BIG DATA IN MACHINE LEARNING

Bài 8: PySpark Mllib – Tree Models

Phòng LT & Mạng

https://csc.edu.vn/lap-trinh-va-csdl/Big-Data-in-Machine-Learning_198

2020

Nội dung

1. Decision Tree
2. Random Forest
3. Gradient-Boosted Trees

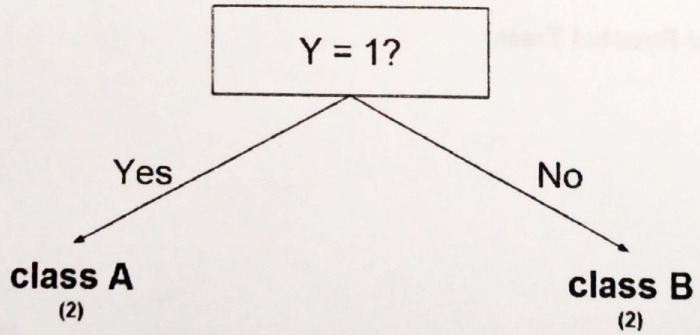
Decision Tree

- Giả sử bộ dữ liệu có 3 thuộc tính X, Y và Z với hai class kết quả là A và B

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B

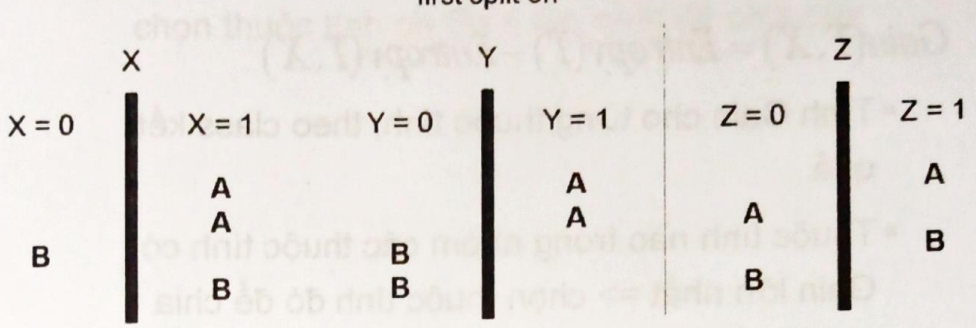
Decision Tree

- Tạo cây theo thuộc tính Y trước, chúng ta có kết quả phân chia hai class rõ ràng:



Decision Tree

- Chúng ta cũng có thể tạo cây theo các thuộc tính khác trước, như sau:



Decision Tree

- **Vấn đề đặt ra là làm thế nào để chúng ta có thể chọn được thuộc tính giúp cây phân chia tốt nhất (thuần khiết nhất)?**
  - Entropy & Information Gain
  - Hoặc Gini

Decision Tree

- Entropy & Information Gain

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i) \quad \text{Hay} \quad Entropy = -\sum_{i=1}^{C} p_i * \log_2(p_i)$$

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

  - Tính Gain cho từng thuộc tính, theo class kết quả.
  - Thuộc tính nào trong nhóm các thuộc tính có Gain lớn nhất => chọn thuộc tính đó để chia nhánh cho cây.

Decision Tree

- Tính Gain cho X
  - X: mẫu 1 = 3, mẫu 0 = 1
    - X = 1 & output = A: 2/3
    - X = 1 & Output = B: 1/3

=> *Entropy(2,1) = -1 * (2/3*log$_2$(2/3) + 1/3*log$_2$(1/3)) = -1 (-0.117 + -0.159) = 0.276*

    - X = 0 & Output = A: 0
    - X = 0 & Output = B: 1

=> *Entropy(0,1) = -1 * (0 + 1*log$_2$(1))* = 0

⇒ Entropy(Target, X) = 3/4*Entropy(2,1) + 1/4*Entropy(0,1) = ¾ * 0.276 = 0.207

⇒ Gain = Entropy(Target) - Entropy(Target, X) = 1 – 0.207 = 0.793 (Với Entropy(Target (2, 2) = -1 * (p_A *log$_2$(p_A) + p_B*log$_2$(p_B)) = 1)

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B

Decision Tree

- Tính Entropy cho thuộc tính
  - Làm tương tự cho các thuộc tính còn lại
  - So sánh Gain Information của các thuộc tính => chọn thuộc tính có Gain lớn nhất để chia cây

Decision Tree

- Gini

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

  - Tính gini cho từng thuộc tính, theo class kết quả.
  - Thuộc tính nào trong nhóm các thuộc tính có gini bé nhất => chọn thuộc tính đó để chia nhánh cho cây.

https://[illegible].com/2017/01/30/how-decision-tree-algorithm-work s/

Decision Tree

- Tính gini index cho X
  - X: mẫu 1 = 3, mẫu 0 = 1
    - X = 1 & output = A: 2/3
    - X = 1 & Output = B: 1/3

    => Gini(3,1) = 1 – (2/3*2/3 + 1/3*1/3) = 0.555
    - X = 0 & Output = A: 0
    - X = 0 & Output = B: 1

    => Gini(0,1) = 1 – (0 + 1) = 1

    => Gini(Target, X) = 3/4*0.555 + 1/4*1 = 0.666

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B

Decision Tree

- Tính gini index cho thuộc tính
  - Làm tương tự cho các thuộc tính còn lại
  - So sánh gini của các thuộc tính => chọn thuộc tính có gini nhỏ nhất để chia cây

Decision Tree

❑ Triển khai Decision Tree

- pyspark.ml.classification.DecisionTreeClassifier

http://localhost:8888/notebooks/Chapter04/demo_Decision_Tree_flights_new.ipynb

Decision Tree

- Ví dụ: Xây dựng Decision Tree model để dự đoán flight delay từ 'mon', 'dom', 'dow', 'carrier', 'org', 'km' (= mile * 1.60934), 'depart', 'duration'
  - Đọc dữ liệu

```
data.show(3)
```

```
+---+---+---+-------+------+---+----+------+--------+-----+
|mon|dom|dow|carrier|flight|org|mile|depart|duration|delay|
+---+---+---+-------+------+---+----+------+--------+-----+
| 11| 20|  6|     US|    19|JFK|2153|  9.48|     351|   NA|
|  0| 22|  2|     UA|  1107|ORD| 316| 16.33|      82|   30|
|  2| 20|  4|     UA|   226|SFO| 337|  6.17|      82|   -8|
+---+---+---+-------+------+---+----+------+--------+-----+
only showing top 3 rows
```

Decision Tree

- Chuẩn hóa dữ liệu: "mile" => "km", tạo cột label từ cột "delay", chuyển "carrier" và "org" thành dữ liệu category

```
# Import the required function
from pyspark.sql.functions import round

# Convert 'mile' to 'km'
data = data.withColumn('km', round(data.mile * 1.60934, 0))

# Create 'label' column indicating whether flight delayed (1) or not (0)
data = data.withColumn('label', (data.delay >= 15).cast('integer'))

data.show(3)
```

```
+---+---+---+-------+---+----+------+--------+-----+------+-----+
|mon|dom|dow|carrier|org|mile|depart|duration|delay|    km|label|
+---+---+---+-------+---+----+------+--------+-----+------+-----+
| 11| 20|  6|     US|JFK|2153|  9.48|     351|   NA|3465.0| null|
|  0| 22|  2|     UA|ORD| 316| 16.33|      82|   30| 509.0|    1|
|  2| 20|  4|     UA|SFO| 337|  6.17|      82|   -8| 542.0|    0|
+---+---+---+-------+---+----+------+--------+-----+------+-----+
only showing top 3 rows
```


Decision Tree

- Chuẩn hóa dữ liệu (tt)

```
from pyspark.ml.feature import StringIndexer

# Create an indexer
indexer = StringIndexer(inputCol='carrier', outputCol='carrier_idx')

# Indexer identifies categories in the data
indexer_model = indexer.fit(data)

# Indexer creates a new column with numeric index values
data_indexed = indexer_model.transform(data)

# Repeat the process for the other categorical feature
data_indexed = StringIndexer(inputCol='org', outputCol='org_idx').fit(data_indexed).transform(data_indexed)

data_indexed.show(3)
```

```
+---+---+---+-------+---+----+------+--------+-----+------+-----+-----------+-------+
|mon|dom|dow|carrier|org|mile|depart|duration|delay|    km|label|carrier_idx|org_idx|
+---+---+---+-------+---+----+------+--------+-----+------+-----+-----------+-------+
| 11| 20|  6|     US|JFK|2153|  9.48|     351|   NA|3465.0| null|        6.0|    2.0|
|  0| 22|  2|     UA|ORD| 316| 16.33|      82|   30| 509.0|    1|        0.0|    0.0|
|  2| 20|  4|     UA|SFO| 337|  6.17|      82|   -8| 542.0|    0|        0.0|    1.0|
+---+---+---+-------+---+----+------+--------+-----+------+-----+-----------+-------+
only showing top 3 rows
```

Decision Tree

- Chuyển đổi dữ liệu

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

# Create an assembler object
assembler = VectorAssembler(inputCols=[
    'mon', 'dom', 'dow', 'carrier_idx', 'org_idx', 'km', 'depart', 'duration'
], outputCol='features')

data_pre = assembler.transform(data_indexed)

data_pre.show(3, False)
```

```
+---+---+---+-------+---+----+------+--------+-----+-----+-----+-----------+-------+-----------------------------------------------------------+
|mon|dom|dow|carrier|org|mile|depart|duration|delay|km   |label|carrier_idx|org_idx|features                                                   |
+---+---+---+-------+---+----+------+--------+-----+-----+-----+-----------+-------+-----------------------------------------------------------+
|11 |20 |6  |US     |JFK|2153|9.48  |351     |NA   |3465.0|null |6.0        |2.0    |[11.0,20.0,6.0,6.0,2.0,3465.0,9.48,351.0]                  |
|0  |22 |2  |UA     |ORD|316 |16.33 |82      |30   |509.0 |1    |0.0        |0.0    |[0.0,22.0,2.0,0.0,0.0,509.0,16.33,82.0]                    |
|2  |20 |4  |UA     |SFO|337 |6.17  |82      |-8   |542.0 |0    |0.0        |1.0    |[2.0,20.0,4.0,0.0,1.0,542.0,6.17,82.0]                     |
+---+---+---+-------+---+----+------+--------+-----+-----+-----+-----------+-------+-----------------------------------------------------------+
only showing top 3 rows
```

Decision Tree

- Chia dữ liệu train-test

```
final_data = data_pre.select("features","label")

final_data.show(5, False)
```

```
+-----------------------------------------+-----+
|features                                 |label|
+-----------------------------------------+-----+
|[0.0,22.0,2.0,0.0,0.0,509.0,16.33,82.0]  |1    |
|[2.0,20.0,4.0,0.0,1.0,542.0,6.17,82.0]   |0    |
|[9.0,13.0,1.0,1.0,0.0,1989.0,10.33,195.0]|0    |
|[5.0,2.0,1.0,0.0,1.0,885.0,7.98,102.0]   |0    |
|[7.0,2.0,6.0,1.0,0.0,1180.0,10.83,135.0] |1    |
+-----------------------------------------+-----+
only showing top 5 rows
```

```
train_data,test_data = final_data.randomSplit([0.8,0.2])
```

Decision Tree

- Xây dựng model

```
# Import the Decision Tree Classifier class
from pyspark.ml.classification import DecisionTreeClassifier

# Create a classifier object and fit to the training data
tree = DecisionTreeClassifier(featuresCol='features',
                              labelCol='label',
                              predictionCol='prediction')

# Fit the model to the data and call this tree_model
tree_model = tree.fit(train_data)
```

Decision Tree

- Đánh giá kết quả

```
# Check test dataset
test_model = tree_model.transform(test_data)

# Inspect results
test_model.select('label', 'prediction', 'probability').show(3, False)
```

```
+-----+----------+----------------------------------------+
|label|prediction|probability                             |
+-----+----------+----------------------------------------+
|1    |1.0       |[0.3525645049908971,0.6474354950091029]|
|0    |1.0       |[0.3525645049908971,0.6474354950091029]|
|0    |0.0       |[0.5365093499554764,0.4634906500445236]|
+-----+----------+----------------------------------------+
only showing top 3 rows
```

Decision Tree

Đánh giá kết quả (tt)

```
# Create a confusion matrix
test_model.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0| 1470|
|    0|       0.0| 2673|
|    1|       1.0| 3344|
|    0|       1.0| 1921|
+-----+----------+-----+
```

```
# Calculate the elements of the confusion matrix
TN = test_model.filter('prediction = 0 AND label = prediction').count()
TP = test_model.filter('prediction = 1 AND label = prediction').count()
FN = test_model.filter('prediction = 0 AND label != prediction').count()
FP = test_model.filter('prediction = 1 AND label != prediction').count()

# Accuracy measures the proportion of correct predictions
accuracy = (TN + TP) / (TN + TP + FN + FP)
print(accuracy)
```

```
0.639562074829932
```

Decision Tree

Lưu và load model

```
# Save model
tree_model.save('tree_model_Flights_50k')

from pyspark.ml.classification import DecisionTreeClassificationModel
# Load model from
tree_model2 = DecisionTreeClassificationModel.load('tree_model_Flights_50k')

# Predict new values (Assuming select test_data)
unlabeled_data = test_data.select('features')

predictions = tree_model2.transform(unlabeled_data)

predictions.show(3, False)
```

```
+-----------------------------------+-----------------+----------------------------------------------------------+----------+
|features                           |rawPrediction    |probability                                               |prediction|
+-----------------------------------+-----------------+----------------------------------------------------------+----------+
|(8,[1,5,6,7],[6.0,669.0,9.92,89.0])|[5616.0,10313.0]|[0.3525645049908971,0.6474354950091029]|1.0       |
|(8,[1,5,6,7],[6.0,948.0,17.83,111.0])|[5616.0,10313.0]|[0.3525645049908971,0.6474354950091029]|1.0       |
|(8,[1,5,6,7],[13.0,1617.0,9.17,153.0])|[1205.0,1041.0] |[0.536509349955476 4,0.4634906500445236]|0.0       |
+-----------------------------------+-----------------+----------------------------------------------------------+----------+
only showing top 3 rows
```

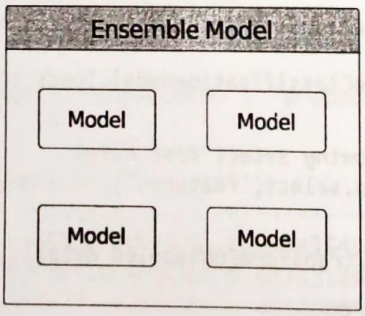
Nội dung

1. Decision Tree
2. Random Forest
3. Gradient-Boosted Trees

Random Forest

❑ Ensemble

- Là một bộ các mô hình (collection of models)

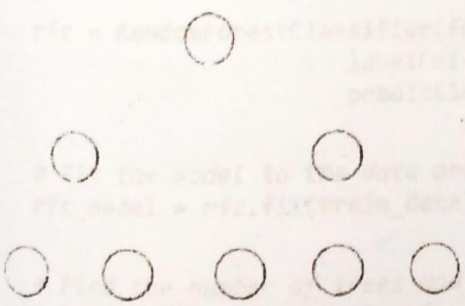


- "Wisdom of the Crowd" – ý kiến tập thể của một nhóm tốt hơn một ý kiến duy nhất

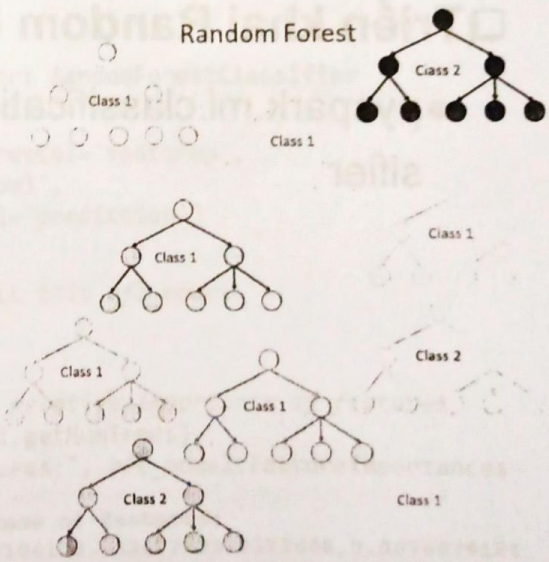
❑Random Forest

- Là một bộ các mô hình Decision Tree
- Để tạo ra sự đa dạng mô hình
  - Mỗi cây được huấn luyện trên một tập con ngẫu nhiên (random subset) của dữ liệu
  - Random subset của các thuộc tính được sử dụng để phân tách cây tại mỗi node
- Trong "rừng" không có hai cây giống nhau

Single Decision Tree



Random Forest



https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523ba65117

Random Forest

- Để cải thiện hiệu suất, chúng ta có thể sử dụng nhiều cây với mẫu các tính năng ngẫu nhiên được chọn làm phần tách.
- Mẫu tính năng ngẫu nhiên mới được chọn cho mỗi cây ở mỗi lần phân tách.
- Có thể làm việc với cả classification & regression

Random Forest

❑ Triển khai Random Forest

- pyspark.ml.classification.RandomForestClassifier

http://localhost:8888/notebooks/Chapter3/demo_Decision_Tree_flights_new.ipynb

Random Forest

- Ví dụ: Xây dựng Random Forest model để dự đoán flight delay từ 'mon', 'dom', 'dow', 'carrier', 'org', 'km' (= mile * 1.60934), 'depart', 'duration'
  - Các bước: Đọc dữ liệu, chuẩn bị dữ liệu, chia dữ liệu làm tương tự như trên Decision Tree

Random Forest

- Xây dựng model

```
from pyspark.ml.classification import RandomForestClassifier

rfc = RandomForestClassifier(featuresCol='features',
                             labelCol='label',
                             predictionCol='prediction')

# Fit the model to the data and call this rfc_model
rfc_model = rfc.fit(train_data)

# Find the number of trees and the relative importance of features
print("Number of trees:", rfc_model.getNumTrees)
print("Relative importance of features:", rfc_model.featureImportances
```

```
Number of trees: 20 Relative importance of features:
(8,[0,1,2,3,4,5,6,7],[0.2448926175081041,0.01249729932373608,0.0076679181
9346727 7,0.07765796599404604,0.20276452363590625,0.0405179280195633,0.354
5246436108443,0.059477103714332 83])
```

Random Forest

Đánh giá kết quả

```
# Check test dataset
rfc_test_model = rfc_model.transform(test_data)

# Inspect results
rfc_test_model.select('label', 'prediction', 'probability').show(3, False)
```

```
+-----+----------+----------------------------------------+
|label|prediction|probability                             |
+-----+----------+----------------------------------------+
|1    |1.0       |[0.42123255007446103,0.5787674499255389]|
|0    |1.0       |[0.378691082264147,0.6213089177358531]  |
|0    |0.0       |[0.5554165418316999,0.4445834581683002] |
+-----+----------+----------------------------------------+
only showing top 3 rows
```

Random Forest

Đánh giá kết quả (tt)

```
# Create a confusion matrix
rfc_test_model.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0| 1693|
|    0|       0.0| 2927|
|    1|       1.0| 3121|
|    0|       1.0| 1667|
+-----+----------+-----+
```

```
# Calculate the elements of the confusion matrix
TN = rfc_test_model.filter('prediction = 0 AND label = prediction').count()
TP = rfc_test_model.filter('prediction = 1 AND label = prediction').count()
FN = rfc_test_model.filter('prediction = 0 AND label != prediction').count()
FP = rfc_test_model.filter('prediction = 1 AND label != prediction').count()

# Accuracy measures the proportion of correct predictions
accuracy = (TN + TP) / (TN + TP + FN + FP)
print(accuracy)
```

```
0.6428571428571429
```

Random Forest

- Lưu và load model

```
# Save model
rfc_model.save('rfc_model_Flights_50k')

from pyspark.ml.classification import RandomForestClassificationModel
# Load model from
rfc_model2 = RandomForestClassificationModel.load('rfc_model_Flights_50k')

# Predict new values (Assuming select test_data)
unlabeled_data = test_data.select('features')

predictions = rfc_model2.transform(unlabeled_data)

predictions.show(3, False)
```


Nội dung

1. Decision Tree
2. Random Forest
3. **Gradient-Boosted Trees**

Gradient-Boosted Trees

❑Gradient boosting bao gồm 3 yếu tố:

- Loss function được tối ưu hóa.
- Weak learner đưa ra các dự đoán.
- Một additive model thêm các weak learner để giảm thiểu (minimize) loss function.

Gradient-Boosted Trees

- Loss Function
  - Là function/equation dùng để xác định "giá trị khác biệt" ("far off") của các dự đoán.
  - Ví dụ: Regression có thể sử dụng squared error, classification có thể sử dụng logarithmic loss.

Gradient-Boosted Trees

- Weak Learner
  - Các Decision tree được sử dụng như weak learner trong gradient boosting.
  - Có thể áp dụng ràng buộc cho những weak learner: chẳng hạn như số lượng tối đa của các layer, node, split hoặc leaf node.

Gradient-Boosted Trees

- Additive Model
  - Từng cây một được thêm vào và các cây hiện có trong model không bị thay đổi.
  - Một gradient descent procedure được sử dụng để giảm thiểu loss khi thêm cây.

❑Gradient-Boosted Trees

- Huấn luyện một weak model **m** sử dụng các data sample được vẽ theo một số phân bố trọng số (weight distribution)
- Tăng trọng số của các mẫu bị phân loại sai theo model m và giảm trọng số của các mẫu được phân loại chính xác theo model m
- Huấn luyện weak model tiếp theo bằng cách sử dụng các mẫu được vẽ theo phân bố trọng số đã cập nhật.

- Theo cách này, thuật toán luôn huấn luyện các model sử dụng các mẫu dữ liệu "khó học" trong các lần trước, dẫn đến một nhóm các model tốt trong việc học các "phần" training data khác nhau..

Gradient-Boosted Trees

- Boosting algorithm lặp lại:
  - Xây dựng một Decision Tree và thêm vào ensemble.
  - Dự đoán label cho từng thực thể huấn luyện (training instance) sử dụng ensemble.
  - So sánh các dự đoán với các label đã biết
  - Tập trung vào các thực thể huấn luyện có dự đoán không chính xác
  - Trở về bước đầu tiên
- Model được cải thiện sau mỗi lần lặp.

Gradient-Boosted Trees

- **Triển khai Gradient-Boosted Trees**
  - pyspark.ml.classification.GBTClassifier

Gradient-Boosted Trees

- Ví dụ: Xây dựng Gradient-Boosted Trees model để dự đoán flight delay từ 'mon', 'dom', 'dow', 'carrier', 'org', 'km' (= mile * 1.60934), 'depart', 'duration'
  - Các bước: Đọc dữ liệu, chuẩn bị dữ liệu, chia dữ liệu làm tương tự như trên Decision Tree

Gradient-Boosted Trees

- Xây dựng model

```
from pyspark.ml.classification import GBTClassifier

gbt = GBTClassifier(featuresCol='features',
                    labelCol='label',
                    predictionCol='prediction')

# Fit the model to the data and call this gbt_model
gbt_model = gbt.fit(train_data)

# Find the number of trees and the relative importance of features
print("Number of trees:", gbt_model.getNumTrees)
print("Relative importance of features:", gbt_model.featureImportances)
```

```
Number of trees: 20 Relative importance of features:
(8,[0,1,2,3,4,5,6,7],[0.1612405509388622,0.184531478887713,0.15117138
747542289,0.09933196433773392,0.1408367162029886,0.06744481401446742,
0.1419597963384068,0.05348329180440522])
```

Gradient-Boosted Trees

- Đánh giá kết quả

```
# Check test dataset
gbt_test_model = gbt_model.transform(test_data)

# Inspect results
gbt_test_model.select('label', 'prediction', 'probability').show(3, False)
```

```
+-----+----------+----------------------------------------+
|label|prediction|probability                             |
+-----+----------+----------------------------------------+
|1    |1.0       |[0.37817251829339604,0.621827481706604] |
|0    |1.0       |[0.2171799937001399,0.78282000629986]   |
|0    |1.0       |[0.49526141794239104,0.5047385820576089]|
+-----+----------+----------------------------------------+
only showing top 3 rows
```


Gradient-Boosted Trees

- Đánh giá kết quả (tt)

```
# Create a confusion matrix
gbt_test_model.groupBy('label', 'prediction').count().show()
```

```
+-----+----------+-----+
|label|prediction|count|
+-----+----------+-----+
|    1|       0.0| 1379|
|    0|       0.0| 2787|
|    1|       1.0| 3435|
|    0|       1.0| 1807|
+-----+----------+-----+
```

```
# Calculate the elements of the confusion matrix
TN = gbt_test_model.filter('prediction = 0 AND label = prediction').count()
TP = gbt_test_model.filter('prediction = 1 AND label = prediction').count()
FN = gbt_test_model.filter('prediction = 0 AND label != prediction').count()
FP = gbt_test_model.filter('prediction = 1 AND label != prediction').count()

# Accuracy measures the proportion of correct predictions
accuracy = (TN + TP) / (TN + TP + FN + FP)
print(accuracy)
```

```
0.6613520408163265
```

Gradient-Boosted Trees

- Lưu và load model

```
# Save model
gbt_model.save('gbt_model_Flights_50k')

from pyspark.ml.classification import GBTClassificationModel
# Load model from
gbt_model2 = GBTClassificationModel.load('gbt_model_Flights_50k')

# Predict new values (Assuming select test_data)
unlabeled_data = test_data.select('features')

predictions = gbt_model2.transform(unlabeled_data)

predictions.select('features', 'probability', 'prediction').show(3, False)
```

```
+-------------------------------------+-----------------------------------------+----------+
|features                             |probability                              |prediction|
+-------------------------------------+-----------------------------------------+----------+
|(8,[1,5,6,7],[6.0,669.0,9.92,89.0])  |[0.37817251829339604,0.621827481706604]  |1.0       |
|(8,[1,5,6,7],[6.0,948.0,17.83,111.0])|[0.2171799937001399,0.78282000629986]    |1.0       |
|(8,[1,5,6,7],[13.0,1617.0,9.17,153.0])|[0.49526141794239104,0.5047385820576089]|1.0       |
+-------------------------------------+-----------------------------------------+----------+
only showing top 3 rows
```


Gradient-Boosted Trees

- So sánh kết quả 3 model: Decision Tree model, Random Forest model, GBT model

- Với MulticlassClassificationEvaluator

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

dtc_predictions = tree_model.transform(test_data)
rfc_predictions = rfc_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)

# Select (prediction, true label) and compute test error
acc_evaluator = MulticlassClassificationEvaluator(labelCol="label",
                                                  predictionCol="prediction",
                                                  metricName="accuracy")
```

Gradient-Boosted Trees


```
dtc_acc = acc_evaluator.evaluate(dtc_predictions)
rfc_acc = acc_evaluator.evaluate(rfc_predictions)
gbt_acc = acc_evaluator.evaluate(gbt_predictions)

print("Results")
print('-'*60)
print('A single decision tree has an accuracy of: {0:2.2f}%'.format(dtc_acc*100))
print('-'*60)
print('A random forest ensemble has an accuracy of: {0:2.2f}%'.format(rfc_acc*100))
print('-'*60)
print('A ensemble using GBT has an accuracy of: {0:2.2f}%'.format(gbt_acc*100))
```

```
Results
------------------------------------------------------------
A single decision tree has an accuracy of: 63.96%
------------------------------------------------------------
A random forest ensemble has an accuracy of: 64.29%
------------------------------------------------------------
A ensemble using GBT has an accuracy of: 66.14%
```

Gradient-Boosted Trees

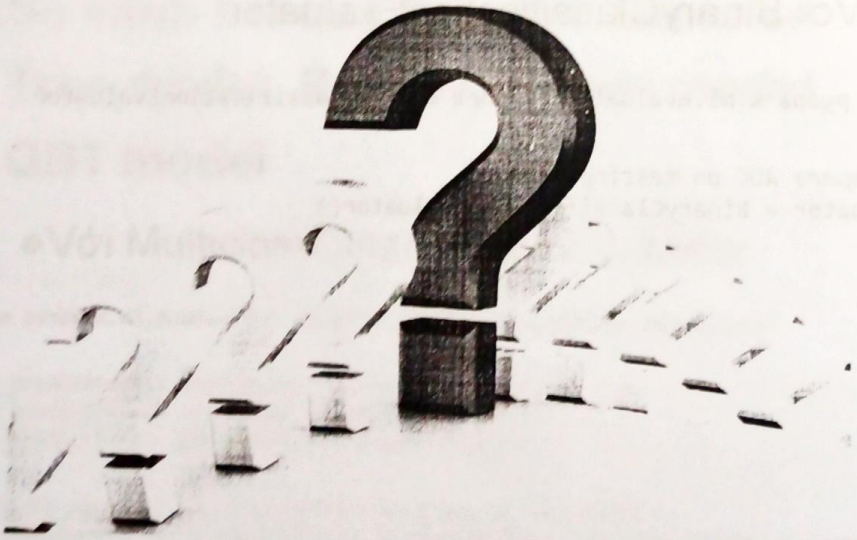
- Với BinaryClassificationEvaluator

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Compare AUC on testing data
evaluator = BinaryClassificationEvaluator()
```

Gradient-Boosted Trees

```
dtc_acc_2 = evaluator.evaluate(dtc_predictions)
rfc_acc_2 = evaluator.evaluate(rfc_predictions)
gbt_acc_2 = evaluator.evaluate(gbt_predictions)


print("Results")
print('-'*60)
print('A single decision tree has an accuracy of: {0:2.2f}%'.format(dtc_acc_2*100))
print('-'*60)
print('A random forest ensemble has an accuracy of: {0:2.2f}%'.format(rfc_acc_2*100))
print('-'*60)
print('A ensemble using GBT has an accuracy of: {0:2.2f}%'.format(gbt_acc_2*100))
```

```
Results
------------------------------------------------------------
A single decision tree has an accuracy of: 64.26%
------------------------------------------------------------
A random forest ensemble has an accuracy of: 69.30%
------------------------------------------------------------
A ensemble using GBT has an accuracy of: 72.48%
```

Chapter 8: Tree Methods

Ex1: College

We will be using a college dataset (College.csv) to try to classify colleges as Private or Public based off these features:

```
Private: A factor with levels No and Yes indicating private or public un
iversity
Apps: Number of applications received
Accept: Number of applications accepted
Enroll: Number of new students enrolled
Top10perc: Pct. new students from top 10% of H.S. class
Top25perc: Pct. new students from top 25% of H.S. class
F.Undergrad: Number of fulltime undergraduates
P.Undergrad: Number of parttime undergraduates
Outstate: Out-of-state tuition
Room.Board: Room and board costs
Books: Estimated book costs
Personal: Estimated personal spending
PhD: Pct. of faculty with Ph.D.'s
Terminal: Pct. of faculty with terminal degree
S.F.Ratio: Student/faculty ratio
perc.alumni: Pct. alumni who donate
Expend: Instructional expenditure per student
Grad.Rate: Graduation rate
```

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('treecode').getOrCreate()
```

In [3]:
```
# Load training data
data = spark.read.csv('College.csv',inferSchema=True,header=True)
```

In [4]:
```
data.count()
```

Out[4]:
```
777
```

In [5]:
```
data.printSchema()
```

```
root
 |-- School: string (nullable = true)
 |-- Private: string (nullable = true)
 |-- Apps: integer (nullable = true)
 |-- Accept: integer (nullable = true)
 |-- Enroll: integer (nullable = true)
 |-- Top10perc: integer (nullable = true)
 |-- Top25perc: integer (nullable = true)
 |-- F_Undergrad: integer (nullable = true)
 |-- P_Undergrad: integer (nullable = true)
 |-- Outstate: integer (nullable = true)
 |-- Room_Board: integer (nullable = true)
 |-- Books: integer (nullable = true)
 |-- Personal: integer (nullable = true)
 |-- PhD: integer (nullable = true)
 |-- Terminal: integer (nullable = true)
 |-- S_F_Ratio: double (nullable = true)
 |-- perc_alumni: integer (nullable = true)
 |-- Expend: integer (nullable = true)
 |-- Grad_Rate: integer (nullable = true)
```

In [6]:
```
data.head()
```

Out[6]:
```
Row(School='Abilene Christian University', Private='Yes', Apps=1660, Accept=123
2, Enroll=721, Top10perc=23, Top25perc=52, F_Undergrad=2885, P_Undergrad=537, O
utstate=7440, Room_Board=3300, Books=450, Personal=2200, PhD=70, Terminal=78, S
_F_Ratio=18.1, perc_alumni=12, Expend=7041, Grad_Rate=60)
```

Spark Formatting of Data

In [7]:
```
# It needs to be in the form of two columns
# ("label","features")

# Import VectorAssembler and Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [8]:
```
data.columns
```

Out[8]:
```
['School',
 'Private',
 'Apps',
 'Accept',
 'Enroll',
 'Top10perc',
 'Top25perc',
 'F_Undergrad',
 'P_Undergrad',
 'Outstate',
 'Room_Board',
 'Books',
 'Personal',
 'PhD',
 'Terminal',
 'S_F_Ratio',
 'perc_alumni',
 'Expend',
 'Grad_Rate']
```

In [9]:
```
assembler = VectorAssembler(
  inputCols=['Apps',
             'Accept',
             'Enroll',
             'Top10perc',
             'Top25perc',
             'F_Undergrad',
             'P_Undergrad',
             'Outstate',
             'Room_Board',
             'Books',
             'Personal',
             'PhD',
             'Terminal',
             'S_F_Ratio',
             'perc_alumni',
             'Expend',
             'Grad_Rate'],
              outputCol="features")
```

In [10]:
```
output = assembler.transform(data)
```

Deal with Private column being "yes" or "no"

In [11]:
```
from pyspark.ml.feature import StringIndexer
```

In [12]:
```
indexer = StringIndexer(inputCol="Private", outputCol="PrivateIndex")
output_fixed = indexer.fit(output).transform(output)
```

In [13]:
```
final_data = output_fixed.select("features",'PrivateIndex')
```

In [14]:
```
train_data,test_data = final_data.randomSplit([0.7,0.3])
```

The Classifiers

In [15]:
```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import GBTClassifier,RandomForestClassifier
from pyspark.ml import Pipeline
```

Create all three models:

In [16]:
```
# Use mostly defaults to make this comparison "fair"

dtc = DecisionTreeClassifier(labelCol='PrivateIndex',featuresCol='features')
rfc = RandomForestClassifier(labelCol='PrivateIndex',featuresCol='features')
gbt = GBTClassifier(labelCol='PrivateIndex',featuresCol='features')
```

Train all three models:

In [17]:
```
# Train the models (its three models, so it might take some time)
dtc_model = dtc.fit(train_data)
rfc_model = rfc.fit(train_data)
gbt_model = gbt.fit(train_data)
```

Model Comparison

Let's compare each of these models!

In [18]:
```
dtc_predictions = dtc_model.transform(test_data)
rfc_predictions = rfc_model.transform(test_data)
gbt_predictions = gbt_model.transform(test_data)
```

Evaluation Metrics:

In [19]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [20]:
```
# Select (prediction, true label) and compute test error
acc_evaluator = MulticlassClassificationEvaluator(labelCol="PrivateIndex",
                                                  predictionCol="prediction",
                                                  metricName="accuracy")
```

In [21]:
```
dtc_acc = acc_evaluator.evaluate(dtc_predictions)
rfc_acc = acc_evaluator.evaluate(rfc_predictions)
gbt_acc = acc_evaluator.evaluate(gbt_predictions)
```

In [22]:
```
print("Results:")
print('-'*80)
print('A single decision tree - accuracy: {0:2.2f}%'.format(dtc_acc*100))
print('-'*80)
print('A random forest ensemble - accuracy: {0:2.2f}%'.format(rfc_acc*100))
print('-'*80)
print('A ensemble using GBT - accuracy: {0:2.2f}%'.format(gbt_acc*100))
```

```
Results:
--------------------------------------------------------------------------------
-
A single decision tree - accuracy: 92.51%
--------------------------------------------------------------------------------
-
A random forest ensemble - accuracy: 95.59%
--------------------------------------------------------------------------------
-
A ensemble using GBT - accuracy: 92.95%
```

Optional Assignment - play around with the parameters of each of these models, can you squeeze some more accuracy out of them? Or is the data the limiting factor?

Chapter 8: Tree Methods

Ex2: Consulting Project - SOLUTION

You've been hired by a dog food company to try to predict why some batches of their dog food are spoiling much quicker than intended! Unfortunately this Dog Food company hasn't upgraded to the latest machinery, meaning that the amounts of the five preservative chemicals they are using can vary a lot, but which is the chemical that has the strongest effect? The dog food company first mixes up a batch of preservative that contains 4 different preservative chemicals (A,B,C,D) and then is completed with a "filler" chemical. The food scientists beelive one of the A,B,C, or D preservatives is causing the problem, but need your help to figure out which one! Use Machine Learning with RF to find out which parameter had the most predicitive power, thus finding out which chemical causes the early spoiling! So create a model and then find out how you can decide which chemical is the problem!

- Pres_A : Percentage of preservative A in the mix
- Pres_B : Percentage of preservative B in the mix
- Pres_C : Percentage of preservative C in the mix
- Pres_D : Percentage of preservative D in the mix
- Spoiled: Label indicating whether or not the dog food batch was spoiled.

Think carefully about what this problem is really asking you to solve. While we will use Machine Learning to solve this, it won't be with your typical train/test split workflow.

In [1]:
```
import findspark
findspark.init()
```

In [2]:
```
#Tree methods Example
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('dogfood').getOrCreate()
```

In [3]:
```
# Load training data
data = spark.read.csv('dog_food.csv',inferSchema=True,header=True)
```

In [4]:
```
data.count()
```

Out[4]:
```
490
```

In [5]:
```
data.printSchema()
```

```
root
 |-- A: integer (nullable = true)
 |-- B: integer (nullable = true)
 |-- C: double (nullable = true)
 |-- D: integer (nullable = true)
 |-- Spoiled: double (nullable = true)
```

In [6]:
```
data.head()
```

Out[6]:
```
Row(A=4, B=2, C=12.0, D=3, Spoiled=1.0)
```

In [7]:
```
data.describe().show()
```

```
+-------+------------------+------------------+------------------+------------------+--------------------+
|summary|                 A|                 B|                 C|                 D|             Spoiled|
+-------+------------------+------------------+------------------+------------------+--------------------+
|  count|               490|               490|               490|               490|                 490|
|   mean| 5.53469387755102| 5.504081632653061| 9.126530612244897| 5.579591836734694| 0.2857142857142857|
| stddev|2.9515204234399057|2.8537966089662063|2.0555451971054275|2.8548369309982857|0.45221563164613465|
|    min|                 1|                 1|               5.0|                 1|                 0.0|
|    max|                10|                10|              14.0|                10|                 1.0|
+-------+------------------+------------------+------------------+------------------+--------------------+
```

In [8]:
```
# Import VectorAssembler and Vectors
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [9]:
```
data.columns
```

Out[9]:
```
['A', 'B', 'C', 'D', 'Spoiled']
```

In [10]:
```
assembler = VectorAssembler(inputCols=['A', 'B', 'C', 'D'],
                            outputCol="features")
```

In [11]:
```
output = assembler.transform(data)
```

In [12]:
```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import DecisionTreeClassifier
```

In [13]:
```
rfc = DecisionTreeClassifier(labelCol='Spoiled',featuresCol='features')
```

In [14]:
```
output.printSchema()
```

```
root
 |-- A: integer (nullable = true)
 |-- B: integer (nullable = true)
 |-- C: double (nullable = true)
 |-- D: integer (nullable = true)
 |-- Spoiled: double (nullable = true)
 |-- features: vector (nullable = true)
```

In [15]:
```
final_data = output.select('features','Spoiled')
final_data.head()
```

Out[15]:
```
Row(features=DenseVector([4.0, 2.0, 12.0, 3.0]), Spoiled=1.0)
```

In [16]:
```
rfc_model = rfc.fit(final_data)
```

In [17]:
```
rfc_model.featureImportances
```

Out[17]:
```
SparseVector(4, {1: 0.0019, 2: 0.9832, 3: 0.0149})
```

Feature at index 2 (Chemical C) is by far the most important feature, meaning it is causing the early spoilage! This is a pretty interesting use of a machine learning model in an alternative way!

Chapter 8: Tree Models

Ex3: KDD 10%

Software to detect network intrusions protects a computer network from unauthorized users, including perhaps insiders. The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between 'bad' connections, called intrusions or attacks, and 'good' normal connections.

Read more: https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data (https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data)

Requirement: We will be using a KDD dataset to try to classify a connection as 'normal.' or others.

In [1]:
```
import findspark
findspark.init()
```

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

In [3]:
```
spark = SparkSession.builder.appName('kdd').getOrCreate()
```

In [4]:
```
df = spark.read.csv('kdd_data/kddcup.data_10_percent.gz',
                    inferSchema=True, header=False)
```

In [5]:
```
df.count()
```

Out[5]:
```
494021
```

In [9]:
```
from pyspark.sql.functions import col
```

In [10]:
```
str(df.columns)
```

Out[10]:
```
"['_c0', '_c1', '_c2', '_c3', '_c4', '_c5', '_c6', '_c7', '_c8', '_c9', '_c10',
'_c11', '_c12', '_c13', '_c14', '_c15', '_c16', '_c17', '_c18', '_c19', '_c20',
'_c21', '_c22', '_c23', '_c24', '_c25', '_c26', '_c27', '_c28', '_c29', '_c30',
'_c31', '_c32', '_c33', '_c34', '_c35', '_c36', '_c37', '_c38', '_c39', '_c40',
'_c41']"
```

In [11]:
```
cols = ['_c0', '_c4', '_c5', '_c6', '_c7', '_c8', '_c9', '_c10',
        '_c11', '_c12', '_c13', '_c14', '_c15', '_c16', '_c17',
        '_c18', '_c19', '_c20', '_c21', '_c22', '_c23', '_c24',
        '_c25', '_c26', '_c27', '_c28', '_c29', '_c30', '_c31',
        '_c32', '_c33', '_c34', '_c35', '_c36', '_c37', '_c38',
        '_c39', '_c40']
for col_name in cols:
    df = df.withColumn(col_name, col(col_name).cast('float'))
```

In [12]:
```
df.show(3)
```

```
+---+---+----+---+-----+------+---+---+---+---+----+----+----+----+----+----+--
---+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+-
|_c0|_c1| _c2|_c3|  _c4|   _c5|_c6|_c7|_c8|_c9|_c10|_c11|_c12|_c13|_c14|_c15|_c
16|_c17|_c18|_c19|_c20|_c21|_c22|_c23|_c24|_c25|_c26|_c27|_c28|_c29|_c30|_c31|_
c32|_c33|_c34|_c35|_c36|_c37|_c38|_c39|_c40|   _c41|
+---+---+----+---+-----+------+---+---+---+---+----+----+----+----+----+----+--
---+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+-
|0.0|tcp|http| SF|181.0|5450.0|0.0|0.0|0.0|0.0| 0.0| 1.0| 0.0| 0.0| 0.0| 0.0|
0.0| 0.0| 0.0| 0.0| 0.0| 0.0| 8.0| 8.0| 0.0| 0.0| 0.0| 0.0| 1.0| 0.0| 0.0| 9.0|
9.0| 1.0| 0.0|0.11| 0.0| 0.0| 0.0| 0.0|normal.|
|0.0|tcp|http| SF|239.0| 486.0|0.0|0.0|0.0|0.0| 0.0| 1.0| 0.0| 0.0| 0.0| 0.0|
0.0| 0.0| 0.0| 0.0| 0.0| 0.0| 8.0| 8.0| 0.0| 0.0| 0.0| 0.0| 1.0| 0.0| 0.0|19.0|
19.0| 1.0| 0.0|0.05| 0.0| 0.0| 0.0| 0.0| 0.0|normal.|
|0.0|tcp|http| SF|235.0|1337.0|0.0|0.0|0.0|0.0| 0.0| 1.0| 0.0| 0.0| 0.0| 0.0|
0.0| 0.0| 0.0| 0.0| 0.0| 0.0| 8.0| 8.0| 0.0| 0.0| 0.0| 0.0| 1.0| 0.0| 0.0|29.0|
29.0| 1.0| 0.0|0.03| 0.0| 0.0| 0.0| 0.0| 0.0|normal.|
+---+---+----+---+-----+------+---+---+---+---+----+----+----+----+----+----+--
---+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+-
only showing top 3 rows
```

In [13]:
```
df.groupBy('_c41').count().show(30)
```

```
+------------------+------+
|               _c41| count|
+------------------+------+
|    warezmaster.|    20|
|          smurf.|280790|
|           pod.|   264|
|          imap.|    12|
|          nmap.|   231|
|   guess_passwd.|    53|
|       ipsweep.|  1247|
|      portsweep.|  1040|
|          satan.|  1589|
|           land.|    21|
|     loadmodule.|     9|
|      ftp_write.|     8|
|buffer_overflow.|    30|
|        rootkit.|    10|
|    warezclient.|  1020|
|       teardrop.|   979|
|          perl.|     3|
|           phf.|     4|
|       multihop.|     7|
|        neptune.|107201|
|          back.|  2203|
|           spy.|     2|
|        normal.| 97278|
+----------------+------+
```

In [14]:
```
str(df.columns)
```

Out[14]:
```
"['_c0', '_c1', '_c2', '_c3', '_c4', '_c5', '_c6', '_c7', '_c8', '_c9', '_c10',
'_c11', '_c12', '_c13', '_c14', '_c15', '_c16', '_c17', '_c18', '_c19', '_c20',
'_c21', '_c22', '_c23', '_c24', '_c25', '_c26', '_c27', '_c28', '_c29', '_c30',
'_c31', '_c32', '_c33', '_c34', '_c35', '_c36', '_c37', '_c38', '_c39', '_c40',
'_c41']"
```

In [15]:
```
train_data,test_data = df.randomSplit([0.8,0.2])
```

In [16]:
```
test_data .groupBy('_c41').count().show(30)
```

```
+----------------+-----+
|             _c41|count|
+----------------+-----+
|    warezmaster.|    1|
|          smurf.|56131|
|            pod.|   68|
|           nmap.|   37|
|           imap.|    2|
|   guess_passwd.|    9|
|        ipsweep.|  251|
|      portsweep.|  208|
|          satan.|  337|
|           land.|    1|
|     loadmodule.|    3|
|buffer_overflow.|    9|
|    warezclient.|  196|
|       teardrop.|  162|
|          perl.|    1|
|       multihop.|    1|
|        neptune.|21600|
|           back.|  455|
|         normal.|19358|
+----------------+-----+
```

In [17]:
```
from pyspark.ml.feature import StringIndexer, OneHotEncoderEstimator
```

In [18]:
```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [19]:
```
# Import class for creating a pipeline
from pyspark.ml import Pipeline
```

In [20]:
```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import GBTClassifier, RandomForestClassifier
```

In [21]:
```
# Convert categorical strings to index values
indexer1 = StringIndexer(inputCol='_c1', outputCol='c1_idx')
indexer2 = StringIndexer(inputCol='_c2', outputCol='c2_idx')
indexer3 = StringIndexer(inputCol='_c3', outputCol='c3_idx')
indexer41 = StringIndexer(inputCol='_c41', outputCol='c41_idx')

# One-hot encode index values
onehot = OneHotEncoderEstimator(
    inputCols=['c1_idx', 'c2_idx', 'c3_idx'],
    outputCols=['c1_dummy', 'c2_dummy', 'c3_dummy']
)

# Assemble predictors into a single column
assembler = VectorAssembler(inputCols=['_c0', 'c1_dummy',
                                       'c2_dummy', 'c3_dummy',
                                       '_c4', '_c5', '_c6',
                                       '_c7', '_c8', '_c9',
                                       '_c10', '_c11', '_c12',
                                       '_c13', '_c14',
                                       '_c15', '_c16', '_c17',
                                       '_c18', '_c19',
                                       '_c20', '_c21', '_c22',
                                       '_c23', '_c24',
                                       '_c25', '_c26', '_c27',
                                       '_c28', '_c29',
                                       '_c30', '_c31', '_c32',
                                       '_c33', '_c34',
                                       '_c35', '_c36', '_c37',
                                       '_c38', '_c39', '_c40'],
                            outputCol='features')

# A linear regression object
dtc = DecisionTreeClassifier(featuresCol='features',
                             labelCol='c41_idx',
                             predictionCol='prediction')
```

In [22]:
```
# Construct a pipeline
pipeline = Pipeline(stages=[indexer1, indexer2, indexer3,
                            indexer41, onehot, assembler, dtc])
```

In [23]:
```
# Train the pipeline on the training data
pipeline = pipeline.fit(train_data)
```

In [24]:
```
# Make predictions on the testing data
predictions = pipeline.transform(test_data)
```

In [25]:
```
# Inspect results
predictions.select("prediction", "c41_idx").show(5)
```

```
+----------+-------+
|prediction|c41_idx|
+----------+-------+
|       5.0|    5.0|
|       5.0|    5.0|
|       5.0|    5.0|
|       5.0|    5.0|
|       5.0|    5.0|
+----------+-------+
only showing top 5 rows
```

In [26]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [29]:
```
# Select (prediction, true label) and compute test error
acc_evaluator = MulticlassClassificationEvaluator(labelCol="c41_idx",
                                                  predictionCol="prediction",
                                                  metricName="accuracy")
```

In [30]:
```
#important: need to cast to float type, and order by prediction, else it won't wo
preds_and_labels = predictions.select(['prediction','c41_idx'])\
                    .withColumn('c41_idx', col('c41_idx')\
                                .cast("float")).orderBy('prediction')
#select only prediction and label columns
preds_and_labels = preds_and_labels.select(['prediction','c41_idx'])
```

In [31]:
```
acc_evaluator.evaluate(preds_and_labels)
```

Out[31]:
```
0.9930587878174644
```

In [32]:
```
from pyspark.mllib.evaluation import MulticlassMetrics
```

In [33]:
```
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

In [34]:
```
# Confusion matrix
metrics = MulticlassMetrics(preds_and_labels.rdd.map(tuple))
```

In [35]:
```
# print(metrics.confusionMatrix().toArray())
```

In [36]:
```
import pandas as pd
pd.set_option('display.max_columns', 30)
```

In [37]:
```
matrix = pd.DataFrame(metrics.confusionMatrix().toArray())
```

In [38]: matrix

Out[38]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	56131.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	21492.0	102.0	0.0	4.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	19292.0	0.0	0.0	63.0	0.0	0.0	1.0	0.0	0.0	0.0	2.0	0.0	0.0
3	0.0	0.0	16.0	439.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.0	13.0	0.0	319.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	31.0	0.0	0.0	220.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	2.0	181.0	0.0	8.0	0.0	17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	184.0	0.0	0.0	12.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	162.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	67.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	37.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.0	2.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [39]:
```
# With accuracy, this model is very good.
# But with confusion matrix: Some classes, which only have a few samples, have wi
```

Make new prediction

In [40]:
```
# New data
df_new = spark.read.csv('kdd_data/kddcup.testdata.unlabeled_10_percent.gz',
                        inferSchema=True,
                        header=False)
```

In [41]: df_new.count()

Out[41]: 311029

In [42]:
```
str(df_new.columns)
```

Out[42]:
```
"['_c0', '_c1', '_c2', '_c3', '_c4', '_c5', '_c6', '_c7', '_c8', '_c9', '_c10',
'_c11', '_c12', '_c13', '_c14', '_c15', '_c16', '_c17', '_c18', '_c19', '_c20',
'_c21', '_c22', '_c23', '_c24', '_c25', '_c26', '_c27', '_c28', '_c29', '_c30',
'_c31', '_c32', '_c33', '_c34', '_c35', '_c36', '_c37', '_c38', '_c39', '_c4
0']"
```

In [43]:
```
# Make predictions on the testing data
predictions_new = pipeline.transform(df_new)
```

In [44]:
```
predictions_new.select('features', 'prediction').show()
```

```
+--------------------+----------+
|            features|prediction|
+--------------------+----------+
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[8,68,78,96,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[2,5,68,78,7...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[2,5,68,78,7...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[0,2,6,68,78...|       2.0|
|(115,[2,5,68,78,7...|       2.0|
|(115,[2,5,68,78,7...|       2.0|
|(115,[2,5,68,78,7...|       2.0|
|(115,[4,68,78,79,...|       2.0|
|(115,[4,68,78,79,...|       2.0|
+--------------------+----------+
only showing top 20 rows
```