Baruch College

CIS 4130 Big Data Technologies

Individual Semester Project

Safiie Bulhakova

[Safiie.bulhakova@baruchmail.cuny.edu](mailto:Safiie.bulhakova@baruchmail.cuny.edu)

Milestone 1: Proposal

Instagram Influencer Dataset: Category Classified

The whole dataset is divided into four parts (four files): post\_info.txt, json\_files.zip, profiles\_influencers.zip and profiles\_brands.zip. The dataset contains post\_info.txt with the list of 1,601,074 Instagram posts; each line represents a post and is composed of four columns of information. ([Post ID], [USER name], [Sponsorship label], [JSON file], [Image files]). The dataset contains 38,113 Instagram influencers in profiles\_influencers.zip file, who are classified into nine categories: beauty, family, fashion, fitness, food, interior, pet, travel. This file contains the number of followers, following, number of posts, bio, email, phone and URL that each influencer have. The dataset contains json\_files.zip file with 1,601,074 Instagram posts with various information of post metadata, such as captions, likes, comments, timestamps, sponsorship, usertags, etc. The dataset contains profiles\_brands.zip which contains Instagram profiles of the 25,282 brands. Also, the whole dataset that I was able to find contains also img\_file.zip file, but I decided to use only json files of post metadata for my project.

For the project I plan to use only post metadata that include JSON files of 37GB. It will be very interesting to model and predict many things by those attributes that this dataset has. I intend to predict the category (of the influencer by number of followers, number of followings, number of likes, number of comments, and number of posts. Also, I will try to visualize category name vs post number, follower count by following number, category number by average number of likes normalized by number of followers and number of comments by number of likes.

A link to a dataset: <https://sites.google.com/site/sbkimcv/dataset#h.4eo4r5p70z10>.

Milestone 2: Data Acquisition

In order to fetch files from Google Drive, there is a utility called 'gdrive'. First, I downloaded the gdrive\_2.1.1\_linux\_386.tar.gz file from GitHub to EC2 instance:

$ curl - SLO https://github.com/prasmussen/gdrive/releases/download/2.1.1

/gdrive\_2.1.1\_linux\_386.tar.gz. Then I de-compressed and Un-tared the file with following commands:

$ gzip -d gdrive\_2.1.1\_linux\_386.tar.gz and $ tar -xf gdrive\_2.1.1\_linux\_386.tar. Also, I changed the file permissions so it would execute (run) with a $ chmod a+x gdrive command. For my project I needed to download four files and in order to do that, first, I had to take needed file’s ID.

For post\_info.txt file, ID is: 1zzKv8Tt3IAkaZ6UfLaUlwM7qk5LMx0Q. For json\_files.zip, ID is: 10m2vzWMZcSDmzyTbFYi4zadk8Yq16T0d.

For profiles\_influencers.zip, ID is:  1RnhDr\_GY7bwiQEI9XEW9ZOsHOwp-1Kq0.

For profiles\_brands.zip, ID is: 1OQbkhgcOAKv1brPE50QhTw6TpwWsgMAo.

Then, in order to download files, I used the gdrive utility. So, for post\_info.txt file I used command:

./gdrive download 1zzKv8Tt3IAkaZ6UfLaUlwM7qk5LMx0Q

The first time I did that, the Authentication needed, and I proceeded to this URL: https://accounts.google.com/o/oauth2/auth?access\_type=offline&client\_id=367116221053- 7n0vf5akeru7on6o2fjinrecpdoe99eg.apps.googleusercontent.com& redirect\_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&response\_type=code&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive&state=state.

On the web browser of this URL, I clicked the button to Allow and then copied the Authorization code. I entered verification code from the link: 4/1ARtbsJr6Gk7IA8tGtOwQr6PoEZ6LUgyHwPTHzIEwluDgeTiNqVzAVximZkA. Then the file download had proceeded. After that I continued downloading other files with same commands.

For json\_files.zip:

./gdrive download 10m2vzWMZcSDmzyTbFYi4zadk8Yq16T0d.

For profiles\_influencers.zip:

./gdrive download 1RnhDr\_GY7bwiQEI9XEW9ZOsHOwp-1Kq0.

And for profiles\_brands.zip:

./gdrive download 1OQbkhgcOAKv1brPE50QhTw6TpwWsgMAo

After that I created a bucket with command:

aws s3api create-bucket --bucket project-data-sb --region us-east-2 --create-bucket-configuration LocationConstraint=us-east-2

In order to load directly to AWS S3 I used the --stdout option.

So, for the post\_info.txt file:

./gdrive download –stdout 1zzKv8Tt3IAkaZ6UfLaUlwM7qk5LMx0Q- | aws s3 cp - s3://project-data-sb/post\_info.txt

For the json\_files.zip:

./gdrive download --stdout 10m2vzWMZcSDmzyTbFYi4zadk8Yq16T0d | aws s3 cp - s3://project-data-sb/json\_files.zip

For the profiles\_influencers.zip:

./gdrive download --stdout 1RnhDr\_GY7bwiQEI9XEW9ZOsHOwp-1Kq0 | aws s3 cp - s3://project-data-sb/profiles\_influencers.zip

For the profiles\_brands.zip:

./gdrive download --stdout 1OQbkhgcOAKv1brPE50QhTw6TpwWsgMAo | aws s3 cp - s3://project-data-sb/profiles\_brands.zip

At the end, I checked my new bucket with files with command:

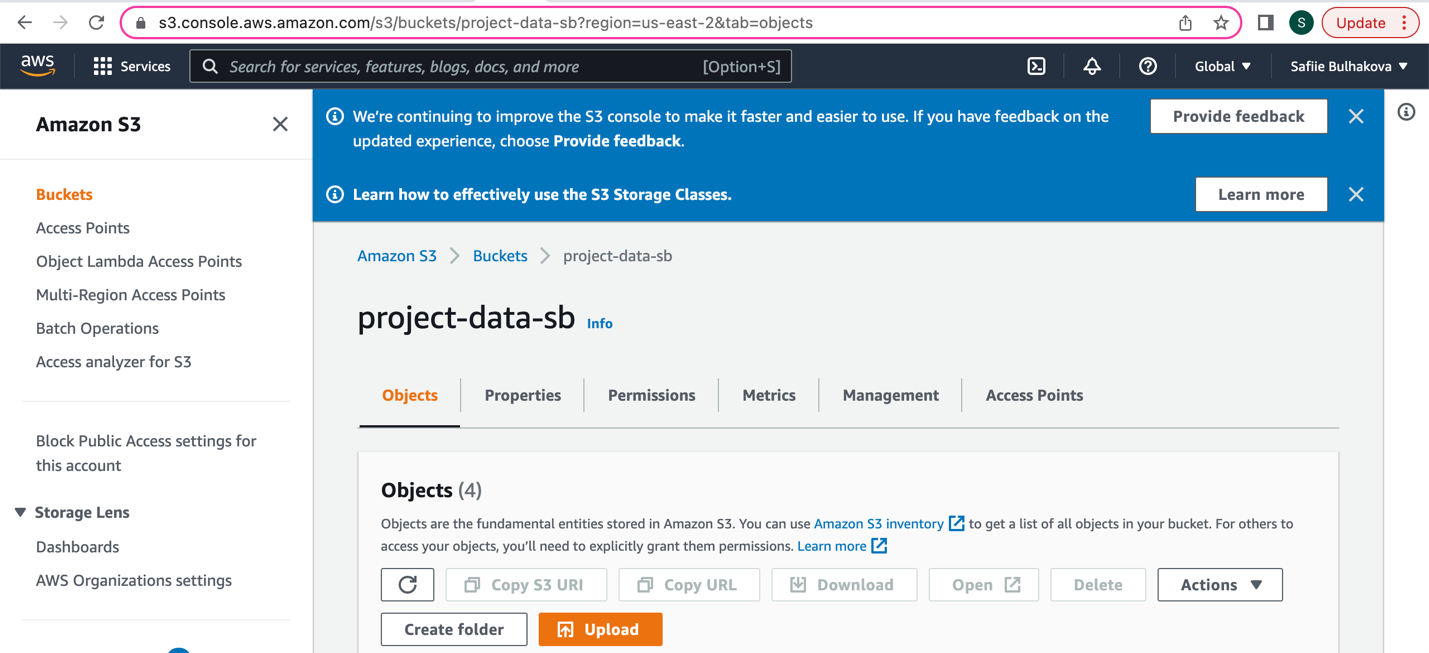
aws s3 ls s3://project-data-sb

Screenshot 1 (Code on AWS CLI):

Text

Description automatically generated

Screenshot 2 (Amazon S3 Bucket: “project-data-sb”):



Screenshot 3 (Four objects of “project-data-sb” bucket):

Graphical user interface, text, application

Description automatically generated

Milestone 3: Exploratory Data Analysis

Using json\_file.zip, and after looking through it and seeing what is inside of every file, I decided to find statistics on: 1) height and width of each picture that each file has; 2) number of likes that each post has; 3) number of comments that each post has and 4) the quantity of words (if any) of each post.

To do that I wrote a code on “jsonstatistics.py”:

import zipfile

import boto3

import pandas as pd

import json

from io import BytesIO

bucket="project-data-sb"

zipfile\_to\_unzip="json\_files.zip"

s3\_client = boto3.client('s3', use\_ssl=False)

s3\_resource = boto3.resource('s3')

zip\_obj = s3\_resource.Object(bucket\_name=bucket, key=zipfile\_to\_unzip)

buffer = BytesIO(zip\_obj.get()["Body"].read())

z = zipfile.ZipFile(buffer)

df = pd.DataFrame(columns=['height','width','likes','comments', 'caption\_word\_count'])

counter = 0

for filename in z.namelist()[1:]:

# Added this break here because otherwise it takes too long to run. I will try to run for entire dataset over the weekend

if counter == 10000:

break

data = json.load(open(filename))

caption\_list = data['edge\_media\_to\_caption']['edges']

caption\_word\_count = 0

if caption\_list:

caption\_word\_count = len(caption\_list[0]['node']['text'].split())

df = df.append({'height': data['dimensions']['height'], 'width': data['dimensions']['width'], 'likes': data['edge\_media\_preview\_like']['count'],'comments': data['edge\_media\_to\_comment']['count'], 'caption\_word\_count': caption\_word\_count}, ignore\_index=True)

counter = counter +1

print(df.astype(float).describe())

Apparently, there is really big number of files and as I mentioned in the comment inside the code, after trying to run the code of all files, and realizing that it takes really long to run, I decided to do and count statistics of the first 10,000 files.

So, here is screenshot of Count, Mean, Std, Min, 25%, 50%, 75%, and Max for “height”, “width”, “likes”, “comments” and “caption\_word\_count” statistics after running jsonstatistics.py:

Graphical user interface, text

Description automatically generated

Using post\_info.txt I decided to find statistics of how many posts each user made, in the file. So, after grouping by the same name of the first and second column (which were unnamed) I wrote a code, and it showed me the number of posts each user made and then statistics of it on “infostatistics.py”:

import boto3

import pandas as pd

df = pd.read\_csv('s3://project-data-sb/post\_info.txt', header=None, sep='\t')

df2 = df.groupby([1])[1].count()

print(df2)

print(df2.describe())

So, first print is showing how many times each username (user) appears on the file (post\_info.txt), which means how many posts each user made. The second print shows statistics of all these users; Count, Mean, Std, Min, 25%, 50%, 75%, and Max.

Text

Description automatically generated

Milestone 4: Coding and Modeling

I plan to predict the category of the influencer by user ‘s number of followers, number of followings, number of likes, number of comments, and number of posts.

First, I tried to read the zipped folder in S3 bucket with pyspark. I tried to find a way to do it, but now I don’t think it’s possible. Then I realized I actually needed to unzip my folders with my data in my EC2 instance and upload them all to S3 bucket so that my pyspark code could read the files. Unzipping the post information files took a very long time because there are about 1.6 million files. If everything would work as I was planning to everything would be done by Friday. However, I faced even more challenges later on.

Then, I tried to read all the necessary data from s3 bucket to pyspark dataframes. Reading all the 1.6 million json files into a single dataframe was constantly running into memory issues. I tried updating spark configuration to increase executor and driver memory, but it did not help. Then, I tried upgrading instance types and having more workers, but it also didn’t work, it was running into the same memory issues. Every attempt instance changes, and cluster start was taking also quite a long time.

Then, I decided to read and process the json file in batches, since I could then ignore most of the data in each batch and only extract what I needed, thus saving memory.

In my script I have a for loop that reads and transforms 10k files at a time. Reading and processing this data for every iteration of the loop took about 20-30 minutes. So, overall with my types of instances and memory I knew it will take hours and hours. I started to run this script on Saturday, but I didn’t add the logic to write processed data back to S3 and on Sunday morning saw that 500k files were processed, but as I tried to write the processed data to S3, my session got logged out and I lost all of the processed data. At this point I added logic to write the processed data back to S3 with each iteration of the loop and ran it again. I knew that reading the data would take a long time that is why I stopped the script when it wrote 100k files. Then I wrote the script to read this processed data from S3 further transforming it by splitting features and label and removing rows with null data.

I then used random forest classifier to train and test on the data. For my features I used number of followers, number of followings, number of likes, number of comments, and number of posts. I was predicting the category of the influencer. I got 74% accuracy and I think it would be much higher if I would be able to train on all the data, I have in S3.

So, overall, if I knew beforehand that pyspark would not be able to process such many files from the data that I have, I would firstly, never used this type of data, I would find one big csv file which it could run from first try even if it was larger in size instead of having 1.6 millions of small files that was really hard to process in pyspark, and secondly, if I knew beforehand that I will face so many problems and constantly would try to understand not only how to do the project, but most importantly what to fix for more than 4-5 days in a row non-stop, I would start predicting only 100k of files from the start, because what I wouldn’t try, nothing worked in order to process all of the 1.6 mil files of post\_info files.

First part of the code:

from pyspark.sql.functions import input\_file\_name, split, col, concat, lit, when

# my attempts at trying to increase memory usage for driver and executors

# new\_conf = spark.sparkContext.getConf().setAll([('spark.executor.memory', '20g'),('spark.driver.memory','30g'), ('spark.executor.memory','30g')])

# spark = SparkSession.builder.config(conf=new\_conf).getOrCreate()

# reading txt file to help with joining between post files and profile files(json\_files.zip)

df1 = spark.read.text('s3a://project-data-sb/post\_info.txt').select(split('value', '\t')[1].alias('account\_name'), split('value', '\t')[3].alias('json\_file'))

data = df1.select(concat(lit('s3a://project-data-sb/post\_files/'),col('json\_file')).alias("full\_path")).collect()

# reading first 10k json files, transforming and writing back to s3

print("reading at range 0")

file\_list = data[0:10000]

file\_list\_mapped = list(map(lambda x: x.full\_path, file\_list))

df = spark.read.json(file\_list\_mapped, multiLine=True).withColumn("input\_file\_name",input\_file\_name()).select(col("edge\_media\_preview\_like.count").alias("like\_count"), col("edge\_media\_to\_comment.count").alias("comment\_count"), split("input\_file\_name", '/')[4].alias('json\_file'))

df.write.parquet("s3a://project-data-sb/parquet/data.parquet")

# reading 10k files at a time, I stopped at 100k however because it took about 20 minutes to read/transform each 10k files

for index in range(10000,1601074,10000):

print("reading at range "+str(index))

file\_list = data[index:index+10000]

file\_list\_mapped = list(map(lambda x: x.full\_path, file\_list))

print("created mapped list")

df = spark.read.json(file\_list\_mapped, multiLine=True).withColumn("input\_file\_name",input\_file\_name()).select(col("edge\_media\_preview\_like.count").alias("like\_count"),col("edge\_media\_to\_comment.count").alias("comment\_count"),split("input\_file\_name",'/')[4].alias('json\_file'))

df.write.mode('append').parquet("s3a://project-data-sb/parquet/data.parquet")

# reading entire post dataset

df = spark.read.parquet("s3a://project-data-sb/parquet/data.parquet")

# reading/transforming entire profile dataset

df3 = spark.read.text('s3://project-data-sb/profile\_files/').withColumn("input\_file\_name",input\_file\_name()).select(split('value', '\t')[1].alias('follower\_num'), split('value', '\t')[2].alias('following\_num'), split('value', '\t')[3].alias('post\_num'), split('value', '\t')[6].alias('category\_num'), split("input\_file\_name", '/')[4].alias('account\_name'))

# performing a couple joins to get all data in one dataframe

first\_join = df1.join(df3, df1.account\_name == df3.account\_name, 'inner')

second\_join = df.join(first\_join, first\_join.json\_file == df.json\_file, 'inner')

# dropping unnecessary columns

final\_df = final\_df.drop(df1.account\_name, df3.account\_name, first\_join.json\_file, df.json\_file)

# transforming category column to be integer instead of string

final\_df = second\_join.withColumn("category\_num\_2",

when(col("category\_num")=='Creators & Celebrities', 1)

.when(col("category\_num")=='Non-Profits & Religious Organizations', 2)

.when(col("category\_num")=='Publishers', 3)

.when(col("category\_num")=='Personal Goods & General Merchandise Stores', 4)

.when(col("category\_num")=='Business & Utility Services', 5)

.when(col("category\_num")=='General Interest', 6)

.when(col("category\_num")=='Transportation & Accomodation Services', 7)

.when(col("category\_num")=='Content & Apps', 8)

.when(col("category\_num")=='Lifestyle Services', 9)

.when(col("category\_num")=='Grocery & Convenience Stores', 10)

.when(col("category\_num")=='Home Services', 11)

.when(col("category\_num")=='Food & Personal Goods', 12)

.when(col("category\_num")=='Local Events', 13)

.when(col("category\_num")=='Professional Services', 14)

.when(col("category\_num")=='Auto Dealers', 15)

.when(col("category\_num")=='Home Goods Stores', 16)

.when(col("category\_num")=='Entities', 17)

.when(col("category\_num")=='Restaurants', 18)

.when(col("category\_num")=='Government Agencies', 19)

.when(col("category\_num")=='Geography', 20)

.when(col("category\_num")=='Home & Auto', 21)

).drop("category\_num")

# writing final dataframe to s3

final\_df.write.parquet("s3a://project-data-sb/parquet/final\_data.parquet")

Second part of the code (training/testing/predicting):

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.classification import RandomForestClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from sklearn.metrics import confusion\_matrix

from pyspark.sql.functions import col

# reading final dataframe from s3

final\_df = spark.read.parquet("s3a://project-data-sb/parquet/final\_data.parquet")

# casting string colunms to int

final\_df = final\_df.withColumn("following\_num",final\_df.following\_num.cast('int')).withColumn("follower\_num",final\_df.follower\_num.cast('int')).withColumn("post\_num",final\_df.post\_num.cast('int'))

# dropping rows with null values

final\_df = final\_df.na.drop()

features = ['like\_count', 'comment\_count', 'follower\_num', 'following\_num', 'post\_num']

va = VectorAssembler(inputCols = features, outputCol='features')

# transforming dataframe to be used with rfc

va\_df = va.transform(final\_df)

va\_df = va\_df.select('features', col('category\_num\_2').alias('label'))

(train, test) = va\_df.randomSplit([0.8, 0.2])

# training/testing/predicting

rfc = RandomForestClassifier(featuresCol="features", labelCol="label")

rfc = rfc.fit(train)

prediction = rfc.transform(test)

evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")

# evaluating model

accuracy = evaluator.evaluate(prediction)

predictions = prediction.select("prediction").collect()

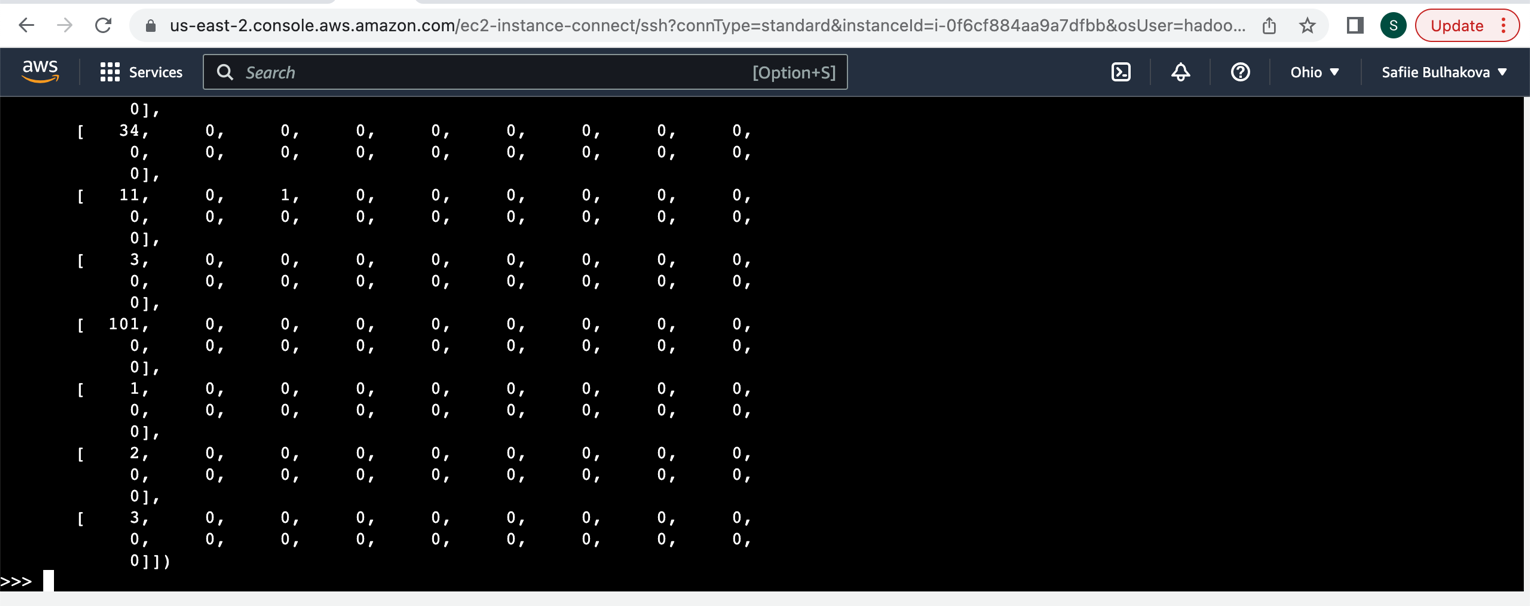
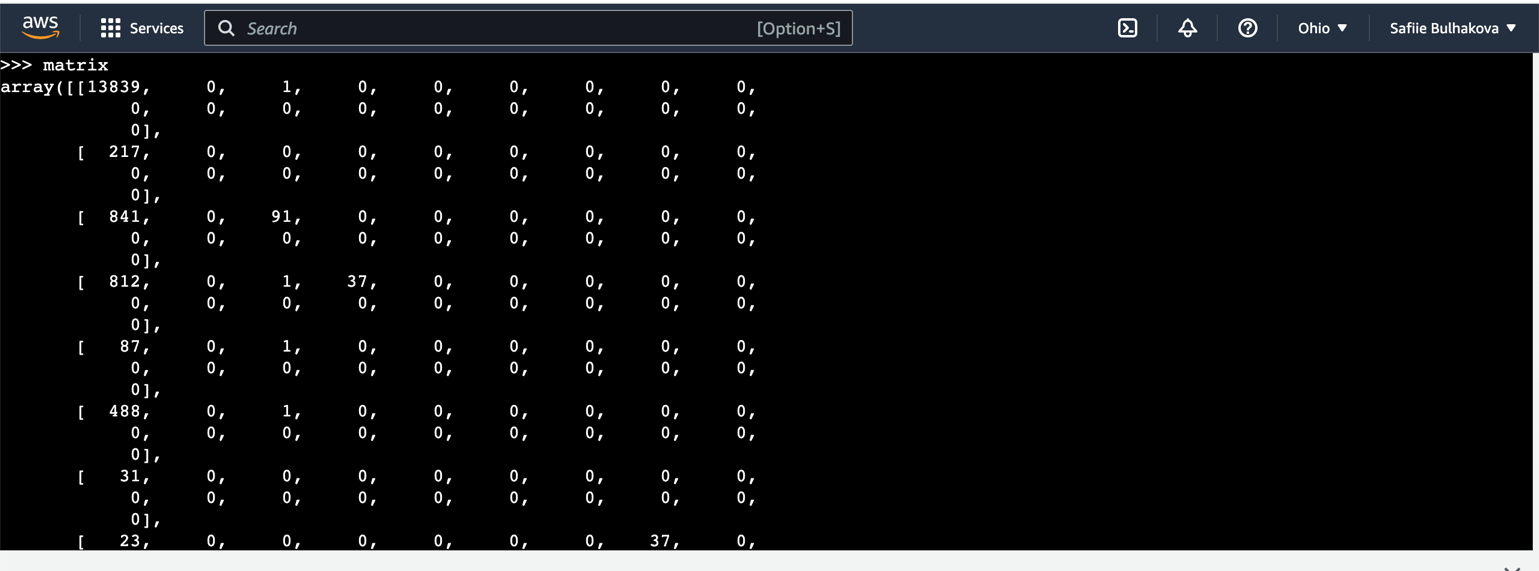
true\_vals = prediction.select("label").collect()

matrix = confusion\_matrix(true\_vals, predictions)

Screenshots of the results:

Table

Description automatically generated with medium confidenceText

Description automatically generated

In my S3 bucket => project-data-sb => unzipped json\_files.zip is post\_files/ which is below on screenshot, all the files that it contains:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Below are screenshots of total bucket size and total number of objects:

Chart, line chart

Description automatically generated

Below are screenshots of most recent attempts of opening clusters and increasing/changing memory, instance types, number of instances in order to understand what would work (there is much more of them). The last one was running more than 19 hours with one master and two core clusters with m5.2xlarge.

Graphical user interface, application

Description automatically generated

Overall, I spent $18.28 with feature to terminate cluster when it is idle after 1 hour. However, when I didn’t run code, I was immediately terminating the cluster. So, all clusters that was running it was for the purpose and running code and not accidentally leaving it unterminated.

Graphical user interface, application, Teams

Description automatically generated

Milestone 5: Visualizing Results

I decided to create four visualizations based on data that I have.

First, I called describe on pandas dataframe to see where exactly most of the data lies, in order to create visualizations from it. Screenshot of the statistics:

Graphical user interface, text, application

Description automatically generated

After that I was able to create visualizations of the data.

So, for my first visualization I decided to create “category vs average number of likes normalized by number of followers”. In order to do that I made final\_df = final\_df.withColumn('like\_count\_normalized\_by\_followers', final\_df.like\_count \* 100 / final\_df.follower\_num). Other three predictions: number of likes by number of comments, follower count by following number and category name by post number. Below is the code:

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.classification import RandomForestClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from sklearn.metrics import confusion\_matrix

from pyspark.sql.functions import col

# reading final dataframe from s3

final\_df = spark.read.parquet("s3a://project-data-sb/parquet/final\_data.parquet")

# casting string columns to int

final\_df = final\_df.withColumn("following\_num",final\_df.following\_num.cast('int')).withColumn("follower\_num",final\_df.follower\_num.cast('int')).withColumn("post\_num",final\_df.post\_num.cast('int'))

# dropping rows with null values

final\_df = final\_df.na.drop()

# add like counts normalized by number of followers

final\_df = final\_df.withColumn('like\_count\_normalized\_by\_followers', final\_df.like\_count \* 100 / final\_df.follower\_num)

# convert to pandas dataframe

pd = final\_df.toPandas()

#plot and save the 4 scatter plot pngs

myplot = pd.plot.scatter('like\_count', 'comment\_count', xlim=[0,10000], ylim=[0, 600], s=1)

myplot.get\_figure().savefig("likes\_comments.png")

myplot = pd.plot.scatter('following\_num', 'follower\_num', xlim=[0,7500], ylim=[1000, 1000000], s=1)

myplot.get\_figure().savefig("following\_follower.png")

myplot = pd.plot.scatter('category\_num\_2', 'post\_num', xlim=[0,20], ylim=[0, 10000], s=1)

myplot.get\_figure().savefig("category\_post.png")

myplot = pd.plot.scatter('category\_num\_2', 'like\_count\_normalized\_by\_followers', xlim=[0,20], ylim=[0, 20], s=1)

myplot.get\_figure().savefig("category\_likes.png")

**Visualizations:**

Chart, scatter chart

Description automatically generated

**Visualization 1:**

On visualization 1 we see category name vs post number. So, for example, we can see that ‘Creators & Celebrities' category (1) and 'Publishers' (3) have many more posts compared to 'Food & Personal Goods' (12) and 'Professional Services' (14).

Chart, scatter chart

Description automatically generated

**Visualization 2:**

On visualization 2 we can see follower count by following number. Here we can see a curve that goes from left to right, which concludes that follower number decreases with the following number. So, for example when a person has more followers it less likely for them to have a lot of followings compared to those who doesn’t have many followers. It is, for example, when celebrities have a million followers, they likely to have not more than 500-1,000 of following number (even less). And when a person has much less than 200k followers they likely to have more following number, comparably, how the graph shows.

Chart

Description automatically generated

**Visualization 3:**

On visualization 3 we can see category number by average number of likes normalized by number of followers. Some categories, like 'Creators & Celebrities' (1), 'Publishers' (3), and 'General Interest' (6) have high variance in the number of likes per post, meaning the quality of the post determines how much of the follower base likes the post. Other categories, like 'Transportation & Accomodation Services' (7), 'Grocery & Convenience Stores' (10) and 'Local Events' (13), don’t have as much variance, and actually have a ceiling to the percentage of followers who like their posts, and post quality might not matter as much.

Chart, scatter chart

Description automatically generated

**Visualization 4:**

On visualization 4 we can see number of comments by number of likes. Although there is high variance among the data points, we can see somewhat of a ceiling to the number of comments on a post, regardless of the number of likes, which is around 100 comments. This is interesting because that must mean that when a post reaches around 100 comments, users that like a post are less likely to comment, regardless of how popular a post is.

Milestone 6: Summary and Conclusions

Description of Data Pipeline:

Diagram

Description automatically generated

The whole dataset is divided into four parts (four files): post\_info.txt, json\_files.zip, profiles\_influencers.zip and profiles\_brands.zip.

Before starting the project, I knew that files are very large, and I will be facing some challenges along the way because of it. However, what I did not know is that there would be huge number of individual files in the zip files that I needed to unzip in order to do the project. It gave me the biggest challenge overall and because of it I was able to run only the part of the data that was given to me, in order, first, to find statistics that I intended to find and then the prediction that I meant to make. Overall, I was able to find statistics of the 10,000 files (milestone 3) and then make a prediction based on the 100,000 files (milestone 4).

I was able to make a prediction of the category of the influencer by user ‘s number of followers, number of followings, number of likes, number of comments, and number of posts. The prediction showed 74% accuracy and I think it would be much higher if I would be able to train it based on all the data that I have in S3. Also, I was able to visualize category name by post number, follower count by following number, category number by average number of likes normalized by number of followers and number of comments by number of likes. I was planning make visualization based on the prediction that I made, but unfortunately, since I was able to use only 100,000 files the visualization would show very slightly that there is an accuracy between category of the influencer and user’s number of followers, number of followings etc. The graph by the results that I received would show very vague and would not be understandable even though I wanted to make it.

Overall, it was a great learning experience in all aspects of it, learning AWS and how it works, how to manage everything inside it and what is needed for my dataset (increasing storage etc). The biggest lesson that I learned was that it is very important with which dataset you are working with, because again, would I have known that because of very big number of files I would not be able to proceed to some results and would not be able to run properly what was needed, I would definitely have done everything in a different way. It was a great experience and also there is a lot more to see, predict and analyze from this dataset.

Citation

1. Prasmussen/gdrive. HitHub. URL: https://github.com/prasmussen/gdrive/releases/tag/2.1. Accessed September 29th, 2022.