APACHE SPARK MACHINE LEARNING

Mlib:main guide

* KORELASI

Menghitung korelasi antara dua rangkaian data adalah operasi umum di Statistik. Di dalam spark.ml kita berikan fleksibilitas untuk menghitung korelasi berpasangan di antara banyak seri. Metode korelasi yang didukung saat ini adalah korelasi Pearson dan Spearman.

Correlation menghitung matriks korelasi untuk input Dataset Vektor dengan menggunakan metode yang ditentukan. Outputnya adalah DataFrame yang berisi matriks korelasi dari kolom vektor.

**import**  **org.apache.spark.ml.linalg** { **Matrix** , **Vektor** }

**impor**  **org.apache.spark.ml.stat.Correlation**

**impor**  **org.apache.spark.sql.Row**

**val**  Data **=**  **Seq** (

**Vektor** . Jarang ( 4 , **Seq** (( 0 , 1,0 ), ( 3 , - 2,0 ))),

**Vektor** . padat ( 4.0 , 5.0 , 0.0 , 3.0 ),

**Vektor** . padat( 6.0 , 7.0 , 0.0 , 8.0 ),

**Vektor** . jarang ( 4 , **Seq** (( 0 , 9.0 ), ( 3 , 1.0 )))

)

**val**  df **=**  data . Peta ( **Tuple1** . berlaku ). toDF ( "fitur" )

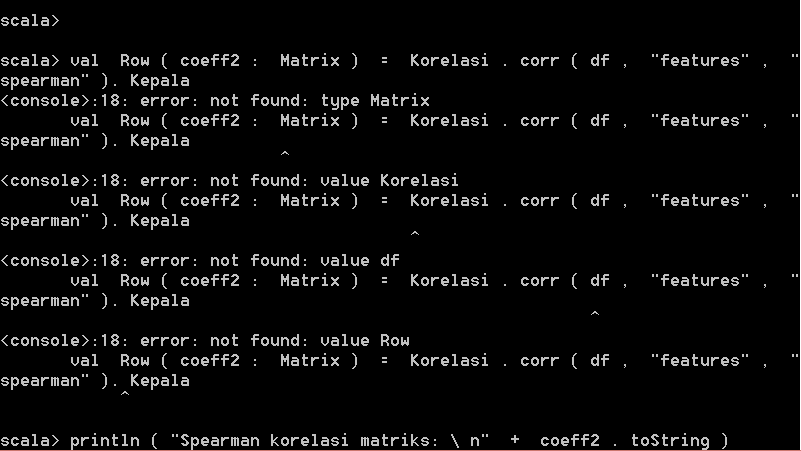
**val**  **Row** ( coeff1 **:**  Matrix ) **=**  **Korelasi** . koreksi ( df, "fitur" ). Kepala

println ( "Pearson korelasi matriks: \ n" + coeff1 . toString )

**val**  **Row** ( coeff2 **:**  Matrix ) **=**  **Korelasi** . corr ( df , "features" , "spearman" ). Kepala

println ( "Spearman korelasi matriks: \ n" + coeff2 . toString )

hasil:



* Uji hipotesis

Uji hipotesis adalah alat yang ampuh dalam statistik untuk mengetahui apakah suatu hasil secara statistik signifikan, apakah hasil ini kebetulan atau tidak. spark.mlSaat ini mendukung Chi-kuadrat Pearson ( χ2χ2

ChiSquareTestmelakukan uji independensi Pearson untuk setiap fitur terhadap label. Untuk setiap fitur, pasangan (fitur, label) diubah menjadi matriks kontingensi yang statistik Chi-kuadratnya dihitung. Semua label dan nilai fitur harus kategoris.

Lihat chisquaretestdocument untuk rincian tentang API.

**import**  **org.apache.spark.ml.linalg** { **Vector** , **Vectors** }

**import**  **org.apache.spark.ml.stat.ChiSquareTest**

**val**  data **=**  **Seq** (

( 0.0 , **Vectors** . Padat ( 0.5 , 10.0 )),

( 0.0 , **Vectors** . Padat ( 1.5 , 20.0 )),

( 1.0 , **Vektor** . padat ( 1,5 , 30,0 )),

( 0,0 , **Vektor** . padat ( 3.5 , 30.0 )),

( 0,0 , **Vectors** . padat ( 3.5 , 40.0 )),

( 1.0 , **Vectors** . padat ( 3.5 , 40.0 ))

)

**val**  df **=**  data . toDF ( "label" , "fitur" )

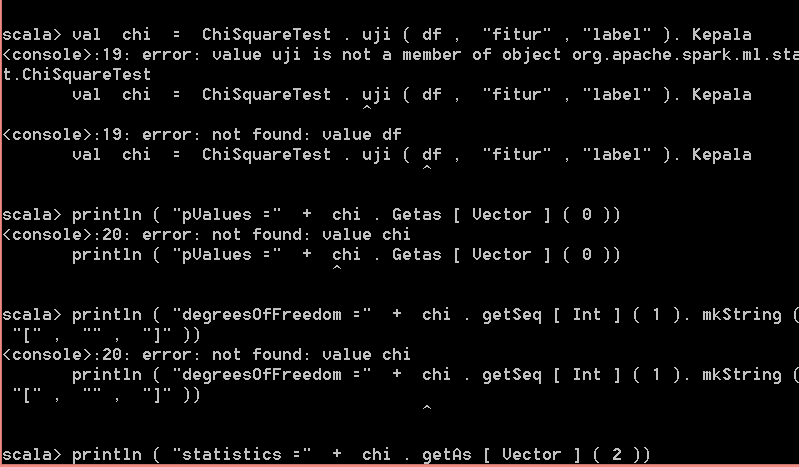
**val**  chi **=**  **ChiSquareTest** . uji ( df , "fitur" , "label" ). Kepala

println ( "pValues =" + chi . Getas [ Vector ] ( 0 ))

println ( "degreesOfFreedom =" + chi . getSeq [ Int ] ( 1 ). mkString ( "[" , "" , "]" ))

println ( "statistics =" + chi . getAs [ Vector ] ( 2 ))

hasil:



* Vector local

Vektor lokal memiliki indeks bilangan bulat dan kode berbasis 0 dan diklik ganda, disimpan pada satu mesin. MLlib mendukung dua jenis vektor lokal: padat dan jarang. Sebuah vektor padat didukung oleh array ganda yang mewakili nilai masuknya, sementara vektor jarang didukung oleh dua array sejajar: indeks dan nilai. Sebagai contoh, sebuah vektor (1.0, 0.0, 3.0)dapat digambarkan dalam format padat seperti [1.0, 0.0, 3.0]dalam format yang jarang seperti (3, [0, 2], [1.0, 3.0]), di mana 3ukuran vektornya.

Kelas dasar vektor lokal adalah [Vector](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.Vector), dan kami menyediakan dua implementasi: [DenseVector](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.DenseVector)dan [SparseVector](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.SparseVector). Sebaiknya gunakan metode pabrik yang diterapkan [Vectors](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.Vectors$)untuk membuat vektor lokal.

Lihat [Vectordokumentasi Scala](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.Vector) dan [Vectorsdokumen Scala](https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.Vectors$) untuk rincian tentang API.

**import** **org.apache.spark.mllib.linalg.**{**Vector**, **Vectors**}

*// Create a dense vector (1.0, 0.0, 3.0).*

**val** dv**:** Vector = **Vectors**.dense(1.0, 0.0, 3.0)

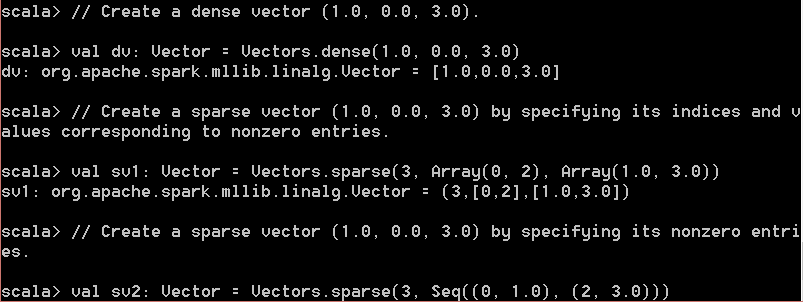
*// Create a sparse vector (1.0, 0.0, 3.0) by specifying its indices and values corresponding to nonzero entries.*

**val** sv1**:** Vector = **Vectors**.sparse(3, **Array**(0, 2), **Array**(1.0, 3.0))

*// Create a sparse vector (1.0, 0.0, 3.0) by specifying its nonzero entries.*

**val** sv2**:** Vector = **Vectors**.sparse(3, **Seq**((0, 1.0), (2, 3.0)))

hasil:



* Titik berlabel

Titik berlabel adalah vektor lokal, padat atau jarang, terkait dengan label / respons. Di MLlib, label poin digunakan dalam algoritma pembelajaran yang diawasi. Kami menggunakan dua kali untuk menyimpan label, jadi kami dapat menggunakan label poin dalam regresi dan klasifikasi. Untuk klasifikasi biner, label harus berupa 0(negatif) atau 1(positif). Untuk klasifikasi multiclass, label harus indeks kelas mulai dari nol: 0, 1, 2, ....

Titik berlabel ditunjukkan oleh kelas kasus labelpoint.

Lihat labelpoint dokumentasi untuk rincian tentang API.

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

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

*// Create a labeled point with a positive label and a dense feature vector.*

**val** pos **=** **LabeledPoint**(1.0, **Vectors**.dense(1.0, 0.0, 3.0))

*// Create a labeled point with a negative label and a sparse feature vector.*

**val** neg **=** **LabeledPoint**(0.0, **Vectors**.sparse(3, **Array**(0, 2), **Array**(1.0, 3.0)))

hasil:

