In [2]:
```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |████████████████████████████████| 212.4 MB 59 kB/s s eta 0:00:01     |████████
     | 76.5 MB 19.4 MB/s eta 0:00:08     |████████████████                | 195.3 MB 21.3 MB/s eta
0:00:01     |████████████████████████████████| 196.4 MB 21.3 MB/s eta 0:00:01     |████████
     |                                | 200.4 MB 21.3 MB/s eta 0:00:01     |████████████████████████████████| 204.5 MB 21.3
MB/s eta 0:00:01
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 35.0 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=212880768 sha256=26607c
1a5bf9e6aa90cef03eb49318566dde520c2b38fbddf7d3c2c06809b309
  Stored in directory: /root/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863dcd846daaa95f598744e71b
02c77
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.2
WARNING: Running pip as root will break packages and permissions. You should install packages reliabl
y by using venv: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
from pyspark.sql import functions as f
import pandas as pd
from pyspark.sql import DataFrameNaFunctions as DFna
from pyspark.sql.functions import udf, col, when
import matplotlib.pyplot as plt
import pyspark as ps
import os, sys, requests, json
from pyspark.sql.functions import col,size,regexp_replace,lit
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

from pyspark.ml.evaluation import RegressionEvaluator, MulticlassClassificationEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from pyspark.sql import Row
import numpy as np
import math
from pyspark.sql.functions import regexp_replace
from pyspark.sql import SparkSession
```

In [4]:
```
def SparkConfig():
  spark = SparkSession.builder.appName("Elephas_APP")\
  .config("spark.yarn.maxAppAttempts","2")\
  .config("spark.num.executors","50")\
  .config("spark.executor.memory","20g")\
  .config("spark.driver.memory", "16g")\
  .config("spark.memory.offHeap.enabled",True)\
  .config("spark.memory.offHeap.size","16g")\
  .config("spark.executor.resource.gpu.amount",1)\
  .getOrCreate()
  spark.sql("set spark.sql.legacy.timeParserPolicy")
  return spark
```

In [5]:
```
spark =SparkConfig()
sc = spark.sparkContext
```

In [6]:
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

In [7]:
```
pro=spark.read.csv('../input/data-science-for-good-careervillage/professionals.csv', header=True,quote=
'"',sep=",",multiLine=True)
ques=spark.read.csv('../input/data-science-for-good-careervillage/questions.csv', header=True,quote='"'
,sep=",",multiLine=True)
ans=spark.read.csv('../input/data-science-for-good-careervillage/answers.csv', header=True,quote='"',se
p=",",multiLine=True)
ans_score=spark.read.csv('../input/data-science-for-good-careervillage/answer_scores.csv', header=True,
quote='"',sep=",",multiLine=True,inferSchema=True)
```

In [8]:
```
from pyspark.sql.functions import lit,row_number,col
from pyspark.sql.window import Window

w = Window().partitionBy(lit('a')).orderBy(lit('a'))

ques = ques.withColumn("ques_id", row_number().over(w))
pro = pro.withColumn("pro_id", row_number().over(w))
```

In [9]:
```
ans_score_new=ans_score.join(ans, ans_score.id==ans.answers_id,'left').select('answers_author_id','answ
ers_question_id','score')
```

In [10]:
```
ans_score_new=ans_score_new.join(pro,ans_score_new.answers_author_id==pro.professionals_id,'left').sele
ct('pro_id','answers_question_id','score')
```

In [11]:
```
ans_score_new=ans_score_new.join(ques,ans_score_new.answers_question_id==ques.questions_id,'left').sele
ct('pro_id','ques_id','score')
```

In [12]:
```
pro_ques_final=ans_score_new
```

In [13]:
```
pro_ques_final=pro_ques_final.na.drop("any")
```

In [15]:
```
(training, testing) = pro_ques_final.randomSplit([0.8, 0.2])
```

In [16]:
```
#als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", item
Col="ques_id", ratingCol="score")
als=ALS(rank=40,maxIter=15,regParam=0.01,coldStartStrategy="drop",implicitPrefs=False, userCol="pro_id"
, itemCol="ques_id", ratingCol="score")
```

In [17]:
```
model = als.fit(training)
```

In [18]:
```
predictions = model.transform(testing)
```

In [19]:
```
evaluator=RegressionEvaluator(metricName="rmse", labelCol="score", predictionCol="prediction")
accuracy_val = evaluator.evaluate(predictions)

print(f"RMSE:{accuracy_val:.4f}")
```

```
RMSE:1.0013
```

In [20]:
```
model = als.fit(pro_ques_final)
```

In [21]:
```
predictions = model.transform(pro_ques_final)
```

In [22]:
```
def recommendations_for_pro(pro_id):
    print('\nInfo of Professional: ')
    pro.filter(pro.pro_id==pro_id).drop('pro_id').show()
    df=userRecs.filter(userRecs.pro_id==pro_id)
    x=df.select('recommendations').collect()
    df = sc.parallelize(x[0][0]).toDF(['id_ques','value'])
    df=df.join(ques,ques.ques_id==df.id_ques).select('questions_id','questions_title','questions_body')
    print('\nRecommendations: ')
    return df
```

In [23]:
```
def add_recommendations_ques(ques_id):
    proRecs=model.recommendForAllItems(10)
    id=np.array(ques.filter(ques.questions_id==ques_id).select('ques_id').collect())[0][0]
    df=proRecs.filter(proRecs.ques_id==int(id))
    x=df.select('recommendations').collect()
    df = sc.parallelize(x[0][0]).toDF(['id_pro','value'])
    df=df.join(pro,pro.pro_id==df.id_pro).select('professionals_id','professionals_location','professio
nals_industry','professionals_headline','professionals_date_joined')
    return df
```

In [24]:
```
def recommendations_for_ques(ques_id,top):
    id=np.array(ques.filter(ques.questions_id==ques_id).select('ques_id').collect())[0][0]
    df=predictions.filter(predictions.ques_id==int(id)).orderBy("prediction", ascending = False).select
('pro_id','prediction')
    df=df.filter(df.prediction>=0)
    df=df.join(pro,pro.pro_id==df.pro_id).select('professionals_id','professionals_location','professio
nals_industry','professionals_headline','professionals_date_joined').distinct()
    if len(np.array(df.select('professionals_id').collect()))<20:
          df_add=add_recommendations_ques(ques_id)
          df=df.union(df_add)
    df=df.limit(top)
    return df
```

In [25]:
```
def acc_lques(id_ques):
    keywords_recom_df = recommendations_for_ques(id_ques,20)
    list_truth=np.array(ans.filter(ans.answers_question_id==id_ques).select('answers_author_id').collec
t())
    list_recom=np.unique(np.array(keywords_recom_df.select('professionals_id').collect()))
    count=0
    sum_count=ans.filter(ans.answers_question_id==id_ques).count()
    for i in list_recom:
        for j in list_truth:
            if i==j:
                count=count+1
                break
    count=float(count/sum_count)
    return count
```

In [ ]:
```
proRecs=recommendations_for_ques('09e3bdc69a6149aa8656bbc18162ac37',10)
proRecs.toPandas()
```

In [28]:
```
test=['01352c4d67fe435ca59e745ff2520d2a',

'03eee1ca07174470b160717027ab46d6',

'04a979f4e7fd49b9a07b6fae7a5727ee',

'062f49f153de4b8793e4e669ec5b5331',

'083965c88d894a9f9e4e71e521641338',

'09e3bdc69a6149aa8656bbc18162ac37',

'0d7fab391dc145a384da4af0a078b77f',

'0db6ed5d24df42f18d19958ccb32cd6e',

'1a039cb9f3064f76b386f84f303edc43',

'1a444e5e5824446eaf37f31effd72ce0']
sum=0
for t in test:
    x=acc_lques(t)
    sum=sum+x
    print(f"{x:.4f}")
score=float(sum/len(test))
score
```

```
1.0000
1.0000
0.6667
0.7273
0.7500
0.7059
0.6000
0.9000
1.0000
0.5769
```

Out[28]:
```
0.7926744823803646
```

In [ ]:
```

```