Koding Dasar dan Tingkat Lanjut (PySpark dan Scala Spark)

September 24, 2018

(Ref. Big Data Community dan https://spark.apache.org/docs/latest/mllib-naive-bayes.html) MK Analisis Big Data Filkom UB (Imam Cholissodin | imamcs@ub.ac.id)

1 Latihan Dasar Scala

In [280]:
```
%%scala
//import java.text.SimpleDateFormat
//import java.util.{Calendar, Date}

// Membuat class dan fungsi
class Opt {
  // deklarasi method add secara umum
  def add(a: Int, b: Int) = a + b

  // nama method sama, tetapi dengan men-set tipe return-nya
  //def add(a: Int, b: Int): Int = a + b

  // men-define method body dalam suatu block dalam kurung kurawal
  /*def add(a: Int, b: Int): Int = {
    a + b
  }*/

  def tambahdanKali(a: Int, b: Int) = {
    val hasil1: Int = a + b; // val --> Immutable atau Read only
    val hasil2: Int = a * b; // val --> Immutable atau Read only
    var Keterangan: String=""; // var --> Mutable atau read-write
    if(hasil1 == hasil2){
      Keterangan = "hasil1 = hasil2";
    } else{
      Keterangan = "hasil1 != hasil2";
    }
    (Keterangan, hasil1, hasil2) // return langsung 3 nilai
  }
}

object ScalaDasar {
```

```
    def main(args: Array[String]) {
        val c = new Opt()
        print("Hasil 1+8 =")
        print(c.add(1,8))
        print("\n")
        print("Hasil c.tambahdanKali(2,8)._1 = ")
        print(c.tambahdanKali(2,8)._1)
        print("\n")
        print("Hasil c.tambahdanKali(2,8)._2 = ")
        print(c.tambahdanKali(2,8)._2)
        print("\n")
        print("Hasil c.tambahdanKali(2,8)._3 = ")
        print(c.tambahdanKali(2,8)._3)


    }
}

ScalaDasar.main(Array())
```

```
Hasil 1+8 =9
Hasil c.tambahdanKali(2,8)._1 = hasil1 != hasil2
Hasil c.tambahdanKali(2,8)._2 = 10
Hasil c.tambahdanKali(2,8)._3 = 16
```

```
Out[280]: defined class Opt

          defined object ScalaDasar
```

2 Koding Map Spark

```
In [329]: %%python
          def add1(x):
              return x+5

          raw_data = sc.parallelize([1,2,3])
          rdd = raw_data.map(lambda x: add1(x))
          #print(rdd.take(3))
          print(rdd.collect())
```

```
[6, 7, 8]
```

```
In [136]: %%scala
          def add1(x: Double) = x+5

          //val rdd = sc.parallelize(1 to 3).map(i => add1(i)).count()
```

```
val rdd = sc.parallelize(List(1, 2, 3, 7)).map(i => add1(i))
//val rdd = sc.parallelize(Vector(1, 2, 3)).map(i => add1(i))
print(rdd.count()+"\n")
print(rdd.sum()+"\n")
print(rdd.mean())
rdd.collect()
```

```
4
33.0
8.25
```

```
Out[136]: add1: (x: Double)Double

          rdd: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[98] at map at <console>:35

          res62: Array[Double] = Array(6.0, 7.0, 8.0, 12.0)
```

3 Koding Filter Spark

```
In [147]: %%python
          def isGanjil(x):
              return x%2==1

          raw_data = sc.parallelize(range(1,16))
          rdd = raw_data.filter(lambda x: isGanjil(x))
          #print(rdd.take(3))
          print(rdd.collect())
```

```
[1, 3, 5, 7, 9, 11, 13, 15]
```

```
In [146]: %%scala
          def isGanjil(x: Double) = x%2==1

          val rdd = sc.parallelize(1 to 15).filter(i => isGanjil(i))
          //val rdd = sc.parallelize(List(1, 2, 3, 7)).filter(i => isGanjil(i))
          //val rdd = sc.parallelize(Vector(1, 2, 3)).filter(i => isGanjil(i))
          print(rdd.count()+"\n")
          print(rdd.sum()+"\n")
          print(rdd.mean())
          rdd.collect()
```

```
8
64.0
8.0
```

```
Out[146]: isGanjil: (x: Double)Boolean
```

```
rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[114] at filter at <console>:34

res65: Array[Int] = Array(1, 3, 5, 7, 9, 11, 13, 15)
```

4 Koding Reduce Spark

```
In [173]: %%python
          def jumlahkan(x,y):
              return x+y

          raw_data = sc.parallelize(range(1,6))
          #rdd = raw_data.reduce(lambda x,y:x+y)
          rdd = raw_data.reduce(lambda x,y:jumlahkan(x,y))
          print(rdd)
```

```
15
```

```
In [184]: %%scala
          def jumlahkan(x: Int, y: Int) = x+y

          val rdd = sc.parallelize(1 to 5).reduce((i,j) => jumlahkan(i,j))
          //val rdd = sc.parallelize(List(1, 2, 3, 7)).reduce((i,j) => jumlahkan(i,j))
          //val rdd = sc.parallelize(Vector(1, 2, 3)).reduce((i,j) => jumlahkan(i,j))
          print(rdd)
```

```
15

Out[184]: jumlahkan: (x: Int, y: Int)Int

          rdd: Int = 15
```

5 Koding Pure Lambda Spark (tanpa memberikan nama fungsi)

```
In [187]: %%python
          raw_data = sc.parallelize(range(1,6))
          rdd = raw_data.reduce(lambda x,y:x+y)
          print(rdd)
```

```
15
```

```
In [203]: %%python
          (lambda x:2*x)(3)
```

```
Out[203]: 6
```

```
In [193]: %%python
          raw_data = sc.parallelize([1,2,3])
          rdd=raw_data.map(lambda x:2*x)
          print(rdd.collect())
```

```
[2, 4, 6]
```

```
In [195]: %%python
          raw_data = sc.parallelize([(1,2),(3,4),(5,6)])
          rdd=raw_data.map(lambda x:x[0])
          print(rdd.collect())
```

```
[1, 3, 5]
```

```
In [196]: %%python
          raw_data = sc.parallelize([1,2,3])
          rdd = raw_data.reduce(lambda x,y:x+y)
          print(rdd)
```

```
6
```

```
In [197]: %%python
          raw_data = sc.parallelize([(1,2),(3,4),(5,6)])
          rdd=raw_data.map(lambda x:x[0]).reduce(lambda x,y:x+y)
          print(rdd)
```

```
9
```

```
In [186]: %%scala
          val rdd = sc.parallelize(1 to 5).reduce((i,j) => (i+j))
          //val rdd = sc.parallelize(List(1, 2, 3, 7)).reduce((i,j) => (i+j))
          //val rdd = sc.parallelize(Vector(1, 2, 3)).reduce((i,j) => (i+j))
          val rdd = (i: Int) => { i * 2 }

          print(rdd)
```

```
15
```

```
Out[186]: rdd: Int = 15
```

```
In [236]: %%scala
          val raw_data = sc.parallelize(List(3))
          val rdd=raw_data.map(x => 2*x)
          rdd.collect()
```

```
Out[236]: raw_data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[198] at parallelize a

          rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[199] at map at <console>:33

          res85: Array[Int] = Array(6)
```

```
In [239]: %%scala
          val raw_data = sc.parallelize(List(1,2,3))
          val rdd=raw_data.map(x => 2*x)
          rdd.collect()

Out[239]: raw_data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[200] at parallelize a

          rdd: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[201] at map at <console>:33

          res88: Array[Int] = Array(2, 4, 6)
```

```
In [306]: %%scala
          object MyScala {
              def main(args: Array[String]) {
                  //import sqlContext.implicits._
                  val df = Seq((1, Seq(1,2)),(2,Seq(3,4)),(3,Seq(5,6))).toDF("kolom 0","kolom 1")
                  df.show

                  // get data pertama pada kolom 1
                  val getVal = df.select($"kolom 1"(0)).as[Double].show
              }
          }

          MyScala.main(Array())
```

```
+-------+-------+
|kolom 0|kolom 1|
+-------+-------+
|      1| [1, 2]|
|      2| [3, 4]|
|      3| [5, 6]|
+-------+-------+

+----------+
|kolom 1[0]|
+----------+
|         1|
|         3|
|         5|
+----------+
```

```
Out[306]: defined object MyScala
```

```
In [315]: %%scala
          val raw_data = sc.parallelize(List(1,2,3))
          val rdd = raw_data.reduce((x,y) => x+y)
          print(rdd)
```

```
6
```

```
Out[315]: raw_data: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[237] at parallelize a

          rdd: Int = 6
```

```
In [322]: %%scala
          object MyScala {
              def main(args: Array[String]) {
                  //import sqlContext.implicits._
                  val df = Seq((1, Seq(1,2)),(2,Seq(3,4)),(3,Seq(5,6))).toDF("kolom 0","kolom 1
                  df.show

                  // get data pertama pada kolom 1
                  val getVal = df.select($"kolom 1"(0)).as[Double].groupBy().sum().show
              }
          }

          MyScala.main(Array())
```

```
+-------+-------+
|kolom 0|kolom 1|
+-------+-------+
|      1| [1, 2]|
|      2| [3, 4]|
|      3| [5, 6]|
+-------+-------+

+---------------+
|sum(kolom 1[0])|
+---------------+
|              9|
+---------------+
```

```
Out[322]: defined object MyScala
```

```
In [259]: %%scala
          val df = Seq(
            ("111", Seq(("111", 1.0), ("333", 0.5), ("666", 0.4))), ("333", Seq())
          ).toDF("item", "other_items")

          df.show

          df.select($"item", $"other_items"(0)("_1").alias("other_items"))
            .na.drop(Seq("other_items")).show
```

```
+----+--------------------+
|item|         other_items|
+----+--------------------+
| 111|[[111,1.0], [333,...|
| 333|                  []|
+----+--------------------+

+----+-----------+
|item|other_items|
+----+-----------+
| 111|        111|
+----+-----------+



Out[259]: df: org.apache.spark.sql.DataFrame = [item: string, other_items: array<struct<_1:stri
```

6 Contoh Load data dari File *.csv

```
In [42]: %%python
         #load data dari file csv
         import pandas as pd
         import numpy as np

         data = pd.read_csv("data/my/Dataset.csv");
         print(data)
```

```
  Fitur 1  Fitur 2  Fitur 3  Kelas
0  Urgent      Yes      Yes  Party
1  Urgent       No      Yes  Study
2    Near      Yes      Yes  Party
3    None      Yes       No  Party
4    None       No      Yes    Pub
5    None      Yes       No  Party
6    Near       No       No  Study
7    Near       No      Yes     TV
8    Near      Yes      Yes  Party
```

```
9  Urgent       No       No  Study
```

```
In [44]: %%python
         sc
```

```
Out[44]: <SparkContext master=local[*] appName=spylon-kernel>
```

7 Koding Naive Bayes di Spark

```
In [325]: %%scala
          import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
          import org.apache.spark.mllib.util.MLUtils

          // Load and parse the data file.
          val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")

          // Split data into training (60%) and test (40%).
          val Array(training, test) = data.randomSplit(Array(0.6, 0.4))

          val model = NaiveBayes.train(training, lambda = 1.0, modelType = "multinomial")

          val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
          val accuracy = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / test.cou

          // Save and load model
          model.save(sc, "target/tmp/myNaiveBayesModel")
          val sameModel = NaiveBayesModel.load(sc, "target/tmp/myNaiveBayesModel")
```

```
Out[325]: import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}

          import org.apache.spark.mllib.util.MLUtils

          data: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Mapl

          training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] =

          test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = Mapl

          model: org.apache.spark.mllib.classification.NaiveBayesModel = org.apache.spark.mllil

          predictionAndLabel: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[27

          accuracy: Double = 0.9814814814814...
```

```
In [ ]: %%python
        from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
        from pyspark.mllib.util import MLUtils
```

```
        import shutil

        # Load and parse the data file.
        data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample_libsvm_data.txt")

        # Split data approximately into training (60%) and test (40%)
        training, test = data.randomSplit([0.6, 0.4])

        # Train a naive Bayes model.
        model = NaiveBayes.train(training, 1.0)

        # Make prediction and test accuracy.
        predictionAndLabel = test.map(lambda p: (model.predict(p.features), p.label))
        accuracy = 1.0 * predictionAndLabel.filter(lambda pl: pl[0] == pl[1]).count() / test.c
        print('model accuracy {}'.format(accuracy))

        # Save and load model
        output_dir = 'target/tmp/myNaiveBayesModel'
        shutil.rmtree(output_dir, ignore_errors=True)
        model.save(sc, output_dir)
        sameModel = NaiveBayesModel.load(sc, output_dir)
        predictionAndLabel = test.map(lambda p: (sameModel.predict(p.features), p.label))
        accuracy = 1.0 * predictionAndLabel.filter(lambda pl: pl[0] == pl[1]).count() / test.c
        print('sameModel accuracy {}'.format(accuracy))
```

```
model accuracy 1.0
```