Hive 配置

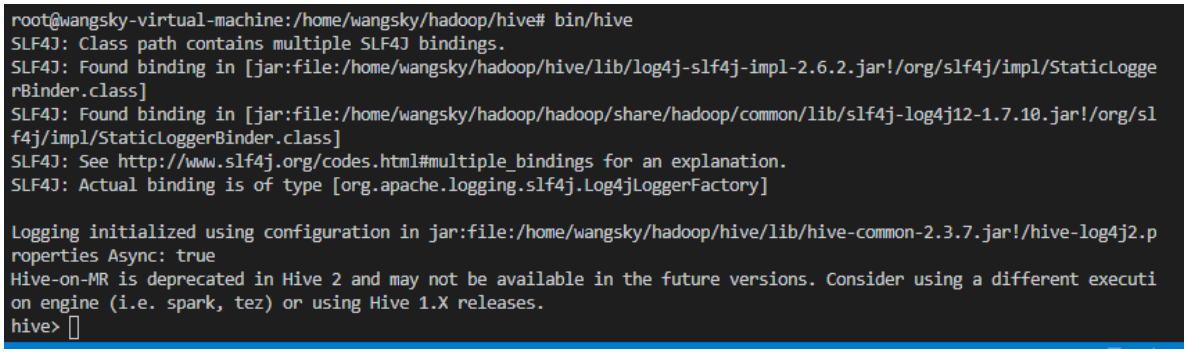
⚠ 操作系统：ubuntu

修改配置，第一次运行hive后报错：

```
Exception in thread "main" java.lang.IllegalArgumentException:
java.net.URISyntaxException: Relative path in absolute URI:
${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D
        at org.apache.hadoop.fs.Path.initialize(Path.java:205)
        at org.apache.hadoop.fs.Path.<init>(Path.java:171)
```

解决办法：将hive-site.xml中的system:java.io.tmpdir全部替换为绝对路径：/home/wangsky/hadoop/hive/tmp

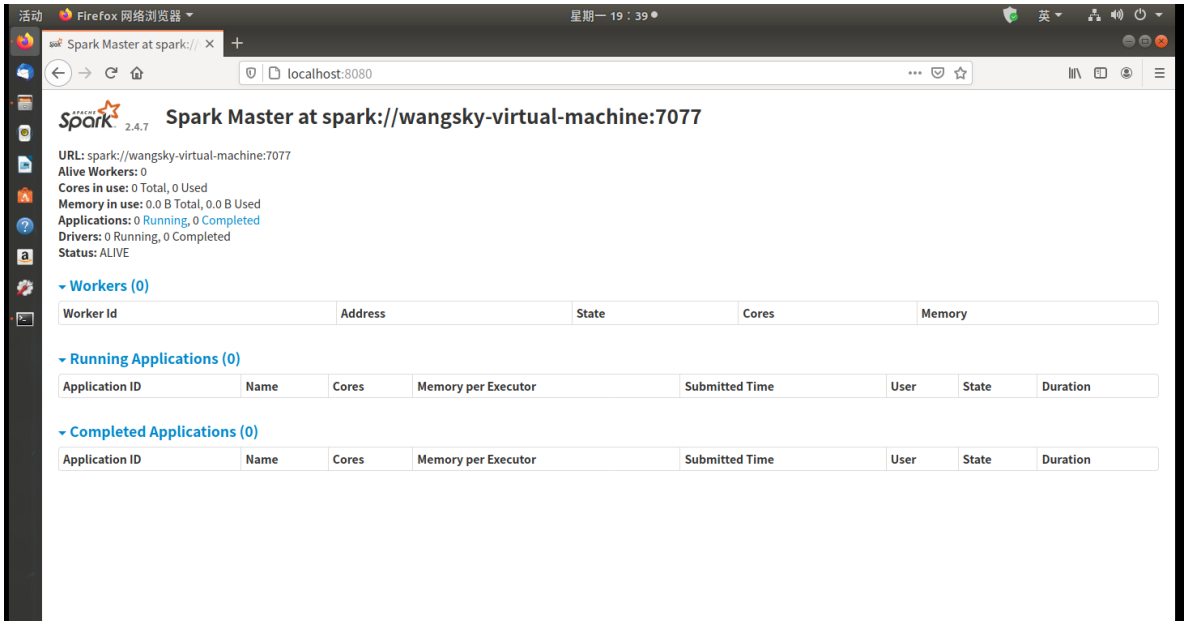
解决之后，hive启动成功：



spark 配置

根据版本对应关系，选择安装spark2.4.7

比较顺利，也没有什么需要特别配置的地方



数据预处理

利用spark-shell将两个表合并，如下：

首先是读取数据：

```
scala> val user_infoDF1=spark.read.format("csv").option("sep",",").option("header","true").load("/home/wangsky/FBDP/user_info_format1.csv")
user_infoDF1: org.apache.spark.sql.DataFrame = [user_id: string, age_range: string ... 1 more field]

scala> val user_logDF1=spark.read.format("csv").option("sep",",").option("header","true").load("/home/wangsky/FBDP/user_log_format1.csv")
user_logDF1: org.apache.spark.sql.DataFrame = [user_id: string, item_id: string ... 5 more fields]
```

然后是sql操作自然连接

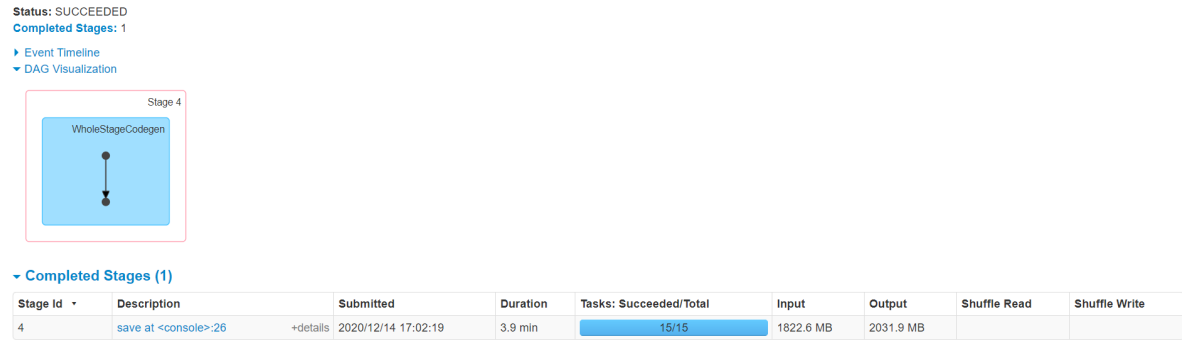
```
scala> user_infoDF1.createOrReplaceTempView("user_info")

scala> user_logDF1.createOrReplaceTempView("user_log")
```

```
scala> val fulldata_DF1=spark.sql("select * from user_info natural join user_log")
20/12/14 13:33:02 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
fulldata_DF1: org.apache.spark.sql.DataFrame = [user_id: string, age_range: string ... 7 more fields]

scala> fulldata_DF1.show()
+-------+---------+------+-------+------+---------+--------+----------+-----------+
|user_id|age_range|gender|item_id|cat_id|seller_id|brand_id|time_stamp|action_type|
+-------+---------+------+-------+------+---------+--------+----------+-----------+
| 328862|        6|     1| 323294|   833|     2882|    2661|      0829|          0|
| 328862|        6|     1| 844400|  1271|     2882|    2661|      0829|          0|
| 328862|        6|     1| 575153|  1271|     2882|    2661|      0829|          0|
| 328862|        6|     1| 996875|  1271|     2882|    2661|      0829|          0|
| 328862|        6|     1|1086186|  1271|     1253|    1049|      0829|          0|
| 328862|        6|     1| 623866|  1271|     2882|    2661|      0829|          0|
| 328862|        6|     1| 542871|  1467|     2882|    2661|      0829|          0|
```

然后进行save操作：这里po一张web ui的截图，确实有点慢，存储一张大表花费了将近4分钟（伪分布式）



Task1-MapReduce

涉及：task1

代码：github仓库 /lab4/lab4_MapReduce

输出：github仓库 /lab4/MP_*（共两个文件）

采用**一次map和一次reduce**完成计数和排序输出工作

数据结构：

> 在reduce中维护了一个**大小为100的堆**，使用了java自带的数据类型PriorityQueue（优先级队列），当堆里面满了时，取出里面的最小元素
>
> 在reduce之后，对堆进行排序，最后输出

其他的就和普通的词频统计没有太大的区别，注意的是，在map中的以下几个条件：

```
//最受欢迎商品
String fields[] = value.toString().split(",");

            if (fields.length != 9) {
                return;
            }
            if (!isNumeric(fields[8])){//去除掉空值或其他违规数据的情况
                return;
            }
            if (Integer.parseInt(fields[8])==0){//如果只是点击，则不计数
                return;
```

```
            }
//=======================
//最受年轻人欢迎商家
String fields[] = value.toString().split(",");

            if (fields.length != 9) {
                return;
            }
            if (!isNumeric(fields[8])){//去除掉空值或其他违规数据的情况
                return;
            }
            if (Integer.parseInt(fields[8])==0){//如果只是点击，则不计数
                return;
            }
            if (!isNumeric(fields[1])){//去除掉空值或其他违规数据的情况
                return;
            }
            if (Integer.parseInt(fields[1])>3){//只保留年龄小于30的
                return;
            }
```

踩到的坑：关于openjdk和jdk的区别

> 在实现堆的时候，我**使用了javafx.util里面的Pair数据类型，但是在linux上编译不通过**
>
> 原因分析：我的window上时Oracle jdk，而linux上安装的是openJDK，两者有一些小小的差别
>
> OracleJDK源码和对应版本的OpenJDK源码进行比较，发现除了文件头的版权注释之外，其余代码基本上都是相同的，只有字体渲染部分存在一点差异，Oracle JDK采用了商业实现，而OpenJDK使用的是开源的FreeType。
>
> 当然，这里的“相同”是建立在两者共有的组件基础上的，Oracle JDK中还会存在一些Open JDK没有的、商用闭源的功能，例如Flight Recorder，OpenJDK中也有少量独有功能。
>
> **javafx.util包在jdk 1.8的类库里面有，但在OpenJDK 8里面是没有的**
>
> 解决办法：自己写了个Pair（反正也比较简单😂）

Task1、2、3、4：Spark

涉及：task1、task2，task3，task4

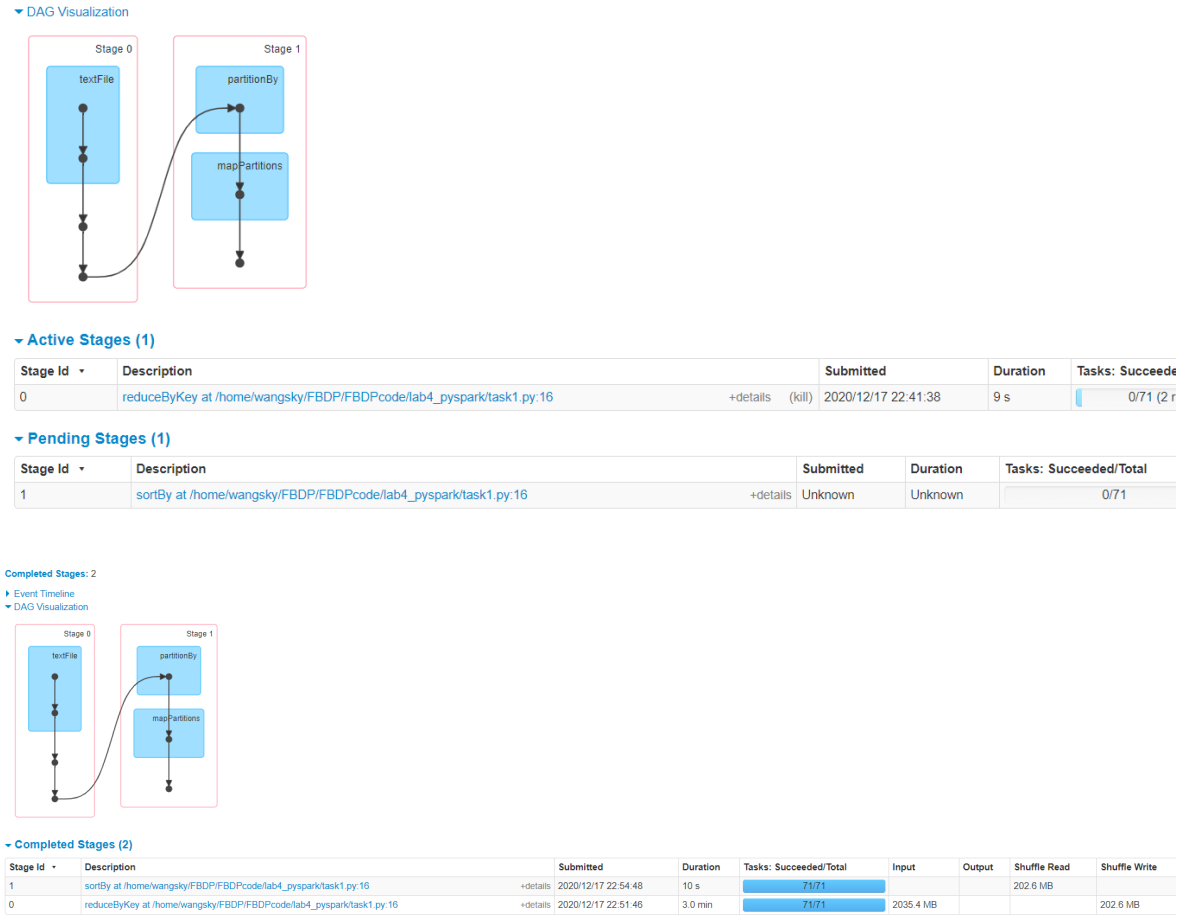
代码：代码：github仓库 /lab4/lab4_pyspark

task1.py：最受欢迎商品和最受年轻人欢迎的商家

代码：github仓库 /lab4/lab4_pyspark/task1.py

输出见：github仓库 /lab4/pyspark_*（共两个文件），与MapReduce输出完全一致

这里简单po几张web ui的截图，但是由于代码运行完毕之后web端口就自动关闭了，所以没有截全



task2.py：spark rdd操作

代码：github仓库 /lab4/lab4_pyspark/task2.py

这里以性别比例为例

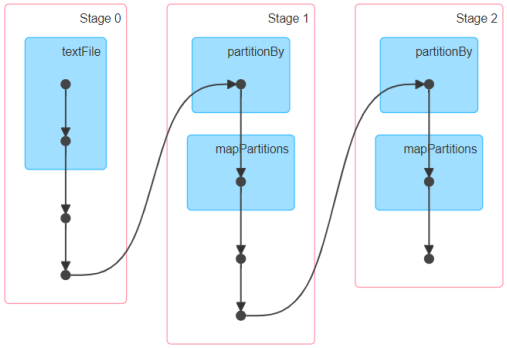
代码的输入是前面数据预处理join了两个表之后的结果，所以直接做map和reduce即可

```
buy_data=d1.filter(lambda x:x.split(',')[8]=='2' and x.split(',')[2]!='2')#所有购买的数据
sex_ratio=buy_data.map(lambda x:(x.split(',')[0],x.split(',')[2]))\
        .reduceByKey(lambda x,y:x)\                              #多次购买只记一次
            .map(lambda x:(x[1],x[0]))\                          #key和value互换，方便做group
                .groupByKey().mapValues(len).collect()           #group并计数
```

Details for Job 0

Status: SUCCEEDED
Completed Stages: 3

▸ Event Timeline
▾ DAG Visualization



▾ **Completed Stages (3)**

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	C
2	collect at /home/wangsky/FBDP/FBDPcode/lab4_pyspark/task2.py:15	+details	2020/12/18 20:06:49	2 s	71/71		
1	groupByKey at /home/wangsky/FBDP/FBDPcode/lab4_pyspark/task2.py:14	+details	2020/12/18 20:06:40	7 s	71/71		
0	reduceByKey at /home/wangsky/FBDP/FBDPcode/lab4_pyspark/task2.py:13	+details	2020/12/18 19:54:30	12 min	71/71	2035.4 MB	

运行结果：

```
wangsky@wangsky-virtual-machine:~$ /usr/bin/python /home/wangsky/FBDP/FBDPcode/lab4_pyspark/task2.py
20/12/18 20:15:44 WARN Utils: Your hostname, wangsky-virtual-machine resolves to a loopback address: 127.0.1.1; using 192.168.91.131 instead (on
)
20/12/18 20:15:44 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
20/12/18 20:15:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
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).
性别分布 [('"', 6436), ('1', 121670), ('0', 285638)]
年龄分布 [('3', 108528), ('"', 2210), ('5', 39461), ('4', 76082), ('1', 24), ('8', 1233), ('7', 6917), ('6', 35112), ('2', 52452), ('0', 91725)]
```

标值为"""的应该就是对应的非合规值（年龄里面一不小心忘记去除0了，不过问题不大，最后计算比例的时候处理即可）

跑了大概15分钟

之后取出非合规值，计算比例：

```
wangsky@wangsky-virtual-machine:~$ /usr/bin/python /home/wangsky/FBDP/FBDPcode/lab4_pyspark
性别分布
   gender   ratio
0       1  121670
1       0  285638
性别比例
   gender     ratio
0       1  0.298717
1       0  0.701283
年龄分布
   age   ratio
0    3  108528
1    5   39461
2    4   76082
3    1      24
4    8    1233
5    7    6917
6    6   35112
7    2   52452
年龄比例
   age     ratio
3    1  0.000075
4    8  0.003855
5    7  0.021629
6    6  0.109791
1    5  0.123389
7    2  0.164010
2    4  0.237898
0    3  0.339353
```

task3.py：spark sql操作

代码：github仓库 /lab4/lab4_pyspark/task3.py

sql的输入是原始数据，而不是上面做了数据预处理的数据

关键是写sql语句，总共分为两步：

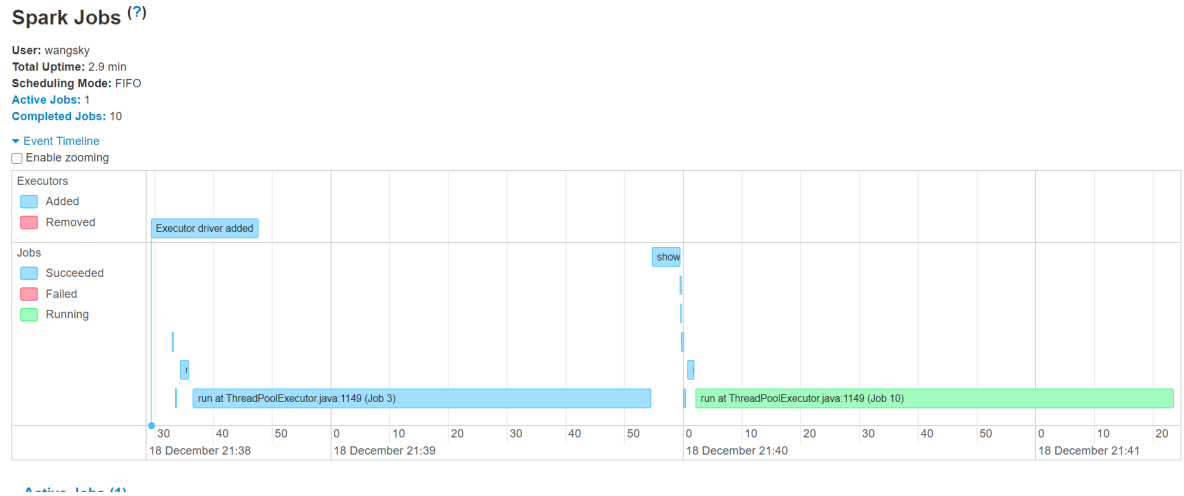
1. 数据过滤

```
data1=spark.sql("select distinct user_id,gender,age_range from user_info natural
join user_log \
        where action_type==\'2\' and gender!=\'2\' and age_range!=\'0\' and
gender!=\'\"\"\' and age_range!=\'\"\"\'")
```

因为前面加了关键字distinct，所以多次购买只计一次

2. 分类计数

跑了大概3分钟




```
wangsky@wangsky-virtual-machine:~$ /usr/bin/python /home/wangsky/FBDP/FBDPcode/lab4_pyspark/t
20/12/18 21:38:27 WARN Utils: Your hostname, wangsky-virtual-machine resolves to a loopback a
)
20/12/18 21:38:27 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
20/12/18 21:38:28 WARN NativeCodeLoader: Unable to load native-hadoop library for your platfo
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).
+------+------+------------------+
|gender|number|           percent|
+------+------+------------------+
|     0|218825| 0.6884753335011327|
|     1| 99015|0.31152466649886734|
+------+------+------------------+

+---------+------+--------------------+
|age_range|number|             percent|
+---------+------+--------------------+
|        7|  6849|0.021548577900830607|
|        3|107826|  0.3392461615907375|
|        8|  1210|0.003806946891517745|
|        5| 39285| 0.12359992449030958|
|        6| 34905| 0.10981940599043544|
|        1|    24|7.550969041026932E-5|
|        4| 75740| 0.23829599798640824|
|        2| 52001| 0.16360747545935062|
+---------+------+--------------------+
```

结果分析：整体排名和用rdd做mapreduce的结果一致，但是总数量有细微差别，主要原因如下：

- 用rdd做mapreduce时，在过滤完数据之后，对重复的user_id统一只取第一个
- 但是在做sql时，是对三元组(user_id,gender,age_range)重复的只取一个，因此可能存在相同user_id，但是gender和age_range不同的出现，导致最后的结果在数量上有细微的差别

task4.py：

代码：github仓库 /lab4/task4.py

首先利用python做了简单的**特征提取**：

```
import sys
from operator import add
import pandas as pd
import os
def trans_action(df):
    dd=pd.DataFrame()
    dd.loc[0,['user_id']]=df.iloc[0,0]
```

```
    dd.loc[0,['seller_id']]=df.iloc[0,5]
    dd.loc[0,['age']]=df.iloc[0,1]
    dd.loc[0,['gender']]=df.iloc[0,2]
    s=[0,0,0,0]
    for i in range(0,len(df)):
        for j in range(0,4):
            if int(df.iloc[i,8])==j:
                s[j]+=1
    dd.loc[0,['0','1','2','3']]=s
    return dd

if __name__ == "__main__":
    path='/home/wangsky/FBDP/fulldata/'
    for name in os.listdir(path):
        print(name)
        print("0%")
        data=pd.read_csv('/home/wangsky/FBDP/fulldata/'+name,sep=',',header=None)
        print("25%")
        data.columns=
['user_id','age','gender','item_id','cat_id','seller_id','brand_id','time','action']

 newdf1=data.groupby(['user_id','seller_id'],as_index=False).apply(trans_action)
        print("50%")
        newdf2=newdf1.join(pd.get_dummies(newdf1.gender,prefix='gender'))
        print("75%")

 newdf2.to_csv('/home/wangsky/FBDP/processed_data/'+name,index=False,header=False)
        print("100%")
```

提取的逻辑如下:

1. 对每个（user，merchant）计数他们的action_type不同类别的数目，即用户在这个商家点击了几次、收藏了几次、购买了几次、加进购物车了几次
2. 由于gender是类别变量，所以做一个one-hot独热编码，而age由于本身具有大小和可比的属性，就不做独热编码，可以直接做特征来使用

然后使用MLLib来做二分类，模型采用支持向量机svm和逻辑回归LR

将数据3、7开分为测试集和训练集，然后做分类器的训练和测试

踩到的坑1：pyspark不支持DataSet

> 由于python不能保证数据类型的安全性，因此pyspark也相应的不支持DataSet
>
> 虽然不支持DataSet，但是仍然可以指定文件读取时的schema，方便后续处理

踩到的坑2：pyspark里面 DescitionTreeModel的predict方法源代码提到

“In Python, predict cannot currently be used within an RDD transformation or action. Call predict directly on the RDD instead.”

```
</pre><pre code_snippet_id="1760790" snippet_file_name="blog_20160713_14_3815520"
name="code" class="python" style="color: rgb(36, 39, 41); font-size: 15px; line-
height: 19.5px;"> def predict(self, x):
        """
```

```
        Predict the label of one or more examples.

        Note: In Python, predict cannot currently be used within an RDD
              transformation or action.
              Call predict directly on the RDD instead.

        :param x:  Data point (feature vector),
                   or an RDD of data points (feature vectors).
        """
        if isinstance(x, RDD):
            return self.call("predict", x.map(_convert_to_vector))

        else:
            return self.call("predict", _convert_to_vector(x))</span>
```

这个call是调用了self._sc方法，导致了model依赖sc

```
class JavaModelWrapper(object):
    """
    Wrapper for the model in JVM
    """
    def __init__(self, java_model):
        self._sc = SparkContext.getOrCreate()
        self._java_model = java_model

    def __del__(self):
        self._sc._gateway.detach(self._java_model)

    def call(self, name, *a):
        """Call method of java_model"""
        return callJavaFunc(self._sc, getattr(self._java_model, name), *a)
```

原因是这里通过py4j来调用java_model（"org.apache.spark.mllib.tree.model.DecisionTreeModel"），导致了依赖SparkContext。

所以做decision tree时不能使用类似于

```
predicedRDD=data.map(lambda x:(x.label,DT.predict(x.feutures)))
```

的操作，因为在map内部不能使用predict

最终跑出来结果如下，SVM表现稍微好一点

```
wangsky@wangsky-virtual-machine:~/FBDP/FBDPcode$ /usr/bin/python /home/wangsky/FBDP/FBDPcode/la
20/12/22 16:10:15 WARN Utils: Your hostname, wangsky-virtual-machine resolves to a loopback add
20/12/22 16:10:15 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
20/12/22 16:10:16 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform
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).
20/12/22 16:11:57 WARN Instrumentation: [608cb84b] Initial coefficients will be ignored! Its di
20/12/22 16:11:58 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.Nativ
20/12/22 16:11:58 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.Nativ
SVM训练集error: 0.06073642920409971
LR训练集error: 0.061066518493252425
SVM测试集error: 0.06268626040489159
LR测试集error: 0.06299455348885007
```