Artículo
Alberto Fuentes · 25 sep, 2019 Lectura de 4 min

Algoritmo de agrupamiento K-medias sobre datos en IRIS

¡Hola a tod@s!

K-Medias es uno de los algoritmos de aprendizaje no supervisado más simples para resolver el problema de agrupamiento. Este problema consiste en formar grupos de objetos con características similares. Por ejemplo, si tenemos una imagen de una pelota roja sobre césped verde, K-Medias separará los pixels de la imagen en dos grupos (*clusters*): un grupo con los pixels que forman la pelota, y otro grupo con los pixels del césped.

Repasaremos un artículo publicado por Niyaz Khafizov en el que implementaremos un ejemplo donde cargaremos un conjunto de datos en IRIS y ejecutaremos el algoritmo *K-Medias* utilizando *Apache Zeppelin* con el conector *Spark*. Utilizaremos *InterSystems IRIS*, *Apache Zeppelin* 0.8.0 y *python*.

Conjunto de datos flor Iris

El conjunto de datos flor Iris contiene diferentes características de la flor Iris de 3 especies distintas. Las especies pueden ser Iris-setosa, Iris-versicolor e Iris-virginica. Cada flor posee 5 características: Longitud del pétalo (*Petal Length*), Anchura del pétalo (*Petal Width*), Longitud del sépalo (*Sepal Length*), Anchura del sépalo (*Sepal Width*) y Especie (*Species*).

Requerimientos

En este otro artículo comentamos cómo conectar InterSystems IRIS con Apache Zeppelin / Spark: Guía rápida para conectar Apache Spark y Apache Zeppelin con InterSystems IRIS.

Necesitaremos también tener instalado **python3** junto con **python3-pip** (el gestor de paquetes de python). Podéis encontrar guías de instalación para Linux, macOS y Windows.

Una vez instalado *python3* y *python3-pip*, instalaremos **pyspark** (para poder interactuar con *spark* desde *python*).
pip3 install pyspark

En *Apache Zeppelin* hemos de cambiar la configuración **zeppelin.pyspark.python** del intérprete de *Spark* y especificar la ruta nuestro intérprete (podemos obtenerla con *which python3* por ejemplo)

zeppelin.pyspark.python	/usr/bin/python3

Finalmente, creamos una nueva nota en Zeppelin y si todo ha ido correctamente podremos ejecutar sin error lo siguiente:

```
%pyspark
import sys
print(sys.version)
```

```
%pyspark
import sys
print(sys.version)
```

```
3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609]
```

Took 1 sec. Last updated by anonymous at July 03 2019, 3:21:43 PM.

Agrupamiento de datos

En primer lugar, cargamos los datos que ya tenemos almacenados en IRIS

```
%pyspark
dataFrame=spark.read.format("com.intersystems.spark").\
option("url", "IRIS://localhost:51773/NEWSAMPLE").option("user", "dev").\
option("password", "123").\
option("dbtable", "DataMining.IrisDataset").load()

dataFrame.show()
```

```
+---+-----------+----------+-----------+----------+-----------+
| ID|PetalLength|PetalWidth|SepalLength|SepalWidth|    Species|
+---+-----------+----------+-----------+----------+-----------+
|  1|        1.4|       0.2|        5.1|       3.5|Iris-setosa|
|  2|        1.4|       0.2|        4.9|       3.0|Iris-setosa|
|  3|        1.3|       0.2|        4.7|       3.2|Iris-setosa|
|  4|        1.5|       0.2|        4.6|       3.1|Iris-setosa|
|  5|        1.4|       0.2|        5.0|       3.6|Iris-setosa|
|  6|        1.7|       0.4|        5.4|       3.9|Iris-setosa|
|  7|        1.4|       0.3|        4.6|       3.4|Iris-setosa|
|  8|        1.5|       0.2|        5.0|       3.4|Iris-setosa|
|  9|        1.4|       0.2|        4.4|       2.9|Iris-setosa|
| 10|        1.5|       0.1|        4.9|       3.1|Iris-setosa|
| 11|        1.5|       0.2|        5.4|       3.7|Iris-setosa|
| 12|        1.6|       0.2|        4.8|       3.4|Iris-setosa|
| 13|        1.4|       0.1|        4.8|       3.0|Iris-setosa|
| 14|        1.1|       0.1|        4.3|       3.0|Iris-setosa|
| 15|        1.2|       0.2|        5.8|       4.0|Iris-setosa|
```

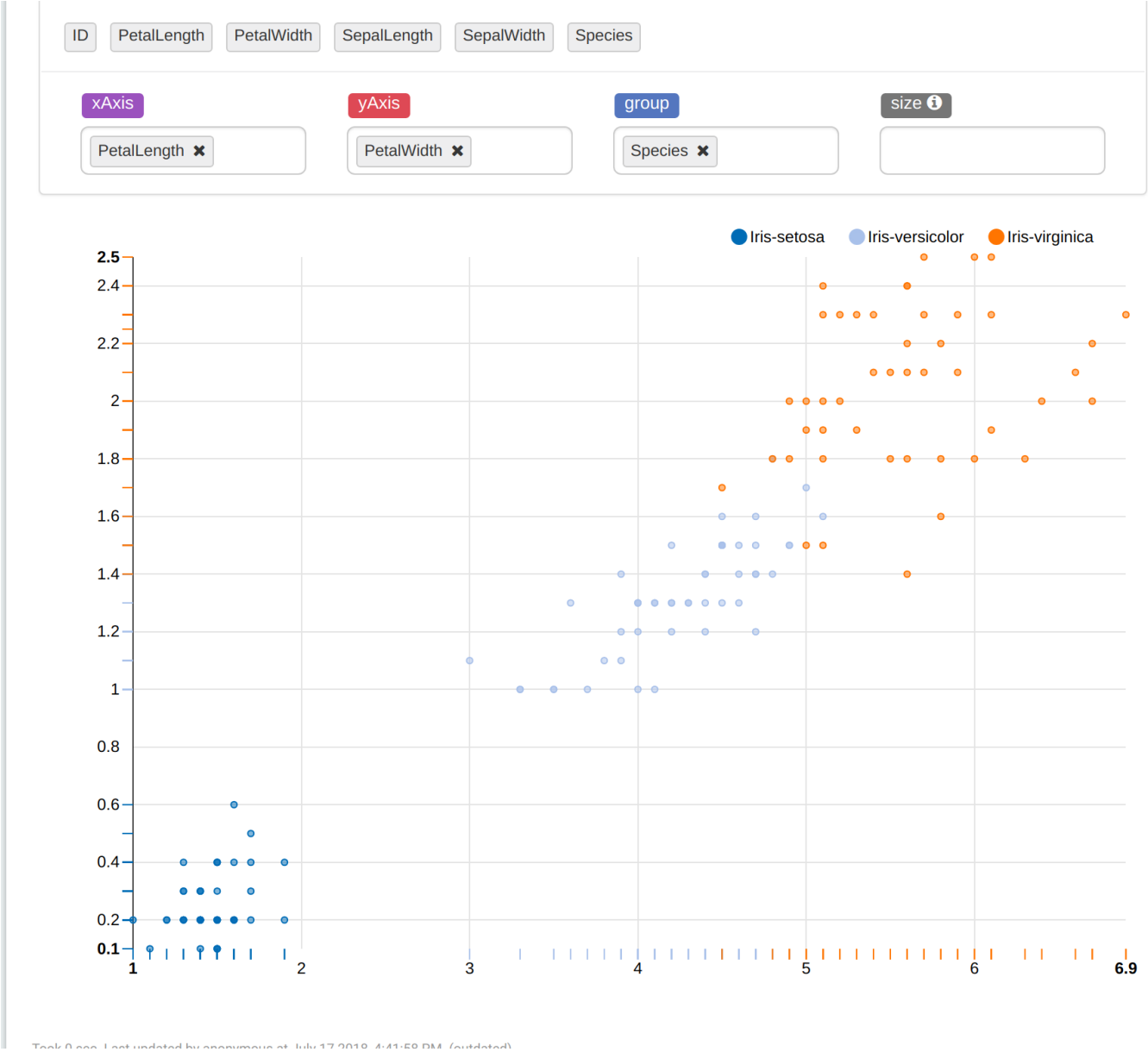
Para ver los diferentes tipos de especie que están cargadas, podemos ejecutar lo siguiente en un nuevo párrafo de Zeppelin:
%pyspark
dataFrame.select("Species").show(150)

Antes de continuar, sería una buena idea echarle un vistazo a los datos que tenemos. En otro párrafo podemos ejecutar:
%pyspark
z.show(dataFrame)

Nos aparecen las 5 características de la flor que almacenamos en la tabla: **PetalLength**, **PetalWidth**, **SepalLength**, **SepalWidth** y **Species**.

ID	PetalLength	PetalWidth	SepalLength	SepalWidth	Species
1	1.4	0.2	5.1	3.5	Iris-setosa
2	1.4	0.2	4.9	3	Iris-setosa
3	1.3	0.2	4.7	3.2	Iris-setosa
4	1.5	0.2	4.6	3.1	Iris-setosa
5	1.4	0.2	5	3.6	Iris-setosa
6	1.7	0.4	5.4	3.9	Iris-setosa
7	1.4	0.3	4.6	3.4	Iris-setosa
8	1.5	0.2	5	3.4	Iris-setosa

Escogemos a continuación la opción de gráfica **Scatter Chart**. De esta forma podemos apreciar los diferentes grupos. No podemos ver una imagen completa porque son vectores de 4-dimensiones, pero para ver los grupos desde otra perspectiva podemos cambiar los parámetros de **xAxis** y **yAxis**.



El objetivo es predecir la especie de la flor utilizando sus características. En un nuevo párrafo escribimos lo siguiente:
%pyspark
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols = ["PetalLength", "PetalWidth", "SepalLength", "SepalWidth"], outputCol="features") // it makes a vector with 4 parameters mentioned in **inputCols** and name it as **outputCol**.

irisFeatures = assembler.transform(dataFrame) // this will add to the table **outputCol** column with vectors.
irisFeatures.show(5)

A continuación, en otro párrafo introducimos lo siguiente:
%pyspark
from pyspark.ml.clustering import KMeans

(trainingData, testData) = irisFeatures.randomSplit([0.7, 0.3]) // split data into two parts randomly

kmeans = KMeans().setK(3).setSeed(101010) // KMeans model with 3 clusters. setSeed makes reproducible results.
model = kmeans.fit(trainingData) // train kmeans model

transformed = model.transform(trainingData) // add a new column to the table with predicted results
transformed.show(150)

```
%pyspark
from pyspark.ml.clustering import KMeans

(trainingData, testData) = irisFeatures.randomSplit([0.7, 0.3])
kmeans = KMeans().setK(3).setSeed(101010)
model = kmeans.fit(trainingData)

transformed = model.transform(trainingData)
transformed.show(150)
```

```
+---+-----------+----------+-----------+----------+---------------+-----------------+----------+
| ID|PetalLength|PetalWidth|SepalLength|SepalWidth|        Species|         features|prediction|
+---+-----------+----------+-----------+----------+---------------+-----------------+----------+
|  3|        1.3|       0.2|        4.7|       3.2|    Iris-setosa|[1.3,0.2,4.7,3.2]|         0|
|  4|        1.5|       0.2|        4.6|       3.1|    Iris-setosa|[1.5,0.2,4.6,3.1]|         0|
|  5|        1.4|       0.2|        5.0|       3.6|    Iris-setosa|[1.4,0.2,5.0,3.6]|         0|
|  7|        1.4|       0.3|        4.6|       3.4|    Iris-setosa|[1.4,0.3,4.6,3.4]|         0|
|  9|        1.4|       0.2|        4.4|       2.9|    Iris-setosa|[1.4,0.2,4.4,2.9]|         0|
| 10|        1.5|       0.1|        4.9|       3.1|    Iris-setosa|[1.5,0.1,4.9,3.1]|         0|
| 11|        1.5|       0.2|        5.4|       3.7|    Iris-setosa|[1.5,0.2,5.4,3.7]|         0|
| 12|        1.6|       0.2|        4.8|       3.4|    Iris-setosa|[1.6,0.2,4.8,3.4]|         0|
| 15|        1.2|       0.2|        5.8|       4.0|    Iris-setosa|[1.2,0.2,5.8,4.0]|         0|
| 16|        1.5|       0.4|        5.7|       4.4|    Iris-setosa|[1.5,0.4,5.7,4.4]|         0|
| 17|        1.3|       0.4|        5.4|       3.9|    Iris-setosa|[1.3,0.4,5.4,3.9]|         0|
| 18|        1.4|       0.3|        5.1|       3.5|    Iris-setosa|[1.4,0.3,5.1,3.5]|         0|
| 19|        1.7|       0.3|        5.7|       3.8|    Iris-setosa|[1.7,0.3,5.7,3.8]|         0|
| 22|        1.5|       0.4|        5.1|       3.7|    Iris-setosa|[1.5,0.4,5.1,3.7]|         0|
| 24|        1.7|       0.5|        5.1|       3.3|    Iris-setosa|[1.7,0.5,5.1,3.3]|         0|
```

Utilizamos el modelo en nuestros datos de prueba:

```
%pyspark
predictions = model.transform(testData)
predictions.show(151)
```

Y evaluamos cómo de acertado ha sido el modelo:

```
%pyspark

SpeciesAndPreds = predictions.select("Species", "prediction").collect()

def getCluster(specie):
  if specie == "Iris-setosa":
  return 0
  elif specie == "Iris-versicolor":
  return 1
  else:
```

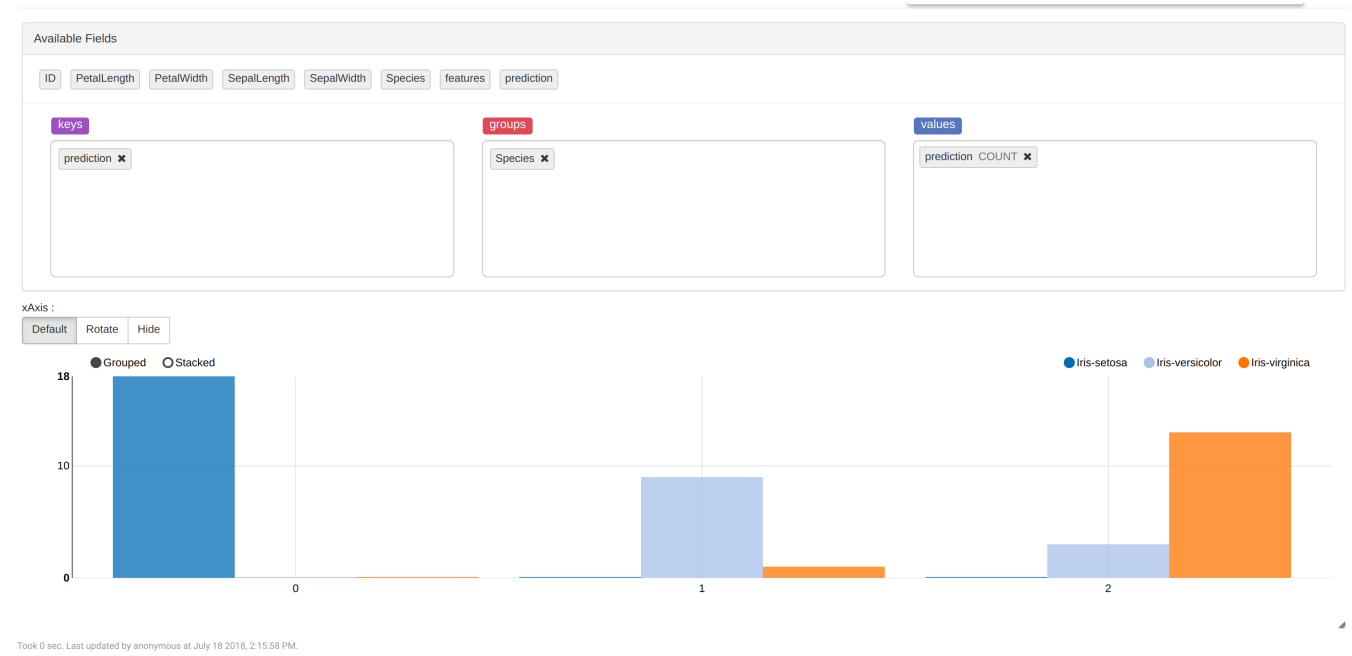
```
  return 2

def getAccuracy(flowers):
  counter = 0;
  for flower in flowers:
  if getCluster(flower[0]) == flower[1]:
  counter += 1
  return counter / len(flowers)

accuracy = getAccuracy(SpeciesAndPreds)
print("accuracy is " + str(accuracy))

// My result is 0.9090909090909091
```

Para ver cuántas flores hay en cada grupo, podemos ejecutar **z.show(predictions)** y escoger el tipo de gráfica **Bar Chart**:



Conclusión

Hemos creado un modelo que predice la especie de datos de flores iris almacenadas en InterSystems IRIS :) con una precisión bastante buena (> 90%).

También hemos comprobado que el grupo "Iris-setosa" puede separarse utilizando el algoritmo K-Medias pero "Iris-virginica" y "Iris-versicolor" no. Así que en resumen, sería muy útil utilizar alguna otra idea para mejorar la precisión del modelo.

#Inteligencia Artificial #Analítica #Machine learning #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/algoritmo-de-agrupamiento-k-medias-sobre-datos-en-iris