FBDP-Exp4

邵一淼 191098180

README还需再补充一些

代码实现

任务一

任务一涉及程序保存在仓库中的Task1文件夹下

设计思路

一个在csv上操作的WordCount

程序结构

类	功能
Runner	入口类
CountMapper	WordCount的mapper
CountReducer	WordCount的reducer
InverseMapper	排序的mapper，将key和value合成新key
InverseReducer	将新key拆回来
TextIntWritable	定义新类型和排序方式

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

Code保存在Task2.ipynb中，基于pySpark完成

导包和pyspark环境设置

```
import findspark
findspark.init()
import pandas as pd
from pyspark import SparkContext
from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import pandas_udf
from pyspark import SQLContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,
LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import OneHotEncoder
from pyspark.sql.types import IntegerType
sc = SparkContext("local", "first")
spark = SparkSession.builder.config("spark.driver.memory", "16g").getOrCreate()
```

读入数据并计算total_loan的分布

```
df=spark.read.options(header='True') .csv("file:///F:/FBDP/实验/实验
四/train_data.csv")
sum=0
ranges=[]
count=[]
for i in range(41):
    left=i*1000
    right=(i+1)*1000
    res=df.filter((((df['total_loan']-str(left))>=0)&(df['total_loan']-str(right)
<0)).count()
    now_range='(('+str(left)+','+str(right)+'))'
    ranges.append(now_range)
    count.append(res)
    print("((%i,%i),%i)" % (i*1000,(i+1)*1000,res))
    sum=sum+res
print("共计%i条" % sum)
```

将结果存入csv

```
data={'range':ranges,'count':count}
df = pd.DataFrame(data, columns=['range', 'count'])
df.to_csv("task2_output.csv")
```

stop SparkContext

```
sc.stop()
```

结果展示

```
((0,1000),2)
((1000,2000),4043)
((2000,3000),6341)
((3000,4000),9317)
((4000,5000),10071)
((5000,6000),16514)
((6000,7000),15961)
((7000,8000),12789)
((8000,9000),16384)
((9000,10000),10458)
((10000,11000),27170)
((11000,12000),7472)
((12000,13000),20513)
((13000,14000),5928)
((14000,15000),8888)
((15000,16000),18612)
((16000,17000),11277)
((17000,18000),4388)
((18000,19000),9342)
((19000,20000),4077)
((20000,21000),17612)
((21000,22000),5507)
((22000,23000),3544)
((23000,24000),2308)
((24000,25000),8660)
((25000,26000),8813)
((26000,27000),1604)
((27000,28000),1645)
((28000,29000),5203)
((29000,30000),1144)
((30000,31000),6864)
((31000,32000),752)
((32000,33000),1887)
((33000,34000),865)
((34000,35000),587)
((35000,36000),11427)
((36000,37000),364)
((37000,38000),59)
((38000,39000),85)
((39000,40000),30)
((40000,41000),1493)
共计300000条
```

任务三

Code保存在Task3.ipynb中，基于pySpark完成

读入数据并将DataFrame注册为SQL临时视图

```
spark = SQLContext(sc)
df=spark.read.options(header='True') .csv("file:///F:/FBDP/实验/实验四/train_data.csv")
df = df.na.fill(-1)
df = df.na.fill('-1')
#df.printSchema() #打印数据的树形结构
#df.show()
df.createOrReplaceTempView("debit")#将DataFrame注册为SQL临时视图
sqlDF = spark.sql("SELECT * FROM debit")
```

统计所有用户所在公司类型 employer_type 的数量分布占比情况并存入csv

```
emp_res=df.groupby('employer_type').count()
emp_res.createOrReplaceTempView("employer")
emp_res.show()
spark.sql("select employer_type,count/300000 from 
employer").toDF("employer_type","ratio").show()
task3_1_df=spark.sql("select employer_type,count/300000 from 
employer").toDF("employer_type","ratio")
task3_1_df.toPandas().to_csv('task3_1_output.csv')
```

3_1结果展示

employer_type	count
幼教与中小学校	29995
上市企业	30038
政府机构	77446
世界五百强	16112
高等教育机构	10106
普通企业	136303

employer_type	ratio
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	0.053706666666666666
高等教育机构	0.033686666666666666
普通企业	0.4543433333333333

统计每个用户最终须缴纳的利息金额并存入csv

```
task3_2_df=spark.sql("select user_id,year_of_loan*monthly_payment*12-total_loan 
from debit").toDF("user_id","total_money")
task3_2_df.show()
task3_2_df.toPandas().to_csv('task3_2_output.csv')
```

3_2结果展示

```
+-------+------------------+
|user_id|       total_money|
+-------+------------------+
|      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|
|     11|1873.5599999999977|
|     12| 5692.279999999999|
|     13|1258.6800000000003|
|     14|6833.5999999999985|
|     15| 9248.200000000004|
|     16| 6197.119999999995|
|     17|1312.4400000000005|
|     18| 5125.200000000001|
|     19|1215.8400000000001|
+-------+------------------+
only showing top 20 rows
```

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并存入csv

```
# 观察work_year分布规律，发现只有5 years/6 years/7 years/8 years/9 years/10+ years几种情况
df.groupby("work_year").count().show()
task3_3_df=spark.sql("select user_id,censor_status,work_year from debit where work_year like '%5%' or work_year like '%6%' or work_year like '%7%' or work_year like '%8%' or work_year like '%9%' or work_year like '%10%'")
task3_3_df.show()
task3_3_df.toPandas().to_csv('task3_3_output.csv')
```

3_3结果展示

```
+---------+-----+

+-------+-------------+---------+
|user_id|censor_status|work_year|
+-------+-------------+---------+
|      1|            2|10+ years|
|      2|            1|10+ years|
|      4|            0|  5 years|
|      5|            2|10+ years|
|      6|            0|  8 years|
|      7|            2|10+ years|
|      9|            0|10+ years|
|     10|            2|10+ years|
|     15|            1|  7 years|
|     16|            2|10+ years|
|     17|            0|10+ years|
|     18|            1|10+ years|
|     20|            1|  7 years|
|     21|            2|10+ years|
|     25|            2|10+ years|
|     26|            0|10+ years|
|     30|            0|10+ years|
|     31|            0|  6 years|
|     33|            1|10+ years|
|     37|            1|  5 years|
+-------+-------------+---------+
only showing top 20 rows
```

任务四

Code保存在Task4.ipynb中，基于pySpark完成

导包和环境配置

```
import findspark
findspark.init()
import pandas as pd
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import pandas_udf
from pyspark import SQLContext
from pyspark.mllib.classification import LogisticRegressionWithLBFGS,
LogisticRegressionModel
from pyspark.mllib.regression import LabeledPoint
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import OneHotEncoder
from pyspark.sql.types import IntegerType
sc = SparkContext("local", "first")
logFile = "file:///F:/FBDP/实验/实验四/logfile1.txt"
logData = sc.textFile(logFile).cache()
spark = SparkSession.builder.config("spark.driver.memory", "16g").getOrCreate()
```

读入数据并填补缺失值

```
df=spark.read.options(header='True') .csv("file:///F:/FBDP/实验/实验四/train_data.csv")
df = df.na.fill(-1)
df = df.na.fill('-1')
```

把categorical变量转换为数值(略显粗暴版)

```
#class
class_indexer = StringIndexer(inputCol='class', outputCol='class_num').fit(df)
df = class_indexer.transform(df)
class_onehoter = OneHotEncoder(inputCol='class_num', outputCol='class_vector')
df = class_onehoter.transform(df)
#sub_class
sub_class_indexer = StringIndexer(inputCol='sub_class',
outputCol='sub_class_num').fit(df)
df = sub_class_indexer.transform(df)
sub_class_onehoter = OneHotEncoder(inputCol='sub_class_num',
outputCol='sub_class_vector')
df = sub_class_onehoter.transform(df)
#work_type
work_type_indexer = StringIndexer(inputCol='work_type',
outputCol='work_type_num').fit(df)
df = work_type_indexer.transform(df)
work_type_onehoter = OneHotEncoder(inputCol='work_type_num',
outputCol='work_type_vector')
df = work_type_onehoter.transform(df)
#employer_type
employer_type_indexer = StringIndexer(inputCol='employer_type',
outputCol='employer_type_num').fit(df)
df = employer_type_indexer.transform(df)
employer_type_onehoter = OneHotEncoder(inputCol='employer_type_num',
outputCol='employer_type_vector')
df = employer_type_onehoter.transform(df)
#industry
industry_indexer = StringIndexer(inputCol='industry',
outputCol='industry_num').fit(df)
df = industry_indexer.transform(df)
industry_onehoter = OneHotEncoder(inputCol='industry_num',
outputCol='industry_vector')
df = industry_onehoter.transform(df)
#work_year
work_year_indexer = StringIndexer(inputCol='work_year',
outputCol='work_year_num').fit(df)
df = work_year_indexer.transform(df)
work_year_onehoter = OneHotEncoder(inputCol='work_year_num',
outputCol='work_year_vector')
df = work_year_onehoter.transform(df)
#df.show(3)
#issue_date
issue_date_indexer = StringIndexer(inputCol='issue_date',
outputCol='issue_date_num').fit(df)
df = issue_date_indexer.transform(df)
issue_date_onehoter = OneHotEncoder(inputCol='issue_date_num',
outputCol='issue_date_vector')
df = issue_date_onehoter.transform(df)
#df.show(3)
#earlies_credit_mon
earlies_credit_mon_indexer = StringIndexer(inputCol='earlies_credit_mon',
outputCol='earlies_credit_mon_num').fit(df)
df = earlies_credit_mon_indexer.transform(df)
earlies_credit_mon_onehoter = OneHotEncoder(inputCol='earlies_credit_mon_num',
outputCol='earlies_credit_mon_vector')
```

```
df = earlies_credit_mon_onehoter.transform(df)
df.show(3)
```

把string列转为int

```
tmpCols=['total_loan', 'year_of_loan', 'interest', 'monthly_payment',
'class_vector','sub_class_vector','work_type_vector','work_year_vector','employe
r_type_vector','industry_vector','issue_date_vector','earlies_credit_mon_vector'
,'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_amou
nt_3mon','recircle_b','recircle_u','initial_list_status','title','policy_code','
f0','f1','f2','f3','f4','f5']
for i in tmpCols:
    if "vector" in i:
        print("")
    else:
        df = df.withColumn(i, df[i].cast('double'))
#df = df.withColumn("total_loan", df["total_loan"].cast(IntegerType()))
df = df.withColumn('is_default', df['is_default'].cast(IntegerType()))
df.show(3)
```

把输入特征合并到一列

```
ata = df.drop('is_default')
feas = data.columns
df_assembler = VectorAssembler(inputCols=['total_loan', 'year_of_loan',
'interest', 'monthly_payment',
'class_vector','sub_class_vector','work_type_vector','work_year_vector','employe
r_type_vector','industry_vector','issue_date_vector','earlies_credit_mon_vector'
,'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_amou
nt_3mon','recircle_b','recircle_u','initial_list_status','title','policy_code','
f0','f1','f2','f3','f4','f5'],outputCol='features')
print(df_assembler)
data = df_assembler.transform(df)
data.show()
```

划分数据集（8：2）

```
data_set = data.select(['features', 'is_default'])
train_df, test_df = data_set.randomSplit([0.8, 0.2])
#print(' train_df shape : (%d , %d)'%(train_df.count(), len(train_df.columns)))
#print(' test_df  shape: :(%d , %d)'%(test_df.count(), len(test_df.columns)))
```

训练逻辑回归模型

```
log_reg = LogisticRegression(labelCol = 'is_default').fit(train_df)
train_pred = log_reg.evaluate(train_df).predictions
train_pred.filter(train_pred['is_default'] == 1).filter(train_pred['prediction']
== 1).select(['is_default', 'prediction', 'probability']).show(10, False)
```

评估模型

```
test_result = log_reg.evaluate(test_df).predictions
test_result.show(3)
```

模型accuracy

```
tp = test_result[(test_result.is_default == 1) & (test_result.prediction ==
1)].count()
tn = test_result[(test_result.is_default == 0) & (test_result.prediction ==
1)].count()
fp = test_result[(test_result.is_default == 0) & (test_result.prediction ==
1)].count()
fn = test_result[(test_result.is_default == 1) & (test_result.prediction ==
0)].count()
# Accuracy
print('test accuracy is : %f'%((tp+tn)/(tp+tn+fp+fn)))
```

模型召回率、准确率、f1 score

```
recal=tp/(tp+fn)
prec=tp/(tp+fp)
print('test recall is : %f'%(recal))
print('test precision is : %f'%(prec))
print('test f1-score is : %f'%(2*recal*prec/(prec+recal)))
```

结果展示

逻辑回归模型评估

accuracy	0.84
recall	0.42
precision	0.66
f1 score	0.51

遇到的问题

1、TypeError:an integer is required(got type bytes)及一系列安装

问题描述：在Anaconda Powershell Prompt中输入pyspark查看是否安装成功时，显示TypeError:an integer is required(got type bytes)

```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\dell> pyspark
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
Traceback (most recent call last):
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\shell.py", line 31, in <module>
    from pyspark import SparkConf
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\__init__.py", line 51, in <module>
    from pyspark.context import SparkContext
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\context.py", line 31, in <module>
    from pyspark import accumulators
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\accumulators.py", line 97, in <module>
    from pyspark.serializers import read_int, PickleSerializer
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\serializers.py", line 72, in <module>
    from pyspark import cloudpickle
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\cloudpickle.py", line 145, in <module>
    _cell_set_template_code = _make_cell_set_template_code()
  File "E:\spark\spark-2.4.8-bin-hadoop2.7\python\pyspark\cloudpickle.py", line 126, in _make_cell_set_template_code
    return types.CodeType(
TypeError: an integer is required (got type bytes)
>>> _
```

解决方案：python3.8以上对于pyspark不是特别兼容，更换成3.6的环境就能顺利解决

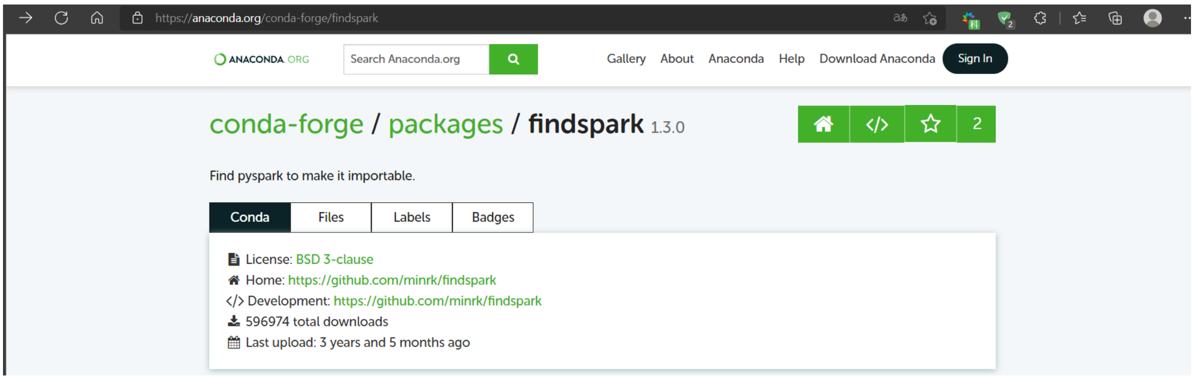
```
Anaconda Powershell Prompt (Anaconda3)
(base) PS C:\Users\dell> conda activate pytorch
(pytorch) PS C:\Users\dell> python
Python 3.6.13 |Anaconda, Inc.| (default, Mar 16 2021, 11:37:27) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
(pytorch) PS C:\Users\dell> pip install pyspark
Collecting pyspark
  Using cached pyspark-3.2.0.tar.gz (281.3 MB)
Collecting py4j==0.10.9.2
  Using cached py4j-0.10.9.2-py2.py3-none-any.whl (198 kB)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.2.0-py2.py3-none-any.whl size=281805912 sha256=0b1aebee496baadef80310948
25d8923fd251afdf8e460eafb5a71bealfceea7
  Stored in directory: c:\users\dell\appdata\local\pip\cache\wheels\e8\d9\e5\78436a0a3899d81410aeb45b200153113667f2e250f
6882ada
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.2 pyspark-3.2.0
(pytorch) PS C:\Users\dell> pyspark
Python 3.6.13 |Anaconda, Inc.| (default, Mar 16 2021, 11:37:27) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.8
      /_/

Using Python version 3.6.13 (default, Mar 16 2021 11:37:27)
SparkSession available as 'spark'.
>>> _
```

但是问题并没有完全解决，在第一次运行代码，也就是import各种包的时候，依然报了如下的错

```
E:\Anaconda3\envs\pytorch\lib\site-packages\pyspark\context.py in __init__(self, master, appName, sparkHome, pyFiles, enviro
nment, batchSize, serializer, conf, gateway, jsc, profiler_cls)
    145         try:
    146             self._do_init(master, appName, sparkHome, pyFiles, environment, batchSize, serializer,
--> 147                           conf, jsc, profiler_cls)
    148         except:
    149             # If an error occurs, clean up in order to allow future SparkContext creation:

E:\Anaconda3\envs\pytorch\lib\site-packages\pyspark\context.py in _do_init(self, master, appName, sparkHome, pyFiles, enviro
nment, batchSize, serializer, conf, jsc, profiler_cls)
    222         # data via a socket.
    223         # scala's mangled names w/ $ in them require special treatment.
--> 224         self._encryption_enabled = self._jvm.PythonUtils.isEncryptionEnabled(self._jsc)
    225         os.environ["SPARK_AUTH_SOCKET_TIMEOUT"] = \
    226             str(self._jvm.PythonUtils.getPythonAuthSocketTimeout(self._jsc))

E:\Anaconda3\envs\pytorch\lib\site-packages\py4j\java_gateway.py in __getattr__(self, name)
   1534         else:
   1535             raise Py4JError(
-> 1536                 "{0}.{1} does not exist in the JVM".format(self._fqn, name))
   1537
   1538     def _get_args(self, args):

Py4JError: org.apache.spark.api.python.PythonUtils.isEncryptionEnabled does not exist in the JVM
```

然后依照众多教程，在代码前面加了两行

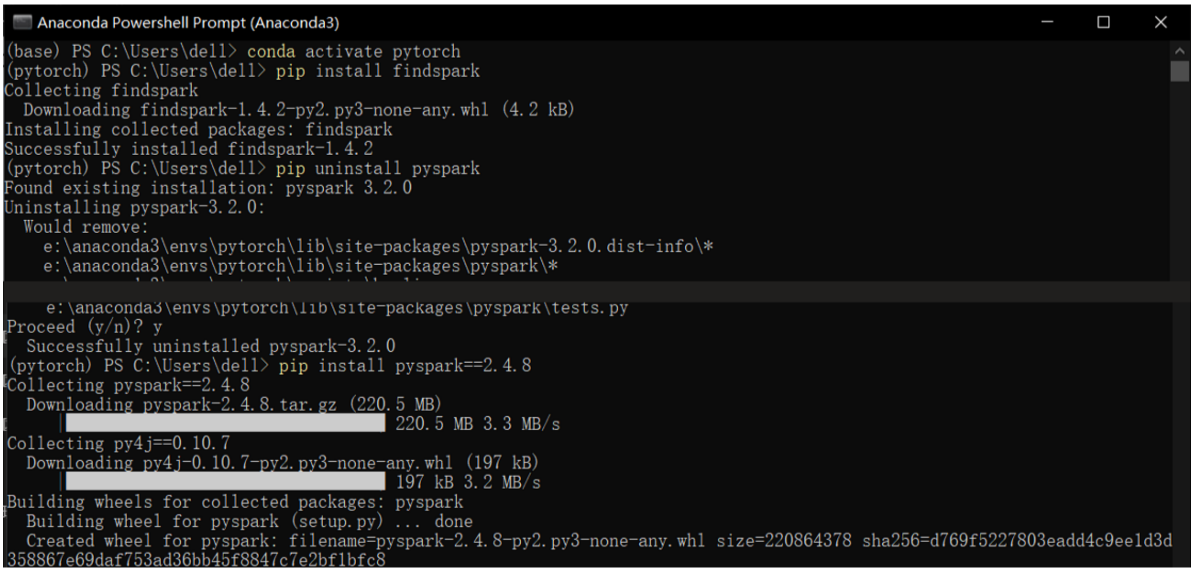
在anaconda的官网上可以找到，findspark是一个寻找spark并让他可导的一个包，然而实际上并没有



这时，报错信息中的Py4J引起了我的注意

> Py4j可以使运行于python解释器的python程序动态的访问java虚拟机中的java对象。Java方法可以像java对象就在python解释器里一样被调用，Java collection也可以通过标准python collection方法调用。Py4j也可以使java程序回调python对象。

最终解决方法：于是事情清晰了起来，原来在Anaconda Powershell Prompt使用pip install pyspark的时候会默认下载最新版的pyspark和py4j，这和我自己装的pyspark版本不匹配，所以需要重新下载对应版本的Py4j，或者重新pip install当前版本pyspark也能获得对应版本的Py4j



其他思考

很不召回的逻辑回归和其他模型

从前面代码实现部分任务四最后的结果来看，使用逻辑回归的分类效果很不理想，于是想到用其他分类模型来尝试

观察任务四的代码，逻辑回归主要是使用的pyspark.ml.classification这个包，于是打开源码来看一下这个包里面还实现了一些什么其他算法，源码可在Anaconda文件夹下的Lib\site-packages\pyspark\ml.classification.py中查看

这个包一共实现了10个分类模型

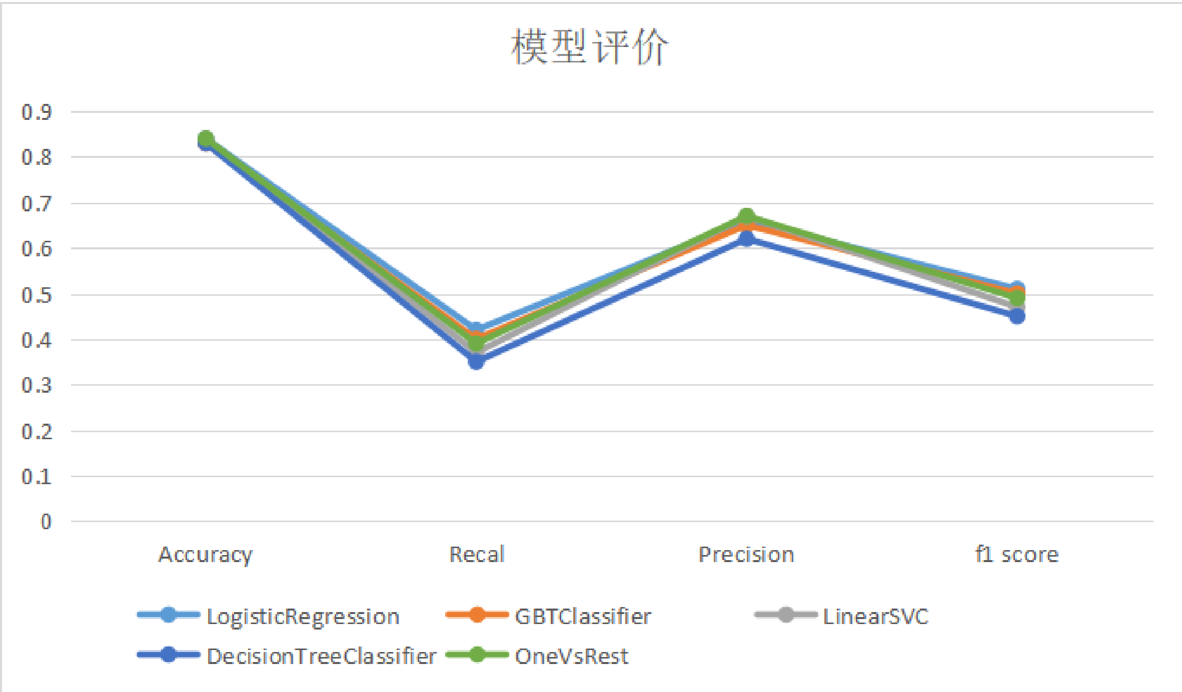
```
__all__ = ['LinearSVC', 'LinearSVCModel',
           'LogisticRegression', 'LogisticRegressionModel',
           'LogisticRegressionSummary', 'LogisticRegressionTrainingSummary',
           'BinaryLogisticRegressionSummary',
'BinaryLogisticRegressionTrainingSummary',
           'DecisionTreeClassifier', 'DecisionTreeClassificationModel',
           'GBTClassifier', 'GBTClassificationModel',
           'RandomForestClassifier', 'RandomForestClassificationModel',
           'NaiveBayes', 'NaiveBayesModel',
           'MultilayerPerceptronClassifier',
'MultilayerPerceptronClassificationModel',
           'OneVsRest', 'OneVsRestModel']
```

首先尝试了GBTClassifier，训练模型花了很久时间，其他各个模型也都跑了一段时间，其中MultilayerPerceptronClassifier时间最久，大约40分钟，模型能训练，但是无法实现预测，总是报错，不知道有没有其他同学能预测出来。

而随机森林把所有样本都预测成了不会违约，这也导致Precision和f1 score无法运算

	GBTClassifier	LinearSVC	RandomForestClassifier	DecisionTreeClassifier	OneVsRest
Accuracy	0.83	0.84	0.80	0.83	0.84
Recal	0.40	0.37	0（那所有样本都预测成了0）	0.35	0.39
Precision	0.65	0.67	\	0.62	0.67
f1 score	0.50	0.47	\	0.45	0.49

把以上模型评价都绘制到一张图中，可以发现，在这个数据集上，各个模型结果差不多。



参考资料

[PySpark - 教程_学习PySpark|WIKI教程 (iowiki.com)]

[pySpark在csv文件中的一些应用 - 知乎 (zhihu.com)]

[Spark MLib 每周一算法 —— LogisticRegression | Spark MLib wctkn (ratlsun.github.io)]