In [1]:
```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |▏                               | 212.4 MB 70 kB/s s eta 0:00:01     |▏                               | 59.2 MB 38.1 MB/s eta 0:00:05     |▎                               | 89.7 MB 42.7 MB/s eta 0:00:03
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |▏                               | 198 kB 57.0 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=fe3622
6ec06d21ff8662045c8609e7bcad227df1b0e9f2c3852605d0e5ba58db
  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 [2]:
```
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

spark = SparkSession.builder.master("local[*]").config("spark.executor.memory", "70g").config("spark.dr
iver.memory", "50g").config("spark.memory.offHeap.enabled",True).config("spark.memory.offHeap.size","20
g").appName("sampleCodeForReference").getOrCreate()

sc = spark.sparkContext
```

In [3]:
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

In [4]:
```
pro=spark.read.csv('../input/data-science-for-good-careervillage/professionals.csv', header=True,quote='"',sep=",",multiLine=True)
email=spark.read.csv('../input/data-science-for-good-careervillage/emails.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)
match=spark.read.csv('../input/data-science-for-good-careervillage/matches.csv', header=True,quote='"',sep=",",multiLine=True)
ans=spark.read.csv('../input/data-science-for-good-careervillage/answers.csv', header=True,quote='"',sep=",",multiLine=True)
ans_score=spark.read.csv('../input/data-science-for-good-careervillage/answer_scores.csv', header=True,quote='"',sep=",",multiLine=True)
```

In [5]:
```
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 [6]:
```
pro_ans1=spark.read.csv('../input/data-collaborative1/collaborative_label1.csv', header=True,quote='"',sep=",",multiLine=True,inferSchema=True)
pro_ans0=spark.read.csv('../input/data-collaborative1/collaborative_label0.csv', header=True,quote='"',sep=",",multiLine=True,inferSchema=True)
```

```
---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
<ipython-input-6-2d6f147e9804> in <module>
----> 1 pro_ans1=spark.read.csv('../input/data-collaborative1/collaborative_label1.csv', header=True,quote='"',sep=",",multiLine=True,inferSchema=True)
      2 pro_ans0=spark.read.csv('../input/data-collaborative1/collaborative_label0.csv', header=True,quote='"',sep=",",multiLine=True,inferSchema=True)

/opt/conda/lib/python3.7/site-packages/pyspark/sql/readwriter.py in csv(self, path, schema, sep, enco
ding, quote, escape, comment, header, inferSchema, ignoreLeadingWhiteSpace, ignoreTrailingWhiteSpace,
 nullValue, nanValue, positiveInf, negativeInf, dateFormat, timestampFormat, maxColumns, maxCharsPerC
olumn, maxMalformedLogPerPartition, mode, columnNameOfCorruptRecord, multiLine, charToEscapeQuoteEsca
ping, samplingRatio, enforceSchema, emptyValue, locale, lineSep, pathGlobFilter, recursiveFileLookup,
 modifiedBefore, modifiedAfter, unescapedQuoteHandling)
    735             path = [path]
    736         if type(path) == list:
--> 737             return self._df(self._jreader.csv(self._spark._sc._jvm.PythonUtils.toSeq(path)))
    738         elif isinstance(path, RDD):
    739             def func(iterator):

/opt/conda/lib/python3.7/site-packages/py4j/java_gateway.py in __call__(self, *args)
   1303         answer = self.gateway_client.send_command(command)
   1304         return_value = get_return_value(
-> 1305             answer, self.gateway_client, self.target_id, self.name)
   1306
   1307         for temp_arg in temp_args:

/opt/conda/lib/python3.7/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
    115                 # Hide where the exception came from that shows a non-Pythonic
    116                 # JVM exception message.
--> 117                 raise converted from None
    118             else:
    119                 raise

AnalysisException: Path does not exist: file:/kaggle/input/data-collaborative1/collaborative_label1.csv
```

In [ ]:
```
pro_ques_final = pro_ans1.union(pro_ans0)
```

In [ ]:
```
pro_ques_final
```

In [ ]:
```
pro_ques_final=pro_ques_final.na.drop(subset=["check"])
```

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

In [ ]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
als = ALS(maxIter=2, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_id", itemCol="ques_id", ratingCol="check")
model = als.fit(training)

# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(pro_ques_final)
```

In [ ]:
```
pred=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('check','pred')
```

In [ ]:
```
pred_final=predictions.withColumn('pred', f.when(f.col('prediction') > 0.5, "1").otherwise("0")).select('pro_id','ques_id','check','pred','prediction')
```

In [ ]:
```
from pyspark.sql.types import DoubleType
pred = pred.withColumn("pred", pred["pred"].cast(DoubleType()))
```

In [ ]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(labelCol="check",metricName="accuracy",predictionCol="pred")
accuracy_val = evaluator.evaluate(pred)

print(f"Accuracy:{accuracy_val*100:.5f}%")
```

In [ ]:
```
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 [ ]:
```
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','professionals_industry','professionals_headline','professionals_date_joined')
    return df
```

In [ ]:
```
proRecs=model.recommendForAllItems(10)
id=np.array(ques.filter(ques.questions_id=='01352c4d67fe435ca59e745ff2520d2a').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','value')
```

In [ ]:
```
def recommendations_for_ques(ques_id,top):
    id=np.array(ques.filter(ques.questions_id==ques_id).select('ques_id').collect())[0][0]
    df=pred_final.filter(pred_final.ques_id==int(id)).orderBy("pred", ascending = False).select('pro_id','pred')
    df=df.join(pro,pro.pro_id==df.pro_id).select('professionals_id','professionals_location','professionals_industry','professionals_headline','professionals_date_joined').distinct()
    if len(np.array(df.select('professionals_id').collect()))<10:
        df_add=add_recommendations_ques(ques_id)
        df=df.union(df_add).distinct()
    df=df.limit(top)
    return df
```

In [ ]:
```
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').collect())
    list_recom=np.array(keywords_recom_df.select('professionals_id').collect())
    count=0
    #num_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/10)
    return count
```

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

In [ ]:
```
def score(test):
    sum=0
    for t in test:
        x=acc_lques(t)
        sum=sum+x
    score=float(sum/len(test))
    return score
```

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

'03eeelca07174470b160717027ab46d6',

'04a979f4e7fd49b9a07b6fae7a5727ee',

'062f49f153de4b8793e4e669ec5b5331',

'083965c88d894a9f9e4e71e521641338',

'09e3bdc69a6149aa8656bbc18162ac37',

'0d7fab391dc145a384da4af0a078b77f',

'0db6ed5d24df42f18d19958ccb32cd6e',

'1a039cb9f3064f76b386f84f303edc43',

'1a444e5e5824446eaf37f31effd72ce0']
```

In [ ]:
```
a = score()
```