SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

Tecnológico Nacional de México
Instituto Tecnológico de Tijuana

Subdirección Académica

Departamento de sistemas y computación

Febrero-Junio 2021

Datos Masivos

Unidad 4
Proyecto Final

Salazar Ibarra Jesus Rodrigo 16212542

Juan Daniel Camacho Manabe 17210534

Profesor
JOSE CHRISTIAN ROMERO HERNANDEZ

Índice

Video:https://drive.google.com/file/d/10GeDwagTouxqd0UtVULqAFS2qqf5gOFR/view?usp=sharing

Pull Request: https://github.com/ElsellamaJesus/BigData/pull/5

Introducción

En esta práctica evaluatoria concluimos el curso comparando el rendimiento de los siguientes algoritmos de machine learning de Logistic Regression. Los cuales son SVM, Decision Tree y Multilayer perceptron. Posteriormente hablaremos de estos modelos de manera contextual a través de un Marco Teórico y aplicaremos su documentación basándonos en ejemplos con código en Spark - Scala, para así finalmente evaluar una comparativa entre estos tres algoritmos de machine learning representados en una tabla de resultados.

Marco Teórico

Marco Histórico

En el presente año, el término Big Data ha dejado de ser algo desconocido para el mundo y lo podemos notar en las aplicaciones de sus algoritmos de machine learning en distintos ámbitos y sectores de la industria, por ejemplo desde análisis de poderosas compañías de tecnología para tomar decisiones que encaminan a los resultados que mayores beneficios ofrecen, hasta pruebas exhaustivas por parte de algoritmos para cooperar en el descubrimiento de soluciones a enfermedades que antes eran difíciles de comprender o desarrollarse, por el poco manejo que se tenía de la masiva cantidad de datos que se llegaban a recolectar.

Un tema reciente referente a Big Data se vio en el suceso de la pandemia por Covid-19, donde se llevaron a cabo análisis de los datos que los gobiernos proporcionaron, donde se realizaron predicciones sobre el posible crecimiento, para asi gracias a los gráficos obtenidos tomar medidas preventivas para minimizar la transmisión de la enfermedad.

Referente al algoritmo de Árboles de Decisión, en el año 2008 en Canada los autores Brydon, M., Gemino, A. Llevaron a cabo un estudio enfocado en los videojuegos donde por medio del algoritmo de árboles de decisión se desarrolló un modelo para evaluar dicho juego.

En cuanto al algoritmo de Multilayer Perceptron, en el año 2019 los autores Sakar, C.O., Polat, S.O., Katircioglu, M. Desarrollaron una investigación utilizando perceptron multicapa y redes neuronales con la finalidad de predecir en tiempo real la intención de los compradores en línea por medio de un sistema, descubriendo hallazgos interesantes.

Por último, en tanto al tema de regresión logística, en el año 2021 se llevó a cabo un estudio en la India del autores Divya, R., Shantha Selva Kumari, R., donde se aplicó SVM para diagnosticar de manera temprana el padecimiento de Alzheimer.

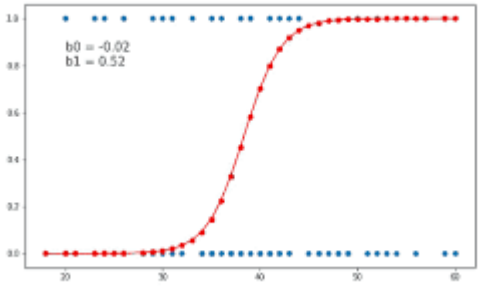
Marco Conceptual

- **Regresión Logística (Logistic Regression):**

El algoritmo de regresión logística es uno de los más utilizados actualmente en aprendizaje automático. Siendo su principal aplicación los problemas de clasificación binaria. Es un algoritmo simple en el que se pueden interpretar fácilmente los resultados obtenidos e identificar por qué se obtiene un resultado u otro. A pesar de su simplicidad funciona realmente bien en muchas aplicaciones y se utiliza como referencia de rendimiento. Por lo tanto, este es un algoritmo con el que los científicos de datos han de estar familiarizados. Ya que comprender los conceptos básicos de la regresión logística son útiles para la entender de otras técnicas más avanzadas.

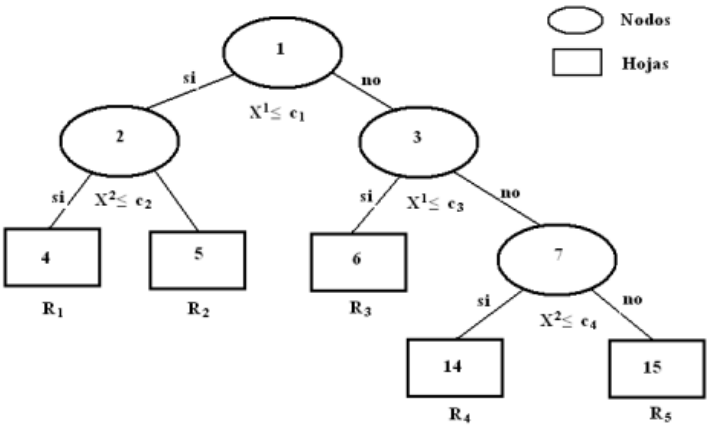
La regresión logística resulta útil para los casos en los que se desea predecir la presencia o ausencia de una característica o resultado según los valores de un conjunto de predictores. Es similar a un modelo de regresión lineal pero está adaptado para modelos en los que la variable dependiente es dicotómica. Los coeficientes de regresión logística pueden utilizarse para estimar la razón de

probabilidad de cada variable independiente del modelo. La regresión logística se puede aplicar a un rango más amplio de situaciones de investigación que el análisis discriminante.



- **Árboles de Decisión (Decision Trees):**

Un árbol de decisión es un modelo predictivo que divide el espacio de los predictores agrupando observaciones con valores similares para la variable respuesta o dependiente.

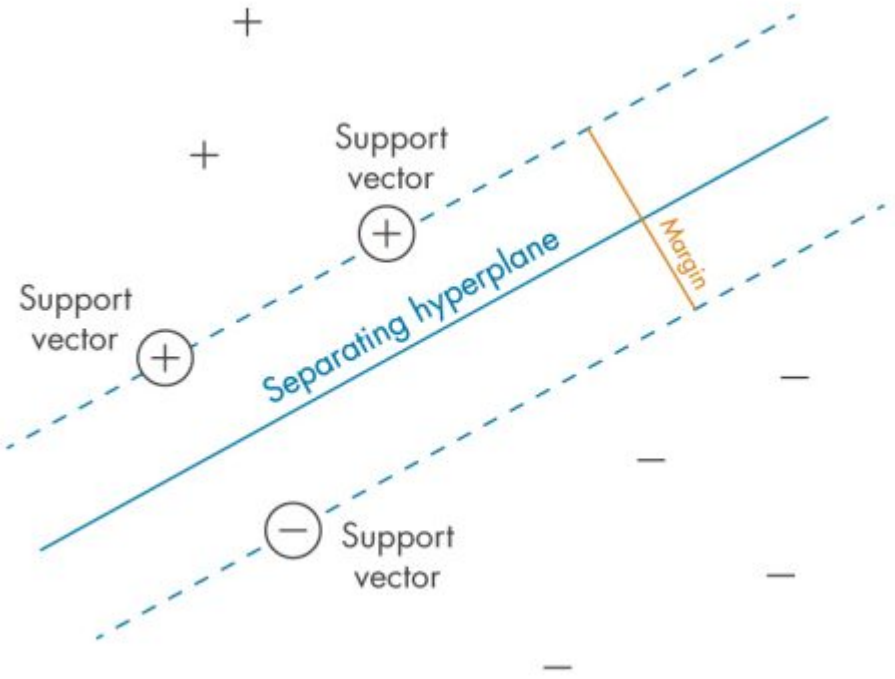


- **Máquinas de vectores de soporte (SVM):**

Las Máquinas de Vectores Soporte constituyen un método basado en aprendizaje para la resolución de problemas de clasificación y regresión. En ambos casos, esta resolución se basa en una primera fase de entrenamiento (donde se les informa con múltiples ejemplos ya resueltos, en forma de pares {problema, solución}) y una segunda fase de uso para la resolución de problemas. En ella, las SVM se convierten en una “caja negra” que proporciona una respuesta (salida) a un problema dado (entrada).

Support vector machine (SVM) es un algoritmo de aprendizaje supervisado que se utiliza en muchos problemas de clasificación y regresión, incluidas aplicaciones médicas de procesamiento de señales, procesamiento del lenguaje natural y reconocimiento de imágenes y voz.

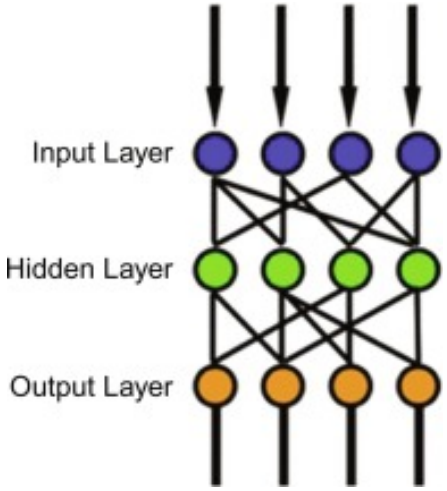
El objetivo del algoritmo SVM es encontrar un hiperplano que separe de la mejor forma posible dos clases diferentes de puntos de datos. “De la mejor forma posible” implica el hiperplano con el margen más amplio entre las dos clases, representado por los signos más y menos en la siguiente figura. El margen se define como la anchura máxima de la región paralela al hiperplano que no tiene puntos de datos interiores. El algoritmo sólo puede encontrar este hiperplano en problemas que permiten separación lineal; en la mayoría de los problemas prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas.



- **Perceptrón multicapa (multilayer perceptron):**

Un perceptrón multicapa (MLP) es una red neuronal artificial de retroalimentación que genera un conjunto de salidas a partir de un conjunto de entradas. Un MLP se caracteriza por varias capas de nodos de entrada conectados como un gráfico dirigido entre las capas de entrada y salida. MLP utiliza la propagación inversa para entrenar la red. MLP es un método de aprendizaje profundo.

El perceptrón multicapa (MLP) es un complemento de la red neuronal de avance. Consta de tres tipos de capas: la capa de entrada, la capa de salida y la capa oculta. La capa de entrada recibe la señal de entrada para ser procesada. La tarea requerida, como la predicción y la clasificación, la realiza la capa de salida. Un número arbitrario de capas ocultas que se colocan entre la capa de entrada y la de salida son el verdadero motor computacional del MLP. De manera similar a una red de alimentación hacia adelante en un MLP, los datos fluyen en la dirección hacia adelante desde la capa de entrada a la de salida. Las neuronas del MLP se entrenan con el algoritmo de aprendizaje de retropropagación. Los MLP están diseñados para aproximarse a cualquier función continua y pueden resolver problemas que no son separables linealmente. Los principales casos de uso de MLP son la clasificación, el reconocimiento, la predicción y la aproximación de patrones.


Input Layer
Hidden Layer
Output Layer

Marco Referencial

Regresión Logística y SVM

Algoritmo genético con selección de características de regresión logística para la clasificación de la enfermedad de Alzheimer

El deterioro cognitivo debe diagnosticarse en la enfermedad de Alzheimer lo antes posible. El diagnóstico temprano le permite a la persona recibir beneficios de tratamiento efectivo además de ayudarla a permanecer independiente por más tiempo. En este artículo, se utilizan diferentes técnicas de selección de características con diferentes clasificadores en la clasificación de esta enfermedad crónica como control normal (NC), deterioro cognitivo leve (MCI) y enfermedad de Alzheimer (EA) basadas en las imágenes de resonancia magnética del conjunto de datos ADNI. La reducción de dimensionalidad juega un papel importante en la mejora del rendimiento de la clasificación cuando hay menos registros con dimensiones altas. Después de diferentes pruebas para seleccionar las características amplias, se encontró que la máquina de vectores de soporte (SVM) con kernel de función de base radial produce mejores resultados con 96.82%, 89.39% y 90.40% de precisión para la clasificación binaria de NC / AD, NC / MCI y MCI / AD, respectivamente, con validación cruzada estratificada diez veces repetida. Al combinar la puntuación del mini examen del estado mental (MMSE) con los datos de la resonancia magnética, ha habido una mejora del 2,7% en la clasificación MCI / AD, pero no tiene mucha influencia en la clasificación NC / AD y NC / MCI.

Árboles de Decisión

Árboles de clasificación y control feedforward analítico de decisiones: un estudio de caso de la industria de los videojuegos

El objetivo de este artículo es utilizar un problema desafiante del mundo real para ilustrar cómo un modelo predictivo probabilístico puede proporcionar la base para el control anticipativo analítico de decisiones. El software de minería de datos comercial y los datos de ventas de una empresa de investigación de mercado se utilizan para crear un modelo predictivo del éxito del mercado en la industria de los videojuegos. Luego se describe un procedimiento para transformar los árboles de

clasificación en un modelo analítico de decisiones que puede resolverse para producir una política de desarrollo de juegos que maximice el valor. El ejemplo del videojuego muestra cómo los modelos predictivos compactos creados por algoritmos de minería de datos pueden ayudar a hacer factible el control anticipativo analítico de decisiones, incluso para problemas grandes y complejos. Sin embargo, el ejemplo también destaca los límites impuestos a la practicidad del enfoque debido a explosiones combinatorias en el número de contingencias que deben modelarse. Mostramos, por ejemplo, cómo el “valor de opción” de las secuelas crea una complejidad que es efectivamente imposible de abordar utilizando herramientas de análisis de decisiones convencionales.

Multilayer Perceptron

Predicción en tiempo real de la intención de compra de los compradores en línea utilizando perceptrón multicapa y redes neuronales recurrentes LSTM

En este documento, proponemos un sistema de análisis del comportamiento del comprador en línea en tiempo real que consta de dos módulos que predice simultáneamente la intención de compra del visitante y la probabilidad de abandono del sitio web. En el primer módulo, predecimos la intención de compra del visitante utilizando datos agregados de visitas a la página que se mantienen durante la visita junto con información de la sesión y del usuario. Las características extraídas se alimentan a bosque aleatorio (RF), máquinas de vectores de soporte (SVM) y clasificadores de perceptrón multicapa (MLP) como entrada. Utilizamos pasos de preprocesamiento de selección de características y sobremuestreo para mejorar el rendimiento y la escalabilidad de los clasificadores. Los resultados muestran que el MLP que se calcula utilizando un algoritmo de retropropagación resistente con retroceso de peso produce una precisión y una puntuación F1 significativamente más altas que RF y SVM. Otro hallazgo es que, aunque los datos del flujo de clics obtenidos de la ruta de navegación seguida durante la visita en línea transmiten información importante sobre la intención de compra del visitante, combinarlos con características basadas en información de la sesión que poseen información única sobre el interés de compra mejora la tasa de éxito del sistema. En el segundo módulo, utilizando sólo datos secuenciales de flujo de clics, entrenamos una red

neuronal recurrente basada en la memoria a corto plazo a largo plazo que genera una salida sigmoidea que muestra la estimación de probabilidad de la intención del visitante de abandonar el sitio sin finalizar la transacción en un horizonte de predicción. Los módulos se utilizan en conjunto para determinar los visitantes que tienen intención de compra pero es probable que abandonen el sitio en el horizonte de predicción y emprendan las acciones correspondientes para mejorar el abandono del sitio web y las tasas de conversión de compra. Nuestros hallazgos respaldan la viabilidad de una predicción precisa y escalable de la intención de compra para el entorno de compra virtual utilizando datos de información de sesión y de flujo de clics.

Implementación

Características del equipo

La ejecución de todos los algoritmos, se llevó a cabo en una computadora con las siguientes características:

- Sistema operativo Linux Ubuntu 20.04.2 LTS de 64 bits
- Procesador Intel Core i5-4590 CPU@3.30GHz de 7ma generación
- 4GB de memoria RAM
- Empleando Visual Studio Code como editor de código
- Empleando la versión de spark 2.11.12
- Con una versión del lenguaje scala 2.4.7
- JDK 8 de Java version 1.8.0_292
- Empleando el dataset banck.csv que se encuentra en la siguiente liga: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

La razón del uso de estas herramientas es porque son con las que hemos trabajado y son las que conocemos mejor, además de ser muy prácticas por supuesto que dependiendo de la cantidad de datos contenidos en el csv tendríamos que, contar con el hardware correspondiente, ya que el esfuerzo computacional matemáticamente hablando es enorme y consume recursos considerables.

Código

Regresión Logística

```
// Import 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,
StringIndexer,VectorIndexer}

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

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

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

//The makes the process of categorizing string variables to numeric
val yes =
df.withColumn("y",when(col("y").equalTo("yes"),1).otherwise(col("y")))
val no =
yes.withColumn("y",when(col("y").equalTo("no"),2).otherwise(col("y")))
val cd = no.withColumn("y",'y.cast("Int"))

// An array of the selected fields is created in assembler
val assembler = new
VectorAssembler().setInputCols(Array("age","balance","day","duration","
campaign","pdays","previous")).setOutputCol("features")

//Transforms into a new dataframe
val features = assembler.transform(cd)

//The column and label are given a new name
val featuresLabel = features.withColumnRenamed("y", "label")

//Select the indexes
val dataIndexed = featuresLabel.select("label","features")
```

```
//Splitting the data in train 70% and test 30%.
val Array(training, test) = dataIndexed.randomSplit(Array(0.7, 0.3))

// We can also use the multinomial family for binary classification
val logisticReg = new
LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam
(0.8).setFamily("multinomial")

// Fit the model
val model = logisticReg.fit(training)

// The test prediction is created
val predictions = model.transform(test)

// An adjustment is made in prediction and label for the final
calculation
val predictionAndLabels =
predictions.select($"prediction",$"label").as[(Double, Double)].rdd

// Results
val metrics = new MulticlassMetrics(predictionAndLabels)
println("Confusion matrix:")
println(metrics.confusionMatrix)
println("Accuracy: " + metrics.accuracy)
println(s"Test Error: ${(1.0 - metrics.accuracy)}")
```

Árboles de Decisión

```
//Importar Librerias
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, StringIndexer,
VectorIndexer}
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.sql.SparkSession

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

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

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

//The makes the process of categorizing string variables to numeric
val yes =
df.withColumn("y",when(col("y").equalTo("yes"),1).otherwise(col("y")))
val cl =
yes.withColumn("y",when(col("y").equalTo("no"),2).otherwise(col("y")))

// An array of the selected fields is created in assembler
val assembler = new
VectorAssembler().setInputCols(Array("age","balance","day","duration","c
ampaign","pdays","previous")).setOutputCol("features")

//Transforms into a new dataframe
val df2 = assembler.transform(df)

//The column and label are given a new name
val featuresLabel = df2.withColumnRenamed("y", "label")

//Select the indexes
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(da
taIndexed)
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))

//Creating the Decision Tree object.
val dt = new
DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("ind
exedFeatures")

//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, dt, labelConverter))

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

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

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

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

//Accuracy and Test Error
val accuracy = evaluator.evaluate(predictions)
println(s"Test Error = ${(1.0 - accuracy)}")

//Show Decision Tree Model
val treeModel =
```

```
model.stages(2).asInstanceOf[DecisionTreeClassificationModel]
println(s"Learned classification tree model:\n
${treeModel.toDebugString}")
```

SVM

```
// Import Libraries
import org.apache.spark.sql.SparkSession
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.ml.classification.{LinearSVC,
LogisticRegression}
import org.apache.spark.ml.feature.{StringIndexer, VectorIndexer,
VectorAssembler}

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

// Spark Session
val spark = SparkSession.builder.getOrCreate()

// The data from the bank-full.csv dataset is imported
val df  = spark.read.option("header","true").option("inferSchema",
"true").option("delimiter",";").format("csv").load("/home/js/Documentos
/GitHub/BigData/Unit_4/Proyecto/bank-full.csv")

//The makes the process of categorizing string variables to numeric
val labelIndexer = new
StringIndexer().setInputCol("y").setOutputCol("indexedLabel").fit(df)
val indexed =
labelIndexer.transform(df).drop("y").withColumnRenamed("indexedLabel",
"label")

// An array of the selected fields is created in assembler
val vectorFeatures = (new
VectorAssembler().setInputCols(Array("age","balance","day","duration","
campaign","pdays","previous")).setOutputCol("features"))

//Transforms into a new dataframe
```

```
val features = vectorFeatures.transform(indexed)

// The column and label are given a new name
val featuresLabel = features.withColumnRenamed("y", "label")

// Select the indexes
val dataIndexed = featuresLabel.select("label","features")
val labelIndexer = new
StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(d
ataIndexed)
val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures")
.setMaxCategories(4).fit(dataIndexed)

// Splitting the data in train 70% and test 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)

// Fit the model
val model = supportVM.fit(training)

// The test prediction is created
val predictions = model.transform(test)

// An adjustment is made in prediction and label for the final
calculation
val predictionAndLabels =
predictions.select($"prediction",$"label").as[(Double, Double)].rdd

// Results
val metrics = new MulticlassMetrics(predictionAndLabels)
println("Confusion matrix:")
println(metrics.confusionMatrix)
println("Accuracy: " + metrics.accuracy)
println(s"Test Error = ${(1.0 - metrics.accuracy)}")
```

Perceptrón Multicapa

```
// Import libraries
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.IntegerType
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import
org.apache.spark.ml.classification.MultilayerPerceptronClassifier
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}
//Error level code.
import org.apache.log4j._
Logger.getLogger("org").setLevel(Level.ERROR)
//Spark session.
val spark =
SparkSession.builder.appName("MultilayerPerseptron").getOrCreate()
//Reading the csv file.
val df  = spark.read.option("header","true").option("inferSchema",
"true").option("delimiter",";").format("csv").load("bank-full.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)
//Fitting indexed and finding labels 0 and 1.
val labelIndexer = new
StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(i
ndexed)
//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).se
tSeed(1234L).setMaxIter(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().setMetricName("accuracy")

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

Resultados

Tabla de resultados de precisión de Algoritmo

No.	Logistic Regression	Decision Tree	SVM	Multilayer Perceptron
1	0,881135	0,892828	0,886322	0,880824
2	0,882846	0,887307	0,879944	0,884485
3	0,883460	0,889872	0,884788	0,884795
4	0,884525	0,891319	0,882414	0,882563
5	0,880431	0,889358	0,883476	0,883021
6	0,882164	0,893579	0,879015	0,879925
7	0,882019	0,894795	0,883693	0,883052
8	0,885966	0,890138	0,883365	0,880328
9	0,884169	0,889715	0,882754	0,883674
10	0,885501	0,894795	0,883359	0,880072
11	0,880930	0,893716	0,879455	0,883646
12	0,882652	0,885450	0,884667	0,883216
13	0,882331	0,894791	0,883445	0,882296
14	0,878913	0,890865	0,883466	0,881482
15	0,883110	0,888791	0,888370	0,882085
16	0,882744	0,892801	0,885921	0,883214
17	0,884238	0,890242	0,883666	0,882548
18	0,884437	0,892389	0,884171	0,886386
19	0,886447	0,892265	0,882254	0,882924
20	0,881059	0,894830	0,879554	0,882262
21	0,883417	0,889342	0,882142	0,880147
22	0,886649	0,892090	0,883488	0,883776
23	0,883306	0,889942	0,885265	0,883599
24	0,884325	0,888660	0,883001	0,883842
25	0,885999	0,893104	0,879934	0,881083
26	0,882396	0,891285	0,880767	0,880818
27	0,884096	0,893551	0,887037	0,879956
28	0,882782	0,890089	0,885622	0,882871
29	0,884221	0,892237	0,886342	0,885022
30	0,885226	0,888715	0,882484	0,883027
Promedio	0,883383	0,891295	0,883339	0,882565

Approx Time (Sec)	15.41	15.92	21.67	21.09

Observaciones:

En cuanto a los resultados de nuestra tabla, después de una muestra de 45,212 registros con 30 iteraciones de los algoritmos de regresión logística, árboles de decisión, SVM y perceptrón multicapa respectivamente, tenemos una tabla que nos muestra una clara comparativa entre los cuatro en cuanto a su posición. En general podría concluirse que el algoritmo de regresión logística fue el mejor en esta ocasión.

Posición	Precisión	Tiempo
1	**Regresión Logística**	**Árboles de Decisión**
2	**Árboles de Decisión**	**SVM**
3	**Perceptrón Multicapa**	**Regresión Logística**
4	**SVM**	**Perceptrón Multicapa**

Conclusiones

En general los algoritmos son muy rápidos, incluso existen algunos especialmente optimizados, la realidad es que los algoritmos varían en algunos segundos en procesamiento, ciertamente en términos humanos no es mucho , pero en tiempo máquina podría ser relevante. Cada algoritmo se ejecuta bajo un modelo matemático diferente y aun así todos los algoritmos se aproximan alrededor del 88% de precisión al menos para este dataset en particular.

Referencias

Conceptos

- IBM. (2014, 19 noviembre). *Regresión Logística*. ibm.com. https://www.ibm.com/docs/es/spss-statistics/SaaS?topic=regression-logistic
- Merayo, P. (2021, 22 febrero). *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/
- MathWorks. (s. f.). *Support Vector Machine (SVM)*. MATLAB & Simulink. Recuperado 17 de junio de 2021, de https://la.mathworks.com/discovery/support-vector-machine.html
- DeepAI. (2020, 25 junio). *Multilayer Perceptron*. https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron
- Machine Learning y Support Vector Machines: porque el tiempo es dinero | Blog. (2020, 1 septiembre). Merkle. https://www.merkleinc.com/es/es/blog/machine-learning-support-vector-machines
- Multilayer Perceptron - an overview | ScienceDirect Topics. (2020). ScienceDirect. https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron
- Techopedia. (2017, 30 marzo). Multilayer Perceptron (MLP). Techopedia.Com. https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp

- Rodríguez, D. (2018, 1 julio). La regresión logística. Analytics Lane. https://www.analyticslane.com/2018/07/23/la-regresion-logistica/

Artículos

- Divya, R., Shantha Selva Kumari, R. y la Iniciativa de neuroimagen de la enfermedad de Alzheimer. Algoritmo genético con selección de características de regresión logística para la clasificación de la enfermedad de Alzheimer. Computación neuronal y aplicación (2021). https://doi.org/10.1007/s00521-020-05596-x
- Thabtah, F., Abdelhamid, N. & Peebles, D. A machine learning autism classification based on logistic regression analysis. Health Inf Sci Syst 7, 12 (2019). https://doi.org/10.1007/s13755-019-0073-5
- Sadat-Hashemi, S.M., Kazemnejad, A., Lucas, C. et al. Predicting the type of pregnancy using artificial neural networks and multinomial logistic regression: a comparison study. Neural Comput & Applic 14, 198–202 (2005). https://doi.org/10.1007/s00521-004-0454-8
- Brydon, M., Gemino, A. Classification trees and decision-analytic feedforward control: a case study from the video game industry. Data Min Knowl Disc 17, 317–342 (2008). https://doi.org/10.1007/s10618-007-0086-6
- Vijayashree, J., Sultana, H.P. A Machine Learning Framework for Feature Selection in Heart Disease Classification Using Improved Particle Swarm Optimization with Support Vector Machine Classifier. Program Comput Soft 44, 388–397 (2018). https://doi.org/10.1134/S0361768818060129
- Sakar, C.O., Polat, S.O., Katircioglu, M. *et al.* Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. *Neural Comput & Applic* 31, 6893–6908 (2019). https://doi.org/10.1007/s00521-018-3523-0