# [SRC] 05.1. Indicators, Correlations and Sampling

val tweets = spark.read.parquet("/dataset/twitter/reference")

tweets.printSchema

tweets.createOrReplaceTempView("tweets")

val friends = tweets.select("user.friendsCount").map(\_.getAs[Long]("friendsCount").toDouble)

println("Count: " + friends.rdd.count)

println("Min: " + friends.rdd.min)

println("Max: " + friends.rdd.max)

println("Sum: " + friends.rdd.sum)

println("Mean: " + friends.rdd.mean)

println("Variance: " + friends.rdd.variance)

println("Stdev: " + friends.rdd.stdev)

val stats = friends.rdd.stats

println(s"Count: : ${stats.count}")

println(s"Min: : ${stats.min}")

println(s"Max: : stats.max}")

println(s"Sum: : ${stats.sum}")

println(s"Mean: : ${stats.mean}")

println(s"Variance : ${stats.variance}")

println(s"Stdev : ${stats.stdev}")

tweets.select(mean("user.friendsCount"), min("user.friendsCount"), max("user.friendsCount")).show()

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.stat.Statistics

val vectors = friends.rdd.map(d => Vectors.dense(d))

val summary = Statistics.colStats(vectors)

println(s"Mean : ${summary.mean}")

println(s"Vairance : ${summary.variance}")

println(s"Non Zeros : ${summary.numNonzeros}")

println(s"NormL1 : ${summary.normL1}")

println(s"NormL2 : ${summary.normL2}")

println(s"Non Zeros : ${summary.numNonzeros}")

friends.rdd.meanApprox(timeout = 100, confidence = 0.9)

friends.rdd.sampleStdev

friends.rdd.sampleVariance

friends.rdd.histogram(5)

val h2 = friends.rdd.histogram(Array(0.0, 3000.0, 629089))

import org.apache.spark.mllib.stat.Statistics

val followers = tweets.select("user.followersCount").map(\_.getAs[Long]("followersCount").toDouble)

val correlation = Statistics.corr(friends.rdd, followers.rdd, "pearson")

tweets.count

val sample = tweets.rdd.sample(fraction = 0.1, withReplacement = true)

sample.count

val f1 = tweets.filter("user.friendsCount < 2000")

f1.count

val f2 = tweets.filter("user.friendsCount >= 2000")

f2.count

val d1 = f1.rdd.map(r => (1, r))

val d2 = f2.rdd.map(r => (2, r))

val d = sc.union(d1, d2)

val fractions = Map(1 -> 0.5, 2 -> 0.5)

val approxSample = d.sampleByKey(withReplacement = false, fractions = fractions)

val exactSample = d.sampleByKeyExact(withReplacement = false, fractions = fractions)

# [SRC] 05.2. Validate Statistical Relevance

val tweets = spark.read.parquet("/dataset/twitter/reference")

val lengths = tweets.select("text").map(\_.getAs[String]("text").length.toDouble)

lengths.createOrReplaceTempView("lengths")

%sql select value, count(\*) from lengths group by value

import org.apache.spark.mllib.linalg.\_

import org.apache.spark.mllib.regression.LabeledPoint

import org.apache.spark.mllib.stat.Statistics

import org.apache.spark.mllib.stat.test.ChiSqTestResult

val v = lengths.rdd.zipWithIndex().map(r => (r.\_2.toInt, r.\_1))

val vec = Vectors.sparse(v.count.toInt, v.collect)

println(Statistics.chiSqTest(vec))

import org.apache.spark.mllib.stat.Statistics

println(Statistics.kolmogorovSmirnovTest(lengths.rdd, "norm", 0, 1))

val cdf = Map(20.0 -> 0.1, 21.0 -> 0.6, 22.0 -> 0.05, 23.0 -> 0.05, 92 -> 0.0, 24.0 -> 0.1)

println(Statistics.kolmogorovSmirnovTest(lengths.filter($"value" >= 20 and $"value" <= 24 ).rdd, cdf))

import org.apache.spark.mllib.stat.KernelDensity

val kd = new KernelDensity()

.setSample(lengths.rdd)

.setBandwidth(3.0)

val densities = kd.estimate(Array(-1.0, 2.0, 5.0))

# [SRC] 05.3. Run SVD and PCA

import org.apache.spark.mllib.linalg.Vectors

val v = tweets

.withColumnRenamed("id", "tweet\_id")

.select(

"tweet\_id",

"user.id",

"user.followersCount",

"user.friendsCount"

)

.rdd.map({

u =>

Vectors.dense(

u.getAs[Long]("tweet\_id").toDouble,

u.getAs[Long]("id").toDouble,

u.getAs[Long]("followersCount").toDouble,

u.getAs[Long]("friendsCount").toDouble

)

})

v.take(10).map(println(\_))

import org.apache.spark.mllib.linalg.distributed.RowMatrix

val m = new RowMatrix(v)

println(s"Rows \* Columns: ${m.numRows} \* ${m.numCols}")

val svd = m.computeSVD(2)

println(s"Left Singular Vectors: ${svd.U}")

println(s"Singular Values: ${svd.s}")

println(s"Right Singular Vectors: ${svd.V}")

val reducedData = m.multiply(svd.V).rows

val reducedMatrix = new RowMatrix(reducedData)

println(s"${reducedMatrix.numRows} Rows \* ${reducedMatrix.numCols} Columns")

val pca = m.computePrincipalComponents(2)

println(pca)

val reducedData = m.multiply(pca).rows

val reducedMatrix = new RowMatrix(reducedData)

println(s"Rows \* Columns: ${reducedMatrix.numRows} \* ${reducedMatrix.numCols}")

# [SRC] 05.4. Extend the Basic Statistics to your Needs

tweets.select("favoriteCount").rdd.map(\_.getAs[Long]("favoriteCount")).sum

tweets.select("createdAt").rdd.map(\_.getAs[Long]("createdAt")).mean

case class SimpleTweet(text: String, createdAt: Long, favoriteCount: Long)

val rdd = tweets.rdd.map( u =>

SimpleTweet(

u.getAs[String]("text"),

u.getAs[Long]("createdAt"),

u.getAs[Long]("favoriteCount")

)

)

class Extensions(rdd: org.apache.spark.rdd.RDD[SimpleTweet]) {

def favoriteCounts = rdd.map(\_.favoriteCount).sum

}

object Extensions {

implicit def addFavoriteCounts(rdd: org.apache.spark.rdd.RDD[SimpleTweet]) = new Extensions(rdd)

}

import Extensions.\_

println(rdd.favoriteCounts)

case class SimpleTweet2(text: String, createdAt: Long, favoriteCount: Long, cleanText: String = "")

class CleanRDD(prev:org.apache.spark.rdd.RDD[SimpleTweet2], pattern: String) extends org.apache.spark.rdd.RDD[SimpleTweet2](prev) {

override def compute(split: org.apache.spark.Partition, context: org.apache.spark.TaskContext): Iterator[SimpleTweet2] = {

firstParent[SimpleTweet2].iterator(split, context).map(simpleTweet => {

val cleanText = simpleTweet.text.replaceAll(pattern, "")

new SimpleTweet2(simpleTweet.text, simpleTweet.createdAt, simpleTweet.favoriteCount, cleanText)

})}

override protected def getPartitions: Array[org.apache.spark.Partition] = firstParent[SimpleTweet2].partitions

}

class Extensions(rdd: org.apache.spark.rdd.RDD[SimpleTweet2]) {

def clean(pattern: String) = new CleanRDD(rdd, pattern)

}

object Extensions2 {

implicit def addClean(rdd: org.apache.spark.rdd.RDD[SimpleTweet2]) = new Extensions(rdd)

}

import Extensions2.\_

val rdd2 = tweets.rdd.map( u =>

SimpleTweet2(

u.getAs[String]("text"),

u.getAs[Long]("createdAt"),

u.getAs[Long]("favoriteCount")

)

)

rdd2.clean("XXX").take(10).map(r => println(r.cleanText))