Malavika:

1.

Good morning all. I am Malavika and this is Christoffer sharing the screen, and the title of our project is search engine for movie cast generation. ***NEXT***

2.

The film industry in itself is a billion-dollar business which has a huge economical impact. The success of a movie depends highly on its cast. That is, its lead actors. ***NEXT***

3.

And for directors choosing actors for their movies, it is a challenging task given the huge number of actors and movies they make every year. Manually skimming through the humongous data to find relevant actors is nearly impossible. In order to assist them put together actors based on their requirements is what this project is about. ***NEXT***

4.

This leads us to our formal problem statement. The user is required to give three inputs, namely, the genres, the actor(s)' description such as gender and age interval along with a plot summary for the movie they are planning to make. Our system would provide a ranked list of relevant actors or group of actors (based on user input). ***NEXT***

5.

Here is a use case example. Let's say the user inputs are as follows. Genre being Adventure, Drama and Scifi. The user wants three actors, a male between age 45 and 55, two females between 30 to 40 and 40 to 50 respectively. The user has also given a short description of the movie plot

Our system takes these inputs and gives out a ranked group of actors as follows. The top ones are Matthew McConaughey, Anne Hathaway and Jessica Chastain with a group score of 9.1. Movie buffs would immediately recognise that this is the cast of the blockbuster Interstellar. We gave the input genres, actors' descriptions and plot of interstellar as It is from IMDb. ***NEXT***

******

Christoffer:

6.

Most of our data comes from the IMDb datasets. They have information about 6.5 million titles and almost 10 million actors. This data includes ratings, genres, movie casts and more. The only data not available from there is the plot summaries which we get from the OMDb API instead.

7.

We are using the same hardware as everyone else on our cluster and are using Hadoop version 3.2.1 and the preview version of Spark 3.0.

8.

We do the following preprocessing on the data.

1. Remove titles that are not a movie.
2. Remove movies that are missing the rating or genres.
3. Remove dead actors and calculate the age on the other ones.
4. Determine gender based on their profession.
5. Generate genre scores for each actor.
6. Generate a graph of the actor to actor relationships.

9.

Skip

10.

There are three main parts to the search algorithm.

1. The first one is to filter out any actor not satisfying the search query.
2. The next step is to calculate the actor rank which is a combination of the genre score and the plot similarity score. The genre score is the average rating the actor has received for that genre, and the similarity score is the maximum similarity between the plots the actor has acted in and the search plot.
3. The last step is to generate groups of actors and their relation scores.

Both the ranked list of actors and the ranked groups are returned to the user.

******

Malavika:

11.

We also did the implementation in python to compare the performance between spark and pure python. The python workflow is essentially the same as that of spark. ***NEXT***

12.

Speaking of the challenges we faced during this project, these three tops the list. ***NEXT***

13.

The first one was the evaluation of plot similarity. After we generate a candidate actors' list based on their descriptions, we look for similarity between the input plot and plot summaries of the movies that these actors have acted in previously. For this purpose, we first tried the cosine similarity with tf-idf, which is well-known for detecting similar documents in text mining. But this gave us unsatisfactory results since, this model depends highly on the number of words in a document. Which is why this was a bad choice for our task, since our plot summaries are small, with very few words. We then tried the word2vec model which is a neural network as the same suggests converts words to vectors and computes various mathematical distances between them. This method was highly promising but yielded poor results since it was computing word-wise similarity while we needed sentence-synonymity.

That is when we chose the universal sentence encoder tensorflow model by google. This is also a word2vec model but is instead trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. This perfectly matched our criteria and gave excellent

results. This model would return a value between 0 and 1 which we multiplied by 10 to normalise it like the genre score. ***NEXT***

14.

The bigger problem with using this model on spark is its size which is nearly 1GB. It usually halts for 30 seconds before the import is done. Loading the model was equally expensive, thus the total time on each partition was a minute before it could start. Because there were multiple partitions per node, the model was loaded several times on each of them. Which is highly expensive and was unacceptable. We looked for various alternatives and ended up doing the calculation on the master node. This means that all of the data had to be transferred over the network to the master node, and then back again to the slave nodes which is not advisable but this was actually faster in our cluster than the former method. The runtime of similarity score is typically about 8 minutes. ***NEXT***

15.

Another major issue was the evaluation of results. We were posed with the question: How do we measure the quality our results? First off, we assume that any movie with a higher rating on IMDb has a good cast. That is, for a given genre, if out of two movies, one has a higher user rating, we assume that its actors are good performers in that genre and vice versa. So we decided to check our system by giving existing movie attributes from IMDb as input, if the original actors show up early in our results, it is considered a good output.

As shown in our use case example, when we gave attributes of the movie interstellar as input, we got the original actors with the highest score. Which is exactly what we want. 😊

******

Christoffer:

16.

The main problem with the relation score is that there is so much data to process. The graph consists of almost one million nodes and each node has between 10 and 500 edges. Using the Interstellar input and Dijkstra's shortest path algorithm we found that calculating one relation score takes seven and a half minutes. If we assume that we are looking for three actors and take the top 30 actors from each list then we need to find 2700 relation scores. The total execution time would then be over 14 days.

17.

Our solution to this problem was to make our own greedy search algorithm. Instead of always finding a path we instead look at all paths of a fixed length and check if the end node is in the path. We were also able to make a version that can do multiple start and end nodes at the same time which greatly improved the performance. If we assume there are 350 edges from a node then the number of paths on a 1 step search is 350, the 2 step search is 60 000 , the 3 step search is 9 million and the 4 step search is over 1 billion. If we consider 2700 relation scores then we cannot do a 4 step search because that would be over 3 trillion comparisons. Instead we do a 2 or 3 step search. The 2 step search is done in 0.048 seconds and the 3 step search is done in 1.63 seconds.

Our reason for why this score is also valid is that very distant relationships does not say a lot about the actors. What we now get is that only close relationships can affect the score.

18.

This is a sample of what the graph could look like, with a significant lower number of nodes and edges. The green circles are the Interstellar actors and as you can see they are connected together by the same value.

19.

Skip

20.

The MRJob / Spark version of the preprocessing is much slower but we didn't spend a lot of time optimizing it because we wanted to focus on the algorithm. The largest difference is seen in the title_ratings script where the Python implementation finishes in 2 seconds, while the MRJob version spends over a minute. This is an example of the overhead associated with running jobs on Hadoop.

21.

In the execution time of the algorithm we can see the improvements we made in the relation score calculation on Spark. The Python version is estimated to finish in 14 days, while the Spark version is done in under 20 minutes. The similarity score is much slower as expected because all of the work is done on the master node.

******

Malavika:

22.

As we've seen the performance of a system with very large data and complex computations are exponentially improved by the use of big data tools like Spark and Hadoop. And thus is highly recommended by various other researchers. ***NEXT***

23.

Here are a few related works. ***NEXT***

THANK YOU.

If you have any questions, let us know.