金融大数据实验四

191870105 刘婷

完结撒花🎉

实验环境及语言：

1. IDEA JAVA
2. ubuntu + pycharm + pyspark
3. ubuntu + pycharm + pyspark
4. ubuntu + pycharm + pyspark

任务一

> 编写 MapReduce 程序，统计每个工作领域 industry 的网贷记录的数量，并按数量从大到小进行排序。
>
> 输出格式：<工作领域> <记录数量>

1.1 主要思路

本题使用两个job完成，第一个job负责数量统计，第二负责按照value排序

```java
// job1
Job solowcjob = Job.getInstance(conf,"solo wordcount");
solowcjob.setJarByClass(WordCount.class);
solowcjob.setMapperClass(SoloTokenizerMapper.class);
solowcjob.setCombinerClass(IntSumReducer.class);
solowcjob.setReducerClass(IntSumReducer.class);
solowcjob.setOutputKeyClass(Text.class);
solowcjob.setOutputValueClass(IntWritable.class);
solowcjob.setOutputFormatClass(SequenceFileOutputFormat.class);
FileInputFormat.addInputPath(solowcjob, new Path(otherArgs.get(0)));// otherArgs的第一个参数是输入路径
FileOutputFormat.setOutputPath(solowcjob,tempDir);

// job2
Job solosortjob = new Job(conf, "sort");
solosortjob.setJarByClass(WordCount.class);
FileInputFormat.addInputPath(solosortjob,tempDir);
solosortjob.setInputFormatClass(SequenceFileInputFormat.class);
solosortjob.setMapperClass(InverseMapper.class);
solosortjob.setReducerClass(SoloSortReducer.class);
FileOutputFormat.setOutputPath(solosortjob, new Path(otherArgs.get(1)));
solosortjob.setOutputKeyClass(IntWritable.class);
solosortjob.setOutputValueClass(Text.class);
```

```
//排序改写成降序
solosortjob.setSortComparatorClass(IntWritableDecreasingComparator.class);
```

1.2 job1：SoloTokenizerMapper + IntSumReducer

1.2.1 SoloTokenizerMapper.class

将csv文本去除表头后按行读取，用“,”进行分割，将linevalue[10]作为key，1作为value

输出`<linevalue[10], 1>`。

```
public void map(Object key, Text value, Context context)
  throws IOException, InterruptedException {
  String line = value.toString();
  String[] linevalue = line.split(",");
  word.set(linevalue[10]);
  context.write(word, one);
}
```

1.2.1 IntSumReducer.class

按照key对value进行求和，输出为`<key, sum>`

```
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
```

1.3 job2：InverseMapper.class + SoloSortReducer.class

1.3.1 InverseMapper.class

将<key,value>,转为<value,key>

1.3.2 SoloSortReducer.class

将按照value排好序<value,key>写为<key,value>，按照从大到小的顺序

```java
public void reduce(IntWritable key,Iterable<Text> values,Context context) throws
IOException, InterruptedException {
  for(Text val: values) {
  context.write(val,key);
  }
}
private static class IntWritableDecreasingComparator extends IntWritable.Comparator {
  public int compare(WritableComparable a, WritableComparable b) {
    return -super.compare(a, b);
  }
  public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
    return -super.compare(b1, s1, l1, b2, s2, l2);
  }
}
```

1.4 结果展示

```
金融业	48216
电力、热力生产供应业	36048
公共服务、社会组织	30262
住宿和餐饮业	26954
文化和体育业	24211
信息传输、软件和信息技术服务业	24078
建筑业	20788
房地产业	17990
交通运输、仓储和邮政业	15028
采矿业	14793
农、林、牧、渔业	14758
国际组织	9118
批发和零售业	8892
制造业	8864
```

任务二

> 编写 Spark 程序，统计网络信用贷产品记录数据中所有用户的贷款金额 total_loan 的分布情况。
>
> 以 1000 元为区间进行输出。输出格式示例：((2000,3000),1234)

2.1 主要思路

- 在map过程中，按行读取文件，选取所需要的total_loan： `s = x.split(",") s[2]` ，将total_loan转为区间作为key，1作为value
- reduce过程中，使用`reduceByKey(lambda x,y:x+y)`，统计不同区间出现次数

```
def map_func(x):
    s = x.split(",")
    total_loan = round(float(s[2]))
    intervalmin = (total_loan // 1000)*1000
    intervalmax = intervalmin + 1000
    interval = "("+str(intervalmin)+","+str(intervalmax)+")"  // 区间表示
    return (interval,1)
```

2.2 map + reduce过程。

```
lines = sc.textFile("train_data.csv").map(lambda x:map_func(x)).cache()
result = lines.reduceByKey(lambda x,y:x+y).collect()

with open("2.csv","w") as file:
    for i in result:
        file.write("%s%s,%f%s\n" % ("(",i[0],i[1],")"))
file.close()
```

2.3 结果展示

部分结果展示,所有结果位于2.csv

2.csv

```
((12000,13000),20513)
((6000,7000),15961)
((9000,10000),10458)
((21000,22000),5507)
((22000,23000),3544)
((17000,18000),4388)
((5000,6000),16514)
((11000,12000),7472)
((13000,14000),5928)
((24000,25000),8660)
((3000,4000),9317)
((25000,26000),8813)
((31000,32000),752)
((26000,27000),1604)
((32000,33000),1887)
((30000,31000),6864)
((19000,20000),4077)
((1000,2000),4043)
((33000,34000),865)
((34000,35000),587)
```

任务三

基于 Hive 或者 Spark SQL 对网络信用贷产品记录数据进行如下统计：

> 统计所有用户所在公司类型 employer_type 的数量分布占比情况。
>
> 输出成 CSV 格式的文件，输出内容格式为：<公司类型>,<类型占比>

3.1.1 主要思路

- 在RDD上，使用`transformation：map`按行读取csv文件，并`,`为划分依据将字符串进行划分作为key
- 在RDD数据集上使用`f(x)`选取题目所需要的列使用`.toDF()`将rdd转为dataframe
- 创建视图loan

```python
def f(x):
    rel = {}
    rel['loan_id']=x[0]
    rel['employment_type']=x[9]
    return rel
loanDF = sc.textFile('train_data.csv').map(lambda line:line.split(',')).map(lambda
x:Row(**f(x))).toDF()
loanDF.createOrReplaceTempView("loan")
```

- 使用sql语句，统计employer_type的占比，
- 在执行完sql语句的dataframe选取题目所需要的`<公司类型>,<类型占比>`
- 按照格式要求输出。

```python
sql = "SELECT employment_type,count(loan_id)/ \
          (select count(loan_id) as all_count \
           from loan )         \
           as count1 \
       From loan \
       GROUP BY employment_type"

loanDF = spark.sql(sql)
loanDF.select("employment_type","count1").write.format("csv").save("3_1.csv")
```

3.1.2 结果展示

文件为3_1.csv

3 > 3_1.csv > part-00000-fbdf77ac-5d42-4b74-81dc-1e3b76e4eb02-c000.csv

```
幼教与中小学校,0.09998333333333333
上市企业,0.10012666666666667
政府机构,0.25815333333333335
世界五百强,0.05370666666666666
高等教育机构,0.03368666666666666
普通企业,0.45434333333333333
```

> 统计每个用户最终须缴纳的利息金额：
>
> 输出成 CSV 格式的文件，输出内容格式为：<user_id>,<total_money>

3.2.1 主要思路

- 在RDD上，使用`transformation: map`按行读取csv文件，并`,`为划分依据将字符串进行划分作为key
- 在RDD数据集上使用`f(x)`选取题目所需要的列使用`.toDF()`将rdd转为dataframe
- 创建视图loan

```python
def f(x):
    rel = {}
    rel['loan_id']=x[0]
    rel['user_id'] = x[1]
    rel['year_of_loan']=x[3]
    rel['monthly_payment']=x[5]
    rel['total_loan']=x[2]
    return rel
loanDF = sc.textFile('train_data.csv').map(lambda line:line.split(',')).map(lambda
x:Row(**f(x))).toDF()
loanDF.createOrReplaceTempView("loan")
```

- 使用sql语句，统计每个用户最终须缴纳的利息金额
- 在执行完sql语句的dataframe选取题目所需要的`<user_id>,<total_money>`
- 按照格式要求输出。

```python
sql2 = "SELECT user_id,year_of_loan*monthly_payment*12 - total_loan as total_money  \
        FROM loan "
loanDF = spark.sql(sql2)
loanDF.select("user_id","total_money").write.format("csv").save("3_2.csv")
```

3.2.2 结果展示

部分结果展示,所有结果位于3_2.csv

3 > 3_2.csv > part-00000-6305d524-4929-4e8b-b2c3-6f3228c55f09-c000.csv

```
0,3846.0
1,1840.6000000000004
2,10465.600000000002
3,1758.5200000000004
4,1056.880000000001
5,7234.639999999999
6,757.9200000000001
7,4186.959999999999
8,2030.7600000000002
9,378.72000000000116
10,4066.760000000002
```

> 统计工作年限 work_year 超过 5 年的用户的房贷情况 censor_status 的数量分布占比情况。
>
> 输出成 CSV 格式的文件，输出内容格式为：<user_id>,<censor_status>,<work_year>

3.3.1 主要思路

- 在RDD上，使用`transformation: map`按行读取csv文件，并`,`为划分依据将字符串进行划分作为key
- 在RDD数据集上使用`f(x)`选取题目所需要的列使用`.toDF()`将rdd转为dataframe
  - 这里将`xx years`转为`int`型数据
    - 转化规则是：

      <1 years : workyear = 0

      2-9 years : workyear = 2-9

      >10 years : worker = 10
- 创建视图loan

```
def f(x):
    rel = {}
    rel['loan_id']=x[0]
    rel['user_id'] = x[1]
    workyear = 0
    if x[11]=="":
        workyear = 0
    else:
        year = x[11].split(" ",1)
        if year[0] == "10+":
            workyear = 10
        elif year[0][0] == "<":
            workyear = 0
        else:
            workyear = int(year[0])
    rel['work_year']=workyear
    rel['censor_status']=x[14]
    return rel
loanDF = sc.textFile('train_data.csv').map(lambda line:line.split(',')).map(lambda
x:Row(**f(x))).toDF()
loanDF.createOrReplaceTempView("loan")
```

- 使用sql语句，统计每个用户最终须缴纳的利息金额
- 在执行完sql语句的dataframe选取题目所需要的`<user_id>,<censor_status>,<work_year>`
- 按照格式要求输出。

```
sql3 = "SELECT user_id,censor_status,work_year F\
        ROM loan \
        Where work_year>5"
loanDF = spark.sql(sql3)
loanDF.select("user_id","censor_status","work_year").write.format("csv").save("3_3.csv"
)
```

3.3.2 结果展示

部分结果展示,所有结果位于3_3.csv

3 > 3_3.csv > part-00000-929108df-7b9f-4900-813a-52db9f85d6e2-c000.csv

```
1,2,10
2,1,10
5,2,10
6,0,8
7,2,10
9,0,10
10,2,10
15,1,7
16,2,10
17,0,10
18,1,10
```

任务四

> 根据给定的数据集，基于 Spark MLlib 或者Spark ML编写程序预测有可能违约的借贷人，并评估实验结果的准确率。

4.1 读取数据

设置属性`inferSchema=True`，pyspark根据读取到的数据形式推断数据的类型。

```
spark=SparkSession.builder.appName("4").getOrCreate()
df_train = spark.read.csv("test/train_data.csv",header='true',inferSchema='true')
```

4.2 数据预处理

（1）缺失值以-1填充

```
df_train = df_train.na.fill(-1)
df_train = df_train.na.fill("-1")
```

（2）无差别类别数据：将String型类别数据，先StringIndexer转为indexer，再用OneHotEncoder转为onehot编码。`work_type`，`employer_type`，`industry`

```
# work_type
work_type_stringIndexer =
StringIndexer(inputCol="work_type",outputCol="work_type_class",stringOrderType="frequencyDesc")
```

```
df_train = work_type_stringIndexer.fit(df_train).transform(df_train)
work_type_encoder =
OneHotEncoder(inputCol="work_type_class",outputCol="work_type_onehot").setDropLast(Fals
e)
df_train = work_type_encoder.fit(df_train).transform(df_train)

#employer_type
employer_type_stringIndexer =
StringIndexer(inputCol="employer_type",outputCol="employer_type_class",stringOrderType=
"frequencyDesc")
df_train = employer_type_stringIndexer.fit(df_train).transform(df_train)
employer_type_encoder =
OneHotEncoder(inputCol="employer_type_class",outputCol="employer_type_onehot").setDropL
ast(False)
df_train = employer_type_encoder.fit(df_train).transform(df_train)

# industry
industry_stringIndexer =
StringIndexer(inputCol="industry",outputCol="industry_class",stringOrderType="frequency
Desc")
df_train = industry_stringIndexer.fit(df_train).transform(df_train)
industry_encoder =
OneHotEncoder(inputCol="industry_class",outputCol="industry_onehot").setDropLast(False)
df_train = industry_encoder.fit(df_train).transform(df_train)
```

（3）有数值意义的数据：将String型数值型数据，转为int型。`work_year`，`class`，`sub_class`

```
@f.udf(returnType = IntegerType())  ## spark.sql 需要句柄
def work_year(x):
    workyear = 0
    if x:
        year = str(x).split(" ",1)
        if year[0] == "10+":
            workyear = 10
        elif year[0][0] == "<":
            workyear = 0
        else:
            workyear = int(year[0])
    return workyear
df_train= df_train.withColumn("work_year",work_year(f.col("work_year")))
```

```
#class
class_stringIndexer = StringIndexer(inputCol="class",outputCol="class_class")
df_train = class_stringIndexer.fit(df_train).transform(df_train)

#sub_class
subclass_stringIndexer = StringIndexer(inputCol="sub_class",outputCol="subclass_class")
df_train = subclass_stringIndexer.fit(df_train).transform(df_train)
```

（4）日期数据：使用datetime库将日期数据转为离最小的日期的月数（考虑到本题中日期最小间隔为月份）。`issue_date`,`earlies_credit_mon`

```
#issue_data
@f.udf(returnType = IntegerType())
def issuedata(x):
    time = 0
    if x == "-1":
        time = 0
    else:
        timeString = x.split("-")
        year = int(timeString[0])
        month = int(timeString[1])
        day = int(timeString[2])
        time1 = datetime(2007, 7, 1)
        time2 = datetime(year, month, day)
        time = (time2-time1).days//30
    return time
df_train= df_train.withColumn("issue_date",issuedata(f.col("issue_date")))
```

（5）将预处理后的数据保存文件到本地，方便后续使用。

（6）进行特征集成，将所有特征合并到一个数组feature中：

```
featuresArray = ['total_loan', 'year_of_loan', 'interest', 'monthly_payment',\
               'class_class','subclass_class','work_type_onehot','work_year',\
               'employer_type_onehot','industry_onehot','issue_date',\
               'house_exist','house_loan_status',\
               'censor_status','marriage','offsprings','use','post_code',\
               'region','debt_loan_ratio','del_in_18month','scoring_low',\
               'scoring_high','pub_dero_bankrup','early_return','early_return_amount',\

'early_return_amount_3mon','recircle_b','recircle_u','initial_list_status',\
               'title','policy_code','f0','f1','f2','f3','f4','f5']
assembler = VectorAssembler().setInputCols(featuresArray).setOutputCol("features")
df_train= assembler.transform(df_train)
```

（7）使用Standard Sclarizer将特征向量标准化：

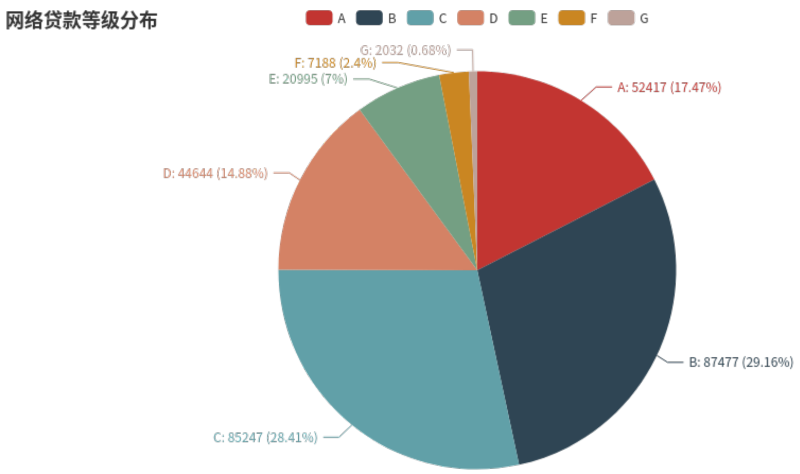
```
scaler = StandardScaler(inputCol="features", outputCol="features_scaled",
withMean=True, withStd=True)
df_train = scaler.fit(df_train).transform(df_train)
```

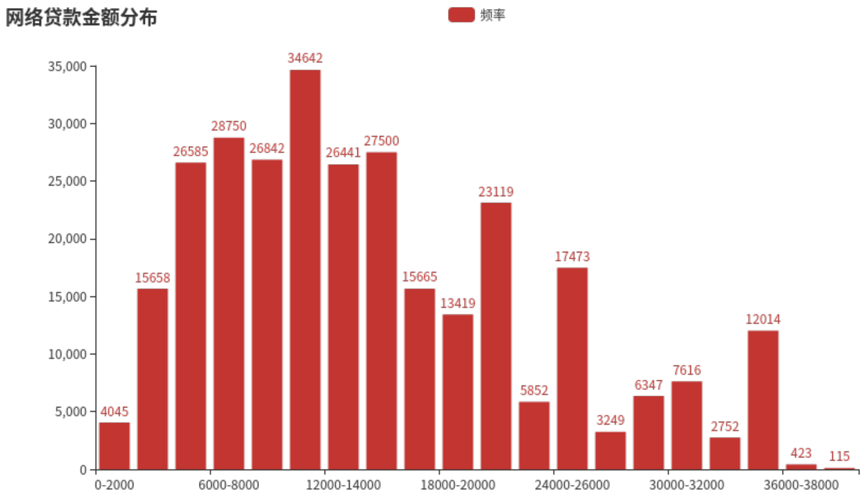
（8）按照8：2的比例划分训练集和测试集：

```
trainingData, testData = df_train.randomSplit([0.8,0.2])
```

4.3 画图查看部分数据特征

绘制网络贷款等级分布、网络贷款金额分布等图






```
total_y = []
attr = []
for i in range(7):
    attr.append(chr(ord('A')+i))
    total_y.append(df_train.filter(df_train['class'] == attr[i]).count())

pie = (
    Pie()
        .add("网络贷款等级", [list(z) for z in zip(attr, total_y)])
        .set_global_opts(title_opts=opts.TitleOpts(title="网络贷款等级分布"))
        .set_series_opts(
        tooltip_opts=opts.TooltipOpts(trigger="item", formatter="{a} <br/>{b}: {c}
({d}%)"),
        label_opts=opts.LabelOpts(formatter="{b}: {c} ({d}%)")
    )
)

attr = ["0-2000", "2000-4000", "4000-6000","6000-8000","8000-10000",
```

```
        "10000-12000","12000-14000","14000-16000","16000-18000","18000-20000",
        "20000-22000","22000-24000","24000-26000","26000-28000","28000-30000",
        "30000-32000","32000-34000","34000-36000","36000-38000","38000-40000"]
money_list = []

for i in range(20):
    money_list.append(df_train.filter((df_train['total_loan'] < (i+1)*2000) &
(df_train['total_loan'] >= i*2000)).count())
bar = (
    Bar()
        .add_xaxis(attr)
        .add_yaxis("频率", money_list)
        .set_global_opts(title_opts=opts.TitleOpts(title="网络贷款金额分布"))
)
```

4.4 二分类评价指标计算函数

```
def eva_index(Predictions):
    # 使用混淆矩阵评估模型性能[[TP,FN],[TN,FP]]
    print("-------------------------------------------------------------------")
    print(str(Predictions))
    TP = Predictions.filter(Predictions['prediction'] ==
1).filter(Predictions['is_default'] == 1).count()
    FN = Predictions.filter(Predictions['prediction'] ==
0).filter(Predictions['is_default'] == 1).count()
    TN = Predictions.filter(Predictions['prediction'] ==
0).filter(Predictions['is_default'] == 0).count()
    FP = Predictions.filter(Predictions['prediction'] ==
1).filter(Predictions['is_default'] == 0).count()
    # 计算查准率 TP/ (TP+FP)
    precision = TP/(TP+FP)
    # 计算查全率 TP/ (TP+FN)
    recall = TP/(TP+FN)
    # 计算F1值
    F1 =(2 * precision * recall)/(precision + recall)
    # 计算准确率
    acc = (TP + TN) / (TP + FN + TN + FP)
    print("The 查准率 is :",precision)
    print("The 查全率 is :",recall)
    print('The F1 is :',F1)
    print('The 准确率 s :', acc)
    # AUC为roc曲线下的面积，AUC越接近与1.0说明检测方法的真实性越高
    auc = BinaryClassificationEvaluator(labelCol="is_default").evaluate(Predictions)
    print("The auc分数 is :",auc)
```

4.5 逻辑斯蒂回归分类

```
lr =
LogisticRegression().setLabelCol("is_default").setFeaturesCol("features_scaled").setMax
Iter(10).setRegParam(0.01).\
    setElasticNetParam(0.8).fit(trainingData)
lrPredictions = lr.transform(testData)
eva_index(lrPredictions)
```

正例与反例权重相同

The 查准率 is : 0.7022688356164384

The 查全率 is : 0.2715385252006952

The F1 is : 0.3916442852879738

The 准确率 s : 0.8298245321134614

The auc分数 is : 0.8424096102126215

4.6 支持向量机分类

```
svm =
LinearSVC(maxIter=100,labelCol="is_default",featuresCol="features_scaled").fit(training
Data)
svmPredictions = svm.transform(testData)
eva_index(svmPredictions)
```

正例与反例权重相同

The 查准率 is : 0.6916340599962735

The 查全率 is : 0.3072084747165439

The F1 is : 0.42544412607449855

The 准确率 s : 0.832612651718784

The auc分数 is : 0.847274674473519

4.7 决策树分类

```
dt =
DecisionTreeClassifier(labelCol="is_default",featuresCol="features_scaled").fit(trainin
gData)
dtPredictions = dt.transform(testData)
eva_index(dtPredictions)
```

正例与反例权重相同

The 查准率 is : 0.6268525311812179

The 查全率 is : 0.35355458081602253

The F1 is : 0.4521113345327548

The 准确率 s : 0.8271365844700068

The auc分数 is : 0.6568277818290477

4.8 随机森林分类

```
rf =
RandomForestClassifier(labelCol="is_default",featuresCol="features_scaled",maxBins=700,
numTrees=50).fit(trainingData)
rfPredictions = rf.transform(testData)
eva_index(rfPredictions)
```

The 查准率 is : 0.8282208588957055

The 查全率 is : 0.011172722006124307

The F1 is : 0.022048015678588925

The 准确率 s : 0.8000567641117251

The auc分数 is : 0.8274599427203987

4.9 结果对比分析

当正例和反例权重相同，数据量不同时：

- 虽然以上算法**查准率都比较高**，即预测为违约的人中，确实违约的人比例较高。
- 但是几个算法的**查全率都比较低**，即在确实违约的人中，被查出来违约的人很少。
- 就贷款而言，不良贷款率是直接影响银行经营状况的指标，即希望算法能提前识别出当前用户是否会违约，如果违约可能性很大，宁愿不贷款，也不会冒险。因此**对查全率要求较高**，所以我们需要设置**二分类代价矩阵**。

	预测类别	
真实类别	未违约	违约
未违约	0	y
违约	x	0

- 解决方法：设置权重，这里仅需要控制比值，相同比值会有相同效果

x： y = 4:1

```
from pyspark.sql.functions import when
trainingData = trainingData.withColumn("classWeights",when(trainingData.is_default ==
1,0.8).otherwise(0.2))
```

在四个算法中分别添加`weightCol="classWeights"`后可以得到

逻辑斯蒂回归

The 查准率 is : 0.4290086493679308

The 查全率 is : 0.806672226855713

The F1 is : 0.5601274069784277

The 准确率 s : 0.7464109241452992

The auc分数 is : 0.8477962140118782

----

svm

The 查准率 is : 0.4308384968573084

The 查全率 is : 0.8060884070058382

The F1 is : 0.5615431542863782

The 准确率 s : 0.748046875

The auc分数 is : 0.8522481027743516

决策树

The 查准率 is : 0.38212282255683494

The 查全率 is : 0.8635529608006672

The F1 is : 0.5298060686690885

The 准确率 s : 0.6932091346153846

The auc分数 is : 0.7681579665443818

随机森林

The 查准率 is : 0.4363719651855245

The 查全率 is : 0.7944954128440367

The F1 is : 0.5633353045535187

The 准确率 s : 0.7534722222222222

The auc分数 is : 0.8494405252697155

- 可以看到查全率有了大幅度的提升，auc分数基本上都有小幅度的提升，准确率变化不大，基本维持在80%左右，F1因为查全率的大幅提升也有了显著提高，虽然查准率有了大幅度的下降，但是权重的设置部分解决了由类别不平衡带来的问题，也证明需要高查全率的应用场景设置权重是有效的。

x：y = 2:1

----

lr

The 查准率 is : 0.5222465353756383

The 查全率 is : 0.6657650042265427

The F1 is : 0.5853368511017799

The 准确率 s : 0.8129253981559095

The auc分数 is : 0.849834836045144

----

svm

The 查准率 is : 0.5054364332138735

The 查全率 is : 0.7033812341504649

The F1 is : 0.5882020287703673

The 准确率 s : 0.8046772841575859

The auc分数 is : 0.8542217416467005

----

dt

The 查准率 is : 0.5357855262108034

The 查全率 is : 0.5676246830092984

The F1 is : 0.5512457414932478

The 准确率 s : 0.8167141659681475

The auc分数 is : 0.751555822005103

----

rf

The 查准率 is : 0.5844083526682134

The 查全率 is : 0.53229078613694

The F1 is : 0.5571333775713339

The 准确率 s : 0.8321709974853311

The auc分数 is : 0.8405251704670197

- 和预想的相同，和`x: y = 4 : 1`相比查准率上升，查全率下降，F1稍有提升，精度基本无变化，auc稍有提升。说明x：y = 2:1该情况能均衡查准率和查全率，并且有较好的精度。

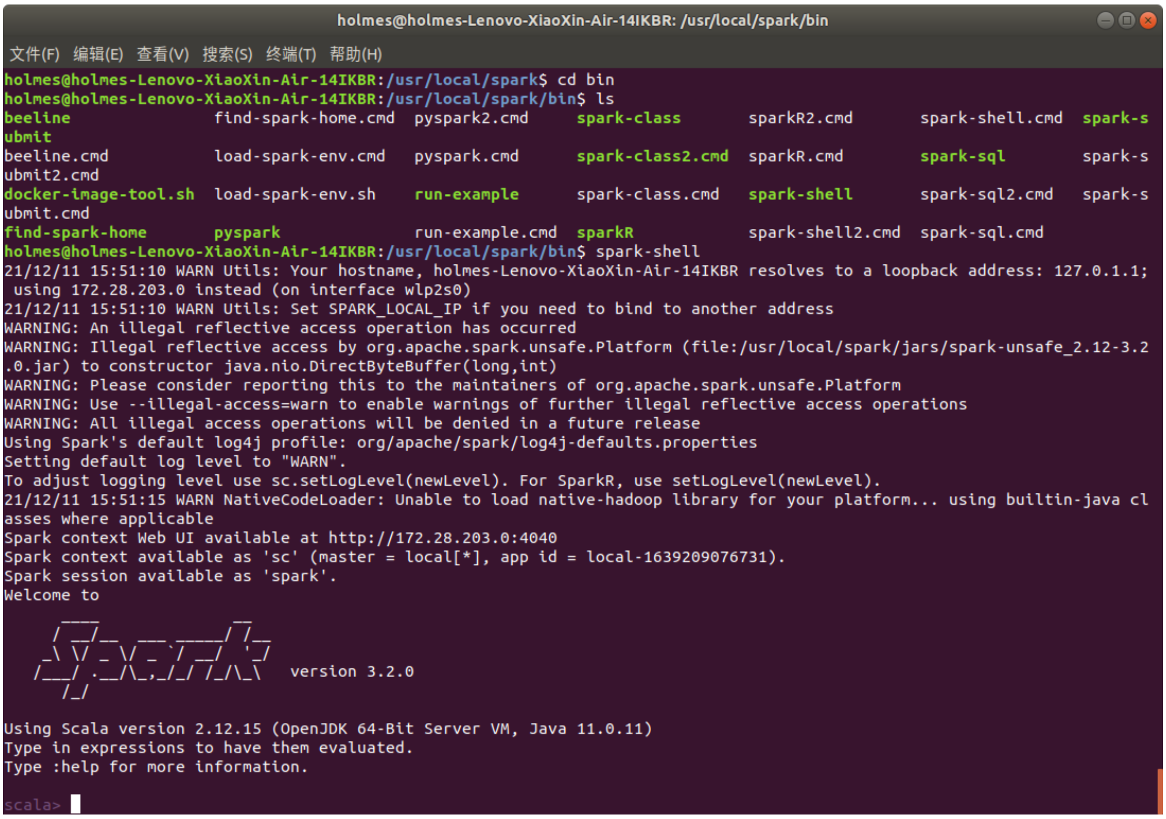
4.10 思考与改进

1. 可以根据不同的应用场景选取不同的指标
2. 针对不同的指标可以有不同的改进方法，比如该题想要提高查全率则需要设置**二分类代价矩阵**
3. 可以针对不同的贷款类型进行预测，将数据集按照class划分，训练出不同参数的模型，对相应的贷款类型的数据进行预测。

pycharm配置pyspark环境：

1. 官网安装pycharm
2. 官网下载spark，解压到 /usr/local中

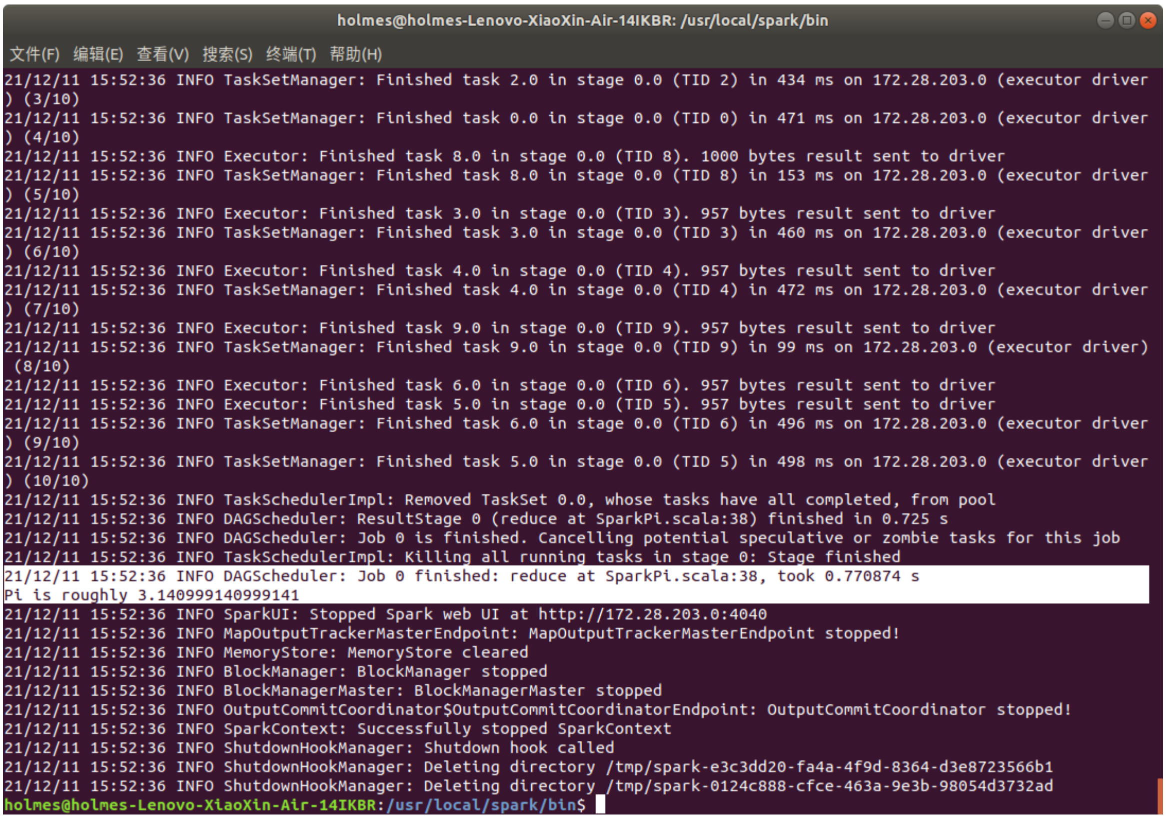
   spark打开及运行成功截图



```
holmes@holmes-Lenovo-XiaoXin-Air-14IKBR: /usr/local/spark/bin
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
holmes@holmes-Lenovo-XiaoXin-Air-14IKBR:/usr/local/spark$ cd bin
holmes@holmes-Lenovo-XiaoXin-Air-14IKBR:/usr/local/spark/bin$ ls
beeline               find-spark-home.cmd  pyspark2.cmd      spark-class       sparkR2.cmd       spark-shell.cmd  spark-submit
beeline.cmd           load-spark-env.cmd   pyspark.cmd       spark-class2.cmd  sparkR.cmd        spark-sql        spark-submit2.cmd
docker-image-tool.sh  load-spark-env.sh    run-example       spark-class.cmd   spark-shell       spark-sql2.cmd   spark-submit.cmd
find-spark-home       pyspark              run-example.cmd   sparkR            spark-shell2.cmd  spark-sql.cmd
holmes@holmes-Lenovo-XiaoXin-Air-14IKBR:/usr/local/spark/bin$ spark-shell
21/12/11 15:51:10 WARN Utils: Your hostname, holmes-Lenovo-XiaoXin-Air-14IKBR resolves to a loopback address: 127.0.1.1; using 172.28.203.0 instead (on interface wlp2s0)
21/12/11 15:51:10 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/spark/jars/spark-unsafe_2.12-3.2.0.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/12/11 15:51:15 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://172.28.203.0:4040
Spark context available as 'sc' (master = local[*], app id = local-1639209076731).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.2.0
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.11)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

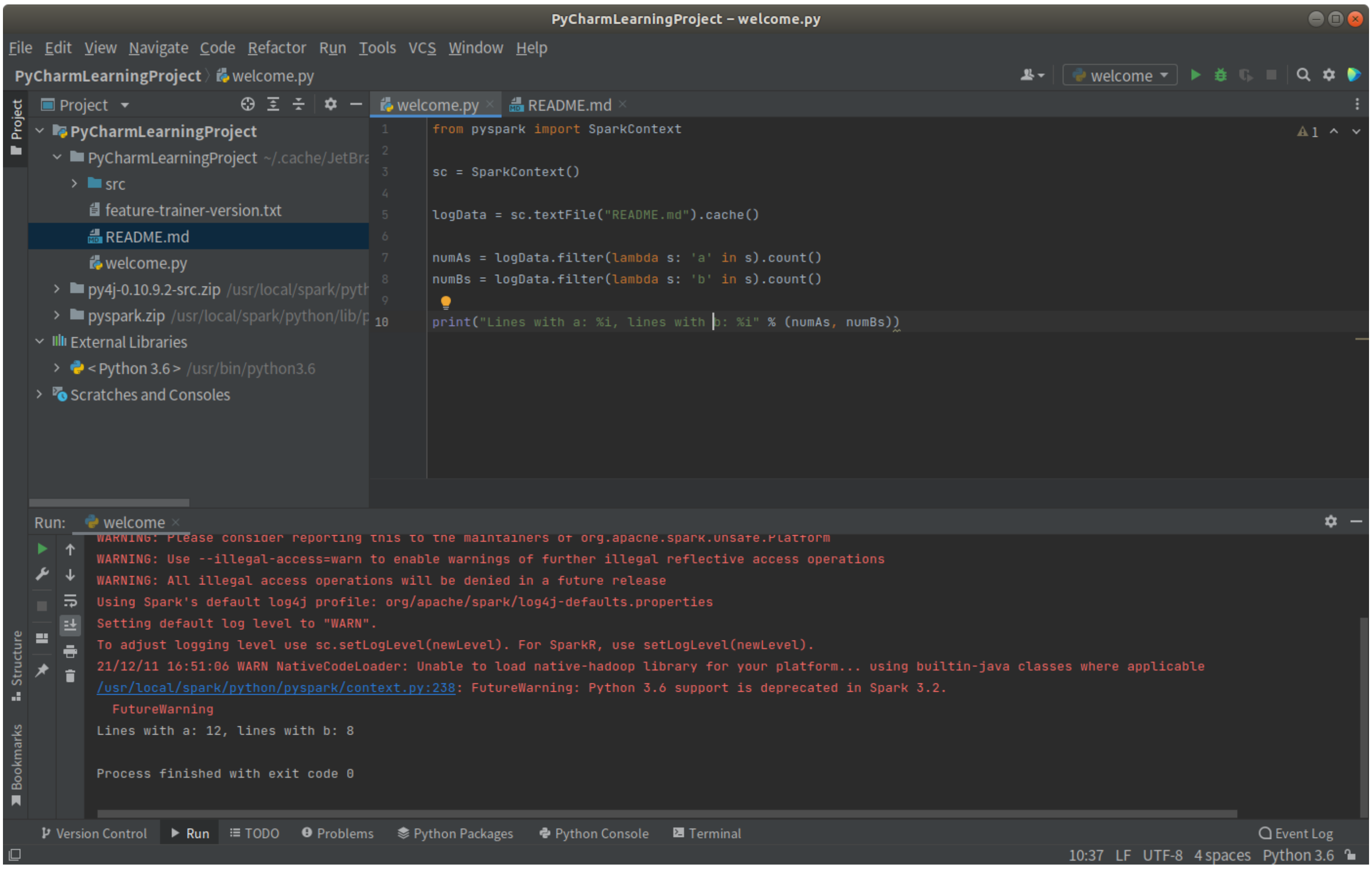


holmes@holmes-Lenovo-XiaoXin-Air-14IKBR: /usr/local/spark/bin

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
21/12/11 15:52:36 INFO TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 434 ms on 172.28.203.0 (executor driver
) (3/10)
21/12/11 15:52:36 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 471 ms on 172.28.203.0 (executor driver
) (4/10)
21/12/11 15:52:36 INFO Executor: Finished task 8.0 in stage 0.0 (TID 8). 1000 bytes result sent to driver
21/12/11 15:52:36 INFO TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 153 ms on 172.28.203.0 (executor driver
) (5/10)
21/12/11 15:52:36 INFO Executor: Finished task 3.0 in stage 0.0 (TID 3). 957 bytes result sent to driver
21/12/11 15:52:36 INFO TaskSetManager: Finished task 3.0 in stage 0.0 (TID 3) in 460 ms on 172.28.203.0 (executor driver
) (6/10)
21/12/11 15:52:36 INFO Executor: Finished task 4.0 in stage 0.0 (TID 4). 957 bytes result sent to driver
21/12/11 15:52:36 INFO TaskSetManager: Finished task 4.0 in stage 0.0 (TID 4) in 472 ms on 172.28.203.0 (executor driver
) (7/10)
21/12/11 15:52:36 INFO Executor: Finished task 9.0 in stage 0.0 (TID 9). 957 bytes result sent to driver
21/12/11 15:52:36 INFO TaskSetManager: Finished task 9.0 in stage 0.0 (TID 9) in 99 ms on 172.28.203.0 (executor driver)
 (8/10)
21/12/11 15:52:36 INFO Executor: Finished task 6.0 in stage 0.0 (TID 6). 957 bytes result sent to driver
21/12/11 15:52:36 INFO Executor: Finished task 5.0 in stage 0.0 (TID 5). 957 bytes result sent to driver
21/12/11 15:52:36 INFO TaskSetManager: Finished task 6.0 in stage 0.0 (TID 6) in 496 ms on 172.28.203.0 (executor driver
) (9/10)
21/12/11 15:52:36 INFO TaskSetManager: Finished task 5.0 in stage 0.0 (TID 5) in 498 ms on 172.28.203.0 (executor driver
) (10/10)
21/12/11 15:52:36 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
21/12/11 15:52:36 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:38) finished in 0.725 s
21/12/11 15:52:36 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
21/12/11 15:52:36 INFO TaskSchedulerImpl: Killing all running tasks in stage 0: Stage finished
21/12/11 15:52:36 INFO DAGScheduler: Job 0 finished: reduce at SparkPi.scala:38, took 0.770874 s
Pi is roughly 3.140999140999141
21/12/11 15:52:36 INFO SparkUI: Stopped Spark web UI at http://172.28.203.0:4040
21/12/11 15:52:36 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
21/12/11 15:52:36 INFO MemoryStore: MemoryStore cleared
21/12/11 15:52:36 INFO BlockManager: BlockManager stopped
21/12/11 15:52:36 INFO BlockManagerMaster: BlockManagerMaster stopped
21/12/11 15:52:36 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
21/12/11 15:52:36 INFO SparkContext: Successfully stopped SparkContext
21/12/11 15:52:36 INFO ShutdownHookManager: Shutdown hook called
21/12/11 15:52:36 INFO ShutdownHookManager: Deleting directory /tmp/spark-e3c3dd20-fa4a-4f9d-8364-d3e8723566b1
21/12/11 15:52:36 INFO ShutdownHookManager: Deleting directory /tmp/spark-0124c888-cfce-463a-9e3b-98054d3732ad
holmes@holmes-Lenovo-XiaoXin-Air-14IKBR:/usr/local/spark/bin$
```

3. 修改配置文件`source ~/.bashrc`添加spark环境变量
4. 在pycharm上的project interpreter上下载py4j
5. 打开project，打开run configurition
6. 设置configurition---Environment--- Environment variables ---点击“...”，点击+，输入两个name，一个是SPARK_HOME，另外一个是PYTHONPATH，设置它们的values，SPARK_HOME的value是安装文件夹spark的绝对路径，PYTHONPATH的value是该绝对路径 / python
7. 在perferences中的project structure中点击右边的“add content root”，添加py4j-some-version.zip和pyspark.zip的路径（这两个文件都在Spark中的python文件夹下）
8. 完成，红线消失，运行正常。


PyCharmLearningProject – welcome.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
PyCharmLearningProject welcome.py
welcome
Project
PyCharmLearningProject
PyCharmLearningProject ~/.cache/JetBra
src
feature-trainer-version.txt
README.md
welcome.py
py4j-0.10.9.2-src.zip /usr/local/spark/pyth
pyspark.zip /usr/local/spark/python/lib/p
External Libraries
< Python 3.6 > /usr/bin/python3.6
Scratches and Consoles
welcome.py
README.md
from pyspark import SparkContext
sc = SparkContext()
logData = sc.textFile("README.md").cache()
numAs = logData.filter(lambda s: 'a' in s).count()
numBs = logData.filter(lambda s: 'b' in s).count()
print("Lines with a: %i, lines with b: %i" % (numAs, numBs))
Run: welcome
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/12/11 16:51:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
/usr/local/spark/python/pyspark/context.py:238: FutureWarning: Python 3.6 support is deprecated in Spark 3.2.
FutureWarning
Lines with a: 12, lines with b: 8
Process finished with exit code 0
Version Control
Run
TODO
Problems
Python Packages
Python Console
Terminal
Event Log
10:37 LF UTF-8 4 spaces Python 3.6