Estatísticas básicas - API baseada no RDD

- Estatísticas resumidas
- Correlações
- Amostragem estratificada
- Testando hipóteses
    - Streaming Teste Significância
- Geração aleatória de dados
- Estimativa da densidade Kernel

Estatísticas resumidas

Nós fornecemos colunas estatísticas resumidas para RDD [Vector] através da função colStats disponível em Statistics.

ColStats() retorna uma instância de MultivariateStatisticalSummary, que contém o máximo, mínimo, média, variância e número de não-zeros na coluna, bem como a contagem total.

Scala

Consulte os documentos do Scala MultivariateStatisticalSummary para obter detalhes sobre a API.

Código-Fonte

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

import org.apache.spark.mllib.stat.{MultivariateStatisticalSummary, Statistics}

val observations = sc.parallelize(
  Seq(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(3.0, 30.0, 300.0)
  )
)

// Compute column summary statistics.
val summary: MultivariateStatisticalSummary = Statistics.colStats(observations)
println(summary.mean)  // a dense vector containing the mean value for each column
println(summary.variance)  // column-wise variance
println(summary.numNonzeros)  // number of nonzeros in each column
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / SummaryStatisticsExample.scala" no repositório Spark.

Java

Consulte os documentos do Java MultivariateStatisticalSummary para obter detalhes sobre a API.

Código-Fonte

```
import java.util.Arrays;


import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.stat.MultivariateStatisticalSummary;
import org.apache.spark.mllib.stat.Statistics;

JavaRDD<Vector> mat = jsc.parallelize(
  Arrays.asList(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(3.0, 30.0, 300.0)
  )
); // an RDD of Vectors

// Compute column summary statistics.
MultivariateStatisticalSummary summary = Statistics.colStats(mat.rdd());
System.out.println(summary.mean());  // a dense vector containing the mean value for each column
System.out.println(summary.variance());  // column-wise variance
System.out.println(summary.numNonzeros());  // number of nonzeros in each column
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaSummaryStatisticsExample.java" no repositório Spark.

Python

Consulte os documentos do Python MultivariateStatisticalSummary para obter mais detalhes sobre a API.

Código-Fonte

```
import numpy as np


from pyspark.mllib.stat import Statistics

mat = sc.parallelize(
    [np.array([1.0, 10.0, 100.0]), np.array([2.0, 20.0, 200.0]), np.array([3.0, 30.0, 300.0])]
)  # an RDD of Vectors
```

```
# Compute column summary statistics.
summary = Statistics.colStats(mat)
print(summary.mean())  # a dense vector containing the mean value for each column
print(summary.variance())  # column-wise variance
print(summary.numNonzeros())  # number of nonzeros in each column
```

Encontre o código de exemplo completo em "examples / src / main / python / mllib / summary_statistics_example.py" no repositório Spark.

Correlações

Calcular a correlação entre duas séries de dados é uma operação comum em Estatística. Em spark.mllib fornecemos a flexibilidade para calcular correlações emparelhadas entre muitas séries. Os métodos de correlação suportados são atualmente a correlação de Pearson e Spearman.

Scala

Statistics fornece métodos para calcular correlações entre séries. Dependendo do tipo de entrada, dois RDD [Double] s ou um RDD [Vector], a saída será um Double ou a matriz de correlação, respectivamente.

Consulte os documentos da Scala da Estatística para obter detalhes sobre a API.

Códifo-Fonte

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

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

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5))  // a series
// must have the same number of partitions and cardinality as seriesX
val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If a
// method is not specified, Pearson's method will be used by default.
val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")
println(s"Correlation is: $correlation")

val data: RDD[Vector] = sc.parallelize(
  Seq(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(5.0, 33.0, 366.0))
)  // note that each Vector is a row and not a column

// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method
// If a method is not specified, Pearson's method will be used by default.
val correlMatrix: Matrix = Statistics.corr(data, "pearson")
println(correlMatrix.toString)
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / CorrelationsExample.scala" no repositório Spark.

Java

Statistics fornece métodos para calcular correlações entre séries. Dependendo do tipo de entrada, dois JavaDoubleRDDs ou um JavaRDD <Vector>, a saída será um Double ou a matriz de correlação, respectivamente.

Consulte os documentos do Statistics Java para obter detalhes sobre a API.

Códifo-Fonte

```
import java.util.Arrays;


import org.apache.spark.api.java.JavaDoubleRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.linalg.Matrix;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.stat.Statistics;

JavaDoubleRDD seriesX = jsc.parallelizeDoubles(
  Arrays.asList(1.0, 2.0, 3.0, 3.0, 5.0));  // a series

// must have the same number of partitions and cardinality as seriesX
JavaDoubleRDD seriesY = jsc.parallelizeDoubles(
  Arrays.asList(11.0, 22.0, 33.0, 33.0, 555.0));

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method.
// If a method is not specified, Pearson's method will be used by default.
Double correlation = Statistics.corr(seriesX.srdd(), seriesY.srdd(), "pearson");
System.out.println("Correlation is: " + correlation);

// note that each Vector is a row and not a column
JavaRDD<Vector> data = jsc.parallelize(
  Arrays.asList(
    Vectors.dense(1.0, 10.0, 100.0),
    Vectors.dense(2.0, 20.0, 200.0),
    Vectors.dense(5.0, 33.0, 366.0)
  )
);

// calculate the correlation matrix using Pearson's method.
// Use "spearman" for Spearman's method.
// If a method is not specified, Pearson's method will be used by default.
Matrix correlMatrix = Statistics.corr(data.rdd(), "pearson");
System.out.println(correlMatrix.toString());
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaCorrelationsExample.java" no repositório Spark.

Python

Statistics fornece métodos para calcular correlações entre séries. Dependendo do tipo de entrada, dois RDD [Double] s ou um RDD [Vector], a saída será um Double ou a matriz de correlação, respectivamente.

Consulte os documentos do Statistics Python para obter mais detalhes sobre a API.

Códifo-Fonte

```
from pyspark.mllib.stat import Statistics


seriesX = sc.parallelize([1.0, 2.0, 3.0, 3.0, 5.0])  # a series
# seriesY must have the same number of partitions and cardinality as seriesX
seriesY = sc.parallelize([11.0, 22.0, 33.0, 33.0, 555.0])

# Compute the correlation using Pearson's method. Enter "spearman" for Spearman's method.
# If a method is not specified, Pearson's method will be used by default.
print("Correlation is: " + str(Statistics.corr(seriesX, seriesY, method="pearson")))

data = sc.parallelize(
    [np.array([1.0, 10.0, 100.0]), np.array([2.0, 20.0, 200.0]), np.array([5.0, 33.0, 366.0])]
)  # an RDD of Vectors

# calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's
method.
# If a method is not specified, Pearson's method will be used by default.
print(Statistics.corr(data, method="pearson"))
```

Encontre o código de exemplo completo em "examples / src / main / python / mllib / correlations_example.py" no repositório Spark.

Amostragem estratificada

Ao contrário das outras funções de estatísticas, que residem em spark.mllib, métodos de amostragem estratificados, sampleByKey e sampleByKeyExact, podem ser executados em RDD's de pares de valores-chave. Para a amostragem estratificada, as chaves podem ser consideradas como um rótulo e o valor como um atributo específico. Por exemplo, a chave pode ser homem ou mulher, ou documento ids, e os respectivos valores podem ser a lista de idades das pessoas na população ou a lista de palavras nos documentos. O método sampleByKey irá inverter uma moeda para decidir se uma observação será amostrada ou não, portanto, requer uma passagem sobre os dados e fornece um tamanho de amostra esperado. SampleByKeyExact requer recursos significativos mais do que a amostragem aleatória simples por stratum usada em sampleByKey, mas fornecerá o tamanho de amostragem exato com 99,99% de confiança. SampleByKeyExact atualmente não é suportado em python.

Scala

SampleByKeyExact () permite aos usuários amostrar exatamente itens ⌈fk·nk⌉∀k∈K, onde fk é a fração desejada para a chave k, nk é o número de pares de valores-chave para a chave k e K é o conjunto de chaves. A amostragem sem substituição requer uma passagem adicional sobre o RDD para garantir o tamanho da amostra, enquanto a amostragem com substituição requer duas passagens adicionais.

Código-Fonte

// an RDD[(K, V)] of any key value pairs

```
val data = sc.parallelize(
  Seq((1, 'a'), (1, 'b'), (2, 'c'), (2, 'd'), (2, 'e'), (3, 'f')))

// specify the exact fraction desired from each key
val fractions = Map(1 -> 0.1, 2 -> 0.6, 3 -> 0.3)

// Get an approximate sample from each stratum
val approxSample = data.sampleByKey(withReplacement = false, fractions =
fractions)
// Get an exact sample from each stratum
val exactSample = data.sampleByKeyExact(withReplacement = false, fractions =
fractions)

Encontre o código de exemplo completo em "examples / src / main / scala / org /
apache / spark / examples / mllib / StratifiedSamplingExample.scala" no
repositório Spark.
```

Java

SampleByKeyExact () permite aos usuários amostrar exatamente itens ⌈fk·nk⌉∀k∈K, onde fk é a fração desejada para a chave k, nk é o número de pares de valores-chave para a chave k e K é o conjunto de chaves. A amostragem sem substituição requer uma passagem adicional sobre o RDD para garantir o tamanho da amostra, enquanto a amostragem com substituição requer duas passagens adicionais.

Código-Fonte

import java.util.*;

```
import scala.Tuple2;

import org.apache.spark.api.java.JavaPairRDD;

List<Tuple2<Integer, Character>> list = Arrays.asList(
    new Tuple2<>(1, 'a'),
    new Tuple2<>(1, 'b'),
    new Tuple2<>(2, 'c'),
    new Tuple2<>(2, 'd'),
    new Tuple2<>(2, 'e'),
    new Tuple2<>(3, 'f')
);

JavaPairRDD<Integer, Character> data = jsc.parallelizePairs(list);
```

```
// specify the exact fraction desired from each key Map<K, Double>
ImmutableMap<Integer, Double> fractions = ImmutableMap.of(1, 0.1, 2, 0.6, 3,
0.3);

// Get an approximate sample from each stratum
JavaPairRDD<Integer, Character> approxSample = data.sampleByKey(false,
fractions);
// Get an exact sample from each stratum
JavaPairRDD<Integer, Character> exactSample = data.sampleByKeyExact(false,
fractions);
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaStratifiedSamplingExample.java" no repositório Spark.

Python

SampleByKey () permite aos usuários amostrar aproximadamente itens ⌈fk·nk⌉∀k∈K, onde fk é a fração desejada para a chave k, nk é o número de pares de valores-chave para a chave k e K é o conjunto de chaves.

Observação: sampleByKeyExact () atualmente não é suportado no Python.

Código-Fonte

an RDD of any key value pairs

```
data = sc.parallelize([(1, 'a'), (1, 'b'), (2, 'c'), (2, 'd'), (2, 'e'), (3,
'f')])

# specify the exact fraction desired from each key as a dictionary
fractions = {1: 0.1, 2: 0.6, 3: 0.3}

approxSample = data.sampleByKey(False, fractions)
```

Encontre o código de exemplo completo em "examples / src / main / python / mllib / stratified_sampling_example.py" no repositório Spark.

Testando hipóteses

O teste de hipóteses é uma ferramenta poderosa na estatística para determinar se um resultado é estatisticamente significativo, se esse resultado ocorreu por acaso ou não. Spark.mllib atualmente suporta o chi-quadrado de Pearson (χ2) testes de ajuste e independência. Os tipos de dados de entrada determinam se o ajuste ou o teste de independência são realizados. O teste de ajuste requer um tipo de entrada de Vetor, enquanto o teste de independência requer uma Matriz como entrada.

Spark.mllib também suporta o tipo de entrada RDD [LabeledPoint] para habilitar a seleção de recursos através de testes de independência chi-squared.

Scala

Estatísticas fornece métodos para executar testes de Pearson chi-quadrado. O exemplo a seguir demonstra como executar e interpretar testes de hipóteses.

Código-Fonte

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
import org.apache.spark.rdd.RDD

// a vector composed of the frequencies of events
val vec: Vector = Vectors.dense(0.1, 0.15, 0.2, 0.3, 0.25)

// compute the goodness of fit. If a second vector to test against is not
supplied
// as a parameter, the test runs against a uniform distribution.
val goodnessOfFitTestResult = Statistics.chiSqTest(vec)
// summary of the test including the p-value, degrees of freedom, test
statistic, the method
// used, and the null hypothesis.
println(s"$goodnessOfFitTestResult\n")

// a contingency matrix. Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0,
6.0))
val mat: Matrix = Matrices.dense(3, 2, Array(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))

// conduct Pearson's independence test on the input contingency matrix
val independenceTestResult = Statistics.chiSqTest(mat)
// summary of the test including the p-value, degrees of freedom
println(s"$independenceTestResult\n")

val obs: RDD[LabeledPoint] =
  sc.parallelize(
    Seq(
      LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0)),
      LabeledPoint(1.0, Vectors.dense(1.0, 2.0, 0.0)),
      LabeledPoint(-1.0, Vectors.dense(-1.0, 0.0, -0.5)
      )
    )
  ) // (feature, label) pairs.

// The contingency table is constructed from the raw (feature, label) pairs and
used to conduct
// the independence test. Returns an array containing the ChiSquaredTestResult
for every feature
// against the label.
val featureTestResults: Array[ChiSqTestResult] = Statistics.chiSqTest(obs)
featureTestResults.zipWithIndex.foreach { case (k, v) =>
  println("Column " + (v + 1).toString + ":")
  println(k)
}  // summary of the test
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / HypothesisTestingExample.scala" no repositório Spark.

Java

Estatísticas fornece métodos para executar testes de Pearson chi-quadrado. O exemplo a seguir demonstra como executar e interpretar testes de hipóteses.

```
Consulte os documentos Java do ChiSqTestResult para obter detalhes sobre a API.
```

Código-Fonte

import java.util.Arrays;

```
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.linalg.Matrices;
import org.apache.spark.mllib.linalg.Matrix;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.linalg.Vectors;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.stat.Statistics;
import org.apache.spark.mllib.stat.test.ChiSqTestResult;

// a vector composed of the frequencies of events
Vector vec = Vectors.dense(0.1, 0.15, 0.2, 0.3, 0.25);

// compute the goodness of fit. If a second vector to test against is not
supplied
// as a parameter, the test runs against a uniform distribution.
ChiSqTestResult goodnessOfFitTestResult = Statistics.chiSqTest(vec);
// summary of the test including the p-value, degrees of freedom, test
statistic,
// the method used, and the null hypothesis.
System.out.println(goodnessOfFitTestResult + "\n");

// Create a contingency matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
Matrix mat = Matrices.dense(3, 2, new double[]{1.0, 3.0, 5.0, 2.0, 4.0, 6.0});

// conduct Pearson's independence test on the input contingency matrix
ChiSqTestResult independenceTestResult = Statistics.chiSqTest(mat);
// summary of the test including the p-value, degrees of freedom...
System.out.println(independenceTestResult + "\n");

// an RDD of labeled points
JavaRDD<LabeledPoint> obs = jsc.parallelize(
  Arrays.asList(
    new LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0)),
    new LabeledPoint(1.0, Vectors.dense(1.0, 2.0, 0.0)),
    new LabeledPoint(-1.0, Vectors.dense(-1.0, 0.0, -0.5))
  )
);

// The contingency table is constructed from the raw (feature, label) pairs and
used to conduct
// the independence test. Returns an array containing the ChiSquaredTestResult
for every feature
// against the label.
ChiSqTestResult[] featureTestResults = Statistics.chiSqTest(obs.rdd());
int i = 1;
for (ChiSqTestResult result : featureTestResults) {
  System.out.println("Column " + i + ":");
  System.out.println(result + "\n");  // summary of the test
  i++;
```

```
}
```

Python

Estatísticas fornece métodos para executar testes de Pearson chi-quadrado. O exemplo a seguir demonstra como executar e interpretar testes de hipóteses.

```
Consulte os documentos do Statistics Python para obter mais detalhes sobre a
API.
```

Código-Fonte

from pyspark.mllib.linalg import Matrices, Vectors

```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.stat import Statistics

vec = Vectors.dense(0.1, 0.15, 0.2, 0.3, 0.25)  # a vector composed of the
frequencies of events

# compute the goodness of fit. If a second vector to test against
# is not supplied as a parameter, the test runs against a uniform distribution.
goodnessOfFitTestResult = Statistics.chiSqTest(vec)

# summary of the test including the p-value, degrees of freedom,
# test statistic, the method used, and the null hypothesis.
print("%s\n" % goodnessOfFitTestResult)

mat = Matrices.dense(3, 2, [1.0, 3.0, 5.0, 2.0, 4.0, 6.0])  # a contingency
matrix

# conduct Pearson's independence test on the input contingency matrix
independenceTestResult = Statistics.chiSqTest(mat)

# summary of the test including the p-value, degrees of freedom,
# test statistic, the method used, and the null hypothesis.
print("%s\n" % independenceTestResult)

obs = sc.parallelize(
    [LabeledPoint(1.0, [1.0, 0.0, 3.0]),
     LabeledPoint(1.0, [1.0, 2.0, 0.0]),
     LabeledPoint(1.0, [-1.0, 0.0, -0.5])]
)  # LabeledPoint(feature, label)

# The contingency table is constructed from an RDD of LabeledPoint and used to
conduct
# the independence test. Returns an array containing the ChiSquaredTestResult
for every feature
# against the label.
featureTestResults = Statistics.chiSqTest(obs)

for i, result in enumerate(featureTestResults):
    print("Column %d:\n%s" % (i + 1, result))
```

Encontre o código de exemplo completo em "examples / src / main / python / mllib / hypothesis_testing_example.py" no repositório Spark.

Além disso, o spark.mllib fornece uma implementação de 1 amostra e 2 lados do teste de Kolmogorov-Smirnov (KS) para distribuições de igualdade de probabilidade. Ao fornecer o nome de uma distribuição teórica (atualmente suportada apenas para a distribuição normal) e seus parâmetros, ou uma função para calcular a distribuição cumulativa de acordo com uma determinada distribuição teórica, o usuário pode testar a hipótese nula de que sua amostra é extraída daquela distribuição. No caso de o usuário testar contra a distribuição normal (distName = "norm"), mas não fornecer parâmetros de distribuição, o teste inicializa a distribuição normal padrão e registra uma mensagem apropriada.

Scala

Statistics fornece métodos para executar um teste de Kolmogorov-Smirnov de 1 amostra e 2 faces. O exemplo a seguir demonstra como executar e interpretar os testes de hipóteses.

```
Consulte os documentos da Scala da Estatística para obter detalhes sobre a API.
```

Código-Fonte

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

```
import org.apache.spark.rdd.RDD

val data: RDD[Double] = sc.parallelize(Seq(0.1, 0.15, 0.2, 0.3, 0.25))  // an
RDD of sample data

// run a KS test for the sample versus a standard normal distribution
val testResult = Statistics.kolmogorovSmirnovTest(data, "norm", 0, 1)
// summary of the test including the p-value, test statistic, and null
hypothesis if our p-value
// indicates significance, we can reject the null hypothesis.
println(testResult)
println()

// perform a KS test using a cumulative distribution function of our making
val myCDF = Map(0.1 -> 0.2, 0.15 -> 0.6, 0.2 -> 0.05, 0.3 -> 0.05, 0.25 -> 0.1)
val testResult2 = Statistics.kolmogorovSmirnovTest(data, myCDF)
println(testResult2)
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / HypothesisTestingKolmogorovSmirnovTestExample.scala" no Spark repo.

Java

Statistics fornece métodos para executar um teste de Kolmogorov-Smirnov de 1 amostra e 2 faces. O exemplo a seguir demonstra como executar e interpretar os testes de hipóteses.

```
Consulte os documentos do Statistics Java para obter detalhes sobre a API.
```

Código-Fonte

import java.util.Arrays;

```
import org.apache.spark.api.java.JavaDoubleRDD;
import org.apache.spark.mllib.stat.Statistics;
import org.apache.spark.mllib.stat.test.KolmogorovSmirnovTestResult;

JavaDoubleRDD data = jsc.parallelizeDoubles(Arrays.asList(0.1, 0.15, 0.2, 0.3,
0.25));
KolmogorovSmirnovTestResult testResult =
  Statistics.kolmogorovSmirnovTest(data, "norm", 0.0, 1.0);
// summary of the test including the p-value, test statistic, and null
hypothesis
// if our p-value indicates significance, we can reject the null hypothesis
System.out.println(testResult);
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaHypothesisTestingKolmogorovSmirnovTestExample.java" no Spark repo.

Python

Statistics fornece métodos para executar um teste de Kolmogorov-Smirnov de 1 amostra e 2 faces. O exemplo a seguir demonstra como executar e interpretar os testes de hipóteses.

```
Consulte os documentos do Statistics Python para obter mais detalhes sobre a
API.
```

Código-Fonte

```
from pyspark.mllib.stat import Statistics


parallelData = sc.parallelize([0.1, 0.15, 0.2, 0.3, 0.25])

# run a KS test for the sample versus a standard normal distribution
testResult = Statistics.kolmogorovSmirnovTest(parallelData, "norm", 0, 1)
# summary of the test including the p-value, test statistic, and null hypothesis
# if our p-value indicates significance, we can reject the null hypothesis
# Note that the Scala functionality of calling Statistics.kolmogorovSmirnovTest
with
# a lambda to calculate the CDF is not made available in the Python API
print(testResult)

Encontre o código de exemplo completo em "examples / src / main / python / mllib
/ hypothesis_testing_kolmogorov_smirnov_test_example.py" no repositório Spark.
```

Streaming Teste Significância

Spark.mllib fornece implementações on-line de alguns testes para suportar casos de uso como testes A / B. Esses testes podem ser realizados em um DStream Spark Streaming [(Boolean, Double)] onde o primeiro elemento de cada tupla indica grupo de controle (false) ou grupo de tratamento (true) eo segundo elemento é o valor de uma observação.

O teste de significância de fluxo contínuo suporta os seguintes parâmetros:

- PeacePeriod - O número de pontos de dados iniciais do fluxo para ignorar, usado para atenuar efeitos de novidade.
- WindowSize - O número de lotes passados para executar o teste de hipóteses. A definição para 0 executará processamento cumulativo usando todos os lotes anteriores.

Scala

StreamingTest fornece testes de hipóteses de transmissão.

Código-Fonte

val data = ssc.textFileStream(dataDir).map(line => line.split(",") match {

```
  case Array(label, value) => BinarySample(label.toBoolean, value.toDouble)
})

val streamingTest = new StreamingTest()
  .setPeacePeriod(0)
  .setWindowSize(0)
  .setTestMethod("welch")

val out = streamingTest.registerStream(data)
out.print()
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / StreamingTestExample.scala" no repositório Spark.

Java

StreamingTest fornece testes de hipóteses de transmissão.

Cógigo-Fonte

import org.apache.spark.mllib.stat.test.BinarySample;

```
import org.apache.spark.mllib.stat.test.StreamingTest;
import org.apache.spark.mllib.stat.test.StreamingTestResult;

JavaDStream<BinarySample> data = ssc.textFileStream(dataDir).map(
  new Function<String, BinarySample>() {
    @Override
    public BinarySample call(String line) {
```

```
      String[] ts = line.split(",");
      boolean label = Boolean.parseBoolean(ts[0]);
      double value = Double.parseDouble(ts[1]);
      return new BinarySample(label, value);
    }
  });

StreamingTest streamingTest = new StreamingTest()
  .setPeacePeriod(0)
  .setWindowSize(0)
  .setTestMethod("welch");

JavaDStream<StreamingTestResult> out = streamingTest.registerStream(data);
out.print();
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaStreamingTestExample.java" no Spark repo.

Geração aleatória de dados

A geração aleatória de dados é útil para algoritmos randomizados, prototipagem e testes de desempenho. Spark.mllib suporta a geração de RDDs aleatórios com i.i.d. Valores extraídos de uma dada distribuição: uniforme, padrão normal ou Poisson.

Scala

RandomRDDs fornece métodos de fábrica para gerar RDD aleatórios duplos ou RDD vetoriais. O exemplo a seguir gera um RDD duplo aleatório, cujos valores segue a distribuição normal padrão N (0, 1), e depois mapeie-o para N (1, 4).

Consulte os documentos do Scala RandomRDDs para obter detalhes sobre a API.

Código-Fonte

```
import org.apache.spark.SparkContext

import org.apache.spark.mllib.random.RandomRDDs._

val sc: SparkContext = ...

// Generate a random double RDD that contains 1 million i.i.d. values drawn from
the
// standard normal distribution `N(0, 1)`, evenly distributed in 10 partitions.
val u = normalRDD(sc, 1000000L, 10)
// Apply a transform to get a random double RDD following `N(1, 4)`.
val v = u.map(x => 1.0 + 2.0 * x)
```

Java

RandomRDDs fornece métodos de fábrica para gerar RDD aleatórios duplos ou RDD vetoriais. O exemplo a seguir gera um RDD duplo aleatório, cujos valores segue a distribuição normal padrão N (0, 1), e depois mapeie-o para N (1, 4).

```
Consulte os documentos do Java RandomRDDs para obter detalhes sobre a API.
```

Código-Fonte

```
import org.apache.spark.SparkContext;

import org.apache.spark.api.JavaDoubleRDD;
import static org.apache.spark.mllib.random.RandomRDDs.*;

JavaSparkContext jsc = ...

// Generate a random double RDD that contains 1 million i.i.d. values drawn from
the
// standard normal distribution `N(0, 1)`, evenly distributed in 10 partitions.
JavaDoubleRDD u = normalJavaRDD(jsc, 1000000L, 10);
// Apply a transform to get a random double RDD following `N(1, 4)`.
JavaDoubleRDD v = u.map(
  new Function<Double, Double>() {
    public Double call(Double x) {
      return 1.0 + 2.0 * x;
    }
  });
```

Python

RandomRDDs fornece métodos de fábrica para gerar RDD aleatórios duplos ou RDD vetoriais. O exemplo a seguir gera um RDD duplo aleatório, cujos valores segue a distribuição normal padrão N (0, 1), e depois mapeie-o para N (1, 4).

```
Consulte os documentos do RandomRDDs Python para obter mais detalhes sobre a
API.
```

Código-Fonte

```
from pyspark.mllib.random import RandomRDDs


sc = ... # SparkContext

# Generate a random double RDD that contains 1 million i.i.d. values drawn from
the
# standard normal distribution `N(0, 1)`, evenly distributed in 10 partitions.
u = RandomRDDs.normalRDD(sc, 1000000L, 10)
# Apply a transform to get a random double RDD following `N(1, 4)`.
v = u.map(lambda x: 1.0 + 2.0 * x)
```

Estimativa da densidade Kernel

A estimativa da densidade do núcleo é uma técnica útil para visualizar as distribuições de probabilidade empírica sem requerer hipóteses sobre a distribuição particular de que as amostras observadas são retiradas. Ele calcula uma estimativa da função de densidade de probabilidade de variáveis aleatórias, avaliadas em um determinado conjunto de pontos. Ele alcança essa estimativa ao expressar o PDF da distribuição empírica em um ponto particular como a média de PDFs de distribuições normais centradas em cada uma das amostras.

Scala

KernelDensity fornece métodos para calcular as estimativas de densidade do kernel a partir de um RDD de amostras. O exemplo a seguir demonstra como fazer isso.

```
Consulte o KernelDensity Scala docs para obter detalhes sobre a API.
```

Código-Fonte

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

```
import org.apache.spark.rdd.RDD

// an RDD of sample data
val data: RDD[Double] = sc.parallelize(Seq(1, 1, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9,
9))

// Construct the density estimator with the sample data and a standard deviation
// for the Gaussian kernels
val kd = new KernelDensity()
  .setSample(data)
  .setBandwidth(3.0)

// Find density estimates for the given values
val densities = kd.estimate(Array(-1.0, 2.0, 5.0))
```

Encontre o código de exemplo completo em "examples / src / main / scala / org / apache / spark / examples / mllib / KernelDensityEstimationExample.scala" no repositório Spark.

Java

KernelDensity fornece métodos para calcular as estimativas de densidade do kernel a partir de um RDD de amostras. O exemplo a seguir demonstra como fazer isso.

```
Consulte os documentos Java do KernelDensity para obter detalhes sobre a API.
```

Código-Fonte

import java.util.Arrays;

```
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.mllib.stat.KernelDensity;

// an RDD of sample data
JavaRDD<Double> data = jsc.parallelize(
  Arrays.asList(1.0, 1.0, 1.0, 2.0, 3.0, 4.0, 5.0, 5.0, 6.0, 7.0, 8.0, 9.0,
9.0));

// Construct the density estimator with the sample data
// and a standard deviation for the Gaussian kernels
KernelDensity kd = new KernelDensity().setSample(data).setBandwidth(3.0);
```

```
// Find density estimates for the given values
double[] densities = kd.estimate(new double[]{-1.0, 2.0, 5.0});

System.out.println(Arrays.toString(densities));
```

Encontre o código de exemplo completo em "examples / src / main / java / org / apache / spark / examples / mllib / JavaKernelDensityEstimationExample.java" no repositório Spark.

Python

KernelDensity fornece métodos para calcular as estimativas de densidade do kernel a partir de um RDD de amostras. O exemplo a seguir demonstra como fazer isso.

```
Consulte o KernelDensity Python docs para obter mais detalhes sobre a API.
```

Código-Fonte

from pyspark.mllib.stat import KernelDensity

```
# an RDD of sample data
data = sc.parallelize([1.0, 1.0, 1.0, 2.0, 3.0, 4.0, 5.0, 5.0, 6.0, 7.0, 8.0,
9.0, 9.0])

# Construct the density estimator with the sample data and a standard deviation
for the Gaussian
# kernels
kd = KernelDensity()
kd.setSample(data)
kd.setBandwidth(3.0)

# Find density estimates for the given values
densities = kd.estimate([-1.0, 2.0, 5.0])
```

Encontre o código de exemplo completo em "examples / src / main / python / mllib / kernel_density_estimation_example.py" no repositório Spark.