Máster en Big Data Deportivo

Módulo 5. Análisis de Datos Deportivos y Machine Learning con Big Data

UCAM
UNIVERSIDAD
CATÓLICA DE MURCIA

Índice

1. Introducción 4
2. Apache Spark 6
   2.1. Funcionalidades Principales 8
   2.2. Contextos Spark 13
3. Instalación y familiarización con la herramienta 19
   3.1. Notebook Cloud (Databricks) 19
   3.2. Standalone Setup 21
4. Trabajando con Apache Spark en Python 27
   4.1. Paquetes y Librerías 27
   4.2. Creación de RDDs 29
   4.3. Operaciones sobre RDDs 30
   4.4. API Dataframes 32
   4.5. API Datasets 34
5. Caso de uso Spark: análisis datos competición 38
6. Ecosistema de Apache Spark. Módulos principales 50
7. Caso de uso de Apache Spark (I). Spark SQL 52
8. Introducción a Machine Learning o Aprendizaje Automático (I) 56
   8.1. Exploración 58
   8.2. Preparación y planificación 59
   8.3. Modelado 64
   8.4. Comunicar resultados 67
9. Machine Learning o Aprendizaje Automático (II) 72
   9.1. Aprendizaje supervisado 74
      9.1.1. Algoritmos de predicción 80

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

9.1.1.1. Regresión Lineal....80

9.1.2. Algoritmos de clasificación....87

9.1.2.1. Modelo logístico....89

9.1.2.1. Árboles de decisión....90

9.1.3. Evaluación de modelos....101

9.1.3.1. Métodos de remuestreo....102

9.1.3.2. Métricas de evaluación de modelos....104

9.2. Aprendizaje no supervisado....111

9.2.1. Algoritmos de agrupación....113

9.2.1.1. Algoritmo k-Means....117

10.Caso de uso de Apache Spark (II). Spark MLlib....119

10.1. Librería spark.ml sobre DataFrames....122

10.1.1. Tipos de datos especiales para Spark ml y Spark mllib....123

10.1.2. Extracción, transformación y selección de características....124

10.1.3. Aprendizaje Supervisado: Regresión....127

10.1.3.1. Regresión Lineal....128

10.1.3.2. Modelo Lineal Generalizada....129

10.1.3.3. Regresión con Árboles de Decisión....131

10.1.3.4. Regresión con Random Forest....132

10.1.3.5. Regresión GBT (Gradient-boosted tree)....133

10.1.3.6. Regresión para el análisis de supervivencia....134

10.1.3.7. Regresión Isotónica....135

10.1.4. Aprendizaje Supervisado: Clasificación....136

10.1.4.1. Regresión logística....136

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10.1.4.1. Clasificación con Árboles de Decisión 138
10.1.4.2. Clasificación con Random Forest 139
10.1.4.3. Clasificación GBT (Gradient-boosted tree) 140
10.1.4.4. Perceptrón multicapa 141
10.1.4.5. Máquina de Vector de Soporte Lineal 142
10.1.4.6. Naive Bayes 143
10.1.5. Aprendizaje no supervisado: Clustering 143
10.1.5.1. K-means 144
10.1.5.2. Latent Dirichlet allocation (LDA) 145
10.1.5.3. Bisecting k-means 145
10.1.5.4. Gaussian Mixture Model (GMM) 146
10.1.6. Evaluación de modelos 147
10.1.6.1. Validación cruzada 147
10.1.6.2. División en entrenamiento y prueba 149
10.2. Librería spark.mllib sobre RDDs 150
11.Caso de uso de Apache Spark (IV). Spark Streaming 153
Anexo I –Trabajar con los resultados de fútbol 165

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

1. Introducción

En este módulo continuaremos adentrándonos en las diferentes herramientas Big Data para el análisis de datos. Partiremos de los conocimientos básicos de Spark y SQL adquiridos en módulos anteriores, para seguir ampliándolos y descubrir herramientas avanzadas que nos permiten sacarles el máximo partido.

Respecto a Spark, haremos un recorrido en profundidad, continuando la introducción realizada en módulos anteriores y descubriendo capacidades necesarias para poder implementar aplicaciones y modelos más complejos. Presentaremos también Spark SQL, para trabajar con fuentes de datos estructuradas, y que además nos da la opción de utilizar sintaxis SQL con Spark e interactuar con varios tipos de fuentes de datos de forma transparente.

Tras obtener los conocimientos necesarios, presentaremos el aprendizaje automático o Machine Learning. Primeramente, veremos en qué consiste y cuáles son sus principales objetivos y capacidades, para luego adentrarnos en los diferentes modelos de aprendizaje.

Veremos con mayor detalle los modelos más representativos, haciendo especial énfasis en aquellas características que comparten y también en las que los diferencian. A través de casos de uso se dará una visión al alumno de la utilidad de los diferentes modelos para entender qué modelo sería el adecuado para un caso de uso concreto que se plantee.

Por último, se hará un recorrido por las librerías Spark más representativas del contenido tratado, como MLib y GraphX.

Todo este contenido permitirá alcanzar los siguientes objetivos:

- Conocer la herramienta Hive y utilizarla para manejar información almacenada.
- Comprender la arquitectura de Apache Spark y su funcionalidad más relevante.
- Realizar una instalación de Apache Spark standalone y levantar un Notebook en entorno cloud para ejecutar código PySpark.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Ser capaz de emplear sintaxis SQL en Spark SQL para trabajar con datos estructurados y distintas fuentes.
- Comprender el concepto Machine Learning, sus objetivos y las situaciones donde se puede aplicar.
- Conocer los modelos de aprendizaje más comunes y ser capaz de establecer una comparativa de resultados entre varios.
- Utilizar librerías como MLib y GraphX para caracterizar una situación determinada con los conceptos desarrollados en el módulo.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

2. Apache Spark

La tecnología Spark nace en el año 2009 dentro de un grupo de investigación de la Universidad de Berkeley en California. Uno de los objetivos que se perseguían con Spark era mejorar la eficiencia de Map Reduce en aquellas tareas más repetitivas.

El proyecto pasaría a liberarse como Open Source y, más tarde, sería transferido a la fundación Apache para su evolución. El equipo original de diseño de Spark fundó la compañía Databricks, que ha sido y es una de las grandes impulsoras de esta tecnología, tanto en la incorporación y evolución de funcionalidad como en sus aportaciones a la comunidad de desarrollo.

En los últimos años, la comunidad de Spark se ha convertido en la comunidad Open Source más importante del ámbito del Big Data, evolucionando y englobando múltiples proyectos nuevos que complementan aún más sus capacidades.

Las ventajas más importantes que aporta Spark frente a otras tecnologías son:

- Velocidad: el procesamiento en Spark es mucho más rápido que en, por ejemplo, Hadoop MapReduce. Esta ventaja es más notable cuando realizamos procesamiento en memoria, siendo la herramienta de analíticas con mejor rendimiento en estas condiciones[1]. En aplicaciones que requieran procesado en disco, la ganancia también es visible frente a otras alternativas, haciendo que Spark sea la elección cuando queremos mejorar el rendimiento y/o reducir los tiempos de computación.

- Usabilidad: Spark presenta diferentes APIs para trabajar con datasets de gran tamaño, con más de 100 operadores para transformación de datos y un con un conjunto de APIs capaces de trabajar con datos semi-estructurados. Estas APIs están

[1] https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

diseñadas para ser lo más simples posibles y están portadas a los lenguajes de programación más habituales (Python, Scala, Java, SQL).

- Unificación: Spark se presenta de forma genérica permitiendo abordar con la misma sintaxis diferentes casos de uso y transformaciones de datasets. La distribución de Spark incluye varias librerías y herramientas adicionales, como las que permiten trabajar con Machine Learning o grafos que veremos más adelante en este módulo. Estas librerías están estandarizadas, permitiendo mejorar la productividad de los desarrolladores y pudiéndose combinar de forma sencilla para aquellas aplicaciones que requieran algoritmos más complejos.

Además, y como hemos venido mencionando, el gran crecimiento de la comunidad y los múltiples aportes, hacen que Spark se integre con varias herramientas que nos permiten abarcar una gran cantidad de situaciones. Ejemplo notable es la gran capacidad que tiene Spark de trabajar con fuentes de datos heterogéneas (ficheros, HDFS, base de datos…) con las mismas APIs y realizando el mismo tratamiento, e incluso permitir aplicaciones en Streaming (donde los datos se vuelven disponibles en tiempo real) como se verá en un ejemplo al final de este módulo.

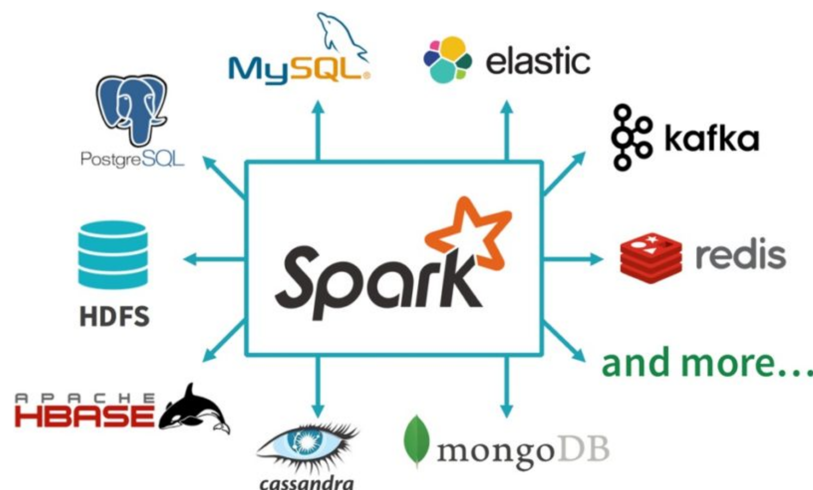


Figura 1: Spark como motor unificado[2]

[2] https://databricks.com/spark

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

2.1. Funcionalidades Principales

Es necesario conocer y comprender las funcionalidades que hacen que Spark presente las ventajas mencionadas anteriormente. Estas funcionalidades junto a un gran número de optimizaciones en los algoritmos de distribución y proceso de tareas son las que explican la mejora de rendimiento de Spark frente a otras alternativas.

RDDs (Resilient Distributed Dataset)

Los RDD son una abstracción de datos que nos permite trabajar de forma sencilla con diferentes datasets. Son el tipo de dato básico en Spark y, tal y como ahí se definen, son una colección de elementos que es tolerante a fallos y que es capaz de operar en paralelo.

Sus características más relevantes son:

- Se crean generalmente a partir de ficheros **distribuidos** (como un almacén HDFS) y residen en memoria. Spark determina cuándo ya no son utilizados para liberar la memoria que ocupan. Las transformaciones/acciones del API de Spark se aplican sobre este tipo de datos.

- Son objetos que están **particionados** en los distintos nodos del clúster (salvo en una instalación standalone), esto significa que cada nodo almacena una parte de los datos, o que están distribuidos en el clúster. Con esto se implementa el mismo principio de tolerancia a fallos que posee de Hadoop.

- Se caracterizan por ser **inmutables**, lo que implica que no se modifican (son de sólo lectura). Las transformaciones que se realizan sobre ellos darán lugar a nuevos RDDs.

- Utilizan **evaluación perezosa** (*lazy-evaluation*). Las transformaciones que realizamos sobre los RDDs no se ejecutan en ese momento, sino que son acumuladas hasta que no se puede retrasar la ejecución. Esto hace que se gane en eficiencia al realizar varias transformaciones simultáneamente y de forma optimizada, pero que sea más difícil de depurar el código cuando hay algún fallo (ya que lo veremos cuando se ejecuta y no en la línea donde indicamos la transformación).

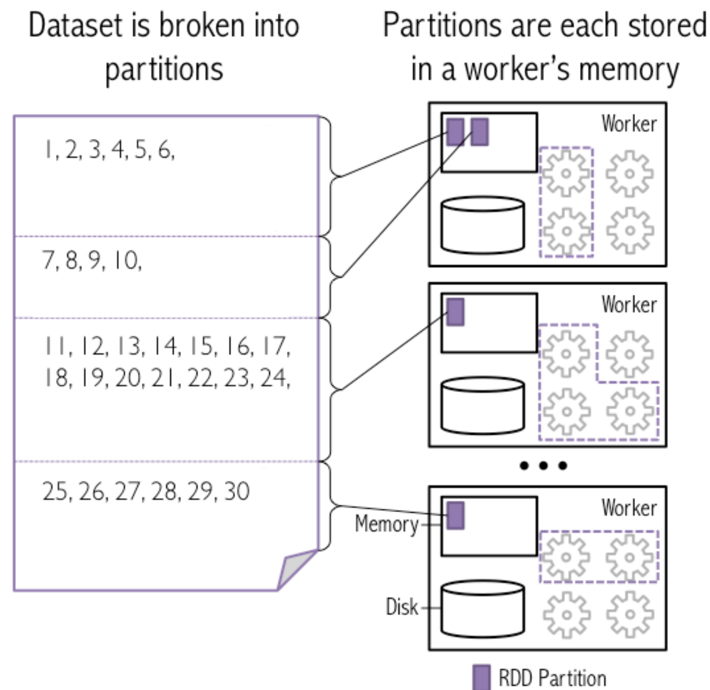


Figura 2: Particiones de un RDD y su distribución a los ejecutores[3]

DAG (Directed Acyclic Graph)

El DAG en Spark es un grafo dirigido que no contiene ciclos, lo que quiere decir que para cada uno de los nodos (*edge*) del grafo no vamos a encontrar una ruta que comience y termine en él. Los vértices (*vertex*) se conectan unos a otros, pero nunca consigo mismos.

Por cada tarea que tengamos programada en Spark sobre un RDD, se va a crear un DAG con distintas etapas que se ejecutarán en el clúster. En esta secuencia tendremos un número finito de nodos y vértices conectados entre sí, donde la dirección del grafo se produce dentro de una línea temporal (en la línea de ejecución, de tarea de comienzo a la de final).

Podemos ver al DAG como una generalización de Map&Reduce, donde sólo teníamos dos nodos definidos (las tareas Map y Reduce). Aquí tenemos una de las ventajas más

[3] https://databricks.com/spark/getting-started-with-apache-spark

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

importantes de Spark frente a MapReduce, no hay necesidad de grabar a disco el resultado de las etapas intermedias del grafo, sino que automáticamente se conectan a la entrada de la siguiente etapa. En MapReduce es necesario escribir los resultados entre las etapas Map y Reduce.

Dicho de otra forma, en Hadoop MapReduce no se conoce qué etapa Map Reduce viene a continuación en nuestro programa, por lo que los resultados siempre se escriben a HDFS y se vuelven a leer para comenzar la siguiente etapa. En la mayor parte de trabajos iterativos eso es una sobrecarga innecesaria, ya que los resultados intermedios no son necesarios.

¿Cómo resuelve Spark esto? Spark divide los RDD en un conjunto de etapas (*stages*) de computación necesarias. En esta determinación de etapas, Spark aplica ciertas optimizaciones para mejorar la ejecución (por ejemplo: agrupando transformaciones, tratando de minimizar el *shuffle*…). Cada una de estas etapas va a estar formada por una serie de tareas (*tasks*), que se basan en las particiones del RDD, y que realizan el mismo cálculo en paralelo.

Si consideramos el siguiente pseudo código de ejemplo donde se crean dos RDD y posteriormente se unen mediante un *join*:

```
# Se asume el context Spark instanciado en la variable sc

val rdd1 = sc.parallelize(…).partitionBy(…)

val rdd2 = sc.parallelize(…).partitionBy(…)

val rdd3 = rdd1.join(rdd2)

rdd3.collect()
```

Tendríamos el siguiente DAG donde las etapas 0 y 1 corresponden a la creación de *rdd1*. La etapa 2 a la creación de *rdd2*, y la etapa 3 donde se realiza el join de ambos. Nótese que en esta etapa Spark está aplicando una optimización, incluyendo la etapa de paralización del segundo RDD en la propia del *join*.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

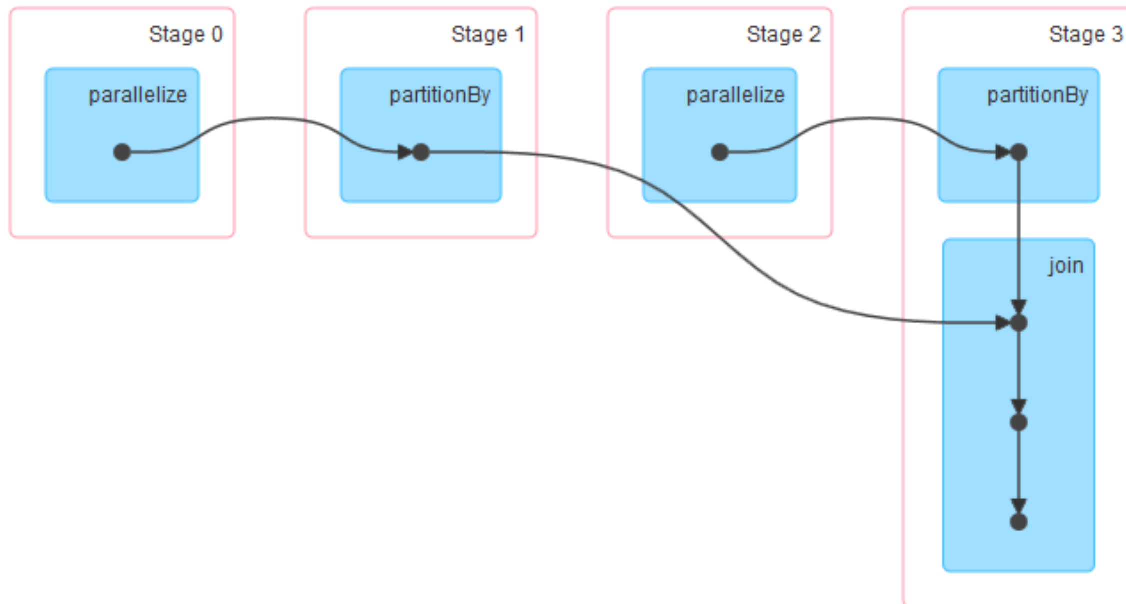


Figura 3: DAG de una instrucción de join entre dos RDD (herramienta Spark UI)

En la siguiente figura vemos otro ejemplo más conceptual de cómo funciona el DAG. En este caso, el programa a ejecutar (Driver Program) consiste en crear un RDD (leyendo un fichero), realizar un filtrado de sus filas según algún criterio y contar el número de filas resultantes.

El DAG Scheduler dividirá nuestro trabajo en *stages* y *tasks* según las particiones del RDD creado. El código de programa y las particiones se enviarán a cada uno de los ejecutores *(Worker Node)* que se encargarán de computar en paralelo.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

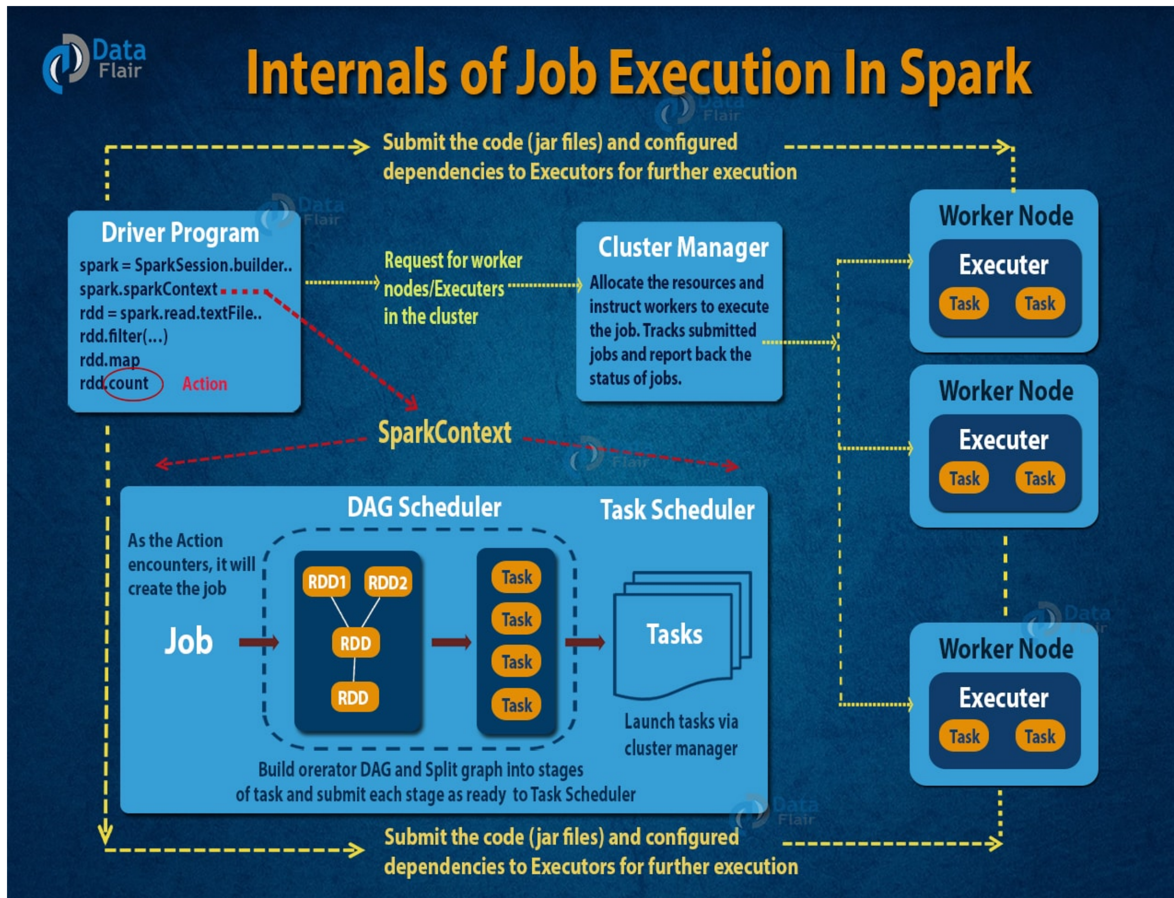


Figura 4: Ejecución de trabajos en Spark, utilidad del DAG[4]

Resumiendo, las ventajas que aporta el DAG en Spark son:

- Permite recuperar pérdidas de RDDs y realizar *shuffle* en caso necesario
- El número de etapas disponibles no se limita a las Map y Reduce como en Hadoop MapReduce. Esto hace que sea mucho más eficiente para ejecutar consultas SQL más complejas.
- La tolerancia a fallos está garantizada ya que, si un nodo queda fuera de servicio, otro retomará sus tareas sobre las particiones encomendadas, teniendo el control

[4] https://data-flair.training/blogs/dag-in-apache-spark/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

completo de la ejecución el DAG.

- El tener contexto de las etapas pasadas, presentes y futuras, hacen que se puedan realizar mejores optimizaciones y acumular las transformaciones sobre el mismo dataset.

2.2. Contextos Spark

Los contextos Spark permiten que la aplicación del *driver* acceda a un clúster a través de un gestor de recursos (o a los recursos de una máquina en modo standalone). Cada uno de los diferentes tipos de contextos en Spark nos va a dar acceso a ciertos componentes del framework, por lo que el uso de uno u otro contexto va a depender de las necesidades de nuestra aplicación como veremos a continuación.

SparkConf

Para crear un contexto es necesario definir una configuración de Spark. Esta SpakConf[5] almacena los parámetros de configuración deseados para nuestra aplicación. La mayoría de los parámetros definen propiedades de nuestra aplicación y algunos se utilizan para que Spark asigne los recursos necesarios del clúster. Por ejemplo, podemos fijar el número de ejecutores (*workers)*, la memoria reservada a cada uno de ellos, los *cores* del procesador asignados a cada uno… Los parámetros más frecuentes que podemos emplear son:

Parámetro	Significado
spark.master	El gestor[6] de recursos del clúster al que conectarse (local, yarn…)

[5] https://spark.apache.org/docs/latest/configuration.html

[6] https://spark.apache.org/docs/latest/submitting-applications.html#master-urls

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

spark.executor.instances	Número de ejecutores (*workers*) de nuestra aplicación
spark.executor.memory	Memoria asignada a cada uno de los ejecutores
spark.executor.cores	Número de cores de procesador asignados a cada ejecutor
spark.app.name	Nombre de nuestra aplicación (útil para depurarla en la consola SpakUI si tenemos varias en ejecución)

Un ejemplo para crear una configuración en modo standalone (en nuestra máquina local):

```
from pyspark import SparkConf

conf = new SparkConf()

        .set(“spark.master”, "local")

        .set(“spark.app.name”, "AppPruebaMBDD")
```

Algunas propiedades también se pueden fijar a través de funciones específicas[7]. De forma equivalente al ejemplo anterior, tendríamos:

[7] https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.SparkConf

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
from pyspark import SparkConf

conf = new SparkConf()

          .setMaster("local")

          .setAppName("AppPruebaMBDD")
```

SparkContext

Una vez que tenemos la configuración definida según los pasos anteriores, podemos instanciar nuestro SparkContext para comenzar a procesar nuestra aplicación. En este caso bastará con:

```
from pyspark import SparkContext

sc = SparkContext(conf)
```

Una vez que nuestra aplicación en el drive tiene un SparkContext asignado, ya puede solicitar los recursos necesarios al gestor (además mediante la configuración le hemos especificado nuestras necesidades). En un funcionamiento típico a través del gestor *Yarn*, se reservarán los contenedores necesarios en los nodos, de forma que cada ejecutor (*executor*) tenga su espacio de procesamiento. Estas relaciones se ilustran en la siguiente figura:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

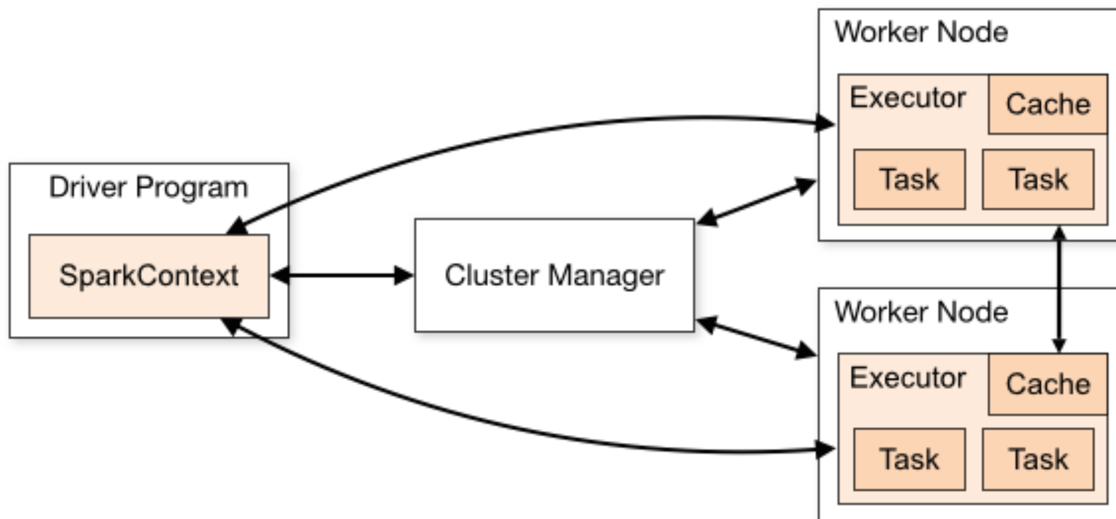


Figura 5: Interacciones del SparkContext en modo clúster[8]

El SparkContext crea un trabajo para nuestra aplicación, que será dividido en diferentes *stages*, las cuales a su vez se dividirán en tasks. Estas últimas se planificarán para ejecutarse en los distintos *executors*.

Las funciones más importantes del SparkContext, por tanto, son:

- Sirve de punto de entrada a la funcionalidad Spark.
- Permite fijar una configuración definida para nuestra aplicación.
- Posibilita la cancelación de Jobs y Stages.
- Crear RDDs y hacer broadcast a los nodos de ficheros no distribuidos (por ejemplo, cuando cargamos un fichero de texto local).

SQLContext / HiveContext

Son contextos que nos permiten acceder a la funcionalidad SQL de Spark y conectar

[8] https://people.apache.org/~pwendell/spark-nightly/spark-master-docs/latest/cluster-overview.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

nuestra aplicación con fuentes de datos externas.

SQLContext nos permite conectarnos a diferentes fuentes de datos de donde procesar la información (por ejemplo, bases de datos), mientras que HiveContext es un superconjunto del anterior que nos permite acceder a Hive.

En versiones de Spark 1.x, HiveContext además proporciona otras funciones avanzadas como las ventanas y la compatibilidad con Hive UDFs (funciones que podemos tener predefinidas en Hive). Esto hace que normalmente se trabaje con HiveContext.

Las diferencias entre ellos son más relevantes en las versiones 1.x de Spark. A partir de la versión 2.0 estas diferencias se van minimizando hasta el punto de crear un nuevo punto de entrada a la funcionalidad de Spark que veremos en el siguiente punto.

```
from pyspark import SparkContext

from pyspark.sql import HiveContext

sc = SparkContext(conf)

sqlContext = HiveContext(sc)

df_tabla = sqlContext.sql('select * from basededatos.tabla')
```

En el ejemplo anterior estaríamos creando directamente el dataframe *df_tabla* a partir de una consulta sobre la tabla *nombretabla* de una base de datos Hive llamada *basededatos*.

SparkSession

Surge tras la necesidad de homogeneizar los diferentes contextos de Spark, proporcionando un único punto de entrada a toda la funcionalidad Spark y facilitando el trabajo a los desarrolladores.

En este caso, no es necesario definir previamente una configuración de SparkConf, ya que

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

se puede realizar en el propio proceso de inicialización.

Los ejemplos anteriores para crear un SparkContext y un HiveContext se podrían resumir en:

```
from pyspark.sql import SparkSession

spark = SparkSession

        .builder()

        .appName("AppPruebaMBDD ")

        .config("spark.executor.instances", “2”)

        .enableHiveSupport()

        .getOrCreate()
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

3. Instalación y familiarización con la herramienta

Para aprender y trabajar con Spark disponemos de numerosas alternativas. La flexibilidad de Spark y el poder ejecutarse sus componentes bajo la máquina virtual de Java, hacen que podamos ejecutarlo hasta en nuestro propio PC en modo *standalone* desplegando tanto el *Master* como los *Slaves* en la misma máquina.

Recordemos que, como vimos en el Módulo de Introducción al Big Data, también tenemos a nuestra disposición varias distribuciones Hadoop que integran Spark entre sus componentes, por lo que podríamos optar por alguna de ellas e instalarla siguiendo sus propias instrucciones.

Por último, tenemos la alternativa de utilizar alguna de las soluciones cloud disponibles, evitando así la necesidad de realizar instalaciones y pudiendo ejecutar trabajos en un clúster de varios nodos sin necesidad de tener la infraestructura física.

Para los contenidos de este módulo será más que suficiente con el uso de una distribución cloud, si bien se incluyen los pasos típicos para la instalación de Spark en un PC con Linux para aquellos usuarios que quieran experimentar esta vía.

3.1. Notebook Cloud (Databricks)

Al igual que en el módulo de Introducción al Big Data, para los ejercicios y ejemplos del módulo utilizaremos una cuenta gratuita en Databricks[9] (Community Edition).

En esta cuenta no necesitamos preocuparnos de la instalación de ninguno de los componentes requeridos para ejecutar Spark, sino que accederemos a la funcionalidad de Spark a través de un Notebook.

Una vez registrados y creado en nuestra cuenta un clúster virtual, podemos proceder a

[9] https://databricks.com/try-databricks

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

crear el Notebook que ejecutaremos en él.

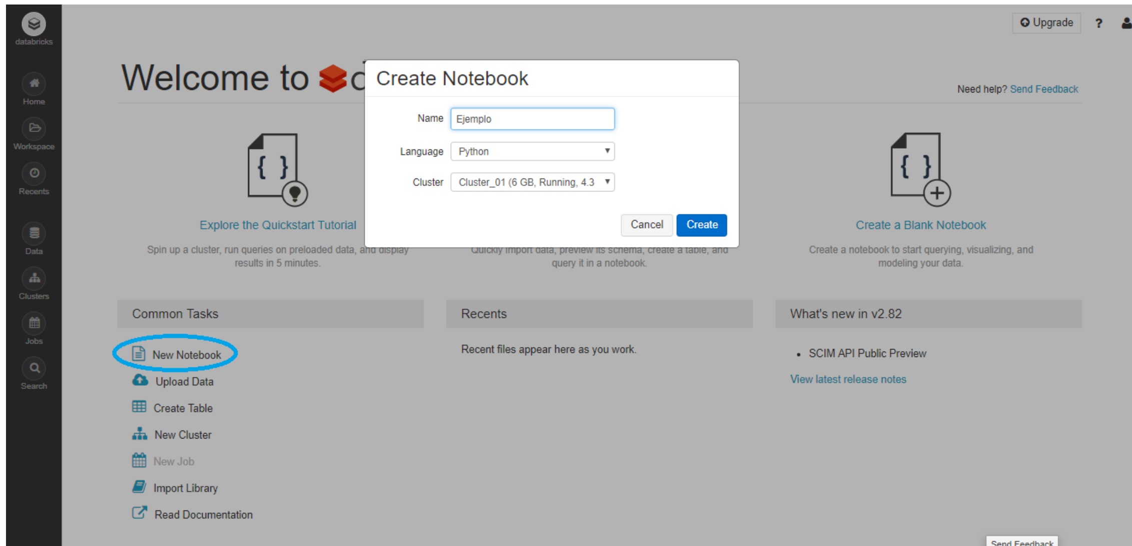


Figura 6: Creación del notebook de trabajo en Databricks

El Notebook nos permitirá introducir código en cada celda y ejecutarlo en nuestro clúster con sólo pulsar en "Run Cell". Por defecto, el Notebook ya tiene asignado un contexto Spark, en este caso un *SparkSession* en la variable *spark*, con lo que Databricks nos facilita enormemente el empezar a trabajar directamente sin necesidad de incluir configuración (se prepara por defecto según el clúster creado) ni las sentencias de inicialización.

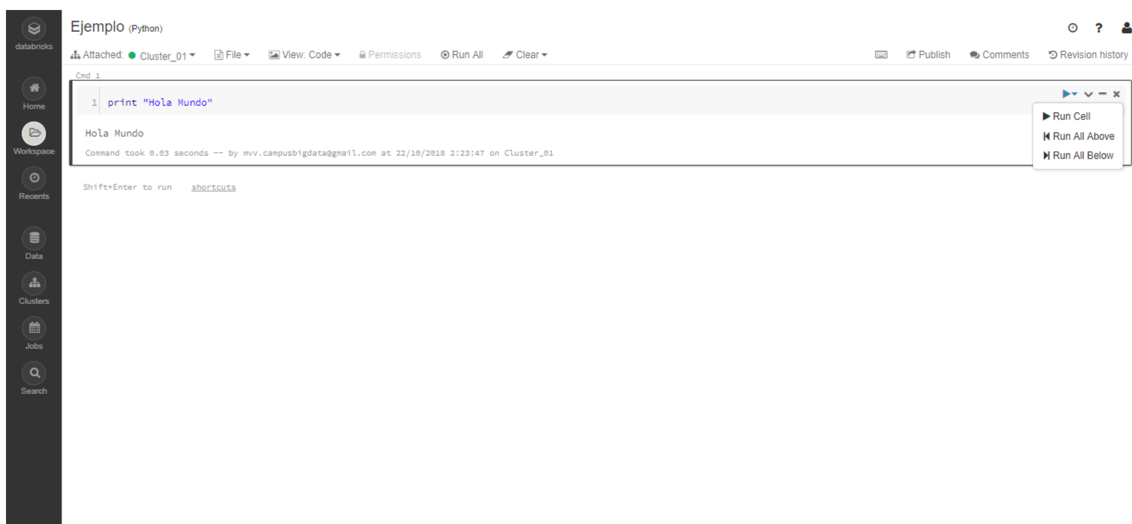


Figura 7: Trabajando en el Notebook

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

3.2. Standalone Setup (opcional)

Se incluye una referencia a la instalación de Spark en un PC Linux para aquellos que deseen practicar en su propio entorno y para los que disponen de alguna destreza, no se limiten únicamente a trabajar en un entorno controlado, es decir, os facilitamos el camino para explorar un poco más, no esperando con ello que todos alcaéis dichas destrezas, dado que todo lo que os solicita para ser evaluados puede ser desarrollado en Databricks.

. Así mismo, en los materiales adicionales se proporcionan enlaces a videos útiles para su instalación. En este caso será una instalación de un único nodo (ya que sólo disponemos del PC), por lo que nos servirá para nuestros primeros desarrollos, pero no para procesar grandes volúmenes de datos.

El proceso de instalación más frecuente es sobre un PC Línux por lo que, si el usuario tiene otro tipo de entorno, se recomienda la instalación de una máquina virtual con, por ejemplo, Ubuntu Linux para facilitar el trabajo. En la web existen múltiples sitios[10] donde ya se pueden descargar máquinas virtuales gratuitas con Ubuntu para el software gratuito Virtualbox[11].

Es importante resaltar que estas instalaciones requieren de un nivel medio de usuario Linux con conocimientos de instalación y administración, ya que dependen en gran medida del sistema instalado y su configuración.

Prerrequisitos

- Tener un sistema operativo Linux instalado y funcionando (puede ser virtualizado como se ha comentado anteriormente).
- Tener instalada y actualizada la versión de Java (por ejemplo, Java 8). Dependiendo de la versión de Spark elegida los requisitos pueden variar. Es

[10] Dos sitios habituales con máquinas virtuales Ubuntu gratuitas: https://www.osboxes.org/ubuntu/ y https://virtualboxes.org/images/ubuntu/
[11] https://www.virtualbox.org

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

recomendable ver las *release notes* de la versión (para Spark 2.4 se requiere[12] Java 8+).

- Tener Python instalado y actualizado según la versión de Spark elegida (para Spark 2.4 se requiere Python 2.7+/3.4+).
- Si se quiere utilizar el API de Scala, también deberá estar instalado y en la versión adecuada

Instalación Independiente

La instalación en modo local requiere de los siguientes pasos:

1. Descargar la versión deseada de Spark de la página oficial

   http://spark.apache.org/downloads.html

Download Apache Spark™

1. Choose a Spark release: 2.4.0 (Nov 02 2018)
2. Choose a package type: Pre-built for Apache Hadoop 2.7 and later
3. Download Spark: spark-2.4.0-bin-hadoop2.7.tgz

Figura 8: Elección de versión a descargar de Spark

2. Copiar el fichero descargado (por ejemplo: *spark-2.4.0-bin-hadoop2.7.tgz*) al directorio deseado (por ejemplo: */home/directorio_elegido*)

[12] https://spark.apache.org/docs/latest/index.html#downloading

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

3. Desempaquetar el fichero:

   $ tar xvf spark-2.4.0-bin-hadoop2.7.tgz

4. Configurar el entorno:

   a. Añadir el fichero ~/.bashrc las siguientes líneas:

      export SPARK_HOME=/home/directorio_elegido/spark-2.4.0-bin-hadoop2.7/

      export PATH=$PATH:$SPARK_HOME/bin

   b. Una vez añadidas y guardado el fichero, cargarlo en el entorno:

      $ source ~/.bashrc

5. Arrancar el Master Node:

   /home/directorio_elegido/spark-2.4.0-bin-hadoop2.7/sbin/start-master.sh

   Una vez arrancado podremos acceder a la consola web con el navegador web en el puerto 8080 de nuestro PC: http://localhost:8080

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

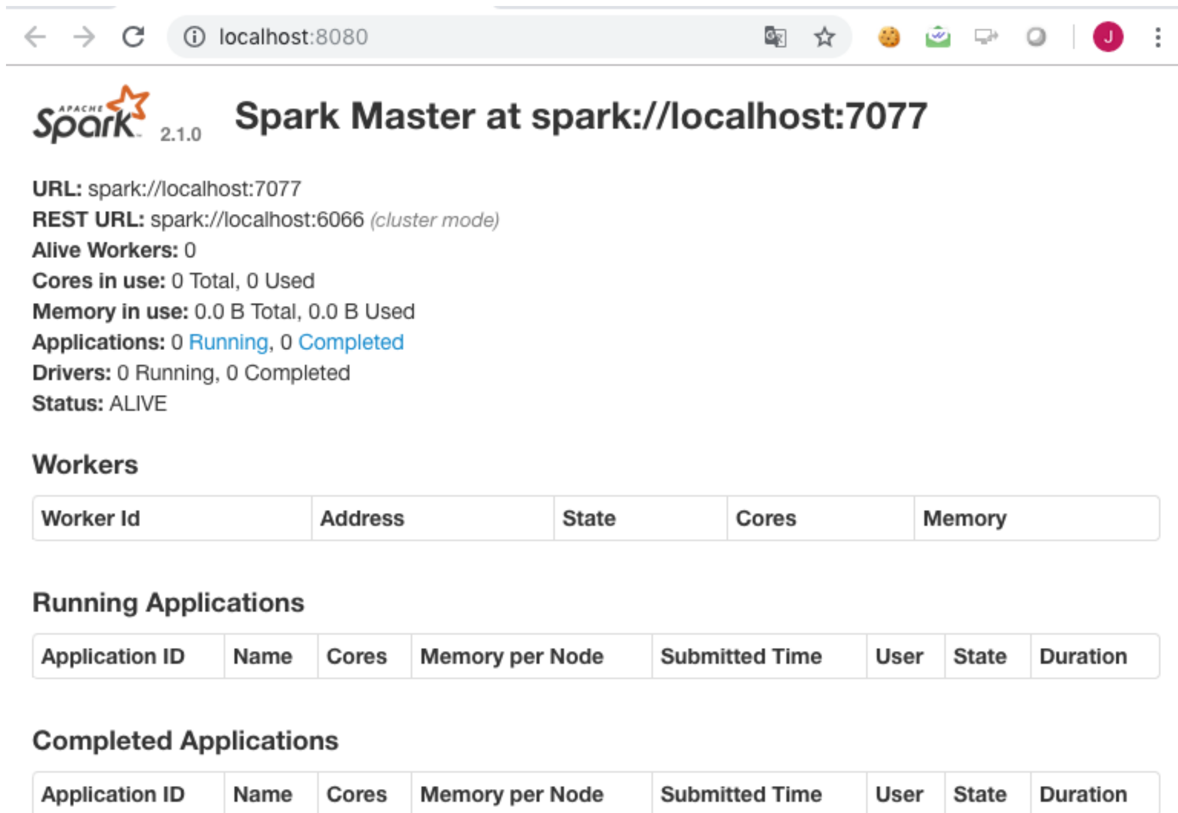


Figura 9: Consola web Spark

6. Arrancar los nodos esclavos (Slave Node):

   /home/directorio_elegido/spark-2.4.0-bin-hadoop2.7/sbin/start-slave.sh spark://*hostname*:7077

   Una vez arrancado algún nodo esclavo, podremos volver acceder a la consola web con el navegador web (http://localhost:8080) y comprobar cómo se ha añadido a la lista en la sección de "*Workers*":

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

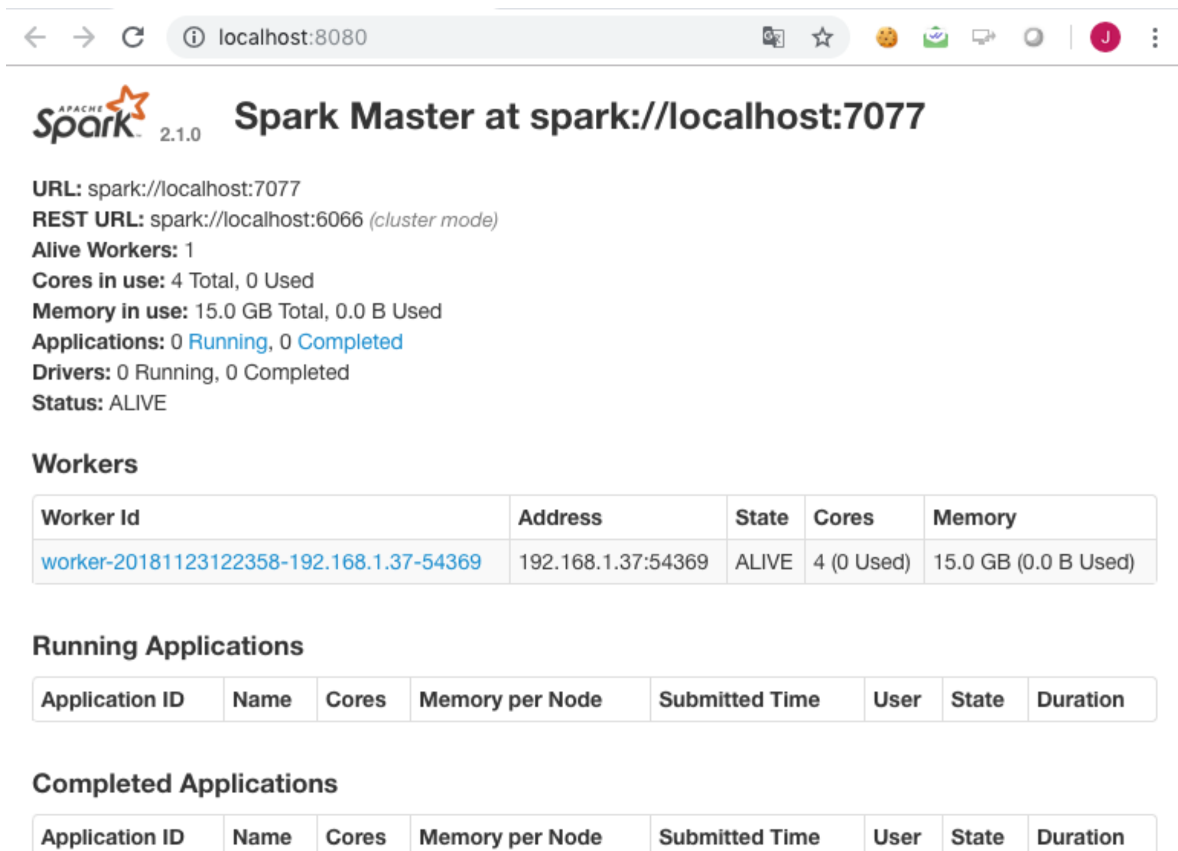


Figura 10: Consola web mostrando el nodo esclavo añadido

7. Arrancar un Spark Shell para empezar a ejecutar código Spark:

   /home/directorio_elegido/spark-2.4.0-bin-hadoop-2.7/bin/spark-shell spark://*hostname*:7077

Instalación PySpark

La instalación de Spark para Python (PySpark) se puede realizar a través del gestor de paquetes de Python PyPi[13]. En este caso bastará con invocar al gestor con el nombre del paquete, pudiendo añadir la versión elegida si queremos una concreta:

[13] https://pypi.org/project/pyspark

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Instalaría la última versión disponible:

```
pip install pyspark
```

Instalaría la version con Spark 2.2:

```
pip install pyspark==2.2.0
```

Si combinamos la instalación *standalone* con la de *pyspark* en la misma máquina, estaremos en condiciones de ejecutar código Python que incorpore rutinas de PySpark en nuestro clúster de prueba local.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

4. Trabajando con Apache Spark en Python

En el módulo de Introducción al Big Data ya vimos cómo realizar algún trabajo sencillo con PySpark (la librería de Spark para Python). El objetivo de este apartado es dar una visión en mayor profundidad de lo que es la creación de aplicaciones Spark bajo Python.

En esencia, cualquier aplicación Spark en Python consistirá en la siguiente secuencia de pasos:

- Inicialización: donde importamos las librerías necesarias, fijamos nuestra configuración Spark e inicializamos los contextos (salvo si trabajamos con Notebooks)
- Carga de datos: en esta fase nos conectaremos a nuestra fuente de datos (fichero, HDFS, bases de datos, sistemas remotos…) y cargaremos los datos en un RDD/Dataframe según necesidad
- Procesado: realizaremos las transformaciones y acciones necesarias sobre nuestros RDD/DataFrames hasta llegar al resultado deseado. En este punto el API Spark que hayamos elegido será determinante
- Final: mostraremos o grabaremos el resultado obtenido, además de finalizar la aplicación

A continuación, veremos con más detalle cada uno de los pasos necesarios.

4.1. Paquetes y Librerías

Una de las ventajas de utilizar Spark con Python, es la posibilidad de instalar fácilmente los paquetes y librerías necesarios para nuestra aplicación. De hecho, y como vimos anteriormente, podemos instalar el propio soporte Spark para Python con un solo comando:

```
pip install pyspark
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Una estructura típica para una aplicación PySpark con versión Spark 2.x y uso del *sparkContext*, sería:

```
# Librerías a utilizer

from pyspark import SparkConf

from pyspark import SparkContext

from pyspark.sql import HiveContext

import pyspark.sql.functions as F



# Configuración (ejemplo reducido)

confspark = SparkConf()

confspark.set("spark.master", "local")

confspark.set("spark.app.name", "AppPruebaMBDD")



# Incialización

sc = SparkContext(conf=confspark)

sqlcx = HiveContext(sc)



# A partir de aquí, la carga y el procesado de los datos…
```

Cuando trabajamos con Notebooks en entornos cloud (como el caso de Databricks), todos

estos pasos suelen ser innecesarios ya que se realizan de forma automática, si bien es cierto que necesitaremos importar las librerías necesarias más allá del SparkContext (o SparkSession).

4.2. Creación de RDDs

En general, hay tres formas con las que podemos crear un RDD: paralelizando una colección, mediante una lectura de un dataset externo y transformando un RDD existente. Veámoslas en detalle:

Paralelizando

El *SparkContext* dispone de un método *paralellize()* que, aplicado a cualquier colección que tengamos en memoria, nos permite distribuir la información y generar el RDD:

```
deportes = [“Football”, “Basketball”, “Tennis”, “Swimming”, “Running”]

miRdd = sc.parallelize(deportes)
```

Dataset Externo

También podemos tener la información en un fichero (dataset) externo, que podremos cargar mediante el método *textFile()* del que el *SparkContext* dispone:

```
fichero = “resultados.txt”

miRdd = sc.textFile(fichero)
```

Transformando un RDD

Dado que los RDD son inmutables, podemos aplicar cualquier transformación sobre un RDD que nos dará como resultado otro distinto (sin destruir el primero):

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
numerosRdd = sc.parallelize([1,2,3,4,5,6,7,8,9,10], 2)



paresRdd = numerosRdd.filter(lambda numberosRdd: numerosRdd % 2 == 0)

paresRdd.collect()

>> [2, 4, 6, 8, 10]
```

4.3. Operaciones sobre RDDs

La programación en Spark consiste fundamentalmente en la aplicación de operaciones sobre los distintos RDD, de formas que vayamos transformando unos en otros y que los vayamos combinando en base a ciertas reglas. Hay dos tipos de operaciones que son las transformaciones y las acciones.

Transformaciones son aquellas operaciones que al ejecutarlas sobre un RDD nos devuelven otro diferente. Un ejemplo típico es *filter()* que nos permite quedarnos con un subconjunto de los datos del primero, como hemos visto en el último ejemplo. Otra operación típica es *map()* que sirve para devolver un resultado por cada registro de entrada que procesemos, en base a una función que definamos.

Las transformaciones se realizan en los *Worker* (ejecutores) y, como vimos en los apartados anteriores, se utiliza “evaluación perezosa”. Es decir, Spark anota las distintas transformaciones que queramos realizar y construye el grafo de transformación (DAG).

Algunas de las transformaciones[14] más frecuentes son:

[14] http://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Transformación	Utilidad
rdd.filter()	Permite seleccionar un subconjunto de elementos de un RDD
rdd.map()	Modifica los elementos de un RDD mediante una función que definamos
rdd.join()	Unir dos RDD

Por su parte, las **acciones** son aquellas que no devuelven otro RDD, sino que retornan un tipo distinto o ejecutan un comportamiento (por ejemplo, grabar un RDD a un fichero). Tenemos que prestar especial atención a aquellas acciones que pueden devolver una gran cantidad de datos, ya que las acciones cuando devuelven valor, éste siempre es un objeto en el Driver Node con lo que consume la memoria disponible en él (aunque se procese parcialmente en cada uno de los nodos).

Acciones[15] habituales son:

Acción	Utilidad
rdd.collect()	Recoge los elementos de un RDD en el Driver (precaución con la memoria)
rdd.count()	Devuelve el número de elementos del RDD
rdd.saveAs___()	Permite guardar el contenido de un RDD en distintos formatos según la función

[15] http://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

	empleada

4.4. API Dataframes

Se introduce a partir de la versión Spark 1.3 y está construida sobre el API de RDDs. Su objetivo es superar alguna de las limitaciones que presentan los RDD. Formalmente, un DataFrame es una colección distribuida de datos organizados en columnas con nombre asignado. Los DataFrame son equivalentes como concepto a las tablas de una base de datos relacional.

Como vimos en el módulo de Introducción al Big Data, es el API más utilizado en la mayoría de las aplicaciones y el que nos va a permitir trabajar con más agilidad en nuestro proyecto. Presenta un mayor nivel de abstracción frente al de RDDs, lo que hace que muchas operaciones las podamos implementar de una forma más natural.

Algunas de las características más relevantes del API Spark DataFrames:

- Procesado de datos: es capaz de procesar numerosos formatos de datos (Csv, Avro…), tanto estructurados como no estructurados, y de trabajar con varios sistemas de almacenamiento (HDFS, MySQL…)
- Optimización: frente a los RDD el API DataFrames presenta optimizaciones a través del motor Catalyst16
- Compatibilidad con Hive: a través de Spark SQL permite ejecutar queries contra un almacén Hive
- Tungsten17: proporciona un backend de ejecución física que se ocupa de manejar

[16] https://databricks.com/glossary/catalyst-optimizer

[17] https://databricks.com/glossary/tungsten

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

la memoria y generar el código para la evaluación de expresiones

- API disponible en varios lenguajes: Java, Scala, Python y R
- Presenta limitaciones al tratar de manipular datos cuya estructura no es totalmente conocida (al no presentar tipado seguro en tiempo de compilación).

En el API de DataFrames también tenemos dos tipos de operaciones como con RDDs: transformaciones y acciones. Varias de ellas se vieron en los casos de uso del módulo de Introducción al Big Data, por lo que a modo resumen se incluye el listado de las más habituales:

Transformación	Utilidad
df.select()	Devuelve un DF con las columnas seleccionadas
df.filter() ó df.where()	Devuelve un DF que contiene las filas seleccionadas (acorde a la función/criterio que se le pase como argumento)
df.drop()	Devuelve un DF donde se ha eliminado la columna del original especificada
df.distinct()	Devuelve un DF donde se han eliminado filas duplicadas
df.orderby() / df.sort()	Devuelve un DF con las filas ordenadas (recibe la columna por la que se ordena y la dirección de la ordenación)

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

df.show() / df.disp() / df.take()	Se utilizan para mostrar el contenido de un DF (según el entorno)

Acción	Utilidad
df.groupBy()	Agrupa el DF por las columnas especificadas (crea un DF GroupedData[18])
df.agg()	Calcula agregados sobre los que calcular variables como las siguientes:
df.count()	Devuelve el número de filas (puede operar sobre GroupedData)
df.avg() o df.mean() / df.max() / df.min()	Devuelve la media / el máximo / el mínimo de las columnas (opera sobre GroupedData)
df.join()	Unión de DFs, con varios tipos

4.5. API Datasets

[18] http://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=pandas#pyspark.sql.GroupedData

Se introduce en la versión Spark 1.6 y viene a incorporar tipado seguro en tiempo de compilación en aquellos lenguajes que lo permiten (Python no se incluye al ser de tipo dinámico), lo cual reduce la probabilidad de errores en tiempo de ejecución (no detectados a la hora de compilar nuestra aplicación).

Para aclarar este punto, supongamos que tenemos una estructura de datos con dos columnas: columnaA y columnaB, y tenemos el siguiente código para DataFrames y DataSets:

```
# df es el DataFrame con columnaA y columnaB

df.filter(df. ningunaColumna > 0)

# ds es el DataSet con columnaA y columnaB

ds.filter($"ningunaColumna" > 0)
```

En el primer caso, con API DataFrames, no tendremos el error hasta el tiempo de ejecución (cuando se vaya a procesar esta línea) en donde se vea que en nuestro DataFrame no hay ninguna columna llamada "*ningunaColumna*". En el segundo caso, el error se generaría en tiempo de compilación, ahorrándonos la necesidad de ejecutar la aplicación para darnos cuenta del error (directamente el compilador nos indicaría que esa línea es incorrecta ya que el DataSet *ds* no tiene ninguna columna con ese nombre).

A partir de Spark 2.0 el API de DataFrame y DataSets se unifica (variando algunas características según el lenguaje elegido). Algunas reglas básicas para elegir entre las distintas APIs de Spark serían:

- Cuando se desea un API con mayor nivel de abstracción con librerías de dominio específico, se recomienda utilizar DataFrame o DataSet.
- Cuando se van a utilizar expresiones de alto nivel, filtros, agregaciones, medias, consultas SQL, accesos por columna, etc., se recomienda utilizar DataFrame o

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

DataSet.

- Si lo que se busca es una alta protección de tipado seguro en tiempo de compilación, además de beneficiarse de las últimas optimizaciones de código, la opción es DataSet.
- Si se busca la unificación y simplificación para trabajar con varias librerías Spark, la opción de mayor compatibilidad es DataFrame o DataSet.
- Si vamos a programar en lenguaje R, la opción es utilizar DataFrame.
- Si somos usuarios de Python (es el caso de los desarrollos de este módulo), debemos utilizar DataFrames, pudiendo emplear RDDs si necesitamos mayor control y menor abstracción.

En el siguiente diagrama se muestra de forma simplificada el universo de APIs Spark a partir de la versión 2.0 con las características más reseñables de cada API.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

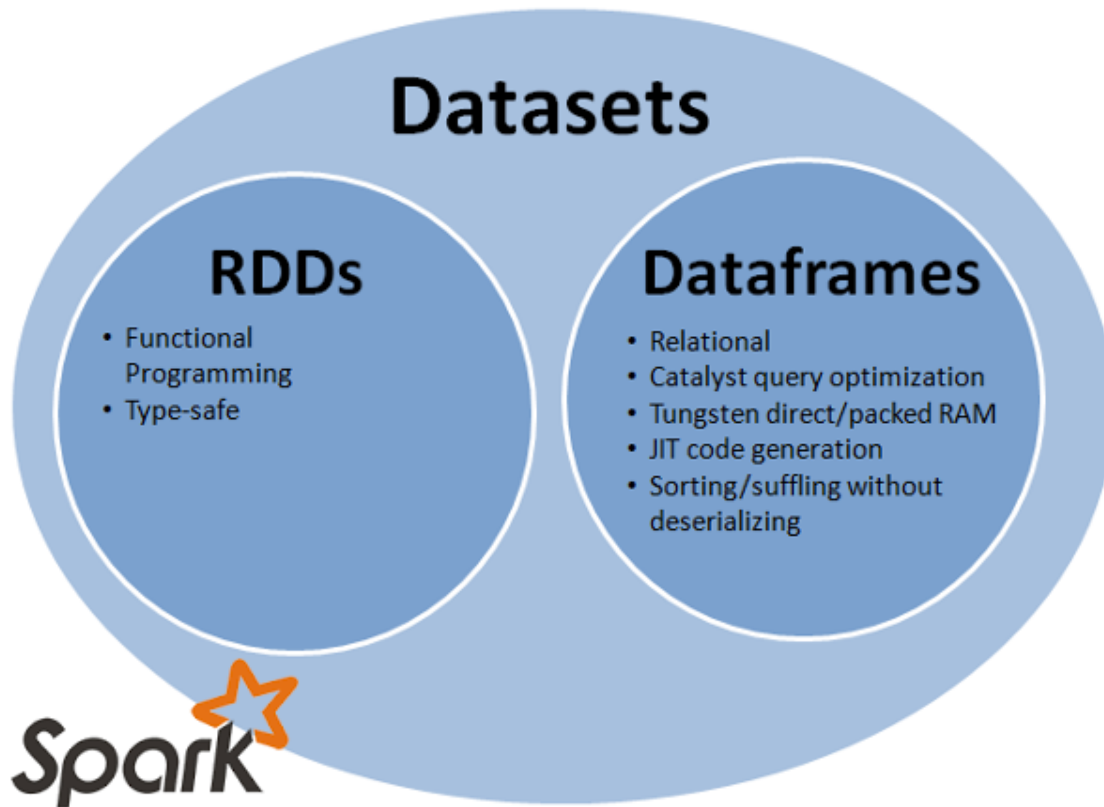


Figura 11: Comparativa APIs[19] Spark

[19] https://spark.apache.org/docs/latest/api.html

BIG DATA International Campus — opta — excellence InNova Consultoría y Formación S.L. — UCAM Universidad Católica de Murcia

5. Caso de uso Spark: análisis datos competición

Una vez comprendidos los apartados anteriores, vamos a ver un ejemplo en código en el que trabajaremos con un DataSet con el API DataFrames de PySpark.

El flujo más frecuente de una aplicación Spark suele ser el siguiente:

- Adquisición y limpieza de datos: es la primera fase donde incluiremos el código que nos permita acceder a los datasets necesarios (que pueden en estar en una o varias fuentes, en el mismo o en distintos formatos, local o remotamente)
    - Limpieza de los datos: en múltiples ocasiones nos encontramos con que los DataSet están incompletos o presentan errores. Un ejemplo es que presenten nulos, o valores NaN, o que tengan datos incorrectos. Es esencial analizar nuestro dataset y solucionar ese tipo de incidencias (por ejemplo, descartando aquellas filas con valores incorrectos) antes de comenzar el procesado que queramos realizar.
    - Formato: según los requisitos de nuestra aplicación, nos puede interesar organizar los datos de una u otra manera. Podríamos, por ejemplo, decidir si cuando tenemos dos dataset diferentes de inicio, queremos trabajar con dos DataFrame separados o es mejor combinarlos en uno con la información necesaria para el procesamiento.
- Análisis exploratorio: una vez que tengamos los datos correctamente estructurados y limpios, viene la fase de análisis. En ella analizaremos los datos que tenemos para que, a la hora de modelar nuestro problema, podamos caracterizarlo mejor. Esta fase nos permite conocer cómo se estructura la información en los distintos campos (columnas) y será clave a la hora de afrontar su procesado.
- Modelado: por último, una vez que tengamos nuestros datos bien organizados y limpios, y hayamos determinado mediante el análisis exploratorio aquellas variables más interesantes y sus relaciones, podemos pasar al modelado de nuestro problema. En este paso podríamos, por ejemplo, intentar predecir el

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

comportamiento de un deportista en una competición a partir de sus variables de rendimiento. Este último apartado lo veremos en detalle en el capítulo de Machine Learning.

En esta ocasión, vamos a continuar trabajando sobre el DataSet de resultados de La Liga de fútbol que ya empleamos en el módulo de Introducción al Big Data, por estar más familiarizados con sus columnas. La primera tarea será cargar los datos que, en este caso al ser la fuente única y en formato csv, es sencillo. Se opta también por un contexto *SparkSession* por su mayor abstracción respecto a los otros contextos, lo cual nos simplificará el trabajo.

Carga de Datos

```
from pyspark.sql import SparkSession

# Incializamos SparkSession
spark = SparkSession.builder.enableHiveSupport().getOrCreate()

# Carga del fichero der la temporada

df = spark.read.option("header","true").option("inferSchema", "true").csv('/FileStore/tables/SP1.csv')
df.show(5)

```

```
▸ (3) Spark Jobs
▸ df: pyspark.sql.dataframe.DataFrame = [Div: string, Date: string ... 62 more fields]
```

```
|Div|    Date|HomeTeam|  AwayTeam|FTHG|FTAG|FTR|HTHG|HTAG|HTR| HS| AS|HST|AST| HF| AF| HC| AC| HY| AY| HR| AR|B365H|B365D|B365A| BWH| BWD| BWA| IWH
|IWD| IWA| LBH| LBD| LBA| PSH| PSD| PSA| WHH| WHD| WHA| VCH|VCD| VCA|Bb1X2|BbMxH|BbAvH|BbMxD|BbAvD|BbMxA|BbAvA|BbOU|BbMx>2.5|BbAv>2.5|BbMx<2.5|BbAv
<2.5|BbAH|BbAHh|BbMxAHH|BbAvAHH|BbMxAHA|BbAvAHA|PSCH|PSCD|PSCA|

|SP1|18/08/17| Leganes|    Alaves|   1|   0|  H|   1|   0|  H| 16|  6|  9|  3| 14| 18|  4|  2|  0|  1|  0|  0| 2.05|  3.2|  4.1|2.05| 3.1| 4.1| 2.1
|3.4| 3.5|2.05| 3.0| 4.2|2.03|3.25|4.52|2.05| 3.1| 4.0|2.05|3.2| 4.4|   35| 2.12| 2.03|  3.4| 3.15| 4.52| 4.17|  31|    2.84|    2.68|    1.53|
1.46|  18| -0.5|   2.07|   2.03|    1.9|    1.86|1.98|3.35|4.63|
|SP1|18/08/17|Valencia|Las Palmas|   1|   0|  H|   1|   0|  H| 22|  5|  6|  4| 25| 13|  5|  2|  3|  3|  0|  1| 1.75|  3.8|  4.5|1.75| 3.9| 4.6|1.75
|3.6| 4.8|1.75| 3.8|4.33|1.78|4.01|4.83| 1.8|3.75| 4.2| 1.8|4.0| 4.6|   35| 1.83| 1.77| 4.04| 3.86| 4.83| 4.46|  33|    1.69|    1.64|     2.4|
2.27|  16|-0.75|   2.05|   1.97|   1.96|    1.91|1.78|4.24|4.43|
|SP1|19/08/17|   Celta|  Sociedad|   2|   3|  A|   1|   1|  D| 16| 13|  5|  6| 12| 11|  5|  4|  3|  1|  0|  0| 2.38| 3.25|  3.2| 2.4| 3.3| 3.0| 2.5
|3.3|2.85|2.35|3.25| 3.0|2.44| 3.4|3.16| 2.4| 3.4| 2.9| 2.4|3.4|3.13|   35|  2.5| 2.39|  3.5| 3.32|  3.2| 3.01|  34|    2.03|    1.98|     1.9|
1.84|  18|-0.25|   2.08|   2.05|   1.87|    1.83|2.12|3.53|3.74|
```

Ejemplo de Limpieza

Nuestro DataSet no presenta a priori problemas de datos erróneos (en la propia web que lo proporciona se indica que estos datos han sido previamente filtrados y corregidos donde era necesario).

Veamos en el siguiente ejemplo cómo podemos generar un DataSet con algunos

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

problemas, en concreto un NaN (al intentar crear como número decimal -*float*- algo que no lo es) y un nulo.

Posteriormente realizaremos un conteo de aquellos *NaN* y *null* encontrados, gracias a las funciones *isnan()* e *isnull()*.

```
1  import pyspark.sql.functions as F
2  
3  # Ejemplo de detección de NaN y nulos
4  # Creamos un DF con un NaN en la segunda columna y un nulo en la primera
5  df_nan_null = spark.createDataFrame([(1, float('NaN')), (None, 1.0)], ("a", "b"))
6  df_nan_null.show()
7  
8  #Contamos el número de NaN
9  df_nan_null.select([F.count(F.when(F.isnan(c), c)).alias(c) for c in df_nan_null.columns]).show()
10 
11 #Contamos el número de nulos
12 df_nan_null.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in df_nan_null.columns]).show()
```

```
▸ (5) Spark Jobs
▸ df_nan_null: pyspark.sql.dataframe.DataFrame = [a: long, b: double]
+----+---+
|   a|  b|
+----+---+
|   1|NaN|
|null|1.0|
+----+---+

+---+---+
|  a|  b|
+---+---+
|  0|  1|
+---+---+

+---+---+
|  a|  b|
+---+---+
|  1|  0|
+---+---+
```

En el primer caso hemos detectado el *NaN* en la columna 'b' y, en el segundo caso, el *null* en la columna 'a'.

Visto este ejemplo, ahora apliquemos esto mismo a nuestro dataset de partida una vez elegidas las columnas con las que queremos trabajar:

```
columnas = ["HomeTeam",
            "AwayTeam",
            "FTHG",
            "FTAG",
            "FTR",
            "HTHG",
            "HTAG",
            "HTR",
            "HS",
            "AS",
            "HST",
            "AST",
            "HC",
            "AC",
            "HF",
            "AF",
            "HY",
            "AY",
            "HR",
            "AR"]

# Nos quedamos con las columnas de la lista 'columnas'
df = df.select(columnas)

#Contamos el número de NaN
df.select([F.count(F.when(F.isnan(c), c)).alias(c) for c in df.columns]).show()

#Contamos el número de nulos
df.select([F.count(F.when(F.isnull(c), c)).alias(c) for c in df.columns]).show()
```

▸ (2) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [HomeTeam: string, AwayTeam: string ... 18 more fields]

```
+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+
|HomeTeam|AwayTeam|FTHG|FTAG|FTR|HTHG|HTAG|HTR| HS| AS|HST|AST| HC| AC| HF| AF| HY| AY| HR| AR|
+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+
|       0|       0|   0|   0|  0|   0|   0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|
+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+

+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+
|HomeTeam|AwayTeam|FTHG|FTAG|FTR|HTHG|HTAG|HTR| HS| AS|HST|AST| HC| AC| HF| AF| HY| AY| HR| AR|
+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+
|       0|       0|   0|   0|  0|   0|   0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|  0|
+--------+--------+----+----+---+----+----+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Vemos, como no hemos encontrado ningún valor *NaN/null* a diferencia del ejemplo anterior.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Análisis Exploratorio

Una vez que tenemos nuestro DataFrame cargado y limpio, vamos a analizar los datos que contiene.

En este caso, hemos filtrado las columnas relacionadas con los goles (FTHG y FTAG), los tiros (HS y AS), los saques de esquina (HC y AC), las faltas cometidas (HF y AF) y las tarjetas amarillas (HY y AY) y rojas (HR y AR).

Una función muy útil es *describe()*[20] que nos da algunos parámetros estadísticos de las columnas que deseemos.

```
1  # Utilizamos describe sobre los goles para ver algunos parámetros estadísticos
2  df.describe("FTHG", "FTAG","HR","AR").show()
```

▸ (1) Spark Jobs

```
+-------+------------------+------------------+-------------------+-------------------+
|summary|              FTHG|              FTAG|                 HR|                 AR|
+-------+------------------+------------------+-------------------+-------------------+
|  count|               380|               380|                380|                380|
|   mean|1.5473684210526315|1.1473684210526316|0.11052631578947368|0.07894736842105263|
| stddev| 1.378450210432462|1.1867200184400557|0.32225252203823884| 0.2796132786780865|
|    min|                 0|                 0|                  0|                  0|
|    max|                 7|                 6|                  2|                  2|
+-------+------------------+------------------+-------------------+-------------------+
```

En el ejemplo anterior vemos como el contenido son 380 filas (el número de partidos jugados en la Liga) para todas las columnas. La mayor cantidad de goles en un partido la ha conseguido el equipo local con 7 tantos, mientras que el mínimo número de goles es, lógicamente, 0.

En número de tarjetas rojas también hay un empate en el máximo de tarjetas en local y visitante, siendo este número 2. También podemos comprobar como la media de goles por partido es mayor en el equipo local que en el visitante y que la media de tarjetas rojas por

[20] http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

partido es mucho menor en los visitantes que en los locales.

Este mismo análisis podemos repetirlo para el caso concreto de un equipo:

```
1 equipo = "Valencia"
2 df_equipo = df.filter(df.HomeTeam.like(equipo) | df.AwayTeam.like(equipo))
3 df_equipo.describe("FTHG", "FTAG","HR","AR").show()
```

```
▸ (1) Spark Jobs
▸ df_equipo: pyspark.sql.dataframe.DataFrame = [Div: string, Date: string ... 62 more fields]
+-------+------------------+------------------+-------------------+-------------------+
|summary|              FTHG|              FTAG|                 HR|                 AR|
+-------+------------------+------------------+-------------------+-------------------+
|  count|                38|                38|                 38|                 38|
|   mean|1.5263157894736843|1.1842105263157894|0.13157894736842105|0.13157894736842105|
| stddev|1.1794832149456596|1.2270655648443332| 0.3425699874501045| 0.4140147948511246|
|    min|                 0|                 0|                  0|                  0|
|    max|                 5|                 6|                  1|                  2|
+-------+------------------+------------------+-------------------+-------------------+
```

Vemos como el equipo ha jugado 38 partidos (19 jornadas), donde ha metido un máximo de 6 goles en un partido (jugando además como visitante) y que tiene más puntería en los partidos jugados en su campo.

Otra función útil es *freqItems()*[21] que nos permite justamente conocer los valores más frecuentes que toma una columna:

[21] http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
1  df_freq = df_equipo.stat.freqItems(["FTHG", "FTAG"], 0.1).show()
2  df_freq = df_equipo.stat.freqItems(["FTHG", "FTAG"], 0.9).show()
```

```
▸ (2) Spark Jobs
+------------------+------------------+
|    FTHG_freqItems|    FTAG_freqItems|
+------------------+------------------+
|[2, 5, 4, 1, 3, 0]|[2, 4, 1, 3, 0, 6]|
+------------------+------------------+

+--------------+--------------+
|FTHG_freqItems|FTAG_freqItems|
+--------------+--------------+
|            []|           [1]|
+--------------+--------------+
```

En el primer ejemplo buscamos los goles del equipo como local y visitante que aparecen al menos un 10% de los partidos (y por tanto son menos probables). En el segundo ejemplo elevamos esa frecuencia al 90% para obtener el número de goles más probable que marcará el equipo elegido como local y como visitante. Observamos que como local no encontramos ningún resultado que supere esa frecuencia, mientras que, como visitante, el 90% o más de las ocasiones mete un gol.

Por último, hay dos funciones que nos van a resultar también muy útiles antes de pasar al modelado. Estas son la covarianza y la correlación que, en esencia, miden la relación entre dos variables (columnas).

Una covarianza elevada entre dos columnas nos indica que a medida que los valores de una se incrementan, los de la otra columna también lo hacen. La correlación es una medida de covarianza normalizada que nos permite entender mejor la relación entre las variables. Veámoslo con un ejemplo, buscando la relación entre algunas variables del equipo elegido:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
1  covar = df_equipo.stat.cov('FTHG', 'HS')
2  print "Covarianza goles/tiros equipo local: {}".format(covar)
3  covar = df_equipo.stat.cov('FTHG', 'HC')
4  print "Covarianza goles/corners equipo local: {}".format(covar)
5  covar = df_equipo.stat.cov('FTHG', 'FTHG')
6  print "Covarianza goles/goles equipo local: {}".format(covar)
7  corr = df_equipo.stat.corr('FTHG', 'HS')
8  print "Correlación goles/tiros equipo local: {}".format(corr)
9  corr = df_equipo.stat.corr('FTHG', 'HC')
10 print "Correlación goles/corners equipo local: {}".format(corr)
11 corr = df_equipo.stat.corr('FTHG', 'FTHG')
12 print "Correlación goles/goles equipo local: {}".format(corr)
```

```
▸ (6) Spark Jobs
Covarianza goles/tiros equipo local: 1.04409672831
Covarianza goles/corners equipo local: -0.099573257468
Covarianza goles/goles equipo local: 1.39118065434
Correlación goles/tiros equipo local: 0.192202346311
Correlación goles/corners equipo local: -0.0309260987643
Correlación goles/goles equipo local: 1.0
```

Si nos fijamos en la covarianza, es más difícil interpretar el grado de relación entre las dos variables elegidas. Cuando comparamos una columna consigo misma (mayor relación posible) obtenemos un valor que debemos interpretar. Si usamos la correlación, este valor es siempre "1" con lo que es fácil de identificar la mayor relación entre dos variables.

En el ejemplo anterior, y fijándonos en la correlación, vemos cómo para este equipo están mucho más relacionados los goles y los disparos que los goles y los saques de esquina.

Esta parte del análisis es clave para entender cómo se relacionan las diferentes variables y cuáles debemos elegir a la hora de modelar un determinado problema.

Preparación del Modelado

Una vez que hemos analizado nuestro dataset y realizado las distintas tareas de limpieza y acomodación de los datos, va a ser útil el realizar algunas transformaciones adicionales para que nuestro modelado presente mejores resultados. A continuación, presentaremos las transformaciones más frecuentes y relevantes.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Comenzaremos transformando una variable numérica en categórica. Esta transformación es útil para reducir el número de valores que toma una variable según lo que nos interese. En el ejemplo vamos a categorizar las faltas cometidas en tres categorías:

```
1  # Paso de Variable numérica HF (Faltas del equipo local) a categórica, con tres categorías
2  df_variables = df.withColumn("CategoriaFaltas", F.when(F.col("HF").between(10,20), "MEDIA").when(F.col("HF")<10, "POCAS").otherwise("MUCHAS"))
3  df_variables.select("HomeTeam", "HF", "CategoriaFaltas").show(5)
4
5  # Creación de una nueva característica TC: total de tarjetas en el partido
6  df_variables = df_variables.withColumn("TotalTarjetas", df.HY + df.HR + df.AY + df.AR)
7  df_variables.select("HomeTeam", "AwayTeam", "CategoriaFaltas", "TotalTarjetas").show(5)
```

```
▸ (2) Spark Jobs
▸ df_variables: pyspark.sql.dataframe.DataFrame = [HomeTeam: string, AwayTeam: string ... 20 more fields]
+--------+---+---------------+
|HomeTeam| HF|CategoriaFaltas|
+--------+---+---------------+
| Leganes| 14|          MEDIA|
|Valencia| 25|         MUCHAS|
|   Celta| 12|          MEDIA|
|  Girona| 15|          MEDIA|
| Sevilla| 14|          MEDIA|
+--------+---+---------------+
only showing top 5 rows

+--------+----------+---------------+-------------+
|HomeTeam|  AwayTeam|CategoriaFaltas|TotalTarjetas|
+--------+----------+---------------+-------------+
| Leganes|    Alaves|          MEDIA|            1|
|Valencia|Las Palmas|         MUCHAS|            7|
|   Celta|  Sociedad|          MEDIA|            4|
|  Girona|Ath Madrid|          MEDIA|            7|
| Sevilla|   Espanol|          MEDIA|            7|
+--------+----------+---------------+-------------+
only showing top 5 rows
```

También hemos generado una nueva **característica**, esto es, una nueva columna a partir de otras existentes. En el ejemplo, hemos calculado el total de tarjetas del partido mediante la suma de las tarjetas amarillas y rojas de cada equipo.

Otra transformación de utilidad es la **normalización**, mediante la cual transformamos el intervalo de valores de nuestra variable (por ejemplo, para hacer que todos los valores estén entre 0 y 1). Hay varios tipos de normalización[22], y la técnica empleada dependerá del algoritmo de Machine Learning que vayamos a aplicar.

En el ejemplo siguiente normalizamos el total de tarjetas creado anteriormente utilizando su media (el valor medio de los valores posibles) y su desviación típica (la dispersión de valores

---

[22]Normalization Techniques: https://pkghosh.wordpress.com/2017/12/05/data-normalization-with-spark/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

respecto de la media). Para dar un poco más de complejidad al ejemplo, lo hemos hecho a través de una **UDF** (User Defined Function) donde definimos una función que se aplicará a cada fila de nuestro DataFrame.

```
# Normalización del número de tarjetas

from pyspark.sql.functions import udf
from pyspark.sql.types import FloatType

# Definimos función que normalizará cada valor (Tipo 'zscore' o Estandarización)
def normalize(valor, media, desviacion):
    return (valor-media)/desviacion

# Registramos la función como UDF
normalize_udf = udf(normalize, FloatType())

# Calculamos la media y la desviación típica antes de normalizar
df_stats = df_variables.select(F.mean(F.col('TotalTarjetas')).alias('media'),F.stddev(F.col('TotalTarjetas')).alias('desviacion')).collect()
media = df_stats[0]['media']
desviacion = df_stats[0]['desviacion']

# Aplicamos la UDF a nuestro DataFrame para normalizar el "TotalTarjetas"
df_normalizado = df_variables.withColumn("TotalTarjetasNormalizado", normalize_udf(df_variables["TotalTarjetas"],F.lit(media),F.lit(desviacion)))
df_normalizado.select("TotalTarjetas", "TotalTarjetasNormalizado").show(5)
```

```
(2) Spark Jobs
df_normalizado: pyspark.sql.dataframe.DataFrame = [HomeTeam: string, AwayTeam: string ... 21 more fields]
+-------------+------------------------+
|TotalTarjetas|TotalTarjetasNormalizado|
+-------------+------------------------+
|            1|              -1.6117728|
|            7|                0.687878|
|            4|             -0.46194738|
|            7|                0.687878|
|            7|                0.687878|
+-------------+------------------------+
only showing top 5 rows
```

Por último, tenemos las **imputaciones de valores**. Lo más habitual es realizar imputación de valores nulos/desconocidos/vacíos de nuestros dataframes. Consiste en reemplazar estas situaciones por un valor adecuado según el análisis a realizar. Podemos sustituir los valores vacíos por un 0, o podemos sustituirlo por otro valor que haga que el registro no tenga influencia negativa en los resultados que buscamos.

En el siguiente ejemplo vamos a imputar nulos por el valor medio de la columna (esto haría, por ejemplo, que al calcular la media tras la imputación el resultado fuera el mismo).

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
1  # Ejemplo de imputación de nulos vía UDF
2
3  # Creamos un DF con un nulo en cada columna
4  df_null = spark.createDataFrame([(1.0, 1.0), (2.0, 2.0), (3.0, None), (None, 4.0), (5.0, 5.0)], ("a", "b"))
5  df_null.show(5)
6
7  # Calculamos la media de cada columna, ya que queremos cambiar esos nulos por la media de su columna
8  df_medias = df_null.select(F.mean(F.col('a')).alias('media_a'), F.mean(F.col('b')).alias('media_b')).collect()
9  media_a = df_medias[0]['media_a']
10 media_b = df_medias[0]['media_b']
11
12 # Utilizamos la función fillna para reemplazar los nulos de cada columna con su media
13 df_null_imp = df_null.fillna({'a':media_a, 'b':media_b})
14 df_null_imp.show(5)
```

```
(7) Spark Jobs
df_null:  pyspark.sql.dataframe.DataFrame = [a: double, b: double]
df_null_imp:  pyspark.sql.dataframe.DataFrame = [a: double, b: double]
+----+----+
|   a|   b|
+----+----+
| 1.0| 1.0|
| 2.0| 2.0|
| 3.0|null|
|null| 4.0|
| 5.0| 5.0|
+----+----+

+----+---+
|   a|  b|
+----+---+
| 1.0|1.0|
| 2.0|2.0|
| 3.0|3.0|
|2.75|4.0|
| 5.0|5.0|
+----+---+
```

Si trabajamos con versiones de Spark iguales o superiores a la 2.2, tenemos la posibilidad de utilizar la clase *Imputer* para realizar esta operación. El siguiente ejemplo es equivalente totalmente al anterior:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
1  # Ejemplo de imputación de nulos vía clase Imputer (Spark >= 2.2.0)
2
3  from pyspark.ml.feature import Imputer
4
5  # Indicamos las columnas a procesar, y el nombre de las columnas finales
6  imputer = Imputer(inputCols=["a", "b"], outputCols=["a_imp", "b_imp"])
7
8  # Trabajamos sobre el mismo df_null con nulos del ejemplo anterior
9  df_null.show(5)
10
11 # Aplicamos la imputación y vemos que el resultado es el mismo
12 df_null_imputer = imputer.fit(df_null).transform(df_null)
13 df_null_imputer.show(5)
```

▸ (10) Spark Jobs

▸ df_null_imputer: pyspark.sql.dataframe.DataFrame = [a: double, b: double ... 2 more fields]

```
Using Spark version 2.3.1
+----+----+
|   a|   b|
+----+----+
| 1.0| 1.0|
| 2.0| 2.0|
| 3.0|null|
|null| 4.0|
| 5.0| 5.0|
+----+----+

+----+----+-----+-----+
|   a|   b|a_imp|b_imp|
+----+----+-----+-----+
| 1.0| 1.0|  1.0|  1.0|
| 2.0| 2.0|  2.0|  2.0|
| 3.0|null|  3.0|  3.0|
|null| 4.0| 2.75|  4.0|
| 5.0| 5.0|  5.0|  5.0|
+----+----+-----+-----+
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

6. Ecosistema de Apache Spark. Módulos principales

En los puntos anteriores hemos profundizado en el corazón de Spark (Spark Core), sus funcionalidades y módulos principales, así como el paradigma de programación a través de sus contextos y APIs.

Una vez hemos adquirido este conocimiento más profundo de lo que es Spark, vamos a ver otros de los módulos que acompañan a la distribución de Spark y la complementan con funcionalidades muy relevantes, especialmente aquellas que nos permiten dotar a nuestras aplicaciones Spark con capacidades de Machine Learning.

En la siguiente figura vemos los componentes más relevantes que se incluyen en Spark sobre su funcionalidad básica:

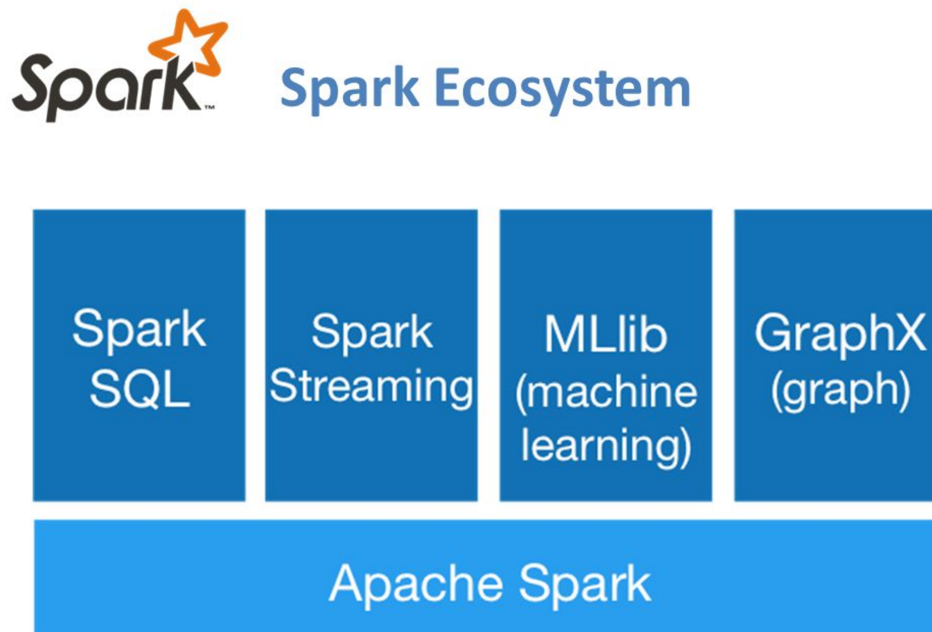


Figura 12: Beneficios del Big Data en las empresas[23]

- Spark SQL: este componente es un framework distribuido para el procesado de

[23] https://spark.apache.org

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

datos estructurados (aquellos que están bien organizados y relacionados, como puede ser una base de datos relacional, hojas de cálculo, ficheros columnares con cabecera, etc.) En estos casos, Spark SQL actúa básicamente como un motor de consultas SQL distribuido.

- Spark Streamming: es un añadido (add-on) al core de las APIs Spark que permite el procesado de flujos de datos generados en tiempo real, de forma escalable, tolerante a fallos y con alto rendimiento. Este add-on puede acceder a fuentes de datos como Kafka, Flume, Kinesis o incluso escuchar en puertos TCP. Un ejemplo de su uso sería el procesado en tiempo real de datos de una competición deportiva durante su desarrollo para predecir el resultado.
- Spark MLib: es la librería que dota de capacidades de aprendizaje automático o Machine Learning a Spark. Su principio de diseño fue el simplificar y hacer escalables las funcionalidades de Machine Learning.
- GraphX: es un API de Spark para el manejo y la ejecución paralela de grafos, que presenta un motor de analíticas de redes de grafos y un almacén de datos.

Gracias a estos módulos, Spark se convierte en una plataforma extremadamente versátil para diferentes tipos de procesado de datos como el análisis en tiempo real, el procesado de datos estructurados, procesado de grafos, etc. Sumado al potente APIs disponible y el soporte para varios lenguajes, la convierten en una de las opciones más usadas en la actualidad y con un ecosistema en continuo crecimiento.

En los siguientes apartados se realizará un recorrido por estos módulos, profundizando especialmente en las técnicas Machine Learning a través de Spark MLib.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

7. Caso de uso de Apache Spark (I). Spark SQL

Como se comentó al ver el ecosistema Spark, uno de los componentes que nos dan más versatilidad a la hora de procesar fuentes de dato estructuras es Spark SQL.

El módulo de Spark SQL nos permite trabajar en Spark con estas fuentes estructuradas o semiestructuradas, esto es, aquellas fuentes donde la información se encuentra en bases de datos, es información etiquetada, modelada y controlada, que solemos encontrar en forma de filas o columnas, con una o varias claves que relacionan unas con otras.

Spark SQL trabaja con el API DataFrames que, como vimos anteriormente, se adaptan muy bien a este tipo de información estructurada. En la práctica, Spark SQL puede construir un DataFrame a partir de fuentes de datos como Cassandra, Hive, Elastic Search, JDBC… además de directamente desde ficheros con formato csv, json, avro, etc. Como vemos, el abanico de posibilidades que nos da es muy grande y abarca los orígenes más típicos, haciendo que Spark SQL esté presente en la gran mayoría de aplicaciones Spark que construyamos.

A nivel conceptual, explicamos que los DataFrame son equivalentes a las tablas de una base de datos relacional (o de un fichero con columnas y una cabecera). El nombre de las columnas del fichero o de los campos de la tabla, sería el nombre de cada campo del DataFrame.

Una vez que hemos empelado Spark SQL para conectarnos a una fuente de datos, estos ya pasan a ser gestionados en memoria mediante Spark como DataFrames.

Como comentamos cuando hablamos de los contextos Spark, para acceder a la funcionalidad de SparkSQL[24] será necesaria la creación de un SQLContext o un SparkSession, según la versión de Spark con la que trabajemos. También es importante

[24] http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

hacer notar que los tipos de datos que soporta Spark SQL están en el paquete *pyspark.sql.types*[25] y que serán frecuentes las operaciones de *cast* o transformación de datos de un tipo a otro. Un ejemplo es cuando estamos leyendo datos de un fichero (a priori todos son cadenas de texto) que puedan contener valores numéricos.

Cuando trabajamos con valores decimales (tipos *float* y *double*), es frecuente que alguno de los datos se rellene con "NaN" (Not a Number). Es un tipo especial que se utiliza para los valores desconocidos, cuando al leer la fuente de datos no se puede determinar el valor adecuado.

Un ejemplo de esto es intentar leer el valor **1'25** en un campo float, ya que el apóstrofe (') no se reconoce como separador decimal y el valor leído será finalmente NaN

Otro aspecto interesante, es la capacidad de crear vistas para poder trabajar con los DataFrames utilizando sintaxis SQL. De esta forma operaremos con ellos como si lo hiciéramos con una tabla de una base de datos.

En el siguiente ejemplo se muestra una primera aplicación PySpark con Spark SQL, donde cargamos dos ficheros de distinto formato y, posteriormente y tras convertirlos en DataFrames, utilizamos una vista para realizar un filtrado sobre ellos.

Los ficheros de datos – *datasets* - empleados han sido:

```
nombre,medallas
Rosa,3
Pablo,1
Pedro,2
María,0
```

Figura 13: Contenido del fichero medallistas.csv

---

25 http://spark.apache.org/docs/latest/api/python/pyspark.sql.html#module-pyspark.sql.types

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
{"nombre":"Rosa", "medallas":3}
{"nombre":"Pablo", "medallas":1}
{"nombre":"Pedro", "medallas":2}
{"nombre":"María", "medallas":0}
```

Figura 14: Contenido del fichero medallistas.json

```
from pyspark.sql import SparkSession

# Incializamos SparkSession para trabajar con SparkSQL
spark = SparkSession.builder.enableHiveSupport().getOrCreate()

# Cargando ficheros en varios formatos (CSV y JSON)

df_csv = spark.read.option("header","true").option("inferSchema", "true").csv('medallistas.csv')
df_json = spark.read.option("header", "true").json("medallistas.json")
print "DataFrame desde CSV:"
df_csv.show(10)
print "DataFrame desde JSON:"
df_json.show(10)

print "Vista para operaciones SQL sobre df_csv"
df_csv.createOrReplaceTempView("DF")

# Nos quedamos con los que han tenido mínimo 2 medallas
spark.sql("SELECT nombre, medallas from DF where medallas > 1").show(10)
```

```
DataFrame desde CSV:
+------+--------+
|nombre|medallas|
+------+--------+
|  Rosa|       3|
| Pablo|       1|
| Pedro|       2|
| María|       0|
+------+--------+

DataFrame desde JSON:
+--------+------+
|medallas|nombre|
+--------+------+
|       3|  Rosa|
|       1| Pablo|
|       2| Pedro|
|       0| María|
+--------+------+

Vista para operaciones SQL sobre df_csv
+------+--------+
|nombre|medallas|
+------+--------+
|  Rosa|       3|
| Pedro|       2|
+------+--------+
```

Figura 15: Código de ejemplo, utilizando Spark SQL

Nótese en el ejemplo anterior que ahora tenemos dos elementos (*df_json* y *df_csv*) que son ambos DataFrame a pesar de sus formatos de origen distintos. Podríamos empezar a realizar cualquier transformación/acción con ellos con independencia de este origen, ya que

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

desde el momento de su creación ya son gestionados por Spark.

Resumiendo, las aportaciones de Spark SQL serían:

- Permite unificar información de fuentes de datos heterogéneas (formatos, sistemas de almacenamiento, etc.).
- Utilizar sintaxis SQL para la transformación de los datos, consiguiendo así una mayor abstracción y evitando las peculiaridades de cada sistema.
- Optimización en las transformaciones sin tener que entrar en cuestiones de bajo nivel.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

8. Introducción a Machine Learning o Aprendizaje Automático (I)

El *Machine Learning* es una disciplina que estudia el análisis de grandes cantidades de datos con el objetivo de obtener conocimiento a partir de ellos. Tratándose por tanto de una tecnología de manejo y análisis de información que aprovecha la capacidad existente hoy en día de procesamiento, almacenamiento y transmisión de datos a gran velocidad y con un bajo costo.

Pero tal y como aventuramos en módulos anteriores, la extracción de información de un conjunto de datos no se basa en una nueva ciencia, sino que cuenta ya con un largo recorrido y ha sido rebautizada con numerosos nombres en su transcurso. En la década de 1960, dentro de la rama de la Estadística se hacía referencia con términos como *data fishing* (buscar o “pescar” en los datos) o *data dredging* (dragado de datos) para referirse al análisis de grandes volúmenes de datos. En la década de 1990, aparece el término *data mining* (minería de datos) el cual aún perdura y se entremezcla con el de *Machine Learning* con el que se conoce actualmente al proceso de obtención de conocimiento a partir de los datos por medio de su análisis.

El desarrollo y continuos cambios de nombre, de la disciplina de extracción de conocimiento de un conjunto de datos, ha tenido mucho que ver con el cambio de concepción de los datos, unido a la gran cantidad que de estos que se generan y almacenan continuamente en cualquier ámbito. Otro factor que también ha favorecido la consolidación del análisis de datos como disciplina es el gran avance en los últimos tiempos en las prestaciones y capacidad computacional, lo que ha permitido explorar numerosos modelos matemáticos que habían sido descartados en su momento por no disponer del desarrollo necesario para que estos pudieran ser viables.

Tradicionalmente, las técnicas de análisis de datos se aplicaban sobre información contenida en bases de datos estructuradas. No obstante, actualmente está cobrando una importancia cada vez mayor el análisis de datos desestructurados como es la información contenida en ficheros de texto (*text mining*), en Internet (*web mining*) … Otro de las

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

necesidades que se han presentado es la necesidad de obtener y presentar resultados en tiempo real, de modo que la información que arrojen dichos algoritmos pueda ser interpretada en los más pronto posible, pudiendo de este modo ser empleados en la alerta temprana frente a alarmas en una cadena de montaje, la detección instantánea del fraude en operaciones bancarias, un sistema de recomendación de productos en una tienda en línea, en la toma de decisiones en los cambios de jugadores que se encuentran disputando cierto evento deportivo, entre otros muchos posibles casos de uso, algunos de ellos recogido en la siguiente dirección[26].

De toda la información que se ha visto a lo largo de los diferentes módulos, podríamos definir los diferentes procesos que hay que realizar durante la extracción de información como una combinación de:

- Recopilación de datos.
- Preparación de datos.
- Aplicación de modelos algorítmicos.
- Análisis de resultados.

De igual manera que se muestra en la siguiente imagen, dejando claro que en todo momento de extracción de información nos debemos de plantear si las decisiones que se han ido tomando son las correctas, o por el contrario es necesario rehacer alguno de los pasos anteriores de modo que el conocimiento que se extraiga al final del todo el proceso sea de la mayor calidad posible.

[26] https://machinelearningmastery.com/practical-machine-learning-problems/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

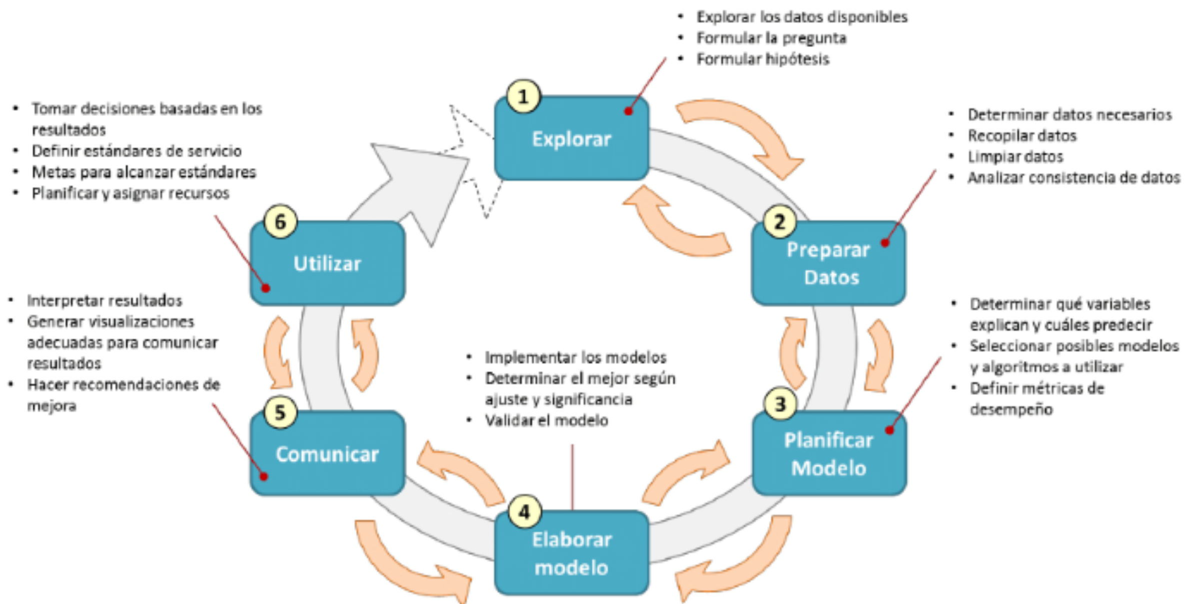


Figura 16: Ciclo de vida de análisis de datos[27]

8.1. Exploración

El proceso de **explorar** estaría compuesto por todas aquellas acciones que nos hagan llegar a recopilar todas aquellas fuentes de información, a las que posteriormente se accederá para la captación de los diferentes datos. Lo que se pretende en esta fase es la integración en un único repositorio de toda la información que va a ser procesada, el lugar en el que se aloja dicha información es conocido con el nombre de *data warehouse*.

Las fuentes de datos que se aglutinan dentro de un proyecto no suelen contener la estructura con la que nos gustaría trabajar, haciéndose necesario por tanto un conjunto de acciones que lleven las diferentes fuentes de información a un esquema con el que nos

---

[27] Extraída de la publicación de Rodríguez, P., Truffello, R., Suchan, K., Varela, F., Matas, M., Mondaca, J., ... & Allende, C. (2016). Apoyando la formulación de políticas públicas y toma de decisiones en educación utilizando técnicas de análisis de datos masivos: el caso de Chile.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

sintamos cómodos. Este conjunto de acciones se enmarca dentro de los sistemas ETL (*Extraction-Transformation-Load*), los cuales podrían ser enmarcados dentro de una fase de preparación del dato, pero en la gran mayoría de las ocasiones las transformaciones que se aplican se realizan buscando tener un almacén de datos unificado, pero sin garantizar la calidad de lo que en él se aloja.

8.2. Preparación y planificación

Tras una primera fase de acciones de exploración, se continua con un conjunto de acciones de preparación en mayor medida de la información alojada en el *data warehouse*. En la fase de **preparación de los datos**, se realizar la selección de variables o características que van a ser estudiadas y la eliminación de registros con lo que no vamos a poder trabajar por el tipo de algoritmos con los que se ha decidido trabajar, encontrándose, por tanto, las tareas que se realicen entremezcladas con la fase de **planificación del modelo**.

El motivo por el que muchos autores no diferencien entre estas dos fases de trabajo, es que consideran que el principal motivo radica en cuál será la información que queramos predecir, objetivo que consideran que deberíamos de tener claro antes incluso de haber comenzado con las tareas de exploración.

De igual modo que pasaba en la primera etapa, la información alojada dentro del *data warehouse*, aun habiendo pasado un primer conjunto de acciones de ETL, es plausible que debamos de implementar nuevos procesos, puesto que en la mayoría de las ocasiones los datos alojados en el *data warehouse* requerirán de una serie de transformaciones que los preparen de cara a los procesos posteriores. Las transformaciones más habituales que se suelen realizar dentro de un conjunto de registros son:

- Generar una nueva característica o variable como discretización de cierta variable cuantitativa, se transforma cierta medida en diferentes grupos cualitativos.
- Generar una nueva característica o variable como numerización de cierta variable cualitativo, se transforma en cuantitativo.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Este procedimiento es empleado cuando los modelos no admiten variables categóricas, un ejemplo lo tendríamos con la regresión lineal. El problema es solucionado con una posible asunción de que las rectas que definen cada una de las categorías son paralelas entre sí, y por tanto se generan tantas variables dummy (variable que toma únicamente los valores 0 ó 1) como categorías hubiera en la variable en cuestión, cada una de las nuevas variables tendrá la unidad cuando para una única categoría existente en la variable cualitativa. Un ejemplo sería la siguiente transformación:

Color
Azul
Naranja
Morado
Rojo
Morado
Azul
Otro
Rojo

→

Rojo	Naranja	Azul	Morado	Otro
0	0	1	0	0
0	1	0	0	0
0	0	0	1	0
1	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	0	0	1
1	0	0	0	0

Tabla 1: Ejemplo de numerización de la variable estado civil

- Generar una nueva característica o variable como función de un conjunto de características ya presentes en los datos.

- Cambiar el rango de valores que toma determinado atributo, el caso más común es transformar el rango de valores a un intervalo de valores comprendidos entre el

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

cero y la unidad, dicha transformación es realizada mediante la siguiente formula:

$$Valor\ transformado = \frac{valor\ -\ valor\ mínimo}{valor\ máximo\ -\ valor\ mínimo}$$

Con una transformación adecuada podemos cambiar el rango de valores entre los que toma valor cierta característica o variable, en aquel que más nos convenga de cara a nuestro modelo.

- En numerosas ocasiones, el número de características de las que se disponen es demasiado grande, haciendo que los modelos requieran de un alto tiempo hasta lograr resultados, dado que deben trabajar con demasiadas variables que no aportan información relevante. Para evitar esto, existen numerosos algoritmos de reducción de la dimensionalidad que buscan reducir el número de características con las que trabajaran los modelos posteriores.

  El procedimiento más conocido a la hora de reducir el número de columnas es el análisis de componentes principales (PCA), que proyecta los atributos iniciales en un espacio de dimensionalidad mucho menor, según la información que queramos seguir conservando, pero teniendo la ventaja de que se eliminan las redundancias y posibles combinaciones entre las diferentes variables.

  El algoritmo más extendido de PCA trabaja con variables de tipo numérico, pero existen variantes del mismo que aceptan variables categóricas. El problema de emplear este tipo de procedimientos es asumir una mejora en los modelos a costa de perder la interoperabilidad de los mismos, dado que las nuevas características con las que se trabajan son combinaciones de las primeras.

  Si se quisiera conservar la interoperabilidad del modelo final, es decir, poder interpretar de manera sencilla cada una de las características que el modelo final emplea, deberemos usar procedimientos de selección del conjunto de características inicial. Los métodos para la selección de características son principalmente dos:

  1. Los basados en la elección de los atributos que mejore describen el

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

problema en cuestión.

2. Los que buscan variables independientes mediante tests de sensibilidad, algoritmos de distancia o heurísticos.

Otro conjunto de técnicas que pueden ser aplicadas son las de selección, con las que pretendemos prescindir de aquello registros que resultan irrelevantes de cara a la aplicación de los modelos posteriores. El filtrado puede realizarse en las dos direcciones de la información que recogen los datos:

- Supresión de atributos que no resultan de interés de cara al modelo que va a ser empleado.
- Proceso de selección de registros representativos. En numerosas ocasiones, la información de un gran conjunto de registros puede ser representada con la muestra de una parte de ellos, haciendo que las conclusiones a las que se llegan tengan la misma calidad, pero realizado de un punto más eficiente, dado que, al trabajar con un menor número de individuos, los resultados del modelo matemáticos son arrojados en un menor tiempo. Las técnicas de selección de individuos representativos se enmarcan dentro de las técnicas de muestreo.

Si pensamos con detenimiento, nos daremos cuenta que, dentro del conjunto de transformaciones y filtros descritos, en ningún momento se ha tratado la calidad de la información con la que vamos a trabajar, es por ello que debemos dedicar un tiempo a analizar la información alojada en cada uno de los registros, dado que en numerosas ocasiones alojan valores ausentes o valores erróneos.

Es muy normal que dentro del conjunto de registro que nos procedemos a analizar que existan valores ausentes, cuya falta de información impacta en gran medida al modelo que vamos a emplear. Para solucionar esta carencia de información, los analistas siguen diferentes acciones, dependiendo del contexto en el que se esté trabajando, las más habituales serían:

- Asumir la falta de información y continuar con el conjunto de modelos que van a

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

ser estudiados.

- Eliminar aquellos registros donde la información no es completa, y cuya falta impacta en las conclusiones finales.

- Imputar un valor en aquellos lugares donde la información no es completa, de modo que, tras la aplicación de alguno de los múltiples procedimientos, tengamos información para aquellos registros donde carecíamos. Esta práctica es arriesgada, dado que estamos modificando el valor de los registros originales, y podemos estar cambiando el comportamiento del conjunto de los datos.

En igual medida que la usencia de información, los registros con los que se trabaja nos podemos encontrar con la existencia de valores erróneos. El disponer de una información con errores hace que la experiencia del analista en la materia se vuelva esencia, dado que será capaz de localizar aquellos individuos a la mayor brevedad, reduciendo por tanto el tiempo dedicado al modelado de los datos. Una vez detectados los registros con valores erróneos, el analista tomará decisiones similares a las que tomaba con los valores ausentes, tales como:

- Asumir el valor erróneo en la información con la que se está trabajando, y continuar con el conjunto de modelos que van a ser estudiados.

- Eliminar aquellos registros donde la información es errónea, y cuya inclusión impacta en las conclusiones finales.

- Imputar un valor en aquello lugares donde la información es errónea, de modo que, tras la aplicación de alguno de los múltiples procedimientos, tengamos información para aquellos registros donde carecíamos de una información verdad. Esta práctica es arriesgada, tal y como comentábamos antes, dado que estamos modificando el valor de los registros originales, y podemos estar cambiando el comportamiento del conjunto de los datos.

En numerosas publicaciones, la detección de registros con información errónea es considerado como el tratamiento de observaciones atípicas, pero es necesario tener claro

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

que no es lo mismo tener un error en la toma de medida de cierto valor, a que en cierto momento dentro de nuestra base de datos aparezca un individuo con un valor extremo pero verídico. Tal y como hemos comentado antes, la decisión que tomé el analista debe de encontrarse justificada, dado que según actué el modelo se verá influido de diferente forma.

8.3. Modelado

Una vez que el analista dispone de la información de un modo adecuado, procede con la fase de **elaboración de modelos**. En la fase de elaboración de modelos, el objetivo es encontrar una representación de la información de modo que se ajuste lo máximo posible a la realidad, lo que hará que el modelo resultante pueda ser aplicado posteriormente obteniendo conocimiento relevante de sus resultados.

Los modelos que el analista aplicará en la etapa de modelado, serán diferentes según el tipo de problema al que esté tratando de dar solución. Los modelos pueden ser clasificados de múltiples maneras, y la que aquí se muestra es según el tipo de conocimiento que estos son capaces de generar:

- Los modelos predictivos son empleados para obtener un valor desconocido de algunos de los atributos que componen la base de información, pera ello es necesario emplear algunos registros de la base de datos.

  Dentro de este tipo de modelos podríamos clasificar los modelos según el tipo de variable que se está intentando predecir.

  - Para los modelos de regresión, el analista trata de encontrar un modelo que, al ser aplicado a un nuevo ejemplo sea capaz de obtener un valor para la variable objetivo.

    La variable objetivo suele ser de tipo cuantitativo.

  - Para los modelos de clasificación, el analista trata de encontrar un modelo que. Al ser aplicado a un nuevo ejemplo sea capaz de clasificar el nuevo

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

registro dentro de un conjunto predefinido de clases.

La variable objetivo suele ser de tipo cualitativa.

Es necesario remarcar que lo expresado anteriormente se trata de una generalización, dado que algunos conjuntos de datos requieren un tratamiento diferente según la información que recogen. Algunos datos que requieren de un análisis especial son los datos referentes a eventos temporales, al procesamiento de textos, a datos geoespaciales, a datos de supervivencia, al procesamiento de imágenes, al procesamiento de sonido, al procesamiento conjunto de imagen y sonido, entre otros muchos.

El principal motivo de necesitar un tratamiento especial como consecuencia que los registros no son independientes dado que, por ejemplo, en los modelos temporales, el valor en cierto momento se encuentra correlacionado con observaciones anteriores.

Otro detalle que hay que tener presente, es la comparación entre los diferentes modelos que el analista puede haber probado, de modo que pueda tomar una decisión sobre cuál de todos ellos presenta como candidato en la etapa de comunicación. Uno de los procedimientos más extendidos es la segmentación del conjunto de datos en dos subconjuntos de instancias, uno dedicado para los entrenamientos de los diferentes modelos y el otro empleado para la evaluación y comparación de los mismos.

Una técnica algo más avanzada de evaluación de modelos, a la vez que la más utilizada, es la técnica de validación cruzada (*k-fold cross validation*). En este caso, para validar un modelo, se elige aleatoriamente un número de elementos de los datos como conjunto de prueba y con el restante, como conjunto de entrenamiento, se construye el modelo; y el proceso se repite hasta que todos los datos han sido empleados como conjunto e prueba. Los resultados arrojados por este procedimiento de evaluación de modelos garantizan que son independientes de la partición entre datos de entrenamiento y prueba, dado que el resultado final es la media entre todas las particiones.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Los modelos descriptivos son empleados para obtener una descripción del conjunto de datos, buscando que la comunicación de la información sea lo más sencilla posible. Algunas de las más empleadas por los analistas son:
    - Generación de grupos (*Clustering*). Mediante estos procedimientos se pretende generar diferentes grupos de instancias que sean lo más homogéneos entre sí, pero lo más heterogéneos con el resto de grupos, es decir, buscamos que los individuos que componen un grupo sean lo más iguales entre sí, y lo más diferentes entre los diferentes grupos.
    - Asociación. Mediante los procedimientos que son categorizados de esta categoría, se busca encontrar combinaciones lineales entre las diferentes características que componen la base de datos. Un ejemplo de este tipo de modelos lo podemos encontrar en los artículos que componen la bolsa de la compra de cierto supermercado:

ID	Leche	Pan	Agua	Cerveza	Pañales	Patatas
1	0	0	0	1	1	1
2	0	1	1	0	0	0
3	0	1	0	0	0	0
4	1	0	1	1	1	0
5	1	0	1	1	1	1

Tabla 2: Ejemplo regla de asociación

En los datos anteriores, vemos cuando en la bolsa de la compra del supermercado encontramos artículos de pañales y patatas, también nos encontramos que hay cerveza, con lo que llegaríamos a la siguiente regla de asociación:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$\{Pañal, Patata\} \Rightarrow \{Cerveza\}$$

Dentro de todas y cada una de las categorías de modelos que han sido descritas, existen numerosos algoritmos que pueden ser aplicados. Lo que hay que resaltar, es que el analista no debe ceñirse a una sola técnica, debiendo explorar dentro de todos aquellos modelos que crea que van a representar los patrones de comportamiento presentes en los datos.

8.4. Comunicar resultados

En el momento en el que el analista dispone de un modelo que representa las características recogidas dentro de los datos, llega el momento de **comunicar** los resultados obtenidos. La tarea de comunicación es una tarea esencial, dado si no se realiza de un modo adecuado, todo el trabajo previo puede no verse reflejado.

Si reflexionamos un tiempo de cuáles son las causas por las que se generan visualizaciones, y las apunta, seguramente llegue a la siguiente lista o sus ideas estén incluidas entre las siguientes:

- *Responder preguntas*: Básicamente la visualización nos permite obtener de forma resumida una respuesta ante una pregunta que tenemos ante los datos.
- *Tomar Decisiones*: Un ejemplo clásico son los cuadros de mandos de indicadores económicos de las empresas para determinar que se vende más, que menos y tomar decisiones en cuanto a la estrategia empresarial.
- *Ver los datos en contexto*: Para poder comparar distintos indicadores, dentro de un contexto de ventas totales, geográficos. Por ejemplo, si se comparan las ventas de una tienda en Madrid con las ventas de otra tienda en Salamanca, de la misma empresa, se tendrá que tener en cuenta la población de la ciudad, en qué zona de la ciudad está, entre otras muchas características.
- *Encontrar patrones*, como se puede ver si en los meses de verano se vende más que en los meses de invierno durante varios años.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- *Presentar argumentos para contar una historia*: Cuando se indica que es una historia, puede ser la presentación de un producto, argumentar un proceso o un análisis.

- *Inspirar* para descubrir nuevas posibilidades de análisis, nuevas preguntas a responder o proponer nuevas ideas que no se hubieran propuesto previamente.

Con estas conclusiones, se ve que realizar visualizaciones está intrínsecamente asociado con los procesos de análisis, resumen de resultados y comunicación.

Si buscamos en un diccionario que es visualizar, existen múltiples definiciones. Una de las más precisas es la que dice que Visualizar son "Tecnologías que transforman datos en información mediante elementos visuales".

En 1987, el autor Michael Cooley en su libro "Arquitecto o Abeja" (Cooley, M. (1980). *Architect or bee?* Slough: Langley Technical Services), considera que existe un flujo de la información o los datos que va de los datos a la sabiduría.

1. Los *datos*, *organizados* y empleados debidamente, pueden convertirse en *información*.

2. La información, absorbida, comprendida y aplicada por las personas, puede convertirse en *conocimientos*.

3. Los *conocimientos aplicados* frecuentemente en un campo pueden convertirse en *sabiduría*.

4. Finalmente, *la sabiduría* es la base de la *acción positiva*.

Independientemente, del tema moral que supone considera que la sabiduría es la base de la acción positiva (todos sabemos que la sabiduría puede emplear para malas prácticas o acciones no positivas para la sociedad), si consideramos los niveles de abstracción o inferencia, vemos que se va evolucionando en mayor complejidad cognitiva según el nivel en que nos encontremos.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Cuando la visualización de datos es eficaz, se altera el equilibrio entre la percepción y la cognición al aprovechar más las capacidades del cerebro.

Al ver (es decir, la percepción visual), este proceso es manejado por la corteza visual situada en la parte posterior del cerebro, es extremadamente rápido y eficiente. Vemos de inmediato, con poco esfuerzo.

Pensando (es decir, la cognición), este proceso es manejado principalmente por la corteza cerebral en la parte delantera del cerebro, es mucho más lento y menos eficiente.

La conclusión es evidente: Los métodos de visualización de datos y presentación tradicionales requieren pensamiento consciente para casi todo el trabajo. Con la visualización de datos se desplaza el equilibrio hacia un mayor uso de la percepción visual, aprovechando nuestros poderosos ojos siempre que sea posible.

El motivo por el cual nos deberíamos de plantearnos visualizar, podría ser definido mediante las dos definiciones siguientes.

William S. Cleveland, en su libro "Visualizing Data", presenta la siguiente definición (se incluye en inglés para no desvirtuar la misma): *"Visualization is critical to data analysis. It provides a front line of attack, revealing intricate structure in data that cannot be absorbed in any other way. We discover unimagined effects, and we challenge imagined ones."*. Básicamente, lo que el autor proponer es que la visualización de datos es la herramienta natural para analizar grandes datos y que no se podrían resolver de otra forma.

Otra definición que considero muy acertada es la descrita por Andy Kirk en su libro "Data Visualisation. A Handbook for Data Driven Design" que se puede ver en la figura, donde se indican los conceptos de representación (forma de la visualización), la presentación (a nivel diseño, colores, usabilidad), datos y comprensión (por muy bonita que sea la visualización si no se entiende, no es útil).

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

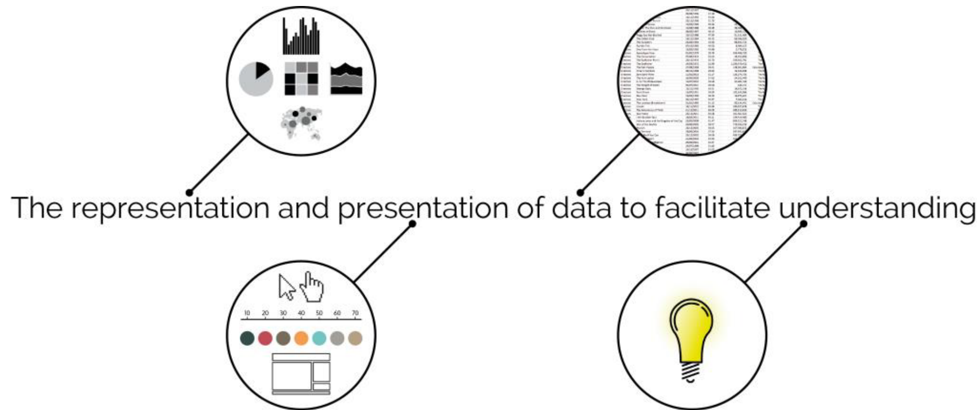


Figura 17: Descripción de Andy Kirk de que representa la visualización[28]

Algunos de los gráficos que pueden ser empleados para la transmisión del conocimiento son lo que se recogen en la siguiente imagen:

[28] Extraída de: Kirk, A. (2016). *Data visualisation: a handbook for data driven design*. Sage.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

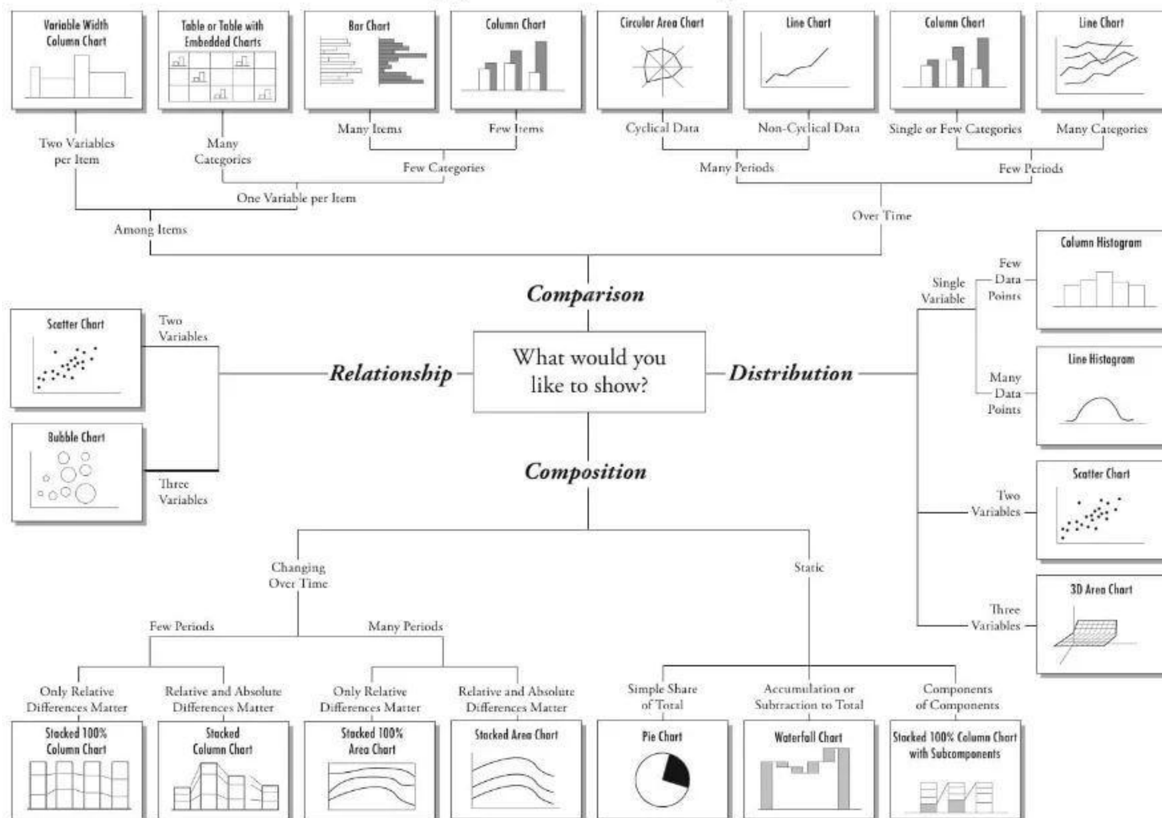


Figura 18: Recopilación de algunos visualizaciones[29]

Una vez concluida la etapa de visualización, se pasa a la etapa de **utilización** en la que se buscar generar con el conocimiento aprendido previamente, extraer nuevas conclusiones antes nuevos escenarios.

Una vez que un modelo llega a la etapa de utilización, deberemos estar atentos cuando el rendimiento del mismo desciendo de los valores esperados, prestando atención de si las instancias que empleamos para su entrenamiento puedan describir escenarios que ya no describen la nueva realidad, haciendo por tanto que, el analista deba volver a plantearse nuevas preguntas, o volver a entrenar el modelo con un conjunto de instancias más novedosas.

---

[29] Imagen extraída de: https://github.com/mainkoon81/Study-V001-Visualization-Tableau

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

9. Machine Learning o Aprendizaje Automático (II)

El aprendizaje automático trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento.

Los principales algoritmos de aprendizaje automático se agrupan en dos tipos dependiendo de su manera de funcionar, según cuál sea el objetivo.

Aprendizaje supervisado

El algoritmo produce una función que establece una correspondencia entre las entradas (también conocidas como predictores) y las salidas (variables objetivo) deseadas del sistema. La base de conocimiento del sistema está formada por ejemplos que se han dado anteriormente, esta base o datos de entrada de conocen como conjunto de entrenamiento.

Básicamente estos algoritmos se dividen en dos grupos dependiendo del tipo de problema que quieren resolver:

- Predicción en los que a partir de unas variables de entrada el algoritmo “predice” que valor numérico tendría la variable que se denominada objetivo o dependiente.
- Clasificación. En este caso el algoritmo trata de etiquetar (clasificar) una serie de vectores que vienen dados por las variables de entrada (variables independientes) utilizando una entre varias categorías (clases).

Aprendizaje no supervisado

Todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

Con relación al aprendizaje no supervisado los modelos más utilizados son:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Reglas de Asociación (*Association Rules* o *AR*)
- Patrones Secuenciales (*Sequential Patterns* o *SP*)
- Agrupamiento (*Clustering*)
- Reducción de la Dimensionalidad (*Dimensionality Reduction*)

Algunos autores hablan de un tercer grupo de aprendizaje, compuesto por conjunto de datos en los que no se dispone de la información en todos los registros de la variable respuesta, pero se decide utilizar la información que aportan dichos registros para enriquecer el modelo de datos. Este tipo de algoritmo combina el aprendizaje supervisado y no supervisado de manera que el modelo debe aprender las estructuras de cara a organizar los datos además de hacer predicciones o clasificaciones.

Se han introducido el nombre de algunos de los grupos de modelos más empleados, compuestos cada uno de ellos por una infinidad de algoritmos de aprendizaje automático, haciendo que los Data Science deban continuar aprendiendo e investigando nuevos procedimientos, de modo que sean capaces de seleccionar el más conveniente cuando se enfrenten ante un nuevo conjunto de datos. Una infografía que os puede venir muy bien para guiar vuestros análisis puede ser la siguiente:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

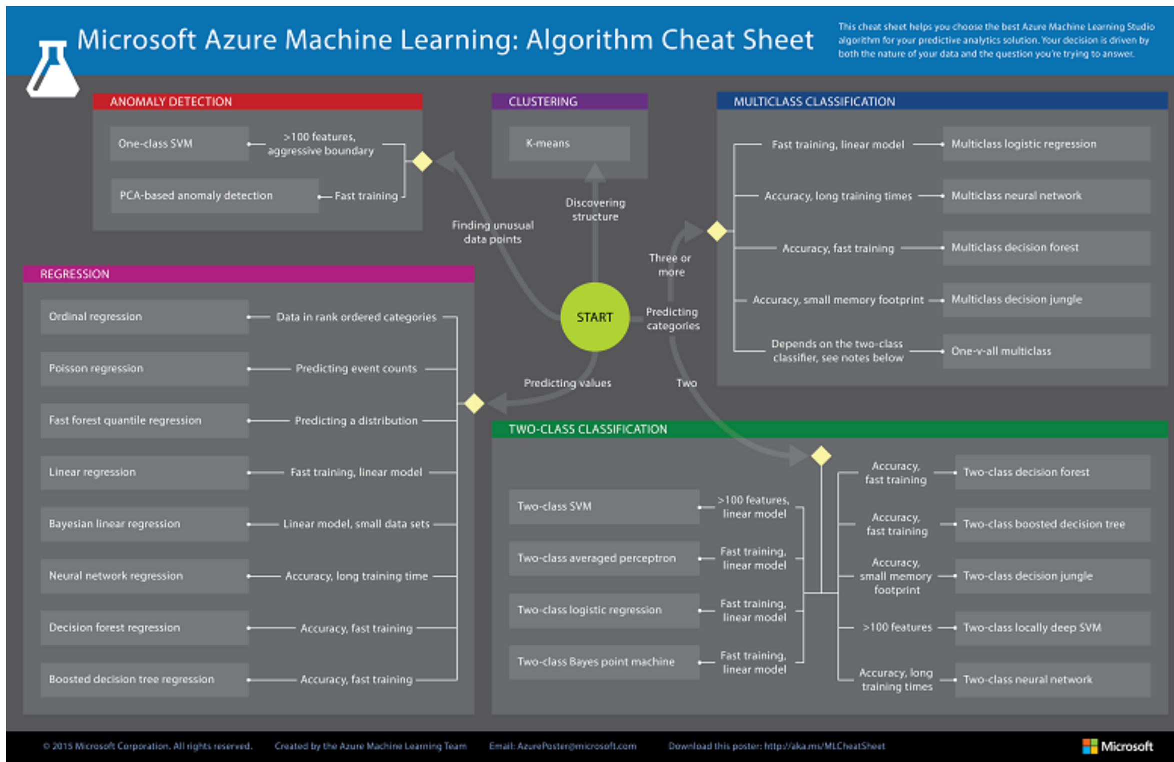


Figura 19: Hoja de referencia rápida de algoritmos de Machine Learning[30]

Recordar que, para llegar a comprender modelos complejos, es muy conveniente conocer otros más simples sobre los que se apoyan, por eso hemos intentado explicar los conceptos básicos de forma muy clara.

9.1. Aprendizaje supervisado

Dentro de los algoritmos de aprendizaje supervisado y siguiendo la clasificación de Jason Brownlee[31] podemos distinguir los distintos tipos:

- **Regresión (*Regression*):** Se suele utilizar para resolver problemas de predicción

[30] Infografía extraída de: https://docs.microsoft.com/es-es/azure/machine-learning/studio/algorithm-cheat-sheet
[31] http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

principalmente y en menor media en clasificación.

Los algoritmos de regresión intentan modelar las relaciones entre variables utilizando métodos que refinan dicho modelo utilizando de forma iterativa las medidas de error que se van obteniendo a medida que se va aplicando dicho modelo. Como principales algoritmos de regresión podemos citar:

- Ordinary Least Squares Regression (OLSR)
- Linear Regression (Predicción)
- Logistic Regression (Clasificación)
- Stepwise Regression
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)

- **Basados en instancias (o en memoria)**: Se utilizan tanto para predicción como para clasificación.

  Normalmente construyen una base con datos de ejemplo y comparan los nuevos datos con los de la base usando una medida de similitud con el objetivo de encontrar a los que se parecen más para hacer la predicción. Se centran, por tanto, en la representación de las instancias almacenadas y las medidas de similitud usadas entre instancias. Los principales algoritmos de este tipo serían:

  - k-Nearest Neighbour (kNN) o k Vecinos más cercanos
  - Learning Vector Quantization (LVQ)
  - Self-Organizing Map (SOM)
  - Locally Weighted Learning (LWL)

- **Regularización (*Regularization*)**: Se utilizan tanto para predicción como para

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

clasificación.

Suelen ser extensiones que se hacen sobre otros métodos (normalmente métodos de regresión) que penalizan los modelos más complejos en favor de modelos más simples que son también más fácilmente generalizables. Los algoritmos de regularización más populares son:

- Ridge Regression
- Least Absolute Shrinkage and Selection Operator (LASSO)
- Elastic Net
- Least-Angle Regression (LARS)

- **Árboles de Decisión (*Decision Tree*)**: Se utilizan principalmente en problemas de clasificación y también frecuentemente en predicción.

  Los Árboles de decisión construyen sus modelos basándose en los valores de los atributos de los datos realizando bifurcaciones en estructuras de tipo árbol hasta que se toma una decisión para cada una de las observaciones a la entrada.

  Por su sencillez de interpretación son unos de los modelos más utilizados en Machine Learning. Los algoritmos relacionados con árboles de decisión más populares son:

  - Classification and Regression Tree (CART)
  - Iterative Dichotomiser 3 (ID3)
  - C4.5 y C5.0
  - Chi-squared Automatic Interaction Detection (CHAID)
  - Decision Stump
  - M5

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Conditional Decision Trees

- **Bayesianos (*Bayesian*)**: Se utilizan para predicción y clasificación y son aquellos que explícitamente usan el teorema de Bayes para llegar a la respuesta. Los algoritmos bayesianos que más se utilizan son:
  - Naive Bayes
  - Gaussian Naive Bayes
  - Multinomial Naive Bayes
  - Averaged One-Dependence Estimators (AODE)
  - Bayesian Belief Network (BBN)
  - Bayesian Network (BN)

- **Redes Neuronales (*Neural Networks*)**: Se utilizan para predicción y clasificación.

  Son modelos inspirados por la estructura y funcionamiento de las redes neuronales biológicas que intentan buscar patrones en los datos. En este tipo de sistemas hay una gran cantidad de algoritmos distintos siendo los más actuales considerados como una nueva variante de redes neuronales que se denomina "*Deep Learning*" y que tratamos en otro apartado. Como algoritmos clásicos relacionados con redes neuronales podemos citar:
  - Perceptron
  - Back-Propagation
  - Hopfield Network
  - Radial Basis Function Network (RBFN)

- **Aprendizaje profundo (*Deep Learning*)**: Se usan principalmente para clasificación.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Como hemos indicado en el apartado anterior los métodos de Deep Learning son una actualización moderna de las redes neuronales que explotan la abundancia de recursos hardware a bajo precio para realizar la computación.

Tratan de construir redes neuronales más grandes y complejas y muchos de sus métodos también se relacionan con el aprendizaje semi-supervisado en el que grandes bases de datos contienen una baja proporción de datos etiquetados. Los algoritmos de Deep Learning más notorios en este momento son:

- Deep Boltzmann Machine (DBM)
- Deep Belief Networks (DBN)
- Convolutional Neural Network (CNN)
- Stacked Auto-Encoders

- **Ensamblados (*Ensemble*)**: Se utilizan fundamentalmente para clasificación, pero no de forma exclusiva.

  Estos algoritmos son modelos que se componen de otros múltiples modelos menos precisos que se entrenan de forma independiente y cuyas predicciones se combinan de alguna forma para llegar a una predicción general de más calidad. En los algoritmos ensamblados se tiene muy en cuenta como determinar qué tipos de clasificadores se han de combinar y de qué manera. Son técnicas muy potentes y por tanto también bastante populares. Los más destacados entre este tipo de algoritmos son:

  - Boosting
  - Bootstrapped Aggregation (Bagging)
  - AdaBoost
  - Stacked Generalization (blending)

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Gradient Boosting Machines (GBM)
- Gradient Boosted Regression Trees (GBRT)
- Random Forest

En la siguiente figura podemos ver un buen esquema que agrupa los diferentes tipos de algoritmos que acabamos de describir.

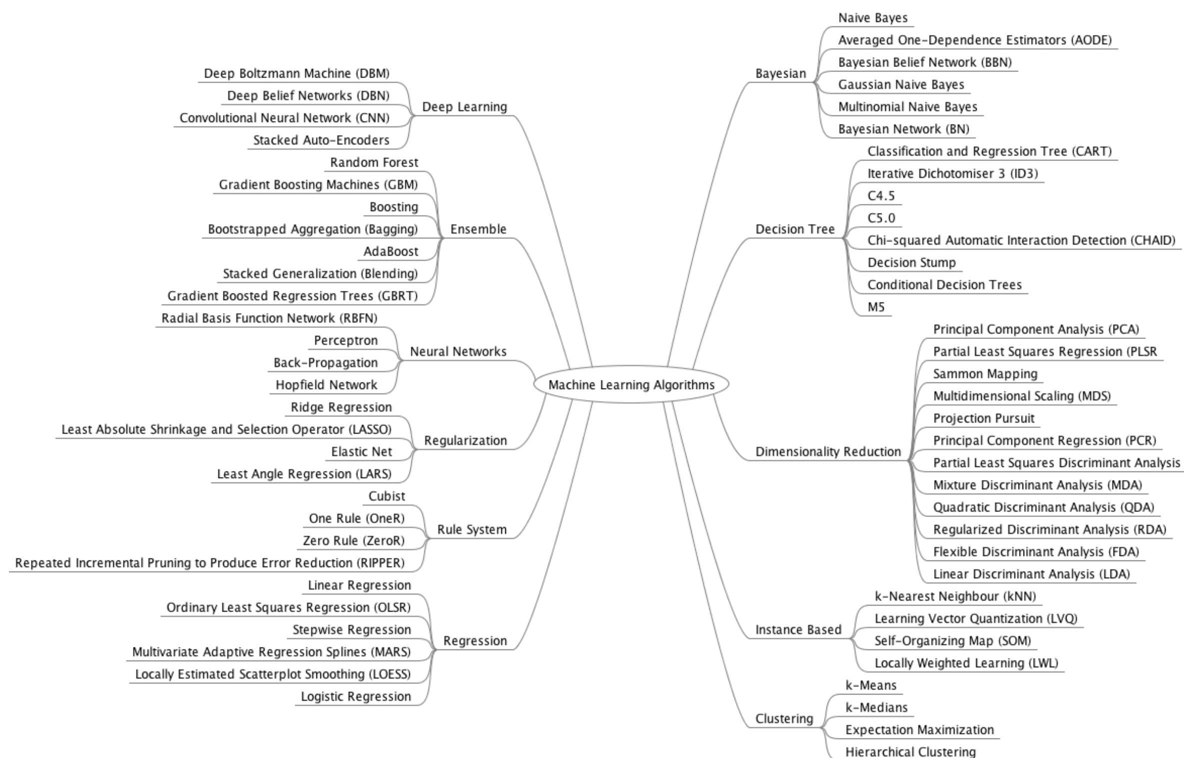


Figura 20: Clasificación de los algoritmos de Machine Learning según Jason Brownlee[32]

En los siguientes epígrafes desarrollaremos algunos de los modelos más básicos sobre el que es desarrollada toda la teoría de los modelos más complejos.

[32] Imagen extraída de: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

9.1.1. Algoritmos de predicción

Tal y como enunciamos anteriormente, este conjunto de algoritmos trata de encontrar cuál es el valor de cierta variable en función de la información recogida en el resto de características.

Este problema suele ser afrontado por los analistas mediante un problema de regresión lineal, procedimientos que detallaremos más en profundidad, mediante procedimientos de regresión no lineal con árboles de decisión, entre los algoritmos más típicos.

Los árboles de predicción los explicaremos en más profundidad en el epígrafe de algoritmos de clasificación, dado que la idea que subyace a dicho procedimiento es muy similar a los árboles de clasificación.

9.1.1.1. Regresión Lineal

Es la forma más simple de regresión lineal y trata de predecir una respuesta cuantitativa de la variable objetivo Y en base a los valores de una única variable predictora X asumiendo que hay una relación lineal aproximada entre ambas.

Dicha relación puede expresarse mediante el siguiente modelo:

$$Y = \beta_0 + \beta_1 X + \varepsilon \qquad \varepsilon \rightarrow N(0, \sigma^2)$$

Donde β_0 y β_1 son dos constantes desconocidas que representan el "*intercept*" o punto donde la recta corta al eje Y y la pendiente (*slope*) en el modelo lineal respectivamente y que ambos se conocen como los coeficientes o parámetros de la ecuación. En el modelo ε representa al error en la obersavación debido a variables no controladas.

Tendríamos por tanto que β_1 representa cuanto se incrementa la respuesta cuando aumenta el valor observado en una unidad la variable X.

Lo que tratamos de encontrar es la recta que mejor ajusta a un conjunto de datos, es decir, la recta con la que menor error global cometeríamos. Por tanto, lo que buscaremos es que

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

el modelo aprenda cuál son los valores de los coeficientes β_0 y β_1 que minimizan el error del conjunto de entrenamiento. O, en otras palabras, queremos hallar el valor de dichos coeficientes que hagan que la línea resultante sea lo más cercana posible a todos los puntos que representan las siguientes observaciones:

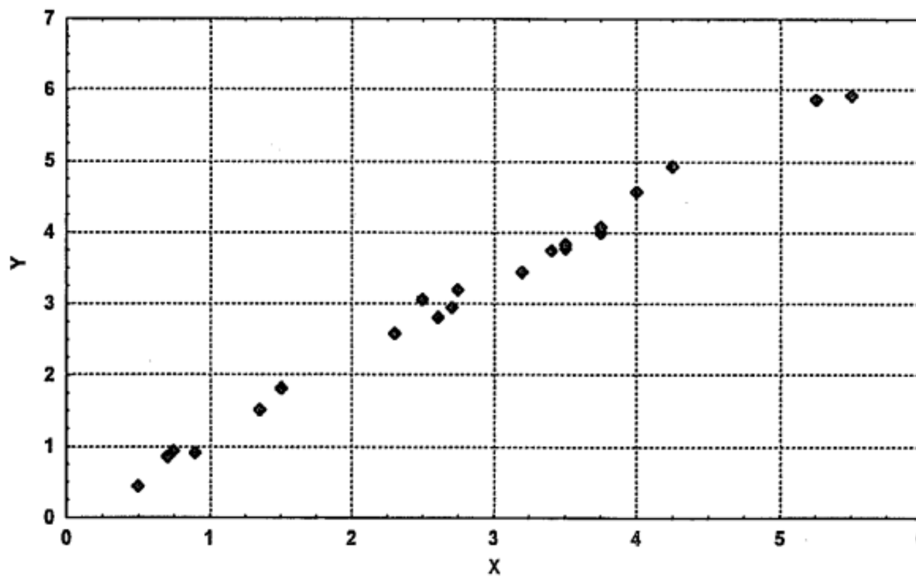


Figura 21:Dos variables con una fuerte dependencia lineal

Estaríamos tratando de ajustar el modelo dentro de un conjunto de observaciones, es decir, dentro de nuestra base de conocimiento compuesta por n observaciones estaríamos asumiendo la siguiente relación:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \qquad i = 1, \dots, n \qquad \varepsilon_i \rightarrow N(0, \sigma^2)$$

Para tener una medida de cómo de bueno es el modelo lineal que se está empleando, podríamos quedarnos con la información de ε_i, es decir, quedarnos con el error cometido por el modelo en cada una de las observaciones. Si pensamos con detenimiento, dicho error estará compuesto por valores tanto positivos como negativos, dado que algunas observaciones habrán quedado por encima de la recta ajusta, mientras que otras quedaron por debajo de ella. Para tener un error del modelo se puede obtener mediante la siguiente ecuación:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$RSS = \varepsilon_1^2 + \varepsilon_2^2 + \cdots + \varepsilon_n^2 = (y_1 - \beta_0 - \beta_1 x_1)^2 + (y_2 - \beta_0 - \beta_1 x_2)^2 + \cdots + (y_n - \beta_0 - \beta_1 x_n)^2$$

La anterior medida representa la suma de cuadrados de los residuos (*residual sum of squares*) o como se conoce más frecuentemente RSS. El valor obtenido en el RSS es empleado para la estimación de la varianza de los residuos mediante la siguiente ecuación:

$$RSE = \sqrt{\frac{RSS}{n-2}}$$

El RSE nos dará una estimación sobre la desviación promedio de cualquier observación respecto a la verdadera recta de regresión, o lo que es lo mismo, estima la desviación estándar de ε.

Al aplicar un modelo de Regresión Lineal, asumen una serie de hipótesis que el analista debe verificar que se cumplen:

1. **Linealidad**: el analista debe comprobar que la relación entre la variable explicativa y la variable respuesta es lineal, si esto no sucede puede ser consecuencia de que la relación entre ambas variables no es lineal o porque variables explicativas relevantes no han sido incluidas en el modelo.

   En muchos casos en el gráfico de la variable respuesta frente a la variable regresora puede verse que la relación no es de tipo lineal. A pesar de ello, el modelo de regresión lineal continúa siendo válido en muchas situaciones porque la relación puede convertirse en lineal por medio de una transformación simple en la variable respuesta Y, o en la variable regresora, X, o en ambas.

   En la búsqueda de la transformación óptima, el procedimiento de Box-Cox fue diseñado para determinar una transformación optima mientras se estima un modelo de regresión lineal. Es muy útil cuando la variabilidad cambia como una función. A menudo, una apropiada transformación estabiliza la variabilidad y produce que las desviaciones alrededor del modelo sean más normalmente distribuidas. Estas transformaciones están propuestas para la variable respuesta, recogidas algunas de ellas en la siguiente tabla:

Transformaciones	Descripción
$Y' = Y$	Datos sin transformar
$Y' = Y^2$	Cuadrado
$Y' = \sqrt{Y}$	Raíz cuadrada
$Y' = \sqrt[3]{Y}$	Raíz cúbica
$Y' = ln(Y)$	Logaritmo
$Y' = \frac{1}{\sqrt{Y}}$	Raíz cuadrada inversa
$Y' = \frac{1}{Y}$	Reciproco

Tabla 3: Transformaciones en la variable respuesta

Podemos encontrar un ejemplo donde no se cumple esta hipótesis en el tercer y cuarto apartado del siguiente enlace: https://gallery.shinyapps.io/slr_diag/

2. **Homogeneidad de varianza**: La hipótesis de homocedasticidad implica que $Var(\varepsilon_i) = \sigma^2 = cte$, se detecta fácilmente en el gráfico de residuos (ε_i) frente a las predicciones $(\hat{y}_i)$. En definitiva, lo que estamos buscando es que la separación de la recta ajustada en todo el espacio sea similar, tal y como se representa en la siguiente imagen:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

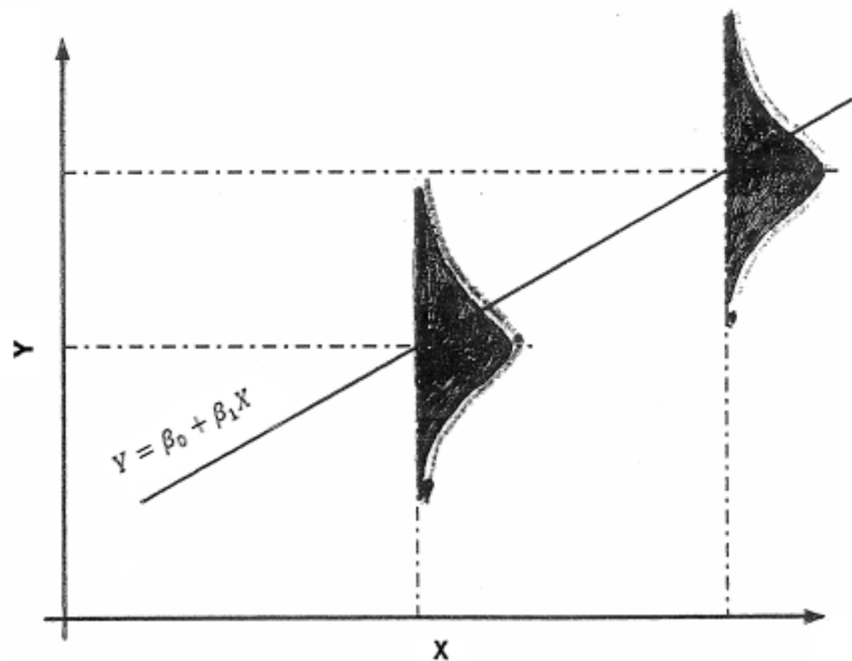


Figura 22: Homogeneidad de varianza

Podemos encontrar un ejemplo donde no se cumple esta hipótesis en el último apartado del siguiente enlace: https://gallery.shinyapps.io/slr_diag/

3. **Normalidad**: Con esta hipótesis lo que se busca encontrar es que los errores del modelo siguen una distribución normal, es preferible trabajar con los residuos estandarizados o estudentizados que tienen la misma varianza, próxima a la unidad.

   Para realizar el contraste de esta hipótesis, podemos llevarlo a cabo mediante procedimientos gráficos o mediante contrastes de normalidad.

   - Gráficos: el gráfico de cajas, el histograma, la estimación no paramétrica de la función de densidad, el gráfico de simetría y los gráfico p-p y q-q.

   - Contrastes de normalidad: contraste de asimetría y curtosis, contraste chi-cuadrado, contraste de Kolmogoroff-Smirnoff o el de Lilliefors.

4. **Independencia**: La hipótesis de que las observaciones muestrales son independientes es una hipótesis básica en el estudio de los modelos de regresión

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

lineal. Con ello se entiende que los errores $\{\varepsilon_i\}_{i=1\ldots n}$ son variables aleatorias independientes.

La falta de independencia, se produce fundamentalmente cuando se trabaja con variables aleatorias que se observan a lo largo del tiempo, esto es, cuando se trabaja con series temporales.

Para que el analista verifique esta hipótesis del modelo puede emplear el contraste de Durbin-Watson, que está diseñado para detectar residuos de un modelo de regresión lineal que tienen un coeficiente de autocorrelación de orden uno distinto de cero. Planteandose por tanto el siguiente contraste de hipótesis:

$$\begin{cases} H_0: \rho(\varepsilon_t, \varepsilon_{t+1}) = 0 \\ H_1: \rho(\varepsilon_t, \varepsilon_{t+1}) \neq 0 \end{cases}$$

El estadístico de contraste de Durbin-Watson es:

$$\hat{d} = \frac{\sum_{i=1}^{n} (\varepsilon_i - \varepsilon_{i-1})^2}{\sum_{i=1}^{n} \varepsilon_i^2}$$

Una vez obtenido el estadístico de contraste $(\hat{d})$, procedemos a buscar dentro de las de dicho contraste el nivel inferior (d_L) y superior (d_U), para nuestro tamaño muestra (n) y el nivel de significación que queramos tener (α). Una vez obtenidos los valores podemos aplicar la siguiente regla:

1. Si $0 < \hat{d} < d_L$ se rechaza H_0 y aceptamos la existencia de autocorrelación positiva.

2. Si $d_L < \hat{d} < d_U$ el contraste no es concluyente.

3. Si $d_U < \hat{d} < 4 - d_U$ se acepta H_0, ésto es, no hay autocorrelación.

4. Si $4 - d_U < \hat{d} < 4 - d_L$ el contraste no es concluyente.

5. Si $4 - d_L < \hat{d} < 4$ se rechaza H_0 y aceptamos la existencia de autocorrelación negativa.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Todo lo que se ha presentado ha sido empleando una única variable de regresión, pero podemos ampliar el modelo para añadir la información recogida en nuevas características, para ello deberemos emplear el modelo de Regresión Lineal General.

En el modelo de Regresión Lineal General se "supone" que la función de regresión que modela la respuesta es lineal. Por tanto, la expresión matemática del modelo de regresión lineal general es:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k + \varepsilon \qquad \varepsilon \rightarrow N(0, \sigma^2)$$

La única diferencia que tendríamos frente al modelo de regresión lineal simple es la introducción de nuevas variables regresoras, con cuya introducción se realiza un mejor ajuste de la respuesta, reduciendo por tanto el error debido a variables que no estaban siendo controladas hasta el momento.

La interpretación de cada uno de los parámetros que componen el modelo es similar al caso de la regresión lineal simple. En el caso de β_i representa cuanto se incrementa la respuesta cuando aumenta el valor observado en una unidad la variable X_i, permaneciendo el resto de variables constantes.

Para calcular el valor de RSE, deberemos de tener en cuenta cuántas son las variables que han sido introducidas en el modelo, en la ecuación anterior habíamos introducido hasta un total de k variables, realizando una estimación del modelo mediante $k + 1$ parametros. La medida del RSE se calculará mediante:

$$RSE = \sqrt{\frac{RSS}{n - k - 1}}$$

Otro de los problemas que deberemos de afrontar a la hora de trabajar con un modelo de Regresión Lineal General será el problema de la multicolinealidad. Es decir, existe una cierta relación entre las variables explicativas lo que hace que los estimadores β_i estén correlacionados. Si está relación es muy fuerte porque dos o más variables regresoras están cercanas a una relación de linealidad.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

En el modelo de Regresión Lineal General se deben todas y cada una de las hipótesis que habíamos enunciado en el modelo de Regresión Lineal:

1. Linealidad
2. Homogeneidad de varianza
3. Normalidad
4. Independencia

Se deberá valorar además si la aportación a la reducción de la variabilidad aportada por cada una de las características, en presencia del resto, compensa en función de la complejidad introducida en el modelo.

En muchas ocasiones, el número de características de las que se dispone información es demasiado elevado, lo que hace que el analista desconozca por donde comenzar a probar con las diferentes características, para saber qué conjunto de ellas podrían modelas el problema al que se está enfrentando. Los procedimientos para descubrir que subconjunto de característica son las que deben ser empleadas se basan en la idea de ir añadiendo características, ir eliminando o una mezcla de ambos:

- Selección hacia delante (*Forward*) en la que el modelo comienza sin predictores y se van añadiendo los más significativos en cada paso.
- Eliminación hacia atrás (*Backward*) en la que el modelo comienza con todos los predictores y se van eliminando las variables menos significativas en cada paso.
- Paso a paso (*Stepwise*), que añade y quita características según necesita en cada paso.

9.1.2. Algoritmos de clasificación

El modelo más sencillo para la variable aleatorio respuesta en términos de los regresores es el modelo de regresión lineal con los errores (ε) siendo variables aleatorias no observables,

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

independientes, con esperanza cero, cuya distribución es la de Bernouilli. Este modelo tiene una serie de limitaciones que nos impiden su utilización:

1. Las probabilidades son valores entre cero y la unidad, mientras que las funciones lineales de variables cuantitativas pueden tomar valores en toda la recta real. Por lo tanto, el modelo de Regresión Lineal puede predecir valores imposibles, valores fuera del intervalo $(0,1)$.

2. No se satisface la condición de homocedasticidad ya que la varianza de la variable respuesta no es constante sobre los valores observados de las variables regresoras.

3. Al no tener la variable respuesta una distribución normal, no se pueden usar las distribuciones muestrales de los estimadores de mínimos cuadrados ordinarios para hacer inferencia sobre el modelo.

4. Es de esperar que los cambios en las variables regresoras tengan menos impacto sobre la probabilidad cuando la respuesta esté próxima a cero o a uno que cuando esté próxima a 0.5, esto no se cumple con el modelo de Regresión Lineal.

Debido a estas dificultades se emplea un modelo no lineal que implique una relación entre las variables regresoras y la probabilidad, de modo que sea curvilínea, monótona, y acotada entre cero y uno. Las funciones de distribución de variables continúas definidas sobre toda la recta real se les puede hacer transformaciones adecuadas que cumplen estos objetivos. Algunas transformaciones clásicas son:

- La función de distribución logística con la que se obtienen los modelos de regresión logística.

- La función de distribución de una normal con la que se tienen los modelos probit.

- La función de distribución de Gumbel reservada para los modelos de valores extremos.

Los modelos de clasificación binaria pueden extenderse al estudio de modelos de clasificación multicategoricos, donde la variable respuesta no toma únicamente valores

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

dicotómicos, sino que puede tomar un conjunto de categorías predefinidas.

Otro modo de afrontar los problemas de clasificación es mediante los árboles de decisión, que también serán visto en este epígrafe.

9.1.2.1. Modelo logístico

Es útil para modelar la probabilidad de un evento ocurriendo como función de otros factores. El análisis de regresión logística se enmarca en el conjunto de Modelos Lineales Generalizados (*GLM*) que usa como función de enlace la función logit.

Los modelos de respuesta binaria presentados anteriormente son un caso especial de Modelos Lineales Generalizados (GLM: Generalized Linear Models) introducidos por Nelder y Wedderburn en 1972 y ampliamente estudiados en el libro de McCullagh y Nelder (McCullagh & Nelder, 1989)[33], y analizados también con profundidad en la publicación de Alan Agresti (Agresti, 2003)[34]

El modelo de regresión logística analiza datos distribuidos binomialmente de la forma:

$$Y_i \rightarrow B(p_i, n_i)$$

donde los números de ensayos Bernoulli n_i son conocidos y las probabilidades de éxito p_i son desconocidas.

Los logits de las probabilidades binomiales desconocidas son modeladas como una función lineal de las características de las que dispongamos información.

$$logit\big(p(x)\big) = \ln\left[\frac{p(x)}{1-p(x)}\right] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

Después de alguna manipulación algebraica llegamos a que:

[33] McCullagh, P., & Nelder, J. A. (1989). *Generalized linear models* (Vol. 37). CRC press.
[34] Agresti, A. (2003). *Categorical data analysis* (Vol. 482). John Wiley & Sons.

$$p(x) = \frac{e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k}}{1+e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k}} = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)}}$$

El interés de este modelo es por tanto ajustar una curva similar a la siguiente que modele el comportamiento de los datos.

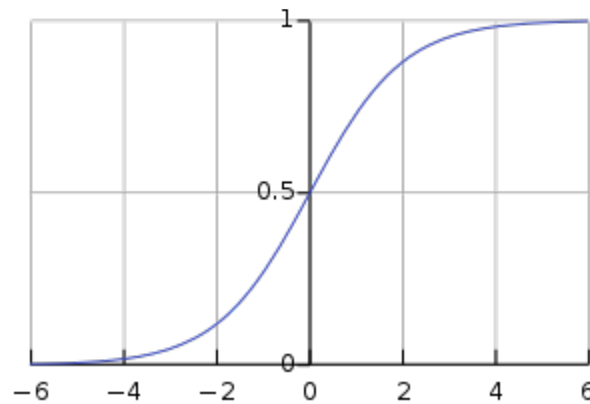


Figura 23: Función logística para una variable regresora[35]

La interpretación de cada uno de los parámetros que componen el modelo es similar al caso de la regresión lineal simple. En el caso de β_i representa cuanto se incrementa la función logit cuando aumenta el valor observado en una unidad la variable X_i, permaneciendo el resto de variables constantes. Si realizamos la exponenciación de la variación obtenida, podremos realizar la interpretación en base al cociente de ventajas (*odd ratio*), que representa cual es la variación en el riesgo de que se presente el fenómeno estudiado con la variación producida en la variable X_i.

9.1.2.1. Árboles de decisión

Los árboles de decisión constituyen una herramienta de aprendizaje supervisado bastante habitual en del aprendizaje automático. Son una técnica muy versátil que puede utilizarse en áreas de muy diversa índole.

[35] Imagen extraída de: https://es.wikipedia.org/wiki/Regresi%C3%B3n_log%C3%ADstica

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Dentro del aprendizaje automático un árbol de decisión es un modelo predictivo que puede ser utilizado para tareas de clasificación o para tareas de regresión, según la naturaleza de la variable de la clase: discreta (árbol de decisión) o continua (árbol de regresión).

Un árbol de decisión es una forma de representar el conocimiento obtenido en el proceso de aprendizaje inductivo. Puede verse como un conjunto de condiciones organizadas en una estructura jerárquica en forma de árbol, formada por diferentes nodos que se conectan con arcos dirigidos, donde existen tres tipos de nodos:

- Nodo raíz: Es el nodo inicial, sólo tiene ramas salientes y en él queda recogida la totalidad de la población o datos de estudio.
- Nodos hijos: Poseen ramas entrantes que provienen de los nodos padre y ramas salientes que apuntan a los nodos hijo.
- Nodo hoja: Representan la partición final, sólo tienen ramas entrantes y se asocia con una etiqueta o valor que caracteriza a los datos que llegan al nodo.

Así, dentro del árbol, cada nodo representa una característica, y cada rama representa un estado de esa variable.

La generación de la estructura de un árbol se fundamenta en el principio de "divide y vencerás". A partir del conjunto completo de datos y dado un criterio de partición determinado, se divide el conjunto en subconjuntos cada vez más pequeños. Recursivamente se va dividiendo cada uno de los subconjuntos creados hasta que todos ellos son puros o hasta que su "pureza" no puede incrementarse, formándose así los nodos hoja del árbol. Y finalmente, a los nodos hoja se les asigna una etiqueta o valor determinado de la variable clase.

Si no se establece ningún límite, la construcción del árbol se detiene cuando el nodo es puro. Sin embargo, este criterio puede dar lugar a un sobreajuste de los datos, que reduce la aplicabilidad del modelo de clasificación aprendido, siendo el modelo construido muy específico, poco general, y por tanto malo para otros conjuntos de datos. Pero, además, si

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

los datos contienen errores en los atributos el modelo intentará ajustarse a los errores, perjudicando el comportamiento global del modelo aprendido.

La mayoría de los **criterios de partición** tratan de maximizar la bondad de la partición, lo cual equivale a minimizar la bondad de la impureza del árbol generado por la partición. Algunos ejemplos son el índice de Gini, la ganancia de información o la razón de ganancia.

Las **reglas de parada** tratan de predecir si conviene seguir construyendo el árbol por una determinada rama o no. Algunas de las condiciones más comúnmente utilizadas como criterios de parada son las siguientes:

- Grado de pureza del nodo.
- Cota de profundidad.
- Umbral de soporte.

Los métodos de **poda** permiten realizar una simplificación del árbol, permitiendo mejorar tanto la precisión del método como la capacidad predictiva, evitando el sobreajuste. Algunos de los algoritmos más comunes son:

- Poda por estimación del error.
- Poda por coste-complejidad.
- Poda pesimista.

La clasificación de una nueva instancia se realiza recorriendo el árbol, empezando por el nodo raíz y siguiendo el camino determinado por cada uno de los arcos en función de la información disponible en la instancia evaluada, hasta llegar a un nodo hoja. La etiqueta asignada a esta hoja es la que se asignará a la instancia a clasificar.

Ventajas	Desventajas
Son auto-explicativos, y cuando su tamaño no	El exceso de sensibilidad sobre el

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

es muy grande son muy fáciles de interpretar. Normalmente son un método no paramétrico por lo que no establecen hipótesis sobre su distribución espacial ni de la estructura del clasificador. Pueden trabajar con un gran número de variables predictivas sin problemas de multicolinealidad. Pueden descubrir fácilmente complejas interacciones entre las instancias de una gran base de datos.	conjunto de entrenamiento. No permiten examinar los efectos marginales de las características objeto de estudio. Dan lugar a pérdida de información, dado que solo se emplean las características que comenzaron en el nodo raíz.

Tabla 4: Principales ventajas y desventajas del uso de árboles

Algunos de los algoritmos de árboles más empleados son:

- CART
- C4.5
- C5.0
- PART
- Random Forest
- Gradient Boosted Machines (GBM)
- ID3

Describiremos cuál es el funcionamiento del algoritmo ID3, comprendiendo como construye un árbol para representar el conjunto de datos observados. El árbol es construido siguiendo las siguientes reglas.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

1. Seleccionar el atributo A que maximice la ganancia $G(A_i)$

2. Crear un nodo para ese atributo con tantos hijos como valores categorías tenga.

3. Introducir los ejemplos en los sucesores según el valor que tenga el atributo A_i.

4. Por cada sucesor:

    a. Si sólo hay ejemplos de una clase, C_k, entonces etiquetarlo con C_k.

    b. Si no, llamar a ID3 con una tabla formada por los ejemplos de ese nodo, eliminado la columna del atributo A.

La fórmula empleada para calcular la ganancia de cada uno de los atributos es:

$$G(S,A) = H(S) - \sum_{i=0}^{k} \frac{|S_{v_i}|}{S} H(S_{v_i})$$

donde $|S_{v_i}|$ = instancias con $A = v_i$

$$H(S) = \sum_{i=0}^{n} P(C_i) I(C_i)$$

$$I(s_i) = \log_2\left(\frac{1}{P(s_i)}\right)$$

Una vez conocida la definición del algoritmo, realizaremos todos los cálculos necesarios para la obtención del árbol completo del siguiente conjunto de datos:

Vista	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	No	No
Soleado	Alta	Alta	Si	No

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Nublado	Alta	Alta	No	Si
Lluvioso	Media	Alta	No	Si
Lluvioso	Baja	Normal	No	Si
Lluvioso	Baja	Normal	Si	No
Nublado	Baja	Normal	Si	Si
Soleado	Media	Alta	No	No
Soleado	Baja	Normal	No	Si
Lluvioso	Media	Normal	No	Si
Soleado	Media	Normal	Si	Si
Nublado	Media	Alta	Si	Si
Nublado	Alta	Normal	No	Si
Lluvioso	Media	Alta	Si	No

Tabla 5: Ejemplo de clasificación con datos partido de tenis

Comenzaremos calculando el atributo que mayor ganancia proporciona:

$$Inercia: E(9+, 5-) = \frac{9}{14}\log_2\left(\frac{14}{9}\right) + \frac{5}{14}\log_2\left(\frac{14}{5}\right) = 0.9403$$

Vista:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$Vista: Soleado = E(2+, 3-) = \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{3}{5}\log_2\left(\frac{5}{3}\right) = 0.971$$

$$Vista: Nublado = E(4+, 0-) = \frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\left(\frac{4}{0}\right) = 0$$

$$Vista: Lluvioso = E(3+, 2-) = \frac{3}{5}\log_2\left(\frac{5}{3}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right) = 0.971$$

$$G(S, Vista) = 0.9403 - \frac{5}{14}\cdot 0.971 - \frac{4}{14}\cdot 0 - \frac{5}{14}\cdot 0.971 = 0.2467$$

Temperatura:

$$Temperatura: Alta = E(2+, 2-) = \frac{2}{4}\log_2\left(\frac{4}{2}\right) + \frac{2}{4}\log_2\left(\frac{4}{2}\right) = 1$$

$$Temperatura: Media = E(4+, 2-) = \frac{4}{6}\log_2\left(\frac{6}{4}\right) + \frac{2}{6}\log_2\left(\frac{6}{2}\right) = 0.9183$$

$$Temperatura: Baja = E(3+, 1-) = \frac{3}{4}\log_2\left(\frac{4}{3}\right) + \frac{1}{4}\log_2\left(\frac{4}{1}\right) = 0.8113$$

$$G(S, Temperatura) = 0.9403 - \frac{4}{14}\cdot 1 - \frac{6}{14}\cdot 0.9183 - \frac{4}{14}\cdot 0.8113 = 0.0292$$

Humedad:

$$Humedad: Alta = E(3+, 4-) = \frac{3}{7}\log_2\left(\frac{7}{3}\right) + \frac{4}{7}\log_2\left(\frac{7}{4}\right) = 0.9852$$

$$Humedad: Normal = E(6+, 1-) = \frac{6}{7}\log_2\left(\frac{7}{6}\right) + \frac{1}{7}\log_2\left(\frac{7}{1}\right) = 0.5917$$

$$G(S, Humedad) = 0.9403 - \frac{7}{14}\cdot 0.9852 - \frac{7}{14}\cdot 0.5917 = 0.1518$$

Viento:

$$Viento: No = E(6+, 2-) = \frac{6}{8}\log_2\left(\frac{8}{6}\right) + \frac{2}{8}\log_2\left(\frac{8}{2}\right) = 0.8113$$

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$Viento: Si = E(3+,3-) = \frac{3}{6}\log_2\left(\frac{6}{3}\right) + \frac{3}{6}\log_2\left(\frac{6}{3}\right) = 1$$

$$G(S, Viento) = 0.9403 - \frac{8}{14} \cdot 0.8113 - \frac{6}{14} \cdot 1 = 0.1518$$

Una vez que disponemos de la ganancia que proporcionan cada uno de los atributos, seleccionamos aquel que mayor ganancia obtiene. Para el conjunto de datos, el atributo que es seleccionado en primer lugar es **Vista**.

Una vez seleccionado el atributo **Vista** se nos presentan tantos escenarios posibles como categorías tiene esta variable.

1. Nublado: Generamos un nodo hijo final etiquetado con la categoría Jugar: Si.
2. Lluvioso: Generamos un nodo hijo con las instancias que tienen esta categoría y buscamos nuevamente el atributo que maximice la ganancia. Las instancias con las que trabajaremos son:

Vista	Temperatura	Humedad	Viento	Jugar
Lluvioso	Media	Alta	No	Si
Lluvioso	Baja	Normal	No	Si
Lluvioso	Baja	Normal	Si	No
Lluvioso	Media	Normal	No	Si
Lluvioso	Media	Alta	Si	No

$$Inercia: E(3+,2-) = \frac{3}{5}\log_2\left(\frac{5}{3}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right) = 0.971$$

Temperatura:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$Temperatura: Media = E(2+, 1-) = \frac{2}{3}\log_2\left(\frac{3}{2}\right) + \frac{1}{3}\log_2\left(\frac{3}{1}\right) = 0.9183$$

$$Temperatura: Baja = E(1+, 1-) = \frac{1}{2}\log_2\left(\frac{2}{1}\right) + \frac{1}{2}\log_2\left(\frac{2}{1}\right) = 1$$

$$G(S, Temperatura) = 0.971 - \frac{3}{5} \cdot 0.9183 - \frac{2}{5} \cdot 1 = 0.02$$

Humedad:

$$Humedad: Alta = E(1+, 1-) = \frac{1}{2}\log_2\left(\frac{2}{1}\right) + \frac{1}{2}\log_2\left(\frac{2}{1}\right) = 1$$

$$Humedad: Normal = E(2+, 1-) = \frac{2}{3}\log_2\left(\frac{3}{2}\right) + \frac{1}{3}\log_2\left(\frac{3}{1}\right) = 0.9183$$

$$G(S, Humedad) = 0.971 - \frac{2}{5} \cdot 1 - \frac{3}{5} \cdot 0.9183 = 0.02$$

Viento:

$$Viento: No = E(3+, 0-) = \frac{3}{3}\log_2\left(\frac{3}{3}\right) + \frac{0}{3}\log_2\left(\frac{3}{0}\right) = 0$$

$$Viento: Si = E(0+, 2-) = \frac{0}{2}\log_2\left(\frac{2}{0}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right) = 0$$

$$G(S, Viento) = 0.971 - \frac{3}{5} \cdot 0 - \frac{2}{5} \cdot 0 = 0.971$$

Una vez que disponemos de la ganancia que proporcionan cada uno de los atributos, seleccionamos aquel que mayor ganancia obtiene. Para el subconjunto de datos, el atributo que es seleccionado es **Viento**.

Una vez seleccionado el atributo **Viento** se nos presentan tantos escenarios posibles como categorías tiene esta variable.

a. No: Generamos un nodo hijo final etiquetado con la categoría Jugar: Si.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

b. Si: Generamos un nodo hijo final etiquetado con la categoría Jugar: No.

3. Soleado: Generamos un nodo hijo con las instancias que tienen esta categoría y buscamos nuevamente el atributo que maximice la ganancia. Las instancias con las que trabajaremos son:

Vista	Temperatura	Humedad	Viento	Jugar
Soleado	Alta	Alta	No	No
Soleado	Alta	Alta	Si	No
Soleado	Media	Alta	No	No
Soleado	Baja	Normal	No	Si
Soleado	Media	Normal	Si	Si

$$Inercia: E(2+,3-) = \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{3}{5}\log_2\left(\frac{5}{3}\right) = 0.971$$

Temperatura:

$$Temperatura: Alta = E(0+,2-) = \frac{0}{2}\log_2\left(\frac{2}{0}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right) = 0$$

$$Temperatura: Media = E(1+,1-) = \frac{1}{2}\log_2\left(\frac{2}{1}\right) + \frac{1}{2}\log_2\left(\frac{2}{1}\right) = 1$$

$$Temperatura: Baja = E(1+,0-) = \frac{1}{1}\log_2\left(\frac{1}{1}\right) + \frac{0}{1}\log_2\left(\frac{1}{0}\right) = 0.9183$$

$$G(S, Temperatura) = 0.971 - \frac{2}{5}\cdot 0 - \frac{2}{5}\cdot 1 - \frac{1}{5}\cdot 0 = 0.571$$

Humedad:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$Humedad: Alta = E(0+,3-) = \frac{0}{3}\log_2\left(\frac{3}{0}\right) + \frac{3}{3}\log_2\left(\frac{3}{3}\right) = 0$$

$$Humedad: Normal = E(2+,0-) = \frac{2}{2}\log_2\left(\frac{2}{2}\right) + \frac{0}{2}\log_2\left(\frac{2}{0}\right) = 0$$

$$G(S, Humedad) = 0.971 - \frac{3}{5}\cdot 0 - \frac{2}{5}\cdot 0 = 0.971$$

Viento:

$$Viento: No = E(1+,2-) = \frac{1}{3}\log_2\left(\frac{3}{1}\right) + \frac{2}{3}\log_2\left(\frac{3}{2}\right) = 0.9183$$

$$Viento: Si = E(1+,1-) = \frac{1}{2}\log_2\left(\frac{2}{1}\right) + \frac{1}{2}\log_2\left(\frac{2}{1}\right) = 1$$

$$G(S, Viento) = 0.971 - \frac{3}{5}\cdot 0.9183 - \frac{2}{5}\cdot 1 = 0.02$$

Una vez que disponemos de la ganancia que proporcionan cada uno de los atributos, seleccionamos aquel que mayor ganancia obtiene. Para el subconjunto de datos, el atributo que es seleccionado es **Humedad**.

Una vez seleccionado el atributo **Humedad** se nos presentan tantos escenarios posibles como categorías tiene esta variable.

a. Normal: Generamos un nodo hijo final etiquetado con la categoría Jugar: Si.

b. Alta: Generamos un nodo hijo final etiquetado con la categoría Jugar: No.

Una vez que el algoritmo solo se encuentra en nodos hijos finales, el algoritmo ha encontrado la solución. De un modo gráfico, la solución que hemos estado desarrollando con los cálculos anteriores, sería la que se muestra en la siguiente imagen.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

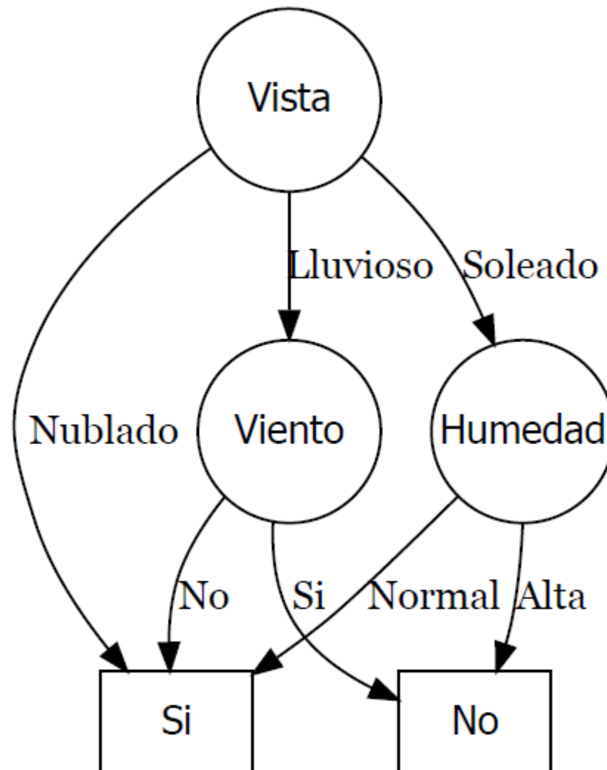


Figura 24: Árbol generado por ID3

9.1.3. Evaluación de modelos

En este apartado vamos a ver un conjunto de técnicas que se emplean para medir la calidad de los modelos planteados por el analista, algunas de ellas fueron introducidas en el apartado 8.3.

En primer lugar, veremos cómo asegurarnos de que el modelo tendrá un buen resultado ajustándose a un conjunto de entrenamiento, pero sin que haya un problema de sobreentrenamiento (overfit) que le haga funcionar deficientemente con otros conjuntos de datos. Aquí veremos las técnicas de remuestreo (resampling).

Por otra parte, existen métricas, dependiendo del tipo de modelo, que se aplican para ver la efectividad del modelo y compararlo con otros modelos, ya sean del mismo tipo, pero con distintos parámetros o de distinto tipo.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

9.1.3.1. Métodos de remuestreo

La razón de ser de los métodos de remuestreo es aplicar repetidamente diferentes conjuntos de entrenamiento y validación con el objetivo de obtener información adicional de como de bien puede ajustarse un modelo. Normalmente se usa para estudiar la variabilidad de un modelo dependiendo del conjunto de instancias que son empleadas para su entrenamiento o evaluación hacen variar los resultados del mismo.

A continuación, vamos a ver los métodos de remuestreo más utilizados a la hora de aplicar modelos de aprendizaje automático.

Particionado de los datos

Uno de los procedimientos más extendidos es la segmentación del conjunto de datos en dos subconjuntos de instancias, uno dedicado para los entrenamientos de los diferentes modelos y el otro empleado para la evaluación y comparación de los mismos.

Este procedimiento es útil cuando el conjunto de datos de entrada es muy grande y el conjunto dedicado para la evaluación de los modelos puede proporcionar una estimación del rendimiento de forma significativa, o cuando se tiene alguna limitación de recursos y te es suficiente con una estimación aproximada de la precisión del modelo.

El problema de este procedimiento radica en que, si entrenamos el modelo y lo evaluamos con distintas particiones del mismo conjunto de datos, seguramente obtengamos resultados distintos, poniendo de manifiesto de esta manera que el conjunto de instancias que son seleccionadas a la hora de entrenar y comparar los modelos influyen fuertemente en la decisión final que se tome. Para disminuir este problema se emplean los procedimientos de validación cruzada.

Validación cruzada (cross validation)

Permite asegurar que cada observación de un conjunto se usa para el entrenamiento y validación del modelo un igual número de veces, reduciendo así la influencia de las observaciones atípicas a la hora de estimar los diferentes parámetros que componen el

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

modelo, obteniendo por tanto al modelo más cercano a la realidad del conjunto de instancias.

Hay varios tipos de validación cruzada, los más utilizados se explican a continuación:

- **Validación cruzada con k particiones**[36] (*k-fold cross validation*), en la que se parte el conjunto de datos original en k particiones (por ejemplo 10) y se ejecuta el algoritmo k (10) veces. Cada vez que se ejecuta el modelo se tomará el 90% del conjunto de instancias originales (9 de 10 particiones) para entrenarlo y el 10% (1 de 10 particiones para hacer la validación) y se irá cambiando la partición empleada para la evaluación de los modelos de modo que sea distinta en cada una de las ejecuciones. Al finalizar el procedimiento, todas y cada una de las particiones habrán sido empleadas en una ocasión para la validación del modelo. En la siguiente figura se representa el procedimiento cuando el número de particiones es igual a cinco, teniéndose que denotar que el procedimiento en verdad no trabajará con particiones de observaciones consecutivas.

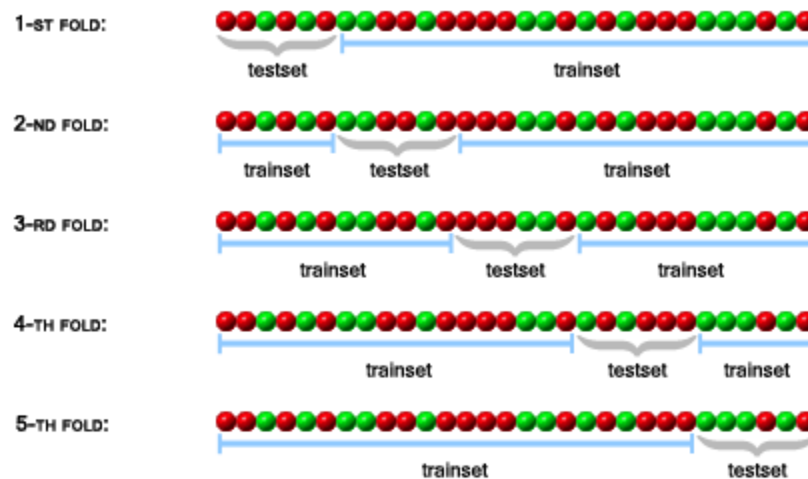


Figura 25 Validación cruzada con 5 particiones[37]

Este tipo de validación es el mejor método de evaluar el rendimiento de un

[36] Más información en: https://en.wikipedia.org/wiki/Cross-validation_(statistics)
[37] Imagen extraída de: http://genome.tugraz.at/proclassify/help/pages/XV.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

algoritmo en un conjunto de datos determinado, pero no aplica también al cálculo de la varianza en las predicciones del algoritmo. Esto quiere decir que si lanzamos el algoritmo con diferentes semillas aleatorias las k particiones que tendíamos serían distintas y también por tanto los resultados que obtendríamos, hecho al que no es sensible este método de validación.

- **Validación cruzada con k particiones repetida** (*repeated k-fold cross validation*). Para poder tener en cuenta la varianza en el algoritmo en este tipo de validación se ejecuta varias veces el procedimiento de validación cruzada con k particiones y se toma la mediana y la desviación estándar de la precisión del algoritmo de cada una de las ejecuciones.

  De esta forma se obtiene una estimación del rendimiento del algoritmo con un conjunto de datos determinado y de como de robusto es a la varianza.

- **Validación cruzada dejando uno fuera** (*Leave One Out Cross Validation*) LOOCV. En este caso se deja una observación fuera y se construye el modelo con el resto de observaciones. El proceso se repite con todas las observaciones.

  Una versión de este procedimiento, en vez de dejar únicamente una instancia, deja fuera un conjunto de instancias, siguiendo la idea de los procedimientos de particionado.

- **Bootstrap**[38]. En este caso se toman muestreos aleatorios del conjunto de datos con reemplazamiento, y se ejecuta el modelo múltiples veces (generalmente miles) para evaluarlo. De forma agregada los resultados proporcionan una indicación de la varianza del rendimiento del modelo.

9.1.3.2. Métricas de evaluación de modelos

A continuación, veremos algunas de las métricas clásicas empleadas a la hora de

[38] Más información en: https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

comparar múltiples **modelos de regresión**:

- RSE (*Residual Estándar Error*). Viene dado por la fórmula:

$$RSE = \sqrt{\frac{RSS}{n-p-1}}$$

donde n es el número de observaciones y p el número de variables predictoras, y donde $RSS = \varepsilon_1^2 + \varepsilon_2^2 + \cdots + \varepsilon_n^2$.

El modelo será mejor cuanto más pequeño sea RSE, dado que menor error se estará cometiendo.

- R^2: El coeficiente de determinación mide la cantidad de variabilidad explicada por el modelo. Viene dado por la fórmula:

$$R^2 = 1 - \frac{RSS}{TSS}$$

donde TSS es $\sum_{i=1}^{n}(y_i - \bar{y})^2$.

Por tanto, cuanto menor sea RSS, mayor TSS y R^2 estará cerca de 1, siendo el modelo mejor.

Deberemos tener en cuenta que esta métrica no tiene presente el sobreajuste del modelo, dado que no premia la simplicidad del modelo. Un modelo igual número de variables que el número de instancias tendrá el valor máximo en esta métrica, pero se encontrará muy próximo al conjunto de datos empleados y lejano a la realidad que se pretende modelar. Por tanto tendríamos que R^2 tiende a estimar de forma optimista el ajuste de la regresión lineal, aumentando con el número de variables que se incluyen en el modelo.

- R_{adj}^2: El coeficiente de determinación corregido mide la cantidad de variabilidad explicada por el modelo, pero penalizando la complejidad del número de parámetros del mismo. Viene dado por la fórmula:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$R_{adj}^2 = 1 - \frac{n-1}{n-p-1}[1-R^2]$$

La interpretación de esta métrica sería idéntica al coeficiente de determinación, pero teniendo presente que $0 \leq R_{adj}^2 \leq R^2 \leq 1$.

- **Test F**: Contraste empleado para ver la significación del modelo lineal planteado, es decir, ver si el conjunto de parámetros empleados describe el problema frente a no emplear ninguno de ellos, estaríamos por tanto planteándonos la siguiente hipótesis:

$$\begin{cases} H_0: \beta_1 = \beta_2 = \cdots = \beta_p = 0 \\ H_1: \beta_1 \neq \beta_2 \neq \cdots \neq \beta_p \neq 0 \end{cases}$$

El estadístico de contraste sería:

$$\hat{F} = \frac{\frac{\sum_{i=1}^{n}\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p - \bar{y}\right)^2}{p-1}}{\frac{\sum_{i=1}^{n} {\varepsilon_i}^2}{n-p}}$$

Una vez obtenido el estadístico de contraste $(\hat{F})$, procedemos a comparar el valor obtenido con una distribución F de Snedecor, también conocida con el nombre de distribución F de Fisher-Snedecor $F_{p-1,n-p}$.

Este mismo contraste podría ser empleado para la comparación de dos modelos que tienen parámetros incrementales, es decir, plantearnos si la inclusión de cierto número de parámetros merece la pena frente a un modelo más simple que careciera de ellos. La hipótesis que nos estaríamos formulando sería:

$$\begin{cases} H_0: \beta_1 = \beta_2 = \cdots = \beta_m = 0;\ \beta_{m+1} = \beta_{m+2} = \cdots = \beta_p = 0 \\ H_1: \beta_1 \neq \beta_2 \neq \cdots \neq \beta_p \neq 0 \end{cases}$$

donde m representa el subconjunto de parámetros que se quieren considerar como nulos, es decir, que la complejidad que aportan no merece la pena:

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$\hat{F} = \frac{\frac{\sum_{i=1}^{n}\left(\beta_0 + \beta_{m+1}x_{m+1} + \cdots + \beta_p x_p - y\right)^2}{p-m}}{\frac{\sum_{i=1}^{n}{\varepsilon_i}^2}{n-p}}$$

Una vez obtenido el estadístico de contraste $(\hat{F})$, procedemos a comparar el valor obtenido con una distribución F de Snedecor $F_{p-m,n-p}$.

Este último contraste podríamos plantearnos realizarlo con la inclusión o supresión de todos y cada uno de los parámetros del modelo, es decir, si en presencia del resto de parámetros la información que aporta cada uno de ellos merece la pena para su conservación.

A continuación, veremos algunas de las métricas clásicas empleadas a la hora de comparar múltiples **modelos de clasificación**:

- BIC: el criterio de información bayesiano es empleado para la comparación de modelos, introduce un término de penalización para ponderar de algún modo la complejidad de parámetros que componen el modelo.

  $$BIC = -2\ln(\hat{L}) + p\ln n$$

  donde $\hat{L}$ es el máximo valor de la función de verosimilitud del modelo.

  Siguiendo este criterio, el modelo que deberíamos de seleccionar sería el que menor BIC hubiera obtenido.

- AIC: el criterio de información de Akaike es empleado para la comparación de modelos, introduce un término de penalización para ponderar de algún modo la complejidad de parámetros que componen el modelo.

  $$AIC = 2p - 2\ln(\hat{L})$$

  Siguiendo este criterio, el modelo que deberíamos de seleccionar sería el que menor AIC hubiera obtenido.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- AIC_c: Se trata de una corrección del criterio de Akaike para tener presente el volumen y complejidad del modelo.

$$AIC_c = AIC + \frac{2p^2 + 2p}{n - p - 1}$$

Siguiendo este criterio, el modelo que deberíamos de seleccionar sería el que menor AIC_c hubiera obtenido.

- $R^2_{McFadden}$:

$$R^2_{McFadden} = 1 - \frac{\ln(\hat{L})}{\ln(\hat{L}_{null})}$$

donde $\hat{L}_{null}$ es el valor de la función de verosimilitud del modelo únicamente con intercept.

Siguiendo este criterio, el modelo que deberíamos de seleccionar sería el que mayor $R^2_{McFadden}$ hubiera obtenido.

- **Matriz de confusión**: No es más que una tabla en la que se colocan en las filas el valor de las observaciones reales y en las columnas las predicciones del modelo con el fin de compararlas.

		Predicción	
		+	-
Clase real	+	TP: Verdadero positivo	FN: Falso negativo
	-	FP: Falso positivo	TN: Verdadero negativo

Tabla 6: Matriz de confusión binaria

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

A partir de la tabla se definen diferentes medidas que no dependen de la prevalencia de la clase positiva:

- Habilidad para administrar tratamiento cuando el modelo detecta un positivo:

$$precisión = \frac{TP}{TP + FP}$$

- Habilidad para detectar un positivo cuando está realmente presente:

$$sensibilidad: Se = \frac{TP}{TP + FN}$$

- Habilidad para excluir un negativo cuando no está presente:

$$especificidad: Sp = \frac{TN}{TN + FP}$$

También podemos definir una serie de parámetros que dependen de la prevalencia de la clase positiva:

- Valor predictivo positivo:

$$VPP = \frac{pSe}{pSe + (1 - p)(1 - Sp)}$$

- Valor predictivo negativo:

$$VPN = \frac{(1 - p)Sp}{(1 - p)Sp + p(1 - Se)}$$

En la siguiente imagen podemos hacernos una idea de cuál es el valor de ambas medidas cuando la $sensibilidad = 0.7$ y la $especificidad = 0.9$.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

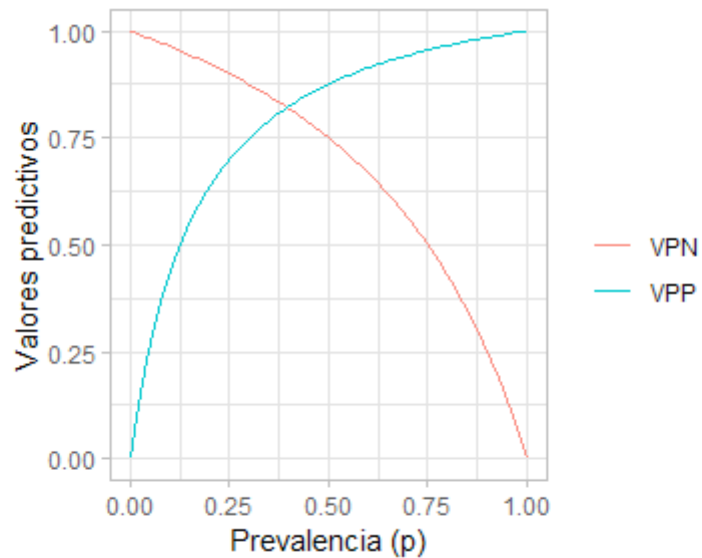


Figura 26: Valores predictivos a diferentes niveles de prevalencia

- **Curva ROC**: Es una curva en la que se representa la ratio de verdaderos positivos frente a los falsos positivos de un clasificador, tal y como se puede apreciar en la siguiente figura:

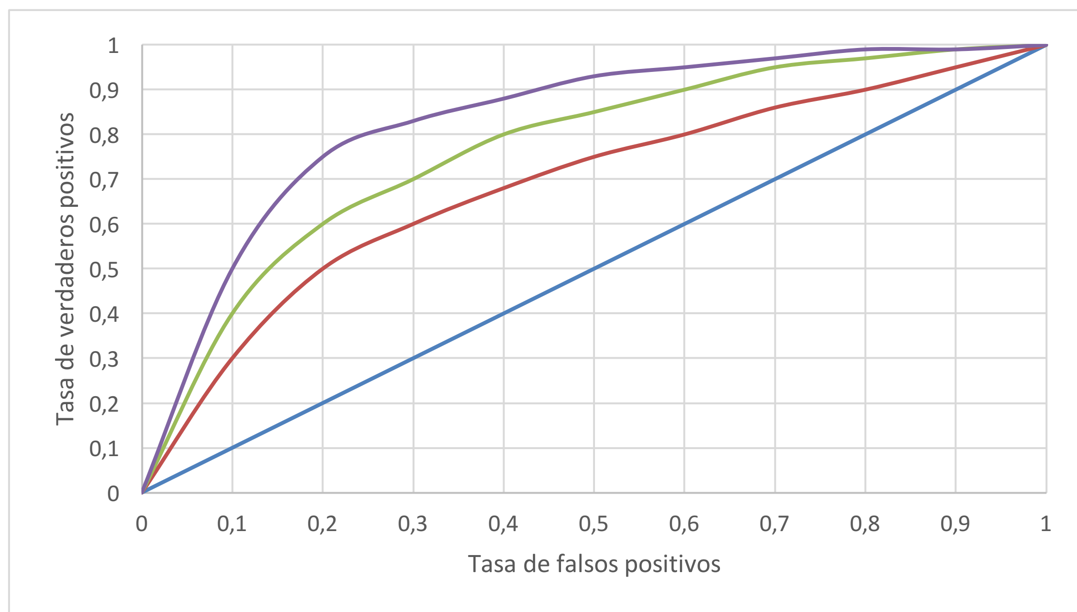


Figura 27: Curvas ROC de varios modelos

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

De todos los modelos anteriores, con el que mejores resultados se esperarían sería con el que se ha construido la curva morada, dado que a medida que nos movemos hacia la izquierda y arriba el clasificador va siendo mejor, pues con menos tasa de desacierto somos capaces de obtener mucho acierto.

De la curva ROC se puede obtener la métrica **AUC** que representaría el área bajo la curva y no serviría para comparar diferentes modelos sin necesidad de pintar sus curvas ROC.

9.2. Aprendizaje no supervisado

Todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos. Por lo tanto, en este caso, el sistema tiene que ser capaz de reconocer patrones para poder etiquetar las nuevas entradas.

Con relación al aprendizaje no supervisado vamos a estudiar aquí tres de los modelos más importantes que utilizamos nosotros en nuestro trabajo en Telefónica:

- **Reglas de Asociación (*Association Rules o AR*)**: Algoritmos de aprendizaje de reglas de aislamiento. Los métodos de aprendizaje de reglas de asociación extraen las reglas que mejor explican las relaciones observadas entre las variables en los datos.

  Estas reglas pueden descubrir asociaciones importantes y comercialmente útiles en grandes conjuntos de datos multidimensionales que pueden ser explotados por una organización. Los algoritmos de aprendizaje de reglas de asociación más populares son:

  - Algoritmo Apriori
  - Algoritmo Eclat

- **Reducción de la dimensionalidad**: Estos métodos de reducción de la dimensionalidad explotan la estructura interna de los datos para resumir o describir

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

dichos datos usando menos información siendo así muy útiles para visualizar datos con mayor grado de dimensión o simplificar los datos de entrada para otros modelos de aprendizaje supervisado principalmente.

Los algoritmos más utilizados relacionados con este tipo de técnicas son:

- Principal Component Analysis (PCA)
- Principal Component Regression (PCR)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Linear Discriminant Analysis (LDA)
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Flexible Discriminant Analysis (FDA)

- **Agrupamiento (*Clustering*)**: Los métodos de agrupación en clúster suelen organizarse según los enfoques de modelado, como los basados en centroides y jerárquicos. Todos los métodos están relacionados con el uso de las estructuras inherentes en los datos para organizar mejor los datos en grupos de máxima similitud.

  Los algoritmos de agrupamiento más populares son:

  - k-Means
  - k-Medians

- Expectation Maximisation (EM)
- Hierarchical Clustering

9.2.1. Algoritmos de agrupación

Los algoritmos de agrupación, tienen la tarea de crear grupos de instancias, las cuales sean similares y siendo las instancias que componen diferentes grupos lo más disimilares posibles. En la literatura, los grupos que se extraen de estos procedimientos se llaman clústeres.

Tendremos, por tanto, que los procedimientos de clustering son una actividad de clasificación no supervisada, pues no necesita que las observaciones estén etiquetadas como clases previamente, y la salida de dichos procedimientos serán las agrupaciones que estos algoritmos hayan encontrado.

Hay que poner en alta estima estos procedimientos, dado que son lo que hacen que podamos comprender de manera global las características de diferentes individuos, que en un principio se creían muy diferentes, pero que en la realidad son similares. Ayudando a los equipos de visualización en su tarea de presentación de conclusiones.

Todos los algoritmos de agrupación se sustentan sobre la idea de una matriz que nos defina cómo de diferentes o cómo de iguales son dos instancias. La matriz de distancias, en muchas ocasiones, es algo complejo de decidir cómo se construye, dado que el criterio que se emplee influirá en los resultados que finalmente arroje el algoritmo.

Las técnicas de clustering se suelen utilizar de forma independiente para extraer conocimiento de la distribución de los datos o como una fase de pre-procesado para otros algoritmos de Machine Learning.

Algunos de los procedimientos empleados para la construcción de la matriz de similitud o desemejanza entre cada par de instancias. Esta matriz, en muchos procedimientos cumple las siguientes desigualdades:

- $d(i,j) \geq 0$, la distancia entre un par de individuos debe de ser mayor o igual que

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

cero. Esta característica es conocida como semipositividad.

- $d(i,i) = 0$, la distancia entre un mismo individuo es cero.
- $d(i,j) = d(j,i)$, la matriz de distancia es simétrica.
- $d(i,j) \leq d(i,k) + d(k,j)$, la distancia menor entre dos puntos es la distancia entre ambos. Esta característica es conocida como desigualdad triangular.

donde i, j, k son todas y cada una de las instancias.

Las anteriores desigualdades se cumplirán al emplear los siguientes procedimientos de cálculo de la distancia entre diferentes individuos, pero debemos tener presente, que por ejemplo la matriz de distancias entre los municipios de cierta región no tiene por qué garantizar la simetría de la matriz.

- **Distancia Euclídea**: define la línea recta entre dos puntos definidos en el espacio euclídeo n-dimensional. En un espacio cartesiano podríamos representar esta distancia de la siguiente manera:

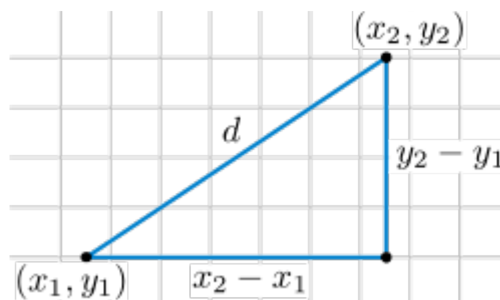


Figura 28: Distancia euclídea en el plano[39]

Se calcula aplicando la siguiente fórmula:

[39] Imagen extraída de: https://es.wikipedia.org/wiki/Distancia_euclidiana

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$d(i,j) = \sqrt{\sum_{k=1}^{p} (i_k - j_k)^2}$$

donde i, j son todas y cada una de las instancias, y se calcula la diferencia entre todas y cada una de las p características que las definen.

El problema de esta métrica es que es sensible a las unidades en las que fueron medidas cada una de las características que definen a los individuos, es decir, una variable cuya escala de medición obtenga medidas muy grandes contribuirá en mayor medida que las diferencias de una característica cuyos valores sean menores. Como consecuencia, los cambios de escala contribuyen en la distancia entre individuos. Una solución a este problema es la tipificación o normalización previa de las características.

- **Distancia de Manhattan**: métrica definida como la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas. En un espacio cartesiano podríamos representar esta distancia de la siguiente manera:

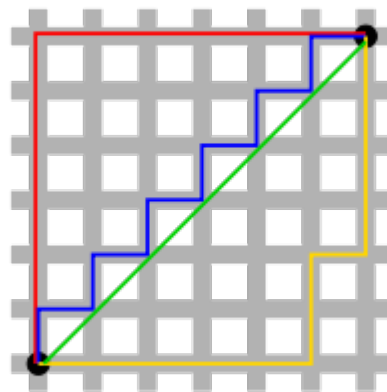


Figura 29: Distancia Manhattan en el plano(roja, azul y amarilla)[40]

Se calcula aplicando la siguiente fórmula:

[40] Imagen extraída de: https://es.wikipedia.org/wiki/Geometr%C3%ADa_del_taxista

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

$$d(i,j) = \sum_{k=1}^{p} |i_k - j_k|$$

donde i,j son todas y cada una de las instancias, y se calcula la diferencia entre todas y cada una de las p características que las definen.

Esta manera de medir la disimetría entre individuos tiene el mismo problema que se había mencionado para la distancia euclídea.

- **Distancia de Mahalanobis**: métrica definida para tener presenta la forma de los datos, evitando que la escala de las variables puede afectar en el cálculo de la disimetría. Se calcula aplicando la siguiente fórmula:

$$d(\vec{i},\vec{j}) = \sqrt{(\vec{i}-\vec{j})^t \Sigma^{-1} (\vec{i},\vec{j})}$$

donde $\vec{i},\vec{j}$ son los valores obtenidos en cada característica en dos instancias, donde Σ es la matriz de covarianzas.

Las definiciones de distancia que han sido presentadas funcionan bien con características numéricas, pero en la mayoría de las ocasiones, nuestra base de conocimiento estará compuesta tanto por variables numéricas como por variables categóricas. Para poder trabajar en las situaciones en las que no tengamos únicamente variables de tipo numérico, podemos emplear el **índice de Jaccard**, o plantearnos la ponderación de cada una de las características (**distancia ponderada**).

Tal y como habíamos enunciado, los procedimientos de agrupación tienen que conseguir grupos de instancias, donde en cada grupo las instancias deben de ser lo más homogéneas posibles entre sí y los más heterogéneas con el resto de clústeres. Algunas de las medidas empleadas para medir la calidad en el agrupamiento son el índice Davies-Bouldin (DB), o el coeficiente de Silhouette, entre otras muchas.

Tal y como habíamos planteado en la descripción, existen numerosos procedimientos de agrupación, en este material nos centraremos en comprender el funcionamiento del algoritmo k-Means. El algoritmo k-Means se encuentra dentro de los algoritmos de

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

agrupación no jerárquicos por la forma en la que segmentan el espacio. Otra categoría dentro de los procedimientos de agrupación son los algoritmos jerárquicos, los cuales generan un dendrograma para representar la cercanía entre instancias, proporcionando una jerarquía entre los diferentes clústeres.

9.2.1.1. Algoritmo k-Means

K-means es un algoritmo de clasificación no supervisada que agrupa las instancias en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster. Se suele usar la distancia cuadrática. Es decir, nos estaríamos planteando lo siguiente:

$$\min_{S} \sum_{j=1}^{k} \sum_{x_i \in S_j} \left\| x_i - \mu_j \right\|^2$$

donde S es el conjunto de datos cuyos elementos son los objetos x_i representados por vectores, donde cada uno de sus elementos representa una característica o atributo. Tendremos k grupos o clústeres con su correspondiente centroide μ_j.

El algoritmo que se emplea es el siguiente:

1. Definir los k centroides iniciales. Para realizar esta elección se pueden utilizar una de las diferentes estrategias:

    a. Elegir los k centroides aleatoriamente.

    b. Seleccionar k instancias como centroides.

2. Asignar cada instancia al centroide que se encuentre más cerca. Para ello el algoritmo tiene que calcular la distancia entre todas las instancias y los centroides.

3. Los valores de los centroides se recalculan tomando la media de los valores de los atributos de las instancias que forman cada clúster.

4. Repetir los pasos 2 y 3 iterativamente hasta que no haya más cambios de instancias

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

entre los diferentes clústeres.

El algoritmo K-means necesita tener definido desde el comienzo el valor del número de agrupaciones que debe de realizar, siendo muy posible que desconozcamos cuál es el número que debemos de fijar.

Para tomar la decisión del número grupos que el algoritmo debe realizar podemos basar nuestra decisión en el método de Elbow, con el que se puede construir una gráfica del siguiente estilo:

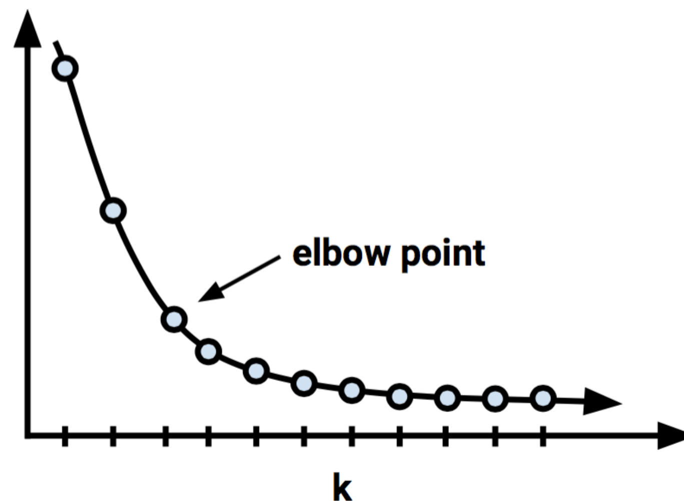


Figura 30: Determinación de número óptimo de clústeres mediante el método de Elbow[41]

El método de Elbow ejecuta el algoritmo para diferente número de grupos y selecciona aquel donde aumentar el número de clústeres no representa un valor significativo en la ganancia.

[41] Imagen extraída de: http://minerandodados.com.br/index.php/2017/12/12/entenda-o-algoritmo-k-means/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10. Caso de uso de Apache Spark (II). Spark MLlib

MLlib es la librería de aprendizaje automático (ML) de Spark. Su objetivo es hacer que el aprendizaje automático sea escalable y fácil. A un alto nivel, proporciona herramientas tales como:

- **Algoritmos ML**: algoritmos comunes de aprendizaje automático, como clasificación, regresión, agrupación y filtrado colaborativo.
- **Característica**: extracción de características, transformación, reducción de dimensionalidad y selección.
- **Fuentes de información**: herramientas para la construcción, evaluación y ajuste de fuentes de información de aprendizaje automático.
- **Persistencia**: guardar y cargar algoritmos, modelos y APIs de flujo ("*pipeline*") de alto nivel.
- **Utilidades**: álgebra lineal, estadísticas, manejo de datos, entre otras muchas.

Está formada por dos paquetes:

- spark.**mllib**[42], que contiene el API original construido sobre RDDs
- spark.**ml**[43], proporciona APIs de más alto nivel construidas sobre DataFrames para crear flujos completos (pipelines) de aprendizaje automático.

Es necesario reseñar que la API basada en MLlib RDD está en modo de mantenimiento a partir de la versión Spark 2.0, dado que MLlib agregará características a la API para alcanzar la paridad de características con la API basada en RDD, tras alcanzar la paridad de características, la API basada en RDD quedará en desuso, y se espera su eliminación en

[42] Más información en: https://spark.apache.org/docs/latest/mllib-guide.html
[43] Más información en: https://spark.apache.org/docs/latest/ml-guide.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

futuras versiones.

La API basada en DataFrame para MLlib proporciona una API uniforme en todos los algoritmos ML y en múltiples idiomas. Los DataFrames facilitan la práctica ML Pipelines, particularmente las transformaciones de características.

Es por todo ello que se recomienda el uso de spark.ml porque el API con los DataFrames es más versátil y flexible y será la que estudiemos en profundidad en este módulo, aunque también haremos una breve reseña al final a spark.mllib porque implementa un mayor número de modelos que spark.ml en algunas de las versiones, haciendo que sea interesante su uso en ciertas ocasiones.

Ambas librerías proporcionan un gran número de funcionalidades relacionadas con el aprendizaje automático que permiten cubrir las necesidades de cualquier proyecto de análisis de datos:

Estadísticas Básicas	• Estadística descriptiva • Correlación • Muestreo
Extracción, transformación y selección de características	**Extracción:** • TF-IDF • Word2Vec **Transformación:** • Tokenizer • Normalizer • VectorAssembler

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

<table>
<tr><td></td><td>Selección:
VectorSlicer
RFormula
ChiSqSelector</td></tr>
<tr><td>Clasificación y Regresión</td><td>Clasificación:
Regresión logística
Perceptrón multicapa
SVM (Support Vector Machine)
Naive Bayes
Regresión:
Regresión lineal
Regresión de supervivencia
Regresión isotónica
Árboles de decisión:
Random Forests
Gradient-Boosted Trees (GBTs)</td></tr>
<tr><td>Clustering</td><td>K-means
Bisecting k-means</td></tr>
</table>

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Reducción de la dimensionalidad	• PCA (análisis de componentes principales)

Tabla 7: Principales funcionalidades de spark.ml (versión 2.4.0)

La lista de funcionalidades que se indican en la Tabla 7 correspondiente a la versión 2.4.0 de Spark, que es la que estamos utilizando en este manual. A medida que las versiones de Spark avanzan esta lista aumenta. En la Tabla no se ha recogido toda la funcionalidad por simplicidad, pudiendo encontrar el listado completo en la guía de Spark.

10.1. Librería spark.ml sobre DataFrames[44]

Spark.ml estandariza los APIs para los algoritmos de Machine Learning haciendo muy sencillo combinar múltiples trabajos dentro de un mismo flujo (pipeline o workflow).

A continuación, detallamos los conceptos clave introducidos por el API de Spark ML donde el concepto de pipeline se inspira principalmente en el proyecto Python de scikit-learn[45]:

- **Dataframe**: Spark ML usa los DataFrames de Spark SQL como los dataset principales para las tareas de ML, donde se pueden almacenar una gran variedad de tipos de datos. Un DataFrame puede tener diferentes columnas almacenando texto, vectores de características, etiquetas, predicciones…

- **Transformer**: Un Transformer es un algoritmo que puede transformar un DataFrame en otro DataFrame, por ejemplo, un modelo ML es un Transformer que transforma un DataFrame con variables predictoras en un DataFrame con predicciones.

- **Estimator**: Un Estimator es un algoritmo que se puede aplicar (fit) sobre un Dataframe para producir un Transformer. Por ejemplo, un algoritmo de aprendizaje es un Estimator que se entrena con un DataFrame y produce un modelo.

[44] Más información en: https://spark.apache.org/docs/latest/ml-guide.html
[45] Más información en: https://scikit-learn.org/stable/

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- **Pipeline**: Un pipeline encadena múltiples Transformers y Estimators juntos especificando un flujo de ML.
- **Parameter**: Todos los Transformers y Estimators comparten ahora un API para especificar los parámetros.
- Como salida a la aplicación de cada uno de los algoritmos se obtiene un objeto tipo modelo de la clase correspondiente al tipo de algoritmo. Por ejemplo, si aplicamos LinearRegression obtenemos LinearRegresionModel.

10.1.1. Tipos de datos especiales para Spark ml y Spark mllib[46]

Además de los datos soportados por los Dataframes las librerías ml y mllib soportan los siguientes tipos de datos especiales:

- **Vectores locales**[47], que pueden ser:
    - **Vectores densos** (dense): en cuyo caso se especifican todos los valores del vector y se pueden crear a partir de arrays de Numpy (numpy.array[1,2,3]) y listas Python [1,2,3].
    - **Vectores dispersos** (sparse): son aquellos que contienen una gran cantidad de ceros, por lo que se definen especificando únicamente los valores no nulos junto con sus posiciones. Se pueden crear a partir de la clase SparseVector de mllib y csc_matrix de Scipy.
- **Puntos etiquetados**[48] (Labeled Points): son vectores locales (densos o dispersos) que

[46] Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html
[47] Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#local-vector
[48] Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#labeled-point

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

tienen asociados una respuesta o etiqueta numérica que a menudo suelen representar una predicción. Estas etiquetas pueden ser de tipo:

- **Double**, en casos de regresión.
- **0/1**, en casos de clasificación binaria.
- **Índices numéricos**, en casos de clasificación multiclase.

Además de estos dos tipos principales existen las **Matrices Locales**[49] densas y dispersas y las **Matrices Distribuidas**[50] que a su vez se diferencian en:

- **RowMatrix**[51].
- **IndexedRowMatrix**[52].
- **CoordinateMatrix**[53].
- **BlockMatrix**[54].

10.1.2. Extracción, transformación y selección de características

En spark.ml existen distintos componentes en el módulo pyspark.ml.feature que permiten

---

49 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#local-matrix
50 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#distributed-matrix
51 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#rowmatrix
52 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#indexedrowmatrix
53 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#coordinatematrix
54 Más información en: https://spark.apache.org/docs/latest/mllib-data-types.html#blockmatrix

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

realizar múltiples tareas sobre los DataFrame que ayudan a preparar los datos de cara a su utilización posterior en la creación de los modelos.

Hay tres tipos de estos componentes, a saber:

- Extractores.
- Transformadores.
- Selectores.

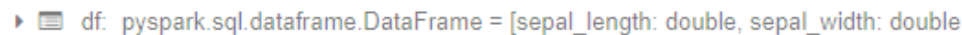
En la siguiente figura podemos ver un ejemplo de utilización del Transformador VectorAssembler que deberemos de usar, lo más probable, al principio de todos los ejemplos de modelos que sean construidos.

VectorAssembler obtiene una columna (outputCol) con un dato tipo Vector() numérico a partir de las columnas numéricas que se le indiquen en el parámetro inputCols. En este caso se crea el Transformador tipo VectorAssembler (vecAssembler) al que se le pasan como columnas de entrada (inputCols) sepal_length, sepal_width, petal_length y petal_width y como columna de salida (outputCol) features. Aplicando este Transformador mediante la función transform() al DataFrame original, obteniendo otro DataFrame similar al original pero con la columna features a mayores.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
from pyspark.sql.types import *

schema = StructType([
  StructField("sepal_length", DoubleType(), True),
  StructField("sepal_width", DoubleType(), True),
  StructField("petal_length", DoubleType(), True),
  StructField("petal_width", DoubleType(), True),
  StructField("class", StringType(), True),
])

df = (spark.read
      .format("csv")
      .option("header", "false")
      .option("sep", ",")
      .schema(schema)
      .load("/FileStore/tables/iris.csv"))

df.show()

```

▸ (1) Spark Jobs




```
+------------+-----------+------------+-----------+-----------+
|sepal_length|sepal_width|petal_length|petal_width|      class|
+------------+-----------+------------+-----------+-----------+
|         5.1|        3.5|         1.4|        0.2|Iris-setosa|
|         4.9|        3.0|         1.4|        0.2|Iris-setosa|
|         4.7|        3.2|         1.3|        0.2|Iris-setosa|
```

Figura 31: Lectura de datos de lirios

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["sepal_length","sepal_width", "petal_length"],
    outputCol="features")

df_custom = assembler.transform(df)
df_custom.show()
```

```
▸ (1) Spark Jobs
▸ df_custom: pyspark.sql.dataframe.DataFrame = [sepal_length: double, sepal_width: double ... 4 m
+------------+-----------+------------+-----------+-----------+-------------+
|sepal_length|sepal_width|petal_length|petal_width|      class|     features|
+------------+-----------+------------+-----------+-----------+-------------+
|         5.1|        3.5|         1.4|        0.2|Iris-setosa|[5.1,3.5,1.4]|
|         4.9|        3.0|         1.4|        0.2|Iris-setosa|[4.9,3.0,1.4]|
|         4.7|        3.2|         1.3|        0.2|Iris-setosa|[4.7,3.2,1.3]|
```

Figura 32: Aplicación del Transformador VectorAssembler a un DataFrame

Los datos empleados en el ejemplo son las medida de un conjunto de lirios, podemos encontrar todos los datos en: https://archive.ics.uci.edu/ml/datasets/iris

10.1.3. Aprendizaje Supervisado: Regresión[55]

ML implementa varios algoritmos para realizar regresiones:

- Regresión Lineal
- Modelo Lineal Generalizada (GLM, Generalized Linear Model)
- Regresión con Árboles de Decisión
- Regresión con Random Forest
- Regresión GBT (Gradient-boosted tree)

[55] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#regression

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Regresión para el análisis de supervivencia
- Regresión Isotónica

10.1.3.1. Regresión Lineal[56]

La clase Python que utilizamos para la implementación de modelos de Regresión con spark.ml es LinearRegression:

> class pyspark.ml.regression.LinearRegression(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, standardization=True, solver='auto', weightCol=None, aggregationDepth=2, loss='squaredError', epsilon=1.35)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[57].

Tras la aplicación del algoritmo de regresión lineal mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: LinearRegressionModel [58].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/linear_regression_with_elastic_net.py

Un ejemplo lo tendremos en la Figura 31 puesto que el modelos de regresión lineal es una extensión del Modelo Lineal Generalizado.

---

[56] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-regression
[57] Más información en: https://en.wikipedia.org/wiki/Generalized_linear_model
[58] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.LogisticRegressionModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10.1.3.2. Modelo Lineal Generalizada[59]

La clase Python que utilizamos para la implementación de modelos de Regresión Lineales Generalizados con spark.ml es GeneralizedLinearRegression:

```
class pyspark.ml.regression.GeneralizedLinearRegression(labelCol='label', featuresCol='features', predictionCol='prediction', family='gaussian', link=None, fitIntercept=True, maxIter=25, tol=1e-06, regParam=0.0, weightCol=None, solver='irls', linkPredictionCol=None, variancePower=0.0, linkPower=None, offsetCol=None)
```

En contraste con la regresión lineal donde se supone que la salida sigue una distribución gaussiana, los modelos lineales generalizados (GLM) son modelos lineales donde la variable de respuesta sigue alguna distribución de la familia de distribuciones exponenciales. Empleando GeneralizedLinearRegression de Spark nos permite la especificación flexible de modelos GLM que se pueden usar para varios tipos de problemas de predicción, dentro de los que incluye regresión lineal, regresión de Poisson, regresión logística, entre otros.

Actualmente spark.ml, solo se admite un subconjunto de distribuciones de familias exponenciales, las cuales se encuentran recogidas en la siguiente tabla:

Familia	Tipo de respuesta	Enlaces soportados
Gaussiano	Continuo	Identidad, Log, Inverso
Binomio	Binario	Logit, Probit, CLogLog
Poisson	Contar	Log, Identidad, Sqrt

[59] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#generalized-linear-regression

Gama	Continuo	Inverso, identidad, Log
Tweedie	Cero-inflado continuo	Potencia

Tabla 8: Familias disponibles para regresión lineal generalizada en Spark 2.4.0

Para poder implementar este modelo empleando los datos del tamaño de lirio podríamos realizarlo del siguiente modo, donde aplicaremos una regresión lineal. Procederemos a explicar el ancho del pétalo con el resto de variable numéricas que tenemos:

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, VectorAssembler
from pyspark.ml.regression import GeneralizedLinearRegression

assembler = VectorAssembler(
    inputCols=["sepal_length","sepal_width", "petal_length"],
    outputCol="features")

pipeline = Pipeline(stages=[assembler])
df_custom = pipeline.fit(df).transform(df)

glr = GeneralizedLinearRegression(family="gaussian", link="identity", maxIter=10, regParam=0.3,
                                  featuresCol="features", labelCol="petal_width")

# Ajustar el modelo
model = glr.fit(df_custom)

# Coeficientes para el modelo de regresión lineal generalizada
# Print the coefficients and intercept for generalized linear regression model
print("Coefficients: " + str(model.coefficients))
print("Intercept: " + str(model.intercept))

# Estadísticos resumen del modelo sobre el conjunto de entrenamiento
summary = model.summary
print("Coefficient Standard Errors: " + str(summary.coefficientStandardErrors))
print("T Values: " + str(summary.tValues))
print("P Values: " + str(summary.pValues))
print("Dispersion: " + str(summary.dispersion))
print("Null Deviance: " + str(summary.nullDeviance))
print("Residual Degree Of Freedom Null: " + str(summary.residualDegreeOfFreedomNull))
print("Deviance: " + str(summary.deviance))
print("Residual Degree Of Freedom: " + str(summary.residualDegreeOfFreedom))
print("AIC: " + str(summary.aic))
print("Deviance Residuals: ")
summary.residuals().show()
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
Coefficients: [-1.18948880647,0.915493394059,1.6352897591]
Intercept: 4.2081536591453155
Coefficient Standard Errors: [1.0131796648784284, 1.1841461440687162, 0.5158337102383642, 4.6134345984110245]
T Values: [-1.174015673330264, 0.7731253432229558, 3.170187846659695, 0.9121520137284925]
P Values: [0.24230024151832108, 0.4406980992598144, 0.0018565311649156868, 0.36319238523044021]
Dispersion: 28.019812470908196
Null Deviance: 4779.999999999997
Residual Degree Of Freedom Null: 149
Deviance: 4090.8926207525965
Residual Degree Of Freedom: 146
AIC: 931.5640363331225
Deviance Residuals:
+-------------------+
|  devianceResiduals|
+-------------------+
|  -3.63539328810523|
| -3.415544352368952|
| -3.673011816564657|
| -4.027469309624417|
|-3.8458915081578335|
|  1.864669068482586|
```

Figura 33: Ejemplo de Regresión Lineal Generalizada con Spark ml 2.4.0 en pySpark

10.1.3.3. Regresión con Árboles de Decisión[60]

La clase Python que utilizamos para la implementación de modelos de Árboles de Decisión con spark.ml es DecisionTreeRegressor:

> class pyspark.ml.regression.DecisionTreeRegressor(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', seed=None, varianceCol=None)

Los árboles de decisión son un conjunto de algoritmos muy populares empleador para las tareas de aprendizaje automático de clasificación y regresión. Los árboles de decisión son ampliamente utilizados, ya que son fáciles de interpretar, se extienden a la configuración de clasificación multiclase, no requieren escalamiento de características y pueden

[60] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-regression

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

capturar la no linealidad de las características e interacciones de características. Los algoritmos de conjuntos de árboles, como los Random Forests and Gradient-boosted, se encuentran entre los de mejor desempeño para las tareas de clasificación y regresión. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[61].

Tras la aplicación del algoritmo de regresión de Árboles de Decisión mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: DecisionTreeRegressionModel[62].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/decision_tree_regression_example.py

10.1.3.4. Regresión con Random Forest[63]

La clase Python que utilizamos para la implementación de modelos de regresión Random Forest con spark.ml es RandomForestRegressor:

```
class pyspark.ml.regression.RandomForestRegressor(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20, featureSubsetStrategy='auto')
```

El algoritmo Random Forest combinan muchos árboles de decisión para reducir el riesgo de sobreentrenamiento. Su uso es compatible para la clasificación binaria y multiclase y para la regresión, utilizando características tanto continuas como categóricas. Si quieres saber

[61] Más información en: https://en.wikipedia.org/wiki/Decision_tree_learning
[62] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.DecisionTreeRegressionModel
[63] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-regression

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[64].

Tras la aplicación del algoritmo de Random Forest mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: RandomForestRegressionModel[65].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/random_forest_regressor_example.py

10.1.3.5. Regresión GBT (Gradient-boosted tree)[66]

La clase Python que utilizamos para la implementación de modelos de Regresión GBT con spark.ml es GBTRegressor:

> class pyspark.ml.regression.GBTRegressor(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10, lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance', featureSubsetStrategy='all')

Los árboles impulsados por gradiente (GBT) son conjuntos de árboles de decisión. Este algoritmo entrena iterativamente una secuencia de árboles de decisión. En cada iteración, el algoritmo usa el conjunto actual para predecir la etiqueta de cada instancia de entrenamiento y luego compara la predicción con la etiqueta verdadera. El conjunto de datos se vuelve a etiquetar para poner más énfasis en instancias de entrenamiento con predicciones deficientes. Por lo tanto, en la siguiente iteración, el árbol de decisión ayudará a corregir errores anteriores. El mecanismo específico para volver a etiquetar las instancias se define mediante una función de error. Con cada iteración, los GBT reducen aún más esta

---

[64] Más información en: https://en.wikipedia.org/wiki/Random_forest
[65] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.RandomForestRegressionModel
[66] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-regression

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

función de error en los datos de entrenamiento. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[67].

Tras la aplicación del algoritmo de regresión GBT mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: GBTRegressionMode[68].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/gradient_boosted_tree_regressor_example.py

10.1.3.6. Regresión para el análisis de supervivencia[69]

La clase Python que utilizamos para la implementación de modelos de Regresión de Supervivencia con spark.ml es AFTSurvivalRegression:

> class pyspark.ml.regression.AFTSurvivalRegression(self, featuresCol="features", labelCol="label", predictionCol="prediction", fitIntercept=True, maxIter=100, tol=1E-6, censorCol="censor", quantileProbabilities=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99], quantilesCol=None, aggregationDepth=2)

Esta clase implementa el modelo AFT (Accelerated Failure Time) de Survival Regression basado en la distribución de Weibull, siendo un modelo de regresión de supervivencia paramétrica para datos censurados. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[70].

Tras la aplicación del algoritmo de regresión AFT mediante un Estimator (fit()) a un

---

[67] Más información en: https://en.wikipedia.org/wiki/Gradient_boosting
[68] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.GBTRegressor
[69] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#survival-regression

[70] Más información en: https://en.wikipedia.org/wiki/Accelerated_failure_time_model

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

DataFrame de training obtenemos un modelo del tipo: AFTSurvivalRegressionModel[71].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/aft_survival_regression.py

10.1.3.7. Regresión Isotónica[72]

La clase Python que utilizamos para la implementación de modelos de Regresión Isotónica con spark.ml es IsotonicRegression:

class pyspark.ml.regression.IsotonicRegression(featuresCol='features', labelCol='label', predictionCol='prediction', weightCol=None, isotonic=True, featureIndex=0)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[73].

Tras la aplicación del algoritmo de regresión isotónica mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: IsotonicRegressionModel[74].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/isotonic_regression_example.py

[71] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.AFTSurvivalRegressionModel
[72] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#isotonic-regression
[73] Más información en: https://en.wikipedia.org/wiki/Isotonic_regression
[74] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.IsotonicRegressionModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10.1.4. Aprendizaje Supervisado: Clasificación[75]

ML implementa varios algoritmos para realizar clasificación:

- Regresión logística
- Árboles de decisión
- Random Forest
- Gradient-Boosted Tree
- Perceptrón multicapa
- Máquina de Vector de Soporte Lineal
- Naive Bayes

10.1.4.1. Regresión logística[76]

La clase Python que utilizamos para la implementación de modelos de Regresión logística con spark.ml es LogisticRegression:

> class pyspark.ml.classification.LogisticRegression(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPredictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, family='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None)

[75] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#classification
[76] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#logistic-regression

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[77].

Tras la aplicación del algoritmo de regresión logística mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: LogisticRegressionModel [78].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/logistic_regression_summary_example.py

Empleando el conjunto de datos de los lirios, podríamos realizar la clasificación entre algún par de especies presentes en el conjunto de datos. Tenemos un ejemplo en la siguiente imagen donde se:

- Añade una nueva variable categórica
- Se transforman las variables para poder emplear el modelo
- Se computa el modelo
- Se extraen estadísticos para la valoración de la calidad del modelo.

La interpretación de los parámetros se realizaría tal y como se detalló anteriormente en la descripción de dicho modelo. Podríamos ampliar el ejemplo habiendo realizado previamente una separación del conjunto de datos en entrenamiento y prueba.

[77] Más información en: https://en.wikipedia.org/wiki/Generalized_linear_model
[78] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.LogisticRegressionModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```python
from pyspark.sql import functions as F

dataset = (df
           .withColumn('categoica', F.when(F.col("sepal_length") < 5, F.lit("A")).otherwise(F.lit("B")))
           .filter(F.col("class") != F.lit("Iris-setosa")))

# Use One-Hot Encoding para convertir todas las variables categóricas en vectores binarios.

from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, VectorAssembler

categoricalColumns = ["categoica"]

stages = []  # etapas en nuestro Pipeline
for categoricalCol in categoricalColumns:
    # Categorica de indexación con StringIndexer
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + "Index")
    # Use OneHotEncoder para convertir variables categóricas en SparseVectors binarios
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    # Añadir etapas. Estos no se ejecutan aquí, pero se ejecutarán todos de una vez más adelante
    stages += [stringIndexer, encoder]

# Convertir la etiqueta en índices de etiquetas usando el StringIndexer
label_stringIdx = StringIndexer(inputCol="class", outputCol="label")
stages += [label_stringIdx]

# Transforma todas las características en un vector usando VectorAssembler
numericCols = ["sepal_length", "sepal_width", "petal_length", "petal_width"]
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]

# Ejecutar las etapas como una tubería. Esto coloca los datos a través de todas las transformaciones de características en una sola llamada.
partialPipeline = Pipeline().setStages(stages)
pipelineModel = partialPipeline.fit(dataset)
preppedDataDF = pipelineModel.transform(dataset)

# Ajustar el modelo de regresión logística.
lrModel = LogisticRegression().fit(preppedDataDF)

# Curva ROC
display(lrModel, preppedDataDF, "ROC")
```

Figura 34: Ejemplo de regresión logística

10.1.4.1. Clasificación con Árboles de Decisión[79]

La clase Python que utilizamos para la implementación de modelos de Árboles de Decisión con spark.ml es DecisionTreeClassifier:

> class pyspark.ml.classification.DecisionTreeClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256,

[79] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#decision-tree-classifier

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

cacheNodeIds=False, checkpointInterval=10, impurity='gini', seed=None)

Los árboles de decisión son un conjunto de algoritmos muy populares empleador para las tareas de aprendizaje automático de clasificación y regresión. Los árboles de decisión son ampliamente utilizados, ya que son fáciles de interpretar, se extienden a la configuración de clasificación multiclase, no requieren escalamiento de características y pueden capturar la no linealidad de las características e interacciones de características. Los algoritmos de conjuntos de árboles, como los Random Forests and Gradient-boosted, se encuentran entre los de mejor desempeño para las tareas de clasificación y regresión. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[80].

Tras la aplicación del algoritmo de regresión de Árboles de Decisión mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: DecisionTreeClassificationModel [81].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/decision_tree_classification_example.py

10.1.4.2. Clasificación con Random Forest[82]

La clase Python que utilizamos para la implementación de modelos de regresión Random Forest con spark.ml es RandomForestClassifier:

class pyspark.ml.classification.RandomForestClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32,

[80] Más información en: https://en.wikipedia.org/wiki/Decision_tree_learning
[81] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.DecisionTreeClassificationModel
[82] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0)

El algoritmo Random Forest combinan muchos árboles de decisión para reducir el riesgo de sobreentrenamiento. Su uso es compatible para la clasificación binaria y multiclase y para la regresión, utilizando características tanto continuas como categóricas. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[83].

Tras la aplicación del algoritmo de Random Forest mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: RandomForestClassificationModel[84].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/random_forest_classifier_example.py

10.1.4.3. Clasificación GBT (Gradient-boosted tree)[85]

La clase Python que utilizamos para la implementación de modelos de Regresión GBT con spark.ml es GBTClassifier:

class pyspark.ml.classification.GBTClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic', maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, featureSubsetStrategy='all')

Los árboles impulsados por gradiente (GBT) son conjuntos de árboles de decisión. Este

---

83 Más información en: https://en.wikipedia.org/wiki/Random_forest

84 https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.RandomForestClassificationModel

85 Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-tree-classifier

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

algoritmo entrena iterativamente una secuencia de árboles de decisión. En cada iteración, el algoritmo usa el conjunto actual para predecir la etiqueta de cada instancia de entrenamiento y luego compara la predicción con la etiqueta verdadera. El conjunto de datos se vuelve a etiquetar para poner más énfasis en instancias de entrenamiento con predicciones deficientes. Por lo tanto, en la siguiente iteración, el árbol de decisión ayudará a corregir errores anteriores. El mecanismo específico para volver a etiquetar las instancias se define mediante una función de error. Con cada iteración, los GBT reducen aún más esta función de error en los datos de entrenamiento. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[86].

Tras la aplicación del algoritmo de regresión GBT mediante un Estimator (fit()) a un DataFrame de training, obtenemos un modelo del tipo: GBTClassificationModel [87].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/gradient_boosted_tree_classifier_example.py

10.1.4.4. Perceptrón multicapa[88]

La clase Python que utilizamos para la implementación de modelos de red neuronal con spark.ml es MultilayerPerceptronClassifier:

```
class pyspark.ml.classification.MultilayerPerceptronClassifier(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, tol=1e-06, seed=None, layers=None, blockSize=128, stepSize=0.03, solver='l-bfgs', initialWeights=None, probabilityCol='probability', rawPredictionCol='rawPrediction')
```

[86] Más información en: https://en.wikipedia.org/wiki/Gradient_boosting

[87] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.GBTClassificationModel

[88] Más información: https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-classifier

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[89].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/multilayer_perceptron_classification.py

10.1.4.5. *Máquina de Vector de Soporte Lineal*[90]

La clase Python que utilizamos para la implementación de modelos de máquina de vectores soporte lineal con spark.ml es LinearSVC:

> class pyspark.ml.classification.LinearSVC(featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[91].

Tras la aplicación del algoritmo de máquina de vectores soporte lineal mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: LinearSVCModel[92].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/linearsvc.pyna

[89] Más información en: https://en.wikipedia.org/wiki/Feedforward_neural_network
[90] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine
[91] Más información en: https://en.wikipedia.org/wiki/Support_vector_machine
[92] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.LinearSVCModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10.1.4.6. Naive Bayes[93]

La clase Python que utilizamos para la implementación de modelos de Naive Bayes con spark.ml es NaiveBayes:

> class pyspark.ml.classification.NaiveBayes(featuresCol='features', labelCol='label', predictionCol='prediction', probabilityCol='probability', rawPredictionCol='rawPrediction', smoothing=1.0, modelType='multinomial', thresholds=None, weightCol=None)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[94].

Tras la aplicación del algoritmo de Naive Bayes mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: NaiveBayesModel[95].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/naive_bayes_example.py

10.1.5. Aprendizaje no supervisado: Clustering[96]

ML implementa varios algoritmos para realizar agrupaciones:

- K-means
- Latent Dirichlet allocation (LDA)

---

[93] Más información en: https://spark.apache.org/docs/latest/ml-classification-regression.html#naive-bayes
[94] Más información en: https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[95] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.NaiveBayesModel
[96] Más información en: https://spark.apache.org/docs/latest/ml-clustering.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

- Bisecting k-means
- Gaussian Mixture Model (GMM)

10.1.5.1. K-means[97]

La clase Python que utilizamos para la implementación de modelos de agrupación empleando k-means con spark.ml es KMeans:

> class pyspark.ml.clustering.KMeans(featuresCol='features', predictionCol='prediction', k=2, initMode='k-means||', initSteps=2, tol=0.0001, maxIter=20, seed=None, distanceMeasure='euclidean')

En el apartado 9.2.1.1 explicamos el funcionamiento de este modelo. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[98].

Tras la aplicación del algoritmo el modelos de agrupación empleando k-means mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: KMeansModel[99].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/kmeans_example.py

---

[97] Más información en: https://spark.apache.org/docs/latest/ml-clustering.html#k-means
[98] Más información en: https://en.wikipedia.org/wiki/K-means_clustering
[99] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.clustering.KMeansModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

10.1.5.2. Latent Dirichlet allocation (LDA)[100]

La clase Python que utilizamos para la implementación de modelos de asignación de Dirichlet latente con spark.ml es LDA:

> class pyspark.ml.clustering.LDA(featuresCol='features', maxIter=20, seed=None, checkpointInterval=10, k=10, optimizer='online', learningOffset=1024.0, learningDecay=0.51, subsamplingRate=0.05, optimizeDocConcentration=True, docConcentration=None, topicConcentration=None, topicDistributionCol='topicDistribution', keepLastCheckpoint=True)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[101].

Tras la aplicación del algoritmo el modelos de asignación de Dirichlet latente mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: LocalLDAModel[102].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/lda_example.py

10.1.5.3. Bisecting k-means[103]

La clase Python que utilizamos para la implementación de modelos de agrupación jerárquica con spark.ml es BisectingKMeans:

[100] Más información en: https://spark.apache.org/docs/latest/ml-clustering.html#latent-dirichlet-allocation-lda
[101] Más información en: https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
[102] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.clustering.LocalLDAModel
[103] Más información en: https://spark.apache.org/docs/latest/ml-clustering.html#bisecting-k-means

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

class pyspark.ml.clustering.BisectingKMeans(featuresCol='features', predictionCol='prediction', maxIter=20, seed=None, k=4, minDivisibleClusterSize=1.0, distanceMeasure='euclidean')

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[104].

Tras la aplicación del algoritmo el modelos de agrupación jerárquica mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: BisectingKMeansModel[105].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/bisecting_k_means_example.py

10.1.5.4. Gaussian Mixture Model (GMM)[106]

La clase Python que utilizamos para la implementación de modelos de agrupación de mezcla gaussiana bayesiana con spark.ml es GaussianMixture:

class pyspark.ml.clustering.GaussianMixture(featuresCol='features', predictionCol='prediction', k=2, probabilityCol='probability', tol=0.01, maxIter=100, seed=None)

Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[107].

Tras la aplicación del algoritmo el modelos de agrupación jerárquica mediante un Estimator

[104] Más información en: https://en.wikipedia.org/wiki/Hierarchical_clustering
[105] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.clustering.BisectingKMeansModel
[106] Más información en: https://spark.apache.org/docs/latest/ml-clustering.html#gaussian-mixture-model-gmm
[107] Más información en: https://en.wikipedia.org/wiki/Mixture_model#Multivariate_Gaussian_mixture_model

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

(fit()) a un DataFrame de training obtenemos un modelo del tipo: GaussianMixtureModel [108].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/gaussian_mixture_example.py

10.1.6. Evaluación de modelos[109]

Tenemos que tener presente que para tener resultados confiables deberemos de guardarnos en todo momento una muestra de las observaciones para poder realizar la evolución final del mismo. Esto lo podemos realizar dividiendo el conjunto original en observaciones dedicadas al entrenamiento y a prueba. En Spark se realiza del siguiente modo:

```
train, test = data.randomSplit([0.9, 0.1], seed=12345)
```

Figura 35: Segmentación de instancias en entrenamiento y prueba

ML implementa varios algoritmos para la selección de modelos óptimos:

- Validación cruzada.
- División en entrenamiento y prueba.

Ambos algoritmos sustentan su ejecución en la clase Python de búsqueda de parámetros dentro de una malla de puntos con spark.ml es ParamGridBuilder:

class pyspark.ml.tuning.ParamGridBuilder

10.1.6.1. Validación cruzada[110]

[108] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.clustering.GaussianMixtureModel
[109] Más información en: https://spark.apache.org/docs/latest/ml-tuning.html
[110] Más información en: https://spark.apache.org/docs/latest/ml-tuning.html#cross-

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

La clase Python que utilizamos para la implementación de la validación cruzada con spark.ml es CrossValidator:

> class pyspark.ml.tuning.CrossValidator(estimator=None, estimatorParamMaps=None, evaluator=None, numFolds=3, seed=None, parallelism=1, collectSubModels=False)

En el apartado 9.1.3.1 explicamos el funcionamiento de este modelo. Si quieres saber más sobre este modelo puedes ver su correspondiente entrada en la Wikipedia[111].

Tras la aplicación del algoritmo validación cruzada mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: CrossValidatorModel[112].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/cross_validator.py

Empleando el conjunto de datos de los lirios, y continuando con la preparación de variables realizada en el ejemplo de una regresión logística, añadiremos las siguientes implementaciones:

- Declaración del modelo a emplear.
- Generación de la rejilla de parámetros con los que se va a probar.
- Declaración del modelo de validación cruzada a emplear.
- De igual modo que antes, se podrían extraen estadísticos para la valoración de la calidad del modelo.

La interpretación de los parámetros se realizaría tal y como se detalló anteriormente en la

validation

[111] Más información en: https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[112] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.tuning.CrossValidatorModel

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

descripción de dicho modelo. Podríamos ampliar el ejemplo habiendo realizado previamente una separación del conjunto de datos en entrenamiento y prueba.

```
34 from pyspark.ml.evaluation import BinaryClassificationEvaluator
35 from pyspark.ml.feature import HashingTF, Tokenizer
36 from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
37
38 # Declarar el modelo de regresión logística.
39 lrModel = LogisticRegression(maxIter=10)
40 stages += [lrModel]
41
42 # Configurar un Pipeline.
43 pipeline = Pipeline(stages=stages)
44
45 # Ahora tratamos el Pipeline como un estimador, envolviéndolo en una instancia de CrossValidator.
46 # Esto nos permitirá elegir conjuntamente los parámetros para todas las etapas del Pipeline.
47 paramGrid = ParamGridBuilder().build()
48
49 crossval = CrossValidator(estimator=pipeline,
50                           estimatorParamMaps=paramGrid,
51                           evaluator=BinaryClassificationEvaluator(),
52                           numFolds=3)  # use 3+ folds in practice
53
54 # Ejecutar la validación cruzada y elija el mejor conjunto de parámetros.
55 lrModel = crossval.fit(dataset).transform(dataset)
56 lrModel.show()
```

Figura 36: Ejemplo de validación cruzada

10.1.6.2. División en entrenamiento y prueba[113]

La clase Python que utilizamos para la implementación de división en subconjuntos con spark.ml es TrainValidationSplit:

> class pyspark.ml.tuning.TrainValidationSplit(estimator=None, estimatorParamMaps=None, evaluator=None, trainRatio=0.75, parallelism=1, collectSubModels=False, seed=None)

En el apartado 9.1.3.1 explicamos el funcionamiento de modelos muy similares. El algoritmo TrainValidationSplit solo evalúa cada combinación de parámetros una vez, en lugar de k

113 Más información en: https://spark.apache.org/docs/latest/ml-tuning.html#train-validation-split

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

veces en el caso de CrossValidator.

Tras la aplicación del algoritmo de división en subconjuntos mediante un Estimator (fit()) a un DataFrame de training obtenemos un modelo del tipo: TrainValidationSplitModel[114].

En el siguiente enlace podremos encontrar un ejemplo de aplicación: https://github.com/apache/spark/blob/master/examples/src/main/python/ml/train_validation_split.py

10.2. Librería spark.mllib sobre RDDs[115]

Como ya hemos comentado anteriormente la librería spark.mllib para realizar aprendizaje automático sobre RDDs es otra alternativa a spark.ml, pero parece clara la decisión de Spark de apostar por esta última, eliminando esta en futuras versiones. Por tanto, aquí sólo vamos a ver un ejemplo con un modelo de clasificación mediante un árbol de decisión para comparar con los ejemplos que hemos visto hasta ahora y valorar las diferencias.

Según se puede observar en el ejemplo los puntos distintos más relevantes de mllib con respecto a ml son:

- Se utilizan RDDs en vez de DataFrames.
- Se emplea Labeled Points en vez de Vectors.
- El modelo se crea de una pasada, no es necesario aplicar la estimación con la función fit().
- Para predecir se utiliza la función predict() en vez de transform().

[114] https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.tuning.TrainValidationSplitModel
[115] Más información en: https://spark.apache.org/docs/latest/mllib-guide.html

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils

def getLabelNum(label):
  if label == "Iris-setosa":
    return 0
  elif label == "Iris-versicolor":
    return 1
  else:
    return 2

data = (sc
        .textFile("/FileStore/tables/iris.csv")
        .map(lambda l: l.split(","))
        .map(lambda p: LabeledPoint(getLabelNum(p[4]), [p[0], p[1], p[2], p[3]])))

# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
model = DecisionTree.trainClassifier(trainingData, numClasses=3, categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5, maxBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification tree model:')
print(model.toDebugString())

# Save and load model
model.save(sc, "target/tmp/DecisionTreeClassificationModel")
sameModel = DecisionTreeModel.load(sc, "target/tmp/DecisionTreeClassificationModel")
```

Figura 37: Árbol de decisión con spark.mllib

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
Test Error = 0.02
Learned classification tree model:
DecisionTreeModel classifier of depth 5 with 17 nodes
  If (feature 2 <= 2.45)
   Predict: 0.0
  Else (feature 2 > 2.45)
   If (feature 3 <= 1.75)
    If (feature 2 <= 4.95)
     If (feature 0 <= 4.95)
      Predict: 2.0
     Else (feature 0 > 4.95)
      Predict: 1.0
    Else (feature 2 > 4.95)
     If (feature 3 <= 1.55)
      Predict: 2.0
     Else (feature 3 > 1.55)
      If (feature 0 <= 6.75)
       Predict: 1.0
      Else (feature 0 > 6.75)
       Predict: 2.0
   Else (feature 3 > 1.75)
    If (feature 2 <= 4.85)
     If (feature 0 <= 5.95)
      Predict: 1.0
     Else (feature 0 > 5.95)
      Predict: 2.0
    Else (feature 2 > 4.85)
     Predict: 2.0
```

Figura 38: Resultados del árbol de decisión con spark.mllib

Los datos empleados en el ejemplo son las medida de un conjunto de lirios, podemos encontrar todos los datos en: https://archive.ics.uci.edu/ml/datasets/iris

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

11. Caso de uso de Apache Spark (IV). Spark Streaming

El procesamiento en tiempo real permite a los usuarios consultar el flujo continuo de datos y detectar condiciones dentro de un pequeño período de tiempo desde el momento de recibir los datos, es decir, extraer conocimiento a la menor brevedad. El período de tiempo de detección y generación de conocimiento puede variar desde unos pocos milisegundos hasta minutos.

El procesamiento en tiempo real también recibe muchos nombres: análisis en tiempo real, análisis de transmisión, análisis de transmisión en tiempo real y procesamiento de eventos. Aunque algunos términos históricamente tenían diferencias, han convergido en el procesamiento en tiempo real.

Un ejemplo de aplicación del procesamiento en tiempo real, sería el funcionamiento de cierto sensor que toma mediciones de temperatura de cierto lugar, podría ser interesante que en cierto monitor se alertara cuando las mediciones fueran anómalas, es decir, si dicho sensor se encontrara dentro de un reactor sería interesante disponer de una alerta con tiempo suficiente como para apagarlo si la temperatura superara cierto valor, o por el contrario, si la temperatura es insuficiente, alertar de un mal funcionamiento; por el contrario si dicho sensor se encontrara en un hogar podría ser interesante para el encendido o el apagado de la climatización.

Como se ha puesto de manifiesto con el ejemplo, los conocimientos derivados del procesamiento de datos son valiosos, manifestando como en algunas situaciones disponer del conocimiento a la menor brevedad, para proceder a la toma de decisiones correctas, puede evitar problemas posteriores, tal y como refleja el ejemplo de la temperatura del núcleo, donde la información es valiosa en todo momento, pero su valor máximo podríamos decir que se obtiene en el punto en el que los operarios tienen suficiente tiempo de reacción ante la variación de la temperatura.

Si regresamos al ejemplo plateado, veremos que el lugar en el que se encuentra instalado cada uno de los dos sensores es clave, dado que las consecuencias que puede tener no disponer de la información adecuada, en el momento preciso, para proceder a la

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

extracción de conocimiento, puede hacer que no se tomen las decisiones adecuadas con la suficiente antelación. Es por ello, que la frecuencia con la que es recibida la información de ambos sensores puede ser muy distinta, en el caso del reactor nos interesará una alta frecuencia, mientras que, en la climatización del hogar, quizás, podríamos asumir una frecuencia que pudiera ser incluso superior a los cinco minutos.

Si analizamos el carácter que tiene los datos del ejemplo, caeremos rápidamente en la cuenta que se ajustan, por su marcado carácter temporal, a un análisis de series temporales, buscando encontrar patrones que se presentan a lo largo del tiempo. Otro conjunto de ejemplos, que podrían ser analizados mediante este conjunto de modelos matemáticos, serían los recolectados mediante los sensores de tráfico, los sensores de estado, los registros de transacciones, los registros de actividad, entre otros muchos. Es decir, la gran mayoría de los datos procedentes del internet de las cosas (IoT) son datos de series de tiempo.

Una de las principales fortalezas que tiene el procesamiento en tiempo real, es permitir manejar grandes volúmenes de información, que en algunas ocasiones no son siquiera almacenables, pero que su procesamiento puede generar gran conocimiento. Otra fortaleza, es que el comenzar el análisis de la información según se dispone de ella, hace que pueda realizarse con mucho menos hardware.

Sin embargo, el procesamiento en tiempo real no es un procedimiento de análisis de datos para todos los casos de uso. Una buena regla general es que, si el procesamiento necesita múltiples pasadas a través de datos del conjunto completo de registros, o se requiere de un reentrenamiento del modelo que se está empleando, entonces las ventajas que obteníamos mediante este procedimiento se pierden.

Spark Streaming es una extensión del Core Spark API que permite el procesamiento de flujos de datos en vivo, escalable, de alto rendimiento y tolerante a fallos. Los datos pueden ser ingeridos de muchas fuentes, como Kafka, y pueden procesarse utilizando complejos algoritmos expresados con funciones de alto nivel como *map*, *reduce*, *join* y *window*. Finalmente, los datos procesados se pueden enviar a sistemas de archivos, bases de datos y paneles de control en vivo.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Figura 39: Flujo Spark Streaming[116]

En el caso de Spark Streaming, los datos no son procesados en su momento de llegada, sino que todos los eventos que llegan dentro de un intervalo de tiempo son ejecutados conjuntamente, es decir, los datos se dividen en lotes, que luego son procesados por el motor de Spark para generar el flujo final de resultados en lotes. Este funcionamiento es representado mediante la siguiente imagen:

Figura 40: Flujo Spark Streaming por lotes[117]

Spark Streaming proporciona una abstracción de alto nivel llamada flujo discretizado o *DStream*, que representa un flujo continuo de datos. Los DStreams se pueden crear a partir de flujos de datos de entrada de diversas fuentes, tal y como se representaba en la imagen anterior, o aplicando operaciones de alto nivel en otros DStreams tales como como *map*, *windowy*, *reduceByKeyAndWindow*. Internamente, un DStream se representa como una secuencia de RDD del mismo tipo, de datos recibidos en un intervalo de tiempo.

[116] Imagen extraída de https://spark.apache.org
[117] Imagen extraída de https://spark.apache.org

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Mientras se ejecuta un programa Spark Streaming, cada DStream genera periódicamente un RDD, ya sea a partir de datos en vivo o transformando el RDD generado por un DStream principal.

A lo largo de este epígrafe aprenderemos, por tanto, cómo comenzar a escribir programas de Spark Streaming con DStreams. Pueden escribir programas de Spark Streaming en Scala, Java o Python.

Para poder simular un flujo de llegada de datos, y que este siempre sea idéntico, lo que vamos a plantear es generar un conjunto de datos que se guarda particionado dentro de una carpeta, otro proceso será el encargado de ir moviendo los datos desde la carpeta donde fueron generados, hasta la carpeta en la que serán consumidos por el programa Spark Streaming.

Los ejemplos serán propuestos para ser probados en un entorno local, el que quiera profundizar en algún aspecto, le animas a investigar Databricks Delta.

Un posible proceso de generación y movimiento de datos con una cierta cadencia en el tiempo, puede ser generado mediante las siguientes funciones:

```
1 import logging
2 from datetime import datetime, timedelta
3 from os import makedirs, path
4 from random import expovariate, normalvariate, randint, random, seed
5 from shutil import move, rmtree
6 from threading import Thread
```

Figura 41: Librerías necesarias

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
 9 def generador_datos(directorio_datos_crudos, directorio_datos_procesar, intervalo_llegada, tasa_llegada):
10     """
11     Genera un nuevo registro y mueve la partición de datos generados cuando corresponda
12     :param directorio_datos_crudos: ruta donde se dejan los registros generados en las particiones correspondientes
13     :param directorio_datos_procesar: ruta que el proceso Streaming usa como referencia
14     :param intervalo_llegada: intervalo en segundos en los que se genera un nuevo fragmento de datos
15     :param tasa_llegada: tasa de generación de los registros
16     """
17     logging.info('Comienza el proceso de generación de datos')
18 
19     # Generemos los directorios que son requeridos
20     logging.warn('Se fuerza el borrado de los directorios:\n\t{raw}\n\t{input}'
21                  .format(raw=directorio_datos_crudos, input=directorio_datos_procesar))
22     vaciar_directorio(directorio_datos_crudos)
23     vaciar_directorio(directorio_datos_procesar)
24 
25     # Fijamos la semilla para obtener siempre la misma simulación en los registros
26     seed(1234)
27 
28     # Guardamos el momento en el que comienza el proceso y la partición que ha sido procesada
29     inicio = int(datetime.now().strftime("%s"))
30     van = 0
31 
32     while True:
33         ahora = nuevo_registro(directorio_datos_crudos, inicio, intervalo_llegada, tasa_llegada)
34         # Se mueve la partición al directorio de ingesta
35         if (int(ahora.strftime("%s")) - inicio) / intervalo_llegada != van:
36             mover_particion(directorio_datos_crudos, directorio_datos_procesar, van)
37             van += 1
```

Figura 42: Controlador del flujo de generación

```
40 def vaciar_directorio(directorio):
41     """
42     Se eliminan todos lo que hayan en cierto directorio
43     :param directorio: ruta del sistema que va a ser vaciada
44     """
45     if path.exists(directorio):
46         rmtree(directorio)
47     makedirs(directorio)
```

Figura 43: Preparación de directorios

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
50 def mover_particion(directorio_datos_crudos, directorio_datos_procesar, van):
51     """
52     Mueve una partición del directorio donde son creados al que son consumidos por el proceso Streaming
53     :param directorio_datos_crudos: ruta donde se dejan los registros generados en las particiones correspondientes
54     :param directorio_datos_procesar: ruta que el proceso Streaming usa como referencia
55     :param van: particiones que van procesadas
56     """
57     fichero = path.join(directorio_datos_crudos, ''.join(['part_', str(van), '.csv']))
58     if path.exists(fichero):
59         logging.info('Se mueve al directorio de procesamiento la partición {part} de datos'.format(part=van))
60         move(fichero,
61              path.join(directorio_datos_procesar, ''.join(['part_', str(van), '.csv'])))
62     else:
63         logging.info('La partición {part} de datis no existía, no ha habido llegada de registros'
64                      .format(part=van))
```

Figura 44: Mover la partición correspondiente

```
67 def nuevo_registro(directorio_datos_crudos, inicio, intervalo_llegada, tasa_llegada):
68     """
69 
70     :param directorio_datos_crudos: ruta donde se dejan los registros generados en las particiones correspondientes
71     :param inicio: instante de tiempo en el que comienza el proceso de generación de datos
72     :param intervalo_llegada: intervalo en segundos en los que se genera un nuevo fragmento de datos
73     :param tasa_llegada: tasa de generación de los registros
74     :return: instante de tiempo en el que se genera el dato
75     """
76     # Se calcula el momento de llegada del nuevo registro
77     ahora = datetime.now()
78     momento = expovariate(1 / tasa_llegada)
79     llegada = ahora + timedelta(seconds=momento)
80     # Calculamos a la partición que correspondería el registro nuevo
81     particion = (int(llegada.strftime("%s")) - inicio) / intervalo_llegada
82     # Se añade un nuevo registro a la partición que se ha obtenido aleatoriamente
83     with open(path.join(directorio_datos_crudos, ''.join(['part_', str(particion), '.csv'])), 'a') as fichero:
84         fichero.write(','.join([
85             str(ahora),
86             str(llegada),
87             str(randint(0, 10)),
88             'A' if random() < 0.3 else 'B',
89             str(round(normalvariate(0, 100), 2)),
90             str(particion)]) + '\n')
91     return ahora
```

Figura 45: Generación de un nuevo registro

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
100 # Definimos las variables de trabajo
101 intervalo_consumo = 5
102 intervalo_llegada = 3
103 tasa_llegada = 75.0
104 directorio_datos_crudos = 'stream_raw'
105 directorio_datos_procesar = 'stream_input'
106 directorio_datos_procesados = 'stream_output'
107
108 vaciar_directorio(directorio_datos_procesados)
109
110 logging.info('El proceso Streaming consume datos en intervalos de {i_c} segundos, el proceso de volcado de datos se '
111              'realiza cada {i_l} segundos, que se encarga de mover la partición que corresponde de la generación de '
112              'los datos con una tasa de llegada de {t_l}'
113              .format(i_c=intervalo_consumo, i_l=intervalo_llegada, t_l=tasa_llegada))
114
115 # Generamos un hilo que será el encargado de generar los datos que van a ser procesados
116 hilo_generador_datos = Thread(target=generador_datos,
117                               args=(directorio_datos_crudos,
118                                     directorio_datos_procesar,
119                                     intervalo_llegada,
120                                     tasa_llegada), )
121 hilo_generador_datos.start()
```

Figura 46: Definición y lanzamiento del hilo generador de datos

Una vez lanzado este programa de generación tendremos, que con el intervalo de tiempo en segundos que se haya definido, un nuevo fichero que pasa del directorio donde se están generando los datos hacia el directorio en el que van se van a procesar.

```
1  from pyspark.sql import functions as F
2  from pyspark.sql import types as T
3
4  schema = T.StructType([
5      T.StructField('tiempo_creacion', T.TimestampType(), True),
6      T.StructField('tiempo_llegada', T.TimestampType(), True),
7      T.StructField('prioridad', T.IntegerType(), True),
8      T.StructField('demanda', T.StringType(), True),
9      T.StructField('retorno', T.FloatType(), True),
10     T.StructField('part', T.IntegerType(), True)])
11
12 df = (spark.readStream
13       .schema(schema)
14       .option('header', 'false')
15       .option('sep', ',')
16       .csv(directorio_datos_procesar))
```

Figura 47: Lectura del conjunto de datos desde el directorio indicado

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
18 (df
19  .filter(F.col('demanda') == 'A')
20  .writeStream
21  .format('csv')
22  .option('path', directorio_datos_procesados)
23  .option('header', 'false')
24  .option('sep', ',')
25  .option('checkpointLocation', 'tmp')
26  .start()
27  .awaitTermination())
```

Figura 48: Escritura tras la aplicación de una transformación

Trabajaremos todos nuestros ejemplos partiendo de una carga similar a la anterior, pero existen numerosas posibilidades que se pueden utilizar como fuentes de entrada de datos a Spark Streaming, además de la lectura desde un directorio de recepción de datos, destacan:

- Kafka: Sistema de mensajería distribuido, rápido y escalable.

  https://kafka.apache.org

- Flume: Sistema distribuido para recolectar, agregar y transferir datos de log.

  https://flume.apache.org

- Amazon Kinesis: Sistema de AWS parecido a Kafka.

  https://aws.amazon.com/en/kinesis

- Twitter: El API de la red social.

  https://dev.twitter.com/overview/documentation

- Zero MQ: Sistema de mensajería distribuido.

  http://zeromq.org

- MQTT: Sistema de mensajería ligero.

  http://mqtt.org

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Otra forma de crear un DStream es mediante datos de transmisión desde una fuente TCP, para lo cual es necesario especificar un nombre de host y puerto a la función *socketTextStream*. Esto puede ser definido tras definir un *StreamingContext*.

Una vez definido el procedimiento con el que se va a construir el *DStream*, es necesario añadir la lógica de cómo va a ser tratada la información recibida. En el primer ejemplo, se aplicaba una transformación de filtrado de un conjunto de datos estructurados, por el contrario, si recibiéramos una serie de palabras sin una estructura y quisiéramos proceder a su recuento, podríamos realizarlo mediante las siguientes instrucciones:

```
1  from __future__ import print_function
2  import sys
3  from pyspark import SparkContext
4  from pyspark.streaming import StreamingContext
5
6  sc = SparkContext(appName="StreamingNetworkWordCount")
7  ssc = StreamingContext(sc, 1)
8
9  lines = ssc.socketTextStream("hostname", "port")
10 counts = (lines
11           .flatMap(lambda line: line.split(" "))
12           .map(lambda word: (word, 1))
13           .reduceByKey(lambda a, b: a+b))
14 counts.pprint()
15
16 ssc.start()
17 ssc.awaitTermination()
```

Figura 49: Recuento de palabras que son recibidas mediante TCP

En ambos ejemplos aparecen dos sentencias que aún no han sido explicadas, que son las dos últimas que aparecen en la imagen anterior. La primera sirve para iniciar la computación streaming, mientras que la siguiente es definida para que el hilo principal de la aplicación espere la finalización del resto de hilos en ejecución. Para detener Spark Streaming se usa la sentencia:

```
18 ssc.stop(False)
```

Figura 50: Sentencia para detener Spark Streaming

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Una vez que se dispone de un *DStream*, podemos aplicar diferentes transformaciones de las disponibles, algunas de las que se pueden realizar son:

map(*func*)	Devuelva un nuevo DStream pasando cada elemento del origen DStream a través de una función (*func*).
flatMap(*func*)	Similar al *map*, pero cada elemento de entrada se puede asignar a 0 o más elementos de salida.
filter(*func*)	Devuelva un nuevo DStream seleccionando solo los registros del origen DStream en el que *func* devuelve verdadero.
repartition(*numPartitions*)	Cambia el nivel de paralelismo en este DStream creando más o menos particiones.
union(*otherStream*)	Devuelva un nuevo DStream que contenga la unión de los elementos del original y del *otherDStream*.
count()	Devuelva un nuevo DStream de RDD de un solo elemento contando el número de elementos en cada RDD del DStream de origen.
reduce(*func*)	Devuelva un nuevo DStream de RDD de un solo elemento agregando los elementos en cada RDD del DStream de origen utilizando una función *func* (que toma dos argumentos y devuelve uno). La función debe ser asociativa y conmutativa para que se pueda calcular en paralelo.
join(*otherStream*, [*numTasks*])	Cuando se le solicitan dos DStreams de (K, V) y (K, W) pares, devuelva un nuevo DStream de (K, (V, W)) pares con todos los pares de elementos para cada clave.

Tabla 9: Algunas de las transformaciones disponibles para un DStream

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Spark Streaming también proporciona cálculos en ventanas de *DStream*, que le permiten aplicar transformaciones en una ventana deslizante de datos. La siguiente figura ilustra esta ventana deslizante, donde se muestra que además es necesario definir la longitud de la ventana (3 en la figura) y el intervalo de deslizamiento en el que se realiza la operación de la ventana (2 en la figura), estos dos parámetros que son necesarios definir deben ser múltiplos del intervalo de lote de la fuente *DStream* (1 en la figura).

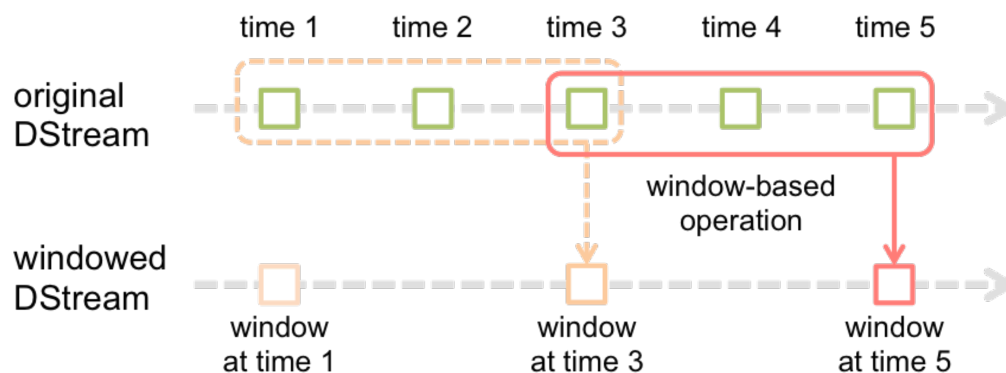


Figura 51: Cálculos sobre una ventanas de DStream[118]

Como resumen podríamos remarcar que:

- Spark Streaming aplica las funciones de procesamiento por lotes de Spark a los datos en tiempo real utilizando batch de información.

- Spark puede ingestar datos de sistemas de archivos y conexiones de socket TCP / IP, pero también de otros sistemas distribuidos, como Kafka, Flume, Twitter y Amazon Kinesis.

- La duración del batch se establece al inicializar un contexto de Spark Streaming. Eso determina como será discretizado el tiempo en el que los datos que entran se dividirán y empaquetarán como un RDD.

  Tiene una gran influencia en el rendimiento de la aplicación y debe encontrarse el

[118] Imagen extraída de https://spark.apache.org

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

equilibrio adecuado.

En la página de la interfaz de usuario web de Spark se muestran una serie de gráficos útiles que pueden ayudar a determina la duración ideal del batch de ingesta de información.

- DStream:
    - Es la abstracción básica en Spark Streaming, que representa una secuencia de RDDs, creados periódicamente desde el flujo de entrada.
    - Tiene métodos que los transforman en otros DStreams. Puedes usar esos métodos para:
        - Filtrar
        - Asignar
        - Reducir
        - Combinar y unir diferentes DStreams.

© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

Anexo I –Trabajar con los resultados de fútbol

Aplicaremos el modelo de regresión lineal apoyándonos en que es un modelo dentro de los Modelos Lienales Generalizados (GLM), una posible forma de computarlo sería:

```
1  # Ejemplo ML: Regresión Lineal
2
3  from pyspark.ml import Pipeline
4  from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, VectorAssembler
5  from pyspark.ml.regression import GeneralizedLinearRegression
6
7  # Generamos una nueva caracteristica, TotalFaltas, para estudiar el modelo conjuntamente con el Total de Tarjetas que tenemos
8  df_modelo = (df_normalizado
9               .withColumn("TotalFaltas", df_normalizado.HF+df_normalizado.AF)
10             .withColumn("label", F.col("TotalFaltas")))
11
12 #
13 assembler = VectorAssembler(
14     inputCols=["TotalTarjetas"],
15     outputCol="features")
16
17 pipeline = Pipeline(stages=[assembler])
18 df_custom = pipeline.fit(df_modelo).transform(df_modelo)
19
20 glr = GeneralizedLinearRegression(family="gaussian", link="identity", maxIter=10, regParam=0.3)
21
22 # Entrenamos el modelo
23 model = glr.fit(df_custom)
24
25 # Imprimimos los coeficientes y el término independiente (intercept) para el el modelo de regresión lineal
26 print("Coefficients: " + str(model.coefficients))
27 print("Intercept: " + str(model.intercept))
28
29 # Resumen del modelo ejecutado sobre el conjunto de entrenamiento
30 summary = model.summary
31 print("Coefficient Standard Errors: " + str(summary.coefficientStandardErrors))
32 print("T Values: " + str(summary.tValues))
33 print("P Values: " + str(summary.pValues))
34 print("Dispersion: " + str(summary.dispersion))
35 print("Null Deviance: " + str(summary.nullDeviance))
36 print("Residual Degree Of Freedom Null: " + str(summary.residualDegreeOfFreedomNull))
37 print("Deviance: " + str(summary.deviance))
38 print("Residual Degree Of Freedom: " + str(summary.residualDegreeOfFreedom))
39 print("AIC: " + str(summary.aic))
40 print("Deviance Residuals: ")
41 summary.residuals().show()
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018

```
Coefficients: [0.86328492528]
Intercept: 23.19321688893625
Coefficient Standard Errors: [0.10887365449383225, 0.63685939281761
T Values: [7.929236226098213, 36.41811230313156]
P Values: [2.5091040356528538e-14, 0.0]
Dispersion: 32.08067765950925
Null Deviance: 14237.73421052639
Residual Degree Of Freedom Null: 379
Deviance: 12126.496155294497
Residual Degree Of Freedom: 378
AIC: 2400.324488130633
Deviance Residuals:
+-------------------+
|  devianceResiduals|
+-------------------+
|  7.943498185783557|
|  8.763788634102383|
|-3.6463565900570316|
|  0.763788634102383|
| -3.236211365897617|
| 3.4903584846627744|
```


© José Miguel Hernandez Izquierdo / Mario Villaizán Vallelado - 2018