Análisis de regresión en Spark

Ahora hemos visto que el análisis exploratorio de datos numérico nos ayuda a establecer los estadísticos de los datos, en especial la variabilidad. También nos ayuda a encontrar relaciones entre variables por medio de las tablas de correlación.

Estas relaciones se pueden confirmar a través del análisis de datos exploratorio gráfico, al crear gráficos de dispersión en 2 y 3 dimensiones. Lo encontrado en el análisis exploratorio numérico se puede confirmar por medio de histogramas y gráficos de caja.

Este análisis te ayudará a determinar cuáles son las variables que debes incluir en tus modelos de predicción. ¿Porqué es importante esto?

Por un fenómeno llamado *"la explosión de la dimensionalidad"*. Este fenómeno indica que conforme se van agregando dimensiones a un modelo, el número de muestras que se requieren comienza a crecer exponencialmente. Si no es posible aumentar las muestras de forma correspondiente, el modelo perderá precisión. Por eso es importante utilizar la menor cantidad de variables dependientes (*features o atributos*) que efectivamente sean predictoras de la variable de interés.

Una vez determinado esto, lo que se hace es crear los modelos de predicción. En esta sección, mostraremos cómo crear modelos de regresión usando diversos algoritmos y técnicas como regresión lineal, bosques de regresión mejorados y regresión de árboles de gradiente mejorado.

¿Cómo la regresión es la técnica de machine learning que más se utiliza para variables continuas, crees que este tipo de regresión te puede servir para el proyecto que vas a realizar en tu organización?

Para crear modelos de machine learning se debe seguir varios pasos. Primero se leen los datos (*y si es necesario se transforman para que puedan ser alimentados a los algoritmos*), después se separa un conjunto de entrenamiento del modelo y un conjunto de prueba, luego se crea el modelo y luego éste se evalúa.

Primero leeremos un conjunto de datos y separaremos los datos en conjunto de datos de entrenamiento y de prueba.

El Conjunto de Datos Boston[TEVG1] Housing

El conjunto de datos Boston Housing consiste en un conjunto de información obtenida del servicio de censos en los estados unidos relacionados con las habitaciones en la ciudad de Boston. El conjunto de datos contiene variables sobre medio ambiente, población y vivienda que pueden explicar el precio medio de las casas. El conjunto de datos tiene las siguientes variables:

CRIM – la tasa de criminalidad per cápita por población
ZN – porcentaje de los terrenos que tienen más de 25,000 pies cuadrados
INDUS – porcentaje de acres asignados a negocios que no son comercios en cada población
CHAS - 1 si está a la orilla del río Charles; 0 si no
NOX – concentración de óxido nítrico (en partes por 10 milliones)
RM – número promedio de cuartos por habitación
AGE – proporción de habitaciones ocupadas construidas antes de 1940
DIS – distancia a dies centros de trabajo en Boston
RAD – índice de accesibilidad a carreteras radiales
TAX – tasa de impuesto predial en unidades de $10,000
PTRATIO – tasa de profesores a alumnos promedio en cada población
B - 1000(Bk - 0.63)^2 donde Bk es la proporción de gente de color en cada población
LSTAT - de personas de clase baja en cada población
MEDV – valor medio de las casas habitadas por su dueño en unidades de $1000

El conjunto de datos se puede obtener de la siguiente dirección:

https://www.kaggle.com/altavish/boston-housing-dataset

Nuestro objetivo es crear un modelo de regresión lineal que permita predecir los precios de las casas en el área de Boston usando como variables independientes el resto de las variables del conjunto de datos.

Instalaremos Spark en Colab como lo hemos hecho antes e iniciaremos una sesión.

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://downloads.apache.org/spark/spark-2.4.8/spark-2.4.8-bin-hadoop2.7.tgz
!tar xf spark-2.4.8-bin-hadoop2.7.tgz
!pip install -q findspark

import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.8-bin-hadoop2.7"

import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Leemos el archivo de datos:

```
path_csv="/content/BostonHousing.csv"
dataset = spark.read.csv(path_csv,inferSchema=True, header =True)

dataset.printSchema()
```

```
root
 |-- crim: double (nullable = true)
 |-- zn: double (nullable = true)
 |-- indus: double (nullable = true)
 |-- chas: integer (nullable = true)
 |-- nox: double (nullable = true)
 |-- rm: double (nullable = true)
 |-- age: double (nullable = true)
 |-- dis: double (nullable = true)
 |-- rad: integer (nullable = true)
```

```
 |-- tax: integer (nullable = true)
 |-- ptratio: double (nullable = true)
 |-- b: double (nullable = true)
 |-- lstat: double (nullable = true)
 |-- medv: double (nullable = true)
```

Para crear modelos de machine learning es necesario pasar los valores de las variables independientes a un vector. Esto ya lo hemos hecho antes al obtener la matriz de correlaciones en la sección pasada:

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

#Input all the features in one vector column
assembler = VectorAssembler(inputCols=['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax',
'ptratio', 'black', 'lstat'], outputCol = 'Attributes')

output = assembler.transform(dataset)

#Input vs Output
finalized_data = output.select("Attributes","medv")

finalized_data.show(10)
```

```
+-------------------+----+
|         Attributes|medv|
+-------------------+----+
|[0.00632,18.0,2.3...|24.0|
|[0.02731,0.0,7.07...|21.6|
|[0.02729,0.0,7.07...|34.7|
|[0.03237,0.0,2.18...|33.4|
|[0.06905,0.0,2.18...|36.2|
|[0.02985,0.0,2.18...|28.7|
|[0.08829,12.5,7.8...|22.9|
|[0.14455,12.5,7.8...|27.1|
|[0.21124,12.5,7.8...|16.5|
|[0.17004,12.5,7.8...|18.9|
```

```
+------------------+----+
only showing top 10 rows
```

En “Atributes” hemos puesto todas las variables dependientes y la variable “medv” es nuestra variable de interés, la variable a predecir.

Finalmente, separamos los datos. 80% se utilizarán para entrenar el modelo, y 20% se utilizarán para probar la eficiencia del modelo

```
#Split training and testing data
train_data,test_data = finalized_data.randomSplit([0.8,0.2])
```

¿Crees tú que estos porcentajes son los apropiados? ¿Bajo qué circunstancias utilizarías porcentajes diferentes?

¿Qué crees tu que podemos hacer ahora?

Pues efectivamente, lo siguiente es crear el modelo de predicción y evaluarlo.

Hemos podido leer un conjunto de datos, separar las variables independientes llamadas “features” de la variable de interés y hemos reservado 80% de los datos para crear el modelo y 20% de los datos para comprobar, científicamente, la validez de este.

Ahora aplicaremos modelos de machine learning que permiten, dado un conjunto de datos que contenga las variables independientes o “features”, hacer predicciones sobre la variable de interés.

Variables Continuas. Regresión

Si la variable de interés es continua, el modelo de machine learning debe realizar una regresión. **¿Por qué crees tu que el tipo de modelo para variable continua debe ser una regresión?**

En esta sección veremos cómo realizar en pySpark regresiones lineales, bosques de regresión aleatorios (*random forrest regressor, método de “bagging”*) y regresión de árboles de gradiente mejorado (*gradient-boosted regression*).

Regresión Lineal (LR)

Lo primero que hacemos es crear el modelo:

```
regressor = LinearRegression(featuresCol = 'Attributes', labelCol = 'medv')
```

Después le alimentamos el conjunto de datos de entrenamiento:

```
#Learn to fit the model from training set
regressor = regressor.fit(train_data)
```

Podemos ver los coeficientes del modelo de la siguiente manera:

```
print ("The coefficient of the model is : %a", regressor.coefficients)
print ("The Intercept of the model is : %f", regressor.intercept)
```

```
The coefficient of the model is : DenseVector([-0.127, 0.0475, 0.0345, 2.2758, -17.5745, 3.4635, 0.0213, -1.2939, 0.373, -0.0137, -0.937, 0.0069, -0.5831])
The Intercept of the model is : 37.604535
```

Y ahora obtenemos las predicciones del conjunto de datos de prueba:

```
#To predict the prices on testing set
Pred_lr = regressor.evaluate(test_data)
```

En la siguiente salida podremos ver qué tanto difieren los valores reales de la variable del costo medio de las casas “medv” contra la predicción del modelo:

```
#Predict the model

Pred_lr.predictions.show(10)
```

```
+------------------+----+-----------------+
|        Attributes|medv|       prediction|
+------------------+----+-----------------+
|[0.01301,35.0,1.5...|32.7| 29.95509351804548|
|[0.01439,60.0,2.9...|29.1|31.647429645578626|
|[0.01538,90.0,3.7...|44.0| 37.37907312463067|
|[0.02177,82.5,2.0...|42.3| 36.96490989639475|
|[0.02875,28.0,15....|25.0| 29.71368183258236|
|[0.03359,75.0,2.9...|34.9| 34.42769709366452|
|[0.03502,80.0,4.9...|28.5|33.628783219120606|
|[0.03548,80.0,3.6...|20.9|21.617668692682496|
|[0.03551,25.0,4.8...|22.9|25.098733639884987|
|[0.03738,0.0,5.19...|20.7|21.678340320583114|
+------------------+----+-----------------+
only showing top 10 rows
```

Bosques de Regresión Aleatorios (RFR)

Para llevar a cabo esta regresión, hay que cambiar el regresor. Hay un cambio adicional. La LR adicional utiliza regressor.evaluate(test_data) para obtener las predicciones del conjunto de prueba, mientras que el RFR utiliza rfr.transform(test_data).

```
#Carry out random forrest regression
from pyspark.ml.regression import RandomForestRegressor
rfr = RandomForestRegressor(featuresCol = 'Attributes', labelCol = 'medv')

#Learn to fit the model from training set
rfr = rfr.fit(train_data)

#To predict the prices on testing set
pred_rfr = rfr.transform(test_data)
```

Para ver las predicciones también hay un cambio. LR utiliza pred_lr.predictions.show(10). Mientras que RFR utiliza pred_rfr.select('Attributes', 'medv', 'prediction').show(10)

```
# Select example rows to display.
pred_rfr.select('Attributes', 'medv', 'prediction').show(10)
```

```
+------------------+----+-----------------+
|        Attributes|medv|       prediction|
+------------------+----+-----------------+
|[0.01311,90.0,1.2...|35.4| 32.65645811292001|
|[0.01501,80.0,2.0...|24.5| 27.14638161029571|
|[0.01778,95.0,1.4...|32.9| 33.14358252395231|
|[0.0187,85.0,4.15...|23.1|26.374893105802993|
|[0.02731,0.0,7.07...|21.6|  23.111652706978|
|[0.02763,75.0,2.9...|30.8| 29.07739493692007|
|[0.02899,40.0,1.2...|26.6|27.416704973707986|
|[0.02985,0.0,2.18...|28.7| 27.98489345272795|
|[0.03306,0.0,5.19...|20.6|21.639503965384122|
|[0.03359,75.0,2.9...|34.9| 33.90141751915762|
+------------------+----+-----------------+
only showing top 10 rows
```

Regresión de Árboles de Gradiente Mejorado (RBT)

Para realizar la regresión RBT sólo hay un cambio con respecto a la regresión RFR. Sólo hay que cambiar el regresor.

```
#Carry out Gradient-boosted tree regression
from pyspark.ml.regression import GBTRegressor
gbt = GBTRegressor(featuresCol = 'Attributes', labelCol = 'medv')

#Learn to fit the model from training set
gbt = gbt.fit(train_data)

#To predict the prices on testing set
pred_gbt = gbt.transform(test_data)

# Select example rows to display.
```

```
pred_gbt.select('Attributes', 'medv', 'prediction').show(10)
```

```
+------------------+----+-----------------+
|        Attributes|medv|       prediction|
+------------------+----+-----------------+
|[0.01311,90.0,1.2...|35.4| 36.21966418729401|
|[0.01501,80.0,2.0...|24.5| 29.66006597998819|
|[0.01778,95.0,1.4...|32.9|34.967534599610886|
|[0.0187,85.0,4.15...|23.1| 30.67733213239926|
|[0.02731,0.0,7.07...|21.6|  22.7435857954403|
|[0.02763,75.0,2.9...|30.8| 29.52014486379554|
|[0.02899,40.0,1.2...|26.6| 28.51104007201439|
|[0.02985,0.0,2.18...|28.7|25.193206478437705|
|[0.03306,0.0,5.19...|20.6| 23.24690130273002|
|[0.03359,75.0,2.9...|34.9| 33.98189601586928|
+------------------+----+-----------------+
only showing top 10 rows
```

Hemos visto cómo crear varios modelos de regresión y de clasificación. La evaluación de los modelos sin embargo, se puede utilizar usando diversas métricas. Cada una tiene sus ventajas y desventajas. Veamos las más importantes.

Evaluación de la Regresión

Para evaluar la regresión se puede usar el cuadrado medio de los errores (MSE), la raíz cuadra del cuadrado medio de los errores (RMSE) y el promedio de los errores absolutos (MAE). Sin embargo, el más utilizado es el factor de determinación, también llamado R cuadrada o R2. Este es el cuadrado de la correlación entre las predicciones y los valores reales.

Al entrenar un modelo de regresión, el modelo en sí tiene un atributo más o menos complicado llamado “Summary”. Este contiene los valores MSE, RMSE, MAE y R2 calculados con el conjunto de entrenamiento.

Evaluación de la Regresión Lineal

En la regresión lineal, conocer cada uno de los coeficientes es importante. Lo mismo que la calidad de cada uno de los coeficientes para tener la opción de eliminar variables inútiles del modelo. En la regresión lineal, el sumario también contiene los errores estándar, los valores t y los valores p para cada uno de los coeficientes del modelo derivado. El siguiente código que te regalamos creará una tabla parecida al reporte de regresión que se puede obtener en R. Recuerde que el t value debe ser mayor a 1.96 y el p value menor a 0.05:

```
import numpy as np
print ("Note: the last rows are the information for Intercept")
print ("##","-----------------------------------------------")
print ("##","  Estimate   |   Std.Error | t Values  |  P-value")
coef = np.append(list(regressor.coefficients),regressor.intercept)
Summary=regressor.summary

for i in range(len(Summary.pValues)):
    print ("##",'{:10.6f}'.format(coef[i]),\
    '{:10.6f}'.format(Summary.coefficientStandardErrors[i]),\
    '{:8.3f}'.format(Summary.tValues[i]),\
    '{:10.6f}'.format(Summary.pValues[i]))

print ("##",'---')
print ("##","Mean squared error: % .6f" \
        % Summary.meanSquaredError, ", \
        RMSE: % .6f" \
        % Summary.rootMeanSquaredError )
print ("##","Multiple R-squared: %f" % Summary.r2, ", \
        Total iterations: %i"% Summary.totalIterations)
```

```
Note: the last rows are the information for Intercept
## -----------------------------------------------
##   Estimate   |   Std.Error | t Values  |  P-value
## -0.114036   0.035192   -3.240   0.001296
##   0.047415   0.015132    3.133   0.001858
##   0.006564   0.066727    0.098   0.921693
```

```
##   2.250883   1.030145    2.185   0.029481
## -18.983913   4.630595   -4.100   0.000050
##   3.703236   0.471172    7.860   0.000000
##   0.004915   0.014770    0.333   0.739475
##  -1.501433   0.222753   -6.740   0.000000
##   0.350518   0.077495    4.523   0.000008
##  -0.014597   0.004315   -3.383   0.000790
##  -0.957594   0.145719   -6.572   0.000000
##   0.008163   0.003234    2.524   0.012002
##  -0.525544   0.056105   -9.367   0.000000
##  38.938289   5.869098    6.634   0.000000
## ---
## Mean squared error:  22.245915 ,        RMSE:  4.716557
## Multiple R-squared: 0.739218 ,        Total iterations: 1
```

Sin embargo, los valores MSE, RMSE, MAE y R2 realmente importantes son los que se obtienen del conjunto de prueba. Para obtenerlos hay que crear un “RegressionEvaluator”, que funciona con todas las regresiones que vimos. A este evaluador se le pasa el vector de predicciones creado con el método “evaluate” o “transform” de la siguiente manera:

```
from pyspark.ml.evaluation import RegressionEvaluator
eval_lr = RegressionEvaluator(labelCol="medv", predictionCol="prediction", metricName="rmse")

print("Linear regression model\n")

# Root Mean Square Error
rmse = eval_lr.evaluate(Pred_lr.predictions)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval_lr.evaluate(Pred_lr.predictions, {eval_lr.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval_lr.evaluate(Pred_lr.predictions, {eval_lr.metricName: "mae"})
print("MAE: %.3f" % mae)
```

```
# r2 - coefficient of determination
r2 = eval_lr.evaluate(Pred_lr.predictions, {eval_lr.metricName: "r2"})
print("r2: %.3f" %r2)
```

```
Linear regression model

RMSE: 5.175
MSE: 26.779
MAE: 3.412
r2: 0.646
```

El mismo procedimiento se puede seguir para los otros regresores. Lo único que hay que recordar es que, si se utilizó el método “evaluate” del modelo para obtener las predicciones, pase el atributo “predictions” del vector de predicciones al evaluador. Si se utilizó el método “transform”, pase todo el vector de predicciones al evaluador

Evaluación de Bosques de Regresión Aleatorios

```
eval_rfr = RegressionEvaluator(labelCol="medv", predictionCol="prediction", metricName="rmse")

print("Regression Forrest model\n")

# Root Mean Square Error
rmse = eval_rfr.evaluate(pred_rfr)

print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval_rfr.evaluate(pred_rfr, {eval_rfr.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval_rfr.evaluate(pred_rfr, {eval_rfr.metricName: "mae"})
print("MAE: %.3f" % mae)
```

```
# r2 - coefficient of determination
r2 = eval_rfr.evaluate(pred_rfr, {eval_rfr.metricName: "r2"})
print("r2: %.3f" %r2)
```

Regression Forrest model

RMSE: 3.667
MSE: 13.446
MAE: 2.530
r2: 0.823

Evaluación de Árboles de Gradiente Mejorados

```
eval_gbt = RegressionEvaluator(labelCol="medv", predictionCol="prediction",
metricName="rmse")

print("Gradient Boot Tree model\n")

# Root Mean Square Error
rmse = eval_gbt.evaluate(pred_gbt)

print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval_gbt.evaluate(pred_gbt, {eval_gbt.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval_gbt.evaluate(pred_gbt, {eval_gbt.metricName: "mae"})
print("MAE: %.3f" % mae)

# r2 - coefficient of determination
```

```
r2 = eval_gbt.evaluate(pred_gbt, {eval_gbt.metricName: "r2"})
print("r2: %.3f" %r2)
```

Gradient Boot Tree model

RMSE: 4.092
MSE: 16.741
MAE: 2.591
r2: 0.779

Podemos ver que el mejor resultado se obtiene con RFR con R2 igual a 0.823.

¿Cómo podrías utilizar la regresión en los proyectos que vas a iniciar en tu empresa?