the dependent variable is a binary variable that contains data encoded as 1 - 0, yes - no, open - closed, etc.
Logistic regression is used to describe data and explain the relationship between a dependent binary variable and one or more independent nominal, ordinal, interval, or ratio level variables [11].
As mentioned in the introduction, the fit of this model is important. Adding independent variables to a logistic regression model will always increase the amount of variance explained in the log probabilities (generally expressed as R^2). However, adding more and more variables to the model can result in an overfit, reducing the generalizability of the model beyond the data that the model fits into.
This algorithm helps us to know the probability of whether an example belongs to a class or not, we are going to use the inverse of the logit function and what we can call as the logistic function and in many cases also named as the sigmoid function by a representation in form of S:
$\phi\ (z)$ = 11 + $e - z$
Here we can indicate that z is the input to the network and results from the linear combination of the weights and the respective characteristics of each example, that is, z = wT

x = w0 + w1x1 + ... + wmxm. Each characteristic will have a weight because each one of them will influence more or less for the final decision of which class it belongs to [14].

Multilayer perceptron

The perceptron is very useful for classifying data sets that are linearly separable. They encounter serious limitations with data sets that do not fit this pattern as discovered with the XOR problem. Logic functions typically have two inputs and one output that depends on the values of the inputs. The inputs and outputs can take two values: True and False, or 0 and 1. Thus for each logical function there is a table that indicates what the output will be given the combination of the input values [9].
The multilayer perceptron (MLP) breaks this constraint and classifies data sets that are not linearly separable. The Perceptron consists of an input layer and an output layer that are fully connected. MLPs have the same input and output layers, but can have multiple layers hidden between the layers mentioned above.
Steps to develop the MLP:

1. The inputs are pushed forward through the MLP by taking the input point product with the weights between the input layer and the hidden layer (WH). This point product produces a value in the hidden layer.
2. MLPs use trigger functions in each of their calculated layers. You push the calculated output on the current layer through a trigger function.
3. Once the calculated output in the hidden layer has been pushed through the trigger function, push it to the next layer in the MLP by taking the dot product with the corresponding weights.
4. Repeat steps two and three until you reach the output layer.
5. In the output layer, the calculations will either be used for a backpropagation algorithm that corresponds to the trigger function that was selected for the MLP (in the case of training) or a decision will be made based on the result (in the case of the test) [10].



Implementation

The tools used here are Spark with the Scala language. Below we show what each one consists of.

Scala is a programming language designed to program using patterns in a concise way. In the same way, it integrates principles of object orientation and functional programming, allowing programmers to be more productive.
Among its advantages we can highlight thatis written less code compared to other languages, which allows us to have fewer bugs and make it much easier to read and understand.

Spark is one of the most widely used Big Data technologies worldwide, used by large corporations, small businesses and academia. It is a is a platform that is oriented to handle large volumes of distributed data that offers us several advantages and reduces execution times.
Spark offers us an interactive shell in Scala with which we can experiment and test, testing all the API that the framework offers us.

Code

```
// Big Data Project

//1.- Objective: Comparison of the performance of the following machine learning algorithms

// - SVM

// - Decision Three

// - Logistic Regression

// - Multilayer perceptron


// - With the following data set: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

// The objective of classification is to predict if the client will subscribe a term deposit




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

//Cargamos librerias a utilizar

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

//Reducción de errores

Logger.getLogger("org").setLevel(Level.ERROR)

//Creamos nuestra sesion de spark

val spark = SparkSession.builder().getOrCreate()

// Cargamos nuestro dataset

val data = spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("csv").load("bank-full.csv")

//Creamos un vector donde almacenaremos las columnas que vamos a utilizar con el nombre de features

val assembler = new VectorAssembler().setInputCols(Array("age","balance","day","duration","campaign","pdays","previous")).setOutputCol("features")

//Aplicamos indexer la columna Y para que los valores de si y no los tome como valores numericos y podamos trabajar con ellos

val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")

//utilizamos randomSplit para crear datos de trabajo y test divididos en 70 y 30

val Array(training, test) = data.randomSplit(Array(0.7, 0.3), seed = 11L)

//Utilizamos linearSVM con los caracteristicas y el Label del dataset

val lsvc = new LinearSVC().setLabelCol("label").setFeaturesCol("features").setPredictionCol("prediction").setMaxIter(10).setRegParam(0.1)

//se crea un nuevo pipeline

val pipeline = new Pipeline().setStages(Array(labelIndexer, assembler, lsvc))
```

```
// ajustamos el modelo para entrenar los datos

val model = pipeline.fit(training)

// se transforman los datos del modelo

val result = model.transform(test)

// Generamos predicciones

val predictionAndLabelsrdd = result.select($"prediction", $"label").as[(Double, Double)].rdd

//inicialice un objeto multiclassMetrics

val metrics = new MulticlassMetrics(predictionAndLabelsrdd)

println("-------------- SVM ----------------")

// imprime la precision del algoritmo

println(s"Accuaracy Test = ${(metrics.accuracy)}")

//Tiempo de ejecucion

val t1 = System.nanoTime

val duration = (System.nanoTime - t1) / 1e9d

println("Tiempo de ejecucion: " + duration)

//Memoria usada

val mb = 1024*1024

val runtime = Runtime.getRuntime

println("** Used Memory:  " + (runtime.totalMemory - runtime.freeMemory) / mb)

println("** Free Memory:  " + runtime.freeMemory / mb)

println("** Total Memory: " + runtime.totalMemory / mb)

println("** Max Memory:   " + runtime.maxMemory / mb)
```

```
//////////////////////////////////    D E C I S I O N   T H R E E      //////////////////////////////////////

//Cargamos librerias a utilizar

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


//Reducción de errores

Logger.getLogger("org").setLevel(Level.ERROR)


//Creamos nuestra sesion de spark

val spark = SparkSession.builder().getOrCreate()
```

```
//Importamos nuestro DataSet

val data =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").
format("csv").load("bank-full.csv")



//Creamos nuestro label indexer para comparar

val labelIndexer = new
StringIndexer().setInputCol("y").setOutputCol("indexedLabel").fit(data)



//Inicializamos el vector asembler por datos de tipo numericos y agregamos la
columna features como output

val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")
).setOutputCol("features")



// transformamos los datos

val features = assembler.transform(data)



// Identifica categoricamente nuestro dataset en vector

val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCat
egories(4).fit(features)



//Dividimos nuestro dataset en 70% entrenamiento y 30% de prueba

val Array(trainingData, testData) = features.randomSplit(Array(0.7, 0.3))



//Creamos un objeto DecisionTree
```

```
val dt = new
DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatu
res")


//Rama de prediccion

val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(l
abelIndexer.labels)


//Juntamos los datos en un pipeline

val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, dt,
labelConverter))


//creamos modelo de entrenamiento

val model = pipeline.fit(trainingData)


//Transformacion de datos en el modelo

val predictions = model.transform(testData)


//Desplegamos predicciones

predictions.select("predictedLabel", "y", "features").show(5)


println("-------------- Decision Three ----------------")


//se genera el modelo de arbol

val treeModel = model.stages(2).asInstanceOf[DecisionTreeClassificationModel]
```

```
println(s"Learned classification tree model:\n ${treeModel.toDebugString}")


//Evaluamos la eficiencia del modelo

val evaluator = new MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredictionCol("prediction").setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)

println(s"Accuaracy Test = ${(accuracy)}")


//Tiempo de ejecucion

val t1 = System.nanoTime

val duration = (System.nanoTime - t1) / 1e9d

println("Tiempo de ejecucion: " + duration)


//Memoria usada

val mb = 1024*1024

val runtime = Runtime.getRuntime

println("** Used Memory:  " + (runtime.totalMemory - runtime.freeMemory) / mb)

println("** Free Memory:  " + runtime.freeMemory / mb)

println("** Total Memory: " + runtime.totalMemory / mb)

println("** Max Memory:   " + runtime.maxMemory / mb)
```

```
//////////////////////////////////          L O G I S T I C    R E G R E S I O N
////////////////////////////////////////

// Cargamos librerias

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


// Disminución de errores

Logger.getLogger("org").setLevel(Level.ERROR)


// Creamos sesion de spark

val spark = SparkSession.builder().getOrCreate()


// Cargamos nuestro dataset

val df =
spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").
format("csv").load("bank-full.csv")


//Creamos un vector donde colocaremos las columnas que queremos utilizar
```

```
val assembler = new
VectorAssembler().setInputCols(Array("age","balance","day","duration","campaign","
pdays","previous")).setOutputCol("features")


//Indexamos la columna Y para que los valores de si y no tomen valor de 1 y 0

val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")

val dataIndexed = labelIndexer.fit(df).transform(df)


// Dividimos los datos en 70% y 30% para el entrenamiento y prueba

val Array(training, test) = dataIndexed.randomSplit(Array(0.7, 0.3), seed = 12345)


// creamos nuestro logistic regresion

val lr = new LogisticRegression()


// creamos nuestro pipeline

val pipeline = new Pipeline().setStages(Array(assembler,lr))


// creamos nuestro modelo ingresandole el 70% de los datos

val model = pipeline.fit(training)


// generamos los resultados

val results = model.transform(test)


// Generamos predicciones
```

```
val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd


//creamos nuestro objeto evaluador para las multiclases de prediction and labels

val metrics = new MulticlassMetrics(predictionAndLabels)


println("-------------- Logistic Regresion ----------------")


//Predice las clases que hay en las columnas

println(metrics.confusionMatrix)


//Muestra la eficacia del modelo

println(s"accuaracy Test = ${(metrics.accuracy)}")

println(s"Test Error = ${(1.0 - metrics.accuracy)}")


//Tiempo de ejecucion

val t1 = System.nanoTime

val duration = (System.nanoTime - t1) / 1e9d

println("Tiempo de ejecucion: " + duration)


//Memoria usada

val mb = 1024*1024

val runtime = Runtime.getRuntime

println("** Used Memory:  " + (runtime.totalMemory - runtime.freeMemory) / mb)
println("** Free Memory:  " + runtime.freeMemory / mb)

println("** Total Memory: " + runtime.totalMemory / mb)

println("** Max Memory:   " + runtime.maxMemory / mb)



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



//Cargamos las librerias que utilizaremos

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


//Disminuye errores

Logger.getLogger("org").setLevel(Level.ERROR)
```

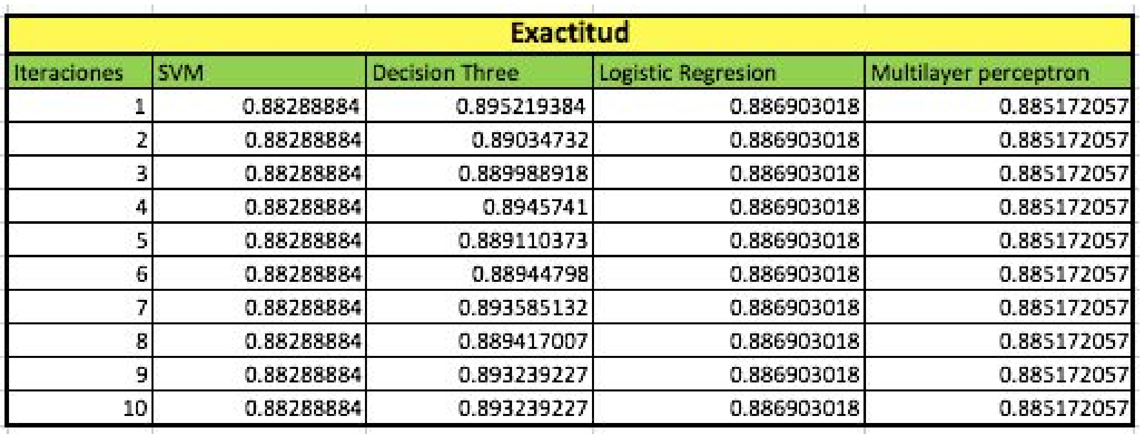
```
//Creamos sesion de spark

val spark = SparkSession.builder().getOrCreate()


//Cargamos nuestro dataset

val df = spark.read.option("header","true").option("inferSchema","true").option("delimiter",";").format("csv").load("bank-full.csv")


//Creamos un vector donde seleccionaremos las columnas a utilizar de nuestro dataset

val assembler = new VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")).setOutputCol("features")

val features = assembler.transform(df)


//Indexamos la columna "y" para poder utilizar los si y no como 0 y 1

val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("label")

val dataIndexed = labelIndexer.fit(features).transform(features)


//Dividimos nuestro dataset en porciones de 70% y 30% para el entrenamiento y prueba de nuestro modelo

val split = dataIndexed.randomSplit(Array(0.7, 0.3), seed = 1234L)


//Asignamos 70% del total al entrenamiento

val train = split(0)
```

```
//Asignamos el 30% del total a las pruebas

val test = split(1)


//Asignamos el valor a las capas de nuestro modelo

val layers = Array[Int](5, 2, 3, 2)


//Creamos un entrenador para nuestro modelo con sus parametros

val trainer = new
MultilayerPerceptronClassifier().setLayers(layers).setBlockSize(128).setSeed(1234L
).setMaxIter(100)


//Entrenamos nuestro modelo con el entrenamiento creado arriba

val model = trainer.fit(train)


//Proyecta resultados de nuestro modelo

val result = model.transform(test)


// creamos predicciones con columnas prediction y label

val predictionAndLabels = result.select("prediction", "label")

//visualizamos

predictionAndLabels.show(10)


println("-------------- Multilayer perceptron ----------------")


//muestra la eficiencia del modelo
```

```
val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")

println(s"Accuracy test = ${evaluator.evaluate(predictionAndLabels)}")


//Tiempo de ejecucion

val t1 = System.nanoTime

val duration = (System.nanoTime - t1) / 1e9d

println("Tiempo de ejecucion: " + duration)


//Memoria usada

val mb = 1024*1024

val runtime = Runtime.getRuntime

println("** Used Memory:  " + (runtime.totalMemory - runtime.freeMemory) / mb)

println("** Free Memory:  " + runtime.freeMemory / mb)

println("** Total Memory: " + runtime.totalMemory / mb)

println("** Max Memory:   " + runtime.maxMemory / mb)
```

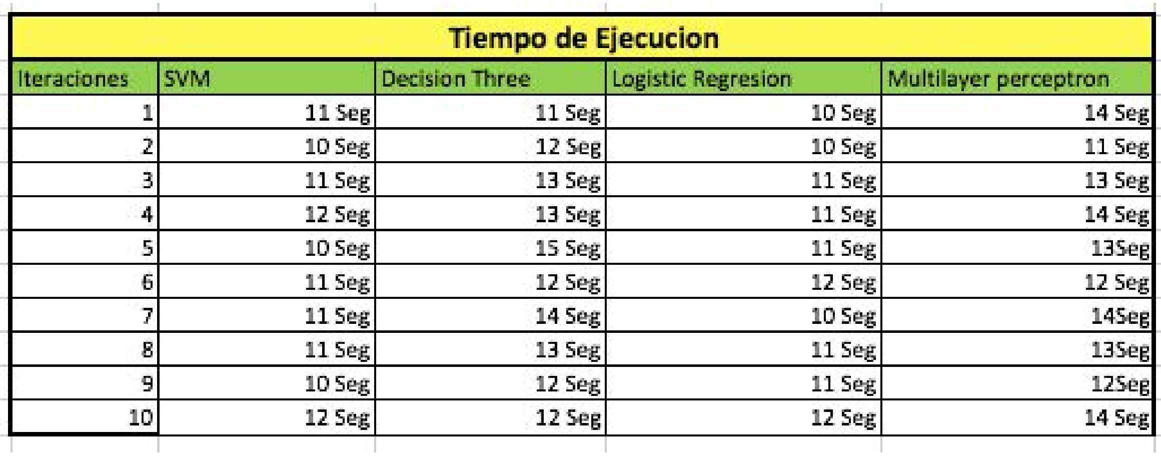
Results

The following table shows the results of 10 iterations with each of the algorithms, the first is SVM, which showed no change with 88% accuracy, the second is Decision Tree in which the results were more noticeable, varying between the 88% and 89% accuracy, in the following two algorithms which are Logistic regression and Multilayer Perceptron there was no change in accuracy, showing their result with 88%.



Exactitud				
Iteraciones	SVM	Decision Three	Logistic Regresion	Multilayer perceptron
1	0.88288884	0.895219384	0.886903018	0.885172057
2	0.88288884	0.89034732	0.886903018	0.885172057
3	0.88288884	0.889988918	0.886903018	0.885172057
4	0.88288884	0.8945741	0.886903018	0.885172057
5	0.88288884	0.889110373	0.886903018	0.885172057
6	0.88288884	0.88944798	0.886903018	0.885172057
7	0.88288884	0.893585132	0.886903018	0.885172057
8	0.88288884	0.889417007	0.886903018	0.885172057
9	0.88288884	0.893239227	0.886903018	0.885172057
10	0.88288884	0.893239227	0.886903018	0.885172057

The following table shows the execution time of each of the algorithms mentioned above.
SVM had a minimal variation between 10 and 12 seconds, dominating the runtime of 11 seconds, Decision Tree from 11 to 15 seconds, Logistic Regression from 10 to 12 seconds, as well as SVM dominating 11 seconds, and Multilayer perceptron from 11-14 seconds.



Tiempo de Ejecucion				
Iteraciones	SVM	Decision Three	Logistic Regresion	Multilayer perceptron
1	11 Seg	11 Seg	10 Seg	14 Seg
2	10 Seg	12 Seg	10 Seg	11 Seg
3	11 Seg	13 Seg	11 Seg	13 Seg
4	12 Seg	13 Seg	11 Seg	14 Seg
5	10 Seg	15 Seg	11 Seg	13Seg
6	11 Seg	12 Seg	12 Seg	12 Seg
7	11 Seg	14 Seg	10 Seg	14Seg
8	11 Seg	13 Seg	11 Seg	13Seg
9	10 Seg	12 Seg	11 Seg	12Seg
10	12 Seg	12 Seg	12 Seg	14 Seg

Conclusions

Through the comparative classification study that was carried out, we were able to identify the precision measure of the algorithms and the comparison of this precision.
The main thing that we observed in the tables is that the Decision Tree algorithm is more effective when analyzing the data with 89% accuracy and the lowest was SVM with 88% accuracy.
On the other hand, we evaluate the execution time in which the one with the least time has SVM and Logistic Regression, both with 11 seconds.

Although machine learning algorithms are very useful, with these tables we can find out which algorithm will fit the dataset and will be more useful for solving our problems.

References

[1] Muñoz López, M. Predictor para el Síndrome de Lynch: Comparativa y análisis de algoritmos de machine learning.
[2]Zubiaga, A., Fresno, V., & Martínez, R. (2009). Comparativa de Aproximaciones a SVM Semisupervisado Multiclase para Clasificación ́on de Páginas Web. *Procesamiento del lenguaje natural*, (42), 63-70.
[3] Garre, M., Cuadrado, J. J., Sicilia, M. A., Rodríguez, D., & Rejas, R. (2007). Comparación de diferentes algoritmos de clustering en la estimación de coste en el desarrollo de software. *REICIS. Revista Española de Innovación, Calidad e Ingeniería del Software*, *3*(1), 6-22.
[4] Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE transactions on systems, man, and cybernetics 21(3), 660–674 (1991)
[5] G. R. Dattatreya and L. N. Kanal, " Decision trees in pattern recognition," In Progress in Pattern Recognition 2, Kanal and Rosenfeld (eds.) , Elsevier Science Publisher B.V., 189-239 (1985).
[6] Betancourt, G. A. (2005). Las máquinas de soporte vectorial (SVMs). *Scientia et technica*, *1*(27).
[7] https://www.uv.mx/anmarin/slides/180205Gonzalez.pdf
[8] Jara Estupiñan, J., Giral, D., & Martínez Santa, F. (2016). Implementation of algorithms based on support vector machine (SVM) for electric systems: topic review. *Tecnura*, *20*(48), 149-170.
[9]http://powerhousedm.blogspot.com/2007/10/el-problema-xor.html#:~:text=El%20problema%20XOR,problemas%20de%20reconocimiento%20de%20patrones.&text=El%20resultado%20de%20XOR%20ser%C3%A1,sino%20ser%C3%A1%20Falso%20(0).
[10]https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron
[11] https://www.statisticssolutions.com/what-is-logistic-regression/
[12] [Moro et al., 2011] S. Moro, R. Laureano and P. Cortez
[13] https://archive.ics.uci.edu/ml/datasets/Bank+Marketing.
[14]https://eprints.ucm.es/48800/1/Memoria%20TFM%20Machine%20Learning_Juan_Zamorano_para_difundir%20%282%29.pdf