TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE SEPTIEMBRE 2020 - ENERO 2021

INGENIERÍA EN SISTEMAS COMPUTACIONALES

DOCENTE
JOSÉ CHRISTIAN ROMERO HERNÁNDEZ

CLASE
BIG DATA BDD-1704 SC9A, L - V 18:00 - 19:00 (91L4/0308)

BIG DATA - FINAL PROJECT

INTEGRANTES

HUGO ANDRÉS PACHECO RAMÍREZ	16210790
SANDOVAL CASTRO SEBASTIÁN	16212076

Tijuana, Baja California, enero 2021

BIG DATA - FINAL PROJECT

Pacheco Ramírez Hugo Andrés, Sandoval Castro Sebastián
Departamento de Ciencias y Computación. Instituto Tecnológico de Tijuana
{hugo.pacheco, sebastian.sandoval16}@tectijuana.edu.mx

Abstract: There are a lot of algorithms implemented to analyze data, but, ¿what is the best of them? This research implements Decision Tree, Logistic Regression, Multilayer Perceptron and Support Vector Machine. It makes a comparison between the results looking for the most efficient algorithm.

Key Words: Big Data, Algorithm, Classification, Model, AI, Machine Learning, Decision Tree, Logistic Regression, Multilayer Perceptron, SVM.

1. Introduction

In this project, we implement four classification algorithms: Decision Tree, Logistic Regression, Multilayer Perceptron and Support Vector Machine, using the dataset bank.csv.

We use scala programming language to structure the model, and apache spark, explaining why we use it and showing the code of each algorithm.

We interpret the final results, using the averages of ten different performances, and we proceed to conclude the results based on the time of performance.

2. Theoretical Framework

Introduction to the theoretical framework.

Big Data

Big Data is about the growing challenge that organizations face as they deal with large and fast-growing sources of data or information that also present a complex range of analysis and use problems. These can include:

- Having a computing infrastructure that can ingest, validate, and analyze high volumes (size and/or rate) of data
- Assessing mixed data (structured and unstructured) from multiple sources
- Dealing with unpredictable content with no apparent schema or structure
- Enabling real-time or near-real-time collection, analysis, and answers

Big Data technologies describe a new generation of technologies and architectures,designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis.[2]

Artificial Intelligence

It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. [3]

Machine Learning

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment. They are considered the working horse in the new era of the so-called big data. Techniques based on machine learning have been applied successfully in diverse fields ranging from pattern recognition, computer vision, spacecraft engineering, finance, entertainment, and computational biology to biomedical and medical applications. [4]

Decision Tree

Decision Tree is widely applied in many areas, such as classification and recognition. Traditional information entropy and Pearson's correlation coefficient are often applied as measures of splitting rules to find the best splitting attribute. However, these methods can not handle uncertainty, since the relation between attributes and the degree of disorder of attributes can not be measured by them. [5]

Logistic Regression

Logistic regression is a useful statistical technique to understand complex phenomena. The use of this statistic is discussed, along with how to understand research when it is used.

An extension of this procedure is called multiple regression, in which multiple independent (also called predictor) variables in the analysis are used to explain one dependent variable. The dependent variable is continuous (interval or ratio level data) or dichotomous (yes/no). This analysis is complex because the variables are tested simultaneously. For example, this kind of analysis would be appropriate to predict nurse satisfaction based on age, clinical specialty, and level of education. Results can indicate how much of the

variability of the dependent variable is explained by each of the independent variables. [6]

Multilayer Perceptron

A multilayer perceptron is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard linear perceptron in that it uses three or more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than the perceptron in that it can distinguish data that is not linearly separable, or separable by a hyperplane. MLP networks are general-purpose, flexible, nonlinear models consisting of a number of units organised into multiple layers. The complexity of the MLP network can be changed by varying the number of layers and the number of units in each layer. Given enough hidden units and enough data, it has been shown that MLPs can approximate virtually any function to any desired accuracy. This paper presents the performance comparison between Multi-layer Perceptron (back propagation, delta rule and perceptron). [7]

Support Vector Machine

The Support Vector Machine was developed by Vapnik and others (Cortes & Vapnik. 1995,Vapnik et al. 1997) as an alternative to machine learning for neural networks and perceptrons (Rummelhart et al. 1986). The model is based on incorporating the vector of inputs in a multidimensional feature space Z by means of a mapping not linear.

Therefore, it is a question of separating the problem into different spaces dimensional. The support vector machine defines the margin of the maximum distance between the different classes of the hyperplane. [8]

3. Methodology

Scala programming language and Spark v.2.4.7 was used to perform the code run and obtain the analyzed results.

We decided to use Spark because this project requires parallelization and run code in fast mode, and Apache Spark is an industry standard for doing that [9].

And we decided to use Scala because Scala fuses object-oriented and functional programming in a statically typed programming language [10], and this helps us a lot to work with massive amounts of data meanwhile we have a better run time.

The code structure was as it shows next:

Decision Tree Code

```
1.  //Necessary libraries.
2.  import org.apache.spark.sql.SparkSession
3.  import org.apache.spark.ml.Pipeline
4.  import org.apache.spark.ml.linalg.Vectors
5.  import
    org.apache.spark.ml.evaluation.MulticlassCl
    assificationEvaluator
6.  import
    org.apache.spark.ml.classification.Decision
    TreeClassifier
7.  import
    org.apache.spark.ml.classification.Decision
    TreeClassificationModel
8.  import
    org.apache.spark.ml.feature.IndexToString
9.  import
    org.apache.spark.ml.feature.StringIndexer
10. import
    org.apache.spark.ml.feature.VectorIndexer
11. import
    org.apache.spark.ml.feature.VectorAssembler
12.
13. //Error level code.
14. import org.apache.log4j._
15. Logger.getLogger("org").setLevel(Level.ERRO
    R)
16.
17. //Spark session.
18. val spark =
    SparkSession.builder.appName("DecisionTree"
    ).getOrCreate()
19.
20. //Reading the csv file.
21. val df  =
    spark.read.option("header","true").option("
    inferSchema",
    "true").option("delimiter",";").format("csv
    ").load("C:/Users/Sebas/Desktop/unit4/bank.
    csv")
22.
23. //Indexing.
24. val labelIndexer = new
    StringIndexer().setInputCol("y").setOutputC
    ol("indexedLabel").fit(df)
25. val indexed =
    labelIndexer.transform(df).drop("y").withCo
    lumnRenamed("indexedLabel", "label")
26.
```

```
//Vector of the numeric category columns.
val vectorFeatures = (new VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")).setOutputCol("features"))

//Transforming the indexed value.
val features = vectorFeatures.transform(indexed)

//Renaming the column y as label.
val featuresLabel = features.withColumnRenamed("y", "label")

//Union of label and features as dataIndexed.
val dataIndexed = featuresLabel.select("label","features")

//Creation of labelIndexer and featureIndexer for the pipeline, Where features with distinct values > 4, are treated as continuous.
val labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(dataIndexed)
val featureIndexer = new VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(4).fit(dataIndexed)

//Training data as 70% and test data as 30%.
val Array(trainingData, testData) = dataIndexed.randomSplit(Array(0.7, 0.3))

//Creating the Decision Tree object.
val decisionTree = new DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures")

//Creating the Index to String object.
val labelConverter = new IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(labelIndexer.labels)

//Creating the pipeline with the objects created before.
val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, decisionTree, labelConverter))

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

//Making the predictions transforming the testData.
val predictions = model.transform(testData)

//Showing the predictions.
predictions.select("predictedLabel", "label", "features").show(5)

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

//Accuracy.
val accuracy = evaluator.evaluate(predictions)

//Accuracy and Test Error.
println(s"Accuracy: ${(accuracy)}")
println(s"Test Error: ${(1.0 - accuracy)}")
```

Logistic Regression Code

```
//Necessary libraries.
import org.apache.spark.sql.SparkSession
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorIndexer
```

```
//Error level code.
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERRO
R)

//Spark session.
val spark =
SparkSession.builder.appName("LogisticRegre
ssion").getOrCreate()

//Reading the csv file.
val df  =
spark.read.option("header","true").option("
inferSchema",
"true").option("delimiter",";").format("csv
").load("C:/Users/Sebas/Desktop/unit4/bank.
csv")

//Indexing.
val labelIndexer = new
StringIndexer().setInputCol("y").setOutputC
ol("indexedLabel").fit(df)
val indexed =
labelIndexer.transform(df).drop("y").withCo
lumnRenamed("indexedLabel", "label")

//Vector of the numeric category columns.
val vectorFeatures = (new
VectorAssembler().setInputCols(Array("balan
ce","day","duration","pdays","previous")).s
etOutputCol("features"))

//Transforming the indexed value.
val features =
vectorFeatures.transform(indexed)

//Renaming the column y as label.
val featuresLabel =
features.withColumnRenamed("y", "label")

//Union of label and features as
dataIndexed.
val dataIndexed =
featuresLabel.select("label","features")

//Training data as 70% and test data as
30%.
val Array(trainingData, testData) =
dataIndexed.randomSplit(Array(0.7, 0.3))

//Logistic regression object.
val logisticReg = new
LogisticRegression().setMaxIter(10).setRegP
aram(0.3).setElasticNetParam(0.8).setFamily
("multinomial")

//Fitting the model with the training data.
val model = logisticReg.fit(trainingData)

//Making the predictions transforming the
testData.
val predictions = model.transform(testData)

//Obtaining the metrics.
val predictionAndLabels =
predictions.select($"prediction",$"label").
as[(Double, Double)].rdd
val metrics = new
MulticlassMetrics(predictionAndLabels)

//Confusion matrix.
println("Confusion matrix:")
println(metrics.confusionMatrix)

//Accuracy and Test Error.
println("Accuracy: " + metrics.accuracy)
println(s"Test Error: ${(1.0 -
metrics.accuracy)}")
```

Multilayer Perceptron Code

```
//Necessary libraries.
import org.apache.spark.sql.SparkSession
import
org.apache.spark.sql.types.IntegerType
import
org.apache.spark.ml.evaluation.MulticlassCl
assificationEvaluator
import
org.apache.spark.ml.classification.Multilay
erPerceptronClassifier
import
org.apache.spark.ml.feature.StringIndexer
import
org.apache.spark.ml.feature.VectorAssembler
```

```
//Error level code.
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERRO
R)

//Spark session.
val spark =
SparkSession.builder.appName("MultilayerPer
ceptron").getOrCreate()

//Reading the csv file.
val df  =
spark.read.option("header","true").option("
inferSchema",
"true").option("delimiter",";").format("csv
").load("C:/Users/Sebas/Desktop/unit4/bank.
csv")

//Indexing.
val labelIndexer = new
StringIndexer().setInputCol("y").setOutputC
ol("indexedLabel").fit(df)
val indexed =
labelIndexer.transform(df).drop("y").withCo
lumnRenamed("indexedLabel", "label")

//Vector of the numeric category columns.
val vectorFeatures = (new
VectorAssembler().setInputCols(Array("balan
ce","day","duration","pdays","previous")).s
etOutputCol("features"))

//Transforming the indexed value.
val features =
vectorFeatures.transform(indexed)

//Fitting indexed and finding labels 0 and
1.
val labelIndexer = new
StringIndexer().setInputCol("label").setOut
putCol("indexedLabel").fit(indexed)

//Splitting the data in 70% and 30%.
val splits =
features.randomSplit(Array(0.7, 0.3))
val trainingData = splits(0)
val testData = splits(1)

//Creating the layers array.
val layers = Array[Int](5, 4, 1, 2)

//Creating the Multilayer Perceptron object
of the Multilayer Perceptron Classifier.
val multilayerP = new
MultilayerPerceptronClassifier().setLayers(
layers).setBlockSize(128).setSeed(1234L).se
tMaxIter(100)

//Fitting trainingData into the model.
val model = multilayerP.fit(trainingData)

//Transforming the testData for the
predictions.
val prediction = model.transform(testData)

//Selecting the prediction and label
columns.
val predictionAndLabels =
prediction.select("prediction", "label")

//Creating a Multiclass Classification
Evaluator object.
val evaluator = new
MulticlassClassificationEvaluator().setMetr
icName("accuracy")

//Accuracy and Test Error.
println(s"Accuracy:
${evaluator.evaluate(predictionAndLabels)}"
)
println(s"Test Error: ${1.0 -
evaluator.evaluate(predictionAndLabels)}")
```

Support Vector Machine Code

```
//Necessary libraries.
import org.apache.spark.sql.SparkSession
import
org.apache.spark.mllib.evaluation.Multiclas
sMetrics
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.linalg.Vectors
import
org.apache.spark.ml.classification.LinearSV
C
import
org.apache.spark.ml.classification.Logistic
Regression
```

```
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.VectorAssembler

//Error level code.
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERROR)

//Spark session.
val spark = SparkSession.builder.appName("SVM").getOrCreate()

//Reading the csv file.
val df  = spark.read.option("header","true").option("inferSchema", "true").option("delimiter",";").format("csv").load("C:/Users/Sebas/Desktop/unit4/bank.csv")

//Indexing.
val labelIndexer = new StringIndexer().setInputCol("y").setOutputCol("indexedLabel").fit(df)
val indexed = labelIndexer.transform(df).drop("y").withColumnRenamed("indexedLabel", "label")

//Vector of the numeric category columns.
val vectorFeatures = (new VectorAssembler().setInputCols(Array("balance","day","duration","pdays","previous")).setOutputCol("features"))

//Transforming the indexed value.
val features = vectorFeatures.transform(indexed)

//Renaming the column y as label.
val featuresLabel = features.withColumnRenamed("y", "label")

//Union of label and features as dataIndexed.
val dataIndexed = featuresLabel.select("label","features")

//Creation of labelIndexer and featureIndexer for the pipeline, Where features with distinct values > 4, are treated as continuous.
val labelIndexer = new StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(dataIndexed)
val featureIndexer = new VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(4).fit(dataIndexed)

//Training data as 70% and test data as 30%.
val Array(training, test) = dataIndexed.randomSplit(Array(0.7, 0.3))

//Linear Support Vector Machine object.
val supportVM = new LinearSVC().setMaxIter(10).setRegParam(0.1)

//Fitting the trainingData into the model.
val model = supportVM.fit(trainingData)

//Transforming testData for the predictions.
val predictions = model.transform(testData)

//Obtaining the metrics.
val predictionAndLabels = predictions.select($"prediction",$"label").as[(Double, Double)].rdd
val metrics = new MulticlassMetrics(predictionAndLabels)

//Confusion matrix.
println("Confusion matrix:")
println(metrics.confusionMatrix)

//Accuracy and Test Error.
println("Accuracy: " + metrics.accuracy)
println(s"Test Error = ${(1.0 - metrics.accuracy)}")
```

4. Results

As a result, we obtained the next table (**Fig 1.** *Accuracy and Test Error Results*) based on the classification algorithms previously implemented: Decision Tree, Logistic Regression, Multilayer Perceptron and Support Vector Machine. Using the dataset "bank.csv".

Fig 1. *Accuracy and Test Error Results.*

Name	Accuracy	Error	Seconds
Decision Tree	0.8814	0.1185	11.40
Logistic Regression	0.8862	0.1137	12.50
Multilayer Perceptron	0.8852	0.1147	22.80
Support Vector Machine	0.8883	0.1116	31.10

As we can see, the fastest algorithm to produce a result is the Decision Tree algorithm, with an accuracy of 0.8814, an error of 0.1185 and 11.40 seconds of performance.

The best accuracy is obtained with the performance of the Support Vector Machine algorithm, with an accuracy of 0.8883, an error of 0.1116 and 31.10 seconds of performance.

And compared with the Decision Tree algorithm, with only 1.10 seconds more of performance, we can obtain a better accuracy (0.8862) using the Logistic Regression algorithm.

More information in the annexes.

5. Conclusions

As conclusion, we can make the following statements in function of the bank.csv dataset:

First of all, we can say that the decision tree model is the most efficient model based on the time.

We can see too that the Support Vector Machine model takes more runtime, but the accuracy is greater compared to the other models, with the lowest test error too.

And, as a general conclusion, we can confirm that as greater the time that the model takes, better is the accuracy and lower the error.

6. References

[1] Author. (Date). Name of the research. Date when the research was recovered, from https://www.url.com/

[2] Villars, R. L., Olofson, C. W., & Eastwood, M. (2011). Big data: What it is and why you should care. *White paper, IDC*, *14*,1-14.,from http://www.tracemyflows.com/

[3] McCarthy, J. (2007). What is artificial intelligence? Computer *Science Department, Stanford University*, 2, from https://citeseerx.ist.psu.edu/

[4] El Naqa, I., & Murphy, M. J. (2015). What is machine learning?. In Machine Learning in Radiation Oncology (pp. 3-11).Springer,Cham,from https://link.springer.com/

[5] Li, M., Xu, H., & Deng, Y. (2019). Evidential decision tree based on belief entropy. Entropy, 21(9), 897. from https://www.mdpi.com/

[6] Connelly, L. (2020). Logistic regression. Medsurg Nursing, 29(5), 353-354. from https://search.proquest.com/

[7] Alsmadi, M. khalil, Omar, K. B., Noah, S. A., & Almarashdah, I. (2009). Performance Comparison of Multi-layer Perceptron (Back Propagation, Delta Rule and Perceptron) algorithms in Neural Networks. 2009 IEEE International Advance Computing Conference. from https://ieeexplore.ieee.org/

[8] Oliver-Muncharaz, J. (2018). Análisis y clasificación de indicadores técnicos mediante support vector machine. *Finance, Markets and Valuation*, *4*(1), 81-93.

[9] Spark. (2018). Apache spark. Recovered in january, 04th, 2021, from: http://acme.byu.edu/wp-content/uploads/2020/01/Spark.pdf

[10] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger. (2004). An Overview of the Scala Programming Language. Recovered in january, 04th, 2021, from: https://infoscience.epfl.ch/record/52656/files/ScalaOverview.pdf

7. Annexes

[T1] Decision Tree Table

Decision Tree excel table.

Algorithm	Accuracy	Error	Seconds	Attempt
Decision Tree	0.884146341463414	0.115853658536585	23	1
	0.874251497005988	0.125748502994011	12	2
	0.881005173688100	0.118994826311899	11	3
	0.886575735821967	0.113424264178032	9	4
	0.873218304576144	0.126781695423855	10	5
	0.861630516080777	0.138369483919222	9	6
	0.895131086142322	0.104868913857677	10	7
	0.888408927285817	0.111591072714182	9	8
	0.885100074128984	0.114899925871015	11	9
	0.885317750182615	0.114682249817384	10	10
Average	0.881478540637613	0.118521459362386	11.40	

[T2] Logistic Regression Table

Logistic Regression excel table.

Algorithm	Accuracy	Error	Seconds	Attempt
Logistic Regression	0.880365939479240	0.119634060520759	14	1
	0.882010196649672	0.117989803350327	13	2
	0.897869213813372	0.102130786186627	12	3
	0.875891583452211	0.124108416547788	12	4
	0.894584837545126	0.105415162454873	12	5
	0.890350877192982	0.109649122807017	13	6
	0.881824981301421	0.118175018698578	12	7
	0.888467374810318	0.111532625189681	11	8
	0.887218045112781	0.112781954887218	13	9
	0.883875739644970	0.116124260355029	13	10
Average	0.886245878900209	0.113754121099790	12.50	

[T3] Multilayer Perceptron Table

Multilayer Perceptron excel table.

Algorithm	Accuracy	Error	Seconds	Attempt
Multilayer Perceptron	0.870466321243523	0.129533678756476	19	1
	0.881811204911742	0.118188795088257	21	2
	0.874364560639070	0.125635439360929	22	3
	0.892363636363636	0.107636363636363	20	4
	0.890494296577946	0.109505703422053	22	5
	0.892065761258041	0.107934238741958	23	6
	0.883636363636363	0.116363636363636	24	7
	0.895788722341184	0.104211277658815	25	8
	0.883738042678440	0.116261957321559	27	9
	0.887605850654349	0.112394149345650	25	10
Average	0.885233476030429	0.114766523969570	22.80	

[T4] Support Vector Machine Table

Support Vector Machine excel table.

Algorithm	Accuracy	Error	Seconds	Attempt
Support Vector Machine	0.888375673595073	0.111624326404926	20	1
	0.888375673595073	0.111624326404926	19	2
	0.888375673595073	0.111624326404926	21	3
	0.888375673595073	0.111624326404926	22	4
	0.888375673595073	0.111624326404926	23	5
	0.888375673595073	0.111624326404926	25	6
	0.888375673595073	0.111624326404926	25	7
	0.888375673595073	0.111624326404926	37	8
	0.888375673595073	0.111624326404926	57	9
	0.888375673595073	0.111624326404926	62	10
Average	0.888375673595073	0.111624326404926	31.10	