cf . install pyspark TP

```
from pyspark.mllib.clustering import KMeans
from numpy import array from math import
sqrt
# charger les données
data = sc.textFile("data/mllib/kmeans_data.txt")
#préparer les données
splitedData = data.map(lambda line: array([float(x) for x in line.split(' ')]))
# créer le modèle
clusters = KMeans.train(splitedData , 3, maxIterations=10)
# afficher les centres des clusters
print(clusters.clusterCenters)
```

```
from pyspark.mllib.clustering import KMeans
from numpy import array from math import
sqrt
# charger les données
data = sc.textFile("/Users/ebenahme/Downloads/ds/3D_spatial_network.txt")
#préparer les données
 parsedData = data.map(lambda line: array([float(x) for x in line.split(',')]))
# créer le modèle
clusters = KMeans.train(parsedData, 3, maxIterations=20)
# afficher les centres des clusters
 clusters.clusterCenters
```

```
from pyspark.mllib.fpm import FPGrowth
# charger les données
data = sc.textFile("data/mllib/sample_fpgrowth.txt")
#Préparer les données
splitedData= data.map(lambda line: line.strip().split(' '))
# Appliquer FP-Growth
model = FPGrowth.train(splitedData, minSupport=0.2, numPartitions=10)
result = model.freqItemsets().collect()
#Afficher le résultat
for item in result:
    print(item)
```

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree from
pyspark.mllib.util import MLUtils
# charger les données
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt').cache()
# Appliquer les arbres de décisions
model = DecisionTree.trainClassifier(data, numClasses=2,
categoricalFeaturesInfo={},
                                     impurity='gini', maxDepth=5)
#Afficher le modèle
print(model.toDebugString())
# Evaluer le résultat
predictions = model.predict(data.map(lambda x: x.features))
predictions.collect()
labelsAndPredictions = data.map(lambda lp: lp.label).zip(predictions)
labelsAndPredictions.collect()
trainErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() /
float(data.count())
print('Training Error = ' + str(trainErr))
print('Learned classification tree model:')
print(model)
```

```
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD,
LinearRegressionModel
def mapping(line):
line= line.replace('-1.000000', '0.000000')
line= line.replace('-', '')
return line
def toWriteFile(data):
        return ''.join(str(d) for d in data)
data = sc.textFile("data/mllib/duke 2.csv")
parsed=data.map(mapping)
parsed.collect()
lines=parsed.map(toWriteFile)
lines.saveAsTextFile('ex1.txt')
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree from
pyspark.mllib.util import MLUtils
# charger les données
data = MLUtils.loadLibSVMFile(sc, '/data/mllib/ex1.txt/part-00000').cache()
# Appliquer les arbres de décisions
model = DecisionTree.trainClassifier(data, numClasses=2,
categoricalFeaturesInfo={},impurity='gini',  maxDepth=7)
#Afficher le modèle
print(model.toDebugString())
```

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
# Charger les données
data = MLUtils.loadLibSVMFile(sc, 'data/mllib/sample_libsvm_data.txt')
# Découper les données ensemble d'apprentissage et de test (70%,30%)
(trainingData, testData) = data.randomSplit([0.7, 0.3])
# Appliquer les forêts aléatoires.
model = RandomForest.trainClassifier(trainingData, numClasses=2,
categoricalFeaturesInfo={}, numTrees=3)
# Evaluer le modèle
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() /
float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())
```

```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD,
LinearRegressionModel
#Préparer les données
def MapLine(line):
        val = [float(x) for x in line.split(';')]
        if(val[11]==5.0):
    return LabeledPoint(0, val[:10])
  else:
    return LabeledPoint(1, val[:10])
data = sc.textFile("/data/mllib/winequality-red.csv")
data=data.filter(lambda line: 'fixed acidity' not in line)
labelData = data.map(MapLine)
(trainingData, testData) = labelData.randomSplit([0.7, 0.3])
model = RandomForest.trainClassifier(trainingData, numClasses=2,
categoricalFeaturesInfo={}, numTrees=3, impurity='gini')
# Afficher le modèle
print(model.toDebugString())
# Evaluer le modèle
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() /
float(testData.count())
print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
```

```
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWithSGD,
LinearRegressionModel
# charger et préparer les données
def parsePoint(line):
    values = [float(x) for x in line.replace(',', ' ').split(' ')]
    return LabeledPoint(values[0], values[1:])
data = sc.textFile("data/mllib/ridge-data/lpsa.data")
parsedData = data.map(parsePoint)
# créer le modèle
model = LinearRegressionWithSGD.train(parsedData, iterations=100)
# évaluer le résultat
VP = parsedData.map(lambda p: (p.label, model.predict(p.features)))
MSE = VP.map(lambda (v, p): (v - p)**2) .reduce(lambda x, y: x + y) / data.count()
print("Mean Squared Error = " + str(MSE))
# Sauvegarder le modèle
model.save(sc, "target/tmp/pythonLinearRegressionWithSGDModel")
OurModel = LinearRegressionModel.load(sc,
"target/tmp/pythonLinearRegressionWithSGDModel")
```