

实验4 个贷违约预测

朱彤轩 191840376

实验4 个贷违约预测

1 写在前面

1.1 环境

1.2 文件结构

2 任务一

3 任务二

4 任务三

4.1 统计 employer_type 数量分布

4.2 统计用户缴纳的利息金额

4.3 筛选工作年限超5年的员工

5 任务四

5.1 数据处理

5.1.1 数据读取

5.1.2 处理缺失值

5.1.3 属性处理

5.2 生成训练集和测试集

5.3 计算数据权重

5.4 模型训练与预测结果

5.4.1 随机森林分类

5.4.2 SVM支持向量机

5.4.3 逻辑回归

6 遇到的问题及解决

6.1 bdkit python相关

6.2 类型转换

7 致谢

1 写在前面

1.1 环境

- 任务1在Windows下IntelliJ用Maven管理项目完成
- 任务2-3在bdkit上完成（后又在jupyter上重新执行便于展示）
- 任务4在windows下local模式spark，采用jupyter notebook编辑器
- mpr程序用java语言，spark程序都用python语言

1.2 文件结构

```
E:\大三上课程\金融大数据\实验\Experiment4\predictLoanDefaulters>tree
卷 Data 的文件夹 PATH 列表
卷序列号为 6241-C227
E:..
├── pic
├── task1
│   ├── output
│   ├── src
│   │   ├── main
│   │   │   ├── java
│   │   │   └── resources
│   │   └── test
│   │       └── java
│   └── target
│       ├── classes
│       ├── generated-sources
│       └── annotations
├── task2
├── task3
└── task4
```

- **pic** 中放本文档中所有用到的图片
- **task1** 任务一相关文件
 - output 存放industry统计的结果
 - src/main/java 存放mapreduce代码
 - pom.xml maven依赖配置
- **task2** 任务二相关文件
 - task2.ipynb 任务二jupyter notebook
 - TotalLoanDistribution.txt total_loan分布统计文件
- **task3** 任务三相关文件
 - task3.ipynb 任务三jupyter notebook
 - employerType.csv
 - userTotalMoney.csv
 - censorStatusCondition.csv
- **task4** 任务四相关文件
 - task4.ipynb 任务四jupyter notebook

2 任务一

目标：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。

使用2个job，第一个任务用于统计industry出现的次数，第二个用于按数量从大到小进行排序。

job1:

Mapper:

将csv文本一行一行读取进来，对读进来的每一行做判断：

1. 如果偏移量是0，则说明是第一行（表头），故跳过，
2. 其余的为每一项数据，用`,`进行分割，找出代表industry的值

输出`<industry, 1>`。

```
public static class countIndustryMapper
    extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);

    @Override
    public void map(Object key, Text value, Context context
    ) throws IOException, InterruptedException {
        if (!key.toString().equals("0")){
            String[] line = value.toString().split(",");
            context.write(new Text(line[10]), one);
        }
    }
}
```

Reducer:

对每一个`industry_value`出现次数进行求和，输出为`<industry, count>`

```
public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    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);
    }
}
```

job1:

Mapper:

读取<industry, count>并倒置成<count, industry>。

Reducer:

在分发过程中重写比较函数使之从大到小按照出现频数排序并输出。

```
public static class SortIndustry extends Reducer<IntWritable, Text, Text, NullWritable>{
    private Text result = new Text();

    @Override
    protected void reduce(IntWritable key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException{
        for(Text val: values){
            result.set(val.toString());
            String str=result+" "+key;
            context.write(new Text(str),NullWritable.get());
        }
    }
}

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);
    }
}
```

结果如下:

```
E: > 大三上课程 > 金融大数据 > Project > predictLoanDefaulters > calculateIndustry > output > ≡ part-r-00000
1  金融业 48216
2  电力、热力生产供应业 36048
3  公共服务、社会组织 30262
4  住宿和餐饮业 26954
5  文化和体育业 24211
6  信息传输、软件和信息技术服务业 24078
7  建筑业 20788
8  房地产业 17990
9  交通运输、仓储和邮政业 15028
10 采矿业 14793
11 农、林、牧、渔业 14758
12 国际组织 9118
13 批发和零售业 8892
14 制造业 8864
```

3 任务二

目标：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。

将csv文本以text文本的形式一行一行读取进来，对读进来的每一行做判断：

1. 如果以loan_id开头则说明是表头，要跳过这一行；
2. 其余的为每一项数据，用,进行分割，找出代表total_loan的值；对total_loan除以1000，表留余数，形成<total_loan//1000, 1>的key-value对；然后reducer统计出现频率。

```
# 用filter过滤掉第一行
lines = spark.read.text(sys.argv[1]).rdd.map(lambda r: r[0]).filter(lambda line: not line.startswith("loan_id"))

counts = lines.map(lambda x: float(x.split(',')[2])/1000) \
    .map(lambda x: (x, 1)) \
    .reduceByKey(add) \
    .sortBy(lambda x: x[0])
```

最后根据结果进行符合输出格式的输：

```
output = counts.collect()

with open("TotalLoanDistribution.txt", "w") as f:
    for (i, count) in output:
        f.write("((%i,%i),%i)\n" % (i * 1000, (i + 1) * 1000, count))
f.close()
spark.stop()
```

部分结果：

```
E: > 大三上课程 > 金融大数据 > 实验 > Experiment4 > ≡ TotalLoanDistribution.txt
1  ((0,1000),2)
2  ((1000,2000),4043)
3  ((2000,3000),6341)
4  ((3000,4000),9317)
5  ((4000,5000),10071)
6  ((5000,6000),16514)
7  ((6000,7000),15961)
8  ((7000,8000),12789)
9  ((8000,9000),16384)
10 ((9000,10000),10458)
11 ((10000,11000),27170)
```

4 任务三

4.1 统计 employer_type 数量分布

目标：统计所有用户所在公司类型 *employer_type* 的数量分布占比情况

1. 统计总共记录数字 `total_num`;
2. 新建一列 `employer_type_count`，全部赋值为1;
3. 按照 `employer_type` 进行 `groupby`，对 `employer_type_count` 求和，及统计出每种员工的数量，存进 `result1`;
4. 对 `result1` 每一项都除以总记录数，得到每种占比;
5. 按照格式要求输出。

```
total_num = df.count() # 总行数
df = df.withColumn('employer_type_count', functions.lit(1))
result1 = df.groupby('employer_type').agg({'employer_type_count': 'sum'})
result1 = result1.withColumn('employer_type_prop', result1['sum(employer_type_count)']/total_num).select('employer_type', 'employer_type_prop')

output1 = result1.collect()
with open("employerType.csv", "w") as f1:
    for (type_name, prop) in output1:
        f1.write("%s,%f\n" % (type_name, prop))
f1.close()
```

结果：

```
E: > 大三上课程 > 金融大数据 > 实验 > Ex
1  幼教与中小学校,0.099983
2  上市企业,0.100127
3  政府机构,0.258153
4  世界五百强,0.053707
5  高等教育机构,0.033687
6  普通企业,0.454343
7
```

4.2 统计用户缴纳的利息金额

新建一列 `total_money`，按照公式计算即可：

```
result2 = df.withColumn('total_money', df['year_of_loan']*df['monthly_payment']*12-df['total_loan']).select('user_id', 'total_money')
output2 = result2.collect()
with open("userTotalMoney.csv", "w") as f2:
    for (user_id, money) in output2:
        f2.write("%s,%f\n" % (user_id, money))
f2.close()
```

部分结果：

user_id	total_money
0	3846.0
1	1840.6000000000004
2	10465.600000000002
3	1758.5200000000004
4	1056.8800000000001

only showing top 5 rows

4.3 筛选工作年限超5年的员工

1. 由于 `work_year` 是字符串形式，所以设定函数 `cal_work_year` 将其转换为数值型。

原始	转换后
None	0
10+ years	11
1-10 years	本身
< 1 years	0

2. 用 `filter` 和 `select` 筛选出 `<user_id, censor_status, int_work_year>` 的数据。

```
def cal_work_year(work_year):
    if work_year == None:
        return 0
    elif '<' in work_year:
        return 1
    else:
        year = work_year.split(' ')[0]
        year = year.split('+')[0]
        return int(year)

# 返回类型为字符串类型
udf_cal_work_year = udf(cal_work_year, IntegerType())
# 使用
result3 = df.withColumn('int_work_year', udf_cal_work_year(df.work_year))

result3 = result3.select(result3.user_id, result3.censor_status, result3.int_work_year).filter(
    result3.int_work_year > 5)

output3 = result3.collect()
with open("censorStatusCondition.csv", "w") as f3:
    for (user_id, censor_status, work_year) in output3:
        f3.write("%s,%s,%s\n" % (user_id, censor_status, work_year))
f3.close()
```

部分结果:

user_id	censor_status	int_work_year
1	2	10
2	1	10
5	2	10
6	0	8
7	2	10

only showing top 5 rows

5 任务四

5.1 数据处理

5.1.1 数据读取

读取csv文件时令 `inferSchema=True`，pyspark会根据读取到的数据形式推断数据的类型。然后再用`printSchema`查看每列的数据类型，看有无错误，经检查是没有错误的。

```
df = spark.read.options(header='True', inferSchema='True', delimiter=',').csv(sys.argv[1]) #读取数据
df.printSchema()
```

```
In [5]: df.printSchema()

root
 |-- loan_id: integer (nullable = true)
 |-- user_id: integer (nullable = true)
 |-- total_loan: double (nullable = true)
 |-- year_of_loan: integer (nullable = true)
 |-- interest: double (nullable = true)
 |-- monthly_payment: double (nullable = true)
 |-- class: string (nullable = true)
 |-- sub_class: string (nullable = true)
 |-- work_type: string (nullable = true)
 |-- employer_type: string (nullable = true)
 |-- industry: string (nullable = true)
 |-- work_year: string (nullable = true)
 |-- house_exist: integer (nullable = true)
 |-- house_loan_status: integer (nullable = true)
 |-- censor_status: integer (nullable = true)
 |-- marriage: integer (nullable = true)
 |-- offsprings: integer (nullable = true)
 |-- issue_date: string (nullable = true)
```


5.1.2 处理缺失值

对于数值变量，缺失值填充为-1，对于字符串变量，缺失值填充为'-1'。注意pyspark对类型要求很严格，只会填充同类型的。

```
df = df.na.fill(-1)
df = df.na.fill('-1')
```

5.1.3 属性处理

(1) 本身为数值型数据的其他特征不做特殊处理。

(2) `loan_id`与`user_id`作为唯一标识符不参与预测模型。

(3) `class`, `sub_class`，将class中的字母A看作10，B看作20以此类推，与subclass的数字相加，合并为一个数字特征。

```
def cal_sub_class(sub_class):
    if sub_class[0] == 'A':
        return 10 + int(sub_class[1])
    elif sub_class[0] == 'B':
        return 20 + int(sub_class[1])
    elif sub_class[0] == 'C':
        return 30 + int(sub_class[1])
    elif sub_class[0] == 'D':
        return 40 + int(sub_class[1])
    elif sub_class[0] == 'E':
        return 50 + int(sub_class[1])
    elif sub_class[0] == 'F':
        return 60 + int(sub_class[1])
    else:
        return 70 + int(sub_class[1])

udf_cal_sub_class = udf(cal_sub_class, IntegerType())
df = df.withColumn('int_sub_class',
udf_cal_sub_class(df.sub_class))
```

(4) 对`work_year`，小于1年的记为0，大于10年的记录为11，其他的转换为准确年数。

```
def cal_work_year(work_year):
    if work_year == '-1':
        return -1
    elif '<' in work_year:
        return 0
    elif '+' in work_year:
        return 11
    else:
        year = work_year.split(' ')[0]
        return int(year)

udf_cal_work_year = udf(cal_work_year, IntegerType())
df = df.withColumn('int_work_year',
udf_cal_work_year(df.work_year))
```

(5) 对于 `issue_date` 和 `earlies_credit_mon` 这两个日期特征，转化为距离现在以年为单位的时间长度，如2015-06-01记录为6.5，表示距离现在6.5年了。

```
def cal_issue_date(issue_date):
    result = (2021 - int(issue_date[:4])) + (12 - int(issue_date[5:7])) / 12
    return float(result)

def cal_earlies_credit_mon(earlies_credit_mon):
    result = 0
    if earlies_credit_mon[:3] == 'Jan':
        result = (2021 - int(earlies_credit_mon[-4:])) + 1
    elif earlies_credit_mon[:3] == 'Feb':
        result = (2021 - int(earlies_credit_mon[-4:])) + 11 / 12
    elif earlies_credit_mon[:3] == 'Mar':
        result = (2021 - int(earlies_credit_mon[-4:])) + 10 / 12
    elif earlies_credit_mon[:3] == 'Apr':
        result = (2021 - int(earlies_credit_mon[-4:])) + 9 / 12
    elif earlies_credit_mon[:3] == 'May':
        result = (2021 - int(earlies_credit_mon[-4:])) + 8 / 12
    elif earlies_credit_mon[:3] == 'Jun':
        result = (2021 - int(earlies_credit_mon[-4:])) + 7 / 12
    elif earlies_credit_mon[:3] == 'Jul':
        result = (2021 - int(earlies_credit_mon[-4:])) + 6 / 12
    elif earlies_credit_mon[:3] == 'Aug':
        result = (2021 - int(earlies_credit_mon[-4:])) + 5 / 12
    elif earlies_credit_mon[:3] == 'Sep':
        result = (2021 - int(earlies_credit_mon[-4:])) + 4 / 12
    elif earlies_credit_mon[:3] == 'Oct':
        result = (2021 - int(earlies_credit_mon[-4:])) + 3 / 12
    elif earlies_credit_mon[:3] == 'Nov':
        result = (2021 - int(earlies_credit_mon[-4:])) + 2 / 12
    else:
        result = (2021 - int(earlies_credit_mon[-4:])) + 1 / 12
    return float(result)
```

```

udf_cal_issue_date = udf(cal_issue_date, FloatType())
udf_cal_earlies_credit_mon = udf(cal_earlies_credit_mon,
FloatType())

df = df.withColumn('float_issue_date',
udf_cal_issue_date(df.issue_date))
df = df.withColumn('float_earlies_credit_mon',
udf_cal_earlies_credit_mon(df.earlies_credit_mon))

```

(6) 对 `work_type`，`employer_type`，`industry` 转化为数字特征进行处理。

运用 `StringIndexer` 对三个 `string` 类型的数据进行数字化处理，处理结果为 `原名_index`，之后将原来的删去。

字符串类型属性转数字特征

```

strings = ['work_type', 'employer_type', 'industry']
indexes = [StringIndexer(inputCol=s, outputCol=s + "_index") for s in strings]
pipeline = Pipeline(stages=indexes)
model = pipeline.fit(df)
df = model.transform(df) # 对strings里面的列进行transform
df = df.drop(*strings) # 删去strings里面的内容

for col in ['loan_id', 'user_id', 'class', 'sub_class', 'issue_date', 'earlies_credit_mon']:
    df = df.drop(col)
data = df.drop('is_default')
assembler = VectorAssembler(inputCols=data.columns, outputCol="features")
data = assembler.transform(df)

```

上述数据处理的部分结果如下：

In [8]: `df.select('int_work_year', 'int_sub_class', 'float_issue_date', 'float_earlies_credit_mon').show(5)`

int_work_year	int_sub_class	float_issue_date	float_earlies_credit_mon
-1	25	6.5	37.833332
11	33	11.166667	30.0
11	42	5.3333335	25.25
2	21	8.583333	21.5
5	21	4.6666665	21.833334

only showing top 5 rows

```
In [10]: df.select('work_type_index', 'employer_type_index', 'industry_index').show(5)
```

work_type_index	employer_type_index	industry_index
1.0	0.0	9.0
0.0	0.0	11.0
2.0	2.0	5.0
1.0	0.0	1.0
0.0	1.0	0.0

only showing top 5 rows

5.2 生成训练集和测试集

运用 `VectorAssembler` 将所有有效的属性继承成 `features` 向量，然后随机按照8:2的比例将整体划分为训练集和测试集。

4.1 去除无关列

```
In [171]: for col in ['loan_id', 'user_id', 'class', 'sub_class', 'issue_date', 'earlies_credit_mon', 'work_year']:  
          df = df.drop(col)
```

4.2 划分训练集和测试集

```
In [172]: df_feats = df.drop('is_default')  
assembler = VectorAssembler(inputCols=df_feats.columns, outputCol="features")  
data = assembler.transform(df) #许多属性, features, is_default  
train_set, test_set = data.randomSplit([0.8, 0.2])
```

5.3 计算数据权重

因为数据集具有一定的不平衡性，仿照sklearn算class_weight的思路，为每个数据项赋予sample weight。

4.3 生成数据权重

```
In [173]: y_collect = train_set.select("is_default").groupBy("is_default").count().collect()
unique_y = [x["is_default"] for x in y_collect]
total_y = sum([x["count"] for x in y_collect])
unique_y_count = len(y_collect)
bin_count = [x["count"] for x in y_collect]

class_weights_spark = {i: ii for i, ii in zip(unique_y, total_y / (unique_y_count * np.array(bin_count)))}
print(class_weights_spark)

{1: 2.503360748131758, 0: 0.6247903056067853}
```

```
In [174]: mapping_expr = F.create_map([F.lit(x) for x in chain(*class_weights_spark.items())])
train_set = train_set.withColumn("weight", mapping_expr.getItem(F.col("is_default")))

train_set.select(['features', 'weight', 'is_default']).show(5)
```

features	weight	is_default
[1000.0, 3.0, 5.32, ...]	0.6247903056067853	0
[1000.0, 3.0, 6.99, ...]	0.6247903056067853	0
[1000.0, 3.0, 7.21, ...]	0.6247903056067853	0
[1000.0, 3.0, 7.35, ...]	0.6247903056067853	0
[1000.0, 3.0, 7.35, ...]	0.6247903056067853	0

only showing top 5 rows

非常注意！前期对属性的各种处理可以在划分训练集、测试集前一并进行，但是计算权重这件事情之能在训练集上进行，因为在模型训练出来之前，测试集是不能动的，前面的处理没有将训练集和测试集混起来，不然就有用label预测label的毛病。

更不能将权重weight作为一个属性，这样会出现很好看的结果，各种指标都是1，但这在逻辑上是不对的！

5.4 模型训练与预测结果

5.4.1 随机森林分类

模型训练与预测：

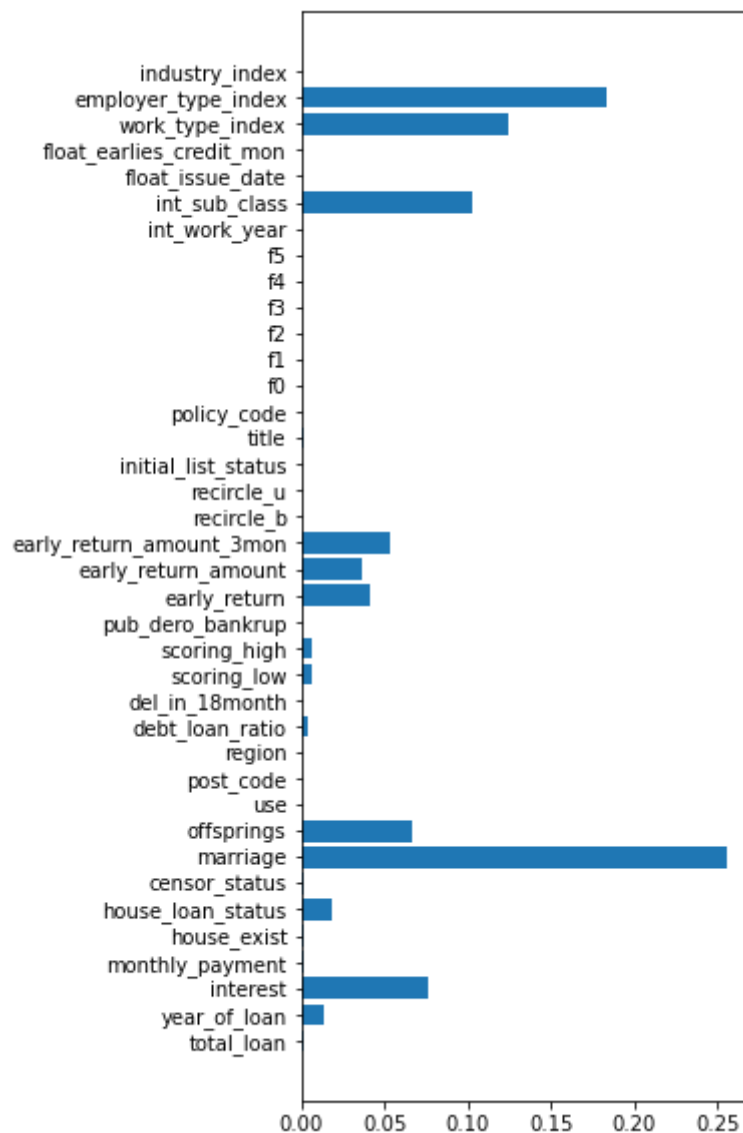
```
In [176]: model_rf = RandomForestClassifier(labelCol='is_default', maxDepth=7, maxBins=700, numTrees=50, weightCol='weight').fit(train_set)
predictions_rf = model_rf.transform(test_set)
```

```
In [177]: predictions_rf.select(['is_default', 'prediction', 'probability']).show(10, False)
```

is_default	prediction	probability
1	0.0	[0.6167966658853142, 0.3832033341146858]
0	0.0	[0.7571229989542049, 0.2428770010457951]
1	1.0	[0.34249927131852664, 0.6575007286814732]
0	1.0	[0.49908166526659825, 0.5009183347334017]
0	0.0	[0.7666004375831156, 0.23339956241688448]
0	0.0	[0.861818939303072, 0.13818106069692804]
0	0.0	[0.7513639918350212, 0.24863600816497883]
0	0.0	[0.6445826392391486, 0.3554173607608515]
0	0.0	[0.6668882331919371, 0.3331117668080628]
0	0.0	[0.8144055197916006, 0.18559448020839941]

only showing top 10 rows

重要特征：



相关指标：

精确率：0.4427646765943964
召回率：0.8156424581005587
F1分数：0.5739599835709676
准确率：0.7585622132074217
auc：0.8605717064923736

5.4.2 SVM支持向量机

模型训练与预测：

```
from pyspark.ml.classification import LinearSVC
model_svm = LinearSVC(maxIter=100, labelCol='is_default', weightCol='weight').fit(train_set)
predictions_svm = model_svm.transform(test_set)
```

```
predictions_svm.select(['is_default', 'prediction']).show(10, False)
```

is_default	prediction
1	0.0
0	0.0
1	1.0
0	1.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0
0	0.0

only showing top 10 rows

相关指标:

精确率: 0.40208270579629074
 召回率: 0.7791211540065038
 F1分数: 0.530426884650318
 准确率: 0.7249451353328457
 auc: 0.8178927237891506

5.4.3 逻辑回归

模型训练与预测:

```
from pyspark.ml.classification import LogisticRegression
model_lr = LogisticRegression(regParam=0.01, labelCol='is_default', weightCol='weight').fit(train_set)
predictions_lr = model_lr.transform(test_set)
```

```
predictions_lr.select(['is_default', 'prediction', 'probability']).show(10, False)
```

is_default	prediction	probability
1	0.0	[0.8298544021177874, 0.1701455978822125]
0	0.0	[0.8032571712274261, 0.19674282877257385]
1	1.0	[0.26023130518954074, 0.7397686948104593]
0	1.0	[0.4537802912605894, 0.5462197087394106]
0	0.0	[0.7563577993246436, 0.24364220067535627]
0	0.0	[0.7889071150381572, 0.21109288496184272]
0	0.0	[0.6881214888257705, 0.3118785111742295]
0	0.0	[0.6553438995039562, 0.3446561004960438]
0	0.0	[0.6568593770922325, 0.3431406229077674]
0	0.0	[0.8965711222858308, 0.1034288777141691]

only showing top 10 rows

相关指标:

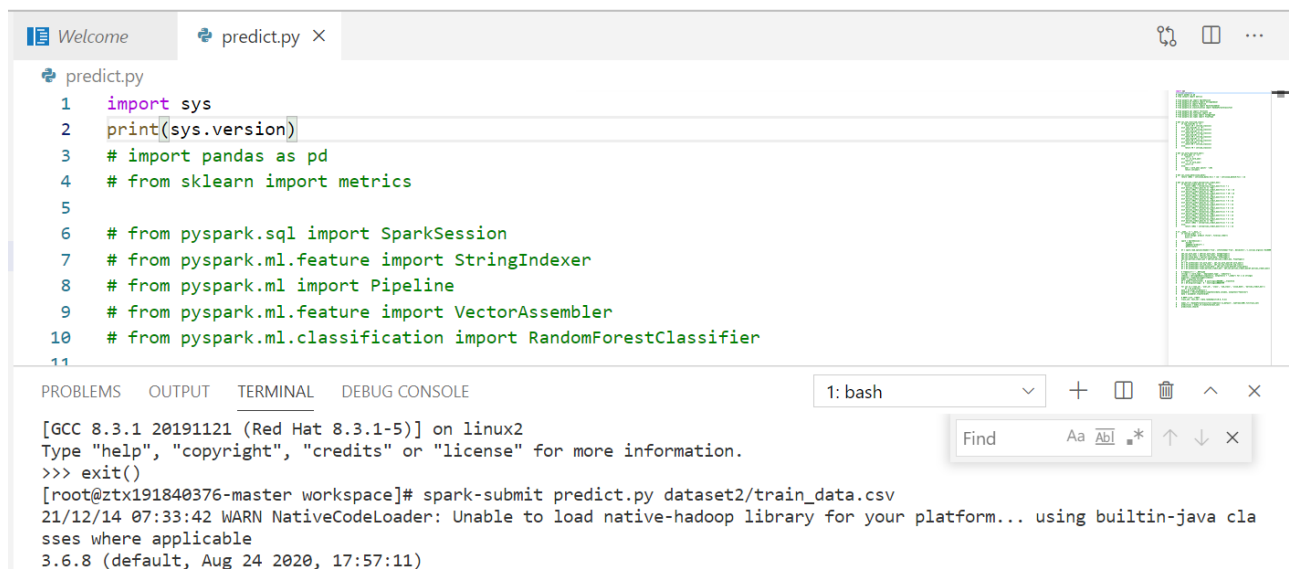
```
精确率: 0.40543161978661496
召回率: 0.7667806220295172
F1分数: 0.5304109589041096
准确率: 0.7292844317350535
auc: 0.8153035546650128
```

6 遇到的问题及解决

6.1 bdkit python相关

在使用bdkit的时候发现StringIndexer貌似依赖numpy，而环境中是没有的。经测试，发现：

(1) bdkit的console中是python2，submit到pyspark中是python3.



The screenshot shows an IDE with a file named 'predict.py' open. The code in the file is as follows:

```
1 import sys
2 print(sys.version)
3 # import pandas as pd
4 # from sklearn import metrics
5
6 # from pyspark.sql import SparkSession
7 # from pyspark.ml.feature import StringIndexer
8 # from pyspark.ml import Pipeline
9 # from pyspark.ml.feature import VectorAssembler
10 # from pyspark.ml.classification import RandomForestClassifier
11
```

Below the code editor is a terminal window. The terminal output shows the execution of the script in a bash shell:

```
1: bash
[GCC 8.3.1 20191121 (Red Hat 8.3.1-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
[root@ztx191840376-master workspace]# spark-submit predict.py dataset2/train_data.csv
21/12/14 07:33:42 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cla
sses where applicable
3.6.8 (default, Aug 24 2020, 17:57:11)
```

(2) pip install显示没有pip这个命令，说明pip与python分开的，用命令**python -m pip install --upgrade pip**安装pip

(3) 在console里pip install numpy的时候需要用**pip3**不然安装到python2的环境中去了

6.2 类型转换


```
In [11]: df.earlies_credit_mon.apply(cal_earlies_credit_mon).head(20)
Out[11]:
0      37.833333
1      30.000000
2      25.250000
3      21.500000
4      21.833333
5      19.666667
6      21.416667
7      16.250000
8      20.750000
9      29.333333
10     35.916667
11     17.416667
12     25.500000
```

执行对`earlies_credit_mon`的处理操作之后，在打印前20个数的时候发现第2个、第15个和第17个是`null`。在python中用pandas dataframe的`apply`试过之后发现没有问题。观察这3个数的共性，发现相比其他的数字有小数，这3个数字其实是整数。

而在前面函数输出类型的时候，我给的是`float`类型。所以在函数的返回值那里显式将结果返回为`float`类型，这样一来就解决了数值“莫名其妙”变`null`的问题。

```
else:
    result = (2021 - int(earlies_credit_mon[-4:])) + 1 / 12
    return float(result)
```

7 致谢

感谢施宇同学在我转换变成环境配置上提供的教程；

感谢邵路婷同学给我在啥也不懂的状态下引路，同时我们一起探讨了为什么会出现`accuracy 100%`，`F1 100%`的不正常情况。