

# Taller 2

## Unidad 2 - Ecosistema Hadoop

Miércoles 17 de enero de 2018 - 20:30 - 21:45


### Introducción

En este taller usaremos herramientas comunes de Machine Learning en Big Data usando la librería Spark MLlib. El objetivo es familiarizarse con las herramientas y ver su potencial con ejemplos simples. Usaremos la máquina virtual de Cloudera para el desarrollo del taller, en su versión 5.10. Deben tener disponible un notebook con la máquina virtual instalada.

Las respuestas a las preguntas debe enviarlas al correo [ialillo@uc.cl](mailto:ialillo@uc.cl) con el subject "Big Data Taller 1 <nombre>" al finalizar la actividad.

### Máquina virtual de Cloudera

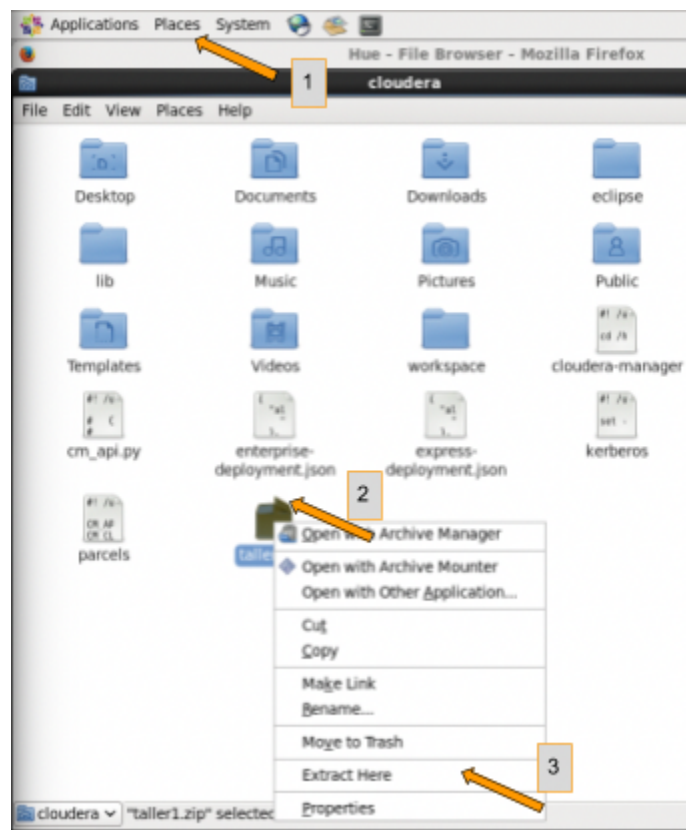
La máquina virtual de Cloudera se basa en una distribución de Linux (CentOS 6.7). Le recomendamos que asigne al menos 4 Gb de RAM y 2 procesadores antes de inicializar la máquina virtual sobre el host descargado (VMWare o VirtualBox) y verificar su correcto funcionamiento. Como primer paso luego de inicializar la máquina virtual realizaremos algunos ajustes en la configuración de la máquina para mejorar su usabilidad a la largo de la actividad:

- En la máquina virtual (CentOs), ajuste el layout del teclado a su teclado real (el layout default es en inglés). Para esto, dentro de la máquina virtual vaya a System->Preferences->Keyboard, y en la pestaña "Layouts" presione "Add". En gran parte de los notebooks, bastará con seleccionar la pestaña "By country", y seleccionar "Chile" en el combo-box "Country". Presione "Add" para agregar el layout. Borre el anterior layout seleccionándolo y presionando "Remove".
- Al descargar archivos desde internet, probablemente queden almacenados en la carpeta `/home/cloudera/Downloads`. Puede copiar los archivos descargados a su carpeta raíz (`/home/cloudera`) usando un terminal (puede usar el ícono  para abrir un terminal) y escribir el siguiente comando:

```
$ mv /home/cloudera/Downloads/* /home/cloudera
```

Otra alternativa es abrir un explorador de archivos (menú “Places”), copiar el archivo correspondiente en la carpeta “Downloads” y pegarlo en “Home Folder”.

Los archivos para este taller se encuentran en la página del curso ([ep.ingenieriauc.cl](http://ep.ingenieriauc.cl)). Descargue los archivos correspondientes al Taller 2 (taller2.zip). Guárdelos en /home/cloudera (Home Folder). Para descomprimir el archivo, vaya a al menú “Places”, elija “Home Folder”, y sobre el ícono de “taller2.zip” despliegue el menú contextual, para elegir después “Extract Here”.



Para poder ejecutar los comandos de esta actividad tiene 2 alternativas:

- Utilizar IPython Notebook (**Opción recomendada**)
- Correr los scripts directamente utilizando `spark-submit`

### IPython Notebook

Para utilizar IPython Notebook, en el terminal debes pararte sobre tu home (/home/cloudera/) y correr el script para iniciar IPython Notebook, con los siguientes comandos:

```
$ cd
$ ./pyspark_jupyter.sh
```

Tu navegador se abrirá automáticamente y te mostrará todos los archivos disponibles, debes fijarte en tener los archivos adecuados tal y como se muestra en la siguiente imagen. Luego para cada actividad seleccionar el notebook correspondiente.

jupyter		Logout
<input type="checkbox"/>	taller2_association.ipynb	3 minutos ago
<input type="checkbox"/>	taller2_classification.ipynb	13 minutos ago
<input type="checkbox"/>	taller2_kmeans.ipynb	2 minutos ago

## Spark-Submit

Las instrucciones para utilizar este método las encontrarás en cada una de las secciones.

## Parte 1: Clustering con k-means

En esta parte usaremos la base de datos Iris (<https://archive.ics.uci.edu/ml/datasets/iris>). Esta base de datos es muy pequeña, y consiste en 150 registros de 4 características. El número de clases reales es tres (Iris Setosa, Iris Versicolour e Iris Virginica). Los atributos de cada registro son:

1. Largo del sépalos en cm
2. Ancho del sépalos en cm
3. Largo del pétalo en cm
4. Ancho del pétalo en cm

Usando el algoritmo k-means implementado en el módulo MLlib de Spark, haremos un clustering ignorando la clase (tipo de flor), que después compararemos para ver si encontramos clusters semánticos (es decir, con significado interpretable).

Una muestra de la base de datos es el siguiente, donde los atributos están separados por comas, y al último se muestra la clase:

```
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
4.6,3.1,1.5,0.2,Iris-setosa
...
```

Usaremos funciones de Spark y Spark MLlib para generar los clusters.

Pyspark genera muchos logs en su ejecución. Por lo mismo, guardaremos las salidas importantes en un archivo de texto `output_kmeans.txt`.

Si estás utilizando IPython Notebook debes abrir el notebook `taller2_kmeans.ipynb` y correr su contenido. Luego puedes saltarte hasta la sección de preguntas.

Si no estás ocupando IPython Notebook, los comandos que se ejecutarán en esta parte están contenidos en el script `taller2_kmeans.py`:

```
from pyspark import SparkContext
sc = SparkContext()
sc.setCheckpointDir('/user/cloudera')
```

```

import os
import numpy as np

fid = open('output_kmeans.txt','w')

# lectura de datos de ratings
iris_data = sc.textFile('file:///home/cloudera/iris.data')

# obtenemos los atributos
iris_attrib_raw = iris_data.map(lambda line: np.array([l for l in
line.split(",")[:-1]]))

# solo usamos los datos validos
iris_attrib = iris_attrib_raw.filter(lambda x: len(x)==4)

# obtenemos la clase, y la guardamos en el vector iris_data con c
iris_class = iris_data.map(lambda line: line.split(",")).filter(lambda x:
len(x)==5).map(lambda l: l[4]).collect()

# Entreno k-means

# K: numero de clusters que encontrara kmeans
K = 4

from pyspark.mllib.clustering import KMeans
model = KMeans.train(iris_attrib, k=K, maxIterations = 100)

# Guardo los centros de los clusters
centers = model.clusterCenters

fid.write('Centros de clusters:\n\n')
for c in centers:
    fid.write(str(c) + '\n')

# encuentro el label predicho y lo comparo con la clase real
labels = model.predict(iris_attrib).collect()
fid.write('\n\nLabels (real - predicted)\n\n')
for l,c in zip(labels, iris_class):
    fid.write(str(l) + ' - ' + str(c) + '\n')
fid.close()

```

La línea “K = 4” define el número de clusters que deberá encontrar el algoritmo de k-means.

Para ejecutar el script, en un terminal ejecute

```
$ spark-submit taller2_kmeans.py
```

Una vez termine de correr el código, podrá ver el resultado del clustering (centros de clusters y vectores de predicción y clase real) en el archivo output\_kmeans.txt.

## Preguntas

- Interprete semánticamente los centros de los clusters generados con  $K=4$ .
- Ejecute el script cambiando a  $K=2$ . ¿Qué sucede con la predicción, en relación a las clases reales?
- Ejecute el script cambiando a  $K=3$ , que es igual al número de clases reales. ¿Encuentra una relación entre los clusters generados y las clases reales?
- En un algoritmo de entrenamiento no supervisado como k-means, no se dispone a priori de una separación de los datos en clases, y se trata de encontrar grupos de elementos similares entre sí. Comente con sus palabras alternativas de elegir cuántos clusters usaría (el valor de  $K$ ) en un caso real donde sólo disponga de atributos.
- ¿Cuál flor, de los tres tipos, es más simple de identificar de las otras dos?

## Parte 2: Reglas de asociación

Para esta parte encontraremos reglas de asociación entre atributos de una base de datos relacionada a canastas de compras de un supermercado. Usará la base de datos `grocery.csv`, que contiene la categoría del artículo de cada canasta.

Usaremos el algoritmo FP-Growth, que es comúnmente utilizado para estos fines debido a su eficiencia, y además permite una implementación paralela por lo que es apto para datos distribuidos.

Algunos ejemplos de canastas de compra se muestran a continuación (los elementos son separados por comas ','):

```
citrus fruit,semi-finished bread,margarine,ready soups
tropical fruit,yogurt,coffee
whole milk
pip fruit,yogurt,cream cheese ,meat spreads
other vegetables,whole milk,condensed milk,long life bakery product
whole milk,butter,yogurt,rice,abrasive cleaner
rolls/buns
other vegetables,UHT-milk,rolls/buns,bottled beer,liquor (appetizer)
pot plants
whole milk,cereals
...
```

Usaremos funciones de Spark y Spark MLlib para generar las reglas de asociación más comunes entre dos o más categorías de productos..

Pyspark genera muchos logs en su ejecución. Por lo mismo, se guardan las salidas importantes en un archivo de texto `output_association.txt`.

Si estás utilizando IPython Notebook debes abrir el notebook `taller2_association.ipynb` y correr su contenido. Luego puedes saltarte hasta la sección de preguntas.

Si no estás ocupando IPython Notebook, los comandos que ejecutará en esta parte están contenidos en el script `taller2_association.py`:

```
from pyspark import SparkContext
sc = SparkContext()
sc.setCheckpointDir('/user/cloudera')

# abro los datos, los separo en header y data
fid = open('groceries.csv')
data = [f.replace('\n','').replace('\r','').split(",") for f in fid.readlines()]

# creo un RDD de Spark para manejar los datos
rdd_data = sc.parallelize(data)

# importo el modelo de FP-Growth
from pyspark.mllib.fpm import FPGrowth

model = FPGrowth.train(rdd_data, minSupport=0.01, numPartitions=10)
result = model.freqItemsets().collect()

min_num_items = 2

sorted_filtered_result = sorted([(fi[0], fi[1]) \
    for fi in result if len(fi[0])>=min_num_items], \
    key = lambda x:x[1], reverse=True)

fid = open('output_association.txt','w')
for fi in sorted_filtered_result:
    fid.write(str(fi) + '\n')
fid.close()
```

Para ejecutar el algoritmo, abra una terminal y ejecute:

```
$ spark-submit taller2_association.py
```

Al ejecutar el algoritmo, se genera en el archivo `output_association.txt` lineas similares a las siguientes:

```
([u'other vegetables', u'whole milk'], 736)
([u'rolls/buns', u'whole milk'], 557)
([u'yogurt', u'whole milk'], 551)
([u'root vegetables', u'whole milk'], 481)
([u'root vegetables', u'other vegetables'], 466)
...
```

El significado de cada línea es qué tan frecuentemente en el set de transacciones se generan patrones frecuentes. Por ejemplo, el patrón más común es que usuarios compren conjuntamente de categorías “other vegetables” y “whole milk”.

## Preguntas

- Cambie el parámetro `min_num_items` del valor 2 al 3 y ejecute nuevamente el código. Comente los resultados de `output_association.txt`, en términos de número de patrones encontrados.
- ¿Para qué cree que sería de utilidad encontrar patrones comunes dentro de una canasta de compra?
- Explique en sus palabras cómo podría usar el algoritmo FP-Growth en un caso donde en vez de ítems dentro de un supermercado analiza las páginas web visitadas por usuarios.
- Continuando la pregunta anterior, proponga un método (en palabras) para encontrar grupos de usuarios similares usando FP-Growth y k-means.

## Parte 3: Clasificación

En esta parte entrenaremos clasificadores para un set de datos supervisado, que corresponde a datos de clientes bancarios. La tarea es crear un modelo predictor que trabaje con características de los clientes. Entre estas características están edad (age), tipo de trabajo (job), estado civil (married), entre otras características. Puede revisar el set de entrenamiento completo en archivo `bank-additional-full.csv`. Como se trata de un modelo supervisado, sabemos de antemano qué es lo que se quiere predecir; en este caso, si el cliente toma (yes) o no toma (no) una cuenta de ahorro. Note que la categoría que se desea predecir se presenta como el último dato de cada registro.

Usaremos dos tipos de clasificadores:

- Naïve Bayes
- Regresión Logística

En un modelo no supervisado, de tipo clustering, no se tiene una salida o semántica de los atributos (en la parte 1, sólo la usamos para comparar cómo se comportaría un modelo no supervisado). Cuando se dispone, es necesario obtener métricas de rendimiento del modelo predictor sobre datos que no se hayan presentado durante el entrenamiento. Estos datos están disponibles en el archivo `bank-additional.csv`.

Usaremos la métrica de exactitud (accuracy), que mide cuántas instancias de prueba son correctamente clasificadas con respecto al total de instancias de prueba.

Si estás utilizando IPython Notebook debes abrir el notebook `taller2_classification.ipynb` y correr su contenido. Luego puedes saltarte hasta la sección de preguntas.

Si no estás ocupando IPython Notebook, los comandos que ejecutará en esta parte están contenidos en el script `taller2_classification.py`:

Para ejecutar los modelos, debe ejecutar en un terminal:

```
$ spark-submit taller2_classification.py
```

## Preguntas

- Interprete el archivo `bank-additional-full.csv`.
- Muestre las matrices de confusión y el accuracy obtenido con ambos clasificadores. Los valores están dentro de los logs entregados por el programa, cerca del final.
- ¿Conviene usar la métrica accuracy cuando las clases positiva y negativa están muy desbalanceadas? Por ejemplo, cuando un 5% de los datos de entrenamiento corresponden a la clase positiva y 95% a la negativa.
- Métricas de rendimiento que son menos sensibles al desbalance de datos en relación a las clases son precision ( $TP/(TP+FP)$ ) y recall ( $TP/(TP+FN)$ ). Modifique el código para encontrar y mostrar estas métricas. Muestre los valores de precision y recall para ambos clasificadores.
- Comente sus resultados en relación a qué métrica usaría finalmente para medir el rendimiento de los clasificadores.
- Observando las métricas de rendimiento, ¿cuál clasificador cree que es mejor para la tarea de predecir la contratación de un producto bancario? Justifique.
- Opine sobre la validez de transformar datos categóricos a numéricos para entrenar un clasificador.