ALUMNI PROFILE MATCHING

Team Name - SnackWeb
Members -
Mohana Medisetty
Preetham Pathi
Renuka Marepalli
Yu Hsin Wang

Course : Distributed Data Systems - MSDS 697, University of San Francisco

Problem/Motivation Statement

Networking with people having similar past experience and professional/career goals has become increasingly essential for any working professional, especially in the tech industry. It has become even more important for graduate students pursuing tech programs abroad and who are on the lookout for jobs in a challenging job market. Networking with alumni can be especially important for graduate students as they can leverage the shared academic backgrounds, get mentorship support and potentially get a referral in a company that the alumni may be currently or previously associated with.

Usually, students get an opportunity to only reach out to alumni to whom they are introduced to during university meetups and fireside chats for a very brief moment.Moreso, the alumni's career track may be very different from that of the current student's track and hence his/her advices may not be the most ideal given the difference in backgrounds. This could also overwhelm the alumni or they might not be able to answer all the current students' questions. For suppose, if there are 2 students - one with a Finance background with several years of experience and the other with a Computer Science background with no/little experience. Alumni with relative or similar backgrounds can make better and relevant suggestions .

In this project, we aimed to build a machine learning model to realize the goal of picking the top 5 similar alumni profiles for each current student to whom they can reach out by determining the similarity scores for the students and the alumni. We will also focus largely on building a robust and distributed data processing pipeline that will feed into our ML model to achieve this goal.

Dataset & Analytics Goals:

We have two primary data sources for this project.

Data sources :

1. List of alumni profiles for each cohort

   We reached out to the administrative head to get a list of alumni of all previous cohorts of the graduate program to have a focussed list of profiles to crawl over and maintained the file in a storage bucket on Google cloud platform (GCP)

2. Scraped linkedin profiles data(in json) per cohort

   We leveraged the rapidAPI interface for integration with linkedin developer APIs in order to fetch scraped linkedin profile data of each of the alumni into our data system.

Analytics Goals:

- Use Bert embedding to vectorize the user profiles
- Use GloVe embedding to get the top 5 similar profiles, and compare them with Bert Model
- Do Pre-processing, feature Engineering, and check how models perform after that
- Perform TF-IDF on the transformed/Simplified data

Overview of data engineering pipeline

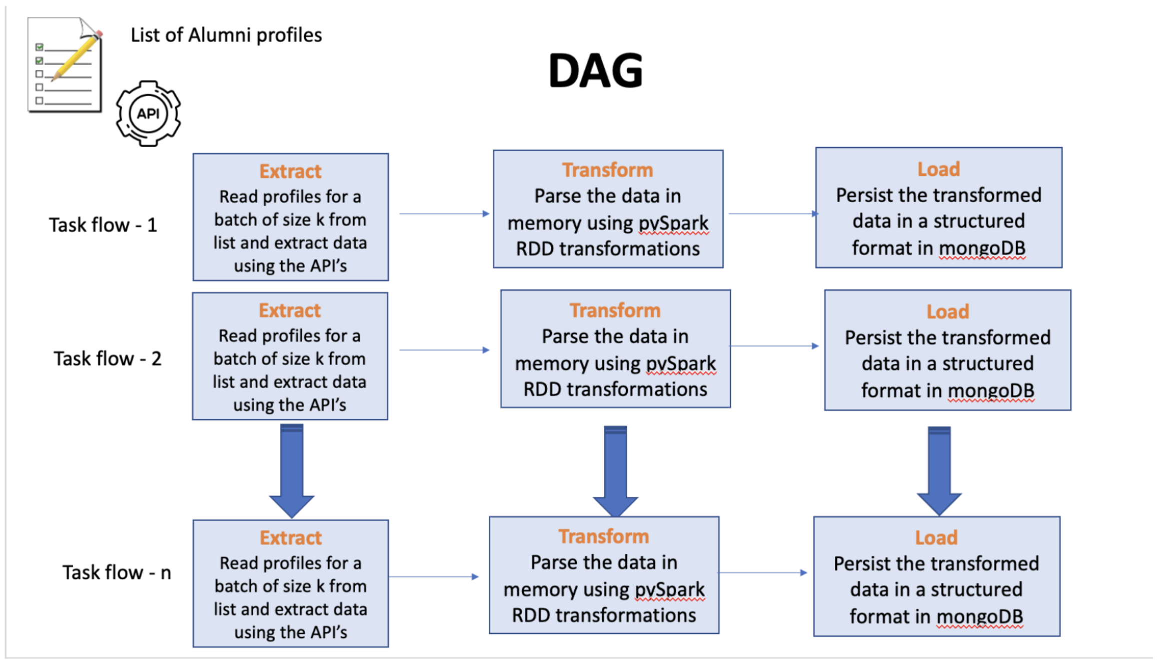
As for any data engineering pipeline, the series of processes that we used to fetch, transform and load data are as follows

Data ingestion —> Data Transformation —> Data Storage

The above set of tasks constitute what we call an ETL pipeline. An ETL pipeline is responsible for extracting data, transforming into desired format and finally loading the organized and aggregated data into a data lake.

Most of the time, these tasks are repetitive and can be executed with zero or very less manual intervention and hence is best to be automated.

We automated these series of tasks using **Apache Airflow DAG Scheduler**.
A DAG (Directed Acyclic Graph) is a graph based representation consisting of nodes and edges The nodes represent tasks, whereas the edges model the dependencies between the tasks along with the direction of workflow.
Below is a representation of tasks that we modeled using a DAG.


List of Alumni profiles
API
DAG
Task flow - 1
Extract
Read profiles for a batch of size k from list and extract data using the API's
Transform
Parse the data in memory using pySpark RDD transformations
Load
Persist the transformed data in a structured format in mongoDB
Task flow - 2
Task flow - n

DAG Task flow

We defined two tasks that will run for each batch of cohort - one for **extracting** data and the other for **transforming** and **loading** the data.

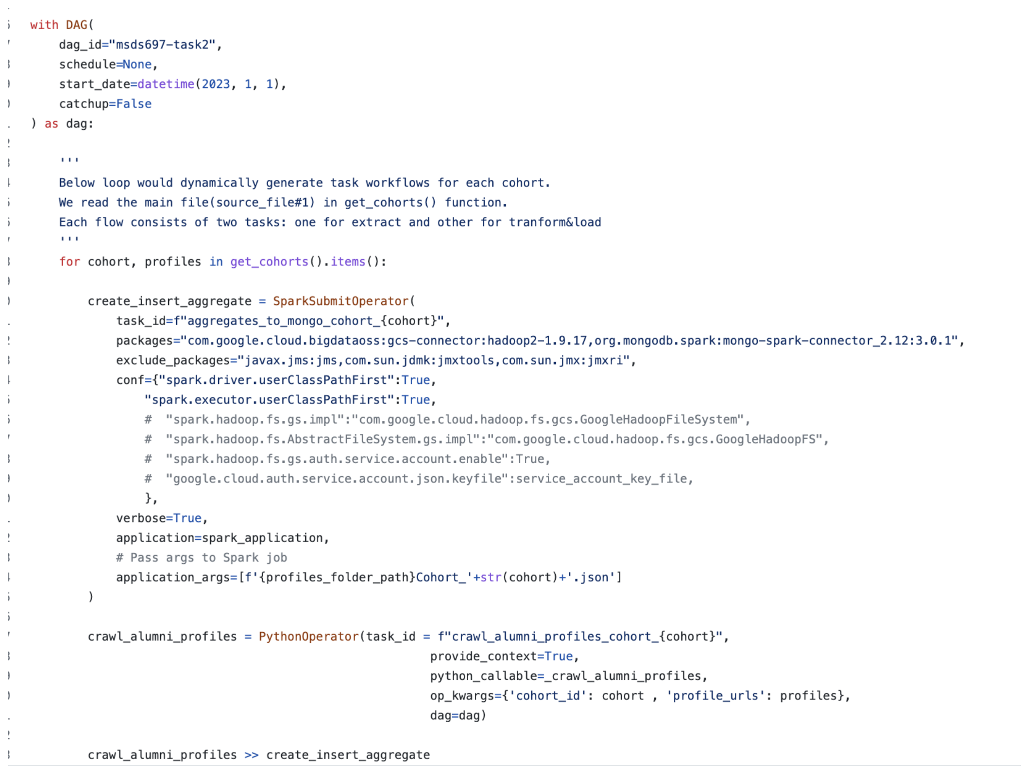
The above defined tasks are encapsulated within a DAG block and named using a dag_id.
The DAG is created with a start date of January 1st, 2023. We have scheduled to run on an ad-hoc basis for now but the DAG could be modified to run on an yearly basis to run the task flow as and when a list of alumni from the most recent cohort is available.
The DAG contains a loop that **dynamically generates task workflows** for each cohort. The function get_cohorts() is called to read the main file (source_file#1) and retrieve a list of profiles for each cohort.

Task 1

We defined the **first task** of our DAG template to implement the **Extract block** of our **ETL pipeline**.The task is defined to invoke a script that crawls over the list of alumni within a cohort (fetched from GCP bucket) and pulls the scraped linkedin profile data as an API response . The response data is stored in json formatted files on the GCP storage bucket for every cohort. **Exceptions** in case of api invocation failure or an undesirable response are handled by logging and tracking a list of profiles in the GCP bucket for which the script failed to fetch data.
This script is defined as a task and executed within airflow with the help of a **PythonOperator**.

A **PythonOperator** by definition is used to execute Python functions or methods as tasks in a DAG. The operator can run any Python code, including custom scripts or libraries, and can pass arguments dynamically to the function being executed.

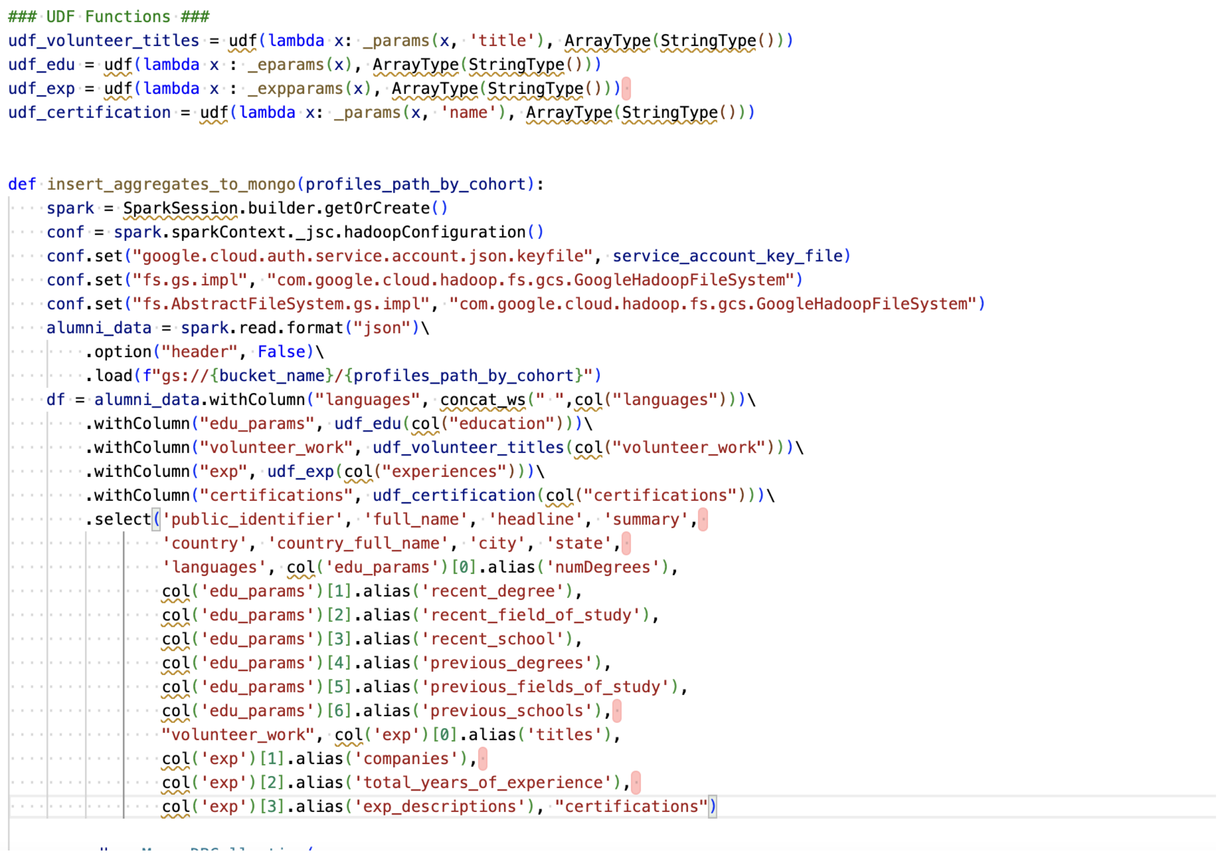
In our DAG, this operator runs a Python function **_crawl_alumni_profiles** to crawl alumni profiles for the specified cohort.
The function takes two arguments, cohort_id and profile_urls, which are passed using the **op_kwargs** parameter. This aids the functions invoked further in line to read these parameters to crawl over the list of profiles pertaining to only a single cohort at a time. The **provide_context=True** parameter is used to pass the **Airflow context** to the function, which provides information about the current execution context.



```python
with DAG(
    dag_id="msds697-task2",
    schedule=None,
    start_date=datetime(2023, 1, 1),
    catchup=False
) as dag:

    '''
    Below loop would dynamically generate task workflows for each cohort.
    We read the main file(source_file#1) in get_cohorts() function.
    Each flow consists of two tasks: one for extract and other for tranform&load
    '''
    for cohort, profiles in get_cohorts().items():

        create_insert_aggregate = SparkSubmitOperator(
            task_id=f"aggregates_to_mongo_cohort_{cohort}",
            packages="com.google.cloud.bigdataoss:gcs-connector:hadoop2-1.9.17,org.mongodb.spark:mongo-spark-connector_2.12:3.0.1",
            exclude_packages="javax.jms:jms,com.sun.jdmk:jmxtools,com.sun.jmx:jmxri",
            conf={"spark.driver.userClassPathFirst":True,
                "spark.executor.userClassPathFirst":True,
                #  "spark.hadoop.fs.gs.impl":"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem",
                #  "spark.hadoop.fs.AbstractFileSystem.gs.impl":"com.google.cloud.hadoop.fs.gcs.GoogleHadoopFS",
                #  "spark.hadoop.fs.gs.auth.service.account.enable":True,
                #  "google.cloud.auth.service.account.json.keyfile":service_account_key_file,
                },
            verbose=True,
            application=spark_application,
            # Pass args to Spark job
            application_args=[f'{profiles_folder_path}Cohort_'+str(cohort)+'.json']
        )

        crawl_alumni_profiles = PythonOperator(task_id = f"crawl_alumni_profiles_cohort_{cohort}",
                                               provide_context=True,
                                               python_callable=_crawl_alumni_profiles,
                                               op_kwargs={'cohort_id': cohort , 'profile_urls': profiles},
                                               dag=dag)

        crawl_alumni_profiles >> create_insert_aggregate
```

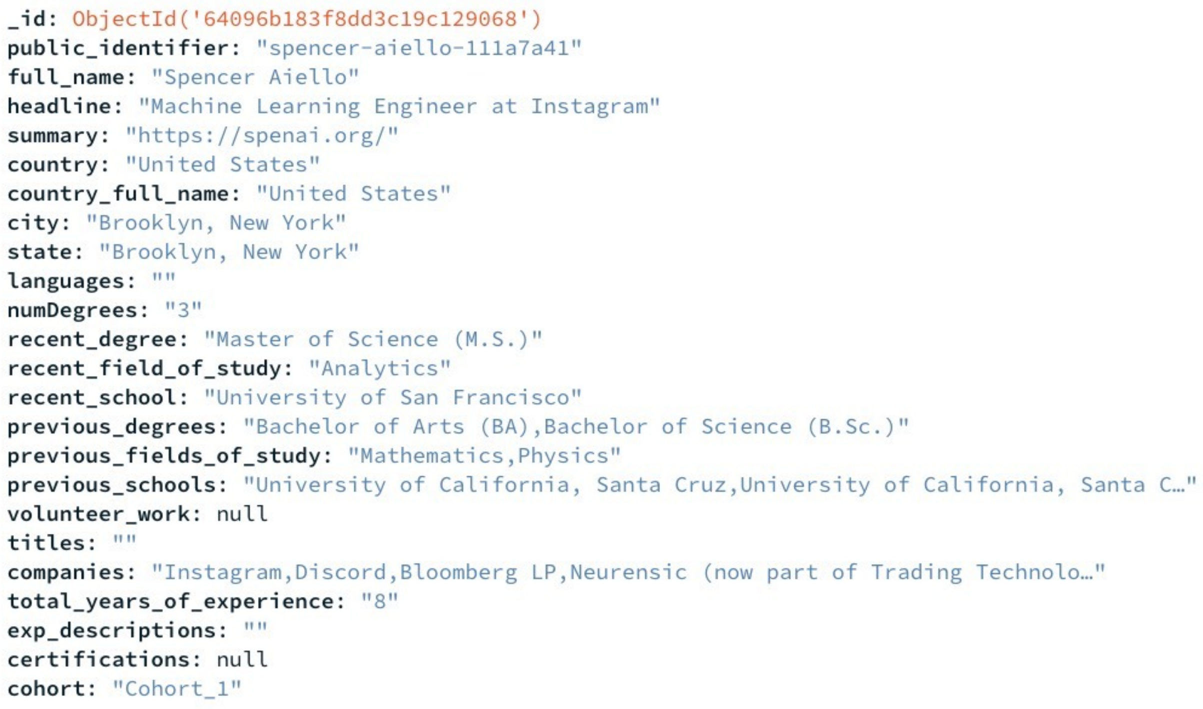
Task 2:

The **second task** of our **DAG** implements the **Transform and Load blocks** of our ETL pipeline.
We have used **Apache pySpark** library inorder to perform the first level of data cleaning and transformations.We encountered some missing data and inconsistency issues with our crawled profile data and handled the same using transformations via pySpark. The initial data also was deeply nested, hence we wrangled the most important fields within an alum's profile such as the education details, professional experience details, profile summary etc and flattened our final data field.. We defined spark **UDF's (user defined functions)** in order to define the transformations necessary on each column.We also **engineered features** such as total_year_of_experience from the nested experiences information.



```
### UDF Functions ###
udf_volunteer_titles = udf(lambda x: _params(x, 'title'), ArrayType(StringType()))
udf_edu = udf(lambda x : _eparams(x), ArrayType(StringType()))
udf_exp = udf(lambda x : _expparams(x), ArrayType(StringType()))
udf_certification = udf(lambda x: _params(x, 'name'), ArrayType(StringType()))


def insert_aggregates_to_mongo(profiles_path_by_cohort):
    spark = SparkSession.builder.getOrCreate()
    conf = spark.sparkContext._jsc.hadoopConfiguration()
    conf.set("google.cloud.auth.service.account.json.keyfile", service_account_key_file)
    conf.set("fs.gs.impl", "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem")
    conf.set("fs.AbstractFileSystem.gs.impl", "com.google.cloud.hadoop.fs.gcs.GoogleHadoopFileSystem")
    alumni_data = spark.read.format("json")\
        .option("header", False)\
        .load(f"gs://{bucket_name}/{profiles_path_by_cohort}")
    df = alumni_data.withColumn("languages", concat_ws(" ",col("languages")))\
        .withColumn("edu_params", udf_edu(col("education")))\
        .withColumn("volunteer_work", udf_volunteer_titles(col("volunteer_work")))\
        .withColumn("exp", udf_exp(col("experiences")))\
        .withColumn("certifications", udf_certification(col("certifications")))\
        .select('public_identifier', 'full_name', 'headline', 'summary',
                'country', 'country_full_name', 'city', 'state',
                'languages', col('edu_params')[0].alias('numDegrees'),
                col('edu_params')[1].alias('recent_degree'),
                col('edu_params')[2].alias('recent_field_of_study'),
                col('edu_params')[3].alias('recent_school'),
                col('edu_params')[4].alias('previous_degrees'),
                col('edu_params')[5].alias('previous_fields_of_study'),
                col('edu_params')[6].alias('previous_schools'),
                "volunteer_work", col('exp')[0].alias('titles'),
                col('exp')[1].alias('companies'),
                col('exp')[2].alias('total_years_of_experience'),
                col('exp')[3].alias('exp_descriptions'), "certifications")
```

Each of the flattened profile documents was **aggregated** with every record on the alumni list as a key, value pair and each aggregate thus formed was loaded into **mongoDB** as a **json document** using apis from **pyMongo** library.



```
_id: ObjectId('64096b183f8dd3c19c129068')
public_identifier: "spencer-aiello-111a7a41"
full_name: "Spencer Aiello"
headline: "Machine Learning Engineer at Instagram"
summary: "https://spenai.org/"
country: "United States"
country_full_name: "United States"
city: "Brooklyn, New York"
state: "Brooklyn, New York"
languages: ""
numDegrees: "3"
recent_degree: "Master of Science (M.S.)"
recent_field_of_study: "Analytics"
recent_school: "University of San Francisco"
previous_degrees: "Bachelor of Arts (BA),Bachelor of Science (B.Sc.)"
previous_fields_of_study: "Mathematics,Physics"
previous_schools: "University of California, Santa Cruz,University of California, Santa C…"
volunteer_work: null
titles: ""
companies: "Instagram,Discord,Bloomberg LP,Neurensic (now part of Trading Technolo…"
total_years_of_experience: "8"
exp_descriptions: ""
certifications: null
cohort: "Cohort_1"
```

Aggregated JSON Document Structure

MongoDB, for the unversed, is a **distributed** documented oriented NOSql database, designed for scalability.
It maintains records in JSON like documents and provides flexibility in terms of schema.

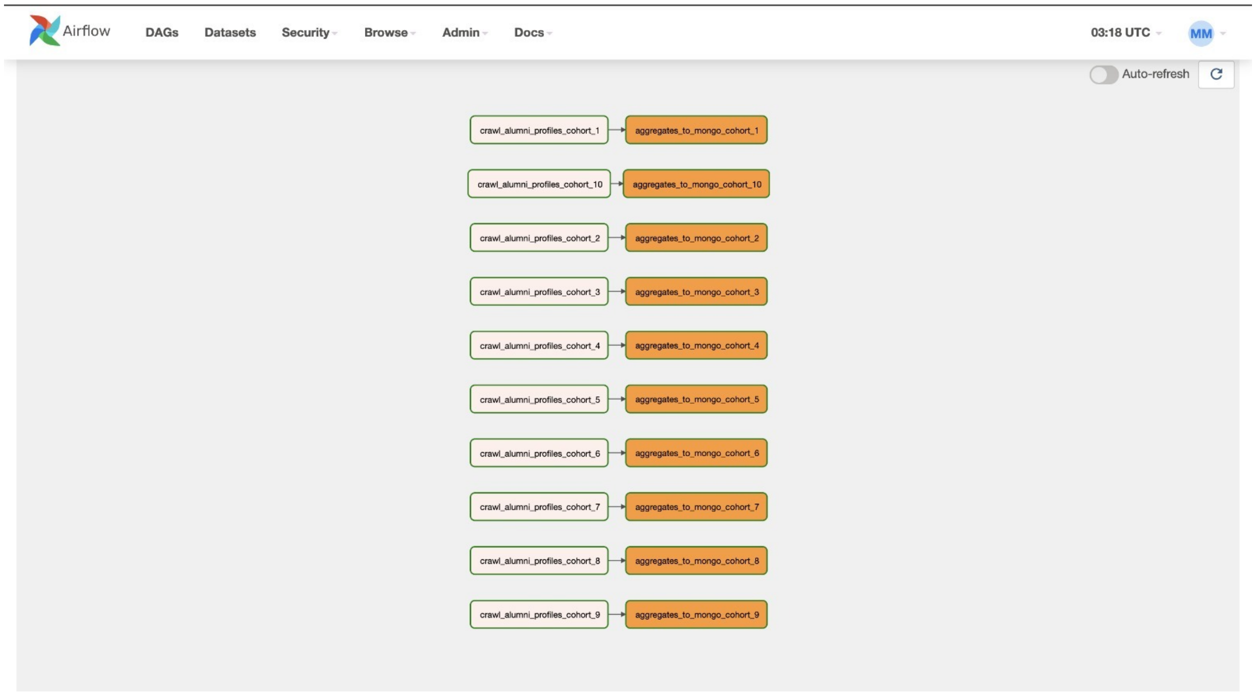
As we predominantly used spark for our data cleaning and transformation, it is important to establish Spark context in order for spark jobs to be executed by the driver node even from within an automated airflow task.

To achieve the same we used a **SparkSubmitOperator** DAG operator. This operator helps to submit a Spark job to a cluster as a task in a DAG. This operator takes the following parameters:
The operator takes the following parameters:

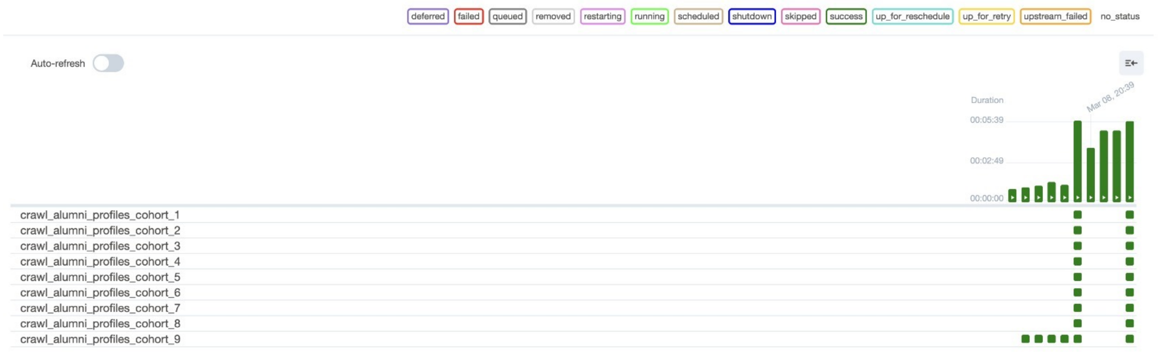
application: A string that specifies the location of the Spark application to be executed.
application_args: A list of strings that specifies any arguments to be passed to the Spark application.
packages: A comma-separated list of Maven packages required by the Spark job.
exclude_packages: A comma-separated list of Maven packages to exclude from the Spark job.
conf: A dictionary of configuration settings for the Spark job.
verbose: A boolean value indicating whether to run the Spark job in verbose mode.
Once these tasks are defined and associated with their respective scripts for dynamic invocation, we have to establish the dependency relationship between the tasks.

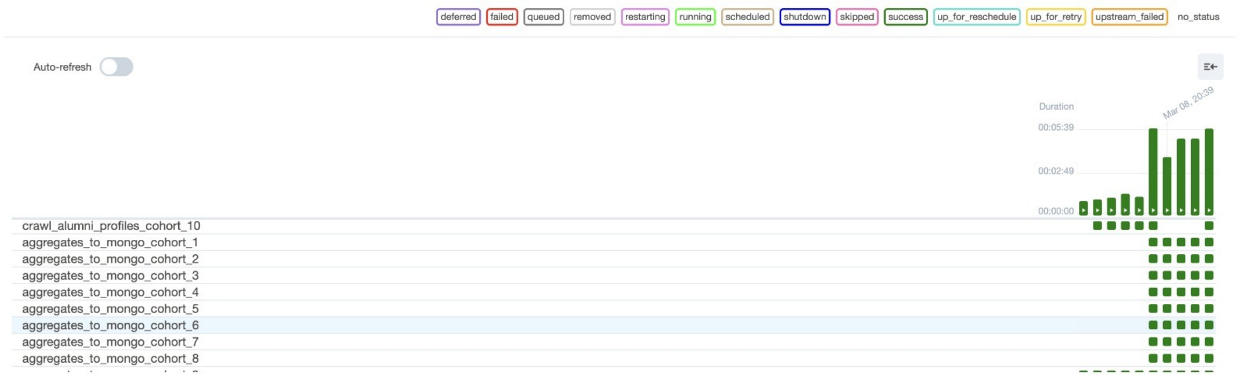
The **>> operator** connects the two tasks in a DAG, indicating that the **PythonOperator** should be executed before the **SparkSubmitOperator**. This means that the alumni profiles should be crawled and staged first before the data is transformed using Spark and is written to Mongo.

Below is a snip of the DAG task flows as a flow chart with aforementioned two tasks, created dynamically for every cohort. Note that the **dynamic task flow generation** is done iteratively in a loop, and by passing parameters to the airflow operators during execution .


Airflow
DAGs
Datasets
Security
Browse
Admin
Docs
03:18 UTC
MM
Auto-refresh
crawl_alumni_profiles_cohort_1
aggregates_to_mongo_cohort_1
crawl_alumni_profiles_cohort_10
aggregates_to_mongo_cohort_10
crawl_alumni_profiles_cohort_2
aggregates_to_mongo_cohort_2
crawl_alumni_profiles_cohort_3
aggregates_to_mongo_cohort_3
crawl_alumni_profiles_cohort_4
aggregates_to_mongo_cohort_4
crawl_alumni_profiles_cohort_5
aggregates_to_mongo_cohort_5
crawl_alumni_profiles_cohort_6
aggregates_to_mongo_cohort_6
crawl_alumni_profiles_cohort_7
aggregates_to_mongo_cohort_7
crawl_alumni_profiles_cohort_8
aggregates_to_mongo_cohort_8
crawl_alumni_profiles_cohort_9
aggregates_to_mongo_cohort_9

The below two snapshots show the landing page to monitor DAG task flow progress, every successful task can be marked Done in order to progress to the next task whereas every failed task will have to be inspected by using logs and fixed for errors.


deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
Duration
Mar 08, 20:39
00:05:39
00:02:49
00:00:00
crawl_alumni_profiles_cohort_1
crawl_alumni_profiles_cohort_2
crawl_alumni_profiles_cohort_3
crawl_alumni_profiles_cohort_4
crawl_alumni_profiles_cohort_5
crawl_alumni_profiles_cohort_6
crawl_alumni_profiles_cohort_7
crawl_alumni_profiles_cohort_8
crawl_alumni_profiles_cohort_9


deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Auto-refresh
Duration
Mar 08, 20:39
00:05:39
00:02:49
00:00:00
crawl_alumni_profiles_cohort_10
aggregates_to_mongo_cohort_1
aggregates_to_mongo_cohort_2
aggregates_to_mongo_cohort_3
aggregates_to_mongo_cohort_4
aggregates_to_mongo_cohort_5
aggregates_to_mongo_cohort_6
aggregates_to_mongo_cohort_7
aggregates_to_mongo_cohort_8

Data Cleaning, Preprocessing and Cluster specifications

We used a databricks platform cluster to perform preprocessing and modeling on the data. Below are the configurations of our databricks compute and mongoDB database clusters.

MongoDB Cluster specification:

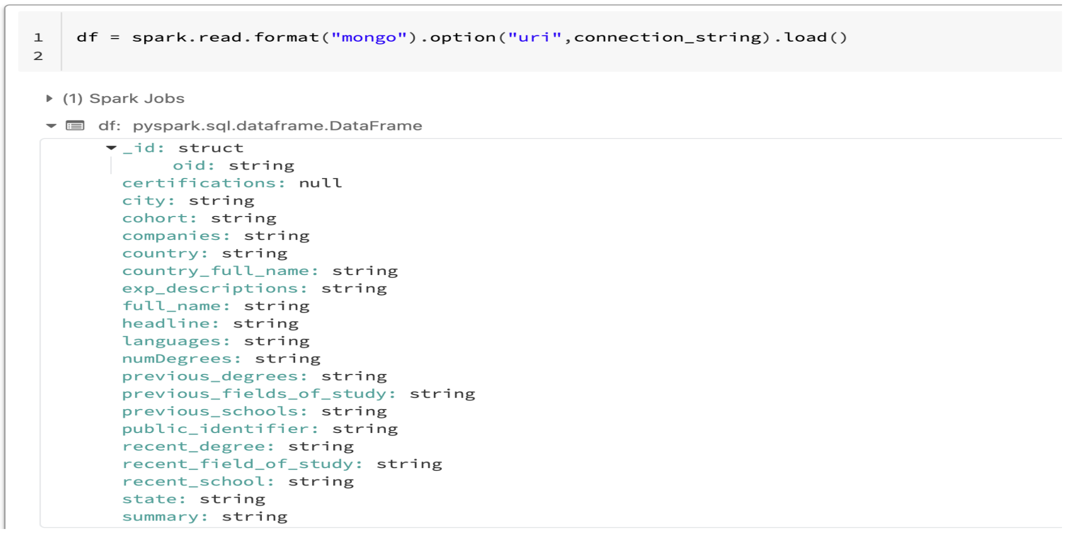
M30 (8 GB RAM, 40 GB Storage per Shard)
2,400 IOPS per Shard, Encrypted, Auto-expand Storage
MongoDB 5.0, Backup, 2 shards

Databricks Cluster specification:

Run Time version - 7.3 LTS (includes Apache Spark 3.0.1, Scala 2.12)
Cluster specification
I3.xlarge
Driver - 30.5 GB Memory 4 cores

2 - 5 Workers (61- 152 GB memory, 8-20 cores)

A connection from spark cluster to **mongoDB** was established via a **Spark-mongo** connector that was installed on the cluster and by passing the necessary credentials , after which data from mongoDB was read into a spark Dataframe.

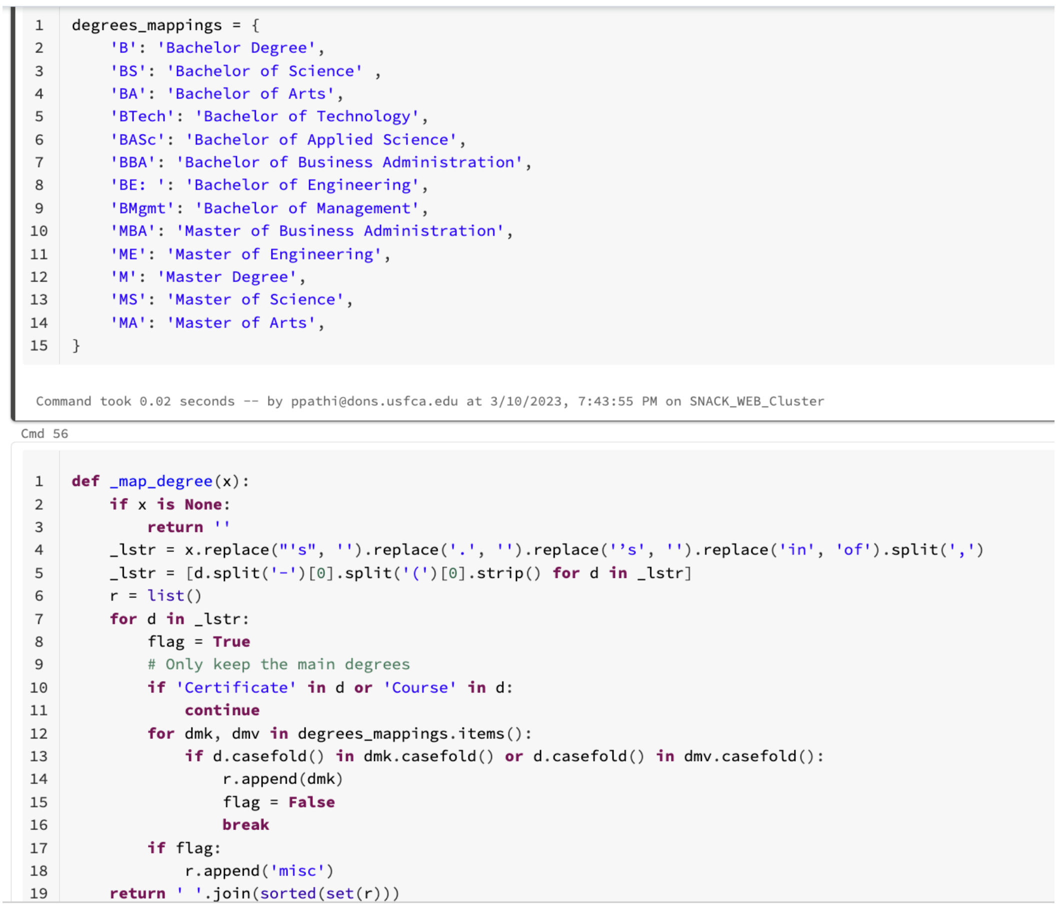


```
df = spark.read.format("mongo").option("uri",connection_string).load()

```

```
(1) Spark Jobs
df:  pyspark.sql.dataframe.DataFrame
    _id: struct
        oid: string
    certifications: null
    city: string
    cohort: string
    companies: string
    country: string
    country_full_name: string
    exp_descriptions: string
    full_name: string
    headline: string
    languages: string
    numDegrees: string
    previous_degrees: string
    previous_fields_of_study: string
    previous_schools: string
    public_identifier: string
    recent_degree: string
    recent_field_of_study: string
    recent_school: string
    state: string
    summary: string
```

Data cleaning

We handled the null value occurrences in our data and dropped columns that were not useful for modeling such information identifiers, names etc.
We had a column **previous_degrees** that holds the names of degrees of the alumni and we observed that the degree titles had a lot of variety . Inorder to generalize the data in this column, we had a custom UDF in place that mapped the degree names to standard degree acronyms as shown below.



```python
degrees_mappings = {
    'B': 'Bachelor Degree',
    'BS': 'Bachelor of Science' ,
    'BA': 'Bachelor of Arts',
    'BTech': 'Bachelor of Technology',
    'BASc': 'Bachelor of Applied Science',
    'BBA': 'Bachelor of Business Administration',
    'BE: ': 'Bachelor of Engineering',
    'BMgmt': 'Bachelor of Management',
    'MBA': 'Master of Business Administration',
    'ME': 'Master of Engineering',
    'M': 'Master Degree',
    'MS': 'Master of Science',
    'MA': 'Master of Arts',
}
```

```
Command took 0.02 seconds -- by ppathi@dons.usfca.edu at 3/10/2023, 7:43:55 PM on SNACK_WEB_Cluster
```

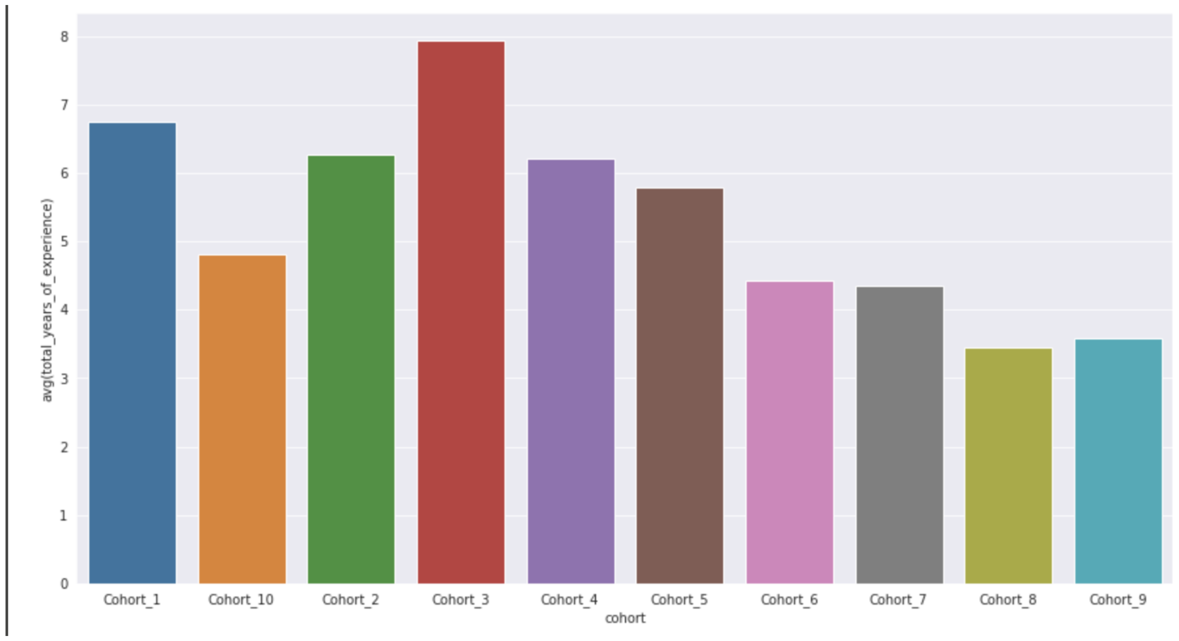
Cmd 56

```python
def _map_degree(x):
    if x is None:
        return ''
    _lstr = x.replace("'s", '').replace('.', '').replace('’s', '').replace('in', 'of').split(',')
    _lstr = [d.split('-')[0].split('(')[0].strip() for d in _lstr]
    r = list()
    for d in _lstr:
        flag = True
        # Only keep the main degrees
        if 'Certificate' in d or 'Course' in d:
            continue
        for dmk, dmv in degrees_mappings.items():
            if d.casefold() in dmk.casefold() or d.casefold() in dmv.casefold():
                r.append(dmk)
                flag = False
                break
        if flag:
            r.append('misc')
    return ' '.join(sorted(set(r)))
```

EDA

Below is some EDA that we performed on the alumni data

- We wanted to find the average total work experience for alums for each cohort and the distribution per cohort looks as follows



Feature engineering:

We performed feature engineering on a column named “city” to obtain the geographical latitude and longitude coordinates, to be used for similarity computations later.

For this phase of our work, we decided to use the below features as input data for our model
Features = ['previous_degrees','previous_fields_of_study','previous_schools','companies','summary', 'city','languages']

All the above features were concatenated to form a single **sequence** for later use .

ML Goals, Outcomes and Execution times:

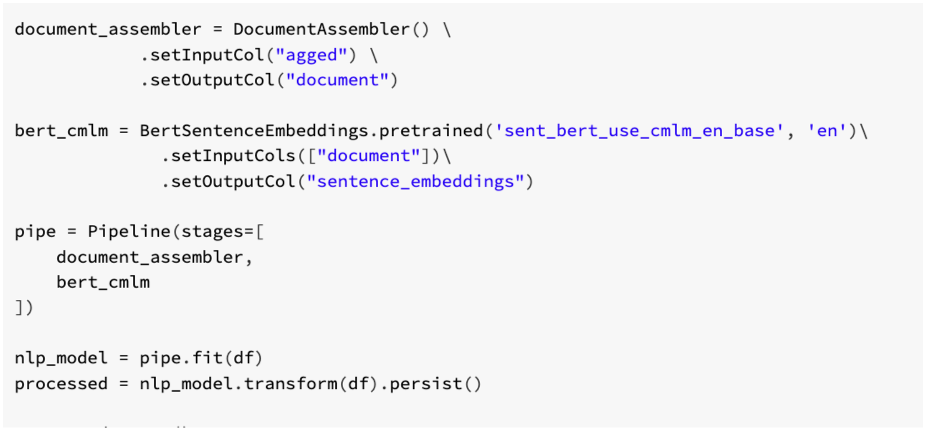
Our end machine learning goal is to get top 5 picks of alumni profiles for an input profile that are of closest similarity based on certain academic and professional background similarities among profiles.

Word cloud generated from alumni Profile data Sequence

In our attempt to achieve this, we tried the below popular NLP approaches to find similarity between the profile sequences.

1) Using BERT Embeddings

We used the SparkNLP library to create a pipeline for processing the text sequence data.

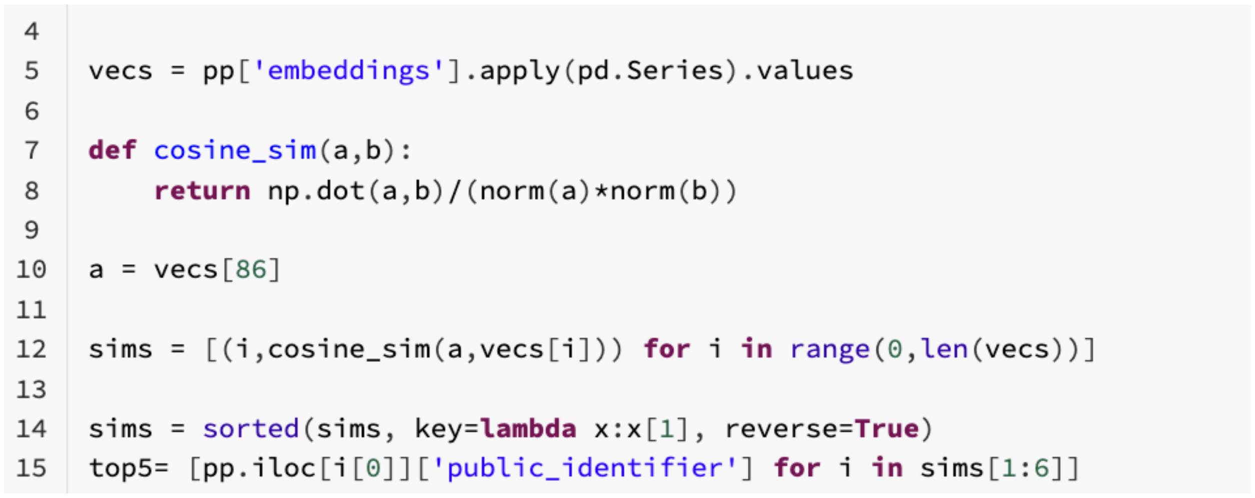


```
document_assembler = DocumentAssembler() \
            .setInputCol("agged") \
            .setOutputCol("document")

bert_cmlm = BertSentenceEmbeddings.pretrained('sent_bert_use_cmlm_en_base', 'en')\
              .setInputCols(["document"])\
              .setOutputCol("sentence_embeddings")

pipe = Pipeline(stages=[
    document_assembler,
    bert_cmlm
])

nlp_model = pipe.fit(df)
processed = nlp_model.transform(df).persist()
```

-The **DocumentAssembler** is used to convert the input text data in the "**agged**" column of the DataFrame df into a **Document** type that can be processed by the pipeline.
- The **BertSentenceEmbeddings** model is used to generate **sentence embeddings** for each sentence in the documents produced by the DocumentAssembler. This model is pretrained on a large corpus of English text and uses a **contextualized** masked language modeling (CMLM) approach to generate high-quality embeddings.
-The Pipeline is created by **combining** the **DocumentAssembler** and **BertSentenceEmbeddings** stages.

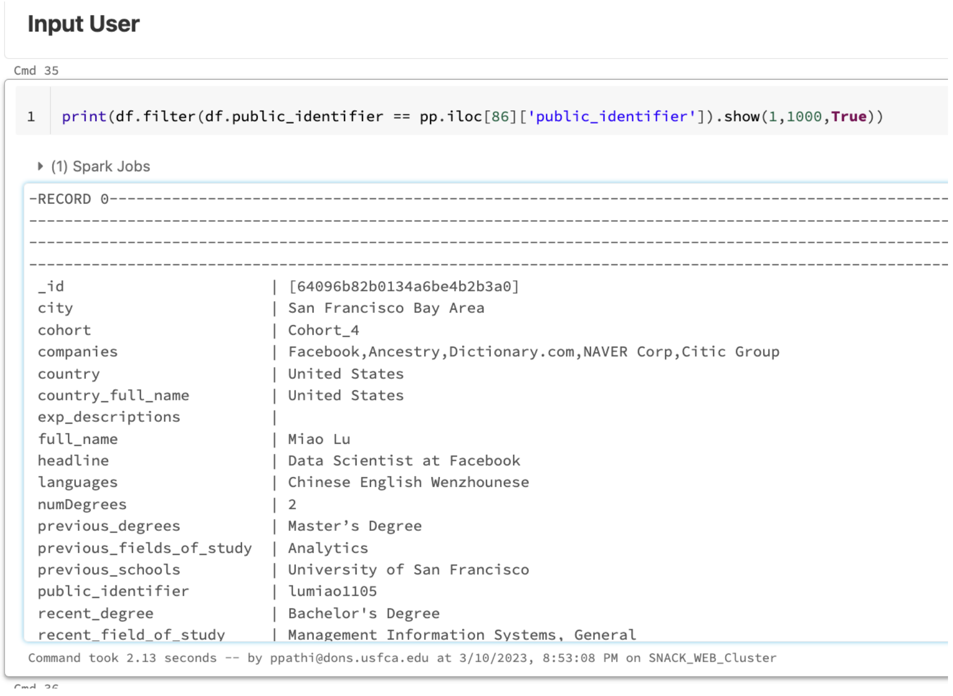
The **fit** method called on the input dataframe trains the pipeline on the input data and returns a trained BERT model . We then obtain the sentence embeddings for our profile sequences by passing the sequence data to this model.

In order to suggest most similar profiles to a given user from their alumni using **BERT Embeddings** we used the technique of computing **cosine similarity b**etween the embedding vectors.



```python

vecs = pp['embeddings'].apply(pd.Series).values

def cosine_sim(a,b):
    return np.dot(a,b)/(norm(a)*norm(b))

a = vecs[86]

sims = [(i,cosine_sim(a,vecs[i])) for i in range(0,len(vecs))]

sims = sorted(sims, key=lambda x:x[1], reverse=True)
top5= [pp.iloc[i[0]]['public_identifier'] for i in sims[1:6]]
```

For a given input user, using cosine similarity, the top 5 similar profiles were matched as follows:



Input User

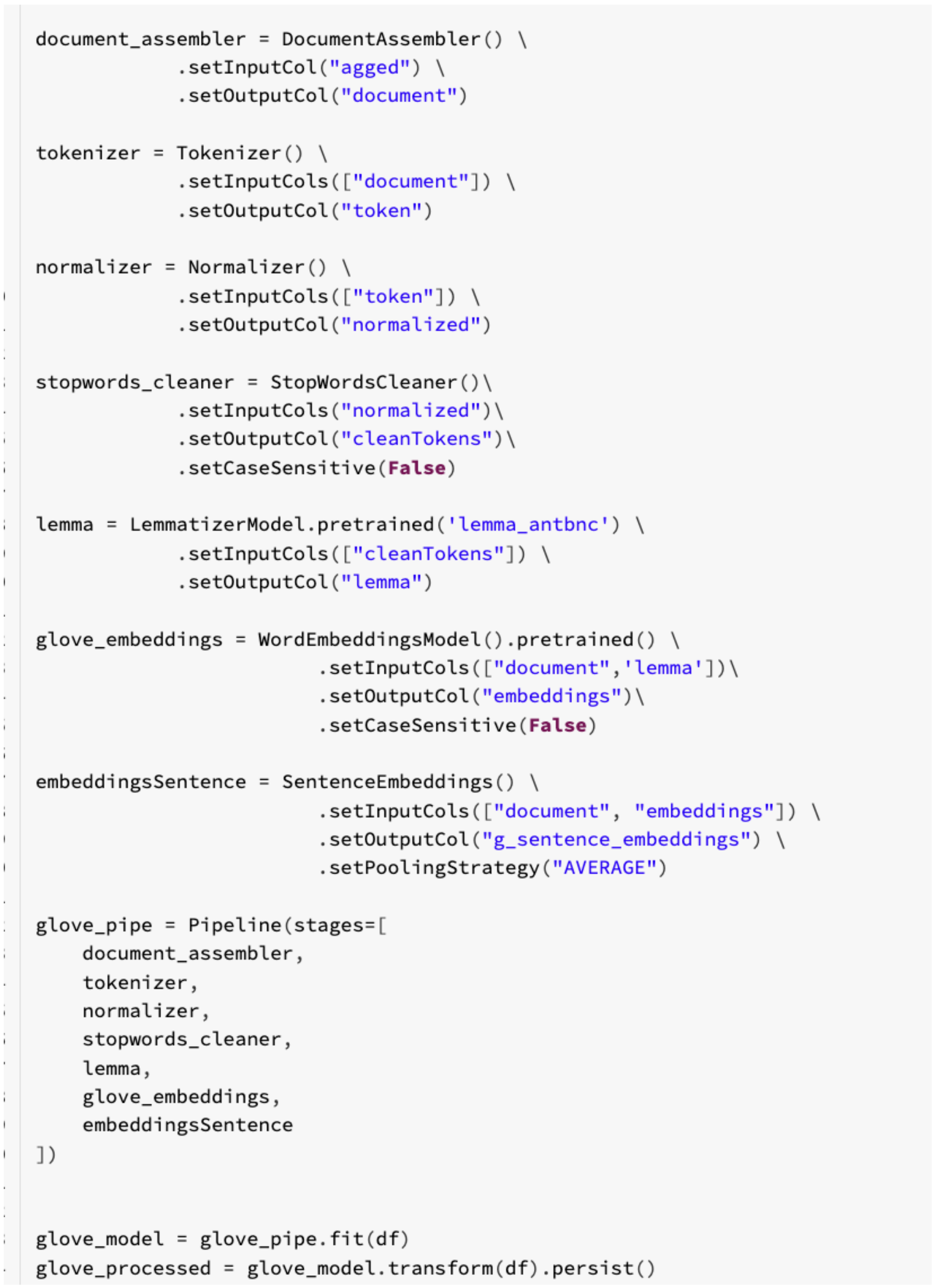
Cmd 35

```python
print(df.filter(df.public_identifier == pp.iloc[86]['public_identifier']).show(1,1000,True))
```

```
-RECORD 0-------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------
 _id                         | [64096b82b0134a6be4b2b3a0]
 city                        | San Francisco Bay Area
 cohort                      | Cohort_4
 companies                   | Facebook,Ancestry,Dictionary.com,NAVER Corp,Citic Group
 country                     | United States
 country_full_name           | United States
 exp_descriptions            |
 full_name                   | Miao Lu
 headline                    | Data Scientist at Facebook
 languages                   | Chinese English Wenzhounese
 numDegrees                  | 2
 previous_degrees            | Master’s Degree
 previous_fields_of_study    | Analytics
 previous_schools            | University of San Francisco
 public_identifier           | lumiao1105
 recent_degree               | Bachelor's Degree
 recent_field_of_study       | Management Information Systems, General
Command took 2.13 seconds -- by ppathi@dons.usfca.edu at 3/10/2023, 8:53:08 PM on SNACK_WEB_Cluster
```

2) Using GLOVE embeddings:

In this approach , we generated Glove sequence embeddings that capture word embeddings with their context and then computed cosine similarity. We built a ML pipeline in order to perform all the necessary NLP operations in order to obtain the embeddings as shown below.



```
document_assembler = DocumentAssembler() \
            .setInputCol("agged") \
            .setOutputCol("document")

tokenizer = Tokenizer() \
            .setInputCols(["document"]) \
            .setOutputCol("token")

normalizer = Normalizer() \
            .setInputCols(["token"]) \
            .setOutputCol("normalized")

stopwords_cleaner = StopWordsCleaner()\
            .setInputCols("normalized")\
            .setOutputCol("cleanTokens")\
            .setCaseSensitive(False)

lemma = LemmatizerModel.pretrained('lemma_antbnc') \
            .setInputCols(["cleanTokens"]) \
            .setOutputCol("lemma")

glove_embeddings = WordEmbeddingsModel().pretrained() \
                        .setInputCols(["document",'lemma'])\
                        .setOutputCol("embeddings")\
                        .setCaseSensitive(False)

embeddingsSentence = SentenceEmbeddings() \
                        .setInputCols(["document", "embeddings"]) \
                        .setOutputCol("g_sentence_embeddings") \
                        .setPoolingStrategy("AVERAGE")

glove_pipe = Pipeline(stages=[
    document_assembler,
    tokenizer,
    normalizer,
    stopwords_cleaner,
    lemma,
    glove_embeddings,
    embeddingsSentence
])


glove_model = glove_pipe.fit(df)
glove_processed = glove_model.transform(df).persist()
```

3) Using TF-IDF Vectorizer

In this approach, we generated the sequence to vector mapping using the popular TF IDF method. TFIDF (Term frequency Inverse Document Frequency) weights the words in documents based on how frequently they appear within a document and across all the documents.

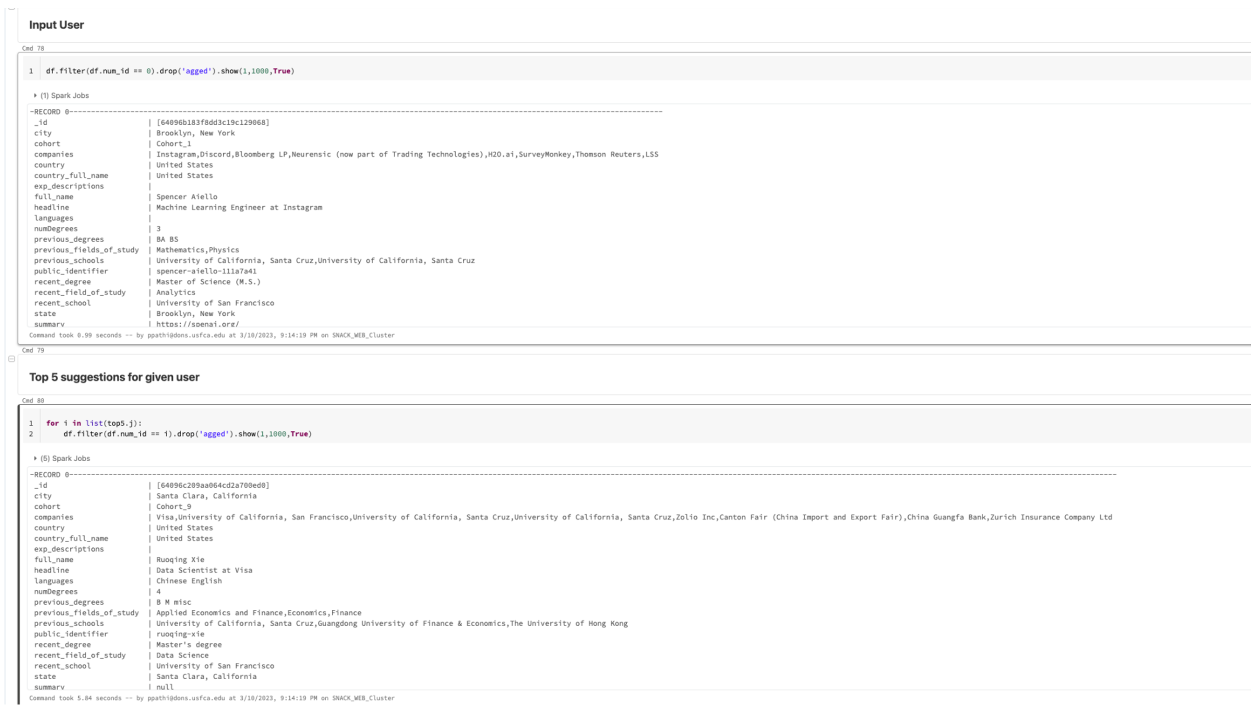
The above is implemented by using functions for Spark MLLib library and a pipeline is built chain the sequence of operations to obtain TF IDF vector.

```
tokenizer = Tokenizer(inputCol="agged", outputCol="words")
remover = StopWordsRemover(inputCol="words", outputCol="filtered")
ngram = NGram(n=3, inputCol="filtered", outputCol="ngrams")
# cv = CountVectorizer(inputCol="ngrams", outputCol="features", minDF=2.0)
hashingTF = HashingTF(inputCol="ngrams", outputCol="tf")
idf = IDF(inputCol="tf", outputCol="feature")
normalizer = Normalizer(inputCol="feature", outputCol="norm")

cv_pipe = Pipeline(stages=[
    tokenizer,
    remover,
    ngram,
    hashingTF,
    idf,
    normalizer
])
cv_model = cv_pipe.fit(df)
cv_processed = cv_model.transform(df).persist()

cv_processed.count()
```

The above approach was backed additionally by a lat_long feature engineered from the city column of the profile, in order to compute proximity by **geographical** distance obtained by the location of the alum.**Geopy** library was used in order to perform all the computations for the lat_long distance calculations among profiles.

ML Outcomes

Since, our problem of suggesting top 5 similar profiles models a **unsupervised** learning approach, we manually compared the results for each of the above discussed approaches and zeroed in the approach with **TFIDF** to compute the feature vector and use of **geographical distance** alongside in order to compute similarity among profiles. The TFIDF approach works best in our case as the vector embeddings are obtained by obtaining the weights for words within and across all documents instead of computing a contextual sentence embedding which will be an overkill in our case given the simplicity of the features we used for modeling.



Input User

Cmd 78

```python
df.filter(df.num_id == 0).drop('agged').show(1,1000,True)
```

(1) Spark Jobs

```
-RECORD 0-----------------------------------------------------------------------------------------------------------------------------
 _id                      | [64096b183f8dd3c19c129068]
 city                     | Brooklyn, New York
 cohort                   | Cohort_1
 companies                | Instagram,Discord,Bloomberg LP,Neurensic (now part of Trading Technologies),H2O.ai,SurveyMonkey,Thomson Reuters,LSS
 country                  | United States
 country_full_name        | United States
 exp_descriptions         |
 full_name                | Spencer Aiello
 headline                 | Machine Learning Engineer at Instagram
 languages                |
 numDegrees               | 3
 previous_degrees         | BA BS
 previous_fields_of_study | Mathematics,Physics
 previous_schools         | University of California, Santa Cruz,University of California, Santa Cruz
 public_identifier        | spencer-aiello-111a7a41
 recent_degree            | Master of Science (M.S.)
 recent_field_of_study    | Analytics
 recent_school            | University of San Francisco
 state                    | Brooklyn, New York
 summary                  | https://spenai.org/
```

Command took 0.99 seconds -- by ppathi@dons.usfca.edu at 3/10/2023, 9:14:19 PM on SNACK_WEB_Cluster

Cmd 79

Top 5 suggestions for given user

Cmd 80

```python
for i in list(top5.j):
    df.filter(df.num_id == i).drop('agged').show(1,1000,True)
```

(5) Spark Jobs

```
-RECORD 0--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 _id                      | [64096c209aa064cd2a700ed0]
 city                     | Santa Clara, California
 cohort                   | Cohort_9
 companies                | Visa,University of California, San Francisco,University of California, Santa Cruz,University of California, Santa Cruz,Zolio Inc,Canton Fair (China Import and Export Fair),China Guangfa Bank,Zurich Insurance Company Ltd
 country                  | United States
 country_full_name        | United States
 exp_descriptions         |
 full_name                | Ruoqing Xie
 headline                 | Data Scientist at Visa
 languages                | Chinese English
 numDegrees               | 4
 previous_degrees         | B M misc
 previous_fields_of_study | Applied Economics and Finance,Economics,Finance
 previous_schools         | University of California, Santa Cruz,Guangdong University of Finance & Economics,The University of Hong Kong
 public_identifier        | ruoqing-xie
 recent_degree            | Master's degree
 recent_field_of_study    | Data Science
 recent_school            | University of San Francisco
 state                    | Santa Clara, California
 summary                  | null
```

Command took 5.84 seconds -- by ppathi@dons.usfca.edu at 3/10/2023, 9:14:19 PM on SNACK_WEB_Cluster

Execution Times:

We had about ~ 500 records of alumni data and for the same, the execution time using a **I3.xlarge** databricks cluster for the operations in below approaches took the following times.

Bert Embedding:
Command took 2.09 minutes

Cosine Similarities:
Command took 0.39 seconds

TF-IDF:
Command took 3.43 seconds

GloVe Embeddings:
Command took 12.97 seconds

TF-IDF after Feature Engineering (Encodings & Geographical distance):
Command took 5.07 seconds

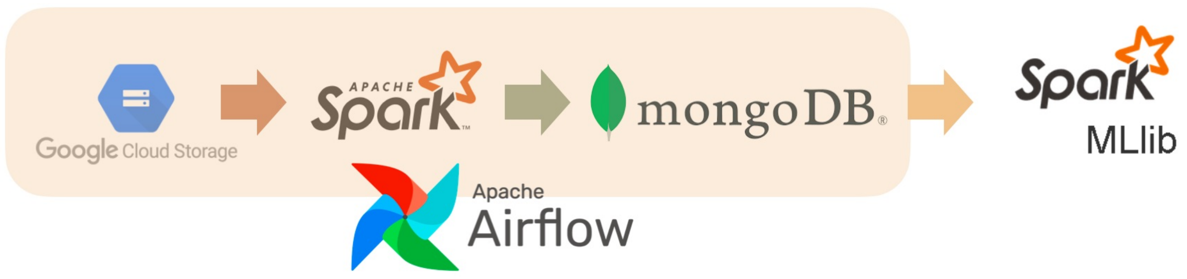
Lessons Learned

Working on this project, we as a team learnt the importance of acquiring proper data in order to build a viable ML model. Data acquired through scraping can pose a lot of missing and inconsistency issues which need to be carefully handled by a series of continuous, cleaning, transforming and preprocessing of data to achieve the best results.

Conclusion

In this way, we were able to build an end to end data processing pipeline using popular distributed computing and automated workflow processing tools and use the data to build a NLP model to suggest top 5 alumni profiles for an input profile.

.



Source code available at: Alumni Profile Matching