spark_mllib

July 1, 2023

1 Spark et MLlib - ML

Régression logisitique avec Spark.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

MLlib est la librairie de machine learning distribué implémenté sur Spark et qui explique en partie son succès. La première mouture de la librairie était Mahout implémentée sur Hadoop. MLlib est devenu le standard. ML est la dernière version et s'appuie sur les DataFrame. On retrouve les mêmes concepts que ceux de scikit-learn tels que les Pipeline.

1.1 Data

[2]:
```
import os
if not os.path.exists("data_adult.txt"):
    from pyquickhelper.filehelper import unzip_files
    unzip_files("data_adult.zip", where_to=".")
assert os.path.exists("data_adult.txt")
import pandas
df = pandas.read_csv("data_adult.txt", sep="\t", encoding="utf-8")
df.head()
```

[2]:
```
   age          workclass  fnlwgt   education  education_num  \
0   39          State-gov   77516   Bachelors             13
1   50   Self-emp-not-inc   83311   Bachelors             13
2   38            Private  215646     HS-grad              9
3   53            Private  234721        11th              7
4   28            Private  338409   Bachelors             13

        marital_status          occupation    relationship    race      sex  \
0        Never-married        Adm-clerical   Not-in-family   White     Male
1   Married-civ-spouse     Exec-managerial         Husband   White     Male
2             Divorced   Handlers-cleaners   Not-in-family   White     Male
3   Married-civ-spouse   Handlers-cleaners         Husband   Black     Male
4   Married-civ-spouse      Prof-specialty            Wife   Black   Female

   capital_gain  capital_loss  hours_per_week  native_country  target
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
```

```
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
```

[3]:
```
df.dtypes
```

[3]:
```
age                int64
workclass         object
fnlwgt             int64
education         object
education_num      int64
marital_status    object
occupation        object
relationship      object
race              object
sex               object
capital_gain       int64
capital_loss       int64
hours_per_week     int64
native_country    object
target            object
dtype: object
```

[4]:
```
cols = list(filter(lambda tu: tu[1] != object, zip(range(len(df.columns)), df.dtypes)))
cols
```

[4]:
```
[(0, dtype('int64')),
 (2, dtype('int64')),
 (4, dtype('int64')),
 (10, dtype('int64')),
 (11, dtype('int64')),
 (12, dtype('int64'))]
```

[5]:
```
column_keep = set(_[0] for _ in cols)
column_keep
```

[5]:
```
{0, 2, 4, 10, 11, 12}
```

[6]:
```
df.to_csv("adult.txt", sep="\t", index=False, header=None)
```

[7]:
```
data = sc.textFile("adult.txt")
```

[8]:
```
col = data.take(2)
```

[9]:
```
col
```

[9]:
```
['39\t State-gov\t77516\t Bachelors\t13\t Never-married\t Adm-clerical\t Not-in-
family\t White\t Male\t2174\t0\t40\t United-States\t <=50K',
 '50\t Self-emp-not-inc\t83311\t Bachelors\t13\t Married-civ-spouse\t Exec-
managerial\t Husband\t White\t Male\t0\t0\t13\t United-States\t <=50K']
```

1.2 Régression logistique (RDD)

On reprend l'exemple de la documentation : Linear Methods - RDD-based API. On exclue les variables catégorielles pour garder l'exemple concis.

[10]:
```
def parsePoint(line):
    spl = line.split('\t')
    values = [float(x) for i, x in enumerate(spl) if i in column_keep]
    target = float(spl[-1].strip() == "<=50K")
    return LabeledPoint(target, values)

# We prepare the training data
parsedData = data.map(parsePoint)
parsedData.collect()[:2]
```

[10]:
```
[LabeledPoint(1.0, [39.0,77516.0,13.0,2174.0,0.0,40.0]),
 LabeledPoint(1.0, [50.0,83311.0,13.0,0.0,0.0,13.0])]
```

[11]:
```
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,␣
 ↪LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint

# Load and parse the data
def parsePoint(line):
    spl = line.split('\t')
    values = [float(x) for i, x in enumerate(spl) if i in column_keep]
    target = float(spl[-1].strip() == "<=50K")
    return LabeledPoint(target, values)

# We prepare the training data
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithLBFGS.train(parsedData)
```

Pendant que ça tourne, il faut regarder la fenêtre terminal qui affiche les messages du serveur de notebook.

[12]:
```
model.numClasses, model.numFeatures, model.weights
```

[12]:
```
(2, 6, DenseVector([0.0045, 0.0, 0.0086, -0.0003, -0.0008, 0.009]))
```

[13]:
```
from pyquickhelper.filehelper import remove_folder
def clean(folder):
    if os.path.exists(folder):
        return remove_folder(folder)
    else:
        return []
clean("target/pythonLogisticRegressionWithLBFGSModel")

# Save and load model
model.save(sc, "target/pythonLogisticRegressionWithLBFGSModel")
sameModel = LogisticRegressionModel.load(sc, "target/
 ↪pythonLogisticRegressionWithLBFGSModel")
```

[14]:
```
# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features)))
def filter_error(ys):
    return ys[0] != ys[1]
trainErr = labelsAndPreds.filter(filter_error).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

```
Training Error = 0.20217438039372254
```

1.3 Régression logisitique (DataFrame)

On s'inspire de l'exemple : Régression Logistique. Le code change, la préparation des données aussi. Les modèles acceptent comme entrées un vecteur colonne créé par un VectorAssembler.

[15]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark SQL basic example").getOrCreate()
```

[16]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
training = spark.createDataFrame(df)
training = training.withColumn('Y', training.target == " <=50K")
training = training.withColumn("Y", training.Y.astype('float'))
training = training.select(["age", "fnlwgt", "education_num", "capital_gain",␣
 ↪"capital_loss",
                            "hours_per_week", "Y"])
assembler = VectorAssembler(
    inputCols=["age", "fnlwgt", "education_num", "capital_gain", "capital_loss",␣
 ↪"hours_per_week"],
    outputCol="features")
training = assembler.transform(training)
```

[17]:
```
training.explain()
```

```
== Physical Plan ==
*Project [age#496L, fnlwgt#498L, education_num#500L, capital_gain#506L,
capital_loss#507L, hours_per_week#508L, cast((target#510 =  <=50K) as float) AS
Y#545, UDF(struct(cast(age#496L as double) AS
age_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#571, cast(fnlwgt#498L as double)
AS fnlwgt_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#572,
cast(education_num#500L as double) AS
education_num_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#573,
cast(capital_gain#506L as double) AS
capital_gain_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#574,
cast(capital_loss#507L as double) AS
capital_loss_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#575,
cast(hours_per_week#508L as double) AS
hours_per_week_double_VectorAssembler_4b1a9ef2a7fdcd07c46f#576)) AS
features#577]
+- Scan ExistingRDD[age#496L,workclass#497,fnlwgt#498L,education#499,education_n
um#500L,marital_status#501,occupation#502,relationship#503,race#504,sex#505,capi
tal_gain#506L,capital_loss#507L,hours_per_week#508L,native_country#509,target#51
0]
```

[18]:
```
head = training.take(2)
head
```

[18]:
```
[Row(age=39, fnlwgt=77516, education_num=13, capital_gain=2174, capital_loss=0,
hours_per_week=40, Y=1.0, features=DenseVector([39.0, 77516.0, 13.0, 2174.0,
0.0, 40.0])),
 Row(age=50, fnlwgt=83311, education_num=13, capital_gain=0, capital_loss=0,
hours_per_week=13, Y=1.0, features=DenseVector([50.0, 83311.0, 13.0, 0.0, 0.0,
13.0]))]
```

[19]:
```
training.schema
```

[19]:
```
StructType(List(StructField(age,LongType,true),StructField(fnlwgt,LongType,true)
,StructField(education_num,LongType,true),StructField(capital_gain,LongType,true
),StructField(capital_loss,LongType,true),StructField(hours_per_week,LongType,tr
ue),StructField(Y,FloatType,true),StructField(features,VectorUDT,true)))
```

[20]:
```
training.groupby("Y").count().collect()
```

[20]:
```
[Row(Y=1.0, count=24720), Row(Y=0.0, count=7841)]
```

[21]:
```
from pyspark.ml.classification import LogisticRegression
```

[22]:
```
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8, labelCol='Y',␣
↪featuresCol="features")

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

```
Coefficients: (6,[],[])
Intercept: 1.1482462553407051
```

[23]:
```
prediction = lrModel.transform(training)
prediction.take(2)
```

[23]:
```
[Row(age=39, fnlwgt=77516, education_num=13, capital_gain=2174, capital_loss=0,
hours_per_week=40, Y=1.0, features=DenseVector([39.0, 77516.0, 13.0, 2174.0,
0.0, 40.0]), rawPrediction=DenseVector([-1.1482, 1.1482]),
probability=DenseVector([0.2408, 0.7592]), prediction=1.0),
 Row(age=50, fnlwgt=83311, education_num=13, capital_gain=0, capital_loss=0,
hours_per_week=13, Y=1.0, features=DenseVector([50.0, 83311.0, 13.0, 0.0, 0.0,
13.0]), rawPrediction=DenseVector([-1.1482, 1.1482]),
probability=DenseVector([0.2408, 0.7592]), prediction=1.0)]
```

[24]:
```

```