# Spark ML

April 12, 2020

## 1 Prática Spark e MLib

[1]:
```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.sql.functions import monotonically_increasing_id
```

[2]:
```
from pyspark.sql import SparkSession

sc = SparkSession.builder.master('spark://172.18.0.4:7077').config('spark.
↪executor.memory', '1g').getOrCreate()
```

[9]:
```
mc_4 = [[0, 0, 0, 0], [0, 0, 0, 0 ], [0, 0, 0, 0 ], [0, 0, 0, 0 ]]
mc_2 = [[0, 0], [0, 0 ]]
```

[60]:
```
def matrix4(resultado):
    if resultado.label != resultado.prediction and resultado.label == 0 and␣
↪resultado.prediction == 1:
        print(resultado.label, resultado.prediction)
    mc_4[int(resultado.label)][int(resultado.prediction)] += 1


def print_matrix(matrix):
  for linha in matrix:
    z = ''
    for num in linha:
      z += str(num) + '\t'
    print(z)

def false_matrix(matrix, n):
  mc = [[0, 0], [0, 0 ]]
  l = 0
  for linha in matrix:
    c = 0
    for num in linha:
      if c == n and l == n:
        mc[0][0] = num
      elif c == n:
        mc[1][0] += num
```

```
      elif l == n:
        mc[0][1] += num
      else:
        mc[1][1] += num
      c += 1
    l += 1
  return mc

def false_negative(matrix):
  return matrix[1][0]/(matrix[0][0] + matrix[1][0])

def false_positive(matrix):
  return matrix[0][1]/(matrix[0][1] + matrix[1][1])

def rate(matrix):
  false_positive, false_negative = 0
  n = len(matrix)
  for t in range(n):
    x = false_matrix(matrix, t)
    false_positive += false_posite(x)
    false_negative += false_negative(x)
  return (false_positive/n, false_negative/n)
```

[8]:
```
df = sc.read.option('delimiter', ',').option('header', 'true').
↪option('inferschema', 'true')\
    .csv('hdfs://172.18.0.9:9000/treinamento.csv')
```

[11]:
```
df_teste = sc.read.option('delimiter', ',').option('header', 'true').
↪option('inferschema', 'true')\
    .csv('hdfs://172.18.0.9:9000/teste.csv')
```

[18]:
```
ds_teste = StringIndexer(inputCol='Classe', outputCol='label')\
    .fit(df).transform(df_teste)

features = ['hora', 'minuto', 'temp_minima', 'temp_maxima', 'latitude_media',␣
↪'longitude_media']

ds_teste = VectorAssembler(inputCols=features, outputCol='features')\
    .transform(ds_teste)

ds_teste = ds_teste.withColumn('id', monotonically_increasing_id())
ds_teste = ds_teste.select('id', 'label', 'features')
```

[15]:
```
ds_treina = StringIndexer(inputCol='Classe', outputCol='label')\
    .fit(df).transform(df)
```

```
features = ['hora', 'minuto', 'temp_minima', 'temp_maxima', 'latitude_media',␣
 ↪'longitude_media']

ds_treina = VectorAssembler(inputCols=features, outputCol='features')\
    .transform(ds_treina)

ds_treina = ds_treina.withColumn('id', monotonically_increasing_id())
```

[19]:
```
ds_treina = ds_treina.select('id', 'label', 'features')
ds_treina.printSchema()
ds_teste.printSchema()
```

```
root
 |-- id: long (nullable = false)
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)

root
 |-- id: long (nullable = false)
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
```

[21]:
```
treinamento = ds_treina
teste = ds_teste
```

### 1.1 Arvore de Desisão

[22]:
```
from pyspark.ml.classification import DecisionTreeClassifier

arvore = DecisionTreeClassifier(labelCol='label', featuresCol='features')
modeloArvore = arvore.fit(treinamento)
```

[23]:
```
modeloArvore.toDebugString
```

[23]:
```
'DecisionTreeClassificationModel (uid=DecisionTreeClassifier_ab3bb341e97c) of
depth 5 with 49 nodes\n  If (feature 2 <= 9.132499500000002)\n   If (feature 2
<= -5.4054995)\n    If (feature 4 <= 39.379245)\n     If (feature 2 <=
-9.376500499999999)\n      If (feature 3 <= 25.890000999999998)\n       Predict:
2.0\n      Else (feature 3 > 25.890000999999998)\n       Predict: 0.0\n     Else
(feature 2 > -9.376500499999999)\n      If (feature 3 <= 24.578501)\n
Predict: 2.0\n      Else (feature 3 > 24.578501)\n       Predict: 0.0\n    Else
(feature 4 > 39.379245)\n     If (feature 3 <= 27.843502)\n      Predict: 2.0\n
Else (feature 3 > 27.843502)\n      If (feature 4 <= 44.201212)\n       Predict:
0.0\n      Else (feature 4 > 44.201212)\n       Predict: 2.0\n   Else (feature 2
> -5.4054995)\n    If (feature 2 <= 6.722500999999999)\n     If (feature 3 <=
```

```
36.846988499999995)\n      Predict: 0.0\n     Else (feature 3 >
36.846988499999995)\n      Predict: 3.0\n    Else (feature 2 >
6.722500999999999)\n     If (feature 3 <= 29.6935035)\n      If (feature 4 <=
23.311282)\n       Predict: 3.0\n      Else (feature 4 > 23.311282)\n
Predict: 0.0\n     Else (feature 3 > 29.6935035)\n      If (feature 5 <=
19.01122)\n       Predict: 0.0\n      Else (feature 5 > 19.01122)\n
Predict: 3.0\n  Else (feature 2 > 9.132499500000002)\n   If (feature 2 <=
13.709498)\n    If (feature 3 <= 34.3925)\n     If (feature 5 <= 26.278988)\n
Predict: 0.0\n     Else (feature 5 > 26.278988)\n      Predict: 3.0\n    Else
(feature 3 > 34.3925)\n     If (feature 2 <= 10.035999499999999)\n      If
(feature 5 <= 73.019625)\n       Predict: 1.0\n      Else (feature 5 >
73.019625)\n       Predict: 3.0\n     Else (feature 2 > 10.035999499999999)\n
If (feature 3 <= 36.213995)\n       Predict: 3.0\n      Else (feature 3 >
36.213995)\n       Predict: 1.0\n   Else (feature 2 > 13.709498)\n    If
(feature 3 <= 28.315004000000002)\n     If (feature 3 <= 23.911004)\n
Predict: 0.0\n     Else (feature 3 > 23.911004)\n      If (feature 2 <=
22.5084985)\n       Predict: 3.0\n      Else (feature 2 > 22.5084985)\n
Predict: 1.0\n    Else (feature 3 > 28.315004000000002)\n     If (feature 2 <=
20.288996)\n      If (feature 3 <= 30.609502)\n       Predict: 3.0\n      Else
(feature 3 > 30.609502)\n       Predict: 1.0\n     Else (feature 2 >
20.288996)\n      Predict: 1.0\n'
```

[24]:
```
teste.printSchema()
```

```
root
 |-- id: long (nullable = false)
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
```

[31]:
```
resultadoArvore = modeloArvore.transform(teste)
```

[32]:
```
resultadoArvore.printSchema()
```

```
root
 |-- id: long (nullable = false)
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

[33]:
```
resultadoArvore.take(5)
```

[33]:
```
[Row(id=0, label=0.0, features=DenseVector([11.7872, 29.0051, 1.123, 28.606,
34.5604, 136.2198]), rawPrediction=DenseVector([10540.0, 0.0, 115.0, 36.0]),
```

```
probability=DenseVector([0.9859, 0.0, 0.0108, 0.0034]), prediction=0.0),
 Row(id=1, label=0.0, features=DenseVector([11.4826, 29.8458, 1.038, 29.306,
34.5608, 136.2202]), rawPrediction=DenseVector([10540.0, 0.0, 115.0, 36.0]),
probability=DenseVector([0.9859, 0.0, 0.0108, 0.0034]), prediction=0.0),
 Row(id=2, label=0.0, features=DenseVector([11.3121, 29.7517, 1.123, 29.006,
34.5586, 136.2205]), rawPrediction=DenseVector([10540.0, 0.0, 115.0, 36.0]),
probability=DenseVector([0.9859, 0.0, 0.0108, 0.0034]), prediction=0.0),
 Row(id=3, label=0.0, features=DenseVector([11.531, 29.5058, 1.404, 29.081,
34.5607, 136.2201]), rawPrediction=DenseVector([10540.0, 0.0, 115.0, 36.0]),
probability=DenseVector([0.9859, 0.0, 0.0108, 0.0034]), prediction=0.0),
 Row(id=4, label=0.0, features=DenseVector([11.4008, 29.4223, 1.504, 29.181,
34.5598, 136.2201]), rawPrediction=DenseVector([10540.0, 0.0, 115.0, 36.0]),
probability=DenseVector([0.9859, 0.0, 0.0108, 0.0034]), prediction=0.0)]
```

### 1.1.1 Accuracy

[34]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol='label',
                                              predictionCol='prediction',
                                              metricName='accuracy')

acc = evaluator.evaluate(resultadoArvore)
acc
```

[34]:
```
0.943173867752919
```

[35]:
```
mc_4 = [[0, 0, 0, 0], [0, 0, 0, 0 ], [0, 0, 0, 0 ], [0, 0, 0, 0 ]]
for row in resultadoArvore.toLocalIterator():
    matrix4(row)
```

### 1.1.2 Matriz de Confusão

[36]:
```
print_matrix(mc_4)
```

```
18714   0       986     230
0       17672   0       777
371     0       17007   0
205     1086    0       7271
```

### 1.1.3 Falso Positivo

[37]:
```
false_positive(false_matrix(mc_4, 0))
```

[37]:
```
0.027004819116569323
```

### 1.1.4 Falso Negativo

[38]:
```
false_negative(false_matrix(mc_4, 0))
```

[38]:
```
0.029860031104199068
```

## 1.2 Random Forest 20

[39]:
```
from pyspark.ml.classification import RandomForestClassifier

floresta20 = RandomForestClassifier(labelCol='label', featuresCol='features',␣
 ↪numTrees=20)
modeloFloresta = floresta20.fit(treinamento)
```

[41]:
```
resultadoFloresta20 = modeloFloresta.transform(teste)
```

[42]:
```
resultadoFloresta20.printSchema()
```

```
root
 |-- id: long (nullable = false)
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

### 1.2.1 Acurácia

[43]:
```
evaluator = MulticlassClassificationEvaluator(labelCol='label',␣
 ↪predictionCol='prediction', metricName='accuracy')
acc = evaluator.evaluate(resultadoFloresta20)
acc
```

[43]:
```
0.9445886907445701
```

[62]:
```
mc_4 = [[0, 0, 0, 0], [0, 0, 0, 0 ], [0, 0, 0, 0 ], [0, 0, 0, 0 ]]
for row in resultadoFloresta20.toLocalIterator():
    matrix4(row)
```

### 1.2.2 Matriz de Confusão

[63]:
```
print_matrix(mc_4)
```

```
18941   0       550     439
0       17869   0       580
402     0       16976   0
117     1476    0       6969
```

### 1.2.3 Falso Positivo

[64]:
```
false_positive(false_matrix(mc_4, 0))
```

[64]: 0.022046857932633362

### 1.2.4 Falso Negativo

[65]:
```
false_negative(false_matrix(mc_4, 0))
```

[65]: 0.026670092497430627

## 1.3 Random Forest 100

[66]:
```
from pyspark.ml.classification import RandomForestClassifier

floresta100 = RandomForestClassifier(labelCol='label', featuresCol='features',␣
 ↪numTrees=100)
modeloFloresta100 = floresta100.fit(treinamento)
```

[67]:
```
resultadoFloresta100 = modeloFloresta100.transform(teste)
```

### 1.3.1 Acuracy

[68]:
```
evaluator = MulticlassClassificationEvaluator(labelCol='label',␣
 ↪predictionCol='prediction', metricName='accuracy')
acc = evaluator.evaluate(resultadoFloresta100)
acc
```

[68]: 0.949859295076105

[69]:
```
mc_4 = [[0, 0, 0, 0], [0, 0, 0, 0 ], [0, 0, 0, 0 ], [0, 0, 0, 0 ]]
for row in resultadoFloresta100.toLocalIterator():
    matrix4(row)
```

### 1.3.2 Matriz Confusão

[70]:
```
print_matrix(mc_4)
```

```
18919   0       665     346
0       18095   0       354
256     0       17122   0
128     1476    0       6958
```

### 1.3.3 Falso Positivo

[71]:
```
false_positive(false_matrix(mc_4, 0))
```

[71]: 0.02245868135773947

### 1.3.4 Falso Negativo

[72]:
```
false_negative(false_matrix(mc_4, 0))
```

[72]: 0.019893280837175568

## 1.4 Ensemble dos 3

[140]:
```
from pyspark.sql.functions import col

resultadoTest = resultadoArvore.selectExpr('id as id_a', 'label', 'prediction␣
↪as prediction_a')\
  .join( resultadoFloresta100.selectExpr('id as id_100', 'label as label_100',␣
↪'prediction as prediction_100')\
    .join(resultadoFloresta20.selectExpr('id as id_20', 'label as label_20',␣
↪'prediction as prediction_20'), col('id_100') == col('id_20')),␣
↪col('id_100') == col('id_a'))
```

[141]:
```
resultadoTest.count()
```

[141]: 64319

[148]:
```
resultadoTest.take(2)
```

[148]:
```
[Row(label=0.0, prediction_a=0.0, prediction_20=0.0, prediction_100=0.0),
 Row(label=0.0, prediction_a=0.0, prediction_20=0.0, prediction_100=0.0)]
```

[143]:
```
resultadoTest = resultadoTest.select('label', 'prediction_a', 'prediction_20',␣
↪'prediction_100')\
    .lambda
```

[ ]:
```

```

[135]:
```
import pyspark.sql.functions as F
from pyspark.sql.types import *

def prediction(value1, value2, value3):
    if value1 == value2 or value1 == value3:
        return value1
    elif value2 == value3:
        return value2
    return value3

#convert to a UDF Function by passing in the function and return type of␣
↪function

udfPrediction = F.UserDefinedFunction(prediction, DoubleType())
```

[154]:
```
resultadoTest = resultadoTest.withColumn("prediction",
                                         F.expr('case when prediction_a =␣
↪prediction_20 or prediction_a = prediction_100 then prediction_a when␣
↪prediction_100 = prediction_20 then prediction_20 else prediction_100 end'))
```

[155]:
```
resultadoTest.take(2)
```

[155]:
```
[Row(label=0.0, prediction_a=0.0, prediction_20=0.0, prediction_100=0.0,
prediction=0.0),
 Row(label=0.0, prediction_a=0.0, prediction_20=0.0, prediction_100=0.0,
prediction=0.0)]
```

[156]:
```
mc_4 = [[0, 0, 0, 0], [0, 0, 0, 0 ], [0, 0, 0, 0 ], [0, 0, 0, 0 ]]
for row in resultadoTest.toLocalIterator():
    matrix4(row)
```

### 1.4.1 Accuracy

[157]:
```
evaluator = MulticlassClassificationEvaluator(labelCol='label',␣
↪predictionCol='prediction', metricName='accuracy')
acc = evaluator.evaluate(resultadoTest)
acc
```

[157]: 0.9485533046222734

### 1.4.2 Matriz Confusão

[158]:
```
print_matrix(mc_4)
```

```
18908   0       666     356
0       17958   0       491
327     0       17051   0
128     1341    0       7093
```

### 1.4.3 Falso Positivo

[159]:
```
false_positive(false_matrix(mc_4, 0))
```

[159]: 0.022733339265059167

### 1.4.4 Falso Negativo

[160]:
```
false_negative(false_matrix(mc_4, 0))
```

[160]: 0.023498424830862986

## 2 Resultados

| Classificador | ACC | FP | FN |
|---|---|---|---|
| Árvore de Decisão | 94,32% | 2,7% | 2,99% |
| Random Forest (20) | 94,56% | 2,2% | 2,67% |
| Random Forest (100) | 94,99% | 2,25% | 1,99% |
| Ensemble dos 3 | 94,86% | 2,27% | 2,35% |