Part II of Project of Spark

Student_ID:16307130214 Name:Liu Baichuan

I. Introduction

- Tasks of this part of Spark Project is using RDD operation to implement some complex statistical work and using pyspark machine package to do some ML tasks. In this part, I learn how to use pyspark machine package and know the difference between ml and mllib.
- My Project runs on Ubuntu16.04 + Spark2.4 + Hadoop2.7 as platform.
- My code firstly writes and runs on Ipython notebook, so it is more convenient for me to paste printscreens in this report. More important, it's my fault to forget to record my code. So in my commit files, you have to run my code in notebook line by line and do not run it as python script in pyspark. I also know if I want to use python script, I have to use SparkConf and SparkContext which are loaded automatically in Ipython.
- Potential application: Have the ability to use large scale distributed system.
- Spark is a great platform to process big data. According to its document[1], Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

II. Q and A

N6. 统计男性、女性中最常用的 5 个名字

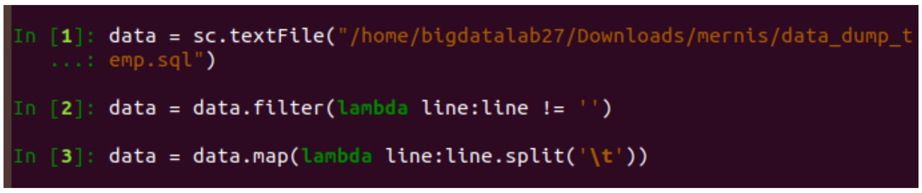
Algorithm：

- Extract name(first name + last name) and gender information
- Register a table
- Use sql GroupBy and OrderBy to find the most common last name respectively

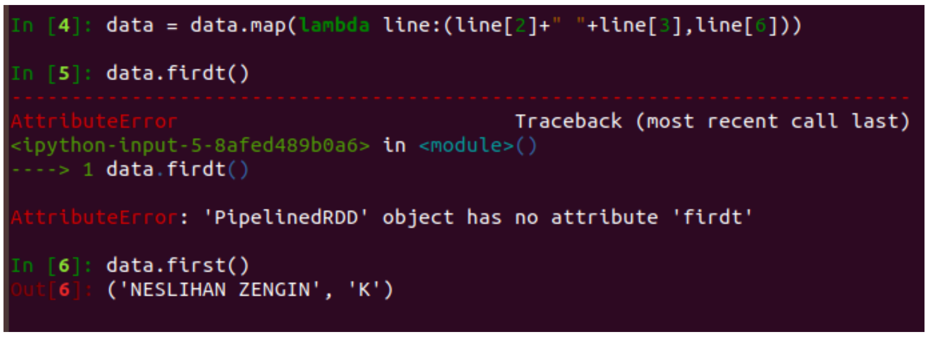
[1] http://spark.apache.org/docs/latest/

Process:

name and gender field

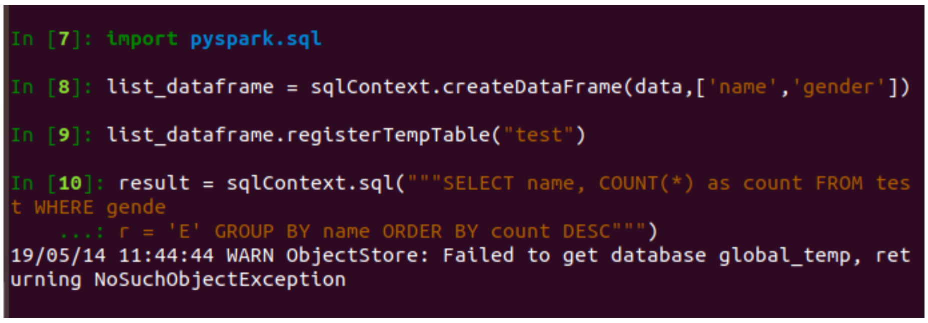


```
In [1]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_t
   ...: emp.sql")

In [2]: data = data.filter(lambda line:line != '')

In [3]: data = data.map(lambda line:line.split('\t'))
```

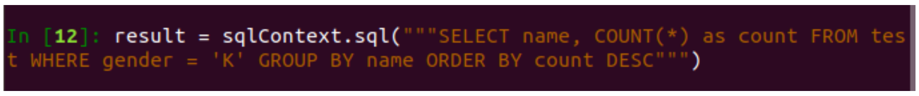


```
In [4]: data = data.map(lambda line:(line[2]+" "+line[3],line[6]))

In [5]: data.firdt()
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-8afed489b0a6> in <module>()
----> 1 data.firdt()

AttributeError: 'PipelinedRDD' object has no attribute 'firdt'

In [6]: data.first()
Out[6]: ('NESLIHAN ZENGIN', 'K')
```

Register table and use sql



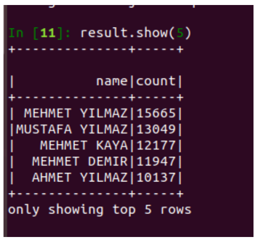
```
In [7]: import pyspark.sql

In [8]: list_dataframe = sqlContext.createDataFrame(data,['name','gender'])

In [9]: list_dataframe.registerTempTable("test")

In [10]: result = sqlContext.sql("""SELECT name, COUNT(*) as count FROM tes
t WHERE gende
    ...: r = 'E' GROUP BY name ORDER BY count DESC""")
19/05/14 11:44:44 WARN ObjectStore: Failed to get database global_temp, ret
urning NoSuchObjectException
```



```
In [12]: result = sqlContext.sql("""SELECT name, COUNT(*) as count FROM tes
t WHERE gender = 'K' GROUP BY name ORDER BY count DESC""")
```

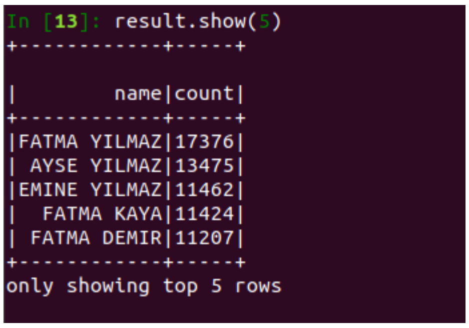
Result:

male



```
In [11]: result.show(5)
+--------------+-----+

|          name|count|
+--------------+-----+
| MEHMET YILMAZ|15665|
|MUSTAFA YILMAZ|13049|
|   MEHMET KAYA|12177|
|  MEHMET DEMIR|11947|
|  AHMET YILMAZ|10137|
+--------------+-----+
only showing top 5 rows
```

female



```
In [13]: result.show(5)
+------------+-----+

|        name|count|
+------------+-----+
|FATMA YILMAZ|17376|
| AYSE YILMAZ|13475|
|EMINE YILMAZ|11462|
|  FATMA KAYA|11424|
| FATMA DEMIR|11207|
+------------+-----+
only showing top 5 rows
```

It is easy to find that MEHMET is the most popular name among male while FATMA is the most popular name among female.

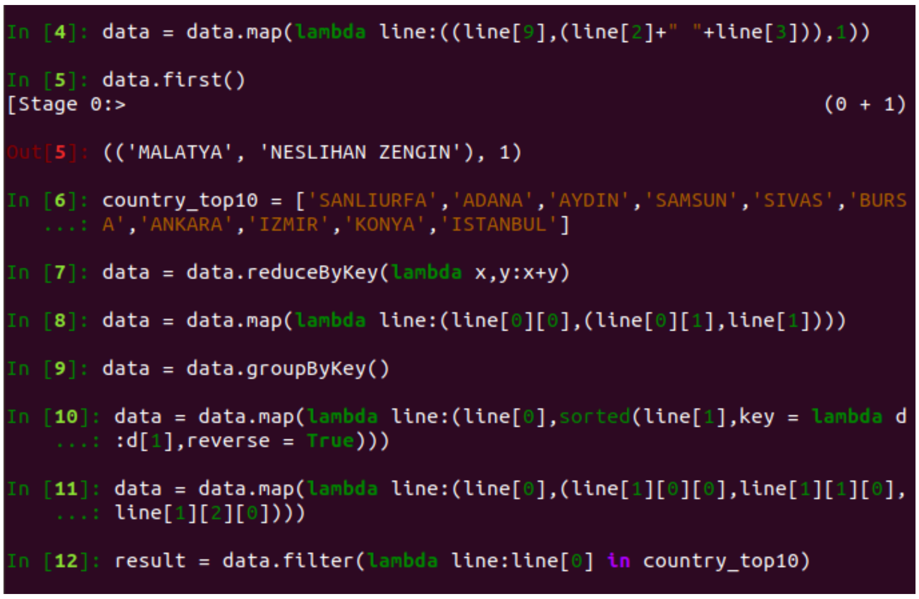
N7. 统计一下该国前 10 大人口城市中，每个城市最受欢迎的 3 个名字是什么

Algorithm:

- Create RDD pair((city,name),1) and count by key
- Change RDD pair ((city, name),count) to (city,(name,count))
- Merge RDD pair by key and sort RDD pair by count in value
- Extract top3 name for top 10 population city(find in N4)

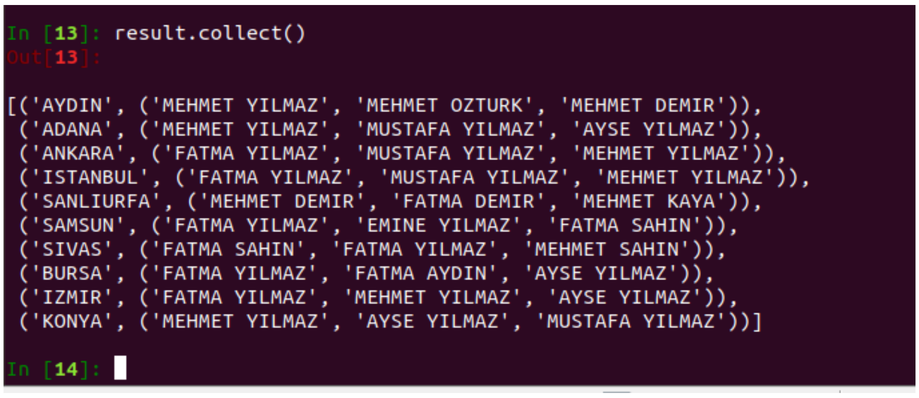
Process:

Get rdd (city,(name,count))



```
In [4]: data = data.map(lambda line:((line[9],(line[2]+" "+line[3])),1))

In [5]: data.first()
[Stage 0:>                                                          (0 + 1)

Out[5]: (('MALATYA', 'NESLIHAN ZENGIN'), 1)

In [6]: country_top10 = ['SANLIURFA','ADANA','AYDIN','SAMSUN','SIVAS','BURS
   ...: A','ANKARA','IZMIR','KONYA','ISTANBUL']

In [7]: data = data.reduceByKey(lambda x,y:x+y)

In [8]: data = data.map(lambda line:(line[0][0],(line[0][1],line[1])))

In [9]: data = data.groupByKey()

In [10]: data = data.map(lambda line:(line[0],sorted(line[1],key = lambda d
    ...: :d[1],reverse = True)))

In [11]: data = data.map(lambda line:(line[0],(line[1][0][0],line[1][1][0],
    ...: line[1][2][0])))

In [12]: result = data.filter(lambda line:line[0] in country_top10)
```

Result:


```
In [13]: result.collect()
Out[13]:

[('AYDIN', ('MEHMET YILMAZ', 'MEHMET OZTURK', 'MEHMET DEMIR')),
 ('ADANA', ('MEHMET YILMAZ', 'MUSTAFA YILMAZ', 'AYSE YILMAZ')),
 ('ANKARA', ('FATMA YILMAZ', 'MUSTAFA YILMAZ', 'MEHMET YILMAZ')),
 ('ISTANBUL', ('FATMA YILMAZ', 'MUSTAFA YILMAZ', 'MEHMET YILMAZ')),
 ('SANLIURFA', ('MEHMET DEMIR', 'FATMA DEMIR', 'MEHMET KAYA')),
 ('SAMSUN', ('FATMA YILMAZ', 'EMINE YILMAZ', 'FATMA SAHIN')),
 ('SIVAS', ('FATMA SAHIN', 'FATMA YILMAZ', 'MEHMET SAHIN')),
 ('BURSA', ('FATMA YILMAZ', 'FATMA AYDIN', 'AYSE YILMAZ')),
 ('IZMIR', ('FATMA YILMAZ', 'MEHMET YILMAZ', 'AYSE YILMAZ')),
 ('KONYA', ('MEHMET YILMAZ', 'AYSE YILMAZ', 'MUSTAFA YILMAZ'))]

In [14]:
```

We may find popular names differ from city to city.

In machine learning(Hard part), I will use Naïve Bayes in machine learning packages of pyspark. In ml and mllib, many classification method only applies for two-category classification. Decision Tree and Naïve Bayes apply for multi-category classification. I do think I will not choose more than 3 features to train model, so Decision Tree is not a good idea. Actually, three machine learning tasks in Hard part ask me to split data as train data, valid data, test data. However, valid data is not so useful for improving model, so I will not use valid for K-cross validation. Tasks ask me to compute topN accuracy. Mllib only provide prediction label while ml provide predict probability, so I have to use ml package. You will see my attempt in H1.

H1. 构建人所在的城市的预测模型：根据给定一个人的所有信息（除了所在城市），预测该人所在的城市。分析该模型 Top1 到 Top5 的准确率

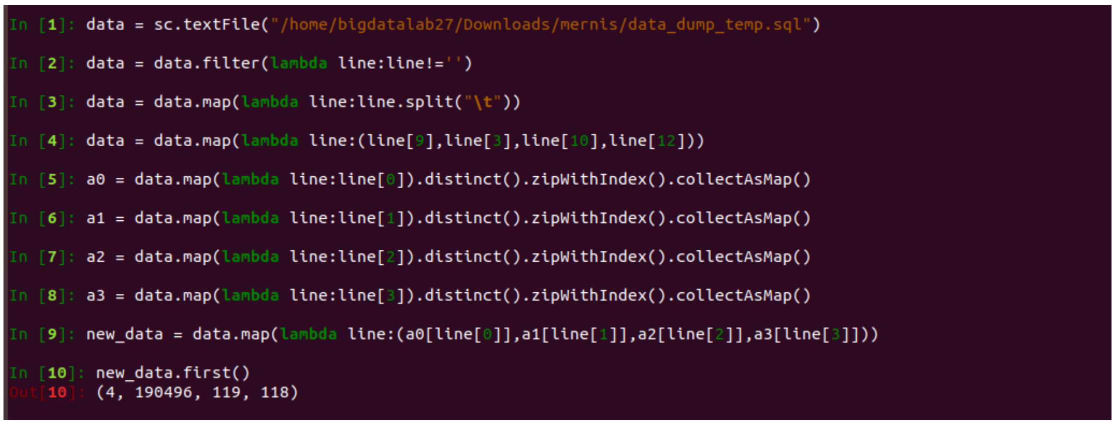
Algorithm：

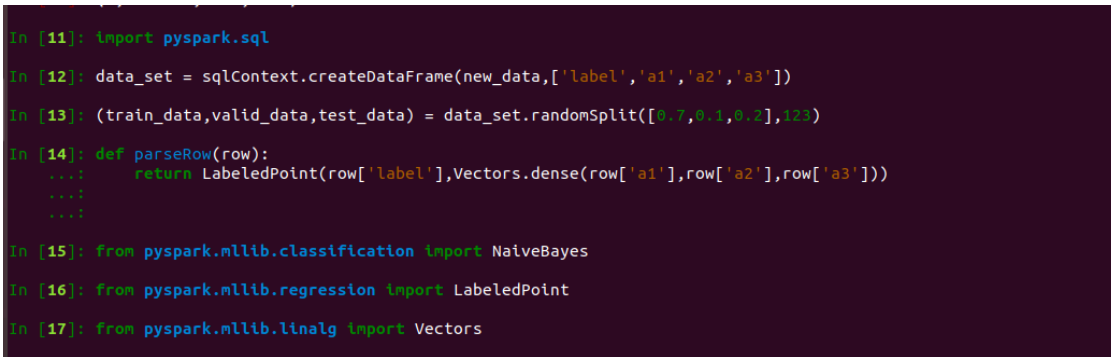
- Choose features
- Change data to specific type which is suitable for ml or mllib
- Use train data to train model
- Use test data to compute accuracy

Process:

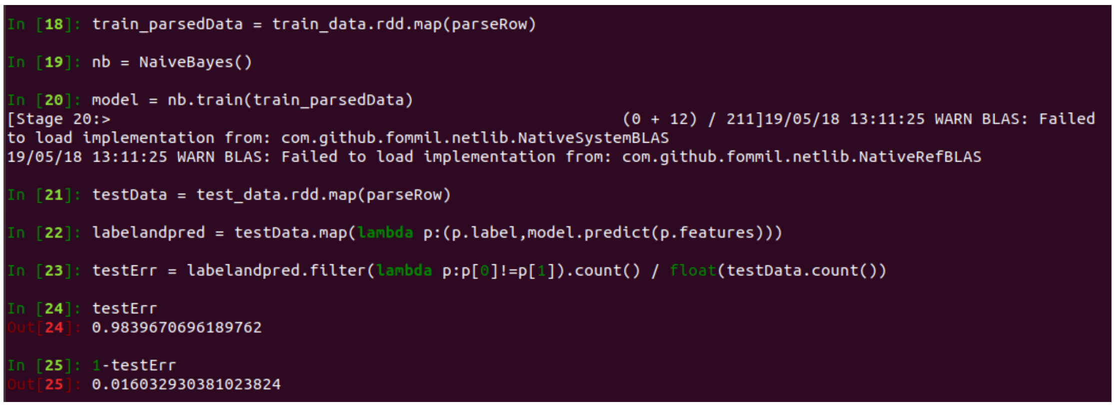
At beginning, I misunderstand this task. I think I can not choose any feature which is about on city, so I choose last_name, id_registration_district, address_district as features.(Also choose a wrong label: id_registration_city, correct label should be address_city)

Use zipWithIndex to transform text features to digital features


```
In [1]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_temp.sql")

In [2]: data = data.filter(lambda line:line!='')

In [3]: data = data.map(lambda line:line.split("\t"))

In [4]: data = data.map(lambda line:(line[9],line[3],line[10],line[12]))

In [5]: a0 = data.map(lambda line:line[0]).distinct().zipWithIndex().collectAsMap()

In [6]: a1 = data.map(lambda line:line[1]).distinct().zipWithIndex().collectAsMap()

In [7]: a2 = data.map(lambda line:line[2]).distinct().zipWithIndex().collectAsMap()

In [8]: a3 = data.map(lambda line:line[3]).distinct().zipWithIndex().collectAsMap()

In [9]: new_data = data.map(lambda line:(a0[line[0]],a1[line[1]],a2[line[2]],a3[line[3]]))

In [10]: new_data.first()
Out[10]: (4, 190496, 119, 118)
```


```
In [11]: import pyspark.sql

In [12]: data_set = sqlContext.createDataFrame(new_data,['label','a1','a2','a3'])

In [13]: (train_data,valid_data,test_data) = data_set.randomSplit([0.7,0.1,0.2],123)

In [14]: def parseRow(row):
    ...:     return LabeledPoint(row['label'],Vectors.dense(row['a1'],row['a2'],row['a3']))
    ...:
    ...:

In [15]: from pyspark.mllib.classification import NaiveBayes

In [16]: from pyspark.mllib.regression import LabeledPoint

In [17]: from pyspark.mllib.linalg import Vectors
```

Definite a parseRow function to construct my label and features as LabeledPoint type which is suitable for mllib. Split the dataset.

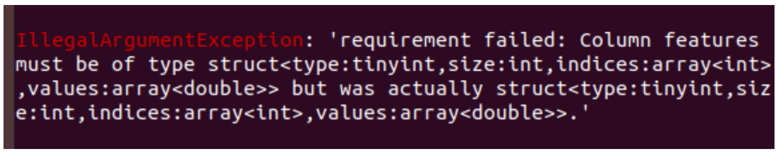

```
In [18]: train_parsedData = train_data.rdd.map(parseRow)

In [19]: nb = NaiveBayes()

In [20]: model = nb.train(train_parsedData)
[Stage 20:>                                                       (0 + 12) / 211]19/05/18 13:11:25 WARN BLAS: Failed
to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
19/05/18 13:11:25 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS

In [21]: testData = test_data.rdd.map(parseRow)

In [22]: labelandpred = testData.map(lambda p:(p.label,model.predict(p.features)))

In [23]: testErr = labelandpred.filter(lambda p:p[0]!=p[1]).count() / float(testData.count())

In [24]: testErr
Out[24]: 0.9839670696189762

In [25]: 1-testErr
Out[25]: 0.016032930381023824
```

Obviously, this model is horrible. So next I will change features as birth_city,id_registration_city. Then I also find mllib can not compute topN. I use ml next. Ml and mllib ask for different data type. At first, I only change my parseRow function.

def parseRow(row):

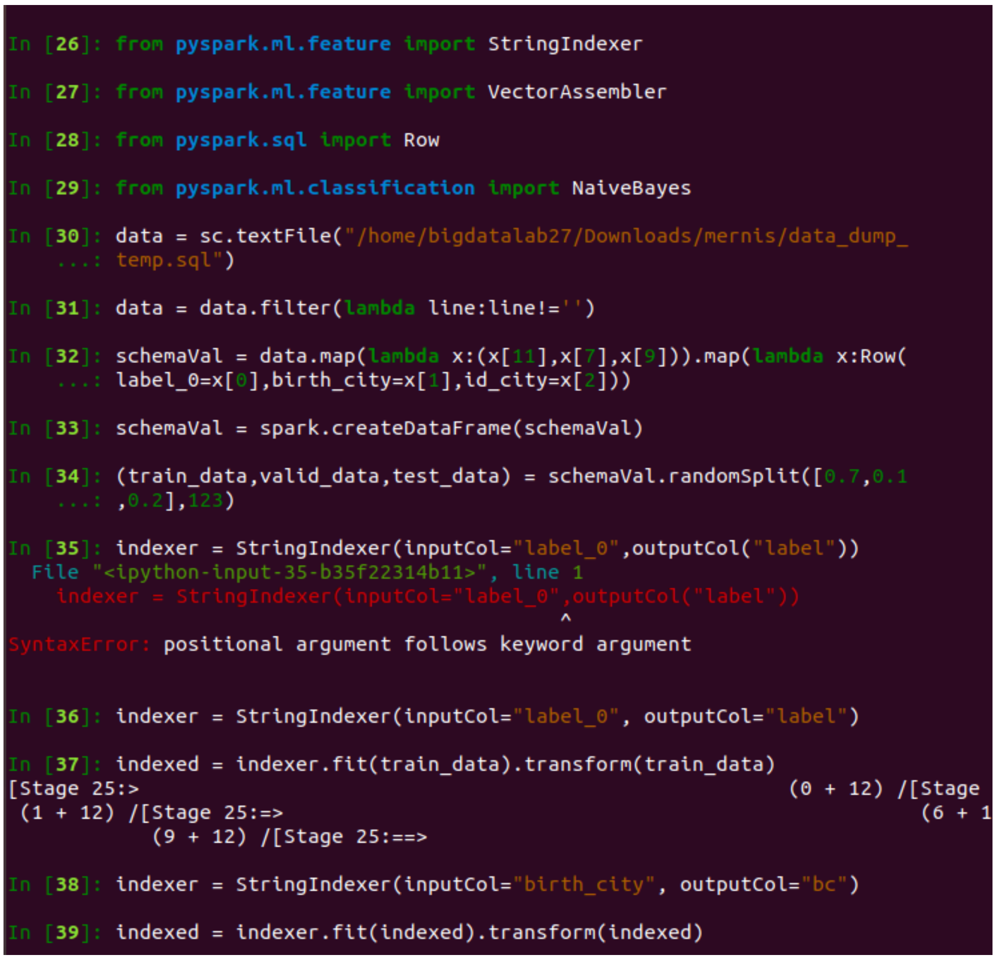
return Row(row['label'],Vectors.dense(row['a1'],row['a2'],row['a3']))

However, I always get this error which I can not solve.



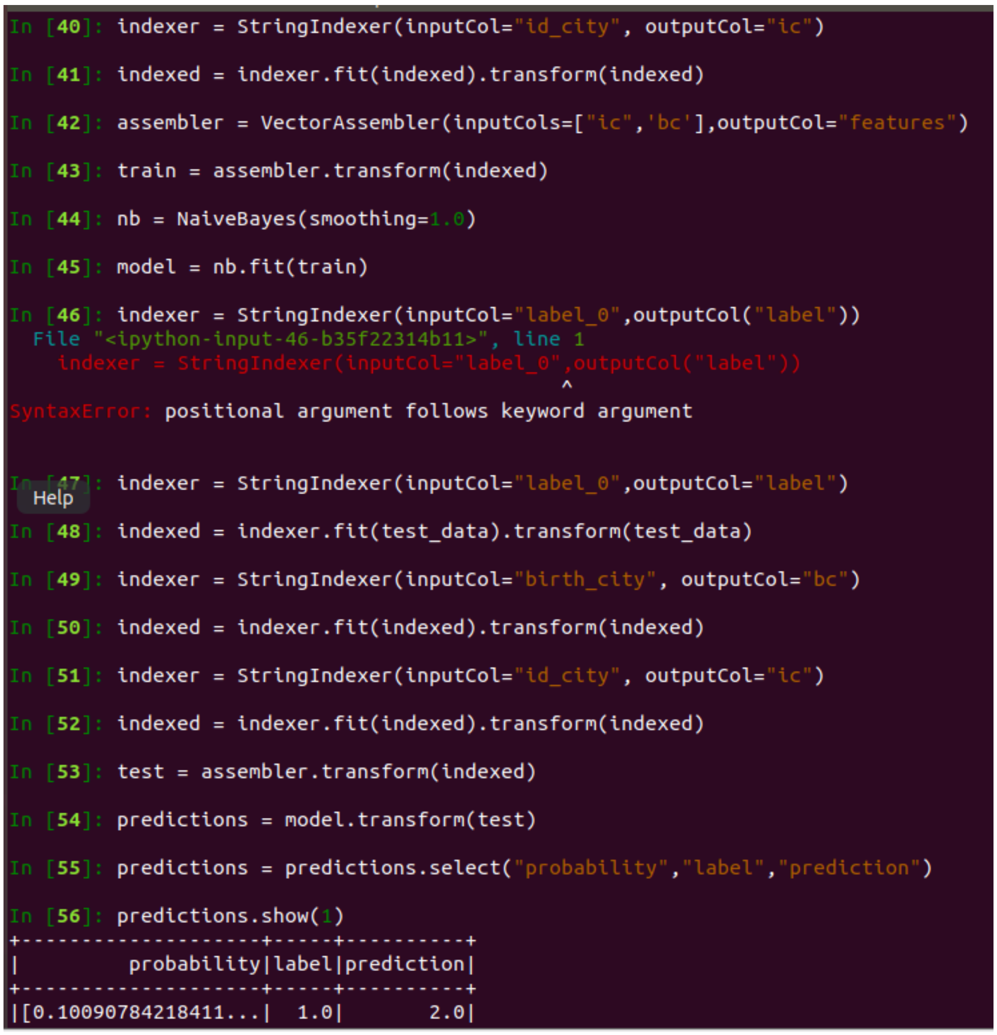
```
IllegalArgumentException: 'requirement failed: Column features
must be of type struct<type:tinyint,size:int,indices:array<int>
,values:array<double>> but was actually struct<type:tinyint,siz
e:int,indices:array<int>,values:array<double>>.'
```

I then completely change my code by using StringIndexer, VectorAssembler for ml.



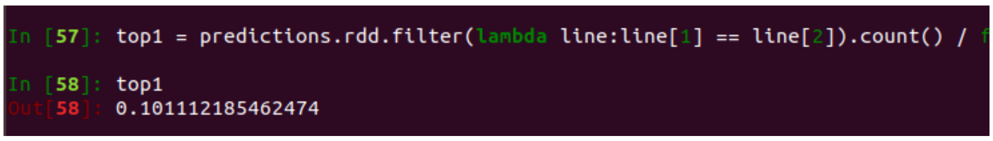
```
In [26]: from pyspark.ml.feature import StringIndexer

In [27]: from pyspark.ml.feature import VectorAssembler

In [28]: from pyspark.sql import Row

In [29]: from pyspark.ml.classification import NaiveBayes

In [30]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_
    ...: temp.sql")

In [31]: data = data.filter(lambda line:line!='')

In [32]: schemaVal = data.map(lambda x:(x[11],x[7],x[9])).map(lambda x:Row(
    ...: label_0=x[0],birth_city=x[1],id_city=x[2]))

In [33]: schemaVal = spark.createDataFrame(schemaVal)

In [34]: (train_data,valid_data,test_data) = schemaVal.randomSplit([0.7,0.1
    ...: ,0.2],123)

In [35]: indexer = StringIndexer(inputCol="label_0",outputCol("label"))
  File "<ipython-input-35-b35f22314b11>", line 1
    indexer = StringIndexer(inputCol="label_0",outputCol("label"))
                                                        ^
SyntaxError: positional argument follows keyword argument


In [36]: indexer = StringIndexer(inputCol="label_0", outputCol="label")

In [37]: indexed = indexer.fit(train_data).transform(train_data)
[Stage 25:>                                                         (0 + 12) /[Stage
 (1 + 12) /[Stage 25:=>                                                       (6 + 1
           (9 + 12) /[Stage 25:==>

In [38]: indexer = StringIndexer(inputCol="birth_city", outputCol="bc")

In [39]: indexed = indexer.fit(indexed).transform(indexed)
```

Choose features and StrngIndexer can transform text features to digital features.



```
In [40]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [41]: indexed = indexer.fit(indexed).transform(indexed)

In [42]: assembler = VectorAssembler(inputCols=["ic",'bc'],outputCol="features")

In [43]: train = assembler.transform(indexed)

In [44]: nb = NaiveBayes(smoothing=1.0)

In [45]: model = nb.fit(train)

In [46]: indexer = StringIndexer(inputCol="label_0",outputCol("label"))
  File "<ipython-input-46-b35f22314b11>", line 1
    indexer = StringIndexer(inputCol="label_0",outputCol("label"))
                                                                 ^
SyntaxError: positional argument follows keyword argument


In [47]: indexer = StringIndexer(inputCol="label_0",outputCol="label")

In [48]: indexed = indexer.fit(test_data).transform(test_data)

In [49]: indexer = StringIndexer(inputCol="birth_city", outputCol="bc")

In [50]: indexed = indexer.fit(indexed).transform(indexed)

In [51]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [52]: indexed = indexer.fit(indexed).transform(indexed)

In [53]: test = assembler.transform(indexed)

In [54]: predictions = model.transform(test)

In [55]: predictions = predictions.select("probability","label","prediction")

In [56]: predictions.show(1)
+--------------------+-----+----------+
|         probability|label|prediction|
+--------------------+-----+----------+
|[0.10090784218411...|  1.0|       2.0|
```

Use train data train Naïve Bayes model. Change test data to suitable struct and use model to predict. Then I find I can use probability for every label. So I can compute topN accuracy.



```
In [57]: top1 = predictions.rdd.filter(lambda line:line[1] == line[2]).count() / f

In [58]: top1
Out[58]: 0.101112185462474
```

Top2 is 20%, top3 is 30%,top4 is 40%,top5 is 50%.

(This part code please refers to H3. H3 and H1 only have different label.)

H2. 姓别预测模型：根据给定的一个人的信息（除了性别），能否给出该人的性别

Algorithm：

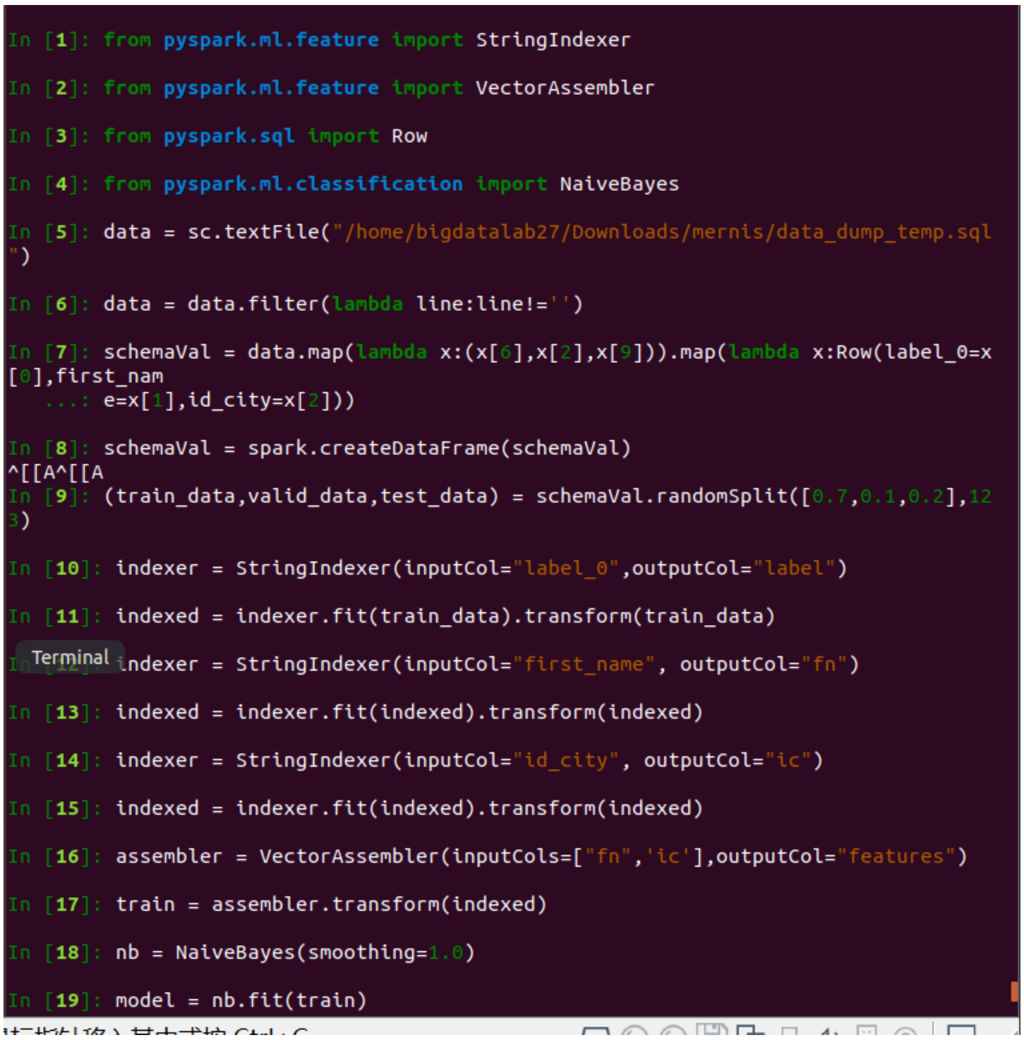
- Choose features
- Change data to specific type which is suitable for ml or mllib

- Use train data to train model
- Use test data to compute accuracy

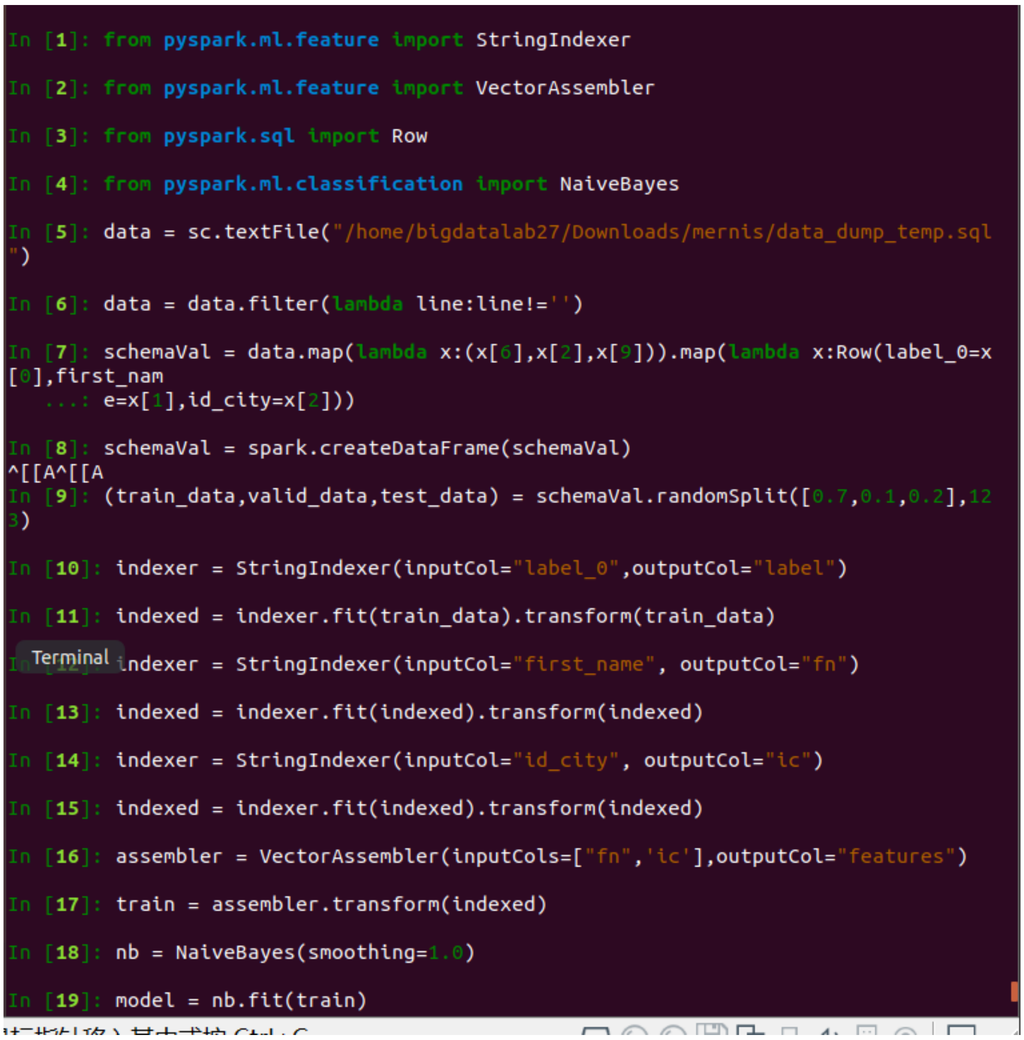
Process:

This tasks is similar to H1 except for label, features and TopN(because labels only have two different values)

I use name and id_registration_city as features.



```
In [1]: from pyspark.ml.feature import StringIndexer

In [2]: from pyspark.ml.feature import VectorAssembler

In [3]: from pyspark.sql import Row

In [4]: from pyspark.ml.classification import NaiveBayes

In [5]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_temp.sql
")

In [6]: data = data.filter(lambda line:line!='')

In [7]: schemaVal = data.map(lambda x:(x[6],x[2],x[9])).map(lambda x:Row(label_0=x
[0],first_nam
   ...: e=x[1],id_city=x[2]))

In [8]: schemaVal = spark.createDataFrame(schemaVal)
^[[A^[[A
In [9]: (train_data,valid_data,test_data) = schemaVal.randomSplit([0.7,0.1,0.2],12
3)

In [10]: indexer = StringIndexer(inputCol="label_0",outputCol="label")

In [11]: indexed = indexer.fit(train_data).transform(train_data)

In [12]: indexer = StringIndexer(inputCol="first_name", outputCol="fn")

In [13]: indexed = indexer.fit(indexed).transform(indexed)

In [14]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [15]: indexed = indexer.fit(indexed).transform(indexed)

In [16]: assembler = VectorAssembler(inputCols=["fn",'ic'],outputCol="features")

In [17]: train = assembler.transform(indexed)

In [18]: nb = NaiveBayes(smoothing=1.0)

In [19]: model = nb.fit(train)
```



```
In [1]: from pyspark.ml.feature import StringIndexer

In [2]: from pyspark.ml.feature import VectorAssembler

In [3]: from pyspark.sql import Row

In [4]: from pyspark.ml.classification import NaiveBayes

In [5]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_temp.sql
")

In [6]: data = data.filter(lambda line:line!='')

In [7]: schemaVal = data.map(lambda x:(x[6],x[2],x[9])).map(lambda x:Row(label_0=x
[0],first_nam
   ...: e=x[1],id_city=x[2]))

In [8]: schemaVal = spark.createDataFrame(schemaVal)
^[[A^[[A
In [9]: (train_data,valid_data,test_data) = schemaVal.randomSplit([0.7,0.1,0.2],12
3)

In [10]: indexer = StringIndexer(inputCol="label_0",outputCol="label")

In [11]: indexed = indexer.fit(train_data).transform(train_data)

In [12]: indexer = StringIndexer(inputCol="first_name", outputCol="fn")

In [13]: indexed = indexer.fit(indexed).transform(indexed)

In [14]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [15]: indexed = indexer.fit(indexed).transform(indexed)

In [16]: assembler = VectorAssembler(inputCols=["fn",'ic'],outputCol="features")

In [17]: train = assembler.transform(indexed)

In [18]: nb = NaiveBayes(smoothing=1.0)

In [19]: model = nb.fit(train)
```

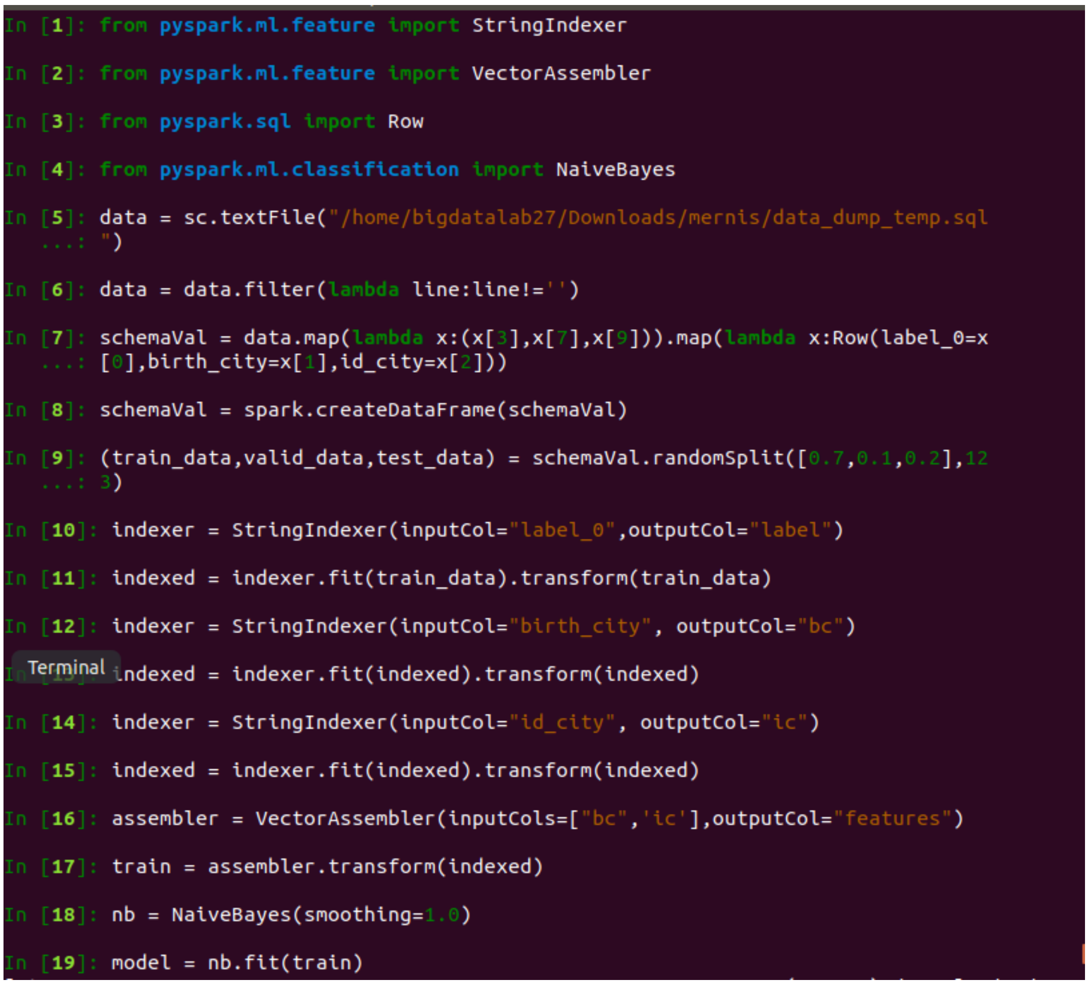
H2. 姓名预测模型：根据给定的一个人的信息（除了性名），能否给出该人的姓氏。分析该模型 TOP1 到 TOP5 的准确率。

Algorithm：

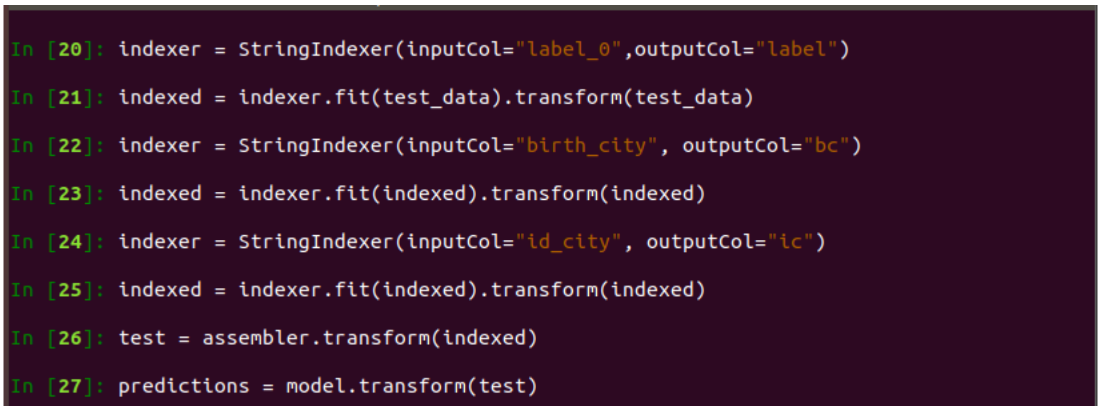
- Choose features
- Change data to specific type which is suitable for ml or mllib
- Use train data to train model
- Use test data to compute accuracy

Process:

This tasks is similar to H1 except for label. TOPN part will be described in this part.

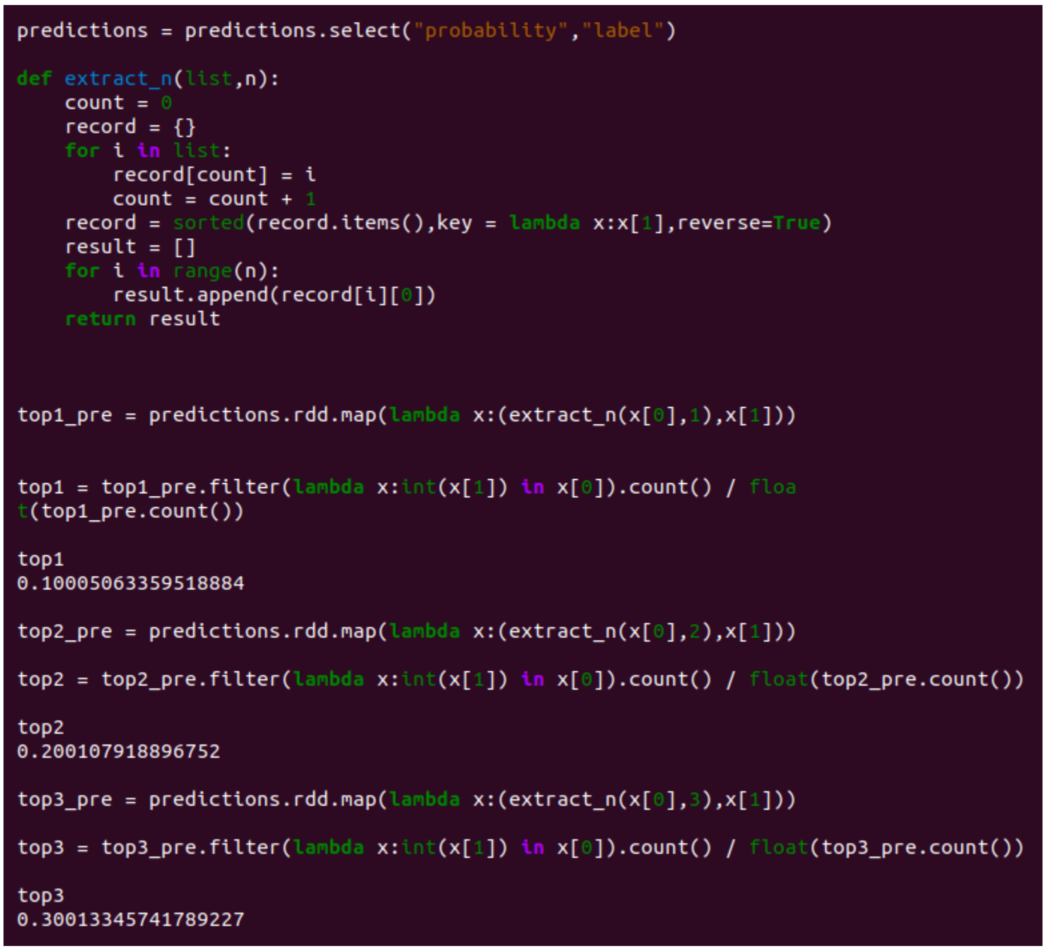


```
In [1]: from pyspark.ml.feature import StringIndexer

In [2]: from pyspark.ml.feature import VectorAssembler

In [3]: from pyspark.sql import Row

In [4]: from pyspark.ml.classification import NaiveBayes

In [5]: data = sc.textFile("/home/bigdatalab27/Downloads/mernis/data_dump_temp.sql
   ...: ")

In [6]: data = data.filter(lambda line:line!='')

In [7]: schemaVal = data.map(lambda x:(x[3],x[7],x[9])).map(lambda x:Row(label_0=x
   ...: [0],birth_city=x[1],id_city=x[2]))

In [8]: schemaVal = spark.createDataFrame(schemaVal)

In [9]: (train_data,valid_data,test_data) = schemaVal.randomSplit([0.7,0.1,0.2],12
   ...: 3)

In [10]: indexer = StringIndexer(inputCol="label_0",outputCol="label")

In [11]: indexed = indexer.fit(train_data).transform(train_data)

In [12]: indexer = StringIndexer(inputCol="birth_city", outputCol="bc")

In [13]: indexed = indexer.fit(indexed).transform(indexed)

In [14]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [15]: indexed = indexer.fit(indexed).transform(indexed)

In [16]: assembler = VectorAssembler(inputCols=["bc",'ic'],outputCol="features")

In [17]: train = assembler.transform(indexed)

In [18]: nb = NaiveBayes(smoothing=1.0)

In [19]: model = nb.fit(train)
```

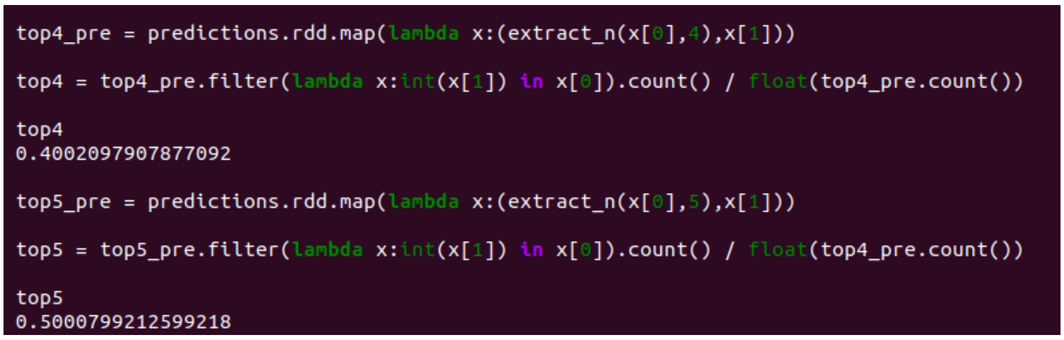


```
In [20]: indexer = StringIndexer(inputCol="label_0",outputCol="label")

In [21]: indexed = indexer.fit(test_data).transform(test_data)

In [22]: indexer = StringIndexer(inputCol="birth_city", outputCol="bc")

In [23]: indexed = indexer.fit(indexed).transform(indexed)

In [24]: indexer = StringIndexer(inputCol="id_city", outputCol="ic")

In [25]: indexed = indexer.fit(indexed).transform(indexed)

In [26]: test = assembler.transform(indexed)

In [27]: predictions = model.transform(test)
```

Only choose label and probability in predictions. Definite a function to extract topN predicted labels, if label shows in them, this prediction will be correct. Function has a parameter N to simplify TopN. I have to mention ml use data frame as data type, when use RDD operation , I have to use .rdd to change data frame to RDD.



```
predictions = predictions.select("probability","label")

def extract_n(list,n):
    count = 0
    record = {}
    for i in list:
        record[count] = i
        count = count + 1
    record = sorted(record.items(),key = lambda x:x[1],reverse=True)
    result = []
    for i in range(n):
        result.append(record[i][0])
    return result



top1_pre = predictions.rdd.map(lambda x:(extract_n(x[0],1),x[1]))


top1 = top1_pre.filter(lambda x:int(x[1]) in x[0]).count() / floa
t(top1_pre.count())

top1
0.10005063359518884

top2_pre = predictions.rdd.map(lambda x:(extract_n(x[0],2),x[1]))

top2 = top2_pre.filter(lambda x:int(x[1]) in x[0]).count() / float(top2_pre.count())

top2
0.200107918896752

top3_pre = predictions.rdd.map(lambda x:(extract_n(x[0],3),x[1]))

top3 = top3_pre.filter(lambda x:int(x[1]) in x[0]).count() / float(top3_pre.count())

top3
0.30013345741789227
```



```
top4_pre = predictions.rdd.map(lambda x:(extract_n(x[0],4),x[1]))

top4 = top4_pre.filter(lambda x:int(x[1]) in x[0]).count() / float(top4_pre.count())

top4
0.4002097907877092

top5_pre = predictions.rdd.map(lambda x:(extract_n(x[0],5),x[1]))

top5 = top5_pre.filter(lambda x:int(x[1]) in x[0]).count() / float(top4_pre.count())

top5
0.5000799212599218
```