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

```
Collecting pyspark
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |[redacted]| 212.4 MB 64 kB/s s eta 0:00:01     |[redacted]
     | 36.8 MB 13.2 MB/s eta 0:00:14     |[redacted]| 41.5 MB 13.2 MB/s eta 0:
00:13     |[redacted]| 46.0 MB 41.7 MB/s eta 0:00:04     |[redacted]
     | 47.8 MB 41.7 MB/s eta 0:00:04     |[redacted]| 61.2 MB 41.7 MB/s e
ta 0:00:04     |[redacted]| 68.8 MB 41.7 MB/s eta 0:00:04
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |[redacted]| 198 kB 45.0 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=b3aa16
0fa72c372b2a090e940fc974b5a7a135712c2d3f317cd4bfa371f1e4ca
  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]:
```
import numpy as np
import pandas as pd
from pyspark.sql import functions as f
from operator import add
from pyspark.ml.feature import RegexTokenizer, CountVectorizer, Tokenizer
from pyspark.ml.feature import StopWordsRemover, VectorAssembler
from pyspark.ml.feature import Word2Vec, Word2VecModel
from pyspark.ml.feature import IDF

from pyspark.ml import Pipeline, PipelineModel


from pyspark.sql.functions import *

from pyspark.sql.types import *

import folium
import html
```

In [4]:
```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

In [5]:
```
#Data cleaning question
def datacleaning_ques(test):
    col_name='questions_body'
    user_regex=r"(@\w{1,15})"
    clean_test=test.withColumn( 'user_mentioned',f.array_remove(
    f.array(
    f.regexp_extract(f.col(col_name),user_regex,1),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,2)]),2),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,3)]),3),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,4)]),4),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,5)]),5),
    f.regexp_extract(
    f.col(col_name), "".join([f"{user_regex}.*?" for i in range(0,6)]),6),), "",).alias('user_mentione
d'))
    clean_test=clean_test.withColumn( 'original_text',f.col(col_name)).withColumn(col_name,f.regexp_rep
lace(f.col(col_name), user_regex,"").alias(col_name))
    # remove hastag
    hashtag_user_regex="#(\w{1,})"
    clean_test=clean_test.withColumn( 'hashtags',f.array_remove(
    f.array(
    f.regexp_extract(f.col(col_name),hashtag_user_regex,1),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,2)]),2),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,3)]),3),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,4)]),4),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,5)]),5),
    f.regexp_extract(
    f.col(col_name), "".join([f"{hashtag_user_regex}.*?" for i in range(0,6)]),6),), "",).alias('hashta
gs'))
    clean_test=clean_test.withColumn(col_name,f.regexp_replace(f.col(col_name), hashtag_user_regex,"$1"
).alias(col_name))
    url_regex=r"((https?|ftp|file):\/{2,3})+([-\w+&@#/%=~|$?!:,.]*)|(www.)+([-\w+&@#/%=~|$?!:,.]*)"
    clean_test=clean_test.withColumn( col_name,f.regexp_replace(f.col(col_name), url_regex,""))

    email_regex=r"[\w.-]+@[\w.-]+\.[a-zA-Z]{1,}"
    clean_test=clean_test.withColumn(col_name,f.regexp_replace(f.col(col_name), email_regex,""))
    #clean_test=clean_test.withColumn('text', html_unescape("text"))

    test_cleaned=(clean_test.withColumn( col_name,f.regexp_replace(f.col(col_name),"[^a-zA-Z]"," ")).wi
thColumn(col_name,f.regexp_replace(f.col(col_name)," +"," ")).withColumn( col_name,f.trim(f.col(col_nam
e))).filter(col(col_name) !=''))
    test_data=test_cleaned.select('professionals_id',col_name).coalesce(2).cache()
    return test_data
```

In [6]:
```
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)
```

Collaborative

In [7]:
```
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 [8]:
```
ans_score_new=ans_score.join(ans, ans_score.id==ans.answers_id,'left').select('answers_author_id','answ
ers_question_id','score')
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')
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')
pro_ques_collar=ans_score_new
pro_ques_collar=pro_ques_collar.na.drop("any")
```

In [9]:
```
(training, test) = pro_ques_collar.randomSplit([0.8, 0.2])
```

In [11]:
```
als = ALS(rank=40,maxIter=15, regParam=0.01,coldStartStrategy="drop",implicitPrefs=False,userCol="pro_i
d", itemCol="ques_id", ratingCol="score")
```

In [12]:
```
model = als.fit(pro_ques_collar)
```

In [13]:
```
predictions = model.transform(pro_ques_collar)
```

In [14]:
```
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 [15]:
```
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 [29]:
```
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()))<15:
        df_add=add_recommendations_ques(ques_id)
        df=df.union(df_add)
    df=df.limit(top)
    return df
```

In [17]:
```
def acc_1ques_collar(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.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
```

Content

In [18]:
```
pro_ans=pro.join(ans,pro.professionals_id==ans.answers_author_id).select('professionals_id','answers_qu
estion_id','answers_id')
pro_ans=pro_ans.join(ans_score,pro_ans.answers_id==ans_score.id).select('professionals_id','answers_que
stion_id','score')
pro_ans=pro_ans.join(ques,pro_ans.answers_question_id==ques.questions_id).select('professionals_id','qu
estions_body','questions_title','score')
pro_ans=pro_ans.withColumn('combined_ques',concat(pro_ans.questions_title,lit(' '),pro_ans.questions_bo
dy))
pro_ques_content=pro_ans.groupBy('professionals_id').agg(f.collect_list('combined_ques').alias('questio
ns_body'))
```

In [19]:
```
from pyspark.sql.functions import udf, col
join_udf = udf(lambda x: ",".join(x))
pro_ques_content=pro_ques_content.withColumn("questions_body", join_udf(col("questions_body")))
```

In [20]:
```
pro_ques_content=datacleaning_ques(pro_ques_content)
pro_ques_content=pro_ques_content.na.drop(subset=["questions_body"])
```

In [21]:
```
# Build the pipeline
tokenizer = RegexTokenizer(gaps = False, pattern = '\w+',inputCol = 'questions_body', outputCol = 'toke
n')
stopWordsRemover = StopWordsRemover(inputCol = 'token', outputCol = 'nostopwrd')
word2Vec = Word2Vec(vectorSize = 100, minCount = 5, inputCol = 'nostopwrd', outputCol = 'word_vec', see
d=123)
pipeline = Pipeline(stages=[tokenizer, stopWordsRemover, word2Vec])

# fit the model
pipeline_mdl = pipeline.fit(pro_ques_content)
```

In [22]:
```
# transform the question data
ques_pipeline_df = pipeline_mdl.transform(pro_ques_content)
```

In [23]:
```
def CosineSim(vec1, vec2):
    return np.dot(vec1, vec2) / np.sqrt(np.dot(vec1, vec1)) / np.sqrt(np.dot(vec2, vec2))
```

In [24]:
```
all_ques_vecs = ques_pipeline_df.select('professionals_id', 'word_vec').rdd.map(lambda x: (x[0], x[1]))
.collect()
```

In [25]:
```
def getQuestionDetails(in_ques):

    a = in_ques.alias("a")
    b = pro.alias("b")

    return a.join(b, col("a.professionals_id") == col("b.professionals_id"), 'inner') \
            .select([col('a.score')]+[col('b.'+xx) for xx in b.columns] )
```

In [26]:
```
def getKeyWordsRecoms(key_words, sim_bus_limit):

    input_words_df = sc.parallelize([(0, key_words)]).toDF(['professionals_id', 'questions_body'])

    # transform the the key words to vectors
    input_words_df = pipeline_mdl.transform(input_words_df)

    # choose word2vec vectors
    input_key_words_vec = input_words_df.select('word_vec').collect()[0][0]

    # get similarity
    sim_bus_byword_rdd = sc.parallelize((i[0], float(CosineSim(input_key_words_vec, i[1]))) for i in al
l_ques_vecs)

    sim_bus_byword_df = spark.createDataFrame(sim_bus_byword_rdd) \
         .withColumnRenamed('_1', 'professionals_id') \
         .withColumnRenamed('_2', 'score') \
         .orderBy("score", ascending = False)

    sim_bus_byword_df=sim_bus_byword_df.na.drop(subset=["score"])

    # return top 10 similar
    a = sim_bus_byword_df.limit(sim_bus_limit)

    return getQuestionDetails(a).select('professionals_id','professionals_location','professionals_indu
stry','professionals_headline','professionals_date_joined')
```

In [30]:
```
def acc_1ques_final(id_ques):
    key_words = str(np.array(ques.filter(ques.questions_id==id_ques).select('questions_body').collect
())[0][0])
    recom_df1 = getKeyWordsRecoms(key_words, 5)
    recom_df2 = recommendations_for_ques(id_ques,15)
    keywords_recom_df=recom_df1.union(recom_df2).distinct()
    list_truth=np.array(ans.filter(ans.answers_question_id==id_ques).select('answers_author_id').collec
t())
    list_recom=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 [31]:
```
test=['01352c4d67fe435ca59e745ff2520d2a',

'03eee1ca07174470b160717027ab46d6',

'04a979f4e7fd49b9a07b6fae7a5727ee',

'062f49f153de4b8793e4e669ec5b5331',

'083965c88d894a9f9e4e71e521641338',

'09e3bdc69a6149aa8656bbc18162ac37',

'0d7fab391dc145a384da4af0a078b77f',

'0db6ed5d24df42f18d19958ccb32cd6e',

'1a039cb9f3064f76b386f84f303edc43',

'1a444e5e5824446eaf37f31effd72ce0']
sum=0
for t in test:
    x=acc_1ques_final(t)
    sum=sum+x
    print(x)
score=float(sum/len(test))
score
```

```
1.0
1.0
0.8888888888888888
0.9090909090909091
0.6666666666666666
0.5882352941176471
0.8
0.75
1.0
0.5769230769230769
```

Out[31]:
```
0.8179804835687188
```

In [ ]: