



《Spark编程基础（Python版）》

教材官网：<http://dblab.xmu.edu.cn/post/spark-python/>

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第8章 Spark MLlib

（PPT版本号：2019年春季学期）



扫一扫访问教材官网

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn ▶▶

主页：<http://www.cs.xmu.edu.cn/linziyu>





课程教材

Spark入门教程(Python版)

<http://dblab.xmu.edu.cn/blog/1709-2/>

纸质教材预期在2019年夏天上市销售



厦门大学林子雨



子雨大数据之Spark入门教程

披荆斩棘，在大数据丛林中开辟学习捷径



扫一扫访问在线教程

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



提纲

- 8.1 Spark MLlib简介
- 8.2 机器学习 workflow
- 8.3 特征抽取、转化和选择
- 8.4 分类与回归



高校大数据课程

公共服务平台

百度搜索厦门大学数据库实验室网站访问平台





8.1 Spark MLlib简介

8.1.1 什么是机器学习

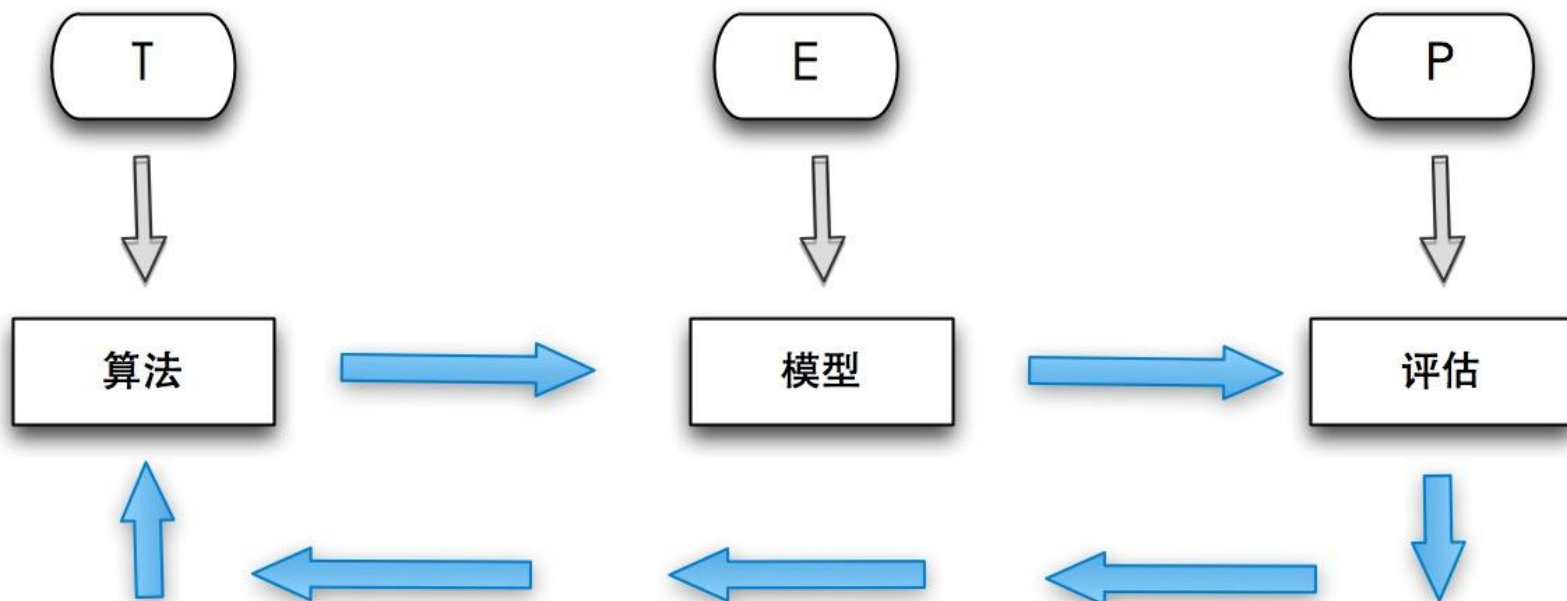
8.1.2 基于大数据的机器学习

8.1.3 Spark 机器学习库MLLib



8.1.1 什么是机器学习

机器学习可以看做是一门人工智能的科学，该领域的主要研究对象是人工智能。**机器学习利用数据或以往的经验，以此优化计算机程序的性能标准。**



机器学习强调三个关键词：算法、经验、性能



8.1.2 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，可以支持在全量数据上进行机器学习
- 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



8.1.3 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



8.1.3 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，这一点很重要
- 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（比如**Weka**）



8.1.3 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。



8.1.3 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DataFrames 高层次的API，可以用来构建机器学习 workflow (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 workflow 式 API 套件



8.1.3 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



8.2 机器学习流水线

8.2.1 机器学习流水线概念

8.2.2 构建一个机器学习流水线



8.2.1 机器学习流水线概念

在介绍流水线之前，先来了解几个重要概念：

- **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。
- 它被ML Pipeline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



8.2.1 机器学习流水线概念

Transformer: 翻译成转换器，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。

技术上，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



8.2.1 机器学习流水线概念

Estimator: 翻译成估计器或评估器，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。从技术上讲，Estimator 实现了一个方法 `fit()`，它接受一个 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。



8.2.1 机器学习流水线概念

- **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。ParamMap是一组（参数，值）对
- **PipeLine:** 翻译为流水线或者管道。流水线将多个 workflow 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



8.2.2 流水线工作过程

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

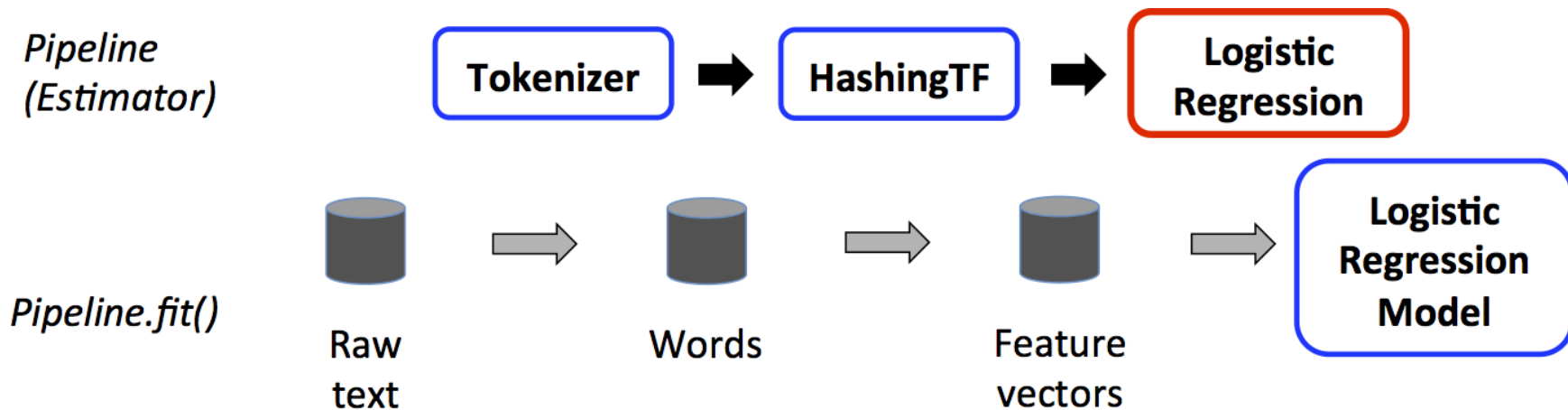
```
>>> pipeline = Pipeline(stages=[stage1,stage2,stage3])
```

然后就可以把训练数据集作为输入参数，调用 Pipeline 实例的 fit 方法来开始以流的方式来处理源训练数据。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



8.2.2 流水线工作过程

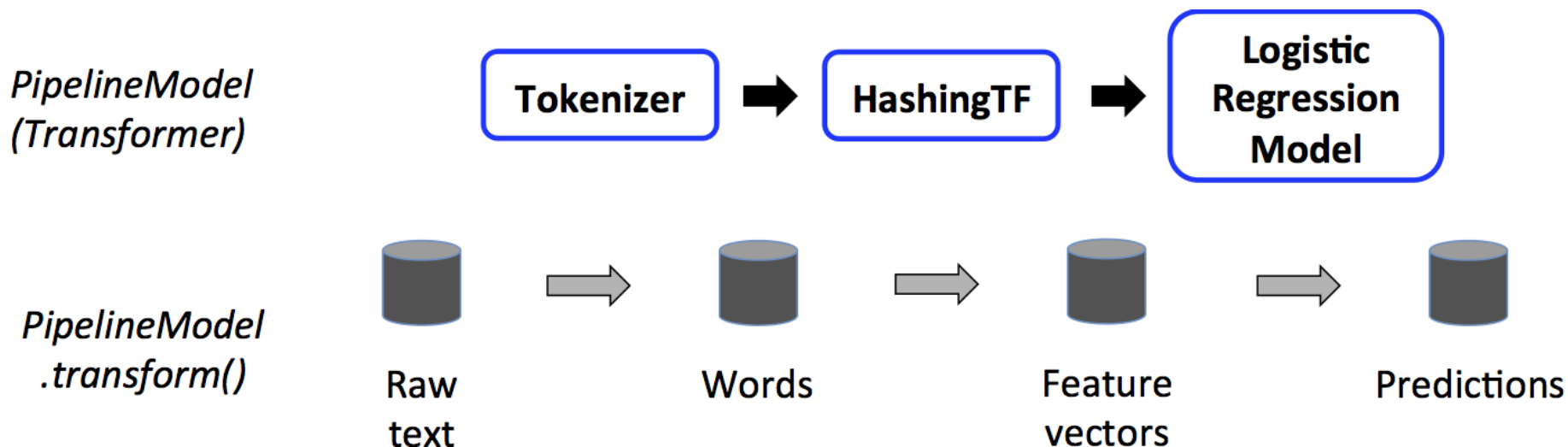
- 流水线的各个阶段按顺序运行，输入的DataFrame在它通过每个阶段时被**转换**





8.2.2 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**。这个管道模型将在测试数据的时候使用。下图说明了这种用法。





8.2.3 构建一个机器学习流水线

本节以**逻辑斯蒂回归**为例，构建一个典型的机器学习过程，来具体介绍一下流水线是如何应用的

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。



8.2.3 构建一个机器学习流水线

- 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，如下代码段所示：

```
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
```



8.2.3 构建一个机器学习流水线

- (1) 引入要包含的包并构建训练数据集

```
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"])
```




8.2.3 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)
```



8.2.3 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，在它的fit()方法运行之后，它将产生一个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



8.2.3 构建一个机器学习流水线

(4) 构建测试数据

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



8.2.3 构建一个机器学习流水线

(5) 调用之前训练好的PipelineModel的transform()方法，让测试数据按顺序通过拟合的流水线，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=[0.155543713844,0.844456286156], prediction=1.000000
(5, l m n) --> prob=[0.830707735211,0.169292264789], prediction=0.000000
(6, spark hadoop spark) --> prob=[0.0696218406195,0.93037815938],
prediction=1.000000
(7, apache hadoop) --> prob=[0.981518350351,0.018481649649],
prediction=0.000000
```



8.3 特征提取、转换和选择

8.3.1 特征提取

8.3.2 特征转换

8.3.3 特征选取



8.3.1 特征抽取：TF-IDF

- “**词频—逆向文件频率**”（TF-IDF）是一种在文本挖掘中广泛使用的特征向量化方法，它可以体现一个文档中词语在语料库中的重要程度。
- 词语由**t**表示，文档由**d**表示，语料库由**D**表示。词频**TF(t,d)**是词语**t**在文档**d**中出现的次数。文件频率**DF(t,D)**是包含词语的文档的个数。
- **TF-IDF**就是在数值化文档信息，衡量词语能提供多少信息以区分文档。其定义如下：

$$IDF(t, D) = \log \frac{|D|+1}{DF(t,D)+1}$$

- **TF-IDF** 度量值表示如下：

$$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$



8.3.1 特征抽取：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，接收词条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词条的词频。

• **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。



8.3.1 特征抽取：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）



8.3.1 特征抽取：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")
```



8.3.1 特征抽取：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| heard about Spa...|[i, heard, about,...|
|  0| wish Java could...|[i, wish, java, c...|
|  1|Logistic regressi...|[logistic, regres...|
+-----+-----+-----+
```



8.3.1 特征抽取：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)
```

```
+-----+-----+
|words                |rawFeatures                |
+-----+-----+
[[i, heard, about, spark, and, i, love,
spark]](2000,[240,333,1105,1329,1357,1777],[1.0,1.0,2.0,2.0,1.0,1.0]) |
[[i, wish, java, could, use, case, classes]
(2000,[213,342,489,495,1329,1809,1967],[1.0,1.0,1.0,1.0,1.0,1.0,1.0])|
[[logistic, regression, models, are, neat]
(2000,[286,695,1138,1193,1604],[1.0,1.0,1.0,1.0,1.0]) |
+-----+-----+
```



8.3.1 特征抽取：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)
```



8.3.1 特征抽取：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
|label|
+-----+
-----+-----+
|(2000,[240,333,1105,1329,1357,1777],[0.6931471805599453,0.6931471805599453,1.386294361119890
6,0.5753641449035617,0.6931471805599453,0.6931471805599453])|0|
|(2000,[213,342,489,495,1329,1809,1967],[0.6931471805599453,0.6931471805599453,0.693147180559
9453,0.6931471805599453,0.28768207245178085,0.6931471805599453,0.6931471805599453])|0|
|(2000,[286,695,1138,1193,1604],[0.6931471805599453,0.6931471805599453,0.6931471805599453,0.69
31471805599453,0.6931471805599453])|1|
+-----+
-----+-----+
```



8.3.2 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**（其他转换器也是同一原理）



8.3.2 特征转换：标签和索引的转化

•StringIndexer

- StringIndexer转换器可以把一系列**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



8.3.2 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
```



8.3.2 特征转换：标签和索引的转化

(2) 其次，构建1个DataFrame，设置StringIndexer的输入列和输出列的名字。

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



8.3.2 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|
+---+-----+-----+
```



8.3.2 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



8.3.2 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|
+---+-----+
```



8.3.2 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



8.3.2 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"])
```



8.3.2 特征转换：标签和索引的转化

然后，构建VectorIndexer转换器，设置输入和输出列，并进行模型训练。

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




8.3.2 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，这里可以看到，共有两个特征被转换，分别是0号和2号。

```
>>> categoricalFeatures =  
indexerModel.categoryMaps.keys()  
>>> print ("Choose"+str(len(categoricalFeatures))+ \  
... "categorical features:"+str(categoricalFeatures))  
Chose 2 categorical features: [0, 2]
```



8.3.2 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]|
+-----+-----+
```



8.4 分类与回归

8.4.1 逻辑斯蒂回归分类器

8.4.2 决策树分类器



8.4.1 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。



8.4.1 逻辑斯蒂回归分类器

任务描述：以iris数据集（iris）为例进行分析（iris下载地址：<http://dbllab.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。



8.4.1 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
```



8.4.1 逻辑斯蒂回归分类器

2.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
```

剩余代码见下一页



8.4.1 逻辑斯蒂回归分类器

```
>>> data = spark.sparkContext. \  
...   textFile("file:///usr/local/spark/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|  
.....  
+-----+-----+  
only showing top 20 rows
```




8.4.1 逻辑斯蒂回归分类器

3.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)
```



8.4.1 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())
```



8.4.1 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])
```



8.4.1 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)
```



8.4.1 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']))
```



8.4.1 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)  
>>> lrAccuracy  
0.7774712643678161 #模型预测的准确率
```



8.4.1 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
(1,3) 0.4332
(2,2) -0.2472
(2,3) -0.1689
Intercept: [-0.11530503231364186,-0.63496556499483,0.750270597308472]
numClasses: 3
numFeatures: 4
```



8.4.2 决策树分类器

决策树（**decision tree**）是一种基本的分类与回归方法，这里主要介绍用于分类的决策树。决策树模式呈树形结构，其中每个内部节点表示一个属性上的测试，每个分支代表一个测试输出，每个叶节点代表一种类别。学习时利用训练数据，根据损失函数最小化的原则建立决策树模型；预测时，对新的数据，利用决策树模型进行分类

决策树学习通常包括**3**个步骤：特征选择、决策树的生成和决策树的剪枝



8.4.2 决策树分类器

我们以iris数据集（iris）为例进行分析（iris下载地址：<http://dmlab.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。



8.4.2 决策树分类器

1. 导入需要的包

```
>>> from pyspark.ml.classification import  
DecisionTreeClassificationModel  
>>> from pyspark.ml.classification import DecisionTreeClassifier  
>>> from pyspark.ml import Pipeline, PipelineModel  
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator  
>>> from pyspark.ml.linalg import Vector, Vectors  
>>> from pyspark.sql import Row  
>>> from pyspark.ml.feature import  
IndexToString, StringIndexer, VectorIndexer
```



8.4.2 决策树分类器

2.第2步：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。

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



8.4.2 决策树分类器

第3步：进一步处理特征和标签，把数据集随机分成训练集和测试集，其中训练集占70%。

```
>>> labelIndexer = StringIndexer(). \  
... setInputCol("label"). \  
... setOutputCol("indexedLabel"). \  
... fit(data)  
>>> featureIndexer = VectorIndexer(). \  
... setInputCol("features"). \  
... setOutputCol("indexedFeatures"). \  
... setMaxCategories(4). \  
... fit(data)  
>>> labelConverter = IndexToString(). \  
... setInputCol("prediction"). \  
... setOutputCol("predictedLabel"). \  
... setLabels(labelIndexer.labels)  
>>> trainingData, testData = data.randomSplit([0.7, 0.3])
```



8.4.2 决策树分类器

第4步：创建决策树模型**DecisionTreeClassifier**，通过**setter**的方法来设置决策树的参数，也可以用**ParamMap**来设置。这里仅需要设置特征列（**FeaturesCol**）和待预测列（**LabelCol**）。具体可以设置的参数可以通过**explainParams()**来获取。

```
>>> dtClassifier = DecisionTreeClassifier(). \  
... setLabelCol("indexedLabel"). \  
... setFeaturesCol("indexedFeatures")
```



8.4.2 决策树分类器

第5步：构建机器学习流水线（Pipeline），在训练数据集上调用fit()进行模型训练，并在测试数据集上调用transform()方法进行预测。

```
>>> dtPipeline = Pipeline(). \
... setStages([labelIndexer, featureIndexer, dtClassifier, labelConverter])
>>> dtPipelineModel = dtPipeline.fit(trainingData)
>>> dtPredictions = dtPipelineModel.transform(testData)
>>> dtPredictions.select("predictedLabel", "label", "features").show(20)
```

predictedLabel	label	features
Iris-setosa	Iris-setosa	[4.4,3.0,1.3,0.2]
Iris-setosa	Iris-setosa	[4.6,3.4,1.4,0.3]
Iris-setosa	Iris-setosa	[4.9,3.1,1.5,0.1]
Iris-setosa	Iris-setosa	[5.0,3.2,1.2,0.2]

剩余代码见下一页



8.4.2 决策树分类器

```
>>> evaluator = MulticlassClassificationEvaluator(). \  
... setLabelCol("indexedLabel"). \  
... setPredictionCol("prediction")  
>>> dtAccuracy = evaluator.evaluate(dtPredictions)  
>>> dtAccuracy  
0.9726976552103888 #模型的预测准确率
```



8.4.2 决策树分类器

第6步：可以通过调用DecisionTreeClassificationModel的toDebugString方法，查看训练的决策树模型结构。

```
>>> treeModelClassifier = dtPipelineModel.stages[2]
>>> print("Learned classification tree model:\n" + \
... str(treeModelClassifier.toDebugString))
```

Learned classification tree model:

DecisionTreeClassificationModel (uid=DecisionTreeClassifier_5427198bb4c1)
of depth 5 with 15 nodes

If (feature 2 <= 2.45)

Predict: 2.0

Else (feature 2 > 2.45)

If (feature 2 <= 4.75)

Predict: 0.0

Else (feature 2 > 4.75)

If (feature 3 <= 1.75)

If (feature 2 <= 4.95)

.....



本章小结

1、机器学习、Spark MLlib的基本概念

2、机器学习 workflow

构建一个机器学习 workflow、特征抽取、转化和选择

[TF-IDF、Word2Vec、CountVectorizer、标签和索引的转化、卡方选择器]

3、分类与回归

(逻辑斯蒂回归分类器、决策树分类器)



附录A：主讲教师林子雨简介



主讲教师：林子雨

单位：厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

个人网页: <http://dblab.xmu.edu.cn/post/linziyu>

数据库实验室网站: <http://dblab.xmu.edu.cn>



扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革委员会副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度和2017年度厦门大学教学类奖教金获得者，荣获2017年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学基金青年基金项目(No.61303004)、1项福建省自然科学基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学研协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过500万字高价值的研究和教学资料，累计网络访问量超过500万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过100万次。



附录B：大数据学习路线图



大数据学习路线图访问地址：<http://dblab.xmu.edu.cn/post/10164/>



附录C： 《大数据技术原理与应用》教材

《大数据技术原理与应用——概念、存储、处理、分析与应用（第2版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-44330-4 定价：49.80元



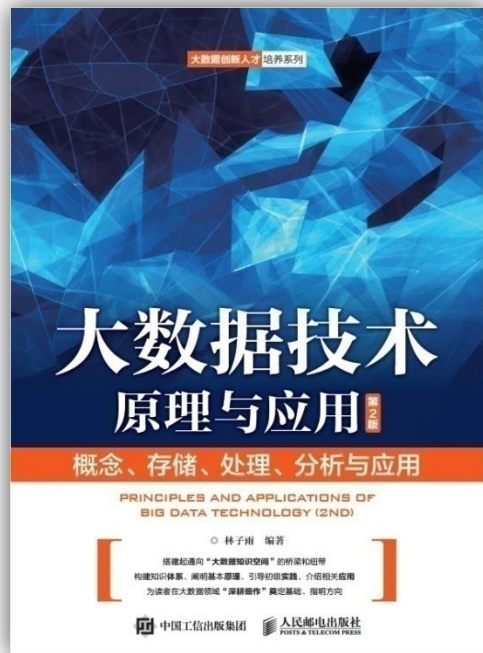
扫一扫访问教材官网

全书共有15章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：

<http://dbl原因.xmu.edu.cn/post/bigdata>





附录D：《大数据基础编程、实验和案例教程》

本书是与《大数据技术原理与应用（第2版）》教材配套的唯一指定实验指导书

大数据教材



1+1黄金组合
厦门大学林子雨编著

配套实验指导书



- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，五套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

清华大学出版社 ISBN:978-7-302-47209-4 定价：59元



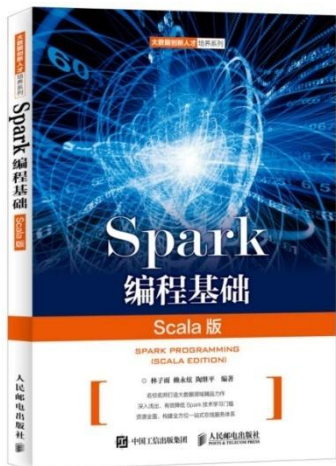
附录E：《Spark编程基础（Scala版）》

《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：<http://dblalab.xmu.edu.cn/post/spark/>



本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



附录F：高校大数据课程公共服务平台



高校大数据课程

公 共 服 务 平 台

<http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/>



扫一扫访问平台主页



扫一扫观看3分钟FLASH动画宣传片

The background of the slide features several faint, light-blue silhouettes of people. At the top, there are two groups of people standing and holding hands. On the right side, a person is shown in profile, looking towards the center. On the left side, two people are shown in profile, one appearing to be speaking or gesturing towards the other. The overall scene suggests a group of people in a meeting or presentation setting.

Thank You!

Department of Computer Science, Xiamen University, 2019