大数据分析处理案例

Fred

目录

基础知识复习

PySpark - RDD Basics

- Initializing Spark
- Loading Data
- Retrieving RDD Information
- Applying Functions
- Selecting Data
- Iterating
- Reshaping Data
- Mathematical Operations
- Sort
- Repartitioning
- Saving
- Stopping SparkContext
- Execution

Initializing Spark

```
from pyspark import SparkContext
sc = SparkContext(master = 'local[2]')
sc.version
sc.pythonVer
sc.master
str(sc.sparkHome)
str(sc.sparkUser())
sc.appName
sc.applicationId
sc.defaultParallelism
sc.defaultMinPartitions
```

Initializing Spark (CONT)

```
from pyspark import SparkConf, SparkContext
conf = (SparkConf()
 .setMaster("local")
 .setAppName("My app")
 .set("spark.executor.memory", "1g"))
sc = SparkContext(conf = conf)

$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Loading Data

```
rdd = sc.parallelize([('a',7),('a',2),('b',2)])
rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
rdd3 = sc.parallelize(range(100))
rdd4 = sc.parallelize([("a",["x","y","z"]), ("b",["p", "r"]
```

```
textFile = sc.textFile("/my/directory/*.txt")
textFile2 = sc.wholeTextFiles("/my/directory/")
```

textFile wholeTextFiles 的区别
https://www.cnblogs.com/jagel-95/p/9766254.html

Retrieving RDD Information

```
rdd.getNumPartitions() List the number of partitions
rdd.count() Count RDD instances 3
rdd.countByKey()
rdd.countByValue()
rdd.collectAsMap()
rdd3.sum()
sc.parallelize([]).isEmpty()
```

Retrieving RDD Information (CONT)

```
rdd3.max()
rdd3.min()
rdd3.mean()
rdd3.stdev()
rdd3.variance()
rdd3.histogram(3)
rdd3.stats()
```

Applying Functions

```
rdd.map(lambda x: x+(x[1],x[0])).collect()]
rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
rdd5.collect()
rdd4.flatMapValues(lambda x: x).collect()
```

Selecting Data

Getting

```
>>> rdd.collect()
>>> rdd.take(2)
>>> rdd.first()
>>> rdd.top(2)
```

Sampling *def sample(withReplacement: Boolean,fraction: Double,seed: Long)*

```
>>> rdd3.sample(False, 0.15, 81).collect()
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect()
>>> rdd5.distinct().collect()
>>> rdd.keys().collect()
```

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)
```

map : 返回*rdd*
foreach: 只执行操作，不返回

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y).collect()
>>> rdd.reduce(lambda a, b: a + b)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)
 .mapValues(list)
 .collect()
>>> rdd.groupByKey()
 .mapValues(list)
 .collect()
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
```

Mathematical Operations

```
>>> rdd.subtract(rdd2).collect()
>>> rdd2.subtractByKey(rdd).collect()
>>> rdd.cartesian(rdd2).collect()
```

Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect()
>>> rdd2.sortByKey().collect()
```

Repartitioning

```
rdd.repartition(4)
```

重新洗牌哈希后分四个桶，使数据更均匀

```
rdd.coalesce(1)
```

不做重新的shuffle会出现分布不均，但其速度快

？？？？？？

Saving

```
rdd.saveAsTextFile("rdd.txt")
rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
    'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
sc.stop()
```

Execution

```
./bin/spark-submit examples/src/main/python/pi.py
```

PySpark SQL Basics

- Initializing SparkSession
- Creating DataFrames
- Inspect Data
- Duplicate Values
- Queries
- Add, Update & Remove Columns
- GroupBy
- Filter
- Sort
- Missing & Replacing Values
- Repartitioning
- Running SQL Queries Programmatically
- Output
- Stopping SparkSession

Initializing SparkSession

```
from pyspark.sql import SparkSession
spark = SparkSession \
 .builder \
 .appName("Python Spark SQL basic example") \
 .config("spark.some.config.option", "some-value") \
 .getOrCreate()
```

Creating DataFrames: From RDD

```
>>> from pyspark.sql.types import *
 Infer Schema
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1]
>>> peopledf = spark.createDataFrame(people)
 Specify Schema
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1]
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) f
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
```

Creating DataFrames: From DataSource

```
 JSON
>>> df = spark.read.json("customer.json")
>>> df.show()
>>> df2 = spark.read.load("people.json", format="json")
 Parquet files
>>> df3 = spark.read.load("users.parquet")
 TXT files
>>> df4 = spark.read.text("people.txt")
```

Inspect Data

```
>>> df.dtypes Return df column names and data types
>>> df.show() Display the content of df
>>> df.head() Return first n rows
>>> df.first() Return first row
>>> df.take(2) Return the first n rows
>>> df.schema Return the schema of df

>>> df.describe().show() Compute summary statistics >>> df
>>> df.count() Count the number of rows in df
>>> df.distinct().count() Count the number of distinct rows
>>> df.printSchema() Print the schema of df
>>> df.explain() Print the (logical and physical) plans
```

Duplicate Values

```
df = df.dropDuplicates()
```

Queries: Select

```
>>> from pyspark.sql import functions as F
Select
>>> df.select("firstName").show() Show all entries in first
>>> df.select("firstName","lastName").show()
>>> df.select("firstName","age",explode("phoneNumbers")
 .alias("contactInfo"))
 .select("contactInfo.type", "firstName", "age")
 .show()
>>> df.select(df["firstName"],df["age"]+ 1)
>>> df.select(df['age'] > 24).show()
```

Queries: When && Like

```
df.select("firstName", F.when(df.age > 30, 1).otherwise(0))
df[df.firstName.isin("Jane","Boris")].collect()
df.select("firstName", df.lastName.like("Smith")).show()

df.select("firstName", df.lastName.startswith("Sm")).show()
df.select(df.lastName.endswith("th")).show()

df.select(df.firstName.substr(1, 3).alias("name")).collect

df.select(df.age.between(22, 24)).show()
```

Add, Update & Remove Columns

```
df = df.withColumn('city',df.address.city) \
 .withColumn('postalCode',df.address.postalCode) \
 .withColumn('state',df.address.state) \
 .withColumn('streetAddress',df.address.streetAddress) \
 .withColumn('telePhoneNumber', explode(df.phoneNumber.numb
 .withColumn('telePhoneType', explode(df.phoneNumber.type))

df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')

df = df.drop("address", "phoneNumber")
df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
df.groupBy("age").count().show()
```

Filter

```
df.filter(df["age"]>24).show()
```

Sort

```
peopledf.sort(peopledf.age.desc()).collect()
df.sort("age", ascending=False).collect()
df.orderBy(["age","city"],ascending=[0,1]).collect()
```

Missing & Replacing Values

```
df.na.fill(50).show()
df.na.drop().show()
df.na.replace(10, 20).show()
```

Repartitioning

```
>>> df.repartition(10).rdd.getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

Running SQL Queries Programmatically

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")

df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people
 .show()
```

Output

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()

>>> df.select("firstName", "city").write.save("nameAndCity
>>> df.select("firstName", "age").write.save("namesAndAges
```

Stopping SparkSession

```
spark.stop()
```

用 Spark 对沃尔玛股票进行数据分析

加载数据

from pyspark.sql import SparkSession spark = SparkSession.builder.appName('walmart').getOrCreate() df = spark.read.csv('walmart_stock.csv', inferSchema=True, header=True)

查看数据

- 所有的列名
- 打印 Schema 表结构
- 输出前 5 行

查看统计信息

```
df.describe().show()
```

修改浮点数精度

```
from pyspark.sql.functions import format_number

summary = df.describe()
summary.select(summary['summary'],
                  format_number(summary['Open'].cast('float
                  format_number(summary['High'].cast('float
                  format_number(summary['Low'].cast('float
                  format_number(summary['Close'].cast('floa
                  format_number(summary['Volume'].cast('int
                 ).show()
```

新建 1 个字段叫做 HV Ratio，它通过 High 和 Volume 的比值得到

```
df_hv = df.withColumn('HV Ratio', df['High']/df['Volume'])
df_hv.show()
```

请问“High”字段最大的一天是哪天？

```
df.orderBy(df['High'].desc()).select(['Date']).head(1)[0]['Date']
```

字段 Close 的均值是多少

```
from pyspark.sql.functions import mean
df.select(mean('Close')).show()
```

Volume 列的最大和最小值是多少

```
from pyspark.sql.functions import min, max
df.select(max('Volume'),min('Volume')).show()
```

Close 字段小于 60 的天数有多少天

df.filter(df['Close'] < 60).count()

请问 High 字段大于 80 的天占总天数的百分比？

df.filter('High $>$ 80').count() * 100/df.count()

请问 High 和 Volume 两列数据的皮尔逊相关系数是多少?

```
df.corr('High', 'Volume')

from pyspark.sql.functions import corr

df.select(corr(df['High'], df['Volume'])).show()
```

每一年最大的 “High” 值是多少？

```
from pyspark.sql.functions import (dayofmonth, hour,
                                   dayofyear, month,
                                   year, weekofyear,
                                  format_number, date_forma

year_df = df.withColumn('Year', year(df['Date']))

year_df.groupBy('Year').max()['Year', 'max(High)'].show()
```

每月的平均 Close 值是多少

```
#Create a new column Month from existing Date column
month_df = df.withColumn('Month', month(df['Date']))

#Group by month and take average of all other columns
month_df = month_df.groupBy('Month').mean()

#Sort by month
month_df = month_df.orderBy('Month')

#Display only month and avg(Close), the desired columns
month_df['Month', 'avg(Close)'].show()
```

Spark ML

Spark Mllib 主要内容

- 机器学习算法：常规机器学习算法包括分类、回归、聚类和协同过滤。
- 特征工程：特征提取、特征转换、特征选择以及降维。
- 管道：构造、评估和调整的管道的工具。
- 存储：保存和加载算法、模型及管道
- 实用工具：线性代数，统计，数据处理等。

Spark 机器学习库（MLlib）中文指南
https://zhuanlan.zhihu.com/p/24320870

API 更迭

- RDD-based API (Spark MLlib, org.apache.spark.mllib)
- Dataset API(Spark ML org.apache.spark.ml)

Spark+AI Summit

Welcome to the largest data & machine learning conference in the world. It's a unique experience for developers, data engineers, data scientists, and decision-makers to collaborate at the intersection of data and ML. Attendees will learn about the latest advances in Apache Spark and ML technologies like TensorFlow, MLflow, PyTorch as well as real-world enterprise AI best practices.

欢迎来到世界上最大的数据和机器学习会议。对于开发人员，数据工程师，数据科学家和决策者来说，这是一种独特的体验，可以在数据和 ML 的交叉点上进行协作。与会者将了解 Apache Spark 和 ML 技术的最新进展，如 TensorFlow，MLflow，PyTorch 以及现实企业 AI 最佳实践。

开源的 AI 算法与 Spark 的结合

- XGBoost on Spark
- TensorFlowOnSpark(Yahoo)
- BigDL(Intel)
- Spark on Angel(腾讯)

ML Pipelines

ML Pipelines 提供了一组基于 DataFrame 构建的统一的高级 API，可帮助用户创建和调整实用的机器学习流程。

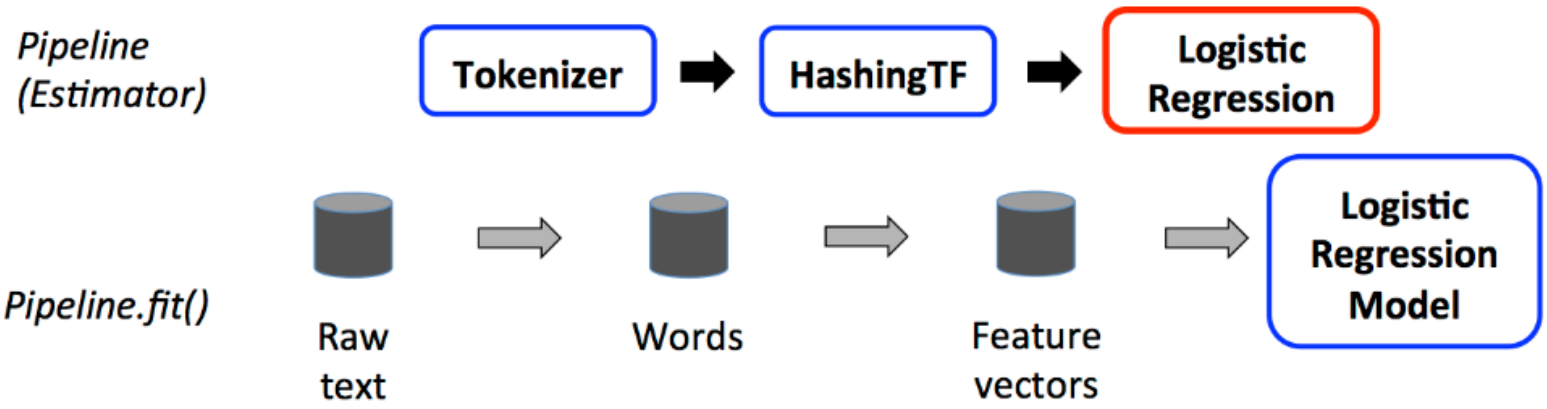
ML Pipelines 之核心概念

- DataFrame：使用 Spark SQL 中的 DataFrame 作为 ML 数据集，它可以包含各种数据类型。例如，DataFrame 可以具有存储文本，特征向量，真实标签和预测的不同列。
- Transformer：Transformer 是一种可以将一个 DataFrame 转换为另一个 DataFrame 的算法。例如，ML 模型是 Transformer，其将具有特征的 DataFrame 转换为具有预测的 DataFrame。
- Estimator：Estimator 是一种算法，可以适应 DataFrame 以生成 Transformer。例如，学习算法是 Estimator，其在 DataFrame 上训练并产生模型。

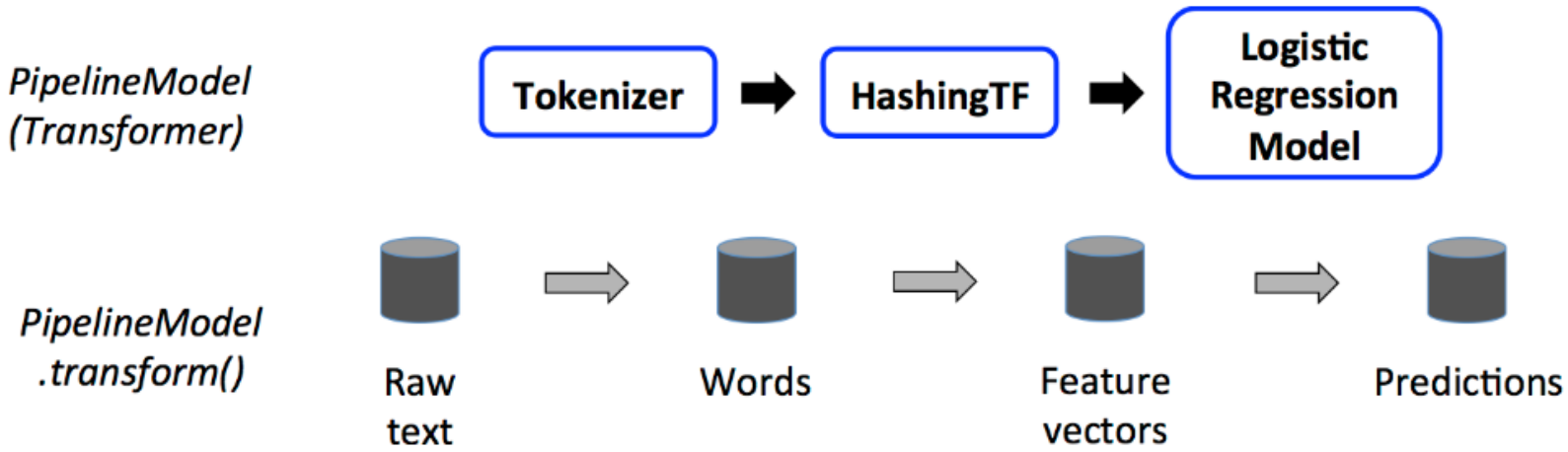
ML Pipelines 之核心概念 (续)

- Pipeline：Pipeline 将多个 Transformer 和 Estimator 链接在一起以指定 ML 工作流程
- Parameter：所有 Transformers 和 Estimators 现在共享一个用于指定参数的通用 API

How it Works：training time



How it Works: test time



Linear Regression: Train

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression


training = spark.createDataFrame([
    (1.0, Vectors.dense([0.0, 1.1, 0.1])),
    (0.0, Vectors.dense([2.0, 1.0, -1.0])),
    (0.0, Vectors.dense([2.0, 1.3, 1.0])),
    (1.0, Vectors.dense([0.0, 1.2, -0.5]))], ["label", "fea

lr = LogisticRegression(maxIter=10, regParam=0.01)

model1 = lr.fit(training)
```

Logistic Regression: Prediction

```
# Prepare test data
test = spark.createDataFrame([
    (1.0, Vectors.dense([-1.0, 1.5, 1.3])),
    (0.0, Vectors.dense([3.0, 2.0, -0.1])),
    (1.0, Vectors.dense([0.0, 2.2, -1.5]))], ["label", "fea

prediction = model1.transform(test)
result = prediction.select("features", "label", "myProbabil
    .collect()

for row in result:
    print("features=%s, label=%s -> prob=%s, prediction=%s"
          % (row.features, row.label, row.myProbability, ro
```

Logistic Regression：复杂一点的

```
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"])

# Configure an ML pipeline, which consists of three stages
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), ou
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

Logistic Regression：复杂一点的

```
# Prepare test documents, which are unlabeled (id, text) tu
test = spark.createDataFrame([
    (4, "spark i j k"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])

# Make predictions on test documents and print columns of i
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "
for row in selected.collect():
    rid, text, prob, prediction = row
    print("(%d, %s) --> prob=%s, prediction=%f" % (rid, tex
```

ML Tuning: model selection and hyperparameter tuning

- Cross-Validation
- Train-Validation Split

http://spark.apache.org/docs/latest/ml-tuning.html

Extracting, transforming and selecting features

Feature Extractors

- TF-IDF
- Word2Vec
- CountVectorizer
- FeatureHasher

Feature Selectors

- VectorSlicer
- RFormula
- ChiSqSelector

Extracting, transforming and selecting features

Locality Sensitive Hashing

- LSH Operations
  - Feature Transformation
  - Approximate Similarity Join
  - Approximate Nearest Neighbor Search
- LSH Algorithms
  - Bucketed Random Projection for Euclidean Distance
  - MinHash for Jaccard Distance

Extracting, transforming and selecting features

Feature Transformers

- Tokenizer/StopWordsRemover/n-gram
- Binarizer
- PCA
- PolynomialExpansion
- Discrete Cosine Transform (DCT)
- StringIndexer
- IndexToString
- OneHotEncoderEstimator
- VectorIndexer

XGBoost on Spark

没有 Python 版本的接口，Scala 版本，如果对 Spark ML 感兴趣的同学，可以参考如下教程

https://xgboost.readthedocs.io/en/latest/jvm/xgboost4j_spark_tutorial.html

把这个解法的 xgboost 改为 spark 的实现
https://www.kaggle.com/viveksrinivasan/eda-ensemble-model-top-10-percentile