Movie Recommender system using Spark and ElasticSearch

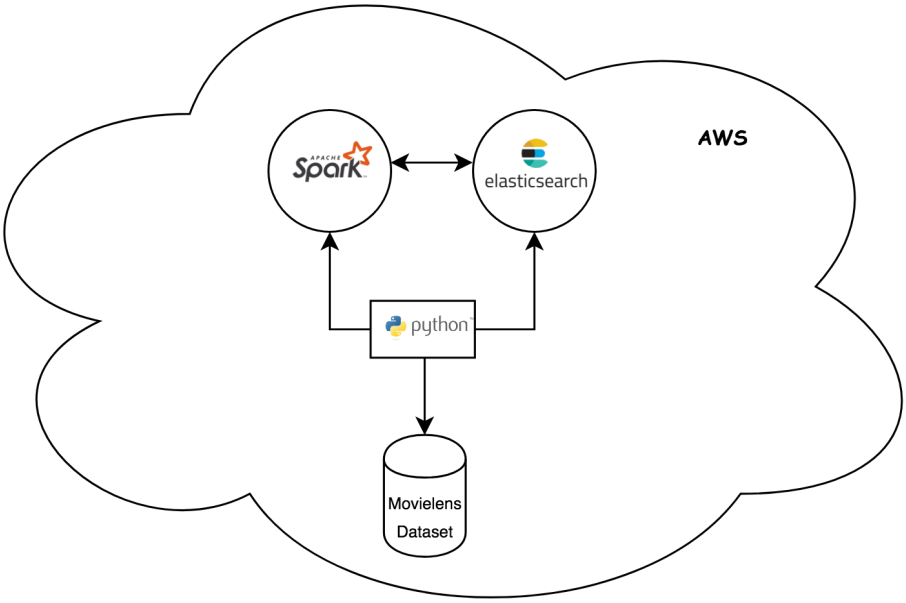
Pranav Kumar Sivakumar, Shemal Somil Lalaji

Problem Description:

Recommendation engines are one of the most commonly used machine-learning based application that can be easily implemented. But the issue of huge datasets makes it harder to develop one of them.

In this project, we propose an approach to provide real-time recommendations, by constructing a large-scale recommender engine using big-data technologies like Apache Spark and ElasticSearch. This approach is highly scalable as it involves frameworks based on distributed systems and cluster computing. So we will be utilizing the related distributed computing services provided by Amazon Web Services (AWS) to accelerate our implementation.

Apache Spark is a unified analytics engine that also has an inbuilt module for implementing machine learning based computations called MLlib. So we will be using it in our project as well for training the model in our recommendation system.



Solution Approach (in High level):

The following steps are the tentative sequence of tasks that we plan to follow in our project,

1. The first part is the data preprocessing which will involve some basic text manipulation.

2. After preprocessing the dataset that contains the movie ratings, we plan to train a collaborative filtering model in Spark using RDD operations and MLlib.

3. We plan to load the data in ElasticSearch after designing a schema for the database. We will be using the Python wrapper for Elasticsearch and probably a connector for Spark and ElasticSearch. We would have to index for mappings for users, movies, and ratings.

3. After successfully loading it, we'll be training the Collaborative filtering model on that data. We chose the collaborative filtering method as it is a sort of technique that enables to model the similarity between the items(movies in our project) based on other similar user's preferences. For this, we would have to explore libraries/methods like ALS(Alternating Least Squares) in MLlib. Our main emphasis in this project is on Spark and ElasticSearch instead of Machine Learning since it is a very small component of our project.

4. Load the trained model's factors into Elasticsearch and probably perform some Dataframe operations for indexing.

5. The last step will consist of generating the recommendations. To do this, we will create some utility functions that handle different tasks such as finding similar movies based on another movie, or based on user preferences, fetching movie meta-data etc. After getting the results from the queries, we plan to display them effectively.(For this we are going to use tables and poster of the movies from the data to display the results generated via elasticsearch.). In case we finish the project early, we plan on using Kibana to display the results in a more better way.(We plan to use Kibana in our project.)

Dataset:

The dataset will be the Movielens 20M dataset that consists of ratings given by a set of users to movies, combined with their metadata information. It was collected and made available by the GroupLens Research.

It is a standard benchmark dataset with 20 million ratings. It includes tag genome data with 12 million relevance scores across 1,100 tags. The dataset describes 5-star ratings and other tags. It contains 20000263 ratings and 465564 tag applications across 27278 movies

The data are contained in six files: genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv, and tags.csv.

For data collection, the users were selected at random for inclusion and they had to rate at least 20 movies. User ids are consistent between ratings.csv and tags.csv (i.e., the same id refers to the same user across the two files).

Considering movies column, only those movies with at least one rating or tag are included in the dataset. Movie ids are also consistent between ratings.csv, tags.csv, movies.csv, and links.csv (i.e., the same id refers to the same movie across these four data files).

The data is a static and open source dataset, so there is no need for a developer account as it can be easily accessed. This dataset, as well as the others, are available for free download at http://grouplens.org/datasets/.

The data will be downloaded and processed on our desktop. The data require some preprocessing steps such as removing white spaces and separating some data columns in the dataset. After that, we can use the data in our project.

Challenges:

The problem is hard in the sense that it has many components attached to it which we need to solve as well as issues in integration with ElasticSearch, Spark and Machine Learning.

Since both of us are new to Elasticsearch, we may come across components of ElasticSearch that we may take time to understand. There will be situations where we need to resolve issues in our machine and try to install ElasticSearch in the right way in our machines.

Another issue is that the dataset is huge and using it on a complex architecture of Elasticsearch and Machine Learning will add to the preprocessing complexities of the project.

One of the major components of this project is integrating Machine Learning with ElasticSearch and Spark. Our goal is to give the best recommendations possible based on the given preferences in real-time.

Timeline:

Assuming that we are going to start working on the project from October 31st and will have to submit it in the middle of December, this is the predicted timeline that we have come up with.

The project consists of the following parts and the expected Deadline of each of them:

1. Load the MovieLens 20M dataset into Spark.
2. Make use of Spark DataFrame operations to clean up the dataset. After the cleaning is finished, it would be loaded into Elasticsearch.

   Both these parts will be done in 1-2 weeks

3. Using Spark MLlib, train a recommendation system based on collaborative filtering.

   This is the Machine Learning part of the project and will take around two weeks.

4. Ingest the trained model into Elasticsearch.
5. Using Elasticsearch queries and some custom scoring methods, generate sample recommendations and test the recommender system.

Both of these will take the remaining time of around 2-2.5 weeks left for the project.

It is not feasible in this project to do two different operations in parallel, so we plan on doing the project by sitting together until completion.

Checkpoint 1:

We are not making any changes to the proposal at the current moment. Instead, we plan to add Kibana to our project for visualization.

Our timeline at the current moment of time looks like the following:

A) Word Done till now:

As per our proposal, we were supposed to do the following in the first 2 weeks:

1. Load the MovieLens 20M dataset into Spark.
2. Make use of Spark DataFrame operations to clean up the dataset. After the cleaning is finished, it would be loaded into Elasticsearch.

The work done by us is mentioned below:

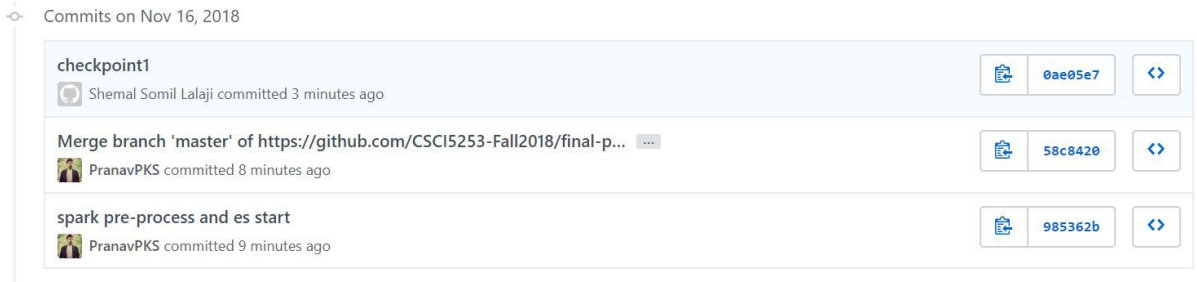
1. We have installed ElasticSearch, Kibana, and Spark in our systems.
2. We have done the data preprocessing using Spark after loading it.
3. We have indexed our dataset in ElasticSearch

Most of the work done so far was the setting up the environment (Elasticsearch, Kibana, and Spark) and learning to index in Elasticsearch. So we have not pushed the work to GitHub at regular intervals.

Once we had reached our checkpoint1 goal, we have pushed it into the GitHub. So the number of commits is just one.

As we mentioned in the proposal, the work done by us in the project was in parallel.

Image of Commits:



B.) Work to be Done:

We are going as per the project proposal. So the following work is left to be done:

1. Using Spark MLlib, train a recommendation system based on collaborative filtering.
2. Ingest the trained model into Elasticsearch.

3. Using Elasticsearch queries and some custom scoring methods, generate sample recommendations and test the recommender system.

   The timeline of the project is the same one as we mentioned in the proposal.

   We have not incurred any more costs right now and do not expect any cost in the future with regards to our project.

   The data preprocessing that's been done so far:

   1. We have separated the genre of each movie into a list so that we can use them to query in our Elasticsearch more efficiently.
   2. Another feature of our data included the property that the titles of the movie also contained the year in which they were released. We have modified this property by separating the year and the title so that we can use them discreetly in the search queries.

Checkpoint 2:

In this part we started with the training of the ALS model then optimizing its parameters. Later ingested the model factors into the ElasticSearch (for querying purposes) and generated recommendations for some sample data.

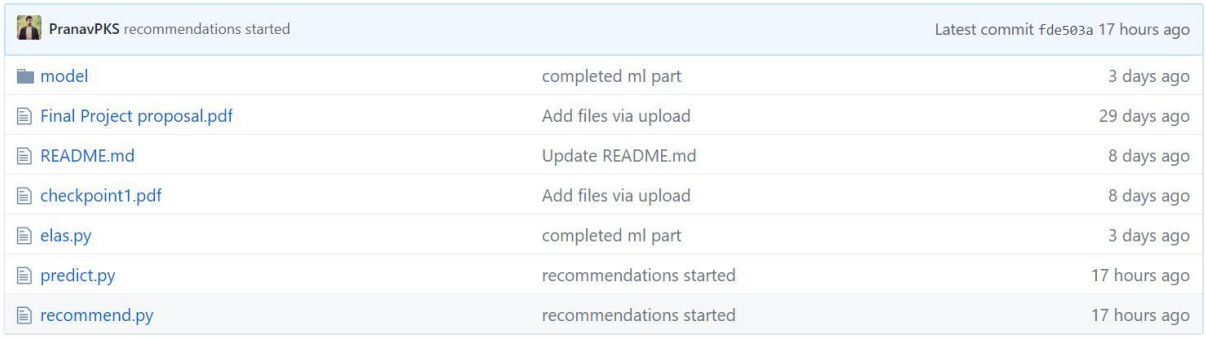
We will present this checkpoint in the following way:

- First we will show the work that we have done via screenshots and code.
- Secondly we will explain the work that we have done as per our report and timeline in the project proposal
- Finally we will describe the work that is left for us to do.

Work in Github:

We will summarize all the list of tasks that we have done in the interval between checkpoint 1 and checkpoint 2 in detail this time as per the suggestions we received in the feedback of checkpoint 1.

In the description it was mentioned that we shall make more advances in the code part of the project. We have definitely worked on it as can be seen as per the image below:



In the recommend.py,we have defined utility functions that help us to achieve the recommendations. There are functions that define the similarity of movies for our project. Also along with that, we have defined the cosine similarity metric for our project that we have made use of. The utility function also allows us

to convert the model vector back and forth in our implementation. The model vector is then stored in our elasticsearch.

Also for our project, the most important method is the fn_query method. This method contains the vector plugin which we use to query against the model vector and the query is structured on the user's preferences.

```
def fn_query(query_vec, q="*", cosine=False):

    return {
    "query": {
        "function_score": {
            "query" : {
                "query_string": {
                    "query": q
                }
            },
            "script_score": {
                "script": {
                        "inline": "payload_vector_score",
                        "lang": "native",
                        "params": {
                            "field": "@model.factor",
                            "vector": query_vec,
                            "cosine" : cosine
                        }
                    }
            },
            "boost_mode": "replace"
        }
    }
}
```

Another key aspect in this function is the payload vector scoring. It is a metric which helps us in a custom analyzer to assign a scoring to the raw vectors. This score is useful when we try to query using elasticsearch as it associates the query which we want with some payload and thereby gives us the flexibility in determining the best recommendations

In predict.py, we have written functions that allow us to store the model in the format that is suitable for the index in the ElasticSearch. They allow us to convert the raw vectors into the proper string format. Unless this is done properly, Elasticsearch would not work on the vector field built using the custom analyzer that we have specified in fn_query method

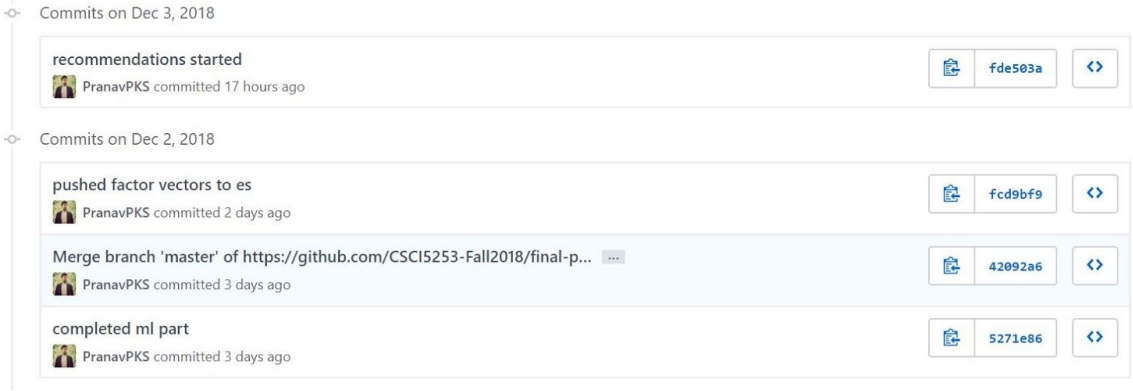
Also we have to create additional functions for mapping all of the data with the model of the Elasticsearch index mappings to ingest the data in the it for querying.

```
model.itemFactors.select("id", vector_struct("features", lit(ver), ts).alias("@model")).write.format("es") \
    .option("es.mapping.id", "id") \
    .option("es.write.operation", "update") \
    .save("demo/movies", mode="append")
```

The above code is used for ingesting data of Items in the elastic search using the elasticsearch connector with options of updating and appending available with alias as @model used.

The trained ALS model is saved in the model folder which contains its corresponding vectors and metadata. The ALS functionality was utilized from the MLlib framework of Spark.

The version of commits are as follows:



We pushed it to the github repository whenever we finished implementing a functionality (that we wanted) as a whole instead of minimal changes. Please note that these commits were made during the project report making and there will have been additional commits as we prepare to proceed beyond our checkpoint 2.

Work as per the proposal and the timeline discussed in the proposal:

Based on the comments that we have received in our previous checkpoint, we have modified the work in the timeline and added more work in our project to satisfy the expectations and increased the scope of our project.

We have completed the following:

A. Using Spark MLlib, train a recommendation system based on collaborative filtering.
B. Ingest the trained model into Elasticsearch.
C. Using Elasticsearch queries and some custom scoring methods, generate sample recommendations and test the recommender system.

For part A, we have made use of collaborative filtering and separated the user ratings and movies ratings with a type of technique called ALS (Alternating Least Squares). This type of filtering is very rarely used and is only used here as it is suitable for the MovieLens dataset.

The core idea used here is Matrix Factorization (MF) of ratings into users and movies and using the vectors obtained to query for elasticsearch.

The part B is made easier due to the work done by the functions in the elas.py code. Here we used the elasticsearch-spark connector to convert the spark dataframe to be inserted into the elasticsearch for querying.

The Part C is still something that we can add more to our work for the final work alongside working with kibana. We will include more description of Part C in our project report as it can be augmented with a lot of other things.

Work to be done:

1. We are also using different types of recommendations to explain the power that elasticsearch gives us in our project.
2. We are also trying to work on a larger piece of data for our project. Currently we are using a smaller amount of data for our convenience.
3. We are currently working on kibana to visualize the results we get via indexing using elasticsearch.

Meetings done by us and other key points:

Since the project requires us to work together for most of the time, we are working together in the campus and updating the github whenever we reach goals that we have decided for our checkpoint. Meetings are done like 3-4 times per week for 5-6 hours and communication for proposal is done over the email/text message.

There has been no cost incurred to us until now.

The challenge right now for us is to merge our elasticsearch with kibana and decide what we need to visualize with respect to our project.The other challenge is work with bigger dataset. Since we were done with the data preprocessing in the first checkpoint, we just need to use similar process on our bigger dataset.

We have added some work to our timeline and as this project requires us to sit together and work in parallel, we have been following that approach.

it would be good to add more details about what's being added in part c. don't skimp on it for the final project!