TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA
DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN
Semestre: Agosto - Diciembre 2019

CARRERA
Ingeniería en Sistemas Computacionales

MATERIA
Datos Masivos

CLAVE:
BDD-1704TI9A

ALUMNOS:
Salcido Guerrero Raúl Alejandro - 15211356

DOCENTE:
MC JOSÉ CHRISTIAN ROMERO

Introducción

Machine Learning es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Aprender en este contexto quiere decir identificar patrones complejos en millones de datos. La máquina que realmente aprende es un algoritmo que revisa los datos y es capaz de predecir comportamientos futuros. Automáticamente, también en este contexto, implica que estos sistemas se mejoran de forma autónoma con el tiempo, sin intervención humana. El aspecto iterativo del machine learning es importante porque a medida que los modelos son expuestos a nuevos datos, éstos pueden adaptarse de forma independiente. Aprenden de cálculos previos para producir decisiones y resultados confiables y repetibles. Es una ciencia que no es nueva – pero que ha cobrado un nuevo impulso.

Contenido del proyecto

Objetivo: Comparación del rendimiento siguientes algoritmos de machine learning

- SVM
- Decision Three
- Logistic Regression
- Multilayer perceptron

Con el siguiente data set: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Contenido del documento de proyecto final

1. Portada
2. Índice
3. Introducción
4. Marco teórico de los algoritmos
5. Implementación (Qué herramientas usaron y porque (en este caso spark con scala))
6. Resultados (Un tabular con los datos por cada algoritmo para ver su performance) y su respectiva explicación.
7. Conclusiones
8. Referencias (No wikipedia por ningún motivo, trate que sean de artículos científicos)

Marco Teórico

SVM

Es un clasificador discriminativo definido formalmente por un hiperplano de separación. En otras palabras, dados los datos de entrenamiento etiquetados (aprendizaje supervisado), el algoritmo genera un hiperplano óptimo que categoriza nuevos ejemplos. En dos espacios dimensionales, este hiperplano es una línea que divide un plano en dos partes donde en cada clase se encuentra a cada lado.

¿Qué hace SVM?

Dado un conjunto de ejemplos de entrenamiento, cada uno marcado como perteneciente a una u otra de dos categorías, un algoritmo de entrenamiento SVM construye un modelo que asigna nuevos ejemplos a una categoría u otra, convirtiéndolo en un clasificador lineal binario no probabilístico.

Support Vector Machine (SVM) es principalmente un método más clásico que realiza tareas de clasificación mediante la construcción de hiperplanos en un espacio multidimensional que separa los casos de diferentes etiquetas de clase. SVM admite tareas de regresión y clasificación y puede manejar múltiples variables continuas y categóricas. Para las variables categóricas, se crea una variable ficticia con valores de casos como 0 o 1. Por lo tanto, una variable dependiente categórica que consta de tres niveles, digamos (A, B, C), está representada por un conjunto de tres variables ficticias: A: {1 0 0}, B: {0 1 0}, C: {0 0 1}

Para construir un hiperplano óptimo, SVM emplea un algoritmo de entrenamiento iterativo, que se utiliza para minimizar una función de error. Según la forma de la función de error, los modelos SVM se pueden clasificar en cuatro grupos distintos:

- Clasificación SVM Tipo 1 (también conocida como clasificación C-SVM)
- Clasificación SVM tipo 2 (también conocida como clasificación nu-SVM)
- Regresión SVM tipo 1 (también conocida como regresión epsilon-SVM)
- Regresión SVM tipo 2 (también conocida como regresión nu-SVM)

Parámetros de ajuste: Kernel, Regularización, Gamma y Margen

Kernel

El aprendizaje del hiperplano en SVM lineal se realiza transformando el problema utilizando algo de álgebra lineal. Aquí es donde el kernel juega un papel.

Para el kernel lineal, la ecuación para la predicción de una nueva entrada utilizando el producto de punto entre la entrada (x) y cada vector de soporte (xi) se calcula de la siguiente manera:

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

Esta es una ecuación que implica calcular los productos internos de un nuevo vector de entrada (x) con todos los vectores de soporte en los datos de entrenamiento. El coeficiente de aprendizaje debe estimar los coeficientes B0 y ai (para cada entrada) a partir de los datos de entrenamiento.

El núcleo polinomial se puede escribir como K (x, xi) = 1 + sum (x * xi) ^ d y exponencial como K (x, xi) = exp (-gamma * sum ((x - xi^2)).

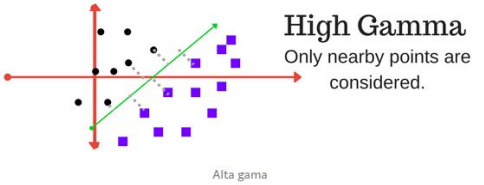
Regularización

El parámetro Regularización (a menudo denominado parámetro C en la biblioteca sklearn de python) le dice a la optimización SVM cuánto desea evitar clasificar erróneamente cada ejemplo de entrenamiento.

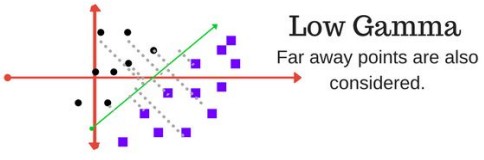
Para valores grandes de C, la optimización elegirá un hiperplano de menor margen si ese hiperplano hace un mejor trabajo al clasificar correctamente todos los puntos de entrenamiento. Por el contrario, un valor muy pequeño de C hará que el optimizador busque un hiperplano de separación de mayor margen, incluso si ese hiperplano clasifica erróneamente más puntos.

Gama

El parámetro gamma define qué tan lejos llega la influencia de un solo ejemplo de entrenamiento, con valores bajos que significan "lejos" y valores altos que significan "cerca". En otras palabras, con gama baja, los puntos alejados de la línea de separación plausible se consideran en el cálculo de la línea de separación. Donde como gama alta significa que los puntos cercanos a la línea plausible se consideran en el cálculo.

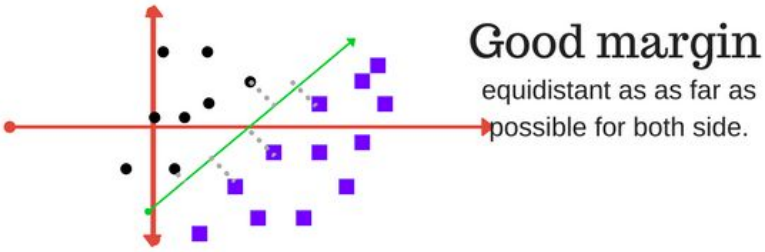


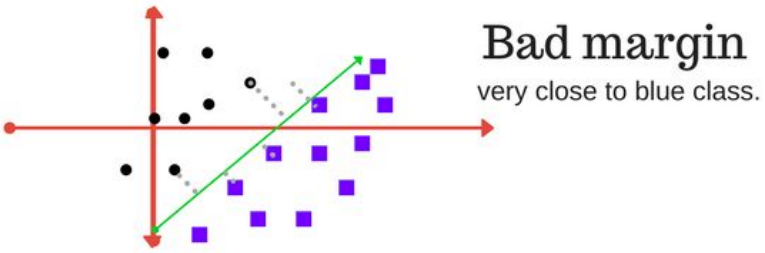
Alta gama



Margen
Y finalmente la última pero muy importante característica del clasificador SVM. SVM to core intenta lograr un buen margen. Un margen es una separación de línea a los puntos de clase más cercanos.

Un buen margen es aquel en el que esta separación es mayor para ambas clases. Las imágenes a continuación dan un ejemplo visual de buenos y malos márgenes. Un buen margen permite que los puntos estén en sus respectivas clases sin cruzar a otra clase.





Ventajas y Desventajas

Ventajas:

- Algoritmo clasificador en base a una sólida teoría. Teoremas de minimización de riesgo son el estado del arte en aprendizaje estadístico.
- Se puede aplicar a datos representados en cualquier espacio de Hilbert (donde pueda definir una medida de distancia).
- Relativamente pocos parámetros a estimar.
- Se pueden formular nuevas extensiones (flexibilidad).

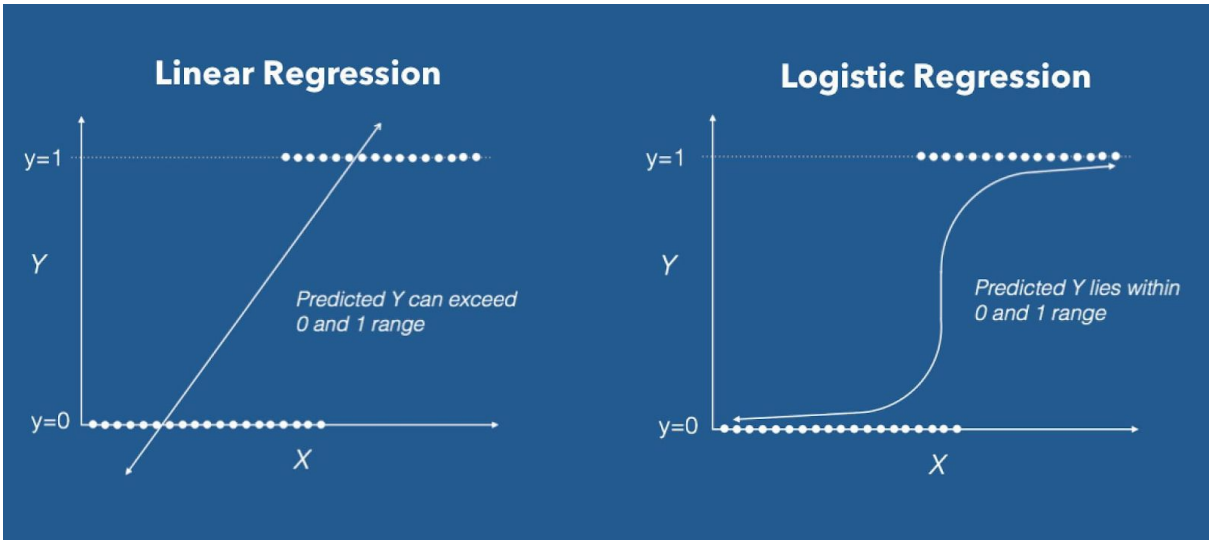
Desventajas:

- Determinar los Kernels a utilizar es complejo.
- Sólo es un clasificador binario.

Logistic Regression

Es un algoritmo de clasificación utilizado para asignar observaciones a un conjunto discreto de clases. Algunos de los ejemplos de problemas de clasificación son correo electrónico no deseado, transacciones en línea fraudes o no fraude, tumor maligno o benigno. La regresión logística transforma su salida usando la función sigmoide logística para devolver un valor de probabilidad.

La regresión logística es un algoritmo de aprendizaje automático que se utiliza para los problemas de clasificación, es un algoritmo de análisis predictivo y se basa en el concepto de probabilidad.

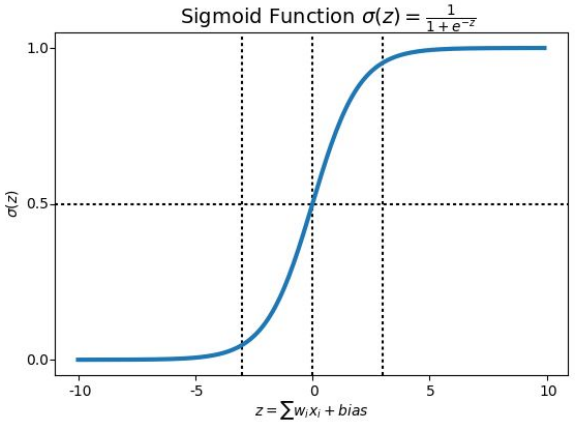


Podemos llamar a una regresión logística un modelo de regresión lineal, pero la regresión logística utiliza una función de costo más compleja, esta función de costo puede definirse como la ' función sigmoidea ' o también conocida como 'función logística' en lugar de una función lineal.

La hipótesis de la regresión logística tiende a limitar la función de costo entre 0 y 1. Por lo tanto, las funciones lineales no lo representan, ya que puede tener un valor mayor que 1 o menor que 0, lo que no es posible según la hipótesis de la regresión logística.

¿Qué es la función sigmoidea?

Para asignar los valores pronosticados a las probabilidades, utilizamos la función Sigmoide. La función asigna cualquier valor real a otro valor entre 0 y 1. En el aprendizaje automático, utilizamos sigmoide para asignar predicciones a probabilidades.



¿Cómo se usa la regresión logística?

La regresión logística sólo debe usarse cuando las variables objetivo caen en categorías discretas y que si hay un rango de valores continuos que podría ser el valor objetivo, la regresión logística no debería usarse. Los ejemplos de situaciones en las que podría usar la regresión logística incluyen:

- Predecir si un correo electrónico es spam o no spam
- Si un tumor es maligno o benigno
- Si un hongo es venenoso o comestible.

Cuando se utiliza la regresión logística, generalmente se especifica un umbral que indica a qué valor se colocará el ejemplo en una clase frente a la otra clase. En la tarea de clasificación de correo no deseado, se puede establecer un umbral de 0,5, lo que haría que un correo electrónico con una probabilidad del 50% o más de ser correo no deseado se clasifique como "correo no deseado" y cualquier correo electrónico con una probabilidad inferior al 50% se clasifique como "no correo no deseado" ".

Aunque la regresión logística es más adecuada para instancias de clasificación binaria, se puede aplicar a problemas de clasificación multiclase, tareas de clasificación con tres o más clases.

Ventajas y Desventajas

Ventajas de la regresión logística

1. La regresión logística funciona bien cuando el conjunto de datos es linealmente separable.

2. La regresión logística es menos propensa al sobreajuste, pero puede sobreajustar en conjuntos de datos de alta dimensión. Debe considerar las técnicas de regularización (L1 y L2) para evitar el ajuste excesivo en estos escenarios.

3. La regresión logística no solo da una medida de cuán relevante es un predictor (tamaño del coeficiente), sino también su dirección de asociación (positiva o negativa).

4. La regresión logística es más fácil de implementar, interpretar y muy eficiente de entrenar.

Desventajas de la regresión logística

1. La principal limitación de la regresión logística es el supuesto de linealidad entre la variable dependiente y las variables independientes. En el mundo real, los datos rara vez son linealmente separables. La mayoría de las veces los datos serían un desastre desordenado.

2. Si el número de observaciones es menor que el número de características, no se debe utilizar Regresión logística, de lo contrario, puede producir un sobreajuste.

3. La regresión logística solo puede usarse para predecir funciones discretas. Por lo tanto, la variable dependiente de Regresión logística está restringida al conjunto de números discretos. Esta restricción en sí misma es problemática, ya que es prohibitiva para la predicción de datos continuos.

Decision Tree

Un árbol de decisiones es una herramienta de soporte de decisiones que utiliza un gráfico o modelo de decisiones en forma de árbol y sus posibles consecuencias, incluidos los resultados de eventos fortuitos, los costos de recursos y la utilidad. Es una forma de mostrar un algoritmo que solo contiene sentencias de control condicional.

Un árbol de decisión es una estructura similar a un diagrama de flujo en el que cada nodo interno representa una "prueba" en un atributo (por ejemplo, si una moneda lanza cara o cruz), cada rama representa el resultado de la prueba y cada nodo hoja representa un etiqueta de clase (decisión tomada después de calcular todos los atributos). Los caminos de raíz a hoja representan reglas de clasificación.

¿Cómo funciona el árbol de decisión?

El árbol de decisión es un tipo de algoritmo de aprendizaje supervisado (que tiene una variable objetivo predefinida) que se usa principalmente en problemas de clasificación. Funciona para variables de entrada y salida categóricas y continuas. En esta técnica, dividimos la población o muestra en dos o más conjuntos homogéneos (o subpoblaciones) basados en el divisor / diferenciador más significativo en las variables de entrada.

Tipos de árboles de decisión

Los tipos de árbol de decisión se basan en el tipo de variable objetivo que tenemos. Puede ser de dos tipos:

1. **Árbol de decisión de variable categórica:** Árbol de decisión que tiene una variable objetivo categórica y luego se llama árbol de decisión de variable categórica.
2. **Árbol de decisión de variable continua:** el árbol de decisión tiene una variable objetivo continua y luego se llama árbol de decisión de variable continua.

Construcción del árbol de decisión

La manera en la que el árbol puede obtener información es dividiendo el conjunto de fuentes en subconjuntos basados en una prueba de valor de atributo. Este proceso se repite en cada subconjunto derivado de una manera *recursiva* llamada *partición recursiva* . La recursión se completa cuando el subconjunto en un nodo tiene el mismo valor de la variable objetivo, o cuando la división ya no agrega valor a las predicciones. La construcción del clasificador de árbol de decisión no requiere ningún conocimiento de dominio o configuración de parámetros y, por lo tanto, es apropiado para el descubrimiento de conocimiento exploratorio. Los árboles de decisión pueden manejar datos de alta dimensión. En general, el clasificador de árbol de decisión tiene buena precisión. La inducción del árbol de decisión es un enfoque inductivo típico para aprender el conocimiento sobre la clasificación.

Fortalezas y debilidades

Las fortalezas de los métodos del árbol de decisión son:

- Los árboles de decisión pueden generar reglas comprensibles.
- Los árboles de decisión realizan la clasificación sin requerir muchos cálculos.
- Los árboles de decisión pueden manejar variables continuas y categóricas.

- Los árboles de decisión proporcionan una indicación clara de qué campos son más importantes para la predicción o clasificación.

Las debilidades de los métodos del árbol de decisión:

- Los árboles de decisión son menos apropiados para las tareas de estimación donde el objetivo es predecir el valor de un atributo continuo.
- Los árboles de decisión son propensos a errores en los problemas de clasificación con muchas clases y un número relativamente pequeño de ejemplos de capacitación.
- El árbol de decisión puede ser computacionalmente costoso de entrenar. El proceso de crecimiento de un árbol de decisión es computacionalmente costoso. En cada nodo, cada campo de división candidato debe ordenarse antes de poder encontrar su mejor división. En algunos algoritmos, se utilizan combinaciones de campos y se debe realizar una búsqueda para obtener pesos de combinación óptimos. Los algoritmos de poda también pueden ser costosos ya que se deben formar y comparar muchos subárboles candidatos.

Implementación

Apache Spark

Apache Spark es un motor de procesamiento distribuido responsable de orquestar, distribuir y monitorear aplicaciones que constan de múltiples tareas de procesamiento de datos sobre varias máquinas de trabajo, que forman un cluster.

Respecto a su propósito general, la virtud de Spark es estar diseñado para cubrir una amplia gama de cargas de trabajo que previamente requerían sistemas distribuidos diferentes. Éstos sistemas incluyen procesamiento batch, algoritmos iterativos, queries interactivas, procesamiento streaming… a menudo empleados todos ellos en un pipeline típico de análisis de datos.

Spark es flexible en su utilización, y ofrece una serie de APIs que permiten a usuarios con diferentes backgrounds poder utilizarlo. Incluye APIs de Python, Java, Scala, SQL y R, con funciones integradas y en general una performance razonablemente buena en todas ellas.
Permite trabajar con datos más o menos estructurados (RDDs, dataframes, datasets) dependiendo de las necesidades y preferencias del usuario.

Características principales

- Trabaja en memoria, con lo que se consigue mucha mayor velocidad de procesamiento.
- También permite trabajar en disco. De esta manera si por ejemplo tenemos un fichero muy grande o una cantidad de información que no cabe en memoria, la herramienta permite almacenar parte en disco, lo que hace perder velocidad. Esto hace que tengamos que intentar encontrar el equilibrio entre lo que se almacena en memoria y lo que se almacena en disco, para tener una buena velocidad y para que el coste no sea demasiado elevado, ya que la memoria siempre es bastante más cara que el disco.
- Nos proporciona API para Java, Scala, Python y R.
- Permite el procesamiento en tiempo real, con un módulo llamado Spark Streaming, que combinado con Spark SQL nos va a permitir el procesamiento en tiempo real de los datos. Conforme vayamos inyectando los datos podemos ir transformándolos y volcándolos a un resultado final.
- Resilient Distributed Dataset (RDD): Usa la evaluación perezosa, lo que significa es que todas las transformaciones que vamos realizando sobre los RDD, no se resuelven, si no que se van almacenando en un grafo acíclico dirigido (DAG), y cuando ejecutamos una acción, es decir, cuando la herramienta no tenga más

opción que ejecutar todas las transformaciones, será cuando se ejecuten. Esto es un arma de doble filo, ya que tiene una ventaja y un inconveniente. La ventaja es que se gana velocidad al no ir realizando las transformaciones continuamente, sino solo cuando es necesario. El inconveniente es que si alguna transformación eleva algún tipo de excepción, la misma no se va a detectar hasta que no se ejecute la acción, por lo que es más difícil de debuggear o programar.

Logistic Regression

```
/*Importamos las librerías*/
import org.apache.spark.ml.classification.LogisticRegression
import org.apache.spark.ml.feature.{VectorAssembler, StringIndexer,
VectorIndexer, OneHotEncoder}
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.sql.SparkSession
val spark = SparkSession.builder().getOrCreate()

/*Se importan los datos del CSV*/
val data  = spark.read.option("header","true").option("inferSchema",
"true").option("delimiter",";").format("csv").load("bank-full.csv")

/*Se categoriza las variables de string a valor numérico*/
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"))

/*Creación del Array con los datos seleccionados*/
val featureCols = Array("age","previous","balance","duration")

/*Creación del Vector en base a los features*/
val assembler = new
VectorAssembler().setInputCols(featureCols).setOutputCol("features")

/*Transformación a un nuevo DF*/
val df2 = assembler.transform(cleanData)

/*Rename de columnas*/
val featuresLabel = df2.withColumnRenamed("y", "label")

/*Selección de index*/
val dataI = featuresLabel.select("label","features")
```

```
/*Creación del Array con los datos de entrenamiento y test*/
val Array(training, test) = dataI.randomSplit(Array(0.7, 0.3), seed =
12345)

/*Modelo de Regresion*/
val lr = new
LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(
0.8)

val lrModel = lr.fit(training)

/*Impresión de los coeficientes e intercepciones*/
println(s"Coefficients: \n${lrModel.coefficientMatrix}")
println(s"Intercepts: \n${lrModel.interceptVector}")


/*Impresión de la precisión*/
println(s"Accuracy: $accuracy")
```

Support Vector Machine

```
/*Importamos las librerias necesarias con las que vamos a trabajar*/
import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.{SparkSession, SQLContext}
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.Transformer
import org.apache.spark.ml.classification.LinearSVC
import org.apache.log4j._

/*Quita los warnings*/
Logger.getLogger("org").setLevel(Level.ERROR)

/*Creamos una sesion de spark y cargamos los datos del CSV en un
datraframe*/
val spark = SparkSession.builder().getOrCreate()
val df =
spark.read.option("header","true").option("inferSchema","true").option("
delimiter",";").format("csv").load("bank-full.csv")

/*Desblegamos los tipos de datos.*/
df.printSchema()
df.show(1)

/*Cambiamos la columna y por una con datos binarios.*/
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"))

/*Desplegamos la nueva columna*/
newcolumn.show(1)

/*Generamos la tabla features*/
val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration","pdays",
"previous")).setOutputCol("features")
```

```
val fea = assembler.transform(newcolumn)

/*Mostramos la nueva columna*/
fea.show(1)

/*Cambiamos la columna y a la columna label*/
val cambio = fea.withColumnRenamed("y", "label")
val feat = cambio.select("label","features")
feat.show(1)

/*SVM*/
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("l
abel")))
val c3 = c2.withColumn("label",'label.cast("Int"))
val linsvc = new LinearSVC().setMaxIter(10).setRegParam(0.1)

/* Fit del modelo*/
val linsvcModel = linsvc.fit(c3)

/*Imprimimos linea de intercepcion*/
println(s"Coefficients: ${linsvcModel.coefficients} Intercept:
${linsvcModel.intercept}")
println(s"Accuracy: $accuracy")
```

Decision Tree

```
//Importamos las librerias necesarias con las que vamos a trabajar
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.DateType
import org.apache.spark.sql.{SparkSession, SQLContext}
import org.apache.spark.ml.feature.VectorIndexer
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.Transformer
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils
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.DecisionTreeClassificationModel
import org.apache.log4j._

//Quita los warnings
Logger.getLogger("org").setLevel(Level.ERROR)

//Creamos una sesion de spark y cargamos los datos del CSV en un
datraframe
val spark = SparkSession.builder().getOrCreate()
val df =
spark.read.option("header","true").option("inferSchema","true").option("
delimiter",";").format("csv").load("bank-full.csv")

//Desblegamos los tipos de datos.
df.printSchema()
df.show(1)

//Cambiamos la columna y por una con datos binarios.
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"))
//Desplegamos la nueva columna
newcolumn.show(1)

//Generamos la tabla features
val assembler = new
VectorAssembler().setInputCols(Array("balance","day","duration","pdays",
"previous")).setOutputCol("features")
val fea = assembler.transform(newcolumn)
//Mostramos la nueva columna
fea.show(1)

//Cambiamos la columna y a la columna label
val cambio = fea.withColumnRenamed("y", "label")
val feat = cambio.select("label","features")
feat.show(1)

//DecisionTree
val labelIndexer = new
StringIndexer().setInputCol("label").setOutputCol("indexedLabel").fit(fe
at)

// features con mas de 4 valores distinctivos son tomados como continuos
val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").
setMaxCategories(4)

//Division de los datos entre 70% y 30% en un arreglo
val Array(trainingData, testData) = feat.randomSplit(Array(0.7, 0.3))

//Creamos un objeto DecisionTree
val dt = new
DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("ind
exedFeatures")

//Rama de prediccion
val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel")
.setLabels(labelIndexer.labels)

//Juntamos los datos en un pipeline
val pipeline = new Pipeline().setStages(Array(labelIndexer,
featureIndexer, dt, labelConverter))

//Create a model of the entraining
```

```
val model = pipeline.fit(trainingData)

//Transformacion de datos en el modelo
val predictions = model.transform(testData)

//Desplegamos predicciones
predictions.select("predictedLabel", "label", "features").show(5)

//Evaluamos la exactitud
val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredi
ctionCol("prediction").setMetricName("accuracy")
val accuracy = evaluator.evaluate(predictions)
println(s"Test Error = ${(1.0 - accuracy)}")

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

Resultados

Iteracion	Decision Tree	Logistic Regression	SVM
1	89.83%	88.32%	88.90%
2	89.83%	88.32%	88.90%
3	89.83%	88.32%	88.90%
4	89.83%	88.32%	88.90%
5	89.83%	88.32%	88.90%
6	89.83%	88.32%	88.90%
7	89.83%	88.32%	88.90%
8	89.83%	88.32%	88.90%
9	89.83%	88.32%	88.90%
10	89.83%	88.32%	88.90%
11	89.83%	88.32%	88.90%
12	89.83%	88.32%	88.90%
13	89.83%	88.32%	88.90%
14	89.83%	88.32%	88.90%
15	89.83%	88.32%	88.90%
Promedio	89.83%	88.32%	88.90%

Conclusión

Si bien los buenos datos superan cualquier algoritmo en cualquier momento. En este caso, analizando estos tres algoritmos personalmente diría que el mejor de ellos es “Logistic Regression”. Aunque está perdiendo terreno frente a otras técnicas con el progreso en la eficiencia y la facilidad de implementación de otros algoritmos complejos. Aunque si nos sujetamos al resultado numérico obtenido en las pruebas queda más que claro que el ganador fue “Decision Tree” y para este caso es la mejor solución.

Referencias

SVM

- https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
- https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/
- http://www.statsoft.com/textbook/support-vector-machines
- https://codeday.me/es/qa/20190518/711022.html

Decision Tree

- https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb
- https://www.geeksforgeeks.org/decision-tree/
- https://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/

Logistic Regression

- https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148
- https://kambria.io/blog/logistic-regression-for-machine-learning/
- https://machinelearning-blog.com/2018/04/23/logistic-regression-101/

Apache Spark y Scala

- https://openwebinars.net/blog/que-es-apache-spark/
- https://www.icemd.com/digital-knowledge/articulos/apache-spark-introduccion-que-es-y-como-funciona/