12

Spark MLlib

- 基于大数据的机器学习
- Mllib介绍
- 基本概念
- 流水线
- TF-IDF
- 特征转换
- 逻辑回归

基于大数据的机器学习

- 传统的机器学习算法，由于技术和单机存储的限制，只能在少量数据上使用，依赖于数据抽样
- 大数据技术的出现，支持在全量数据上使用机器学习算法进行数据分析
- 机器学习算法涉及大量**迭代计算**
- 基于磁盘的MapReduce不适合进行大量迭代计算
- 基于内存的Spark比较适合进行大量迭代计算

Spark 机器学习库MLlib

- Spark提供了一个基于海量数据的**机器学习库**，它提供了常用机器学习算法的分布式实现
- 开发者只需要有 Spark 基础并且了解机器学习算法的原理，以及方法相关参数的含义，就可以轻松的通过调用相应的 API 来实现基于海量数据的机器学习过程

- Pyspark的**即席查询**也是一个关键。算法工程师可以边写代码边运行，边看结果

Spark 机器学习库MLlib

- 需要注意的是，MLlib中只包含能够在集群上运行良好的并行算法
- 有些经典的机器学习算法没有包含在其中，就是因为它们不能并行执行
- 相反地，一些较新的研究得出的算法因为适用于集群，也被包含在MLlib中，例如分布式随机森林算法、最小交替二乘算法。这样的选择使得MLlib中的每一个算法都适用于大规模数据集
- 如果是小规模数据集上训练各机器学习模型，最好还是在各个节点上使用单节点的机器学习算法库

Spark 机器学习库MLlib

•MLlib是Spark的机器学习（Machine Learning）库，旨在简化机器学习的工程实践工作

•MLlib由一些通用的学习算法和工具组成，包括**分类、回归、聚类、协同过滤、降维**等，同时还包括底层的优化原语和高层的流水线（Pipeline）API，具体如下：

•**算法工具**：常用的学习算法，如分类、回归、聚类和协同过滤；

•**特征化工具**：特征提取、转化、降维和选择工具；

•**流水线(Pipeline)**：用于构建、评估和调整机器学习工作流的工具;

•**持久性**：保存和加载算法、模型和管道;

•**实用工具**：线性代数、统计、数据处理等工具。

Spark 机器学习库MLlib

Spark 机器学习库从1.2 版本以后被分为两个包：

- **spark.mllib** 包含基于RDD的原始算法API。Spark MLlib 历史比较长，在1.0 以前的版本即已经包含了，提供的算法实现都是基于原始的 RDD

- **spark.ml** 则提供了基于DataFrame 高层次的API，可以用来构建机器学习工作流（PipeLine）。ML Pipeline 弥补了原始 MLlib 库的不足，向用户提供了一个基于 DataFrame 的机器学习工作流式 API 套件

Spark 机器学习库MLlib

MLlib目前支持4种常见的机器学习问题: 分类、回归、聚类和协同过滤

	离散数据	连续数据
监督学习	Classification、LogisticRegression(with Elastic-Net)、SVM、DecisionTree、RandomForest、GBT、NaiveBayes、MultilayerPerceptron、OneVsRest	Regression、LinearRegression(with Elastic-Net)、DecisionTree、RandomFores、GBT、AFTSurvivalRegression、IsotonicRegression
无监督学习	Clustering、KMeans、GaussianMixture、LDA、PowerIterationClustering、BisectingKMeans	Dimensionality Reduction, matrix factorization、PCA、SVD、ALS、WLS

分类模型

首先以逻辑回归为例，介绍如何训练和测试模型

任务描述

查找出所有包含"spark"的句子，即将包含"spark"的句子的标签设为1，没有"spark"的句子的标签设为0。

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local").appName("Word Count").getOrCreate()
```

分类模型

生成数据

```
# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])



test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])
```

分词

```
wordData = tokenizer.transform(training)
wordData.show()
```

```
+---+---------------+-----+---------------------+
|id |text           |label|words                |
+---+---------------+-----+---------------------+
|0  |a b c d e spark|1.0  |[a, b, c, d, e, spark]|
|1  |b d            |0.0  |[b, d]               |
|2  |spark f g h    |1.0  |[spark, f, g, h]     |
|3  |hadoop mapreduce|0.0 |[hadoop, mapreduce]  |
+---+---------------+-----+---------------------+
```

向量化

```
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
featurizedData = hashingTF.transform(wordData)
featurizedData.select("id","features").show(truncate=False)
```

```
+---+-----------------------------------------------------------------------+
|id |features                                                               |
+---+-----------------------------------------------------------------------+
|0  |(262144,[17222,27526,28698,30913,227410,234657],[1.0,1.0,1.0,1.0,1.0,1.0])|
|1  |(262144,[27526,30913],[1.0,1.0])                                       |
|2  |(262144,[15554,24152,51505,234657],[1.0,1.0,1.0,1.0])                  |
|3  |(262144,[42633,155117],[1.0,1.0])                                      |
+---+-----------------------------------------------------------------------+
```

模型训练与测试

```
lr = LogisticRegression(maxIter=10, regParam=0.001)
lr_model = lr.fit(featurizedData)

wordData_test = tokenizer.transform(test)
featurizedData_test = hashingTF.transform(wordData_test)
prediction = lr_model.transform(featurizedData_test)
prediction.select("id", "text", "probability", "prediction").show(truncate=False)
```

```
+---+------------------+----------------------------------------+----------+
|id |text              |probability                             |prediction|
+---+------------------+----------------------------------------+----------+
|4  |spark i j k       |[0.1596407738787475,0.8403592261212525] |1.0       |
|5  |l m n             |[0.8378325685476744,0.16216743145232562]|0.0       |
|6  |spark hadoop spark|[0.06926633132976037,0.9307336686702395]|1.0       |
|7  |apache hadoop     |[0.9821575333444218,0.01784246665557808]|0.0       |
+---+------------------+----------------------------------------+----------+
```

基本概念

DataFrame：使用Spark SQL中的DataFrame作为数据集，它可以容纳各种数据类型。较之RDD，DataFrame包含了schema 信息，更类似传统数据库中的二维表格。在ML Pipeline中用来存储源数据。例如，DataFrame中的列可以是存储的文本、特征向量、真实标签和预测的标签等

基本概念

Transformer：转换器，是一种可以将一个DataFrame转换为另一个DataFrame的算法。比如一个模型就是一个 Transformer。它可以把一个不包含预测标签的测试数据集 DataFrame 打上标签，转化成另一个包含预测标签的 DataFrame。Transformer实现了一个方法transform()，它通过附加一个或多个列将一个DataFrame转换为另一个DataFrame

基本概念

Estimator：估计器或评估器，它是学习算法或在训练数据上的训练方法的概念抽象。在 Pipeline 里通常是被用来操作 DataFrame 数据并生成一个 Transformer。Estimator实现了一个fit()方法，接收一个DataFrame并产生一个转换器。比如，一个随机森林算法就是一个 Estimator，它可以调用fit()，通过训练特征数据而得到一个随机森林模型。

基本概念

Parameter：Parameter 被用来设置 Transformer 或者 Estimator 的参数。现在，所有转换器和估计器可共享用于指定参数的公共API

PipeLine：流水线或者管道。流水线将多个工作流阶段（转换器和估计器）连接在一起，形成机器学习的工作流，并获得结果输出

流水线

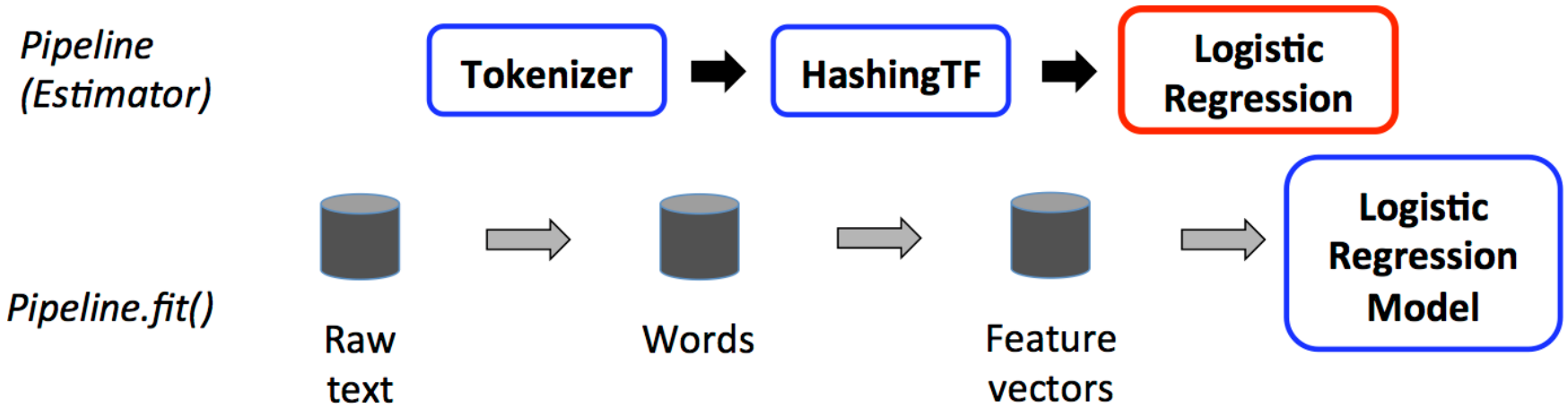
要构建一个 Pipeline流水线，首先需要定义 Pipeline 中的各个**流水线阶段** PipelineStage（包括转换器和评估器），比如指标提取和转换模型训练等。有了这些处理特定问题的转换器和评估器，就可以按照具体的处理逻辑有序地组织PipelineStages并创建一个Pipeline

```
pipeline = Pipeline(stages=[stage1,stage2,stage3])
```

然后就可以把训练数据集作为输入参数，调用 Pipeline 实例的 fit() 方法开始以流的方式处理源训练数据。这个调用会返回一个 PipelineModel 类实例，进而被用来预测测试数据的标签

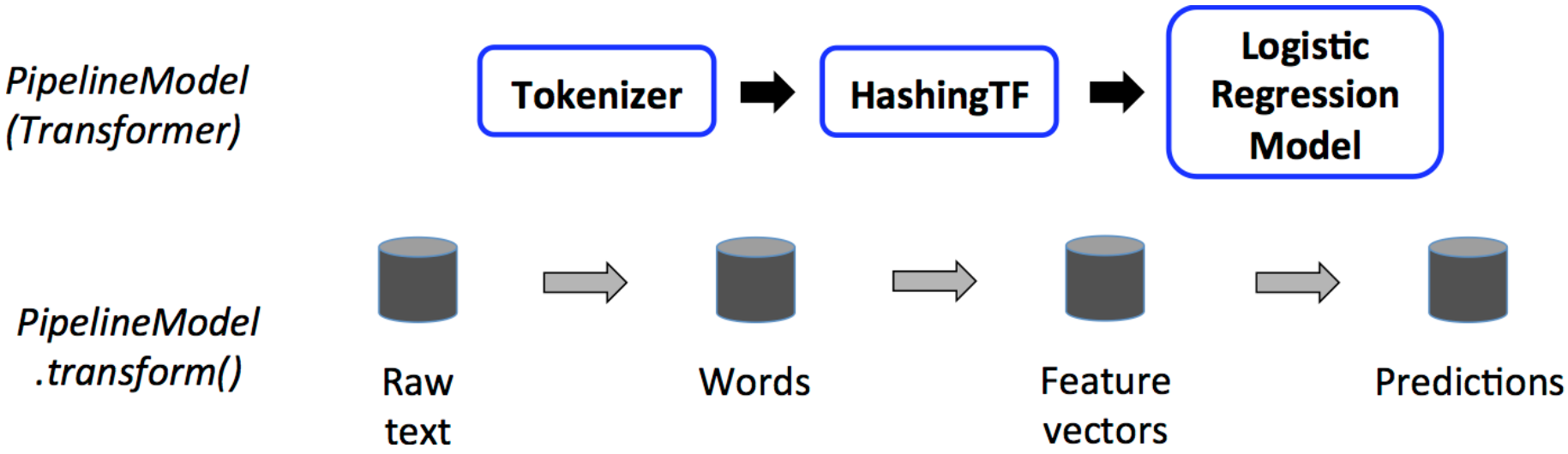
流水线工作过程

流水线的各个阶段按顺序运行，输入的DataFrame在通过每个阶段时被**转换**



流水线工作过程

值得注意的是，流水线本身也可以看做是一个估计器。在流水线的**fit()**方法运行之后，它产生一个PipelineModel，它是一个Transformer。 将在测试数据的时候使用管道模型。



构建一个机器学习流水线

下面以前面的逻辑回归模型为例，介绍流水线是如何使用

- 构建流水线需要使用**SparkSession对象**
- Spark2.0以上版本的pyspark在启动时会自动创建一个名为spark的SparkSession对象，当需要手工创建时，SparkSession可以由其伴生对象的builder()方法创建出来

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local").appName("Word Count").getOrCreate()
```

pyspark.ml依赖numpy包，Ubuntu 自带python3是没有numpy的，执行如下命令安装：

```
sudo pip3 install numpy
```

构建一个机器学习流水线

（1）引入要包含的包并构建训练和测试数据集

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])

test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])
```

构建一个机器学习流水线

（2）定义 Pipeline 中的各个流水线阶段PipelineStage，包括转换器和评估器，具体地，包含tokenizer, hashingTF和lr。

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
```

构建一个机器学习流水线

（3）按照具体的处理逻辑有序地组织PipelineStages，并创建一个Pipeline。

```
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

现在构建的Pipeline本质上是一个Estimator，在调用fit()方法之后，将产生一个PipelineModel，它是一个Transformer。

```
model = pipeline.fit(training)
```

可以看到，model的类型是一个PipelineModel，这个流水线模型将在测试数据的时候使用

构建一个机器学习流水线

（4）调用之前训练好的PipelineModel的transform()方法，让测试数据按顺序通过流水线，生成预测结果

```
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
for row in selected.collect():
    rid, text, prob, prediction = row
    print("(%d, %s) --> prob=%s, prediction=%f" % (rid, text, str(prob), prediction))
```

```
(4, spark i j k) --> prob=[0.1596407738787475,0.8403592261212525], prediction=1.000000
(5, l m n) --> prob=[0.8378325685476744,0.16216743145232562], prediction=0.000000
(6, spark hadoop spark) --> prob=[0.06926633132976037,0.9307336686702395], prediction=1.000
000
(7, apache hadoop) --> prob=[0.9821575333444218,0.01784246665557808], prediction=0.000000
```

TF-IDF

词频 - 逆文档频率（TF-IDF）是一种在文本挖掘中广泛使用的特征向量化方法，它可以体现一个词语在文档中的重要程度，从而构建基于语料库的文档的向量表达。

- TF(Term Frequency)：词频指的是某一个给定的词语在一篇文档中出现的次数。
- IDF(Inverse Document Frequency)：逆文档频率是一个词语普遍重要性的度量。某一特定词语的IDF，由总文档数目除以包含该词的文件的数目，再将得到的商取对数得到。

$$IDF = \log\left(\frac{\text{语料库文档总数} + 1}{\text{出现该词的文档数} + 1}\right)$$

$$TF - IDF = TF \cdot IDF$$

特征提取：TF-IDF

在Spark ML库中，TF-IDF被分成两部分：
- TF (+hashing)
- IDF

- **TF**: HashingTF 是一个Transformer，在文本处理中，接收单词的集合然后把这些集合转化成固定长度的特征向量。这个算法在哈希的同时会统计各个单词的词频。
- **IDF**: IDF是一个Estimator，在一个数据集上应用它的fit()方法，产生一个IDFModel。该IDFModel 接收特征向量（由HashingTF产生），然后计算每一个词在文档中出现的频次。IDF会减少那些在语料库中出现频率较高的词的权重。

特征提取：TF-IDF

过程描述：

- 在下面的代码段中，我们以一组句子开始
- 首先使用分解器Tokenizer把句子划分为单个词语
- 对每一个句子（词袋），使用HashingTF将句子转换为特征向量
- 最后使用IDF重新调整特征向量（这种转换通常可以提高使用文本特征的性能）

特征提取：TF-IDF

（1）导入TF-IDF所需要的包：

```
from pyspark.ml.feature import HashingTF,IDF,Tokenizer
```

（2）创建一个简单的DataFrame，每一个句子代表一个文档

```
sentenceData = spark.createDataFrame([
(0, "I heard about Spark and I love Spark"),
(0, "I wish Java could use case classes"),
(1, "Logistic regression models are neat")]).toDF("label", "sentence")
```

特征提取：TF-IDF

（3）得到文档集合后，即可用tokenizer对句子进行分词

```
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)
wordsData.show()
```

```
+-----+-----------------------------------+------------------------------------------+
|label|sentence                           |words                                     |
+-----+-----------------------------------+------------------------------------------+
|0    |I heard about Spark and I love Spark|[i, heard, about, spark, and, i, love, spark]|
|0    |I wish Java could use case classes  |[i, wish, java, could, use, case, classes]   |
|1    |Logistic regression models are neat |[logistic, regression, models, are, neat]    |
+-----+-----------------------------------+------------------------------------------+
```

特征提取：TF-IDF

（4）得到分词后的文档序列后，即可使用HashingTF的transform()方法把句子**哈希成特征向量**，这里设置哈希表的桶数为2000

```
hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=2000)
featurizedData = hashingTF.transform(wordsData)
featurizedData.select("words","rawFeatures").show(truncate=False)
```

```
+-----+-----------------------------------------------------------------------+
|label|rawFeatures                                                            |
+-----+-----------------------------------------------------------------------+
|0    |(2000,[240,333,1105,1329,1357,1777],[1.0,1.0,2.0,2.0,1.0,1.0])         |
|0    |(2000,[213,342,489,495,1329,1809,1967],[1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
|1    |(2000,[286,695,1138,1193,1604],[1.0,1.0,1.0,1.0,1.0])                  |
+-----+-----------------------------------------------------------------------+
```

特征提取：TF-IDF

（5）调用IDF方法来重新构造特征向量的规模，生成的变量idf是一个评估器，在特征向量上应用它的fit()方法，会产生一个IDFModel（名称为idfModel）。

```
idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
```

特征提取：TF-IDF

（6）调用IDFModel的transform()方法，可以得到每一个单词对应的TF-IDF度量值。

```
rescaledData = idfModel.transform(featurizedData)
rescaledData.select("features", "label").show(truncate=False)
```

```
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
|features
|
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
|(2000,[240,333,1105,1329,1357,1777],[0.6931471805599453,0.6931471805599453,1.3862943611198
906,0.5753641449035617,0.6931471805599453,0.6931471805599453])                             |
|(2000,[213,342,489,495,1329,1809,1967],[0.6931471805599453,0.6931471805599453,0.6931471805
599453,0.6931471805599453,0.28768207245178085,0.6931471805599453,0.6931471805599453])|
|(2000,[286,695,1138,1193,1604],[0.6931471805599453,0.6931471805599453,0.6931471805599453,
0.6931471805599453,0.6931471805599453])                                                    |
+-----------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
```

特征转换：标签和索引的转化

- 在机器学习处理过程中，为了方便相关算法的实现，经常需要把标签数据（一般是字符串）转化成**整数索引**，或是在计算结束后将整数索引还原为相应的标签

- Spark ML包中提供了几个相关的转换器，例如：StringIndexer、IndexToString、OneHotEncoder、VectorIndexer，它们提供了十分方便的特征转换功能，这些转换器类都位于org.apache.spark.ml.feature包下

- 值得注意的是，用于特征转换的转换器和其他的机器学习算法一样，也属于ML Pipeline模型的一部分，可以用来构成机器学习流水线，以StringIndexer为例，其存储着进行标签数值化过程的相关超参数，是一个Estimator，对其调用fit()方法即可生成相应的模型StringIndexerModel类，很显然，它存储了用于DataFrame进行相关处理的参数，是一个Transformer（其它转换器也是同一原理）

特征转换：标签和索引的转化

StringIndexer

StringIndexer转换器可以把一列**类别型的特征**（或标签）进行编码，使其数值化，索引的范围从0开始，该过程可以使得相应的特征索引化，使得某些无法接受类别型特征的算法可以使用，并提高诸如决策树等机器学习算法的效率

索引构建的顺序为标签的频率，优先编码频率较大的标签，所以出现频率最高的标签为0号
如果输入的是数值型的，会首先把它转化成字符型，然后再对其进行编码

特征转换：标签和索引的转化

（1）首先，引入所需要使用的类

```
from pyspark.ml.feature import StringIndexer
```

（2）其次，构建1个DataFrame，设置StringIndexer字名的列出输和列入输的。

```
df = spark.createDataFrame([(0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c")],["id", "category"])
indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
```

特征转换：标签和索引的转化

（3）然后，通过fit()方法进行模型训练，用训练出的模型对原数据集进行处理，并通过indexed.show()进行展示。

```
model = indexer.fit(df)
indexed = model.transform(df)
indexed.show()
```

```
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          2.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
+---+--------+-------------+
```

特征转换：标签和索引的转化

IndexToString

- 与StringIndexer相对应，IndexToString的作用是把标签索引的一列重新映射回原有的字符型标签

- 其主要使用场景一般都是**和StringIndexer配合**，先用StringIndexer将标签转化成标签索引，进行模型训练，然后在预测标签的时候再把标签索引转化成原有的字符标签

特征转换：标签和索引的转化

```
from pyspark.ml.feature import IndexToString, StringIndexer
toString = IndexToString(inputCol="categoryIndex", outputCol="originalCategory")
indexString = toString.transform(indexed)
indexString.select("id", "originalCategory").show()
```

```
+---+----------------+
| id|originalCategory|
+---+----------------+
|  0|               a|
|  1|               b|
|  2|               c|
|  3|               a|
|  4|               a|
|  5|               c|
+---+----------------+
```

特征转换：标签和索引的转化

VectorIndexer

- 之前介绍的StringIndexer是针对单个类别型特征进行转换，倘若所有特征都已经被组织在一个向量中，又想**对其中某些单个分量进行处理**时，Spark ML提供了VectorIndexer类来解决向量数据集中的类别性特征转换

- 通过为其提供maxCategories超参数，它可以自动识别哪些特征是类别型的，并且将原始值转换为类别索引。它基于不同特征值的数量来识别哪些特征需要被类别化，那些取值可能性最多不超过maxCategories的特征需要会被认为是类别型的

特征转换：标签和索引的转化

首先引入所需要的类，并构建数据集

```
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.linalg import Vector, Vectors
df = spark.createDataFrame([
(Vectors.dense(-1.0, 1.0, 1.0),),
(Vectors.dense(-1.0, 3.0, 1.0),),
(Vectors.dense(0.0, 5.0, 1.0), )], ["features"])
```

后然VectorIndexer练训型模行进并，列出输和入输置设，器换转。

```
indexer = VectorIndexer(inputCol="features", outputCol="indexed", maxCategories=2)
indexerModel = indexer.fit(df)
```

特征转换：标签和索引的转化

接下来，通过VectorIndexerModel的categoryMaps成员来获得被转换的特征及其映射，到看以可里这，换转被征特个两有共，分别是0号和2号。

```
categoricalFeatures = indexerModel.categoryMaps.keys()
print ("Choose"+str(len(categoricalFeatures))+ "categoricalfeatures:"+str(categoricalFeatures))
```

```
Choose2categoricalfeatures:KeysView({0: {0.0: 0, -1.0: 1}, 2: {1.0: 0}})
```

特征转换：标签和索引的转化

最后，把模型应用于原有的数据，并打印结果。

```
indexed = indexerModel.transform(df)
indexed.show()
```

```
+--------------+-------------+
|      features|      indexed|
+--------------+-------------+
|[-1.0,1.0,1.0]|[1.0,1.0,0.0]|
|[-1.0,3.0,1.0]|[1.0,3.0,0.0]|
| [0.0,5.0,1.0]|[0.0,5.0,0.0]|
+--------------+-------------+
```

逻辑斯蒂回归分类器

逻辑斯蒂回归（logistic regression）是统计学习中的经典分类方法，属于对数线性模型。logistic回归的因变量可以是二分类的，也可以是多分类的。

任务描述：以iris数据集（iris）为例进行分析（iris下载地址：

http://dblab.xmu.edu.cn/blog/wp-content/uploads/2017/03/iris.txt)

iris以鸢尾花的特征作为数据来源，数据集包含150个数据集，分为3类，每类50个数据，每个数据包含4个属性，是在数据挖掘、数据分类中非常常用的测试集、训练集。为了便于理解，这里主要用后两个属性（花瓣的长度和宽度）来进行分类。

逻辑斯蒂回归分类器

首先我们先取其中的后两类数据，用二项逻辑斯蒂回归进行二分类分析

第1步：导入本地向量Vector和Vectors，导入所需要的类。

```
from pyspark.ml.linalg import Vector,Vectors
from pyspark.sql import Row,functions
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer,HashingTF, Tokenizer
from pyspark.ml.classification import LogisticRegression, \
 LogisticRegressionModel,BinaryLogisticRegressionSummary, LogisticRegression
```

逻辑斯蒂回归分类器

第2步：我们定制一个函数，来返回一个指定的数据，然后读取文本文件，第一个map把每行的数据用“,”隔开，比如在我们的数据集中，每行被分成了5部分，前4部分是鸢尾花的4个特征，最后一部分是鸢尾花的分类；我们这里把特征存储在Vector中，创建一个Iris模式的RDD，然后转化成dataframe；最后调用show()方法来查看一下部分数据。

```
def f(x):
    rel = {}
    rel['features']=Vectors. dense(float(x[0]),float(x[1]),float(x[2]),float(x[3]))
    rel['label'] = str(x[4])
    return rel
```

逻辑斯蒂回归分类器

```
data = spark.sparkContext. \
    textFile("file:///data/iris.txt"). \
    map(lambda line: line.split(',')). \
    map(lambda p: Row(**f(p))). \
    toDF()
data.show()
```

```
+-----------------+-----------+
|         features|      label|
+-----------------+-----------+
|[5.1,3.5,1.4,0.2]|Iris-setosa|
|[4.9,3.0,1.4,0.2]|Iris-setosa|
|[4.7,3.2,1.3,0.2]|Iris-setosa|
|[4.6,3.1,1.5,0.2]|Iris-setosa|
|[5.0,3.6,1.4,0.2]|Iris-setosa|
|[5.4,3.9,1.7,0.4]|Iris-setosa|
|[4.6,3.4,1.4,0.3]|Iris-setosa|
|[5.0,3.4,1.5,0.2]|Iris-setosa|
|[4.4,2.9,1.4,0.2]|Iris-setosa|
```

逻辑斯蒂回归分类器

第3步：分别获取标签列和特征列，进行索引并进行重命名。

```
labelIndexer = StringIndexer(). \
    setInputCol("label"). \
    setOutputCol("indexedLabel"). \
    fit(data)

featureIndexer = VectorIndexer(). \
    setInputCol("features"). \
    setOutputCol("indexedFeatures"). \
    fit(data)
```

逻辑斯蒂回归分类器

第4步：设置LogisticRegression算法的参数。这里设置了循环次数为100次，规范化项为0.3等，具体可以设置的参数，可以通过explainParams()来获取，还能看到程序已经设置的参数的结果。

```
lr = LogisticRegression(). \
    setLabelCol("indexedLabel"). \
    setFeaturesCol("indexedFeatures"). \
    setMaxIter(100). \
    setRegParam(0.3). \
    setElasticNetParam(0.8)
print("LogisticRegression parameters:\n" + lr.explainParams())
```

```
LogisticRegression parameters:
aggregationDepth: suggested depth for treeAggregate (>= 2). (default: 2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penal
ty is an L2 penalty. For alpha = 1, it is an L1 penalty. (default: 0.0, current: 0.8)
family: The name of family which is a description of the label distribution to be used in t
he model. Supported options: auto, binomial, multinomial (default: auto)
```

逻辑斯蒂回归分类器

第5步：设置一个IndexToString的转换器，把预测的类别重新转化成字符型的。构建一个机器学习流水线，设置各个阶段。上一个阶段的输出将是本阶段的输入。

```
labelConverter = IndexToString(). \
    setInputCol("prediction"). \
    setOutputCol("predictedLabel"). \
    setLabels(labelIndexer.labels)
lrPipeline = Pipeline(). setStages([labelIndexer, featureIndexer, lr, labelConverter])
```

逻辑斯蒂回归分类器

第6步：把数据集随机分成训练集和测试集，其中训练集占70%。Pipeline本质上是一个评估器，当Pipeline调用fit()的时候就产生了一个PipelineModel，它是一个转换器。然后，这个PipelineModel就可以调用transform()来进行预测，生成一个新的DataFrame，即利用训练得到的模型对测试集进行验证。

```
trainingData, testData = data.randomSplit([0.7, 0.3])
lrPipelineModel = lrPipeline.fit(trainingData)
lrPredictions = lrPipelineModel.transform(testData)
```

逻辑斯蒂回归分类器

第7步：输出预测的结果，其中，select选择要输出的列，collect获取所有行的数据，用foreach把每行打印出来。

```
preRel = lrPredictions.select( \
    "predictedLabel", \
    "label", \
    "features", \
    "probability"). \
    collect()
for item in preRel:
    print(str(item['label'])+','+ \
    str(item['features'])+'-->prob='+ \
    str(item['probability'])+',predictedLabel'+ \
    str(item['predictedLabel']))
```

```
Iris-setosa,[4.4,2.9,1.4,0.2]-->prob=[0.5671476056093737,0.2408679503460851,0.1919844440445
411],predictedLabelIris-setosa
Iris-setosa,[4.6,3.1,1.5,0.2]-->prob=[0.5605817423924923,0.244521634686978,0.19489662292052
978],predictedLabelIris-setosa
Iris-setosa,[4.8,3.1,1.6,0.2]-->prob=[0.5539946054657883,0.24818715713930922,0.197818237394
90245],predictedLabelIris-setosa
Iris-versicolor,[4.9,2.4,3.3,1.0]-->prob=[0.3615463177428955,0.325227393665029,0.3132262885
9207563],predictedLabelIris-setosa
```

逻辑斯蒂回归分类器

第8步：对训练的模型进行评估。创建一个MulticlassClassificationEvaluator实例，用setter方法把预测分类的列名和真实分类的列名进行设置，然后计算预测准确率。

```
evaluator = MulticlassClassificationEvaluator(). \
        setLabelCol("indexedLabel"). \
        setPredictionCol("prediction")
lrAccuracy = evaluator.evaluate(lrPredictions)
```

0.653703703703703

逻辑斯蒂回归分类器

第9步：可以通过model来获取训练得到的逻辑斯蒂模型。lrPipelineModel是一个PipelineModel，因此，可以通过调用它的stages方法来获取模型，具体如下：

```
lrModel = lrPipelineModel.stages[2]
print ("Coefficients: \n " + str(lrModel.coefficientMatrix)+ \
       "\nIntercept: "+str(lrModel.interceptVector)+ \
       "\n numClasses: "+str(lrModel.numClasses)+ \
       "\n numFeatures: "+str(lrModel.numFeatures))
```

```
Coefficients:
 3 X 4 CSRMatrix
(0,2) -0.267
(0,3) -0.304
(1,3) 0.2365
Intercept: [0.9520258341191563,-0.3389411867950237,-0.6130846473241327]
 numClasses: 3
 numFeatures: 4
```