

实验四

1. 环境选择与搭建

本实验的环境选择了Python3.10, Java-11, Pyspark进行操作。

因为之前的实验已经进行了Java环境的配置以及环境变量的设置, 在此处不再赘述Java的配置过程, 直接进行Pyspark的配置

1. scala安装

```
1 | wget https://downloads.lightbend.com/scala/2.11.12/scala-2.11.12.deb
2 | sudo dpkg -i scala-2.11.12.deb
3 | scala -version
```

输出如下:

```
hadoop@ubuntu:~$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
hadoop@ubuntu:~$
```

这说明scala已经配置成功。

2. py4j安装

```
1 | pip install py4j
```

3. spark安装

```
1 | cd /usr/local
2 | wget http://mirrors.tuna.tsinghua.edu.cn/apache/spark/spark-2.4.0/spark-2.12.0-bin-hadoop3.tgz
3 | tar -xvf spark-2.12.0-bin-hadoop3.tgz
4 | ##改名
5 | mv spark-2.12.0-bin-hadoop3 spark
```

4. 环境变量配置

```
1 vim ~/.bashrc
2 ## 添加以下语句
3 export SPARK_HOME=/usr/local/spark
4 export PATH=$PATH:$SPARK_HOME/bin
5 ## exit
6 ## environment variables play
7 source ~/.bashrc
```

5. 配置spark-env.sh

```
1 ## copy spark-env-template.sh
2 cp spark-env-template.sh spark-env.sh
3 vim spark-env.sh
4 ## add variables to connect spark with hadoop, java and python,hbase
5 export JAVA_HOME=/usr/lib/jvm/default-java
6 export HADOOP_HOME=/usr/local/hadoop
7 export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
8 export PATH="/usr/bin/python3:$PATH"
9 export PATH=$PATH:/usr/local/hadoop/sbin:/usr/local/hadoop/bin
10 export
11   PATH=$PATH:/usr/local/hadoop/sbin:/usr/local/hadoop/bin:/usr/local/hbase
    -2.5.6
11 SPARK_WORKER_MEMORY=4g
```

6. pyspark安装

```
1 #打开终端
2 pip install pyspark
```

4. 进入pyspark, 检查安装情况。

```
1 pyspark
```


接下来的任务均在vscode利用pyspark完成算法设计

2.任务一

- 1.编写Spark程序，统计application_data.csv中所有用户的贷款金额AMT_CREDIT 的分布情况。

思路设计

读取数据，利用pyspark的dataframe创建AMT_CREDIT的下界区间，然后对相同下界的数据进行groupby，在这个基础上对分组进行count的数量统计，最后使用pyspark.sql选取输出的列并计算上界，利用orderby进行排序，最后在输出上把上下界进行元组的合并一起输出即可。代码如下：

- 1.初始化一个Spark session

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, floor
3
4 # Initialize a Spark session
5 spark =
   SparkSession.builder.appName("CreditAmountDistribution").getOrCreate()
```

- 2.数据读取与下界列创建

```
1 # Load the data
2 df =
   spark.read.csv(r'/home/hadoop/Workspace/exp_3/input/application_data.csv'
   , header=True, inferSchema=True)
3 # Define the range step for loan amounts
4 range_step = 10000
5 # Create a new column for the loan amount range
6 df = df.withColumn('AMT_CREDIT_RANGE', (floor(col('AMT_CREDIT')) /
   range_step) * range_step))
```

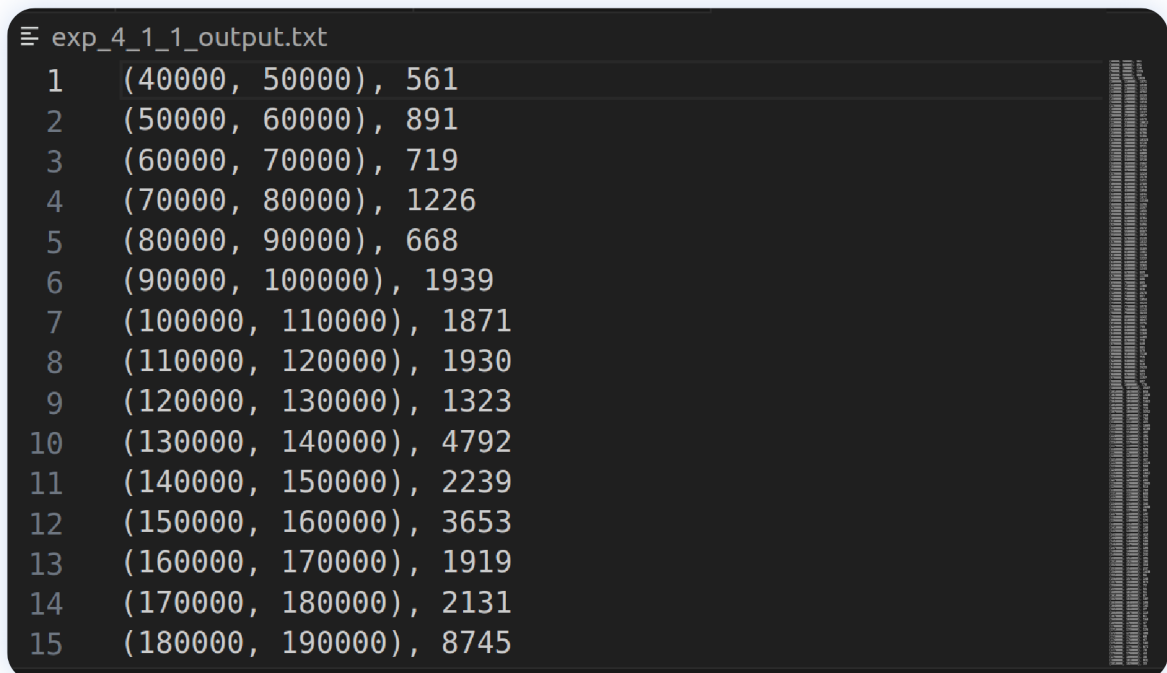
- 3.分组与输出列选取

```
1 # Group by the range and count the occurrences
2 distribution = df.groupBy('AMT_CREDIT_RANGE').count()
3
4 # Shift the range by one step for the upper bound
5 distribution = distribution.select(
6     col('AMT_CREDIT_RANGE').alias('RANGE_START'),
7     (col('AMT_CREDIT_RANGE') + range_step).alias('RANGE_END'),
8     col('count').alias('RECORD_COUNT')
9 ).orderBy('RANGE_START')
```

4. 打开文件并写入数据

```
1 # 指定输出文件路径
2 output_file_path = 'exp_4_1_1_output.txt'
3
4 # 打开文件并写入数据
5 with open(output_file_path, 'w') as file:
6     for row in distribution_list:
7         file.write(f"{{row['RANGE_START'], row['RANGE_END']}},
8         {{row['RECORD_COUNT']}}\n")
9
10 print(f"Data has been written to {output_file_path}")
11 # Stop the session
12 spark.stop()
```

5. 结果



```
exp_4_1_1_output.txt
1 (40000, 50000), 561
2 (50000, 60000), 891
3 (60000, 70000), 719
4 (70000, 80000), 1226
5 (80000, 90000), 668
6 (90000, 100000), 1939
7 (100000, 110000), 1871
8 (110000, 120000), 1930
9 (120000, 130000), 1323
10 (130000, 140000), 4792
11 (140000, 150000), 2239
12 (150000, 160000), 3653
13 (160000, 170000), 1919
14 (170000, 180000), 2131
15 (180000, 190000), 8745
```

2. 编写 Spark 程序，统计 application_data.csv 中客户贷款金额 AMT_CREDIT 比客户收入 AMT_INCOME_TOTAL 差值最高和最低的各十条记录。

思路设计

与上一题做法类似，选取所需要的列，对两列进行减法操作即可。需要注意的是，要对差值进行排序与前十行的选取进行输出。代码如下：

1. 初始化一个 Spark session

```

1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, floor, count
3 # Initialize a Spark session
4 spark = SparkSession.builder.appName("Minus").getOrCreate()

```

2. 数据读取与列选取

```

1 # Load the data
2 df =
    spark.read.csv(r'/home/hadoop/Workspace/exp_3/input/application_data.csv'
3 , header=True, inferSchema=True)
4 df =
    df.withColumn('SK_ID_CURR', col('SK_ID_CURR')).withColumn('AMT_CREDIT', col(
    'AMT_CREDIT')).withColumn('AMT_INCOME_TOTAL', col('AMT_INCOME_TOTAL'))

```

3. 用sql语句进行减法操作

```

1
2 distribution = df
3 # Shift the range by one step for the upper bound
4 distribution = distribution.select(
5     col('SK_ID_CURR').alias('SK_ID_CURR'),
6     col('NAME_CONTRACT_TYPE').alias('NAME_CONTRACT_TYPE'),
7     col('AMT_CREDIT').alias('AMT_CREDIT'),
8     col('AMT_INCOME_TOTAL').alias('AMT_INCOME_TOTAL'),
9     (col('AMT_CREDIT') - col('AMT_INCOME_TOTAL')).alias('MINUS'),
10    # col('count').alias('RECORD_COUNT')
11 ).orderBy(col('MINUS').desc())

```

4. 选取前十名和后十名，降序排序并文件输出

```

1 top10 = distribution.limit(10).collect()
2 bottom10 = distribution.orderBy(col('MINUS')).limit(10).collect()
3
4 # 指定输出文件路径
5 output_file_path = 'exp_4_1_2_output.txt'
6
7 # 打开文件并写入数据
8 with open(output_file_path, 'w') as file:
9     for row in top10:

```

```

10         file.write(f"{row['SK_ID_CURR']},
    row['NAME_CONTRACT_TYPE'],row['AMT_CREDIT'],row['AMT_INCOME_TOTAL']},
    {row['MINUS']}\n")
11     file.write("\n")
12     for row in bottom10:
13         file.write(f"{row['SK_ID_CURR']},
    row['NAME_CONTRACT_TYPE'],row['AMT_CREDIT'],row['AMT_INCOME_TOTAL']},
    {row['MINUS']}\n")
14
15     print(f"Data has been written to {output_file_path}")
16     # Stop the session
17     spark.stop()

```

5. 结果

```

(433294, 'Cash loans', 4050000.0, 405000.0), 3645000.0
(210956, 'Cash loans', 4031032.5, 430650.0), 3600382.5
(434170, 'Cash loans', 4050000.0, 450000.0), 3600000.0
(315893, 'Cash loans', 4027680.0, 458550.0), 3569130.0
(238431, 'Cash loans', 3860019.0, 292050.0), 3567969.0
(240007, 'Cash loans', 4050000.0, 587250.0), 3462750.0
(117337, 'Cash loans', 4050000.0, 760846.5), 3289153.5
(120926, 'Cash loans', 4050000.0, 783000.0), 3267000.0
(117085, 'Cash loans', 3956274.0, 749331.0), 3206943.0
(228135, 'Cash loans', 4050000.0, 864900.0), 3185100.0

(114967, 'Cash loans', 562491.0, 117000000.0), -116437509.0
(336147, 'Cash loans', 675000.0, 18000090.0), -17325090.0
(385674, 'Cash loans', 1400503.5, 13500000.0), -12099496.5
(190160, 'Cash loans', 1431531.0, 9000000.0), -7568469.0
(252084, 'Cash loans', 790830.0, 6750000.0), -5959170.0
(337151, 'Cash loans', 450000.0, 4500000.0), -4050000.0
(317748, 'Cash loans', 835380.0, 4500000.0), -3664620.0
(310601, 'Cash loans', 675000.0, 3950059.5), -3275059.5
(432980, 'Cash loans', 1755000.0, 4500000.0), -2745000.0
(157471, 'Cash loans', 953460.0, 3600000.0), -2646540.0

```

3. 任务二

基于Hive或者Spark SQL对application_data.csv进行如下统计：

1. 统计所有男性客户（CODE_GENDER=M）的小孩个数（CNT_CHILDREN）类型占比情况。输出格式为：

思路设计

同样的，读取数据后，先筛选出所有CODE_GENDER=='M'的数据，然后按照CNT_CHILDREN进行groupBy与数据统计，最后再计算百分比与文件输出。

1. 初始化一个Spark session

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, floor, count
3
4 # Initialize a Spark session
5 spark = SparkSession.builder.appName("exp_4_2_1").getOrCreate()
```

2. 数据读取与列选取

```
1 # Load the data
2 df =
    spark.read.csv(r'/home/hadoop/Workspace/exp_3/input/application_data.csv',
        header=True, inferSchema=True)
3
4 # Create a new column for the loan amount range
5 df = df.withColumn('CODE_GENDER',
    col('CODE_GENDER')).withColumn('CNT_CHILDREN', col('CNT_CHILDREN'))
```

3. 筛选男性数据并进行groupBy计数与百分比计算

```
1 # Filter for CODE_GENDER = 'm' and group by CNT_CHILDREN
2 distribution = df.filter(col('CODE_GENDER') =
    'M').groupBy(col('CNT_CHILDREN')).agg(count(col('CODE_GENDER')).alias('SUM_
    GENDER'))
3
4 # Calculate TYPE_RATIO using a subquery
5 distribution = distribution.select(
6     col('CNT_CHILDREN'),
7     (col('SUM_GENDER') / df.filter(col('CODE_GENDER') =
    'M').count()).cast('string').alias('TYPE_RATIO')
8 )
```

4. 文件输出


```

1 rows = distribution.collect()
2 # Write the results to a text file
3 output_file_path = 'exp_4_2_1_output.txt'
4 with open(output_file_path, 'w') as file:
5     for row in rows:
6         file.write(f"{row['CNT_CHILDREN'], row['TYPE_RATIO']}\n")
7 print(f"Data has been written to {output_file_path}")
8
9 # Stop the Spark session
10 spark.stop()

```

4. 结果

```

≡ exp_4_2_1_output.txt
1 (1, '0.21568832751120798')
2 (6, '1.0470307160738252E-4')
3 (3, '0.013763694685843193')
4 (5, '3.1410921482214756E-4')
5 (4, '0.0016181383793868207')
6 (8, '9.518461055216593E-6')
7 (7, '3.807384422086637E-5')
8 (11, '9.518461055216593E-6')
9 (2, '0.09911573496797038')
10 (0, '0.6693191444807204')
11 (14, '9.518461055216593E-6')
12 (9, '9.518461055216593E-6')
13

```

- 统计每个客户出生以来每天的平均收入(avg_income)=总收入 (AMT_INCOME_TOTAL) / 出生天数 (DAYS_BIRTH), 统计每日收入大于1的客户, 并按照从大到小排序, 保存为csv。输出格式为:

```

1 <SK_ID_CURR>, <avg_income>

```

思路设计

同样的，读取数据后，利用select计算平均收入，并用filter进行筛选，最后从大到小进行排序输出csv即可。

1. 初始化一个Spark session

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import col, floor, count
3
4 # Initialize a Spark session
5 spark = SparkSession.builder.appName("exp_4_2_2").getOrCreate()
```

2. 数据读取与列选取

```
1 # Load the data
2 df =
3     spark.read.csv(r'/home/hadoop/Workspace/exp_3/input/application_data.csv',
4                     header=True, inferSchema=True)
5
6 # Create a new column for the loan amount range
7 df =
8     df.withColumn('SK_ID_CURR', col('SK_ID_CURR')).withColumn('AMT_INCOME_TOTAL',
9                        col('AMT_INCOME_TOTAL')).withColumn('DAYS_BIRTH', col('DAYS_BIRTH'))
```

3. 筛选数据并排序

```
1 # Filter for CODE_GENDER = 'm' and group by CNT_CHILDREN
2 distribution = df.select(
3     col('SK_ID_CURR'),
4     (-col('AMT_INCOME_TOTAL')/col('DAYS_BIRTH')).alias('avg_income')
5 )
6
7 # Calculate TYPE_RATIO using a subquery
8 distribution =
9     distribution.filter(col('avg_income')>1).orderBy(col('avg_income').desc())
```

4. 文件输出

```

1 rows = distribution.collect()
2 # Write the results to a text file
3 output_file_path = 'exp_4_2_2_output.txt'
4 with open(output_file_path, 'w') as file:
5     for row in rows:
6         file.write(f"{row['CNT_CHILDREN'], row['TYPE_RATIO']}\n")
7 print(f"Data has been written to {output_file_path}")
8
9 # Stop the Spark session
10 spark.stop()

```

5. 结果

```

exp_4_2_2_output.csv
1 (114967, 9274.673008323425)
2 (336147, 1146.2105196128375)
3 (385674, 996.2364401151207)
4 (190160, 547.945205479452)
5 (219563, 417.51716459454445)
6 (310601, 373.63408059023834)
7 (157471, 360.4325190228274)
8 (252084, 348.9995346672871)
9 (199821, 269.6548418024928)
10 (337151, 243.75710958236283)
11 (141198, 243.62367661212704)
12 (429258, 241.65939450896153)
13 (196091, 240.76187758596092)
14 (317748, 240.44883783061715)
15 (432980, 239.56558773424192)
16 (217276, 235.32048408785298)
17 (445335, 234.30843510366373)

```

4. 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。

思路设计

利用pyspark.ml完成任务即可。注意数据预处理部分与实验二中的操作一致，此处不再赘述。

1. 读取训练集与测试集数据

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import split, col, mean, stddev
3 from tqdm import tqdm
4 from pyspark.ml.feature import VectorAssembler
5 from pyspark.ml.classification import DecisionTreeClassifier
6 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
7
8 # read in data
9 spark = SparkSession.builder.appName("DecisionTree").getOrCreate()
10 # df = spark.read.text("iris.data")
11 df_train = spark.read.csv(r'/home/hadoop/Workspace/exp_3/train.csv',
12                             header=True, inferSchema=True)
13 df_test =
14     spark.read.csv(r'/home/hadoop/Workspace/exp_3/test_original.csv',
15                     header=True, inferSchema=True)
```

2. 选取特征列

```
1 df_train = df_train.withColumn('AMT_INCOME_TOTAL',
2                                 col('AMT_INCOME_TOTAL'))\
3     .withColumn('NAME_EDUCATION_TYPE', col('NAME_EDUCATION_TYPE'))\
4     .withColumn('AMT_CREDIT', col('AMT_CREDIT'))\
5     .withColumn('NAME_HOUSING_TYPE', col('NAME_HOUSING_TYPE'))\
6     .withColumn('NAME_FAMILY_STATUS', col('NAME_FAMILY_STATUS'))\
7     .withColumn('NAME_INCOME_TYPE', col('NAME_INCOME_TYPE'))\
8     .withColumn('NAME_CONTRACT_TYPE', col('NAME_CONTRACT_TYPE'))\
9     .withColumn('REGION_POPULATION_RELATIVE', col('REGION_POPULATION_RELATIVE'))\
10     .withColumn('DAYS_BIRTH', col('DAYS_BIRTH'))\
11     .withColumn('DAYS_EMPLOYED', col('DAYS_EMPLOYED'))\
12     .withColumn('DAYS_REGISTRATION', col('DAYS_REGISTRATION'))\
13     .withColumn('DAYS_ID_PUBLISH', col('DAYS_ID_PUBLISH'))\
14     .withColumn('WEEKDAY_APPR_PROCESS_START', col('WEEKDAY_APPR_PROCESS_START'))\
15     .withColumn('HOUR_APPR_PROCESS_START', col('HOUR_APPR_PROCESS_START'))\
16     .withColumn('ORGANIZATION_TYPE', col('ORGANIZATION_TYPE'))\
17     .withColumn('TARGET', col('TARGET'))
```

```

18 df_test = df_test.withColumn('AMT_INCOME_TOTAL', col('AMT_INCOME_TOTAL'))\
19     .withColumn('NAME_EDUCATION_TYPE', col('NAME_EDUCATION_TYPE'))\
20     .withColumn('AMT_CREDIT', col('AMT_CREDIT'))\
21     .withColumn('NAME_HOUSING_TYPE', col('NAME_HOUSING_TYPE'))\
22     .withColumn('NAME_FAMILY_STATUS', col('NAME_FAMILY_STATUS'))\
23     .withColumn('NAME_INCOME_TYPE', col('NAME_INCOME_TYPE'))\
24     .withColumn('NAME_CONTRACT_TYPE', col('NAME_CONTRACT_TYPE'))\
25
26     .withColumn('REGION_POPULATION_RELATIVE', col('REGION_POPULATION_RELATIVE'))\
27
28     .withColumn('DAYS_BIRTH', col('DAYS_BIRTH'))\
29     .withColumn('DAYS_EMPLOYED', col('DAYS_EMPLOYED'))\
30     .withColumn('DAYS_REGISTRATION', col('DAYS_REGISTRATION'))\
31     .withColumn('DAYS_ID_PUBLISH', col('DAYS_ID_PUBLISH'))\
32
33     .withColumn('WEEKDAY_APPR_PROCESS_START', col('WEEKDAY_APPR_PROCESS_START'))\
34
35     .withColumn('HOUR_APPR_PROCESS_START', col('HOUR_APPR_PROCESS_START'))\
36     .withColumn('ORGANIZATION_TYPE', col('ORGANIZATION_TYPE'))\
37     .withColumn('TARGET', col('TARGET'))

```

3. 向量化特征

```

1 feature_cols = df_train.columns[:-1] # 假设最后一列是标签列
2 vector_assembler = VectorAssembler(inputCols=feature_cols,
3                                     outputCol="features")
4 train_data = vector_assembler.transform(df_train)
5 test_data = vector_assembler.transform(df_test)

```

4. 模型训练

```

1 bst = DecisionTreeClassifier(featuresCol="features",
2                               labelCol="TARGET", maxDepth=5)
3 bst_model = bst.fit(train_data)
4 predictions = bst_model.transform(test_data)

```

5. 分数评估

```
1 evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
2 labelCol="TARGET", metricName="f1")
3 # 计算 F1 分数
4 f1_score = evaluator.evaluate(predictions)
5 print("F1 Score:", f1_score)
6 evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",
7 labelCol="TARGET", metricName="accuracy")
8 accuracy = evaluator.evaluate(predictions)
9 print("accuracy:", accuracy)
```

6. 结果

5. 可能的改进之处

在测试过程中，发现precision会出错，这是因为数据存在bias，需要在处理中解决标签的分配问题。