Module Name

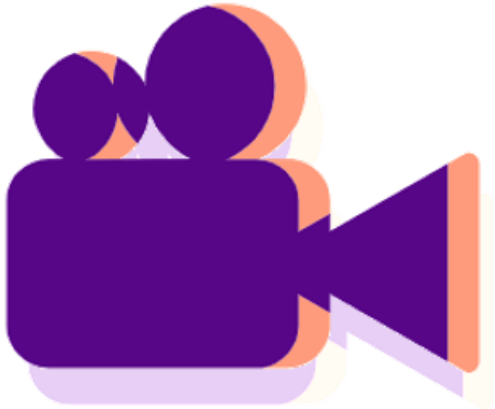
E5006 Practice Module in Big Data Engineering and Web Analytics

Title of Report

Recommendation System for MUBI website powered by Big Data Engineering

File name of the Document

EBA5006_MUBI.seek



MUBI.
SEEK

Date of Report

21 November 2020

Team 4

MUBI.seek

Team Members

Liu Yijia (A0215459Y)

Sun Jiayi (A0215385A)

Wen Jingtian (A0215275H)

Wu Yating (A0215495Y)

Zhang Siyu (A0215430W)

1. Introduction and Background

MUBI is a global film platform that provides a hand-picked films for users. The users on MUBI can enjoy the streaming ad-free on its proprietary technology, access to its international film criticism and news publication Notebook, and weekly cinema tickets to selected new release films.

The site now has 7,000,000 registered users, however, **only 1.4%** of them are paying customers, which is the main reason that the company **has yet to turn a profit**. Worse still, emerging competitors (e.g. DOGO and FMovies) are seizing the **portal traffic** from MUBI website.

The MUBI management decided to change the business model from manual movie selection to **an intelligent movie recommendation system** based on its users' behaviour. We MUBI.seek team is hired to achieve the following business objectives. Moreover, we plan to build a **friend recommendation system** for MUBI, which can also promote achievement of objectives:

- Attract more and maintain current portal traffic inbound MUBI website:
  - Build a recommendation system which can recommend MUBI users with their most-interested-and-loved movies, providing personalized experience.
  - Combine with social media analysis (Twitter streaming data) to track people's current points of interest and popularity of movies, trying to reinforce the movie recommendation system.
  - Combine with text analytics to predict the lack rating score from a user on a movie based on the user's review text, trying to reinforce the movie recommendation system.

- Improve users' stickiness and time on site:
  - Build a friend recommendation system which can recommend MUBI users with friends who have similar taste on movies, which can increase users' adhesiveness and stay time on MUBI website.

- Increase users' conversion rate to the paying subscriber
  - Getting satisfied with the movies recommended by MUBI and interactions with netizens with common hobbies, the time spent on site by users and the possibility of user conversion will increase, which is beneficial to MUBI's revenue.

The good thing is that, through previous operation, the website has accumulated a huge amount of data (2.7 GB) about movies, users and rating information, which is quite suitable to apply big data analytics on and to build a complete recommendation system, providing users with more personalized services.

2. General Architecture

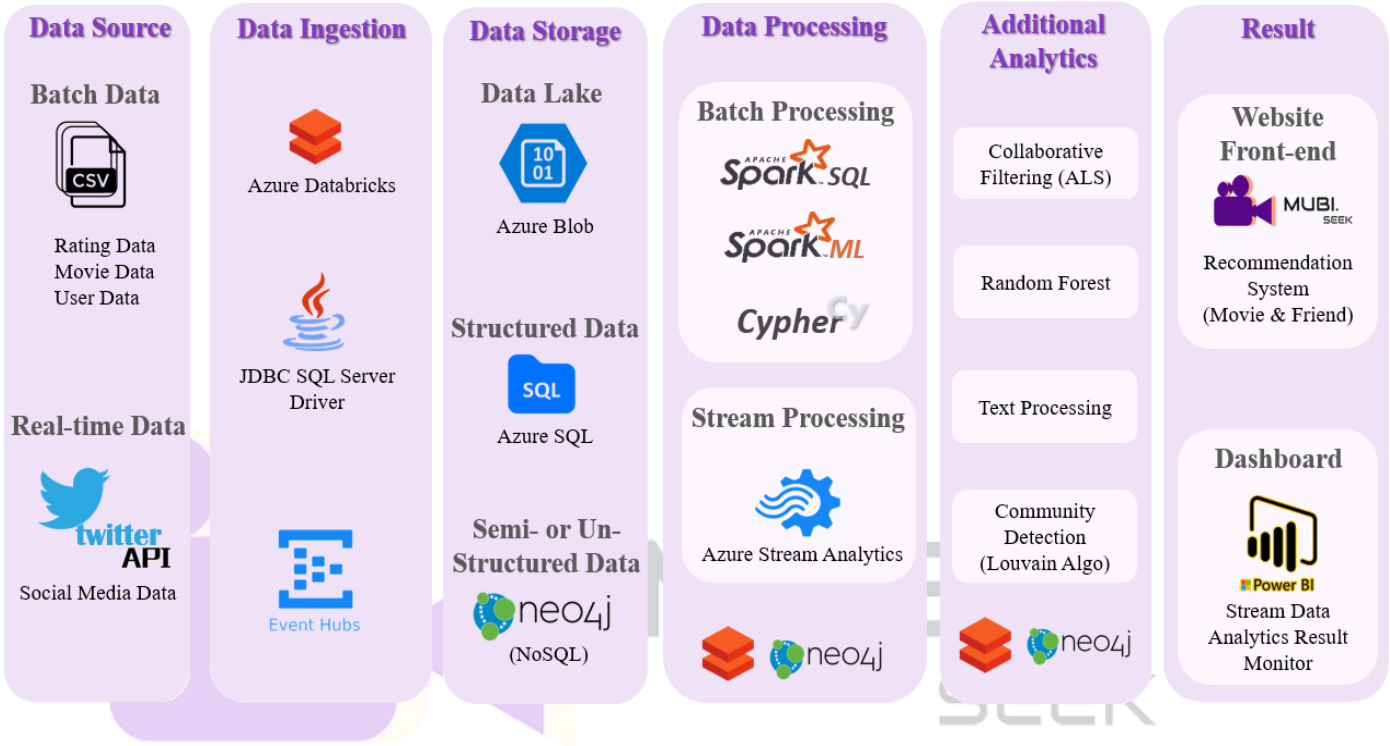
2.1 Platforms

Our project is mainly achieved by **Azure Cloud Service**. System is fully designed on Azure, including data storage, data ingestion, data engineering and AI service (data science, analytics and machine learning) on Azure Databricks. It deploys auto-scaling compute clusters with highly optimized Spark that performs up to 50x faster and enables users to store any data and share data across the virtual machines at a reliable and fast rate. What's more, Azure also enables our team to collaborate online conveniently and continuously, which means it also has the strong collaborative office capabilities in the Covid19 environment.

Besides, due to the lack of Louvain Algorithm in Spark system, we also use **Neo4j** as our graph database (NoSQL storage) and community detection analysis tool. So, the whole project is mainly based on Azure Cloud Platform, with Neo4j as the supplement.

2.2 Overall Architecture

Our system architecture consists of 6 layers, namely Data Source, Data Ingestion, Data Storage, Data Processing, Additional Analytics and Result Interpretation. The overview of our overall architecture is shown as below.



The data source consists of batch data and real-time data. Firstly, we use Azure Blob as our data lake. Then we mount Azure Blob Storage in DBFS (Databricks File Systems) and use JDBC SQL Server driver to ingest structured data to Azure SQL database. Neo4j graph database is also used to store the graph data which is transformed from Azure SQL (the structured data has been transformed into graph network). These parts will be talked about in detail later.

For different types of data, we apply different methods to do data processing, consisting of batch processing and streaming processing. As for the analytics part, we apply some algorithm such as collaborative filtering, random forest and community detection. And finally, the website front-end and power BI are used to show our analytic results.

3. Scope of work

3.1 Data Preparation

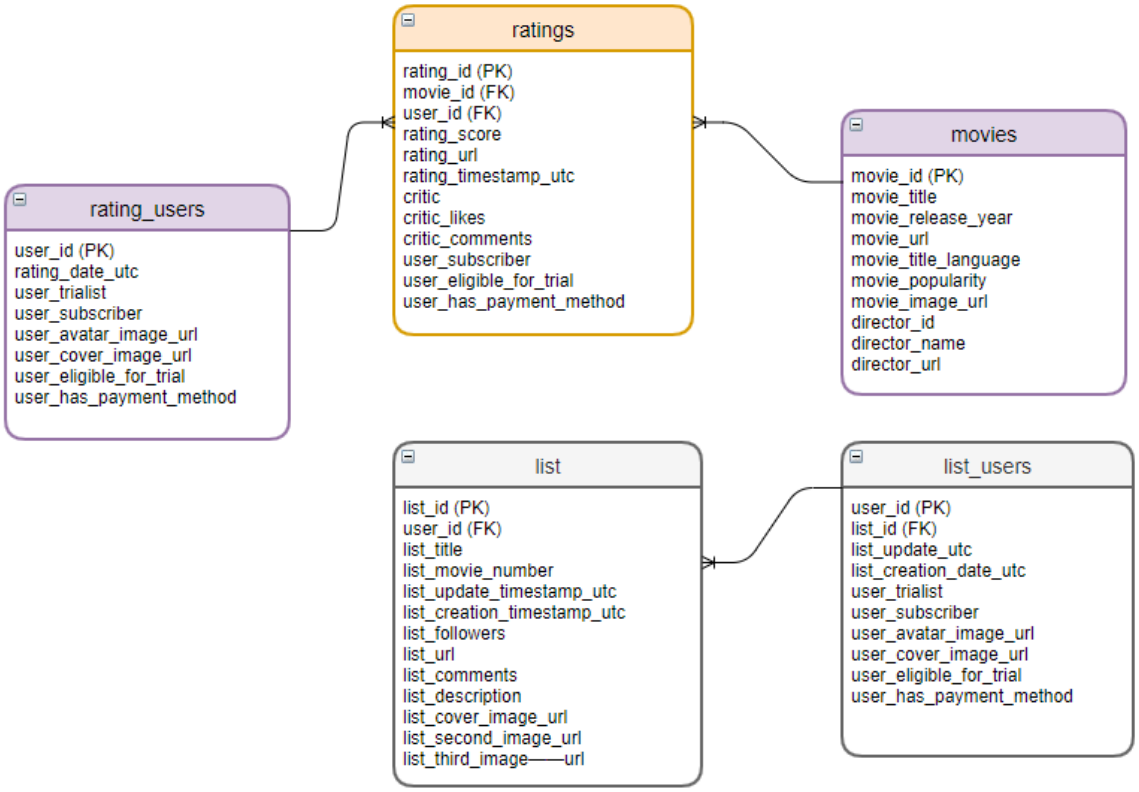
3.1.1 Data Source

There are mainly 2 data sources in our project, including structured data and semi-structured data.

- The first dataset is from Kaggle website, called MUBI SVOD Platform Database for Movie Lovers (https://www.kaggle.com/clementmsika/mubi-sqlite-database-for-movie-lovers), which is the main dataset in this project. It is about 2.7 GB, containing over 15 million movie

ratings, over 745,000 movie critics and over 225,000 movies. It also contains user IDs and user profile images.

Entity	Description
ratings	This table contains movie ratings data on MUBI.
movies	This table contains movie data for movies registered on MUBI
ratings_users	This table is three times smaller than the ratings table. It contains additional user information on a daily granularity. Only the user information related to the last rating for a specific day is stored in this file.
lists	This table contains lists data from MUBI.
lists_users	This table contains additional user information related to the list table.

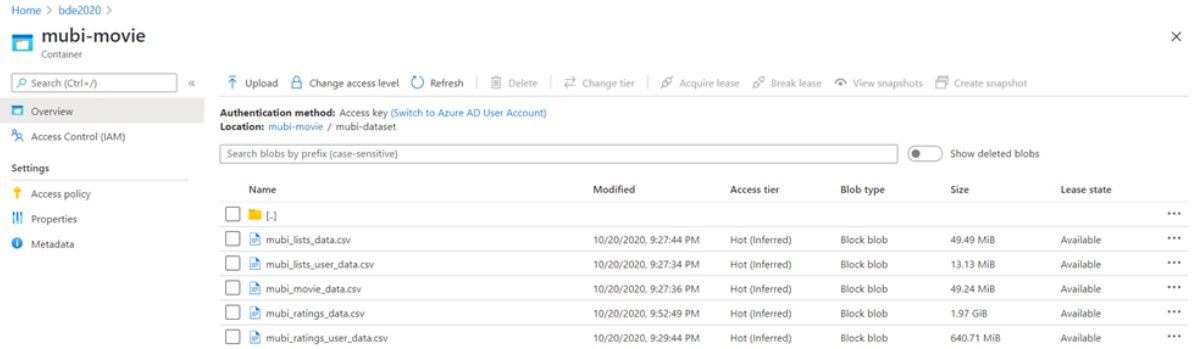


- The second data source is real-time social media data (text messages) from Twitter API (https://developer.twitter.com/en/docs/twitter-api), which will be used for social media analysis to track current points of interest and popularity of movies.

3.1.2 Batch Data Ingestion and Storage

- 3.1.2.1 Batch Data Storage

Azure Blob storage is Microsoft's object storage solution for the cloud. It could storage massive amounts of structured or unstructured data. After registration on Microsoft Azure and building a subscription, we create a (blob) storage account. Then we store MUBI batch data in storage account containers.



- 3.1.2.2 Mount Blob Storage in DBFS

Since we choose to Databricks for data engineering and data science, next step we need to connect Databricks File System to Blob Storage. We create a notebook in Azure Databricks and use Python to mount Azure Blog Storage containers to DBFS. We use the following commands to realize the mounting.

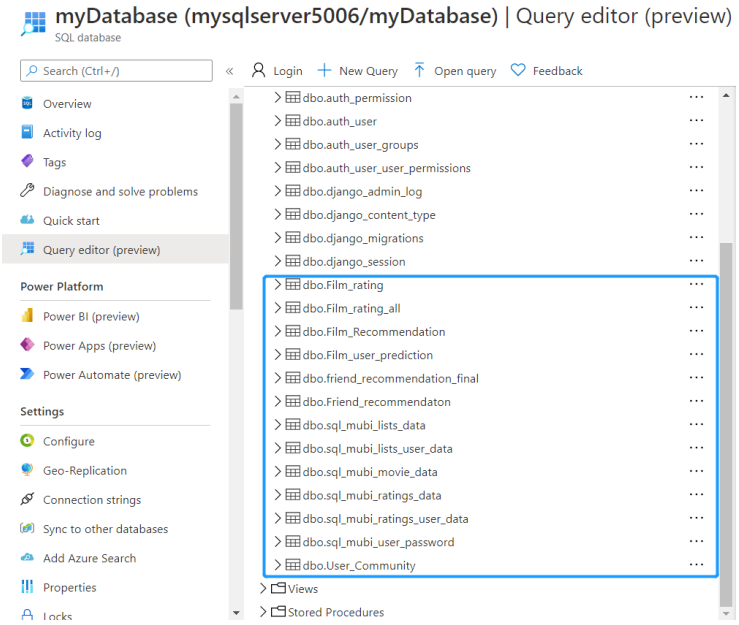
```
Cmd 1

# set up the adls connection to azure blob storage
dbutils.fs.mount(
  source = "wasbs://mubi-movie@bde2020.blob.core.windows.net",
  mount_point = "/mnt/bde2020",
  extra_configs = {"fs.azure.account.key.bde2020.blob.core.windows.net":"pbyADxHtN8msYFf6OAqCXb0kGD6qGc0Qtk61AEveir+76jlPrZmISSMUR/YLwbED+txNVAkb8GKC+xQURnN6WA=="})

▸ (1) Spark Jobs
Out[18]: True

Command took 24.64 seconds -- by liuyijia5280@163.com at 2020/11/16 下午9:35:40 on 5006-cluster

Cmd 2

# dbutils.fs.ls('/mnt/bde2020/mubi-dataset/mubi_lists_data.csv')
dbutils.fs.ls('/mnt/bde2020')
# dbutils.fs.unmount("/mnt/bde2020")

Out[19]: [FileInfo(path='dbfs:/mnt/bde2020/mubi-dataset/', name='mubi-dataset/', size=0)]

Command took 0.10 seconds -- by liuyijia5280@163.com at 2020/11/16 下午9:36:43 on 5006-cluster
```

After mounting successfully, we could access the files in Blob storage containers from Databricks.

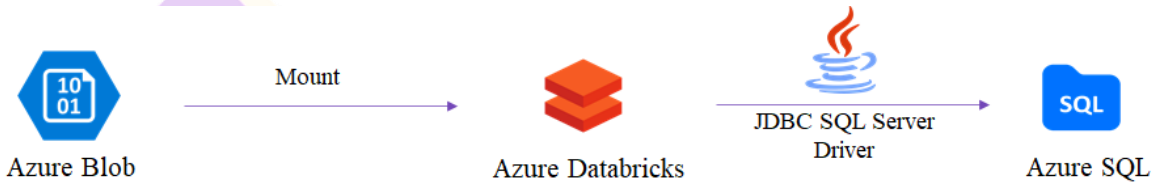
- 3.1.2.3 Data Ingestion from DBFS to Azure SQL Database

After data processing on MUBI dataset, we could ingest the data into Azure SQL Database, which could be used for further data management and data visualization. JDBC is a Java Database Connectivity Driver for use with SQL Server and Azure SQL Database. We create the JDBC URL and then push down a query to the database engine. We use 'write_to_dw' command to load dataframe to SQL Database.

```
Cmd 7

connection_string =
"jdbc:sqlserver://mysqlserver5006.database.windows.net:1433;database=myDatabase;user=azureuser@mysqlserver5006;password=BDEbde5006;encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;"

Command took 0.04 seconds -- by e0535585@u.nus.edu at 2020/11/15 下午4:54:57 on 5006-cluster

Cmd 8

def write_to_dw(df, target_table, insert_method):
  df.write.mode(insert_method)\
    .format("jdbc")\
    .option("url",connection_string)\
    .option("driver","com.microsoft.sqlserver.jdbc.SQLServerDriver")\
    .option("dbtable",target_table)\
    .option("AutoCommit","true")\
    .save()

  print("dataframe has been loaded into database")

Command took 0.03 seconds -- by e0535585@u.nus.edu at 2020/11/15 下午4:54:57 on 5006-cluster

Cmd 10

write_to_dw(mubi_lists_user_data,"sql_mubi_lists_user_data","append")
```



➢ 3.1.2.4 Summary for Batch Data Storage and Ingestion

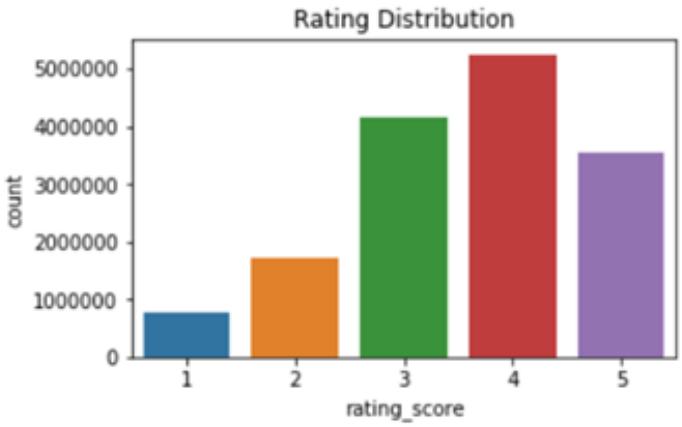


In summary, we have built an entire pipeline for batch data storage and ingestion. All the works are done on Microsoft Azure Cloud and we could use processed data in SQL DB for further website construction.

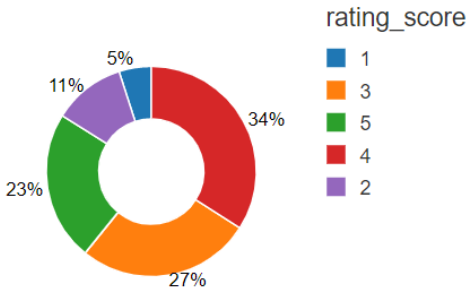
3.2 Data Analytics

3.2.1 Data Exploration and Visualization

The data source we got for EDA and visualization comes from Data Mart that was built previously, mubi_ratings_data table. Seaborn package was used to plot the Rating Distribution of the dataset.

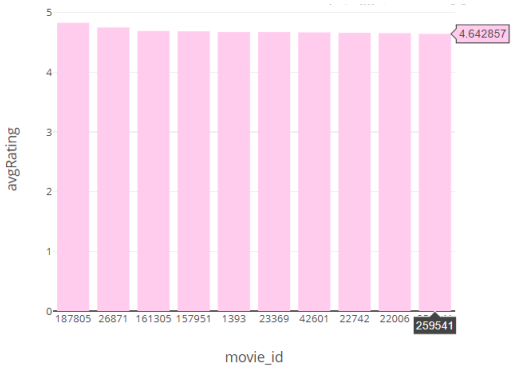


To show the specific percentage of each rating score, we also use pie chart. And from these two charts, we can know over 80% of the ratings are higher than 2.

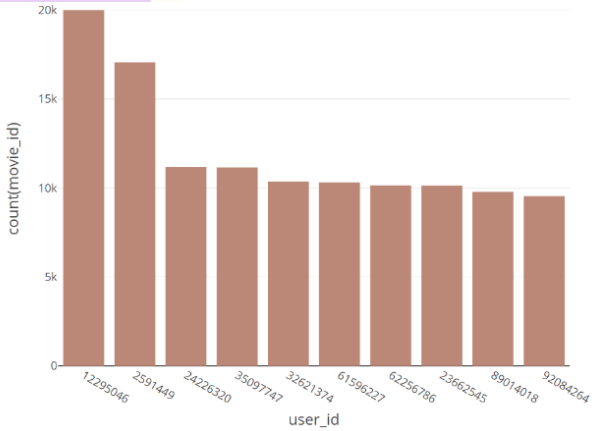


We also make use of the visualization tool in the Azure Databricks to automatically generate the remaining graphs.

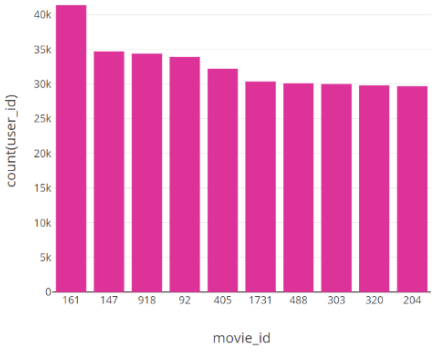
Here, we get the top 10 movies with highest ratings.



Below are top 10 active reviewers who have given out the greatest number of reviews.



The top 10 movies with the greatest number of reviews are also displayed below. And they are also the most popular movies.



3.2.2 Movie Recommendation System

- 3.2.2.1 Collaborative Filtering

In this part, alternate least square method (ALS) is adopted as the collaborative filtering algorithm.

This algorithm computes the similarity between users and between items through feature matrix, which are U(m x K) matrix representing the user's characteristic matrix, where each user is described by k characteristics, and the V(N x K) matrix representing the item's feature matrix, where each item is also described by k features, and use the similarity to predict the rating of all the movies of the user. Finally, the predicted score will be sorted and the top N movies will be returned to the user for recommendation, and the recommendation matrix will be stored in Azure SQL.

The ALS model require input data to be three columns, which are user id, item id and score. So, we drop the useless column and create a new spark dataframe containing these three columns. However, the quality of the dataset is not good. Careful data cleaning and transformation must be done. So, we filter some text data in the user_id and movie_id column and convert the dataset into integer. Furthermore, we remove the null data as well as decimal data because they cannot be converted to integer directly and may cause error.

```
#read mubi_ratings_data file from blob
rating_df = spark.read.format('csv').load('/mnt/bde2020/mubi-dataset/mubi_ratings_data.csv',header='true',inferSchema=True)
#rating_df = spark.read.text('/mnt/bde2020/mubi-dataset/mubi_ratings_data.csv')
rating_df = rating_df.select("user_id","movie_id","rating_score")
from pyspark.sql import functions as F
movie_rating = rating_df.filter(F.col("user_id").cast("int").isNotNull() == True)
movie_rating = movie_rating.filter(F.col("movie_id").cast("int").isNotNull() == True)
movie_rating = movie_rating.filter(F.col("rating_score").cast("int").isNotNull() == True)
movie_rating = movie_rating.select("user_id","movie_id",regexp_replace("rating_score",".0","").alias('rating_score'))
movie_rating = movie_rating.withColumn("user_id", movie_rating["user_id"].cast("Int"))
movie_rating = movie_rating.withColumn("movie_id", movie_rating["movie_id"].cast("Int"))
movie_rating = movie_rating.withColumn("rating_score", movie_rating["rating_score"].cast("Int"))
```

```
movie_rating = movie_rating.filter(isnull("user_id") == False)
movie_rating = movie_rating.filter(isnull("movie_id") == False)
movie_rating = movie_rating.filter(isnull("rating_score") == False)
```

To evaluate the model performance, we spilt the data into training and testing data in 70:30 proportion and use the testing data to evaluate the trained model and calculate the RMSE. And the RMSE is less than 1 which indicate that the model quality is good. Thus, we can use this to recommend movies to users.

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating_score", predictionCol="prediction")
rmse = evaluator.evaluate(predictions)

# Display using `displayHTML`
displayHTML("<span style='font-size:12pt;color:blue'>The Root Mean Square Error is %s</span>" % str(rmse))
```

▸ (6) Spark Jobs

The Root Mean Square Error is 0.9022588138400557

We deploy the model on all the registered users of MUBI, and use algorithm built-in function to recommend 5 movies ranked by predictive score to them. And the recommendation result will be displayed on a front-end website built by python Django with movie photo.

	user_id	recommendations
1	12	▸ [{"movie_id": 98752, "rating": 25.746304}, {"movie_id": 41168, "rating": 24.466278}, {"movie_id": 46327, "rating": 24.3996}, {"movie_id": 26691, "rating": 23.819828}, {"movie_id": 231047, "rating": 22.319477}]
2	27	▸ [{"movie_id": 22868, "rating": 19.355377}, {"movie_id": 89959, "rating": 18.701933}, {"movie_id": 92568, "rating": 17.246956}, {"movie_id": 41168, "rating": 16.3008}, {"movie_id": 24376, "rating": 15.108241}]
3	28	▸ [{"movie_id": 97731, "rating": 19.12667}, {"movie_id": 92235, "rating": 18.822477}, {"movie_id": 87045, "rating": 18.016947}, {"movie_id": 97141, "rating": 17.812872}, {"movie_id": 89189, "rating": 17.078222}]
4	606	▸ [{"movie_id": 89959, "rating": 27.39087}, {"movie_id": 118315, "rating": 23.779303}, {"movie_id": 22868, "rating": 23.05103}, {"movie_id": 60591, "rating": 22.787518}, {"movie_id": 63340, "rating": 21.402893}]
5	11041	▸ [{"movie_id": 22868, "rating": 24.417929}, {"movie_id": 92235, "rating": 22.972216}, {"movie_id": 89959, "rating": 22.553558},

- 3.2.2.2 Text Processing

We have tried to predict ratings by using the review text data and use random forest algorithm and sentiment analysis method to do prediction based on text features.

```
# Split the data into training and test sets (30% of totoal data for testing)
(trainingText, testingText) = df_review_clean.randomSplit([0.7, 0.3])

# Configure an ML pipeline, which consists of tree stages: tokenizer, hashingTF, and tf-idf.
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="rawFeatures")
idf = IDF(minDocFreq=3, inputCol="rawFeatures", outputCol="features")

# Pipeline Architecture - Random Forest

rf = RandomForestClassifier(numTrees = 100,maxDepth = 6, maxBins = 32)
pipeline_rf = Pipeline(stages=[tokenizer, hashingTF, idf, rf])
```

By writing some user defined functions, we remove stop words, hyperlinks, and punctuations in the review text data. Then we tokenize the cleaned review text, use TF-IDF, which can reflect how important a word is to a document, to construct features, and use RandomForestClassifier function to predict ratings. But the accuracy of this model is only 14 percent.

So further, we try another method by using Textblob, a python library, to do sentiment analysis.

We classify the ratings scores into Positive Negative and Neutral.

```
def condition(r):
    if (r >= 0.05):
        label = "positive"
    elif(r <= -0.05):
        label = "negative"
    else:
        label = "neutral"
    return label
sentiment_polarity_udf = udf(lambda x: condition(x), StringType())

def sentiment_classfication(l):
    if (l >= 4.0):
        label = "positive"
    elif(l <= 2.0):
        label = "negative"
    else:
        label = "neutral"
    return label
sentiment_classfication_udf = udf(lambda x: sentiment_classfication(x), StringType())
df_review_sentiment  = df_review_clean.withColumn("sentiment_score", sentiment_analysis_udf(df_review_clean['text'] ))
df_review_sentiment  = df_review_sentiment.withColumn("sentiment_polarity", sentiment_polarity_udf(df_review_sentiment['sentiment_score']))
df_review_sentiment = df_review_sentiment.withColumn("actual_sentiment_polarity", sentiment_classfication_udf(df_review_sentiment['label']))
```

The classification outcome will give a specific number range from -1 to 1. So we need to define the outcome number greater than or equal to 0.5 as positive, number less than or equal to -0.5 as negative and the remaining as neutral.

In the end, the accuracy of this model is 46%, which shows improvement from previous model.

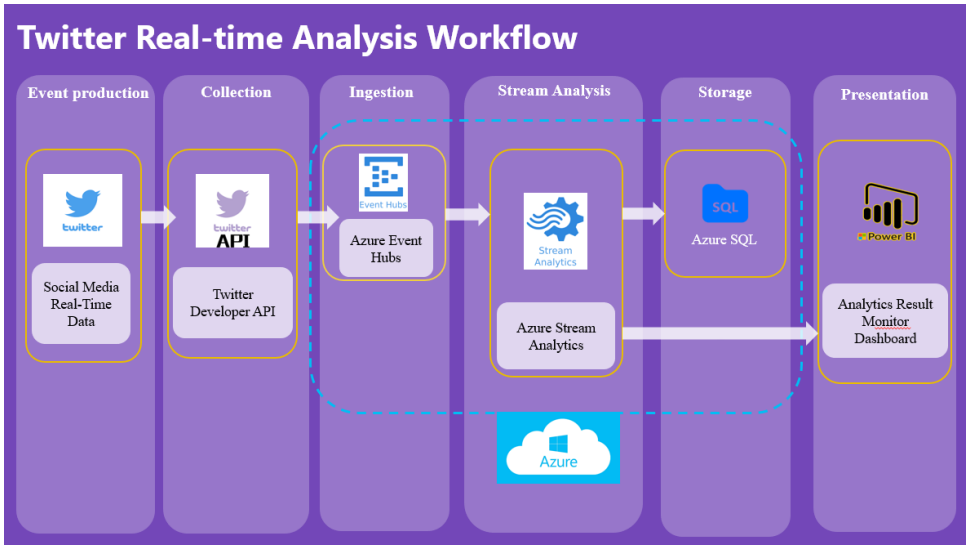
The future improvements we think can be achieved by trying some other feature extraction methods or some other prediction algorithms like Neural network.

The main difference between the basic text processing method and TextBlob method is that, by using TextBlob, we can easily create sentiment polarity, which means, we can classify each review as “negative”, “positive”, or “neutral”. And by such kind of classification, we may improve our prediction accuracy.

- 3.2.2.3 Social Media Analysis (Twitter Streaming Data)

As we all know that Twitter is a place where people share their thoughts and emotions on actual trends. It is also a strategic ground where all sorts of marketing operations happen, where brands, including companies and public personalities, can get a generous insight on the audience’s perception

from analyzing what people are tweeting about them. For our project, we focus on what people are talking about movies.



This is the workflow showing how I realize the real-time twitter data analysis.To get the access, we apply for the Twitter developer API. After building Twitter Client Core Apllication, we successfully push the real-time data to the Event Hubs. We choose Event Hubs for ingestion because it is simple, secure and scalable. Also because it can ingest data from anywhere and develop across platforms with support for popular protocols like twitter. Also, we deploy Azure Stream Analytics, which could build streaming pipelines in minutes. It can run complex analytics with no need to learn new processing frameworks or provision virtual machines (VMs) or clusters. Also, it could realize outputs into Azure SQL for storage & Power BI for the visualization and further analysis.

Azure Event Hubs contain the following key components:

Event producers: which could be any entity that sends data to an event hub.

Partitions: Each consumer only reads a specific subset, or partition, of the message stream.

Consumer groups: A view (state, position, or offset) of an entire event hub.

The client application gets tweet events directly from Twitter. For this, it needs permission to call the Twitter Streaming API. To configure this permission, you can create an application in Twitter that will generate unique credentials (such as OAuth tokens). You can then configure the client application to use these credentials when making API calls.

In practice, we track down the key word ‘movie’ to see what people are talking about on twitter. Take this tweet as an example, MUSE posted a tweet about the movie Gremlins yesterday afternoon. We ingest this real-time tweet and collect the content of the text for further analysis. This text box shows our raw data format. We can get insights from Tweets for our movie recommendation system.

Generally, below is our final result of public tweets analysis, showing what people are talking about movies, and their sentiments.

3.2.3 Friend Recommendation System

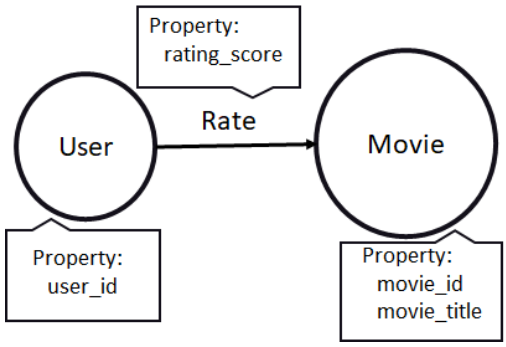
- 3.2.3.1 Why Choose Social Network Analysis

The sociality attribute of a website/app can really boost a lot. Nowadays, not only those websites specifically for SNS purpose (e.g. Facebook, Twitter) need social network analysis, many non-SNS websites also try to encourage their users to do social networking activities on their own websites, especially those based on users’ hobbies, like music website or book website.

So we try to build a friend recommendation system for users on MUBI, which is beneficial to MUBI’s customer stickiness, time spent on site, DAUs and conversion rate to paying customers.

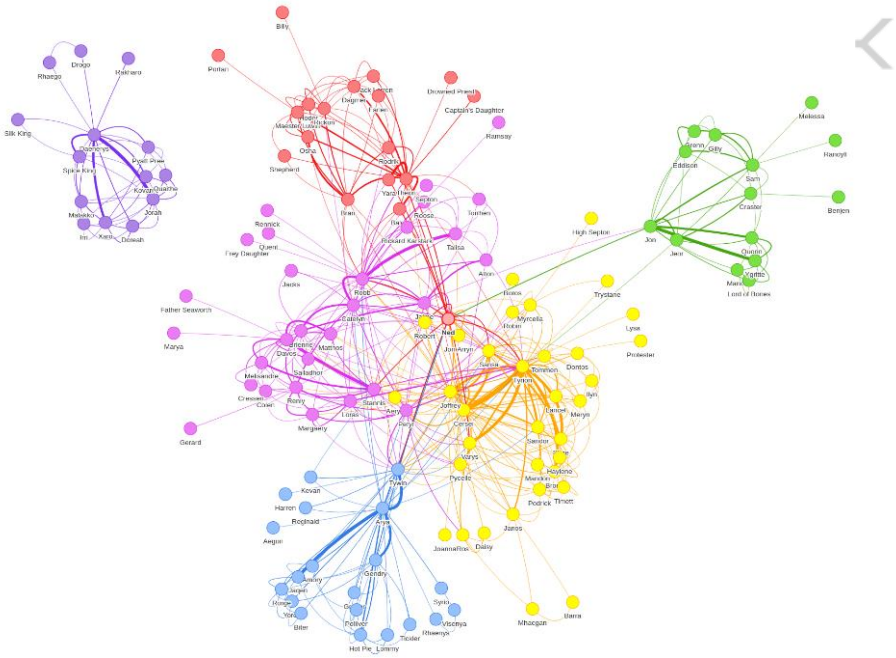
- 3.2.3.2 Graph Model and Choices of Graph Algorithm and Analysis Tool

The sheets we used to build our friend recommendation system are the “mubi_ratings_data.csv” and “mubi_movie_data.csv” from our dataset. We can firstly design a simple graph model to explore the potential relationships.



There are 2 types of nodes: user nodes and movie nodes. And rating behavior from users on movies acts as edges here. Each node and each edge have its own properties.

The whole MUBI datasets have 15 million edges. What we want to do is to cluster these users into different communities based on their rating behaviors, and recommend users in the same community to be friends. Taking an overview at all community detection algorithms introduced in our class, we find that Louvain Algorithm is the most convenient, quick and suitable one for our purpose: it can maximize the presumed accuracy of groupings by comparing relationship weights and densities to a defined estimate or average.



After comparison of graph analysis tools, we finally chose Neo4j to do this rather than Spark GraphX or Graph Frame, because Neo4j has a more complete community detection algorithm system.

Community Detection Algorithms	***Neo4j***	***Spark***
Triangle Count and Clustering Coefficient	Yes	Yes
Strongly Connected Components	Yes	Yes

Connected Components	Yes	Yes
Label Propagation	Yes	Yes
Louvain Algorithm	**Yes**	No

- 3.2.3.3 Ingestion Pipeline

For the ingestion pipeline to the Neo4j database, because there are over 15 million edges, we can't use the *Load CSV* method we've learned in class to ingest such a big data. Instead, we use *Neo4j-batch-import Method* on our computer's CMD. But before that, we should strictly modify the csv files' header format, otherwise Neo4j cannot ingest correctly.

User Nodes table

user_id:ID(User_ID)	:LABEL

Movie Nodes table

movie_id:ID(Movie_ID)	title	:LABEL

Relationships table

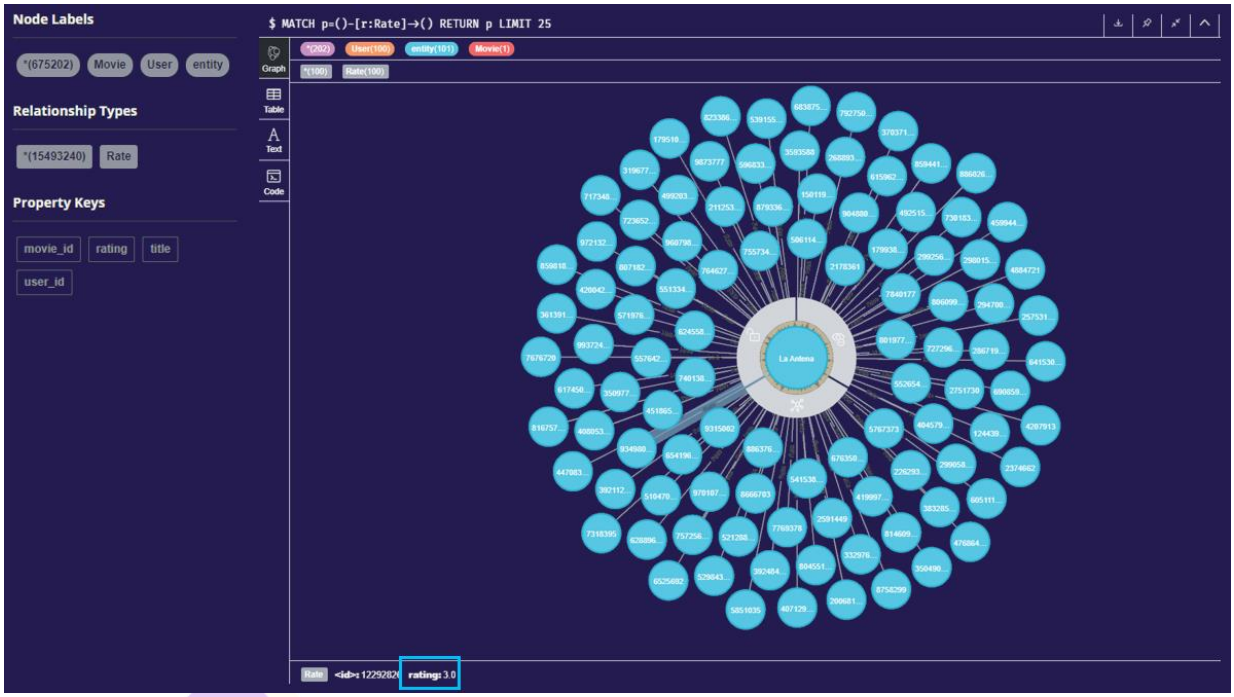
:START_ID(User_ID)	:END_ID(Movie_ID)	rating	:TYPE

Then we can enter a command line under the $Neo4j-home$ path in our computer's CMD, to import this big dataset into the Neo4j graph database, the command line is showed as follows:

```
E:\neo4j-community-3.5.14\bin>
neo4j-import.bat
--into ../data/databases/graph.db
--id-type string
--nodes:entity ../import/user_detail.csv
--nodes:entity
../import/movie_detail.csv
--relationships:rating
../import/user_movie.csv
```

- 3.2.3.4 Neo4j Graph Database

Here you can see is the graph data stored in our Neo4j database, just taking the movie "La Antena" and 100 users with their ratings on it for example.



➢ 3.2.3.5 Algorithm Cypher Code and Analysis Result

Then we can apply the Louvain Algorithm to detect communities in our database. Here is the cypher code and analysis result: for instance, we can recommend user “41579158” and user “4208563” to be friends because they belong to the same community.

```
call algo.louvain.stream('entity','Rate')
YIELD nodeId,community
return algo.getNodeById(nodeId).user_id as user_id, community
```

$ call algo.louvain.stream('entity','Rate') YIELD nodeId,community return algo…

Table | Text | Code

user_id	community
"41579158"	340301
"85981819"	167631
"4208563"	340301
"9820140"	47479
"68654088"	167631
"97262846"	167631
"52128819"	167631
"57756708"	167631
"58420503"	167631

➢ 3.2.3.6 Analysis result deployment

Finally, we can export our analysis result as a csv file, merge it with the user_avarta_image_url column and upload it to our Azure SQL platform using Spark SQL and PySpark in Databricks.

```
user_community_data = spark.read.format('csv').load('/mnt/bde2020/mubi-dataset/user_community.csv',header='true',inferSchema=True)
mubi_ratings_user_data = spark.read.format('csv').load('/mnt/bde2020/mubi-dataset/mubi_ratings_user_data.csv',header='true',inferSchema=True)
user_image_data = mubi_ratings_user_data.select("user_id","user_avatar_image_url")
# ta for community info
ta = user_community_data.alias('ta')
# tb for image info
tb = user_image_data.alias('tb')
left_join = ta.join(tb, ta.user_id == tb.user_id,how='left')
left_join.show()
friend_recommendation_table = left_join.select("community","ta.user_id","user_avatar_image_url")
friend_recommendation_table.show()
```

```
def write_to_dw(df, target_table, insert_method):
  df.write.mode(insert_method)\
    .format("jdbc")\
    .option("url",connection_string)\
    .option("driver","com.microsoft.sqlserver.jdbc.SQLServerDriver")\
    .option("dbtable",target_table)\
    .option("AutoCommit","true")\
    .save()

write_to_dw(friend_recommendation_table,"friend_recommendation_final","append")
```

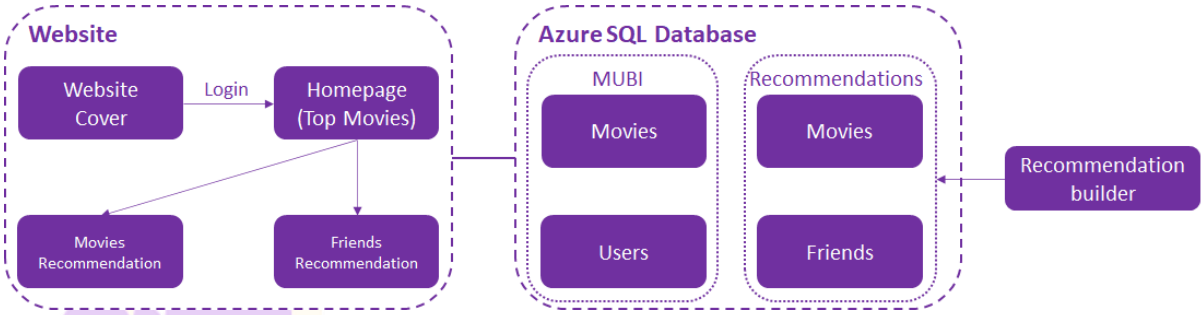
The merged and final “friend_recomendation_table” is shown as follows:

```
+---------+-------+--------------------+
|community|user_id|user_avatar_image_url|
+---------+-------+--------------------+
|     1183|    606| https://graph.fac...|
|   167631|    853| https://assets.mu...|
|     1183|  11041| //mubi.com/assets...|
|   193324|  11841| //mubi.com/assets...|
|   193324|  12127| //mubi.com/assets...|
|    47479|  12393| //mubi.com/assets...|
|    47479|  12393| //mubi.com/assets...|
|    47479|  12393| //mubi.com/assets...|
|    47479|  12393| //mubi.com/assets...|
|    47479|  14102| //mubi.com/assets...|
|   193324|  15634| https://graph.fac...|
|     1183|  16295| //mubi.com/assets...|
|     1183|  20018| https://graph.fac...|
|     1183|  20018| https://graph.fac...|
|     1183|  20018| https://graph.fac...|
|     1183|  20018| https://graph.fac...|
|   167631|  25831| https://graph.fac...|
|   167631|  26646| //mubi.com/assets...|
```

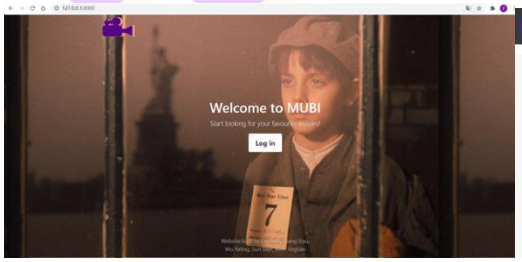
The MUBI deployment team member can apply Spark SQL language on our final table, to achieve friend recommendation.

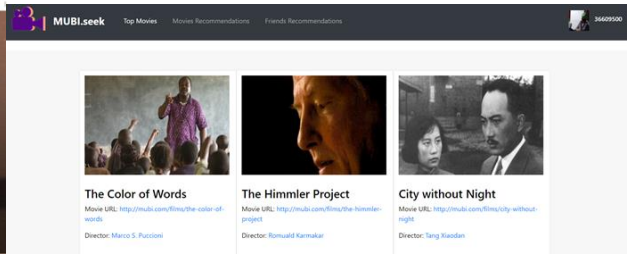
3.2.4 Web Front-end Deployment

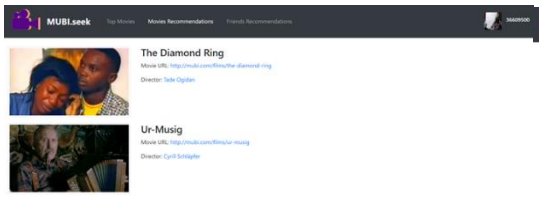
We used Django framework to build a demo website 'MUBI.seek' to show the results of movies and friends recommendation. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. We connected PyCharm to Azure SQL Database thus to extract processed MUBI data and recommendation data.



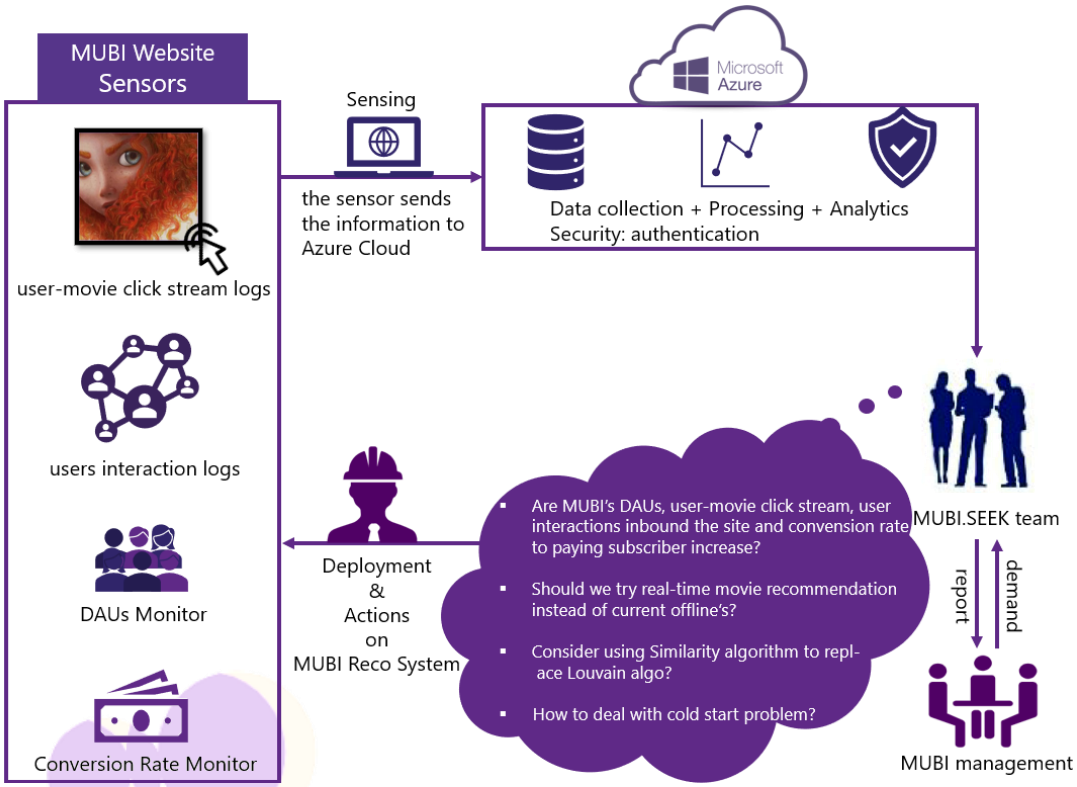
On the MUBI.seek website, we recommended 5 movies and 15 users for test MUBI user '36609500'.







4. Always-on Functional Architecture



5. Implementation Journal Report

	Milestone Capability		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
Workstream	**Initiative**	**Assigned**														
Project Preparation	Literature Survey	All	X													
	Problem Study	All		X												
	Observations and analytics model formulation	All		X												
	Solution proposal	All			X	X										
Project Implementation	ETL	Liu Yijia					X	X								
	Movie Recommendation	Wen Jingtian							X	X	X	X	X			
	Social Media Analysis (Twitter Streaming data)	Sun Jiayi							X	X	X	X	X			
	Text Analytics	Wu Yating							X	X	X	X	X			
	Friend Recommendation	Zhang Siyu							X	X	X	X	X			
	Web Front-end Design	Liu Yijia, Wen Jingtian									X	X	X	X		
Demonstration	PPT Design and Rehearse	All												X	X	
Report		All													X	X

*Granularity – day

6. References

[1] Falk, Kim. *Practical Recommender Systems*. Manning Publications Company, 2019.

[2] Njray. "Movie Recommendations on Azure." *Azure Architecture Center*, 1 Sept. 2019, docs.microsoft.com/en-us/azure/architecture/example-scenario/ai/movie-recommendations.

[3] MSFT, Gregory Suarez. *Using Apache Flume with HDInsight*. 18 Mar. 2014, web.archive.org/web/20190217104751/blogs.msdn.microsoft.com/bigdatasupport/2014/03/18/using-apache-flume-with-hdinsight/.

[4] Hrasheed-Msft. "Connect to Kafka Using Virtual Networks - Azure HDInsight." *Connect to Kafka Using Virtual Networks - Azure HDInsight | Microsoft Docs*, 4 Mar. 2020, docs.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-connect-vpn-gateway.

[5] Hrasheed-Msft. "An Introduction to Apache Kafka on HDInsight - Azure." *Microsoft Docs*, 25 Feb. 2020, docs.microsoft.com/en-us/azure/hdinsight/kafka/apache-kafka-introduction.