

实验四实验报告

实验四实验报告

Spark安装与配置

下载Spark

IDEA安装Scala插件

新建Scala项目

导入Spark的jar包

任务一

1. 统计application_data.csv中贷款金额AMT_CREDIT 的分布情况

思路

全部代码

在Spark WebUI中查看

2. 统计application_data.csv中AMT_CREDIT-AMT_INCOME_TOTAL最高和最低的各十条记录

思路

全部代码

在Spark WebUI中查看

任务二

1. 统计所有男性客户的小孩个数类型占比情况

思路

全部代码

在Spark WebUI查看

2. 统计每个客户每天的平均收入，并按照从大到小排序

思路

全部代码

在Spark WebUI中查看

任务三

Step1: 数据预处理及特征提取

Step2: 划分训练集和测试集

Step3: 训练模型

Step4: 评估模型效果

全部代码:

在Spark WebUI中查看

遇到的困难及解决方法

Spark安装与配置

本次实验的使用的是**IDEA+Scala**

安装配置的主要参考网站如下:

[在idea 2021 上 配置本地 scala 2.12 spark 3.0.2 开发环境](#)

[解决java.io.FileNotFoundException: HADOOP_HOME and hadoop.home.dir are unset.](#)

下载Spark

下载地址: <https://spark.apache.org/downloads.html>

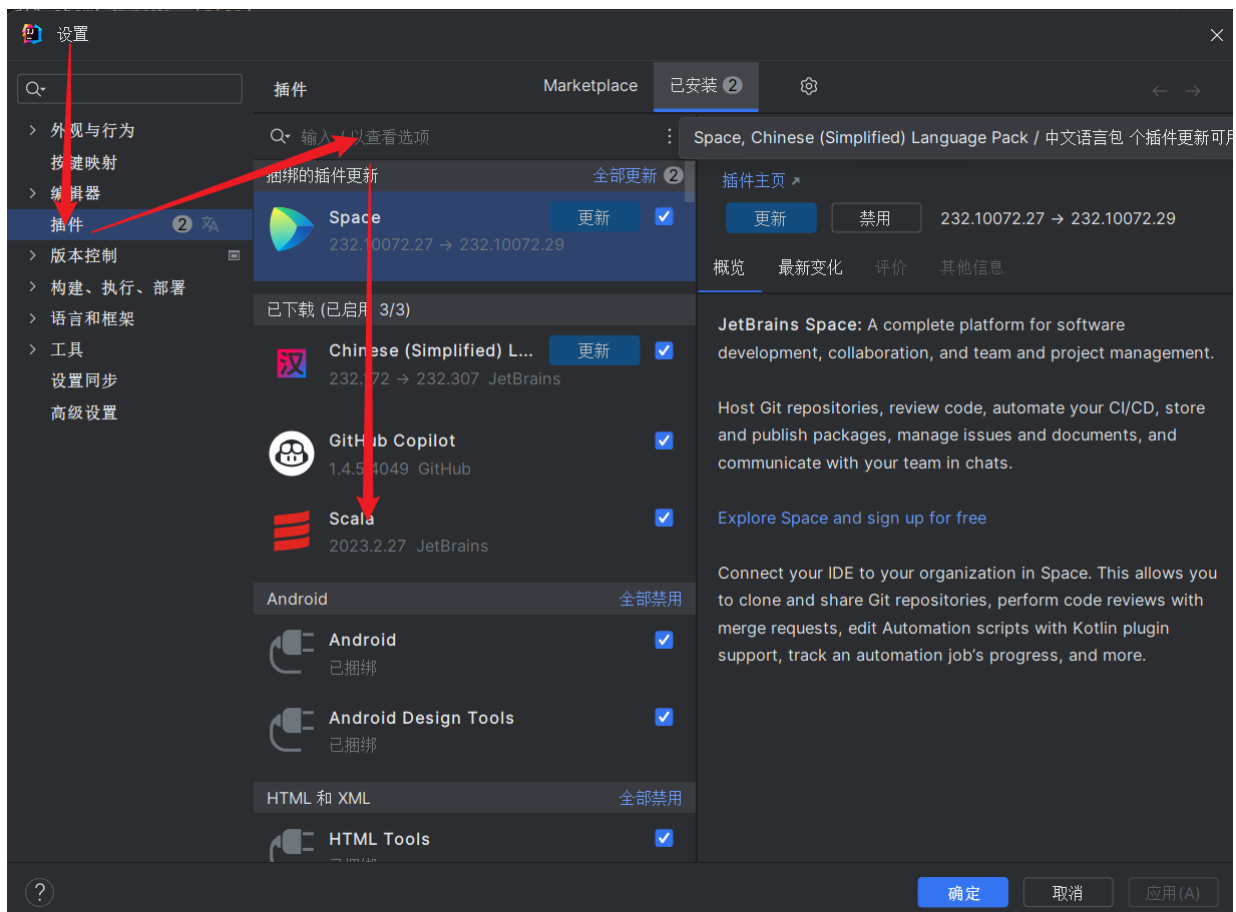
Download Apache Spark™

1. Choose a Spark release: 3.5.0 (Sep 13 2023) ▾
2. Choose a package type: Pre-built for Apache Hadoop 3.3 and later ▾
Pre-built for Apache Hadoop 3.3 and later
Pre-built for Apache Hadoop 3.3 and later (Scala 2.13)
Pre-built with user-provided Apache Hadoop
Source Code
3. Download Spark: spark-3.5.0-bin-
by
4. Verify this release using
following these procedur

注意，这里应该选择第二个，也就是（Scala 2.13）。之前在用第一个的时候出了一点bug，但用第二个就好使了

IDEA安装Scala插件

这一步很简单，就在 **设置→插件** 中搜索并下载scala即可，然后重新启动



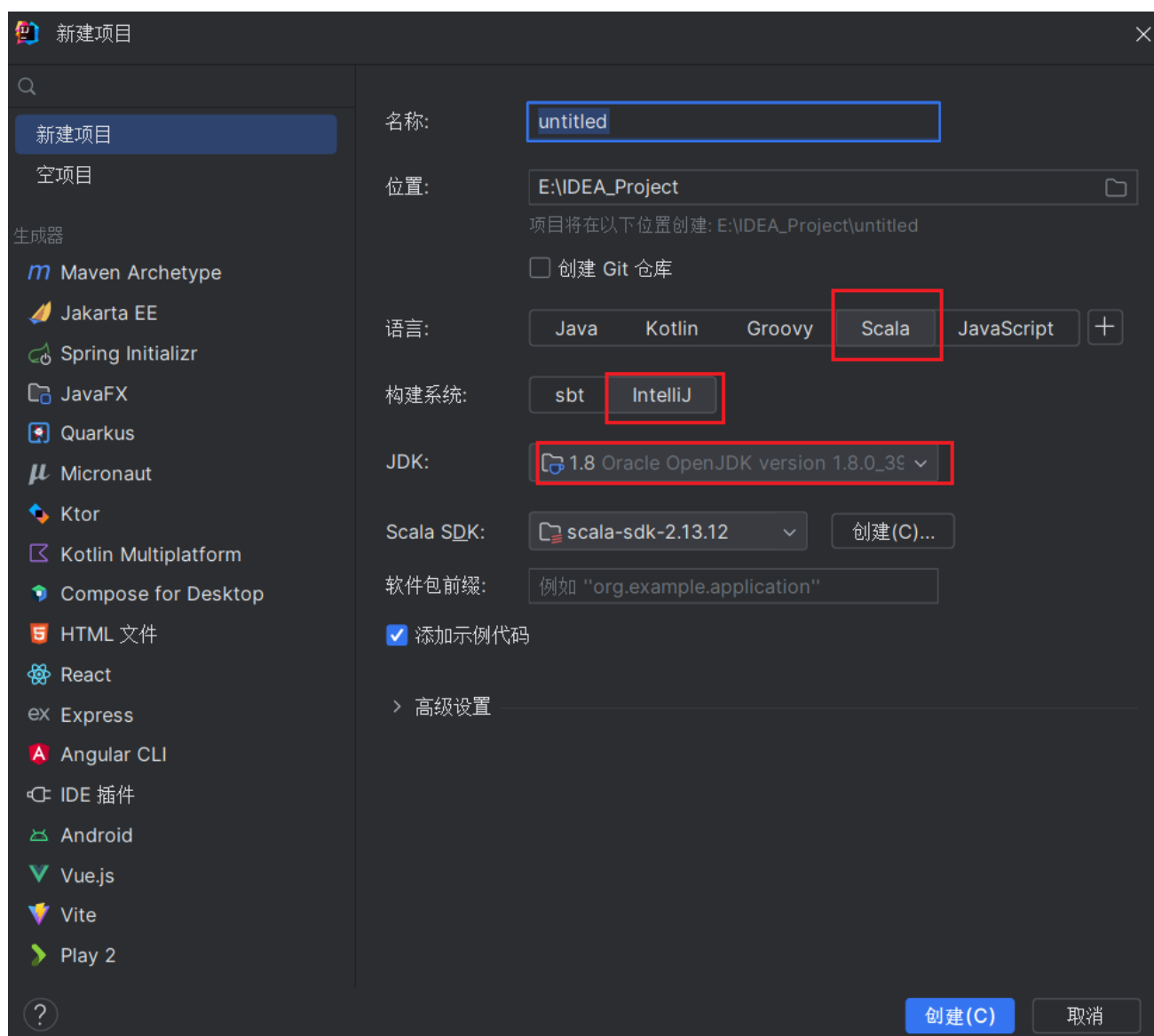
新建Scala项目

Spark运行需要java环境，因为之前已经配好了，所以直接导入即可

然后对于Scala SDK，没有下也没关系，直接在创建项目中下载即可。

注意对应顺序

	Version	Scala	Vulnerabilities	Repository	Usages	Date
3.5.x	3.5.0	2.13 2.12		Central	45	Sep 13, 2023



导入Spark的jar包

这一步也比较简单： 项目结构→库→+ 导入下载的Spark文件中的jar目录即可。

任务一

1. 统计application_data.csv中贷款金额AMT_CREDIT 的分布情况

思路

- 1. 首先，用 local[*] 模式创建一个 SparkSession 对象
- 2. 用 spark.read 方法创建一个DataFrameReader对象，读取application_data.csv，读取结果如下图所示：

```
23/12/19 22:52:24 INFO CodeGenerator: Code generated in 55.4724 ms
```

SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_EN
100002	Cash Loans	M	N	Y	0	202500	406597.5	Working	Secondary / secon...	Single / not married	House / apartment	0.018801	-9461	
100003	Cash Loans	F	N	N	0	270000	1293502.5	State servant	Higher education	Married	House / apartment	0.003541	-16765	
100004	Revolving Loans	M	Y	Y	0	67500	135000	Working	Secondary / secon...	Single / not married	House / apartment	0.010032	-19046	
100006	Cash Loans	F	N	Y	0	135000	312682.5	Working	Secondary / secon...	Civil marriage	House / apartment	0.008019	-19005	
100007	Cash Loans	M	N	Y	0	121500	513000	Working	Secondary / secon...	Single / not married	House / apartment	0.028663	-19932	
100008	Cash Loans	M	N	Y	0	99000	490495.5	State servant	Secondary / secon...	Married	House / apartment	0.035792	-16941	
100009	Cash Loans	F	Y	Y	1	171000	1560726	Commercial associate	Higher education	Married	House / apartment	0.035792	-13778	
100010	Cash Loans	M	Y	Y	0	360000	1530000	State servant	Higher education	Married	House / apartment	0.003122	-18850	
100011	Cash Loans	F	N	Y	0	112500	1019610	Pensioner	Secondary / secon...	Married	House / apartment	0.018634	-20099	

3. 根据AMT_CREDIT分组

使用 groupby 函数，并在 groupBy 中，使用 floor(col("AMT_CREDIT") / 10000) * 10000 表达式对 AMT_CREDIT 列的值进行处理，实现按照每一万为一个范围进行分组。

分组结果如下：全部结果可查看附件

credit_range	credit_range_upper	count
40000	50000	561
50000	60000	891
60000	70000	719
70000	80000	1226
80000	90000	668
90000	100000	1939
100000	110000	1871
110000	120000	1930
120000	130000	1323
130000	140000	4792
140000	150000	2239
150000	160000	3653
160000	170000	1919
170000	180000	2131
180000	190000	8745

全部代码

```

1  def Task1_1(df: DataFrame): DataFrame = {
2      // 将AMT_CREDIT列转换为Double类型
3      val dfWithAmount = df.withColumn("AMT_CREDIT",
4          col("AMT_CREDIT").cast("Double"))
5
6      // 统计 AMT_CREDIT 的分布情况
7      val creditDistribution = dfWithAmount.groupBy(
8          floor(col("AMT_CREDIT") / 10000) * 10000 as "credit_range"
9      )
10     .count()
11     .orderBy("credit_range")
12     //creditDistribution添加一列，计算每个区间的上界
13     val creditDistribution1 =
14         creditDistribution.withColumn("credit_range_upper", col("credit_range")+10000)
15
16     //count列和credit_range_upper列交换位置
17     val df1 =
18         creditDistribution1.select("credit_range", "credit_range_upper", "count")
19
20     // 显示结果

```

```

16     df1.show(false)
17     // 将其写入csv文件,模式设置为overwrite,保存列名
18     df1.write.mode(SaveMode.Overwrite).option("header",
19         "true").csv("E:\\IDEA_Project\\exp4\\src\\output\\Task1_1")
20 }

```

在Spark WebUI中查看

Spark 3.5.0 Jobs Stages Storage Environment Executors SQL / DataFrame Spark application UI

Spark Jobs (?)

User: Tony-x
Total Uptime: 51 s
Scheduling Mode: FIFO
Completed Jobs: 9

Event Timeline

Completed Jobs (9)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
8	csv at Main.scala:44 csv at Main.scala:44	2023/12/20 15:48:19	0.2 s	1/1 (2 skipped)	1/1 (17 skipped)
7	csv at Main.scala:44 csv at Main.scala:44	2023/12/20 15:48:19	0.5 s	1/1 (1 skipped)	1/1 (16 skipped)
6	csv at Main.scala:44 csv at Main.scala:44	2023/12/20 15:48:19	0.1 s	1/1 (1 skipped)	1/1 (16 skipped)
5	csv at Main.scala:44 csv at Main.scala:44	2023/12/20 15:48:16	2 s	1/1	16/16
4	show at Main.scala:42 show at Main.scala:42	2023/12/20 15:48:16	0.3 s	1/1 (1 skipped)	1/1 (16 skipped)
3	show at Main.scala:42 show at Main.scala:42	2023/12/20 15:48:13	3 s	1/1	16/16

Spark 3.5.0 Jobs Stages Storage Environment Executors SQL / DataFrame Spark application UI

Details for Job 8

Status: SUCCEEDED
Submitted: 2023/12/20 15:48:19
Duration: 0.2 s
Associated SQL Query: 3
Completed Stages: 1
Skipped Stages: 2

Event Timeline

DAG Visualization

Stage 11 (skipped): Scan csv, WholeStageCodegen (1), exchange

Stage 12 (skipped): AQEShuffleRead, WholeStageCodegen (2), exchange

Stage 13: AQEShuffleRead, WholeStageCodegen (3), WriteFiles

2. 统计application_data.csv中AMT_CREDIT-AMT_INCOME_TOTAL最高和最低的各十条记录

思路

本题也较为简单，主要流程如下：

1. 获取所需列，并计算 **差值** 列
2. dataframe排序，取最前面的10个和最后10个
3. 输出结果

SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	差值	
433294	Cash loans	4050000	405000	3645000.0	
210956	Cash loans	4031032.5	430650	3600382.5	
434170	Cash loans	4050000	450000	3600000.0	
315893	Cash loans	4027680	458550	3569130.0	
238431	Cash loans	3860019	292050	3567969.0	
240007	Cash loans	4050000	587250	3462750.0	
117337	Cash loans	4050000	760846.5	3289153.5	
120926	Cash loans	4050000	783000	3267000.0	
117085	Cash loans	3956274	749331	3206943.0	
228135	Cash loans	4050000	864900	3185100.0	
114967	Cash loans	562491	117000000	-1.16437509E8	
336147	Cash loans	675000	18000090	-1.732509E7	
385674	Cash loans	1400503.5	13500000	-1.20994965E7	
190160	Cash loans	1431531	9000000	-7568469.0	
252084	Cash loans	790830	6750000	-5959170.0	
337151	Cash loans	450000	4500000	-4050000.0	
317748	Cash loans	835380	4500000	-3664620.0	
310601	Cash loans	675000	3950059.5	-3275059.5	
432980	Cash loans	1755000	4500000	-2745000.0	
157471	Cash loans	953460	3600000	-2646540.0	

全部代码

```
1 def Task1_2(df: DataFrame): DataFrame = {
2     // 选取所需列
3     val df1 =
4         df.select("SK_ID_CURR", "NAME_CONTRACT_TYPE", "AMT_CREDIT", "AMT_INCOME_TOTAL")
5         // 添加一列: AMT_CREDIT-AMT_INCOME_TOTAL
6         val df2 = df1.withColumn("差值", col("AMT_CREDIT") - col("AMT_INCOME_TOTAL"))
```

```

6      // 选取差值最大和最小的10行
7      val df3 = df2.orderBy(col("差值").desc).limit(10)
8      val df4 = df2.orderBy(col("差值").asc).limit(10)
9      // 将两个DataFrame合并
10     val df5 = df3.union(df4)
11     df5.show(false)
12     // 保存结果
13     df5.write.mode(SaveMode.Overwrite).option("header",
14               "true").csv("E:\\IDEA_Project\\exp4\\src\\output\\Task1_2")
15     df5
16   }

```

在Spark WebUI中查看

Spark application UI

Spark Jobs (?)

User: Tony-x
Total Uptime: 30 s
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline

Completed Jobs (5)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

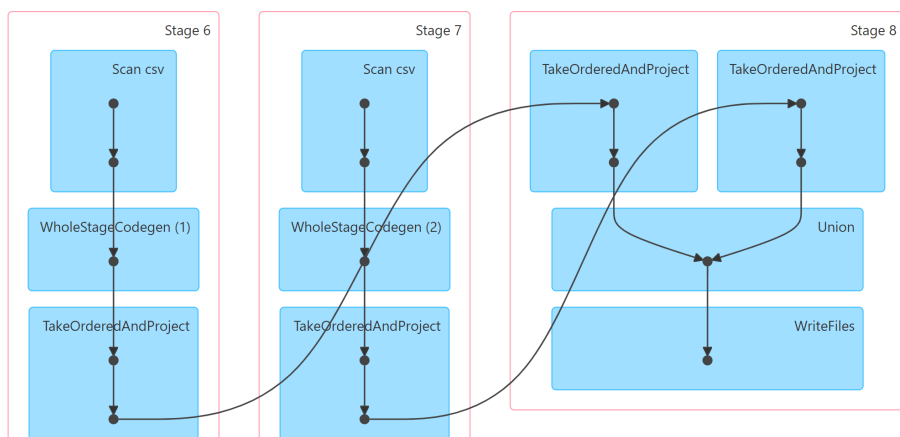
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	csv at Main.scala:61 csv at Main.scala:61	2023/12/20 15:51:06	1 s	3/3	34/34
3	show at Main.scala:59 show at Main.scala:59	2023/12/20 15:51:04	2 s	3/3	34/34
2	show at Main.scala:164 show at Main.scala:164	2023/12/20 15:51:03	0.4 s	1/1	1/1
1	load at Main.scala:22 load at Main.scala:22	2023/12/20 15:51:00	2 s	1/1	16/16
0	load at Main.scala:22 load at Main.scala:22	2023/12/20 15:50:59	0.7 s	1/1	1/1

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Details for Job 4

Status: SUCCEEDED
Submitted: 2023/12/20 15:51:06
Duration: 1 s
Associated SQL Query: 3
Completed Stages: 3

Event Timeline
DAG Visualization



任务二

1. 统计所有男性客户的小孩个数类型占比情况

思路

1. 筛选出所需的列，并用filter函数筛选出男性客户

```
23/12/20 00:38:30 INFO CodeGenerator: Code generated in 6.564701 ms
```

```
+-----+-----+
|CODE_GENDER|CNT_CHILDREN|
+-----+-----+
|M          |0           |
|M          |0           |
|M          |0           |
|M          |0           |
|M          |0           |
|M          |0           |
|M          |1           |
```

2. 根据 `CNT_CHILDREN` 使用 `groupby`，并用 `count()` 求出小孩个数的总数，升序排序

注意，读取的数据中 `CNT_CHILDREN` 默认类型是String，所以需要先转换成int类型再进行升序排序

```

+-----+-----+
|CNT_CHILDREN|count|
+-----+-----+
|0           |70318|
|1           |22660|
|2           |10413|
|3           |1446 |
|4           |170  |
|5           |33   |
|6           |11   |
|7           |4    |
|8           |1    |
|9           |1    |
|11          |1    |
|14          |1    |
+-----+-----+

```

3. 计算全部男性客户，并求出不同小孩数的占比

```

+-----+-----+-----+-----+
|CNT_CHILDREN|count|rate          |
+-----+-----+-----+-----+
|0           |70318|0.6693191444807204 |
|1           |22660|0.21568832751120798 |
|2           |10413|0.09911573496797038 |
|3           |1446 |0.013763694685843193 |
|4           |170  |0.0016181383793868207|
|5           |33   |3.1410921482214756E-4|
|6           |11   |1.0470307160738252E-4|
|7           |4    |3.807384422086637E-5 |
|8           |1    |9.518461055216593E-6 |
|9           |1    |9.518461055216593E-6 |
|11          |1    |9.518461055216593E-6 |
|14          |1    |9.518461055216593E-6 |
+-----+-----+-----+-----+

```

全部代码

```

1  def Task2_1(df: DataFrame): DataFrame = {
2      //统计所有男性客户 (CODE_GENDER = M) 的 小 孩个数 (CNT_CHILDREN) 类型占比情况
3      val df1 = df.select("CODE_GENDER", "CNT_CHILDREN")
4      //选取df1中CODE_GENDER = M的行
5      val df2 = df1.filter(col("CODE_GENDER") === "M")
6      df2.show(false)
7      //计算全部男性个数，也就是df2的行数
8      val male_count = df2.count()
9      //df2根据CNT_CHILDREN分组


```

```

10     val df3 = df2.groupBy("CNT_CHILDREN").count()
11     //CNT_CHILDREN转化成Int类型
12     val df4 =
13     df3.withColumn("CNT_CHILDREN", col("CNT_CHILDREN").cast("Int"))
14     //根据CNT_CHILDREN升序排列
15     val df5 = df4.orderBy(col("CNT_CHILDREN").asc)
16     df5.show(false)
17     //计算每个CNT_CHILDREN的占比
18     val df6 = df5.withColumn("rate", col("count")/male_count)
19     df6.show(false)
20     df6.write.mode(SaveMode.Overwrite).option("header",
21     "true").csv("E:\\IDEA_Project\\exp4\\src\\output\\Task2_1")
22 }

```

在Spark WebUI查看


3.5.0

[Jobs](#)
[Stages](#)
[Storage](#)
[Environment](#)
[Executors](#)
[SQL / DataFrame](#)

Spark application UI

Spark Jobs (?)

User: Tony-x
Total Uptime: 31 s
Scheduling Mode: FIFO
Completed Jobs: 14

[Event Timeline](#)
[Completed Jobs \(14\)](#)

Page:
 1 Pages. Jump to . Show items in a page.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
13	csv at Main.scala:83 csv at Main.scala:83	2023/12/20 21:11:03	0.3 s	1/1 (2 skipped)	1/1 (17 skipped)
12	csv at Main.scala:83 csv at Main.scala:83	2023/12/20 21:11:02	94 ms	1/1 (1 skipped)	1/1 (16 skipped)
11	csv at Main.scala:83 csv at Main.scala:83	2023/12/20 21:11:02	80 ms	1/1 (1 skipped)	1/1 (16 skipped)
10	csv at Main.scala:83 csv at Main.scala:83	2023/12/20 21:11:01	0.8 s	1/1	16/16
9	show at Main.scala:82 show at Main.scala:82	2023/12/20 21:11:01	79 ms	1/1 (1 skipped)	1/1 (16 skipped)
8	show at Main.scala:82 show at Main.scala:82	2023/12/20 21:11:00	1 s	1/1	16/16

[Event Timeline](#)
☐ Enable zooming

Executors

Added

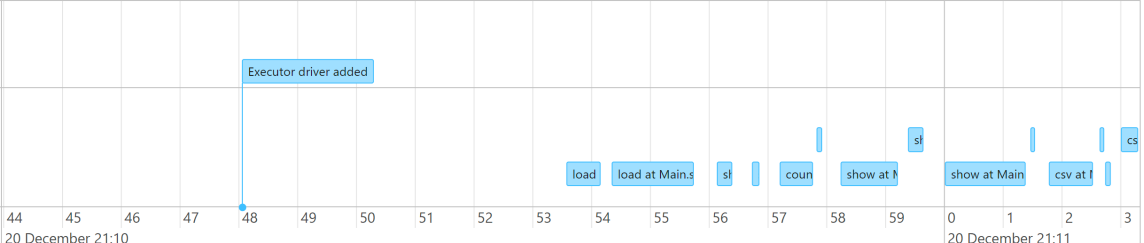
Removed

Jobs

Succeeded

Failed

Running



2. 统计每个客户每天的平均收入，并按照从大到小排序

思路

1. 选取相关列，并计算平均收入

$$avg_income = \frac{AMT_INCOME_TOTAL}{DAYS_BIRTH}$$

注意，从原始表格中读取的数据出生天数是负数，因此在计算的时候需要给出生天数乘以-1

2. 筛选每日平均收入大于1的客户并倒序排列

```
df2.filter(col("avg_income") > 1).orderBy(col("avg_income").desc)
即可实现
```

SK_ID_CURR	AMT_INCOME_TOTAL	DAYS_BIRTH	avg_income
114967	117000000	-12615	9274.673008323425
336147	18000090	-15704	1146.2105196128375
385674	13500000	-13551	996.2364401151207
190160	9000000	-16425	547.945205479452
219563	4500000	-10778	417.51716459454445
310601	3950059.5	-10572	373.63408059023834
157471	3600000	-9988	360.4325190228274
252084	6750000	-19341	348.9995346672871
199821	3375000	-12516	269.6548418024928
337151	4500000	-18461	243.75710958236283
141198	2025000	-8312	243.62367661212704
429258	3600000	-14897	241.65939450896153
196091	3375000	-14018	240.76187758596092

全部代码

```
1 def Task2_2(df: DataFrame): DataFrame = {
2     // 选取所需列
3     val df1 = df.select("SK_ID_CURR", "AMT_INCOME_TOTAL", "DAYS_BIRTH")
4     // 添加一列: AMT_INCOME_TOTAL/DAYS_BIRTH
5     val df2 = df1.withColumn("avg_income",
6                               col("AMT_INCOME_TOTAL").cast("Double") / (-
7                               col("DAYS_BIRTH").cast("Double")))
8     // 筛选出avg_income大于1的行并降序排列
9     val df3 = df2.filter(col("avg_income") >
10    1).orderBy(col("avg_income").desc)
11     df3.show(false)
12     df3
13 }
```

在Spark WebUI中查看

Spark 3.5.0

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

Spark application UI

Spark Jobs (?)

User: Tony-x
Total Uptime: 18 s
Scheduling Mode: FIFO
Completed Jobs: 4

▶ Event Timeline

▼ Completed Jobs (4)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Job Id ▼	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	show at Main.scala:93 show at Main.scala:93	2023/12/20 21:13:40	0.8 s	1/1	16/16
2	show at Main.scala:164 show at Main.scala:164	2023/12/20 21:13:39	0.2 s	1/1	1/1
1	load at Main.scala:22 load at Main.scala:22	2023/12/20 21:13:38	1 s	1/1	16/16
0	load at Main.scala:22 load at Main.scala:22	2023/12/20 21:13:37	0.5 s	1/1	1/1

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

任务三

根据给定的数据集，基于Spark MLlib 或者Spark ML编写程序对贷款是否违约进行分类，并评估实验结果的准确率。可以训练多个模型，比较模型的表现。

本实验中选出的特征有：

```
1 "CNT_CHILDREN", "AMT_INCOME_TOTAL", "AMT_CREDIT",
  "NAME_CONTRACT_TYPE_index", "DAYS_BIRTH", "DAYS_EMPLOYED",
  "DAYS_REGISTRATION", "DAYS_ID_PUBLISH",
  "CODE_GENDER_index", "FLAG_OWN_CAR_index", "FLAG_OWN_REALTY_index"
```

Step1: 数据预处理及特征提取

在本章节中，我们会将原数据进行数据预处理，并以此构造出特征向量。

1. 将String类型的数据编码为数值型数据

由于原数据中的许多数据是String类型的，无法直接作为特征，所以需要先将其编码，这里使用的是 `org.apache.spark.ml.feature.StringIndexer` 将其转换成数值数据。

```
1 // 将所需的string类型转换为数值类
2 val columnsToIndex = Array("NAME_CONTRACT_TYPE", "CODE_GENDER",
  "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "TARGET") // 列名数组
3 val columnsToIndex_output = Array("NAME_CONTRACT_TYPE_index",
  "CODE_GENDER_index", "FLAG_OWN_CAR_index", "FLAG_OWN_REALTY_index",
  "label") // 改变为数值型的数组
4 val indexedColumns = columnsToIndex.zip(columnsToIndex_output).map {
  case (colName, output) =>
5     // 遍历上述列，分别构造StringIndexer
6     new StringIndexer()
7       .setInputCol(colName)
8       .setOutputCol(s"${output}")
9   }
10 val indexedDF = indexedColumns.foldLeft(df) { (accDF, indexer) =>
11     indexer.fit(accDF).transform(accDF)
12 }
```

2. 将所有数值型数据向量化

在将String类型的数据编码成数值型数据后，所有的特征都是数值型数据。下面使用 Spark ML 中的 `VectorAssembler` 构建特征向量。在机器学习中，特征向量是用来描述样本特征的一种形式，它将各个特征组合成一个向量，供机器学习模型使用。

构造后数据如下图所示：

```
23/12/20 15:09:19 INFO CodeGenerator: Code generated in 5.9416 ms
+-----+-----+
|features|label|
+-----+-----+
|[0.0,202500.0,406597.5,0.0,-9461.0,-637.0,-3648.0,-2120.0,1.0,0.0,0.0]|1.0|
|[0.0,270000.0,1293502.5,0.0,-16765.0,-1188.0,-1186.0,-291.0,0.0,0.0,1.0]|0.0|
|[0.0,67500.0,135000.0,1.0,-19046.0,-225.0,-4260.0,-2531.0,1.0,1.0,0.0]|0.0|
```

Step2: 划分训练集和测试集

本阶段比较简单，使用下列语句即可。

```
1 | val Array(trainDF, testDF) = preprocessed_df.randomSplit(Array(0.8, 0.2))
```

Step3: 训练模型

由于有Spark ML的强大支持，只需要调包并设置参数即可。

```
1 | val dt = new DecisionTreeClassifier()
2 |   .setFeaturesCol("features")
3 |   .setLabelCol("label")
4 |   .setMaxBins(15)
5 |   .setImpurity("gini")
6 |   .setSeed(10)
7 | // 训练模型
8 | val dtModel = dt.fit(trainDF)
9 | // 在测试集上预测
10 | val testPredictions = dtModel.transform(testDF)
11 | testPredictions
```

- `.setMaxBins(15)`：指定最大的分箱数。在构建决策树时，数据会根据特征值的范围进行分箱处理。这里设置的最大分箱数为15。
- `.setImpurity("gini")`：指定用于分裂节点的不纯度度量方式。在决策树的构建中，"gini"表示使用基尼不纯度进行分裂，用来衡量数据的纯度。
- `.setSeed(10)`：设置随机种子，用于在构建决策树时引入随机性，有助于提高模型的泛化能力。

Step4: 评估模型效果

本阶段也比较简单，直接调用 `import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator` 即可。

```
1 | //计算测试集召回率
2 | val evaluator1 = new MulticlassClassificationEvaluator()
3 |   .setMetricName("weightedRecall")
4 | val recall = evaluator1.evaluate(testPredictions)
5 | //计算测试集精确率
```

```

6  val evaluator2 = new MulticlassClassificationEvaluator()
7    .setMetricName("weightedPrecision")
8  val precision = evaluator2.evaluate(testPredictions)
9  //计算测试集F1值
10 val evaluator3 = new MulticlassClassificationEvaluator()
11    .setMetricName("f1")
12 val f1 = evaluator3.evaluate(testPredictions)
13 print("recall   : " + recall+"\n")
14 print("precision: " + precision+"\n")
15 print("f1-score : " + f1+"\n")

```

可以看见，使用决策树的预测贷款是否违约的模型效果如下：

```

recall   : 0.9188987127811727
precision: 0.8443748443508963
f1-score : 0.8800619216916292
23/12/20 15:09:40 INFO SparkUI: Stopped Spark web UI at http://172.27.131.130:4040

```

全部代码：

定义的函数：

```

1  def Task3_get_preprocessed_df(df: DataFrame): DataFrame = {
2    // 将所需的string类型转换为数值类
3    val columnsToIndex = Array("NAME_CONTRACT_TYPE", "CODE_GENDER",
4      "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "TARGET") // 列名数组，你希望转换的列
5    val columnsToIndex_output = Array("NAME_CONTRACT_TYPE_index",
6      "CODE_GENDER_index", "FLAG_OWN_CAR_index", "FLAG_OWN_REALTY_index",
7      "label")
8    val indexedColumns = columnsToIndex.zip(columnsToIndex_output).map {
9      case (colName, output) =>
10        new StringIndexer()
11          .setInputCol(colName)
12          .setOutputCol(s"${output}")
13    }
14    val indexedDF = indexedColumns.foldLeft(df) { (accDF, indexer) =>
15      indexer.fit(accDF).transform(accDF)
16    }
17    // 构建特征向量
18    val assembler = new VectorAssembler()
19      .setInputCols(Array("CNT_CHILDREN", "AMT_INCOME_TOTAL",
20        "AMT_CREDIT", "NAME_CONTRACT_TYPE_index", "DAYS_BIRTH", "DAYS_EMPLOYED",
21        "DAYS_REGISTRATION", "DAYS_ID_PUBLISH", "CODE_GENDER_index",
22        "FLAG_OWN_CAR_index", "FLAG_OWN_REALTY_index"))
23    .setOutputCol("features")

```



```

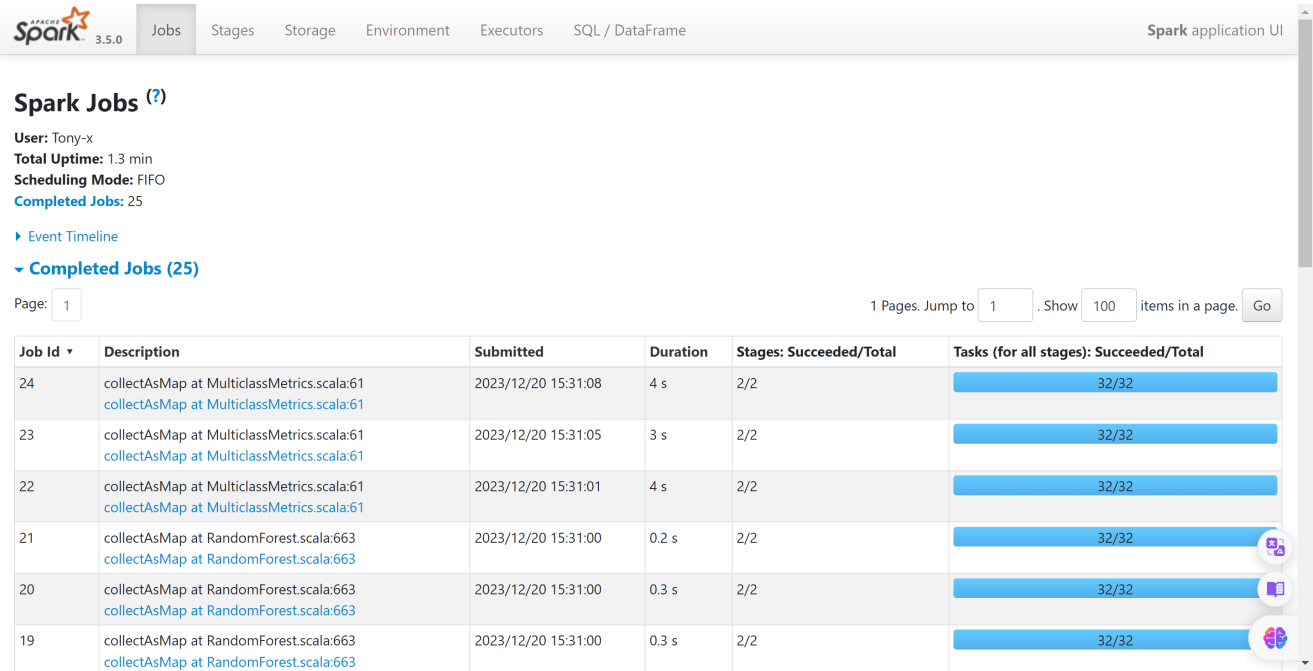
18     val assembledDF = assembler.transform(indexedDF)
19     //打印features和label列
20     assembledDF.select("features", "label").show(false)
21     assembledDF
22 }
23 def Model(trainDF: DataFrame, testDF: DataFrame): DataFrame = {
24     // 创建决策树模型
25     val dt = new DecisionTreeClassifier()
26         .setFeaturesCol("features")
27         .setLabelCol("label")
28         .setMaxBins(15)
29         .setImpurity("gini")
30         .setSeed(10)
31     //创建随机森林模型
32     // val rf = new RandomForestClassifier()
33     //     .setFeaturesCol("features")
34     //     .setLabelCol("label")
35     //     .setNumTrees(10)
36     //     .setMaxBins(10)
37     //     .setImpurity("gini")
38     //     .setSeed(10)
39
40     // 训练模型
41     val dtModel = dt.fit(trainDF)
42
43     // 在测试集上预测
44     val testPredictions = dtModel.transform(testDF)
45     testPredictions
46 }
47 def Evaluate(testPredictions: DataFrame): Unit = {
48     //计算测试集召回率
49     val evaluator1 = new MulticlassClassificationEvaluator()
50         .setMetricName("weightedRecall")
51     val recall = evaluator1.evaluate(testPredictions)
52     //计算测试集精确率
53     val evaluator2 = new MulticlassClassificationEvaluator()
54         .setMetricName("weightedPrecision")
55     val precision = evaluator2.evaluate(testPredictions)
56     //计算测试集F1值
57     val evaluator3 = new MulticlassClassificationEvaluator()
58         .setMetricName("f1")
59     val f1 = evaluator3.evaluate(testPredictions)
60     print("recall    : " + recall+"\n")
61     print("precision: " + precision+"\n")
62     print("f1-score  : " + f1+"\n")
63 }

```

主函数调用：

```
1 //Task3
2 // 数据预处理
3 val preprocessed_df=Task3_get_preprocessed_df(df)
4 // 随机划分训练集和测试集
5 val Array(trainDF, testDF) = preprocessed_df.randomSplit(Array(0.8, 0.2))
6
7 // 训练模型
8 val testPredictions = Model(trainDF,testDF)
9 // 评估模型
10 Evaluate(testPredictions)
```

在Spark WebUI中查看



遇到的困难及解决方法

1. 在保存数据时报错：`java.io.FileNotFoundException: HADOOP_HOME and hadoop.home.dir are unset.`

该问题可能是因为本次实验直接是在本机上跑的，本机上没有安装hadoop，在按照这篇文章安装hadoop和winutils后问题就解决了。文章[链接](#)

2. 关于DataFrame

在读取数据的时候，如果仅仅使用spark.read的话，会默认所有数据都是String类型，但如果加上`.option("header", "true")`，就可以让读取的时候自动判别数据类型，减少了后续类型转化工作

3. 关于Spark WebUI

如果用IDEA+Scala，程序在运行完就会立即关闭Spark端口，导致无法访问 `localhost:4040`，我们可以在程序运行完毕后等待一会 `Thread.sleep(1000000)`，就可以打开Spark端口

4. 关于一个dataframe在写入文件时出现多个csv文件

Spark DataFrame在写入文件时，默认会根据数据的分区（partitions）来生成对应数量的文件，每个分区的数据会被写入一个单独的文件中。这在分布式计算环境下是为了更好地利用集群资源，提高数据的并行处理能力。如果DataFrame有多个分区，写入时就会生成多个文件。

因此可以再写入时添加参数 `.coalesce(1)` 解决问题