Heterogeneous Knowledge in Recommendation Systems

Project report submitted for

VIth Semester Mini Project

in

Department of computer science and engineering

By,

Aditi Agrahari (17100002)

Jai Kumar Dewani (17100018)

Utkarsh Raj Singh (17100055)

Guided by,

Dr. Muneendra Ojha

Asst. Professor, Department of CSE, Dr. SPM IIIT Naya Raipur

Department of Computer Science and Engineering

Dr. Shyama Prasad Mukherjee

International Institute of Information Technology, Naya Raipur

(A Joint Initiative of Govt. of Chhattisgarh and NTPC)

Email: iiitnr@iiitnr.ac.in, Tel: (0771) 2474040, Web: www.iiitnr.ac.in

CERTIFICATE

This is to certify that the project titled “Heterogeneous Knowledge in Recommendation Systems” by “Aditi Agrahari, Jai Kumar Dewani and Utkarsh Raj Singh” has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree/diploma.

(Signature of Guide)

Dr. Muneendra Ojha
Assistant Professor
Department of Computer Science and Engineering
Dr. SPM IIIT-NR
July, 2020

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

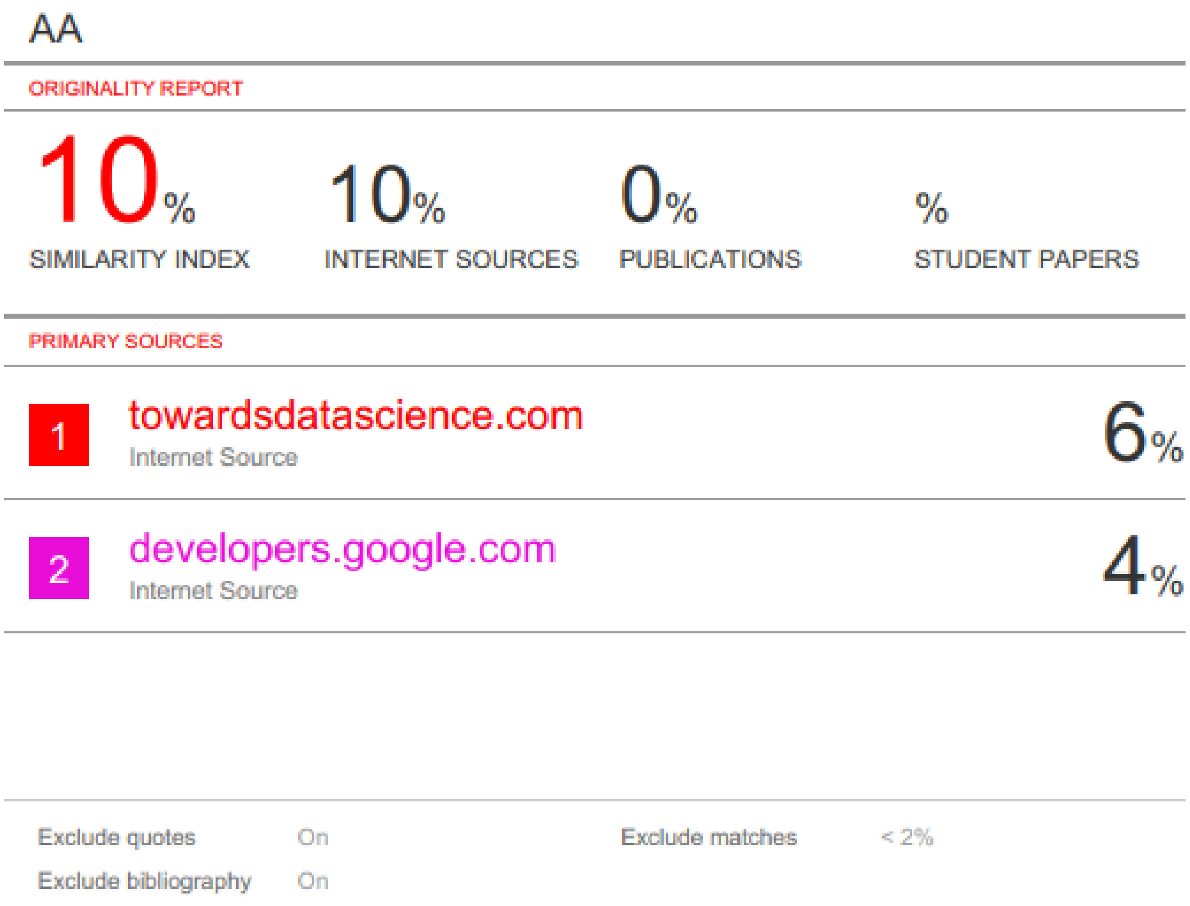
Aditi Agrahari
(17100002)

Jai Kumar Dewani
(17100018)

Utkarsh Raj Singh
(17100055)

Date : 08.07.2020

PLAGIARISM REPORT



AA

ORIGINALITY REPORT

10%	10%	0%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	towardsdatascience.com Internet Source	6%
2	developers.google.com Internet Source	4%

Exclude quotes	On	Exclude matches	< 2%
Exclude bibliography	On		

APPROVAL SHEET

This project report entitled "Heterogeneous Knowledge in Recommendation Systems" by "Aditi Agrahari, Jai Kumar Dewani and Utkarsh Raj Singh" is approved for VIth Semester Minor Project.

(Signature of Mentor)

Dr. Muneendra Ojha

Date: 08.07.2020 Place: Naya Raipur

ABSTRACT

Building recommendation engines has been an area of active research over the past decade. The invention of new techniques and algorithms have given us engines with amazing accuracy. These models have proven themselves to be of immense business value, for they help all the stakeholders reach their customers and clients according to their personal taste. This drives sales, and generates an economy. Hence, large organisations like Google, Facebook, Amazon, and many more are actively investing in newer methods and techniques to deliver a more customised user experience to the clients.

Knowledge Graphs have emerged as a very useful information store for a plethora of domains. The inferences a machine can generate using facts available in a data store are a rich source of common sense knowledge. They have the unique ability to act as a common store for information from a wide variety of sources, and help the observer get the cross domain knowledge from a single source.

In this work, all the major techniques and approaches for building recommendation engines are qualitatively and quantitatively analysed. Several topics such as Content based filtering, matrix factorisation, and deep learning models are examined and experimented with. We then go on to study the current system using both of the empirical method of matrix factorisation and the modern deep learning method, to give both width and depth in the content sea.

The Deep Learning Model is very similar to the Model based matrix factorization. In matrix factorization, we decay our original sparse matrix into a result of 2 low position symmetrical lattices. For deep learning execution, we needn't bother with them to be symmetrical, we need our model to gain proficiency with the estimations of the installing framework itself. The client inactive highlights and film idle highlights are gazed upward from the implanting frameworks for explicit movie user blends.

We also discuss a baseline algorithm for generating knowledge graphs from free text available on the internet. We believe that if done properly we can leverage the immense volume of free text available on the World Wide Web and extract knowledge from them to enhance our engine.

The aim of this work is also to establish a ground for anyone to build upon recommendation architecture from heterogeneous sources. We are building Knowledge Graphs from the sources which are different from the sources from which the data for training the recommendation model is taken off. Instead, we are hypothesizing that it can actually build better results if we could incorporate the Knowledge from different sources and convert them into embeddings for the deep neural network to benefit. And in this way we have better recommendations.

ACKNOWLEDGEMENTS

We would like to take this opportunity to extend our thankfulness to the driving people behind the ideas and motivation backing the work. Dr. Muneendra Ojha, our mentor, who has been very helpful and accommodating of our ideas. He has been a source of constant motivation, driving us to give our best which has now culminated into this work.

Ashish Singh Patel, who has been a source of new ideas time and again. We have gained several insights related to the technologies and architectures in our discussions with him.

The Google Developers Club, for publishing excellent resources for anyone to get well versed and running in the field of Recommendation System and Deep Learning. They have covered the existing literature and all the major techniques discussed in this work in a very extensive manner, so as to make a pool of resources from several outstanding people.

TABLE OF CONTENTS

LIST OF FIGURES

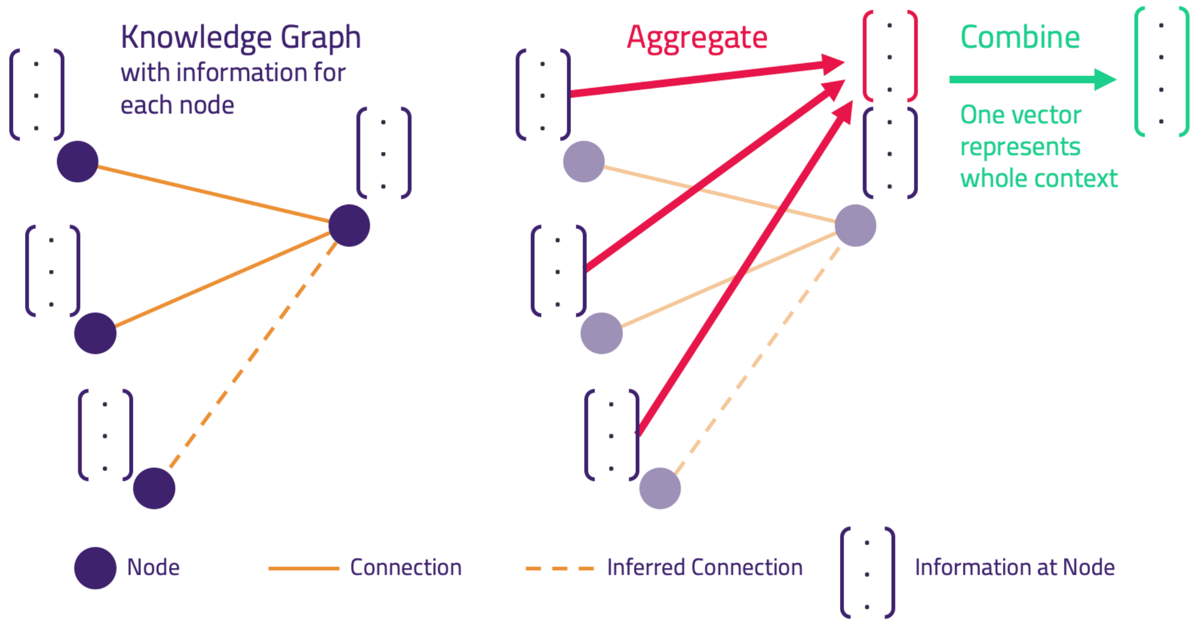
1. INTRODUCTION

Recommendation Systems is an area of both research and business importance. Till as of late, individuals for the most part would in general purchase items prescribed to them by their companions or the individuals they trust. This used to be the essential strategy for buy when there was any uncertainty about the item. In any case, with the coming of the computerized age, that circle has extended to incorporate online locales that use a type of proposal motor. Organizations like Amazon, Netflix, and Linkedin influence recommender frameworks to assist clients with finding new and applicable things (items, recordings, occupations), making a magnificent client experience while driving steady income.

A recommendation engine channels the information utilizing various calculations and prescribes the most pertinent things to clients. It first catches the previous conduct of a client and dependent on that, suggests items which the clients may probably purchase.

This work aims to explore the different methods and techniques given in the literature, such as **content based filtering, collaborative filtering, nearest neighbour generation using KNN, and latent feature extraction methods using Matrix Factorization and Deep Neural Networks.**

Graphs have a rich history, beginning with Leohnard Euler in the eighteenth century to an entire scope of diagrams today. Inside the field of software engineering there are numerous utilizations of graphs: diagram databases, information charts, semantic charts, calculation diagrams, interpersonal organizations, transport charts and some more.
Graphs have assumed a key job in the ascent of Google (their first advancement was utilizing PageRank to control look, today their Knowledge Graph has developed in significance) and Facebook. From governmental issues to minimal effort universal air travel, graph calculations have majorly affected our reality.
In this work, the incorporation of **Knowledge graphs in the recommendation process** will be studied.



1.1 Graphical representation of the KG Vector (source: blog.grakn.ai)

2. LITERATURE REVIEW

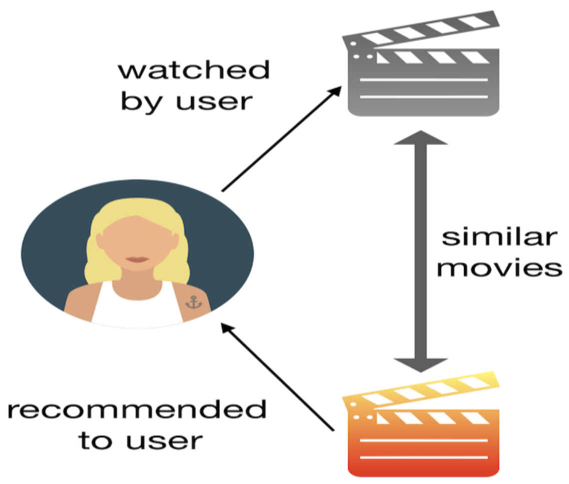
Recommendation systems' literature mentions two types, content-based filtering and collaborative filtering.

2.1. Content-based filtering[1]

Content-based sifting, additionally alluded to as intellectual separating, suggests things dependent on a correlation between the substance of the things and a client profile. It uses the similarity between items to recommend items, like items similar to what the user likes, dislikes and based on the user's previous actions.

A content-based recommender works with information that the client gives, either unequivocally (evaluating) or certainly (tapping on a connection, giving an underlying rundown of interests). In view of that information, a client profile is produced, which is then used to make proposals to the client. As the client gives more data sources or takes activities on those proposals, the motor turns out to be increasingly precise. The recommender focuses on additional information of items, such as the length of a movie, the cast of the movie along with additional user information such as age, sex, to build a user profile.

The content-based filtering method employs the exploitation approach for making recommendations wherein the model chooses items similar to those for what the user has already expressed a preference, i.e., the model exploits the known information to proceed.



2.1 Content based filtering

TF-IDF (Term Frequency — Inverse Document Frequency) and cosine similarity between space vector model techniques are commonly used in content-based filtering.

Advantages:

- The model predictions are not dependent upon user data (the past ratings) and hence overcomes user cold start by referring to the user profile or user's chosen interests, thereby
- The model can catch the particular interests of a client and can suggest specialty things that not very many different clients are keen on.

Disadvantages:

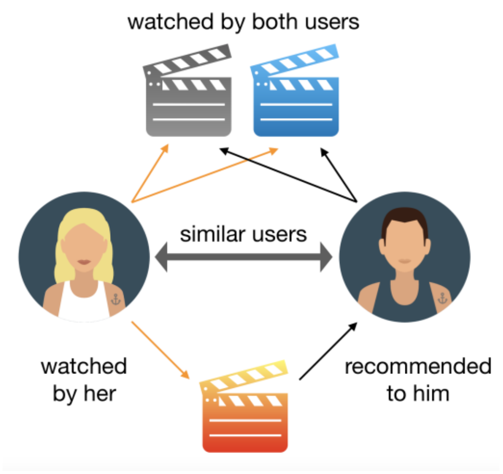
- Since the highlights of the things are hand-designed somewhat, this method requires a great deal of space information. Accordingly, the model requires and relies exceptionally upon hand-built highlights made by hand.
- Frameworks depending solely on content-based separating just prescribe things firmly identified with those the client has recently evaluated. Such frameworks never uncover novel things that clients may appreciate outside their typical arrangement of decisions.

2.2. Collaborative filtering

To work on some of the limitations of content-based filtering, similarities between queries/users and items simultaneously is used in collaborative filtering to provide recommendations. This allows the system to recommend an item to user A based on the interests of a similar user B. It is generally more accurate then content filtering however, it suffers from cold start problems. (If a new user exists and does not have any intra-dependencies among others, we can't recommend anything).

Collaborative methods for recommender systems are based solely on the past interactions recorded between users and items in order to produce new recommendations. These methods are explorative, i.e., they not only make recommendations based on the user's history but also consider similar users' history and recommend items that may be different from the user's past preferences.

This also leads to the embedding (latent factors) that can be learned automatically and not relying and being limited on domain knowledge. Embedding allows us to convert large discrete spatial features to a lower dimension continuous feature, i.e., "A set of movies" to features like "Artistic/blockbuster" and "Adult/children". The only difference is that in real practice the embeddings themselves alone don't have any meaning like the one in this example but it helps in encapsulating the patterns in lower dimensions.



2.2 Collaborative filtering

Advantages:

- No area information is important in light of the fact that the embeddings are naturally learned.
- Serendipity - The model can assist clients with finding new interests. In disengagement, the framework may not have the foggiest idea about whether the client is keen on a given thing, yet the model may even now suggest it in light of the fact that comparable clients are keen on that thing.

Disadvantages:

- Cold Start Problem - The expectation of the model for the guaranteed (client, thing) pair is the spot result of the relating embeddings. Along these lines, if a thing isn't seen during preparation, the framework can't make an insert for it and can't question the model with this thing.

- Difficult to incorporate side highlights for the query item - Side highlights are any highlights past the question or thing ID. For film suggestions, the side highlights may incorporate nation or age. Counting accessible side highlights improves the nature of the model.

In general, there are two methods to achieve collaborative filtering.

2.2.1. Matrix Factorization based method[2]

The primary presumption behind matrix factorisation is that there exists an entirely low dimensional inactive space of highlights in which we can speak to the two clients and things and with the end goal that the communication between a client and a thing can be gotten by figuring the dab result of comparing thick vectors in that space.

At the point when a client offers criticism to a specific film they saw, this assortment of input can be spoken to in a type of a network, where each line speaks to every client, while every segment speaks to various motion pictures. Clearly the grid will be meager since there are a great many films and not every person is going to watch or rate each film.

One quality of matrix factorization is the way that it can consolidate certain input data that isn't legitimately given however can be determined by examining client conduct. Utilizing this quality we can assess if a client is going out to see like a film that he/she never observed.

2D Embedding[3]
The following is an example of a two feature embedding.
For feature 1, a negative value implies that the movie is an arthouse movie whereas a positive value implies that it is a blockbuster movie.
For feature 2, a negative value implies that the movie is for children whereas a positive value implies that it is an adult movie.
The embedding represents these features in a consolidated form which is not interpretable, i.e., we can never know for certain which features are taken into consideration.
Our users are again placed in the same embedding space to best explain the feedback matrix: we want the dot product of the item embedding and user embedding for each (user, item) pair, to be close to 1 when the user watched the movie, and to 0 otherwise.

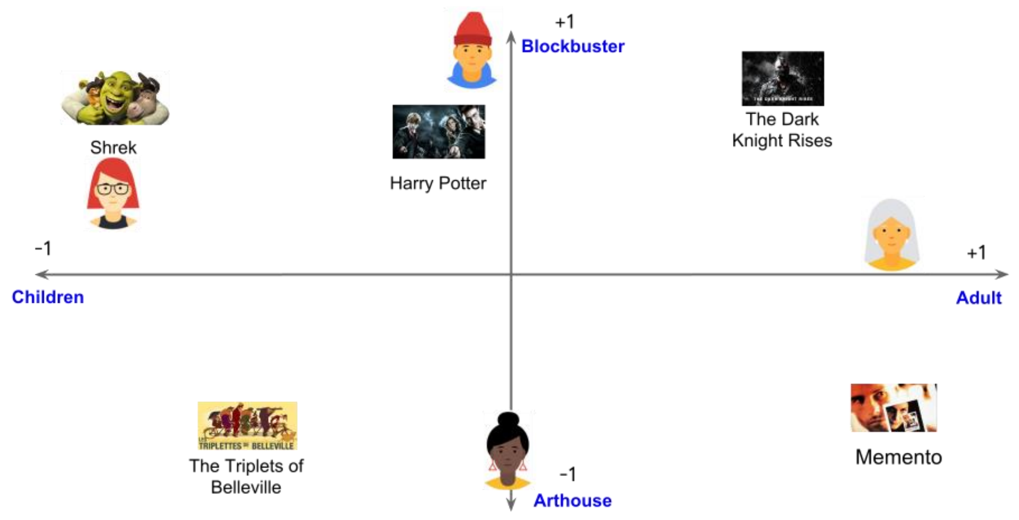
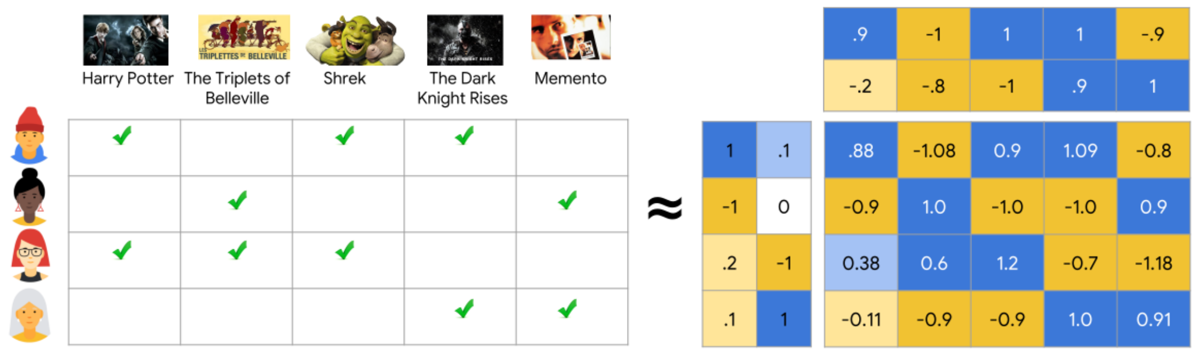


Figure 2.3 Visualization of Embedding space (source Google developers)

Matrix factorization is an embedding model. Given the feedback matrix A ∈ R^{mxd}, where *m* is the number of users (or queries) and *n* is the number of items, the model learns:

- A user embedding matrix U ∈ R^{mxd}, where row i is the embedding for user i.
- An item embedding matrix V ∈ R^{nxd}, where row j is the embedding for item j.



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

≈

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

1	.1	.88	-1.08	0.9	1.09	-0.8
-1	0	-0.9	1.0	-1.0	-1.0	0.9
.2	-1	0.38	0.6	1.2	-0.7	-1.18
.1	1	-0.11	-0.9	-0.9	1.0	0.91

Figure 2.4 Matrix factorization (source Google developers)

The embeddings are learned such that the product $\Box\Box^{\Box}$ is a good approximation of the feedback matrix A.

Matrix factorization ordinarily gives a more conservative portrayal than learning the full network. The full network has O(mn) passages, while the inserting lattices U, V have O((m+n)d) sections, where the installing measurement d is regularly a lot less than m and n. Therefore, lattice factorization finds shrouded structure in the information, expecting that perceptions lie near a low-dimensional subspace.

Objective Function

Weighted Matrix Factorization decomposes the objective into the following two sums:

$$\Box\Box\Box \sum_{(\Box,\Box)\in\Box\Box\Box}^{\Box} (\Box_{\Box\Box} - <\Box_{\Box}, \Box_{\Box}>)^{2} + \Box_{0} \sum_{(\Box,\Box)\notin\Box\Box\Box} (<\Box_{\Box}, \Box_{\Box}>)^{2}$$

Here, □ $_0$ is a hyperparameter that weights the two terms so that the objective is not dominated by one or the other.

Drawbacks of the Matrix factorization based method[4]:

- Can't deal with new things - The expectation of the model for guaranteed (client, thing) pair is the spot result of the comparing embeddings. Along these lines, if a thing isn't seen during preparing, the framework can't make an inserting for it and can't inquiry the model with this thing.
- Hard to include side features for query/item - The side features might include country or age.
- Folding Problem - Unrelated items might be close to relevant items in the embedding space

These lead us to the development of DNN based models.

2.2.2. Deep Neural Network Based

Deep neural network (DNN) models can address the constraints of matrix factorization. DNNs can without much of a stretch consolidate inquiry highlights and thing highlights (because of the adaptability of the info layer of the system), which can help catch the particular interests of a client and improve the significance of proposals.

Deep learning can demonstrate the non-straight connections in the information with non-direct actuations, for example, ReLU, Sigmoid, Tanh, and so on. This property makes it conceivable to catch the intricate, complicated client entity collaboration designs. Traditional strategies, for example, network factorization and factorization machines are basically straight models. This straight presumption, going about as the premise of numerous customary recommenders, is distorted and will incredibly confine their displaying expressiveness. It is entrenched that neural systems can rough any ceaseless capacity with discretionary exactness by shifting the actuation decisions and mixes. This property makes it conceivable to manage complex collaboration designs and exactly mirror the client's inclination.

One possible DNN model is softmax[5]:

- The input is the given query by the user.
- The output is a probability vector with size equal to the number of items in the data, representing the probability to correlate with each item.

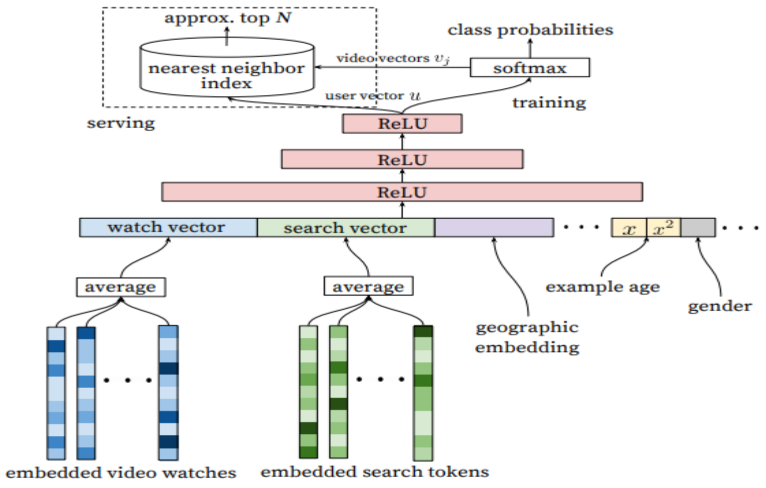
Let us take the example of a YouTube recommendation overview:
The candidate generation network takes occasions from the client's YouTube action history as info and recovers a little subset (several) recordings from a huge corpus. These applicants are proposed to be commonly pertinent to the client with high accuracy. The candidate generation network just gives expansive personalization by means of community oriented separating.

The comparability between clients is communicated regarding coarse highlights, for example, IDs of video watches, search question tokens and socioeconomics. Introducing a couple "best" suggestions in a rundown requires a fine-level portrayal to recognize relative significance among applicants with high review. The positioning system achieves this undertaking by allocating a score to every video as per an ideal target work utilizing a rich arrangement of highlights depicting the video and client. The most noteworthy scoring videos are introduced to the client, positioned by their score.

Model Architecture:
By including concealed layers and non-linear activation functions (for instance, ReLU), the model can catch increasingly complex connections in the information. Nonetheless, expanding the quantity of boundaries likewise normally makes the model harder to prepare and increasingly costly. We can incorporate the input vector, and embedding layer which is further connected with the side features to the hidden layer. The output of the hidden layer is connected to the softmax layer which gives us the probabilistic output vector.
Loss Function:
We must compare the following for calculating the loss:

- Output of the softmax layer (a probability distribution)
- Ground truth, representing the items the user has interacted with.

We can use the cross-entropy loss since comparing two probability distributions, given in the matrix factorization method.



2.5 The model architecture (YouTube recommendation engine)

2.3. Hybrid Recommendation System[7]

The hybrid recommendation system is a blend of collaborative and content-based filtering techniques. In this approach, content is used to infer ratings in case of the sparsity of ratings. This combination is used in most recommendation systems at present.
Past work to join the positive parts of the two strategies has depended on one of two techniques. One strategy produces two proposal sets—one from every method—and afterward joins the outcomes. The subsequent technique, coordinated effort by means of substance, extends every client's thing evaluations into appraisals for the thing's substance components and afterward matches to different clients through a synergistic separating calculation. Like community filtering, collaboration via content can just produce proposals for things that different clients have just evaluated. Its substance based strategies create a lot of halfway scores — for instance, a score for every entertainer, chief, and film sort. It at that point utilizes these halfway scores, as opposed to film evaluations, in collective separating to discover clients with comparative scores.

3. RETRIEVAL, SCORING AND RE-RANKING[6]

At serve time, we could do one of the following:

- For a matrix factorization model, the query (or user) embedding is known statically, and the system can simply look it up from the user embedding matrix.
- For a DNN model, the system computes the query embedding at serve time by running the network on the feature vector.

Once we have the query embedding, we search for item embeddings that are close in the embedding space, the nearest neighbor problem.

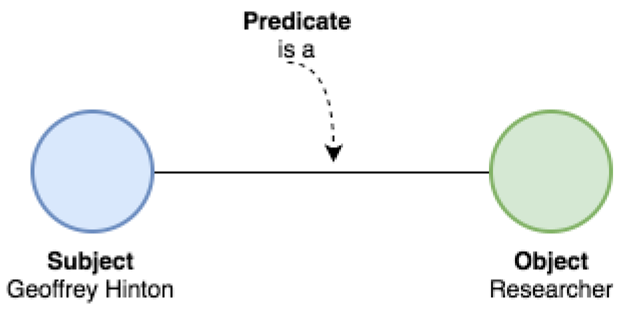
Generally, a separate model is used for scoring the candidates generated in the previous step. This is particularly useful in large datasets, such as youtube, so that more sophisticated and complex mechanisms (incorporating additional features) can be utilized for giving the scores. Finally, models are used for re-ranking, the even smaller candidate set, and takes into account various biases such as the positional bias while rendering the results to the user.

4. KNOWLEDGE GRAPH

The knowledge graph consists of integrated collections of data and information that also contains huge numbers of links between different data.
The key here is that instead of looking for possible answers, under this new model we're seeking an answer. We want the facts—where those facts come from is less important. The data here can represent concepts, objects, things, people and actually whatever you have in mind. The graph fills in the relationships, the connections between the concepts. In a knowledge-graph information represented in a particular formal ontology can be more easily accessible to automated information processing, and how best to do this is an active area of research in computer science like data science.

All data modeling statements (along with everything else) in ontological languages and the world of knowledge-graphs for data are incremental, by their very nature. Enhancing or modifying a data model after the fact can be easily accomplished by modifying the concept. With a knowledge-graph what we are building is a human-readable representation of data that uniquely identifies and connects data with common business terms. This "layer" helps end users access data autonomously, securely and confidently.

For building the knowledge-graph you need linked data. The goal of linked data is to publish structured data in such a way that it can be easily consumed and combined with other linked data, and ontologies as the way we can connect entities and understand their relationships.



4.1 The basic structure of a KG connect (KDNuggets)

To have a triple we need a subject and object, and a predicate linking the two. So as you can see we have the subject <Geoffrey Hinton> related to the object <Researcher> by the predicate <is a>. This may sound easy for us humans, but it needs a very comprehensive framework to do this with machines.

This is the way the knowledge-graph gets formed and how we link data using ontologies and semantics.

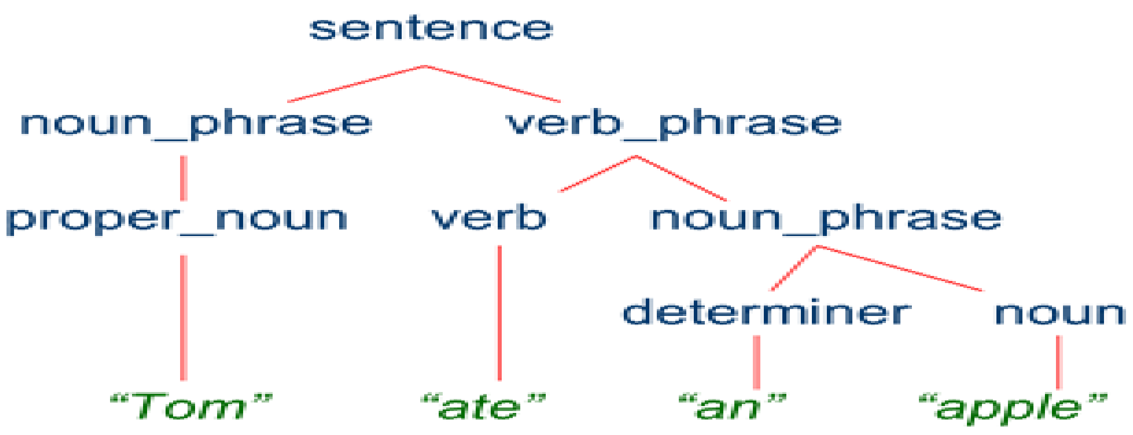
We devise an algorithm to build the KG from heterogeneous source(text reviews):

● Setup a statistical parser for obtaining dependency tree with POS tagged words(*Sample Sentence with its Tree)*

- The Stanford statistical parser breaks each sentence into smaller sentences and does this recursively until each of these small sentences are only one word long, It also creates a Tree connecting each part of smaller sentences while annotating each Level except the last.

- We are using the Stanford statistical parser to convert each line from the dataset to a parsed Tree. This helps in identifying different parts of a sentence and further working with them.

- Let us consider a sample sentence "*Tom ate an apple*". parsed tree of this sample sentence is depicted in the image:



4.2 The Parsed Tree has a ROOT which links to level-1 parts of sentences and these sentences in them can have more levels. The Notations in the tree like NP, VBD, NN, PP are a part of Tree Syntax of Natural Language

● Graph traversal of dependency tree to obtain Subject predicate and object*(SPO)*

- After obtaining the tree, the main motive is to obtain Subject, Predicate and object parts of each sentence. This can be obtained by doing a Depth-first-search on the tree because we know what the parts of sentences will be situated in which sub-tree of the parsed tree. Starting with the Root, moving to the noun_phrase*(NP)* to find the subject and searching in verb_phraset*(VP)* to get the predicate and the object.

● Synstasing SPO data for training of probabilistic engine

- After Obtained Subject, Object and their predicate of each sentence, this data is saved in another file for further reference. This saves us ample time of not computing the same parse tree again and again while computing the probability of a predicate between an object and subject.

● Identifying all predicates linked to each subject and object

- After reading the newly synthesised SPO dataset created in the last step, the next step is to find the most referred objects. To obtain this we are counting the frequency of occurrence of each object in the SPO dataset and picking the 20 classes having the most frequency for both.
- While working with each class lying in most occured object list, the next step is to find all the predicates that are linked to this object while counting their frequencies. After doing the same with all the predicates that are observed in conjunction with the objects along with their frequencies we have a well structure data that looks something like this
-

```
[
        Object1[
[Predicate1_1,frequency1_1],
[Predicate1_2,frequency1_2],
                    ..
..
..
[Predicate1_n,frequency1_n]
], Object2[
[Predicate2_1,frequency2_1],
[Predicate2_2,frequency2_2],
..
..
..
[Predicate2_m,frequency2_m]
]
..
..
..
..
]
```

- Same structure is obtained for the subject by performing the same procedure.

● Generating most probable relations between the classes of Subject-Object
- Before finding the most probable relation we need to find the relations which are common in between a given Subject and Object. To do so we the set of common relations as the chances of finding the correct relation is very high in this

□□ = □□□ □□ □□□□□□□□□□ □□□□□□ □□ □□□□□□□

□□ = □□□ □□ □□□□□□□□□□ □□□□□□ □□ □□□□□□

- If we find some intersection of relations, □□ ∪ □□ then find the most occuring one and calculate its probability using this formula
-

□□□□□□□□□□□ = □□□□□□□□□ □□ □□□□□□□□ / □□□ □□ □□□□□□□□□ □□ □□□ □□□□□□□□□ □□ □□□

- If we don’t find any common relation then we calculate the most probable relation inside the union of relations, □□ ∪ □□ using the same probability formula but with the union as a set.

When we obtain the graph we can then extract the input features to make the recommendation engine better, which is the work for the project.

5. EXPERIMENTS

5.1 The Simple Recommender

The Simple Recommender offers summed up recommendations to each client dependent on film notoriety and (in some cases) classification. The essential thought behind this recommender is that films that are increasingly well known and all the more widely praised will have a higher likelihood of being enjoyed by the normal crowd. This model doesn't give customized proposals dependent on the client.

The execution of this model is incredibly paltry. We should simply sort our films dependent on evaluations and prevalence and show the top motion pictures of our rundown. As an additional progression, we can go in a kind contention to get the top films of a specific classification.

We utilize the TMDB Ratings to come up with Top Movies. IMDB's weighted rating formula is used to develop the model logic. Mathematically, it is represented as follows:

Weighted Rating (WR) = (vv+m.R)+(mv+m.C)

V - the number of votes for the movie
M - minimum votes required to be listed in the chart
R - average rating of the movie
C - mean vote across the whole report

The following stage is to decide a proper incentive for the base votes required to be recorded in the graph. We will utilize 95th percentile as our cutoff. At the end of the day, for a film to include in the outlines, it must have a bigger number of votes than in any event 95% of the motion pictures in the rundown.

5.2 Content Based Recommender

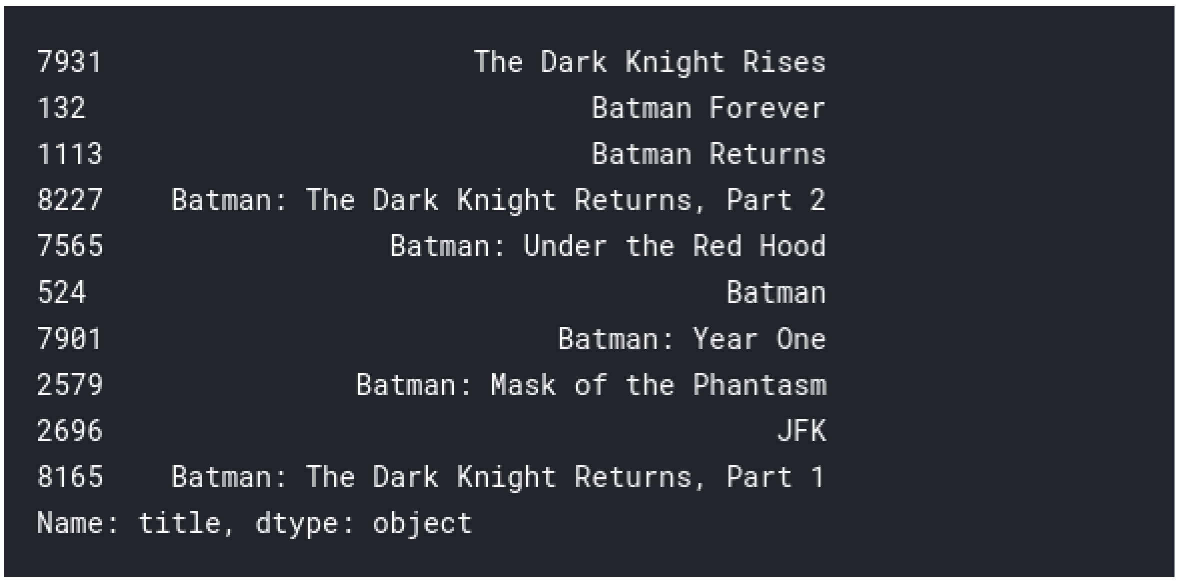
The simple recommender we built in the previous section suffers from some limitations. It does not take into account the actual likes of the individual, and just gives the movies which are considered good by the general public. For example, if the top 20 movies are all thriller and the user likes romance, he or she will probably not like all the movies recommended by the model. To overcome this drawback, we need to devise an algorithm to take into account the personal likes of the user.

To customize our recommendations more, we will fabricate a technique that figures closeness between movies dependent on specific measurements and recommends movies that are generally like a specific film that a user loved. Since we will utilize film metadata (or substance) for this, we call this Content Based Filtering. Our model will be based on:

- Movie Overviews and Taglines
- Movie Cast, Crew, Keywords and Genre

5.2.1 Movie Description Based Recommender

With reference to the code given in the accompanying Colab Notebook, we have the following results for searching "The Dark Knight" movie:



```
7931                        The Dark Knight Rises
132                                Batman Forever
1113                               Batman Returns
8227    Batman: The Dark Knight Returns, Part 2
7565                   Batman: Under the Red Hood
524                                        Batman
7901                            Batman: Year One
2579                 Batman: Mask of the Phantasm
2696                                          JFK
8165    Batman: The Dark Knight Returns, Part 1
Name: title, dtype: object
```

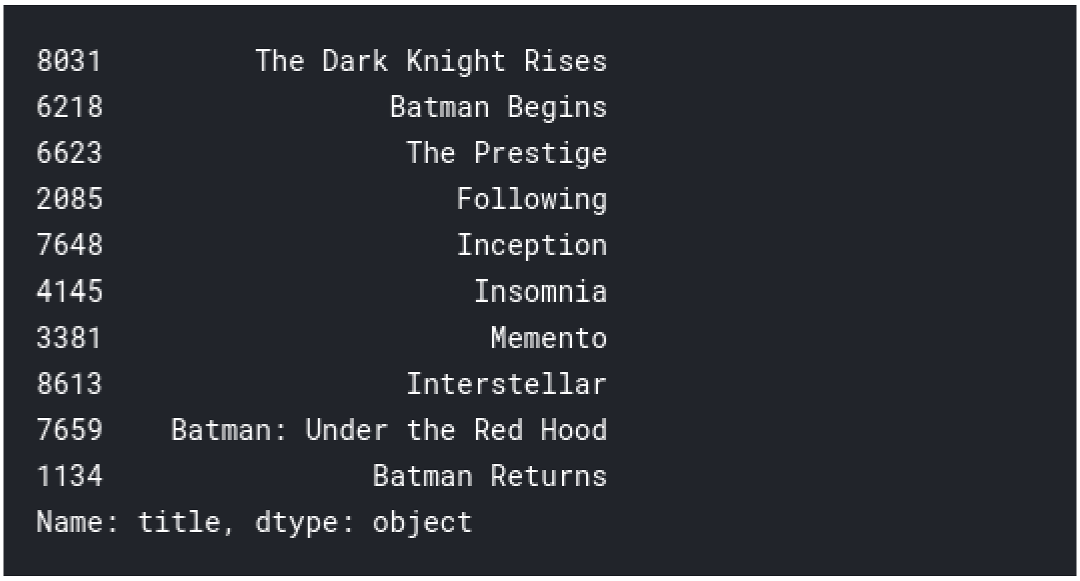
5.1 Results obtained by the model for "The Dark Knight" Movie by Christopher Nolan (Type 1)

We see that for The Dark Knight, our framework can distinguish it as a Batman film and along these lines suggest other Batman films as its top proposals. Be that as it may, shockingly, that is this framework can do right now. This isn't very useful to a great many people as it doesn't take into consideration significant highlights, for example, cast, group, director and classification, which decide the rating and the ubiquity of a film. Somebody who enjoyed The Dark Knight presumably loves it more in view of Nolan and would abhor Batman Forever and each other unacceptable film in the Batman Franchise.

Accordingly, we are going to utilize considerably more intriguing metadata than Overview and Tagline. In the following subsection, we will construct an increasingly advanced recommender that takes class, watchwords, cast and team into thought.

5.2.2 Movie Metadata Based Recommender

With reference to the code given in the accompanying Colab Notebook, we have the following results for searching "The Dark Knight" movie:



```
8031         The Dark Knight Rises
6218                 Batman Begins
6623                  The Prestige
2085                     Following
7648                     Inception
4145                      Insomnia
3381                       Memento
8613                  Interstellar
7659    Batman: Under the Red Hood
1134                Batman Returns
Name: title, dtype: object
```

5.2 Results obtained by the model for "The Dark Knight" Movie by Christopher Nolan (Type 2)

We get much more satisfying results this time. The recommendations seem to have recognized other Christopher Nolan movies (due to the high weightage given to the director) and put them as top recommendations. There is a correlation with the likes of The Dark Knight as well as some of the other ones in the list including Batman Begins, The Prestige and The Dark Knight Rises.

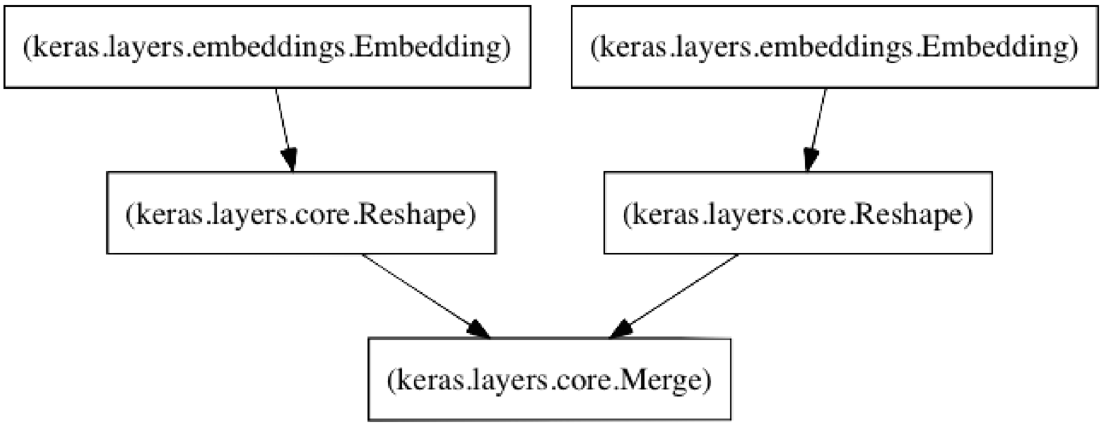
5.3 Deep Learning Based Model

Using deep learning is like that of Model-Based Matrix Factorization. In matrix factorization, we disintegrate our unique sparse matrix into a result of 2 low position symmetrical lattices. For deep learning usage, we needn't bother with them to be symmetrical, we need our model to gain proficiency with the estimations of the implanting network itself. The client inactive highlights and film inert highlights are gazed upward from the installing networks for explicit film client mixes. These are the info esteems for additional straight and non-direct layers. We can pass this contribution to numerous relu, direct or sigmoid layers and gain proficiency with the relating loads by any improvement calculation (Adam, SGD, and so forth.).

The architecture of the model we built for experimentation is given as:

- A left embedding layer that creates a Users by Latent Factors matrix.
- A right embedding layer that creates a Movies by Latent Factors matrix.
- When the input to these layers are (i) a user id and (ii) a movie id, they'll return the latent factor vectors for the user and the movie, respectively.
- A merge layer that takes the dot product of these two latent vectors to return the predicted rating.

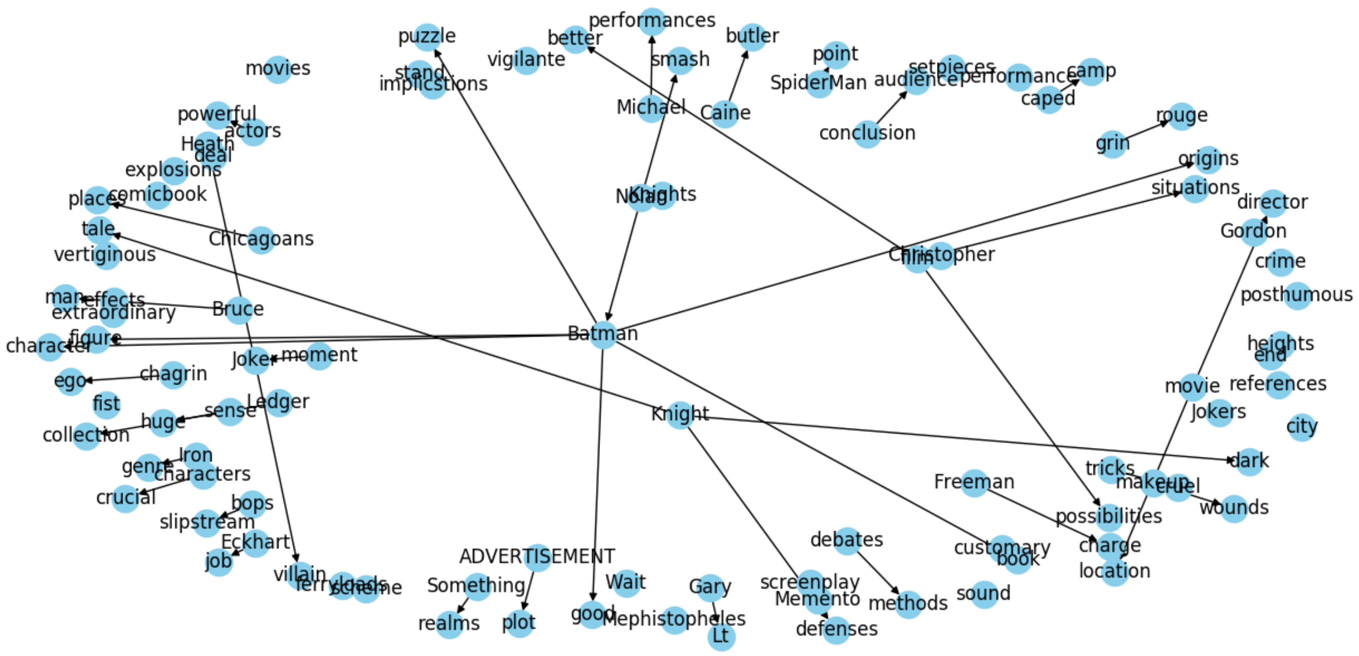
We then compile using MSE as the loss function and AdaMax as the learning algorithm. Now we train the model. For our dataset with nearly 1,000,000 ratings, almost 5,000 users and 3,500 movies, we have trained for almost 20 epochs and made the validation split as 90% - 10%.
The best validation loss is 0.7. Taking the square root of that number, we got the RMSE value of 0.86


(keras.layers.embeddings.Embedding)
(keras.layers.embeddings.Embedding)
(keras.layers.core.Reshape)
(keras.layers.core.Reshape)
(keras.layers.core.Merge)

5.3 The architecture of the Deep Learning Model

5.4 Knowledge Graph Generation

For the sample movie The Dark Knight, we have developed the scraper and got reviews from the IMDB source. As described in the accompanying Google Colab Notebook, we use the Stanford Parser and our Triple Extraction Algorithm for building the Knowledge Graph based on the relevant concepts from the film.



5.4 Knowledge Graph from the reviews for the movie The Dark Knight

6. CONCLUSION

In the present work, we have built a baseline recommendation system with deep neural networks and generated a knowledge graph with the heterogeneous sources corresponding to the film. We have seen how different approaches for building recommendation systems have evolved with time, and how these approaches are utilized in a real time scenario by building some models for the same. We have explored the pros and cons of each of the major models and tested their results across the MovieLens Data from the Internet. We have seen how we can extract triples from the free text and develop a custom Knowledge graph with the central concepts. Information from these graphs can act as an extended group of data points for the movie recommender and help improve the score.

REFERENCES

[1] Van Meteren, Robin, and Maarten Van Someren. "Using content-based filtering for recommendation." Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop. Vol. 30. 2000.

[2] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." Computer 42.8 (2009): 30-37.

[3] Zhang, Fuzheng, et al. "Collaborative knowledge base embedding for recommender systems." Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016.

[4] Doris Xin, Nicolas Mayoraz, Hubert Pham, Karthik Lakshmanan, and John R. Anderson. 2017. Folding: Why Good Models Sometimes Make Spurious Recommendations. In Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys ’17). Association for Computing Machinery, New York, NY, USA, 201–209. DOI:https://doi.org/10.1145/3109859.3109911

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems (RecSys ’16). Association for Computing Machinery, New York, NY, USA, 191–198. DOI:https://doi.org/10.1145/2959100.2959190

[6] Google Developers

[7] J. Salter and N. Antonopoulos, "CinemaScreen recommender agent: combining collaborative and content-based filtering," in IEEE Intelligent Systems, vol. 21, no. 1, pp. 35-41, Jan.-Feb. 2006.