PySpark 커닝 페이퍼

Wenqiang Feng

E-mail: von198@gmail.com, Web: http://web.utk.edu/~wfeng1; https://runawayhorse001.github.io/LearningApacheSpark

Spark 환경 설정

```
from pyspark.sql import SparkSession
spark = SparkSession.builder
        .appName("Python Spark regression example")
        .config("config.option", "value").getOrCreate()
```

데이터 로딩

RDDs로 부터

```
# parallelize( ) 을/를 사용해서
df = spark.sparkContext.parallelize([('1','Joe','70000','1'),
                     ('2', 'Henry', '80000', None)])
.toDF(['Id', 'Name', 'Sallary','DepartmentId'])
# createDataFrame( ) 을 사용해서
df = spark.createDataFrame([('1', 'Joe',   '70000', '1'),
                            ('2', 'Henry', '80000', None)],
                    ['Id','Name','Sallary','DepartmentId'])
```

```
+---+-----+-------+------------+
| Id| Name|Sallary|DepartmentId|
+---+-----+-------+------------+
|  1|  Joe|  70000|           1|
|  2|Henry|  80000|        null|
+---+-----+-------+------------+
```

Data 소스들로 부터

▷ **From .csv**

```
ds = spark.read.csv(path='Advertising.csv',
                    sep=',',encoding='UTF-8',comment=None,
                    header=True,inferSchema=True)
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```

▷ **From .json**

```
df = spark.read.json('/home/feng/Desktop/data.json')
```

```
+----------+--------------------+-------------------+
|        id|            location|          timestamp|
+----------+--------------------+-------------------+
|2957256202|[72.1,DE,8086,52....|2019-02-23 22:36:52|
|2957256203|[598.5,BG,3963,42...|2019-02-23 22:36:52|
+----------+--------------------+-------------------+
```

▷ **From Database**

```
user = 'username'; pw ='password'
table_name = 'table_name'
url='jdbc:postgresql://##.###.###.##:5432/dataset?user='
     +user+'&password='+pw
p='driver':'org.postgresql.Driver','password':pw,'user':user
df = spark.read.jdbc(url=url,table=table_name,properties=p)
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```

▷ **From HDFS**

```
from pyspark.conf import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import HiveContext
sc= SparkContext('local','example')
hc = HiveContext(sc)
tf1 = sc.textFile("hdfs://###/user/data/file_name")
```

```
+-----+-----+---------+-----+
|   TV|Radio|Newspaper|Sales|
+-----+-----+---------+-----+
|230.1| 37.8|     69.2| 22.1|
| 44.5| 39.3|     45.1| 10.4|
+-----+-----+---------+-----+
```


데이터 감사

스키마 확인

```
df.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- TV: double (nullable = true)
 |-- Radio: double (nullable = true)
 |-- Newspaper: double (nullable = true)
 |-- Sales: double (nullable = true)
```

결측값 확인

```
from pyspark.sql.functions import count
def my_count(df):
    df.agg(*[count(c).alias(c) for c in df.columns]).show()
my_count(df_raw)
```

```
+---------+---------+--------+-----------+---------+----------+-------+
|InvoiceNo|StockCode|Quantity|InvoiceDate|UnitPrice|CustomerID|Country|
+---------+---------+--------+-----------+---------+----------+-------+
|   541909|   541909|  541909|     541909|   541909|    406829| 541909|
+---------+---------+--------+-----------+---------+----------+-------+
```

통계 결과 확인

```
# pyspark 함수
df_raw.describe().show()
```

```
+-------+------------------+-----------------+------------------+
|summary|                TV|            Radio|         Newspaper|
+-------+------------------+-----------------+------------------+
|  count|               200|              200|               200|
|   mean|         147.0425|23.264000000000024|30.553999999999995|
| stddev|85.854236314908051|14.846809176168728| 21.778620838522831|
|    min|               0.7|              0.0|               0.3|
|    max|             296.4|             49.6|             114.0|
+-------+------------------+-----------------+------------------+
```

데이터 조작 (다음 페이지에 자세히)

결측값 수정

Function	Description
`df.na.fill()`	#결측값 대체하기
`df.na.drop()`	#결측값을 갖는 행을 제거하기

데이터 Join

Description	Function
#Data join	`left.join(right,key, how='*')` * = left,right,inner,full

UDF를 통한 데이터 정제

```
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
# 사용자 정의 함수
def complexFun(x):
    return results
Fn = F.udf(lambda x: complexFun(x), DoubleType())
df.withColumn('2col', Fn(df.col))
```

특징 줄이기

```
df.select(featureNameList)
```

모델링 파이프라인

범주형 특징과 라벨 데이터 다루기

```
# 범주형 특징 데이터 다룹니다
from pyspark.ml.feature import VectorIndexer
featureIndexer = VectorIndexer(inputCol="features",
                               outputCol="indexedFeatures",
                               maxCategories=4).fit(data)
featureIndexer.transform(data).show(2, True)
```

```
+--------------------+-----+--------------------+
|            features|label|     indexedFeatures|
+--------------------+-----+--------------------+
|(29,[1,11,14,16,1...|   no|(29,[1,11,14,16,1...|
+--------------------+-----+--------------------+
```

```
# 범주형 라벨 데이터 다룹니다
labelIndexer=StringIndexer(inputCol='label',
                           outputCol='indexedLabel').fit(data)
labelIndexer.transform(data).show(2, True)
```

```
+--------------------+-----+------------+
|            features|label|indexedLabel|
+--------------------+-----+------------+
|(29,[1,11,14,16,1...|   no|         0.0|
+--------------------+-----+------------+
```

데이터를 훈련 및 검증 셋으로 분할

```
(trainingData, testData) = data.randomSplit([0.6, 0.4])
```

모델 불러오기

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='indexedFeatures',
                        labelCol='indexedLabel')
```

인덱스 처리된 라벨들을 본래 라벨로 변환하기

```
from pyspark.ml.feature import IndexToString
labelConverter = IndexToString(inputCol="prediction",
                               outputCol="predictedLabel",
                               labels=labelIndexer.labels)
```

Pipeline 화

```
pipeline = Pipeline(stages=[labelIndexer, featureIndexer,
                            lr,labelConverter])
```

모델 훈련 및 예측값 생성

```
model = pipeline.fit(trainingData)
predictions = model.transform(testData)
predictions.select("features","label","predictedLabel").show(2)
```

```
+--------------------+-----+--------------+
|            features|label|predictedLabel|
+--------------------+-----+--------------+
|(29,[0,11,13,16,1...|   no|            no|
+--------------------+-----+--------------+
```

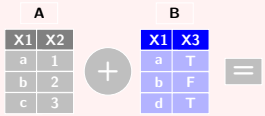
모델 평가

```
from pyspark.ml.evaluation import *
evaluator = MulticlassClassificationEvaluator(
           labelCol="indexedLabel",
           predictionCol="prediction", metricName="accuracy")
accu = evaluator.evaluate(predictions)
print("Test Error: %g, AUC: %g"%(1-accu,Summary.areaUnderROC))
```

```
Test Error: 0.0986395, AUC: 0.886664269877
```

데이터 정제: 데이터 프레임 합치기

Join으로 데이터 변형



결과

X1 X2 X3
a 1 T
b 2 F
c 3 null

함수

```
#X1열을 기준으로 A에 B를 매칭시킵니다
#dplyr::left_join(A, B, by = "x1")
A.join(B,'X1',how='left')
 .orderBy('X1', ascending=True).show()
```

X1 X2 X3
a 1 T
b 2 F
c null T

```
#X1열을 기준으로 B에 A를 매칭시킵니다
#dplyr::right_join(A, B, by = "x1")
A.join(B,'X1',how='right')
 .orderBy('X1', ascending=True).show()
```

X1 X2 X3
a 1 T
b 2 F

```
#X1열을 기준으로 A와 B에 모두 존재하는 행만 유지합니다
#dplyr::inner_join(A, B, by = "x1")
A.join(B,'X1',how='inner')
 .orderBy('X1', ascending=True).show()
```

X1 X2 X3
a 1 T
b 2 F
c 3 null
d null T

```
#모든 값, 모든 행들 유지하기
#dplyr::full_join(A, B, by = "x1")
A.join(B,'X1',how='full')
 .orderBy('X1', ascending=True).show()
```

조인으로 데이터 필터링

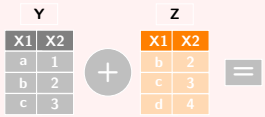
X1 X2
a 1
b 2

```
#B와 매칭되는 모든 A 행 유지하기
#dplyr::semi_join(A, B, by = "x1")
a.join(b,'X1',how='left_semi')
 .orderBy('X1', ascending=True).show()
```

X1 X2
C 3

```
#B와 매칭되지 않은 모든 A행 유지하기
#dplyr::anti_join(A, B, by = "x1")
A.join(B,'X1',how='left_anti')
 .orderBy('X1', ascending=True).show()
```

데이터 프레임 연산



결과

X1 X2
b 2
c 3

```
#Y와 Z 동시에 있는 행들
#dplyr::intersect(Y, Z)
Y.intersect(Z).show()
```

X1 X2
a 1
b 2
c 3
d 4

```
#Y 와 Z에 있는 모든 행들
#dplyr::union(Y, Z)
Y.union(Z).dropDuplicates()
 .orderBy('X1', ascending=True).show()
```

X1 X2
a 1

```
#Y에는 있지만 Z에는 없는 행들
#dplyr::setdiff(Y, Z)
Y.subtract(Z).show()
```

바인딩

X1 X2
a 1
b 2
c 3
b 2
c 3
d 4

```
#z를 y에 새 행으로 추가합니다
#dplyr::bind_rows(Y, Z)
Y.union(Z)
 .orderBy('X1', ascending=True).show()
```

X1 X2 X1 X2
a 1 b 2
b 2 c 3
c 3 d 4

```
#z를 y에 새 열로 추가합니다
#주의: 제가 만든 패키지의 zipDataFrames
#dplyr::bind_cols(Y, Z)
zipDataFrames(Y,Z).show()
```


데이터 정제: 데이터 차원 바꾸기

데이터 분할

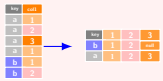
변화 **함수**

```
#ArrayType() tidyr::separate ::한 열을 여러 열로 나눕니다
df.select("key", df.value[0], df.value[1], df.value[2]).show()
#StructType()
df2.select('key', 'value.*').show()
```

```
#한 열을 행으로 나눕니다
df.select("key",F.split("values", ",").alias("values"),
        F.posexplode(F.split("values",",")).alias("pos", "val")
         ).drop("val")
  .select("key",F.expr("values[pos]").alias("val")).show()
```

```
#열들을 행으로 모읍니다
 def to_long(df, by):
    cols, dtypes = zip(*((c,t) for (c, t) in df.dtypes if c not in by))
    # Spark SQL 은 동종 열만 지원합니다
    assert len(set(dtypes))==1,"All columns have to be of the same type"
    # (column_name, column_value) 구조 배열 생성 및 여러 행으로 전개합니다
    kvs = explode(array([
      struct(lit(c).alias("key"), col(c).alias("val")) for c in
    cols])).alias("kvs")
    return df.select(by + [kvs]).select(by + ["kvs.key", "kvs.val"])
```

피벗



```
#행을 열로 펼칩니다
df.groupBy(['key'])
  .pivot('col1').sum('col1').show()
```

데이터 부분 집합(행)

함수 **설명**

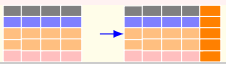
함수	설명
df.na.drop()	#결측값이 있는 행들을 제거합니다
df.where()	#주어진 조건에 부합하는 행들만 유지합니다
df.filter()	#주어진 조건에 부합하는 행들만 유지합니다
df.distinct()	#데이터 프레임에서 유일한 행들만 반환합니다
df.sample()	#데이터 프레임에서 샘플링한 부분집합을 반환합니다
df.sampleBy()	#비복원 추출로 계층화된 샘플을 반환합니다

부분 변수(열)

함수 **설명**

함수	설명
df.select()	#기존 데이터에 표현식을 적용해 새 데이터를 반환합니다

신규 변수 생성



함수 **예제**

```
df.withColumn() df.withColumn('new',1/df.col)
                df.withColumn('new',F.log(df.col))
                df.withColumn('id', psf.monotonically_increasing_id())
                df.withColumn("new", Fn('col')) #Fn:F.udf()
                df.withColumn('new', F.when((df.c1>1)&(df.c2<2),1)
                                     .when((df.c3>3),2).otherwise(3))
```

데이터 셋에서 계산된 열 만들기

데이터 요약

함수 **설명**

```
df.describe()          #기초 통계를 계산합니다
Correlation.corr(df)   #상관 행렬을 계산합니다
df.count()             #행의 개수를 셉니다
```

Description **Demo**

```
#Sum   df.agg(F.max(df.C)).head()[0]
       #다음과 비슷합니다: F.min,max,avg,stddev
```

Data 그룹화

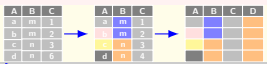
A min_b max_b avg_c
m 1 2 4.5
n 3 4 7.5

```
df.groupBy(['A'])
  .agg(F.min('B').alias('min_b'),
       F.max('B').alias('max_b'),
       F.avg('C').alias('avg_c')).show()
```

A min_b max_b list_c
m 1 2 [4.2,4.5,4.75]
n 3 4 [7.2,7.5,7.75]

```
def quant_pd(val_list):
    quant = np.round(np.percentile(val_list,
[20,50,75]),2)
return list(map(float,quant))
Fn = F.udf(quant_pd,ArrayType(FloatType()))
#GroupBy 와 aggregate
df.groupBy(['A'])
  .agg(F.min('B').alias('min_b'),
       F.max('B').alias('max_b'),
     Fn(F.collect_list(col('C'))).alias('list_c'))
```

Windows



결과 **함수**

A B C D
a m 1 0
b m 2 1
c n 3 0
d n 6 3

```
from pyspark.sql import Window
#차이를 계산하기 위해 windows 를 정의합니다
w = Window.partitionBy(df.B) D = df.C -
F.min(df.C).over(w)
df.withColumn('D',D).show()
```

A B C D
a m 1 1
b m 2 2
c n 3 3
d n 6 4

```
df = df.withColumn("D",
    F.monotonically_increasing_id())
#행 번호를 위해 windows 를 정의합니다
w = Window.orderBy("D")
df.withColumn("D", F.row_number().over(w))
```

A B C D
b m 2 1
a m 1 2
d n 6 1
c n 3 2

```
#순위를 위해 windows 를 정의합니다
w = Window.partitionBy('B')
          .orderBy(df.C.desc())
df.withColumn("D",rank().over(w)).show()
```

변수명 변경

함수 **설명**

```
df.withColumnRenamed() #기존 열의 이름을 변경하여 새 데이터를 반환합니다
```