Dr. Heni Bouhamed
Data Scientist, Maître Assistant à la Fseg-Université de Sfax
Formation Data Science
Module 7 : Apache Spark

TP3 Apache Spark : ML et Data Frames

1) Placer le fichier Iris1.csv dans D : si vous utiliser windows (dans le home si vous utiliser ubuntu).
2) Charger ce fichier avec : df = spark.read.load("D:\Iris1.csv", format="csv", sep=",",inferSchema="true", header="true").
3) Voici une autre possibilité pour charger le fichier avec : df1 = sqlContext.read.format('csv').options(header='true', inferschema='true').load('D:\Iris1.csv').
   Rq : inferSchema : infère automatiquement les types de colonnes.
4) Compter le nombre de ligne du data frame avec : df.count().
5) Afficher les 10 premières lignes du data frame avec : df.show(10).
6) Filtrer et afficher les lignes dont les petal_length sont strictement supérieures à 6 avec : df.filter(df["petal_length"]>6).show().
7) Compter les « species » par groupe avec : df.groupBy(df["species"]).count().show().
8) Afficher les 10 premières lignes avec : df.head(10).
9) Sélectionner les différentes « species » en utilisant une requête sql avec :
   df.registerTempTable("table")
   distinct_classes = sqlContext.sql("select distinct species from table").
   --Naive Bayes--
10) Importer Naives Bayes de ML avec : from pyspark.ml.classification import NaiveBayes.
11) Transformer le data frame df en indexant la variable classe « species » et en créant un vecteur de « features » avec :
    from pyspark.ml.feature import StringIndexer
    speciesIndexer = StringIndexer(inputCol="species", outputCol="speciesIndex")
    from pyspark.ml.feature import VectorAssembler
    vectorAssembler=VectorAssembler(inputCols=["petal_width","petal_length","sepal_width","sepal_length"], outputCol="features")
    data = vectorAssembler.transform(df)
    index_model = speciesIndexer.fit(data)
    data_indexed = index_model.transform(data)
12) Visualiser une partie du résultat avec : data_indexed.show(2).
13) Diviser aléatoirement les données en base d'apprentissage et en base de test avec : trainingData, testData = data_indexed.randomSplit([0.8, 0.2],0.0).
14) Configurer le modèle avec : nb = NaiveBayes().setFeaturesCol("features").setLabelCol("speciesIndex").setSmoothing(1.0).setModelType("multinomial").
15) Lancer l'apprentissage du modèle avec : model = nb.fit(trainingData).
16) Effectuer la classification de la base test avec : classifications = model.transform(testData) (classifications est un data frame contenant les toutes les données tests ainsi que les valeurs de prédictions).
17) Importer les fonctions d'évaluations des modèles de classification avec : from pyspark.ml.evaluation import MulticlassClassificationEvaluator.

18) Configurer l'évaluation avec la métrique accuracy avec : evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy").
19) Calculer le pourcentage de bonnes classifications avec : accuracy = evaluator.evaluate(classifications).
20) Afficher le résultat avec : print("Test set accuracy = " + str(accuracy))
21) #étape optionnelle : visualiser toutes les prédictions avec : classifications.select('prediction').show().
22) Configurer l'évaluation avec la métrique Recall avec : evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="weightedRecall").
23) Exécuter avec : recall = evaluator.evaluate(classifications).
24) Afficher le résultat avec : print("Test set recall = " + str(recall)).
    Rq : l'évaluation supporte aussi le f1score "f1" (par défaut) ainsi que le "weightedPrecision".
    --neural network--
25) Etape 2 et 11.
26) Diviser aléatoirement les données en base d'apprentissage et en base de test avec : trainingData, testData = data_indexed.randomSplit([0.8, 0.2],0.0).
27) Importer la fonction MultilayerPerceptronClassifier avec : from pyspark.ml.classification import MultilayerPerceptronClassifier
28) Choisir le nombre de layers adéquat avec : layers = [4, 5, 4, 3].
29) Configurer le modèle avec : nn = MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("speciesIndex").setFeaturesCol("features").setBlockSize(120).setSeed(1234).
30) Lancer l'apprentissage du modèle avec : model = nn.fit(trainingData).
31) Effectuer la classification de la base test avec : classifications = model.transform(testData).
32) Refaire les étapes de 17 à 20 pour l'évaluation avec :
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator.
    evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(classifications)
    print("Test set accuracy = " + str(accuracy))
33) Essayer une architecture plus profonde avec : layers = [4, 1000, 1000, 3].
34) Configurer le modèle avec les nouvelles couches : nn = MultilayerPerceptronClassifier().setLayers(layers).setLabelCol("speciesIndex").setFeaturesCol("features").setBlockSize(120).setSeed(1234).
35) Lancer l'apprentissage avec : model = nn.fit(trainingData).
36) Effectuer la classification de la base test avec : classifications = model.transform(testData).
37) Refaire les étapes de 17 à 20 pour l'évaluation avec :
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(classifications)
    print("Test set accuracy = " + str(accuracy))
    --Arbre de décision--
38) Répéter les étapes 2 et 11.

39) Diviser aléatoirement les données en base d'apprentissage et en base de test avec : trainingData, testData = data_indexed.randomSplit([0.8, 0.2],0.0).
40) Importer la fonction des arbres de décision avec : from pyspark.ml.classification import DecisionTreeClassifier
41) Configurer le modèle avec : dt = DecisionTreeClassifier().setLabelCol("speciesIndex").setFeaturesCol("features").
42) Lancer l'apprentissage avec : model = dt.fit(trainingData).
43) Effectuer la classification de la base test avec : classifications = model.transform(testData)
44) Refaire les étapes de 17 à 20 pour l'évaluation avec :
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(classifications)
    print("Test set accuracy = " + str(accuracy))
    -- random forest--
45) Répéter l'étape 2 et 11.
46) Diviser aléatoirement les données en base d'apprentissage et en base de test avec : trainingData, testData = data_indexed.randomSplit([0.8, 0.2],0.0).
47) Importer la fonction random forest avec : from pyspark.ml.classification import RandomForestClassifier.
48) Configurer le modèle avec : rf = RandomForestClassifier().setLabelCol("speciesIndex").setFeaturesCol("features").setNumTrees(40).
49) Lancer l'apprentissage avec : model = rf.fit(trainingData).
50) Effectuer la classification de la base test avec : classifications = model.transform(testData).
51) Refaire les étapes de 17 à 20 pour l'évaluation avec :
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(classifications)
    print("Test set accuracy = " + str(accuracy))
    --régression logistique--
52) Répéter l'étape 2 et 11.
53) Diviser aléatoirement les données en base d'apprentissage et en base de test avec : trainingData, testData = data_indexed.randomSplit([0.8, 0.2],0.0).
54) Importer la fonction régression logistique avec : from pyspark.ml.classification import LogisticRegression.
55) Configurer le modèle avec : lr = LogisticRegression().setLabelCol("speciesIndex").setFeaturesCol("features").
56) Lancer l'apprentissage avec : model = lr.fit(trainingData).
57) Effectuer la classification de la base test avec : classifications = model.transform(testData).
58) Refaire les étapes de 17 à 20 pour l'évaluation avec :
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    evaluator = MulticlassClassificationEvaluator(labelCol="speciesIndex", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(classifications)
    print("Test set accuracy = " + str(accuracy))