实验3-4 数据挖掘

> author lyh

0. 实验环境

docker spark-2.4.4集群，pyspark，sparkML

关于ML和MLlib：ml和mllib都是Spark中的机器学习库

- ml主要操作的是DataFrame, 而mllib操作的是RDD，相比于mllib在RDD提供的基础操作，ml在DataFrame上的抽象级别更高，数据和操作耦合度更低
- ml中的操作可以使用pipeline，可以把很多操作(算法/特征提取/特征转换)以管道的形式串起来，然后让数据在这个管道中流动。
- ML中的随机森林支持更多的功能：包括重要度、预测概率输出等，而MLlib不支持
- ML中无论是什么模型，都提供了统一的算法操作接口，比如模型训练都是fit；不像MLlib中不同模型会有各种各样的train（不过ML也有一些缺陷，如评价器evaluator中很多功能都还在实验阶段experimental）

1. 数据分析

train

summary	user_id	age_range	gender	merchant_id	label
count	10000	10000	10000	10000	10000
mean	209849.3373	3.7071	1.0956	2561.4367	0.0568
stddev	122747.77318693754	1.2814342735309296	0.29405660497375835	1452.6312716426876	0.23147163522080014
min	147	1	1	2	0
max	424167	8	2	4993	1

- 首先，原始数据分为train和test两个数据文件，二者的user_id的交集为空。并且，从理论上来说，user_id只是对顾客的一个编号，没有实际意义，任意两个user_id之间应该不存在任何维度的比较关系，所以模型中应该考虑去除user_id这个维度。merchant_id 同理，仅仅是对商家的编号，不存在比较关系，理论上应该去除。但是，如果去除这两个维度，剩下年龄和性别两个维度一共就十几个取值组合又太少，所以还是尝试包括user_id和merchant_id这两个维度。
- 在train和test中，没有女性样本，且均只有约9%的未知性别样本，数据倾斜严重。
- 训练集正负样本严重不平衡，只有少量（5.68%）的label为1，其余都是0。理论上来说，机器学习要求训练样本基本均匀，不均匀的样本，对模型的效果有影响。
- 考虑结合million_user文件中的数据，但是发现，在训练集中的user_id和merchant_id有超过90%都没在million_user文件中出现过，所以这个文件中的数据基本也用不上。。

综上所述，模型训练采取的方法为：

1. 为了使模型正负样本均匀，对label为1的样本进行过采样，使label为1 的样本扩展到4544个，再选取相同数量的label0样本进行拼接，shuffle，得到最终数据集。
2. 对数据集进行0.75/0.25划分为train为test（下图为最终数据集）

summary	user_id	age_range	gender	merchant_id	label
count	9088	9088	9088	9088	9088
mean	207740.8019366197	3.8106294014084505	1.1003521126760563	2570.7718970070423	0.5
stddev	122214.96781058611	1.3271464194114424	0.3004854429118669	1443.634340462476	0.5000275110732277
min	147	1	1	2	0
max	424098	8	2	4993	1

```
#数据预处理
```

```
import pyspark.sql.types as typ
labels = [('user_id', typ.IntegerType()),
          ('age_range', typ.IntegerType()),
          ('gender', typ.IntegerType()),
          ('merchant_id', typ.IntegerType()),
          ('label', typ.IntegerType())
          ]
schema = typ.StructType([
    typ.StructField(e[0], e[1], False) for e in labels
])

train=spark.read.csv('file:///home/Hadoop/share-
files/train_after.csv',header=False,schema=schema)
#train.show(2)
test=spark.read.csv('file:///home/Hadoop/share-
files/test_after.csv',header=False,schema=schema)
#test.show(2)
label1=train.filter('label == 1 ')#568
#x.describe().show()
#过采样
for i in range(3):
    label1=label1.union(label1)
#label1.describe().show()#4544
label0=train.filter('label==0').limit(4544)
even_data=label1.union(label0)
even_data=even_data.sample(fraction=1.0)#打乱顺序
#even_data.describe().show()
```

2. 模型训练及评估

example1：逻辑回归模型，方法一

```
from pyspark.ml.classification import LogisticRegression
model=LogisticRegression().fit(train_df)

for df in [train_df,test_df]:

    result=model.transform(df)
    #result.show()
    print('areaUnderPR',evaluator.evaluate(result, {evaluator.metricName:
"areaUnderPR"}))
    print('areaUnderROC',evaluator.evaluate(result, {evaluator.metricName:
"areaUnderROC"}))
    #print('{}{}'.format('训练准确率：',model1.evaluate(train1).accuracy) )
    #print('accuracy',mev.evaluate(result, {mev.metricName: "accuracy"}))
    #print('precision',mev.evaluate(result, {mev.metricName:
"weightedPrecision"}))
    #print('recall',mev.evaluate(result, {mev.metricName: "weightedRecall"}))
    pred=result.select('prediction').toPandas()['prediction'].tolist()
    pred=list(map(int,pred))
    #pred=df['prediction'].tolist()
    tr=df.select('label').toPandas()['label'].tolist()
    print('precision',precision_score(tr, pred, average='binary')  )
    print('recall',recall_score(tr, pred, average='binary')  )
```

example1：逻辑回归模型，方法二（使用pipeline）

```
# 创建一个管道
from pyspark.ml import Pipeline
#创建整合器
featuresAssembler=ft.VectorAssembler(
    inputCols=['user_id','age_range','merchant_id','gender'],
    outputCol='features'
    )
model=LogisticRegression()
pipeline = Pipeline(stages=[featureAssembler,model ])
model = pipeline.fit(train)
```

模型评估：（precision和recall均指的是正例1）

训练集
```
areaUnderPR 0.53829
areaUnderROC 0.5331
precision 0.5281150
recall 0.4878984651
```
测试集
```
areaUnderPR 0.550709
areaUnderROC 0.55019
precision 0.54808590
recall 0.507785467
```

从结果来看，训练集和测试集上的precision和recall都不是很高，ROC面积略微超过0.5。

多种模型训练及调参评估

> 展示的是模型在平衡数据集中分割出的测试集上的结果

```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import NaiveBayes
```

1. DecisionTree决策树：
```
areaUnderPR 0.5878529201632814
areaUnderROC 0.5897315694836887
precision 0.5593108018555335
recall 0.7301038062283737
```

2. GBTClassifier梯度提升树：
```
areaUnderPR 0.6894846489718466
areaUnderROC 0.7192381499221832
precision 0.647843137254902
recall 0.7145328719723183
```
(maxiter=10)

3. RandomForest随机森林：
```
areaUnderPR 0.6367824073914927
areaUnderROC 0.6331454647104153
precision 0.6114965312190287
recall 0.5337370242214533
```
(numTrees：20)

4. NaiveBayes朴素贝叶斯：
```
areaUnderPR 0.4883617335940174
areaUnderROC 0.48509277587222926
precision 0.4935511607910576
recall 0.49653979238754326
```

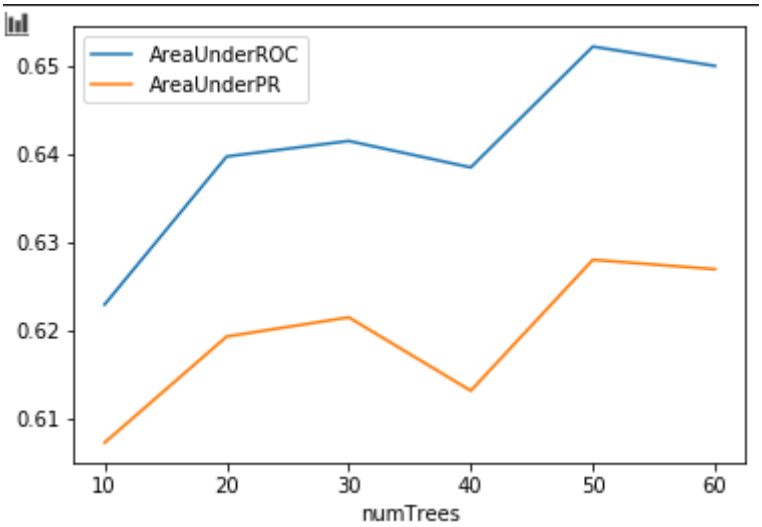
5. SVMClassifier支持向量机：
```
precision 0.50239
recall 1.0
```

可以看到以上几种机器学习方法中，决策树类算法效果优于其他方法，其中梯度提升树效果最好。

模型调参

下面尝试对梯度提升树和随机森林进行调参：

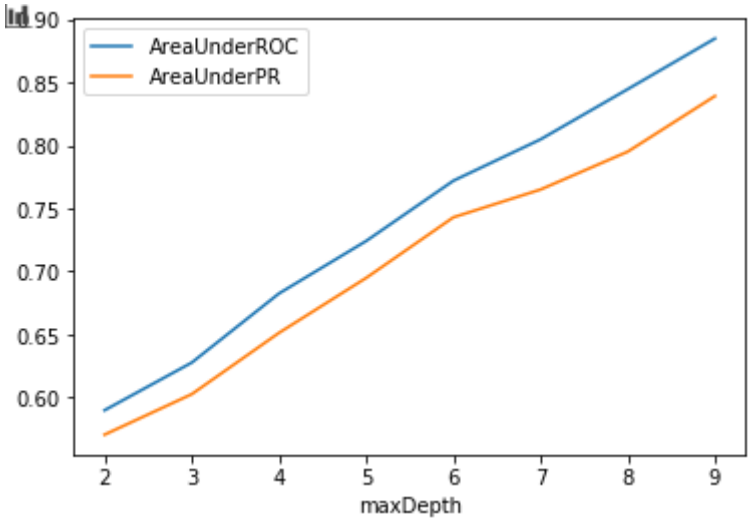
- 对随机森林进行调参：



可以看到基本上决策树的数量越多，模型效果越好。

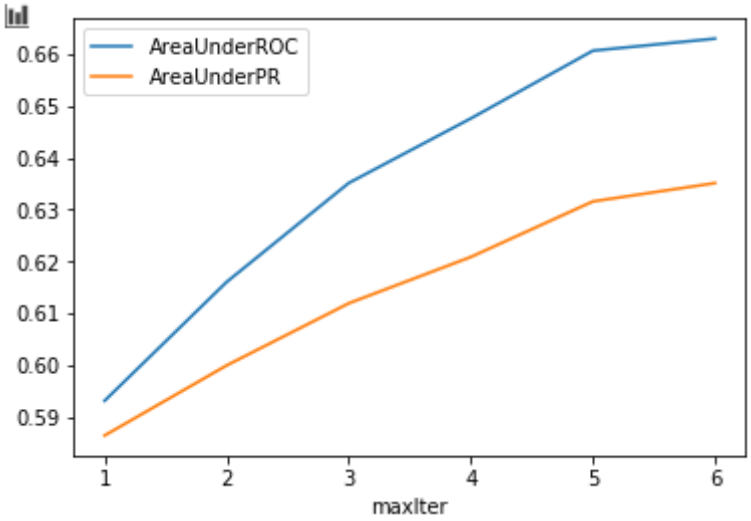
- 对梯度提升树进行调参：

对最大深度进行调参



可以看到，模型效果基本和决策树的深度成线性关系。。。

对最大循环次数调参



可以看到，在最大循环次数等于6左右的时候，深度的增加对模型的改进基本收敛。

最后，对测试集给出预测：

```
result=model.transform(test1)
result.select(['user_id','age_range','gender','prediction']).show()
```

```
+-------+---------+------+----------+
|user_id|age_range|gender|prediction|
+-------+---------+------+----------+
| 360576|        2|     2|       0.0|
| 295296|        2|     1|       1.0|
| 230016|        5|     1|       1.0|
| 164736|        3|     1|       0.0|
| 164736|        3|     1|       0.0|
| 164736|        3|     1|       0.0|
| 296064|        3|     1|       1.0|
| 100992|        4|     1|       0.0|
| 100992|        4|     1|       0.0|
| 102528|        5|     1|       0.0|
| 233856|        6|     1|       1.0|
| 168576|        4|     1|       0.0|
| 106368|        5|     1|       0.0|
| 106368|        5|     1|       0.0|
| 302976|        6|     1|       1.0|
| 302976|        6|     1|       0.0|
|  41088|        2|     1|       0.0|
| 107136|        2|     1|       0.0|
| 107904|        3|     1|       0.0|
|  43392|        4|     1|       0.0|
+-------+---------+------+----------+
only showing top 20 rows
```