Tecnológico Nacional de México
Instituto Tecnológico de Tijuana

Subdirección Académica
Departamento de Sistemas y Computación

Semestre:
Agosto – Diciembre 2021

Carrera:
Ingeniería en Tecnologías de la Información y Comunicaciones

Materia y serie:
Datos Masivos BDD-1704TI9A

Unidad a evaluar: Unidad IV

Nombre de la Tarea:
Evaluación - Unidad 4

Nombre del Alumno:
Flores Gonzalez Luis Diego C16211486
Sifuentes Martinez Manuel Javier 17212934

Nombre del docente:
José Christian Romero Hernández

INDEX

INTRODUCTION

The final evaluation of the Big Data subject is presented below, where the knowledge acquired from different machine learning models and their application will be shown when making the prediction of the question "Has the client subscribed a term deposit?" where the possible answers are binary: 'yes' or 'no', this through the use of a dataset provided by the "UC Irvine Machine Learning Repository" that contains the data from "Bank Marketing" that shows possible patterns when identifying if a client can make a term deposit.

In order to have a prior knowledge of the models to be used, the explanation of each model will be shown in the theoretical framework and starting from these, make the programs in Scala with the use of apache spark that show the operation of each one.

This practice consists mainly of using the SVM, Decision Three, Logistic Regresion and Multilayer perceptron models, with the aim of comparing the performance of the models by generating a comparative table in which the prediction is shown using the aforementioned dataset, this table will contain 30 Executions of each model showing the results of each one as well as the average of each model with respect to its predictions and the total time of the 30 executions of each model.

THEORETICAL FRAMEWORK

1.1 Support Vector Machine (SVM)

It is a discriminative classifier formally defined by a separation hyperplane. In other words, given the tagged training data (supervised learning), the algorithm generates an optimal hyperplane that categorizes new examples. In two dimensional spaces, this hyperplane is a line that divides a plane into two parts where each class is on each side. It is used in many classification and regression problems, including medical applications of signal processing, natural language processing, and image and speech recognition. The idea of SVM is simple: the algorithm creates a line or a hyperplane that separates the data into classes.

What does SVM do?

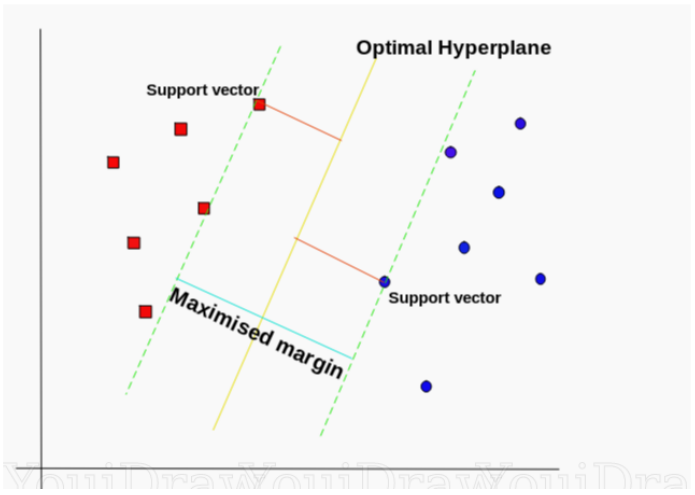
The goal of the SVM algorithm is to find a hyperplane that best separates two different classes of data points. "Best possible" implies the hyperplane with the widest margin between the two classes, this is generated by providing the training data set. The margin is defined as the maximum width of the region parallel to the hyperplane that has no interior data points.

Support Vector Machine (SVM) is primarily a more classical method that performs classification tasks by building hyperplanes in a multidimensional space that separates the cases of different class labels. SVM supports classification and regression tasks and can handle multiple continuous and categorical variables. For categorical variables, a dummy variable with case values such as 0 or 1 is created. Therefore, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three variables dummy: A: {1 0 0}, B: {0 1 0}, C: {0 0 1}

To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function. Based on the form of the error function, SVM models can be classified into four different groups:

- SVM Type 1 classification (also known as C-SVM)
- classificationSVM type 2 classification (also known as nu-SVM classification)
- Regression SVM type 1 (also known as epsilon-SVM)

- regressionSVM type 2 regression (also known as nu-SVM regression)



Fit parameters: Kernel, Regularization, Gamma andMargin

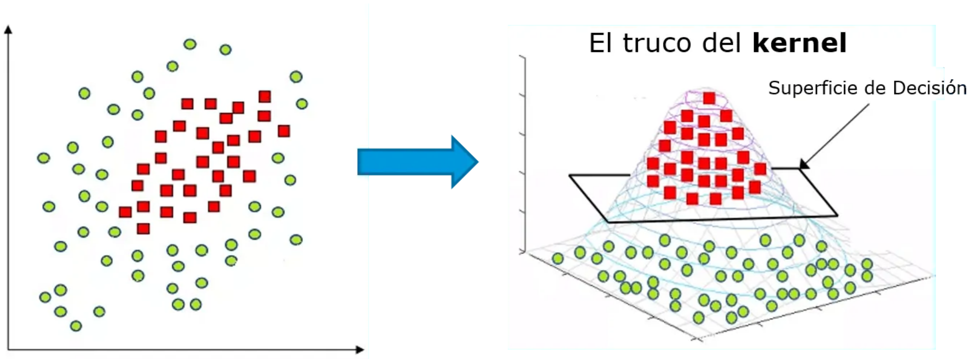
Kernel

The learning of the hyperplane in linear SVM is performed by transforming the problem using some linear algebra. This is where the kernel plays a role. For the linear kernel, the equation for the prediction of a new input using the point product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = B(0) + sum(ai * (x, xi))$$

This is an equation that involves calculating the internal products of a new input vector (x) with all support vectors in the training data. The learning coefficient must estimate the coefficients B0 and ai (for each input) from the training data.

The polynomial kernel can be written as K (x, xi) = 1 + sum (x * xi) ^ d and exponential as K (x, xi) = exp (-gamma * sum ((x - xi²)).

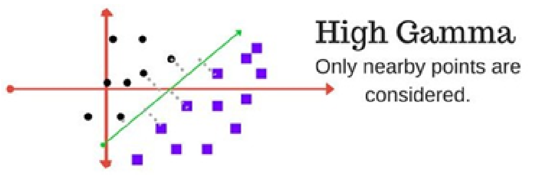


Regularization

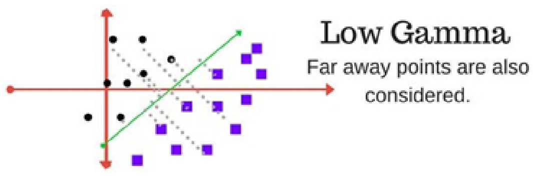
The parameter Regularization (often referred to as the C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a lower margin hyperplane if that hyperplane does a better work by correctly classifying all training points. Conversely, a very small value of C will cause the optimizer to look for a higher margin spacing hyperplane, even if that hyperplane misclassifies more points.

Range

The gamma parameter defines how far apart is influenced by a single training example, with low values signifying “far” and high values signifying “near.” In other words, with low range, points away from the plausible separation line are conserved. ideran in calculating the separation line. Where as high range means that points close to the plausible line are considered in the calculation.



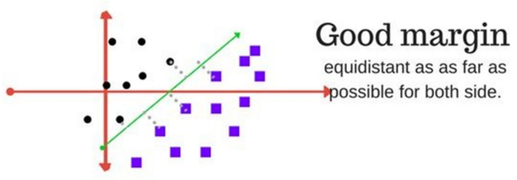
Alta gama

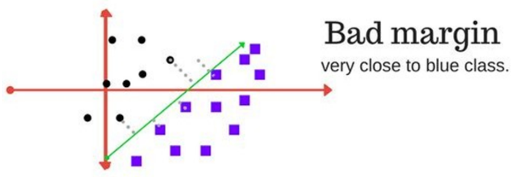


Margin

And finally the last but very important feature of the SVM classifier. SVM to core tries to achieve a good margin. A margin is a line separation to the closest class points.

A good margin is one in which this separation is greater for both classes. The images below give a visual example of good and bad margins. A good margin allows the points to be in their respective classes without crossing to another class.





Advantages and Disadvantages

Advantages:

- Classifier algorithm based on a solid theory. Risk minimization theorems are the state of the art in statistical learning.
- It can be applied to data represented in any Hilbert space (where you can define a measure of distance).
- Relatively few parameters to estimate.
- New extensions can be formulated (flexibility).
- Support vector machines are very popular and perform well in many classification and regression tasks.
- Although SVM algorithms are formulated for binary classification, multiclass SVM algorithms are built by combining several binary classifiers.
- Kernels make SVMs more flexible and capable of handling non-linear problems.
- To construct the decision surface, only the support vectors selected from the training data are required. After training is complete, the rest of the training data is irrelevant, producing a compact representation of the model that is suitable for generating code in an automated way.

Disadvantages:

- Determining the kernels to use is complex.
- It is just a binary classifier.

1.2 Decision Tree

A decision tree is a decision support tool that uses a graph or model of decisions in the form of a tree and its possible consequences, including the results of fortuitous events, resource costs, and utility. It is a way of displaying an algorithm that only contains conditional control statements.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (for example, whether a coin flips heads or tails), each branch represents the test result, and each leaf node represents a class label (decision made after calculating all attributes). Root-to-leaf paths represent classification rules.

How does the decision tree work?

The decision tree is a type of supervised learning algorithm (which has a predefined target variable) that is used primarily in classification problems. It works for categorical and continuous input and output variables. In this technique, we divide the population or sample into two or more homogeneous sets (or subpopulations) based on the most significant divisor / differentiator in the input variables.

DecisionDecision

tree typestree types are based on the type of target variable we have. It can be of two types::

1. **Categorical variable decision treeDecision**tree that has a categorical target variable and is then called a categorical variable decision tree.
2. **Continuous Variable Decision Tree**: The decision tree has a continuous target variable and is later called a continuous variable decision tree.

Decision tree construction

The way the tree can obtain information is by dividing the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive way called recursive partitioning. Recursion completes when the subset at a node has the same value as the target variable, or when the division no longer adds value to the predictions. Building the decision tree classifier does not require any domain knowledge or parameter setting and is therefore appropriate for exploratory knowledge discovery. Decision trees can handle high-dimensional data. In general, the decision tree classifier has good precision. Decision tree induction is a typical inductive approach to learning classification knowledge.

Advantages and disadvantages

Advantages:

- Decision trees can generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees can handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

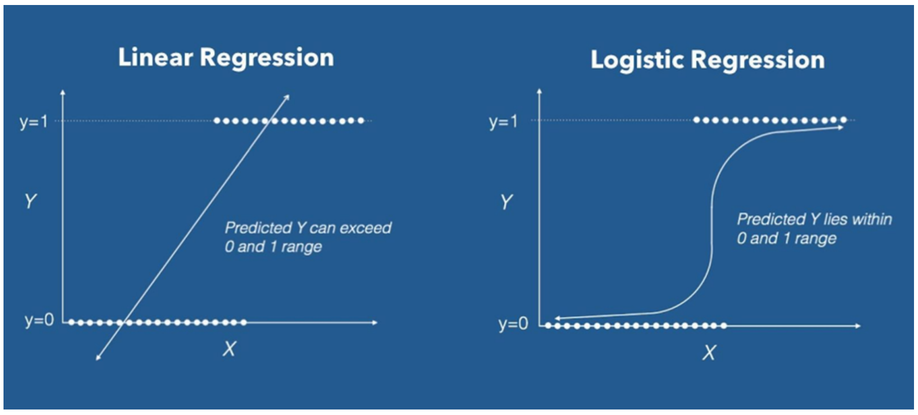
Disadvantages:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many classes and a relatively small number of training examples.
- The decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive.

1.3 Logistic Regression

It is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification issues are spam, fraud or non-fraud online transactions, benign or malignant tumor. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Logistic regression is a machine learning algorithm used for classification problems, it is a predictive analysis algorithm and it is based on the concept of probability.

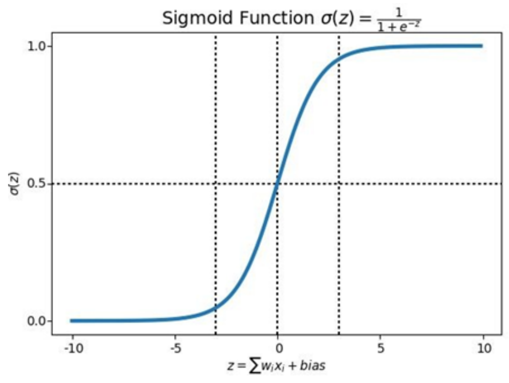

Linear Regression
y=1
Y
Predicted Y can exceed 0 and 1 range
y=0
X
Logistic Regression
y=1
Y
Predicted Y lies within 0 and 1 range
y=0
X

We can call a logistic regression a linear regression model, but logistic regression uses a more complex cost function, this cost function can be defined as the 'sigmoid function' or also known as the 'logistic function' instead of a linear function .

The hypothesis of logistic regression tends to limit the cost function between 0 and 1. Therefore, the linear functions do not represent it, since it can have a value greater than 1 or less than 0, which is not possible according to the hypothesis of logistic regression.

What is sigmoid function?

To map the predicted values to the probabilities, we use the Sigmoid function. The function maps any real value to another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.



How is logistic regression used?

Logistic regression should only be used when the target variables fall into discrete categories and if there is a continuous range of values that could be the target value, then logistic regression should not be used. Examples of situations in which you could use logistic regression include:

- Predicting whether an email is spam or not spam
- If a tumor is malignant or benign
- If a fungus is poisonous or edible.

When using logistic regression, you generally specify a threshold that indicates at what value the example will be placed in one class versus the other class. In the spam classification task, you can set a threshold of 0.5, which would cause an email with a 50% or greater probability of being spam to be classified as "spam" and any email with less than 50% probability will be classified as 'not spam'. "

Although logistic regression is more suitable for instances of binary classification, it can be applied to multiclass classification problems, classification tasks with three or more classes .

Advantages and Disadvantages

Advantages:

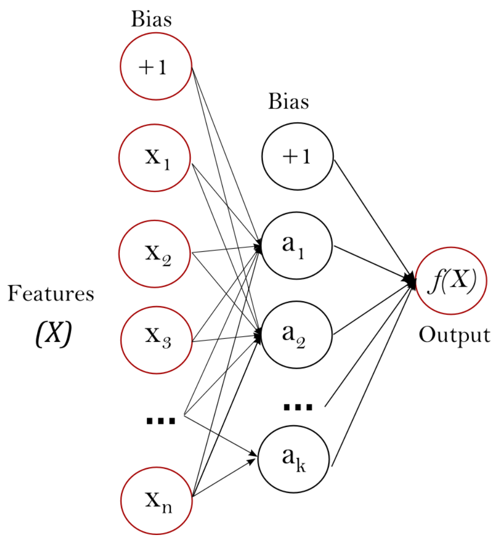
- logistic regression works well when the data set is linearly separableL2..
- logistic regression is less prone to overfitting, but can overfitting in datasets high dimension must consider regularization techniques (L1 and) to avoid overfitting in these scenarios.
- Logistic regression not only gives a measure of how relevant a predictor is (coefficient size), but t Also your association address (positive or negative).
- Logistic regression is easy to implement, interpret, and efficient to train.

Disadvantages:

- The main limitation of logistic regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, data is rarely linearly separable. Most of the time the data would be a messy mess.
- If the number of observations is less than the number of characteristics, then Logistic Regression should not be used, otherwise it may cause an overfitting.
- Logistic regression can only be used to predict discrete functions. Therefore, the dependent variable of Logistic Regression is restricted to the set of discrete numbers. This restriction in itself is problematic, as it is prohibitive for predicting continuous data.

1.4 Multilayer Perceptron Multilayer Perceptron

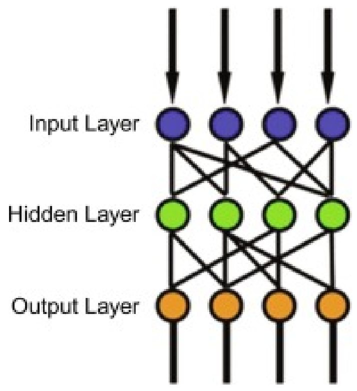
(MLP) is a supervised learning algorithm that learns a function f (.): $R^{m} \rightarrow R^{Or}$training on a data set, where is the number of dimensions for input and is the number of dimensions for output. Given a feature set X = x1, x2, ..., xm and a target “y”, you can learn a nonlinear function approximator for classification or regression. It is different from logistic regression, in that between the input layer and the output layer, there can be one or more non-linear layers, called hidden layers.



MLP layers

- Input layer: The input layer receives the input signal to be processed. consists of a set of neurons {xi | x1, x2, ..., xm} representing the input characteristics.
- Hidden layer: An arbitrary number of hidden layers that are placed between the input layer and the output layer are the true computational engine of the MLP. Similar to a feedforward network in an MLP, data flows in the forward direction from the input layer to the output layer.
- Output layer: The required task, such as prediction and classification, is performed by the output layer. receives the values from the last hidden layer and transforms them into output values.

Neurons in the MLP are trained with the backpropagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems that are not linearly separable. The main use cases for MLP are pattern classification, recognition, prediction, and approximation.



Advantages and Disadvantages

Advantages:

- Ability to learn non-linear models.
- Ability to learn models in real time (online learning) using partial_fit.

Disadvantages:

- MLPs with hidden layers have a non-convex loss function when there is more than one local minimum. Therefore, different random weight initializations can lead to different validation precision.
- MLP requires adjusting a series of hyperparameters.
- MLP is scale sensitive.

IMPLEMENTATION

Apache Spark

Apache Spark is a distributed processing engine responsible for orchestrating, distributing and monitoring applications that consist of multiple data processing tasks on several work machines, which form a cluster.

Regarding its general purpose, the virtue of Spark is that it is designed to cover a wide range of workloads that previously required different distributed systems. These systems include batch processing, iterative algorithms, interactive queries, streaming processing ... all of which are often used in a typical data analysis pipeline.

Spark is flexible in its use, and offers a series of APIs that allow users with different backgrounds to use it. It includes Python, Java, Scala, SQL and R APIs, with built-in functions and in general a reasonably good performance in all of them.
It allows working with more or less structured data (RDDs, dataframes, datasets) depending on the needs and preferences of the user.

Main characteristics It

- works in memory, which achieves much higher processing speed.
- It also allows working on disk. In this way, if, for example, we have a very large file or a quantity of information that does not fit in memory, the tool allows us to store part of it on disk, which makes it lose speed. This means that we have to try to find a balance between what is stored in memory and what is stored on disk, to have a good speed and so that the cost is not too high, since memory is always much more expensive than memory. disk.
- It provides APIs for Java, Scala, Python and R. It
- allows real-time processing, with a module called Spark Streaming, which combined with Spark SQL will allow us to process data in real time. As we inject the data, we can transform it and turn it into a final result.
- Resilient Distributed Dataset (RDD): It uses lazy evaluation, which means that all the transformations that we carry out on the RDDs are not resolved, but instead are stored in a directed acyclic graph (DAG), and when we execute a action, that is,

when the tool has no choice but to execute all the transformations, it will be when they are executed. This is a double-edged sword, as it has both an advantage and a disadvantage. The advantage is that speed is gained by not performing the transformations continuously, but only when necessary. The downside is that if any transformation raises some type of exception, it will not be caught until the action is executed, making it more difficult to debug or program.

SVM

```
// Import libraries
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.LinearSVC
import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.log4j._

//errors in the code run
LoggerHandle.getLogger ("org") .setLevel (Level.ERROR)

// Create a spark session and import dataset to use
val spark = SparkSession. builder (). getOrCreate ()
val df = spark.read.option ("header","true") .option
("inferSchema","true") .option ("delimiter",";") .format (" csv
") .load (" bank.csv ")

// Change the columns" and "to one with binary data.
val change1 = df.withColumn ("y", when (col ("y") .equalTo
("yes"),1) .otherwise (col ("y")))
val change2 = change1.withColumn ("y", when (col ("y") .equalTo
("no"),2) .otherwise (col ("y")))
val newcolumn = change2.withColumn ("y",'y.cast ("Int") )

// Create a new VectorAssembler object called assembler for the
features
val assembler = new VectorAssembler() .setInputCols
(Array("balance","day","duration","pdays","previous")).
SetOutputCol ("features")
val ugly = assembler.transform (newcolumn)

// Rename the column "y" a label
val change = ugly.withColumnRenamed ("y", "label")
val feat = change.select ("label","features")

// SVM: It is required to change the numerical categorical
values to 0 and 1 respectively
val c1 = feat.withColumn ("label", when (col ("label") .equalTo
("1"),0) .otherwise (col ("label")))
val c2 = c1.withColumn ("label ", when (col (" label ") .equalTo
(" 2 "),1) .otherwise (col (" label ")))
val c3 = c2.withColumn (" label ",'label.cast (" Int "))

// Divide the data set into training and test array
val Array(trainingData, testData) = c3.randomSplit (Array(0.7,
0.3))

// Model instance
val linsvc = new LinearSVC() .setLabelCol ("label").
SetFeaturesCol ("features")

// Fit model to training data
val linsvcModel = linsvc.fit (trainingData)

// Perform classification prediction with test data
val lnsvc_prediction = linsvcModel.transform (testData)
lnsvc_prediction.select ("prediction", "label", "features")
.show (10)

// Print the intercept line
println (s "Coefficients: $ {linsvcModel.coefficients}
Intercept: $ {linsvcModel.intercept} ")

// Show the precision of the model
val evaluator = new MulticlassClassificationEvaluator()
.setLabelCol ("label") .setPredictionCol ("prediction")
.setMetricName ("accuracy")
val lnsvc_accuracy = evaluator.evaluate (lnsvc_prediction)
print ("Accuracy of Support Vector Machine is =" +
(lnsvc_accuracy))
```

Result :

```
lnsvc_prediction: org.apache.spark.sql.DataFrame = [label: int,
features: vector ... 2 more fields]
+ ---------- + ----- + --------------- ----- +
| prediction | label | features |
+ ---------- + ----- + -------------------- +
|       1.0|     0| [-3058.0,17.0,882... |
|       1.0|    0| [-1944.0,7.0,623.... |
|       1.0|    0| [-970.0,4.0,489.0... |
|       1.0|    0| [-839.0,16.0,1018... |
|       1.0|    0| [-754.0,9.0,727.0... |
|       1.0|    0| [-639.0,15.0,585.... |
|       1.0|    0| [-537.0,21.0,1039... |
|       1.0|    0| [-477.0,21.0,1532... |
|       1.0|    0| [-468.0,8.0,534.0... |
|       1.0|    0| [-466.0,15.0,901.... |
+ ---------- + ----- + -------------------- +
only showing top 10 rows

Coefficients: [-8.232609095789674E
-10,5.545431393089826E-7,-1.2930229548726792E-7,-5.7752129187854
436E-8,-1.5948962471281635E-6] Intercept: 1.000044130947889
evaluator:
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
= mcEval_051e12751017
lnsvc_accuracy: Double = 0.8830457003184949
Accuracy of Support Vector Machine is = 0.8830457003184949
```

Decision Tree

```
/ * Import libraries * /
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.DateType
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.feature.IndexToString
import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import
org.apache.spark.ml.classification.DecisionTreeClassificationMod
el
import org.apache.log4j._

/ * Handle errors in code run * /
Logger.getLogger ("org") .setLevel (Level.ERROR)
```

```
/ * We create a spark session and load the CSV data into a
datraframe * /
val spark = SparkSession.builder (). getOrCreate ()
val df = spark.read.option ("header","true") .option
("inferSchema","true") .option ("delimiter",";") .format ("csv")
.load ("bank.csv")

// Display the schema and the first 5 lines
df.printSchema ()
df.show (5)

// Change the columns "and" by one with binary data.
val change1 = df.withColumn ("y", when (col ("y") .equalTo
("yes"),1) .otherwise (col ("y")))
val change2 = change1.withColumn ("y", when (col ("y") .equalTo
("no"),2) .otherwise (col ("y")))
val newcolumn = change2.withColumn ("y",'y.cast ("Int") )

// Create a new VectorAssembler object called assembler for the
features
val assembler = new VectorAssembler() .setInputCols
(Array("balance","day","duration","pdays","previous")).
SetOutputCol ("features")
val ugly = assembler.transform (newcolumn)

/ * Rename column "y" a label * /
val change = ugly.withColumnRenamed ("y", "label")
val feat = change.select ("label","features")

// Declare a new StringIndexer object to rename and index the
column label a indexedLabel
val labelIndexer = new StringIndexer() .setInputCol ("label")
.setOutputCol ("indexedLabel") .fit (feat)

// Create a new VectorIndexer object called featureIndexer to
index all the data contained in features, creating a new column
called indexedFeatures
val featureIndexer = new VectorIndexer() .setInputCol
("features") .setOutputCol ("indexedFeatures") .setMaxCategories
(4)
```

```
// Divide the data set into training and test array
val Array(trainingData, testData) = feat.randomSplit (Array(0.7,
0.3))

// Model instance
val dt = new DecisionTreeClassifier() .setLabelCol
("indexedLabel") .setFeaturesCol ("indexedFeatures")

// Convert the indexed data to its original type
val labelConverter = new IndexToString() .setInputCol
("prediction") .setOutputCol ("predictedLabel") .setLabels
(labelIndexer.labels)

// Create a PIpeline object to unite the classification process
into a single object
val pipeline = new Pipeline() .setStages (Array(labelIndexer,
featureIndexer, dt, labelConverter))
// Fit model to training data
val model = pipeline.fit (trainingData)

// Perform classification prediction with test data
val predictions = model.transform (testData)
predictions.select ("predictedLabel", "label", "features") .show
(5)

// Show the precision of the model
val evaluator = new MulticlassClassificationEvaluator()
.setLabelCol ("indexedLabel") .setPredictionCol ("prediction")
.setMetricName ("accuracy")
val accuracy = evaluator.evaluate (predictions)
println (s "Accuracy = $ {accuracy}")
}
```

Result:

```
result:
Accuracy= 0.8925357722377932
```

Logistic Regression

```
/ * Import libraries * /
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.ml.feature. {VectorAssembler,
StringIndexer, VectorIndexer}
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.sql.SparkSessionHandle
import org.apache.log4j._

/ *errors in the code run * /
Logger.getLogger ("org") .setLevel (Level.ERROR)

/ * Create a spark session and import dataset to use * /
val spark = SparkSession.builder (). getOrCreate ()
val data = spark.read.option ("header","true") .option
("inferSchema", "true") .option ("delimiter",";") .format
("csv") .load ("bank.csv")

/ * Change the columns "y" to one with binary data. * /
val yes = data.withColumn ("y", when (col ("y") .equalTo
("yes"),1) .otherwise (col ("y")))
val clean = yes.withColumn ("y", when (col ("y") .equalTo
("no"),2) .otherwise (col ("y")))
val cleanData = clean.withColumn ("y",'y.cast ("Int"))

// Create a new Array with the selected data
val featureCols = Array("age","previous","balance","duration")

// Create a new VectorAssembler object called assembler for the
features
val assembler = new VectorAssembler() .setInputCols
(featureCols) .setOutputCol ("features")
val df2 = assembler.transform (cleanData)

// Rename column "y" to label
val featuresLabel = df2.withColumnRenamed ("y", "label")
val dataI = featuresLabel.select ("label","features")

// Divide the data set into training and test array
val Array(training, test) = dataI.randomSplit (Array(0.7, 0.3),
seed = 12345)

// Model instance
```

```
val lr = new LogisticRegression() .setMaxIter (10) .setRegParam
(0.3) .setElasticNetParam (0.8)

// Fit the model to the training data
val lrModel = lr.fit (training)

// Perform the prediction with the test data
var results = lrModel.transform (test)

// Print the coefficients and intercepts
println (s "Coefficients: \ n $ {lrModel.coefficientMatrix}")
println (s "Intercepts: \ n $ {lrModel.interceptVector}")

// Convert the results to RDD using .as and .rdd
val predictionAndLabels = results.select ($"prediction",
$"label") .as [(Double, Double)]. rdd

// Show model precision
val metrics = new MulticlassMetrics(predictionAndLabels)
println (s "Accuracy = $ {metrics.accuracy}")
```

Result:

```
result:
Coefficients:
3 x 4 CSCMatrix
Intercepts:
[-7.668355241088849,2.8268160217768314,4.841539219312018]

Accuracy= 0.8839272727272727
```

Multilayer Perceptron

```
/*Importar libreras*/
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.SparkSession
import org.apache.spark.ml.feature.VectorAssembler
import
org.apache.spark.ml.classification.MultilayerPerceptronClassifie
r
import
org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.IndexToString
import org.apache.spark.ml.Pipeline
import org.apache.log4j._

/*Controlar errores en la corrida del cdigo*/
Logger.getLogger("org").setLevel(Level.ERROR)

/*Crear un sesión de spark e importar dataset a utilizar*/
val spark = SparkSession.builder().getOrCreate()
val df =
spark.read.option("header","true").option("inferSchema","true").
option("delimiter",";").format("csv").load("bank.csv")

/*Cambiar las columnas "y" por una con datos binarios.*/
val df1 =
df.withColumn("y",when(col("y").equalTo("yes"),1).otherwise(col(
"y")))
val df2 =
df1.withColumn("y",when(col("y").equalTo("no"),0).otherwise(col(
"y")))
val newcolumn = df2.withColumn("y",'y.cast("Int"))

//Crear un nuevo objeto VectorAssembler llamado assembler para
los features
val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration",
"pdays","previous")).setOutputCol("features")
val newDF = assembler.transform(newcolumn)

//Renombrar la columna "y" a label
val cambio = newDF.withColumnRenamed("y", "label")
val finalDF = cambio.select("label","features")


//Declarar un nuevo objeto StringIndexer para renombrar e
indexar la columna label a indexedLabel
val labelIndexer = new
StringIndexer().setInputCol("label").setOutputCol("indexedLabel"
).fit(finalDF)
```

```
//Mostrar la categoría de los datos
println(s"Found labels: ${labelIndexer.labels.mkString("[", ",
", "]")}")

//Crear un nuevo objeto VectorIndexer llamado featureIndexer
para indexar todos los datos contenidos en features, creando una
nueva columna llamada indexedFeatures
val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFea
tures").setMaxCategories(2).fit(finalDF)

//Dividir el conjunto de datos en array de entrenamiento y de
prueba
val splits = c3.randomSplit(Array(0.7, 0.3), seed = 1234L)
val trainingData = splits(0)
val testData = splits(1)

//Especificar las capas de la red neuronal
val layers = Array[Int](5, 6, 5, 2)

//Crear la instancia del método de la librería de clasificación
con el campo de entrada "indexedLabel" y los características del
campo "indexedFeatures"
val trainer = new
MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("
indexedLabel").setFeaturesCol("indexedFeatures").setBlockSize(12
8).setSeed(1234L).setMaxIter(100)

//Convertir los datos indexados a su tipo original
val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predicte
dLabel").setLabels(labelIndexer.labels)



//Crear un objeto PIpeline para unir en un solo objeto el
proceso de clasificación
val pipeline = new Pipeline().setStages(Array(labelIndexer,
featureIndexer, trainer, labelConverter))

// AJustar el modelo a los datos de entrenamiento
val model = pipeline.fit(trainingData)
```

```
//Realizar la predicción con los datos de prueba
val prediction = model.transform(testData)
prediction.select("prediction", "label", "features").show(5)

//Mostrar la precisión del modelo
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLabel").
setPredictionCol("prediction").setMetricName("accuracy")
val accuracy = evaluator.evaluate(prediction)
print("Accuracy of Multilayer Perceptron Classifier is = " +
(accuracy))
```

Resultado:

```
Found labels: [0, 1]
+----------+-----+--------------------+
|prediction|label| features|
+----------+-----+--------------------+
|       1.0|    0|[-3058.0,17.0,882...|
|       1.0|    0|[-1944.0,7.0,623....|
|       1.0|    0|[-1206.0,15.0,382...|
|       1.0|    0|[-770.0,18.0,618....|
|       1.0|    0|[-556.0,8.0,682.0...|
+----------+-----+--------------------+
only showing top 5 rows

Accuracy of Multilayer Perceptron Classifier is =
0.886684037558685
```

RESULTADOS

El porcentaje mostrado a continuación pertenece a la precisión obtenida por cada uno de los métodos al finalizar el proceso de predicción de un conjunto de datos.

Iteración	Decision Tree	Logistic Regression	SVM	Multilayer Perceptron
1	89.09%	88.21%	88.41%	88.66%
2	89.57%	88.21%	88.09%	88.66%
3	89.19%	88.21%	88.77%	88.66%

4	89.47%	88.21%	88.63%	88.66%
5	88.95%	88.21%	88.46%	88.66%
6	89.45%	88.21%	87.97%	88.66%
7	89.36%	88.21%	88.35%	88.66%
8	88.72%	88.21%	88.37%	88.66%
9	88.87%	88.21%	88.36%	88.66%
10	89.15%	88.21%	87.90%	88.66%
11	88.95%	88.21%	88.40%	88.66%
12	89.27%	88.21%	88.05%	88.66%
13	89.09%	88.21%	88.29%	88.66%
14	89.65%	88.21%	88.51%	88.66%
15	89.30%	88.21%	88.20%	88.66%
16	88.73%	88.21%	88.53%	88.66%
17	89.42%	88.21%	88.29%	88.66%
18	89.18%	88.21%	88.77%	88.66%
19	88.89%	88.21%	88.11%	88.66%
20	89.02%	88.21%	87.97%	88.66%
21	89.33%	88.21%	88.36%	88.66%
22	89.50%	88.21%	87.96%	88.66%
23	88.45%	88.21%	88.41%	88.66%
24	89.13%	88.21%	88.65%	88.66%

25	89.00%	88.21%	88.21%	88.66%
26	89.12%	88.21%	88.63%	88.66%
27	89.00%	88.21%	88.74%	88.66%
28	89.20%	88.21%	88.72%	88.66%
29	89.10%	88.21%	88.38%	88.66%
30	89.19%	88.21%	88.41%	88.66%
Tiempo (m in):	3:06	1:42	6 00	6: 05
Promedio (Precisión):	89.14%	88.21%	88.36%	88.66%

Explicación

Se puede observar que existe una clara diferencia de tiempo entre los métodos utilizados en la práctica, logrando Logistic Regression una diferencia de aproximadamente un minuto y medio en comparación con Decision Tree, que es el método más cercano, no siendo igual de eficiente en la precisión alcanzada como lo obteniendo en el tiempo de finalización. Cabe resaltar que las pruebas se realizaron sobre un almacenamiento de disco sólido, lo que ayuda a que la velocidad de lectura sea más rápida que en un disco mecánico, afectando también que el hardware a utilizar dependió de las especificaciones dadas en una máquina virtual, la cual corría el sistema operativo Linux. Habiendo dado contexto del ambiente sobre el cual se realizaron las pruebas, y habiendo sido probados dichos algoritmos de clasificación con un dataset que rebasaba los 40,000 datos, los cuales son más cercanos a un simulado escenario real, se podría decir que Logistic Regression sería un candidato sólido a elegir como "ganador", esto si la prioridad es la rapidez de procesamiento, sacrificando un poco la precisión final de las predicciones, de lo contrario, y dependiendo las circunstancias de trabajo y el origen de los datos, Decission Tree no se queda atrás, logrando un mayor puntaje en cuanto a precisión. SVM ni Multilayer Perceptron se deben de dejar de lado solo tener un mayor tiempo de finalización en los procesos de predicción, o en cuanto a su precisión, pueden llegar a arrojar mejores resultados dependiendo la situación en la que se vayan a utilizar, la decisión final está en el análisis de los datos y el objeto por el cual se quiera utilizar cierto método, ya que todos tienen características, ventajas y desventajas que pueden variar dependiendo la finalidad que se quiera alcanzar con dichas predicciones de datos.

CONCLUSIONES

Este proyecto representa la culminación del curso de Datos masivos donde aplicamos el conocimiento adquirido para realizar múltiples modelos capaces de realizar predicciones acertadas para generar posibles soluciones, para está evaluación se desarrolló una comparación de los modelos más utilizados y eficiente usados en el mundo del aprendizaje automático y principalmente del Big Data, para mostrar el correcto funcionamiento de los mismos y ver sus resultados comparados con los demás al haber construido una tabla que muestra claramente que todas en general generan un rendimiento aceptable ya que cada una varía con respecto la otra pero al final todas con resultados superiores al 80%.

Podemos concluir que dependiendo el problema o petición de un cliente la mejor manera de desarrollar una posible solución es considerar las variables que se tienen y ya teniendo un entendimiento mejor de los modelos hacer una elección concurrente del modelo o si la situación lo requiere es muy recomendable hacer uso de diferentes del mismo tipo como en este caso de este proyecto donde los resultados son categóricos por lo que los modelos de clasificación generan una mejor predicción.

REFERENCES

(Nd). Iartificial.Net. Consultado: Diciembre 5, 2021, de https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

(Nd). Merkleinc.Com. Consultado: Diciembre 9, 2021, de https://www.merkleinc.com/es/es/blog/machine-learning-support-vector-machines

(Nd). Medium.Com. Consultado: Diciembre 9, 2021, de https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

MS (. (2017, Octubre 6). Chapter 4: Decision trees algorithms. Deep Math Machine Learning.Ai. https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1

Decision Tree. (2017, Octubre 16). Geeksforgeeks.Org. https://www.geeksforgeeks.org/decision-tree/

How Decision Tree Algorithm works. (2017, Enero 30). Dataaspirant.Com. https://dataaspirant.com/how-decision-tree-algorithm-works/

Kambria. (2019, Julio 9). Logistic regression for machine learning and classification. Kambria.Io. https://kambria.io/blog/logistic-regression-for-machine-learning/

Pant, A. (2019, Enero 22). Introduction to Logistic Regression - towards data science. Towards Data Science. https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

1.17. Neural network models (supervised). (nd). Scikit-Learn.Org. Consultado: Diciembre 5, 202 1, de https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Abirami, S., & Chitra, P. (2020). Energy-efficient edge based real-time healthcare support system. In Advances in Computers (pp. 339–368). Elsevier.

Qué es Apache Spark. (2018, Julio 2). Openwebinars.Net. https://openwebinars.net/blog/que-es-apache-spark/

Unknown. (2021). Multilayer Perceptron Classifier (Scala). 12/05/2021, de Databricks Sitio web: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3741049972324885/1019862370390522/4413065072037724/latest.html

[Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014