Capstone Project

Machine Learning Engineer Nanodegree

Ramya Sahithi Adari December 20th, 2018

Movie Recommendation system using embeddings and Neural Network

1. Project Definition

1.1 Project Overview:

Movie recommendation engine filters the data using different algorithms and recommends the most relevant movies to users. It first captures the past behavior of a customer and based on that, recommends movies which the users might be likely to watch.

We use MovieLens 20M dataset which are widely used in education, research, and industry. They are downloaded hundreds of thousands of times each year, reflecting their use in popular press programming books, traditional and online courses, and software. These datasets are a product of member activity in the MovieLens movie recommendation system, an active research platform that has hosted many experiments since its launch in 1997.

A movie recommendation system, based on collaborative filtering approach makes use of the information provided by users, analyzes them and then recommends the movie that is best suited to the user at that time.

1.2 Problem Statement:

Building a **collaborative filtering movie recommendation system using Word embeddings,** Matrix factorization methods by defining and implementing a Neural network with Keras.

2. Making use of embeddings in the **DNN model**, implementing **Matrix Factorization** architecture for movie recommendations and predicting ratings the user might give to a movie.
3. Visualizing embeddings with **t-SNE**.

1.3 Metrics:

The Common evaluation metrics for Recommendation systems are:

- **Mean Absolute Error(MAE):** Traditionally, mean absolute error (MAE) has been used to evaluate the performance of Collaborative Filtering algorithms. MAE works well for measuring how accurately the algorithm predicts the rating of a randomly selected item.

- **Root Mean Squared Error(MSE):** Root Mean Squared Error (RMSE) is perhaps the most popular metric used in evaluating accuracy of predicted ratings. The system generates predicted ratings R^ui for a test set T of user-item pairs (u, i) for which the actual ratings R_{ui} are known. Typically, Rui are known because they are hidden in an offline experiment, or because they were obtained through a user study or online experiment. It varies from 0.0 to 1.0 with lower values being signaling less error (therefore "better"). So, here the evaluation of recommending a movie rating score is performed using RMSE. Root Mean Squared Error is not application-specific as such, and so tends to be included in most academic literature when evaluating the predictive accuracy of Recommender Systems.

2. Data set:

- The dataset is obtained from grouplens.org: **https://grouplens.org/datasets/movielens/20m/** MovieLens itself is a research site run by GroupLens Research group at the University of Minnesota. The first automated recommender system was developed there in 1993.
- The datasets describe ratings and free-text tagging activities from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016.
- Users were selected at random for inclusion. All selected users had rated at least 20 movies.

Content: No demographic information is included. Each user is represented by an id, and no other information is provided.

In the project I have 'ratings.csv' and 'movie.csv' two data files to build a DNN movie rating prediction model and the recommendations the model gives for a user based on his/her rating history.

2.1 Data Preprocessing:

There is some preprocessing has been made on MovieLens data set. We take ratings.csv and movie.csv files

- In Ratings file, compactifying user id's and movie id's.
- The column **y** is just a copy of the rating column with the mean subtracted - this will be useful later.

	userId	movieId	rating	y
0	123425	525	4.0	0.47437
1	108287	930	5.0	1.47437
2	109381	4085	4.0	0.47437
3	64404	892	4.5	0.97437
4	137309	1683	2.5	-1.02563

- In Movies file, finding number of ratings (**n_ratings**) per movie, finding mean ratings(**mean_rating**) per movie, adding **key** attribute which holds title of the movie as it is unique with year.

	movieId	title	genres	year	key	n_ratings	mean_rating
0	0	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	1995	Toy Story	49695	3.921240
1	1	Jumanji	Adventure|Children|Fantasy	1995	Jumanji	22243	3.211977
2	2	Grumpier Old Men	Comedy|Romance	1995	Grumpier Old Men	12735	3.151040
3	3	Waiting to Exhale	Comedy|Drama|Romance	1995	Waiting to Exhale	2756	2.861393
4	4	Father of the Bride Part II	Comedy	1995	Father of the Bride Part II	12161	3.064592

We make use of preprocessed 'ratings.csv' file for prediction and 'movies.csv' for further exploration.

3. Methodology: Various methods has been invoked to predict movie ratings by the model.

3.1 Building Deep Neural Network Model with embeddings:

In our dataset, "Ratings" range from 0.5 stars to 5. Our **goal** will be to predict the rating a given user $\mathbf{u_i}$ will give a movie $\mathbf{m_j}$.

3.1.1 What are embeddings?

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

3.1.2 Why embeddings?

It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast, and a given word or document would be represented by a large vector comprised mostly of zero values.

- Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space.
- The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used.
- The position of a word in the learned vector space is referred to as its embedding.

- Embeddings are used for sparse categorical variables, In our dataset,

  userId: Different users will be having different unique id's with (~140K distinct values).

  movieId: Different movies will be having different unique id's with (~27K distinct values)

  An embedding layer would be a good idea for using these variables as inputs to a network.

- We also discuss why other methods are not good choice for this problem,
  1. If we take **numerical input** id's (user id and movie id), that doesn't make any sense in extracting ratings of the movie by the user.
  2. If we consider **one-hot encoding** for large sparse categorical inputs user id, movie id to calculate the activations of our first hidden layer, we'll need to multiply our 165k inputs through about 21 million weights - but most of those products will just be zero.

DNN Model with embedding layers:

- Using **Keras API** we build a DNN model with embedding layers of different sizes 8, 32, 64 and to select best model with best embedding size based on **'Mean absolute error'** and Loss function **"MSE".**

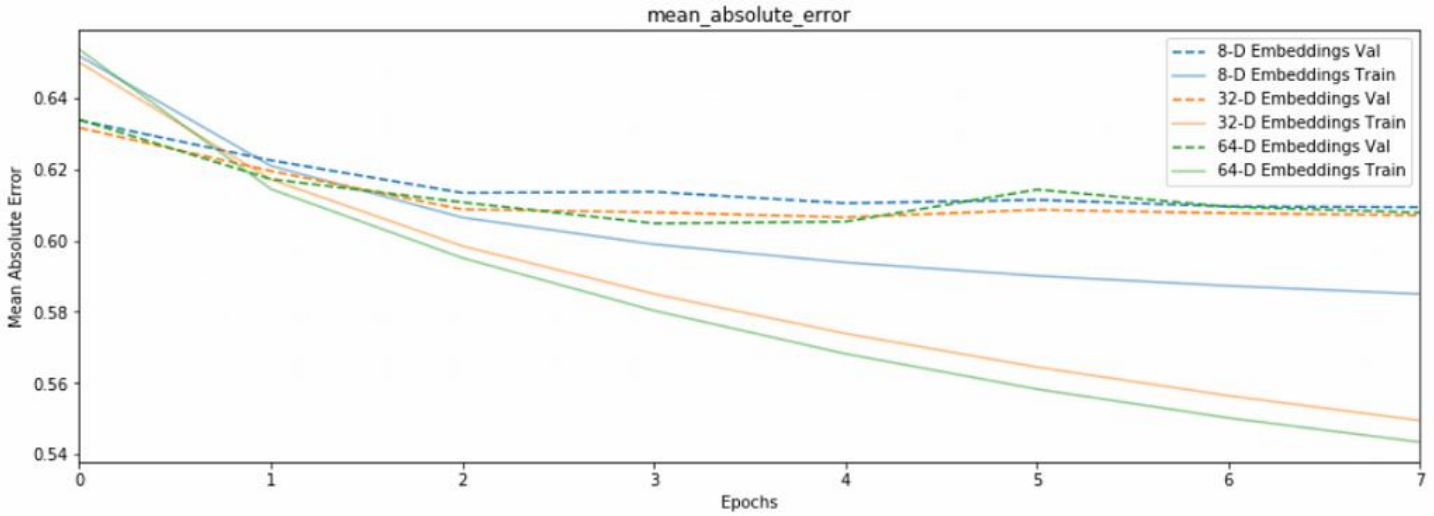


Figure: error comparison between different embedding sizes 8, 32, 64

we can see that, with 64-d embeddings,32-d embeddings our validation error goes down a little (if at all) and our training error goes down a lot which means model with 64-d embeddings is overfitting. Hence, we select 8 as best embedding size.

- DNN model with different embedding sizes for inputs userId and movieId, like larger embedding size for userId and small embedding size for movieId.

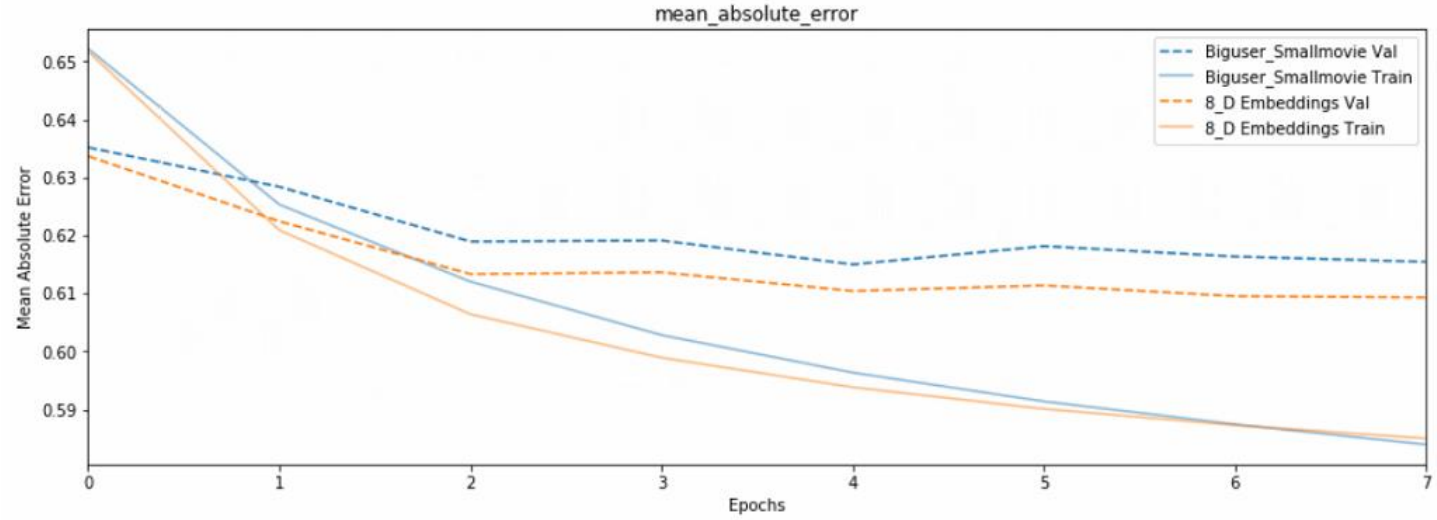


Figure: error comparison between big user and small movie embedding sizes

The varied embedding sizes of users and movies with big user and small movie configuration compared to the 8-d embeddings is not performing well and has lot of variation in its training and validation error.
From the above plots we can see that **embedding size = 8** is the best choice to use in the embedding layers of the model.

Adding Biases to the network (DNN Model): Implemented a modification to model's architecture, by considering **per-movie** biases. In Machine learning, we all know that a **bias** is just a number that gets added to a node's output value. For each movie, we'll learn a single number that we'll add to the output of what was previously our final node.

1. One basic observation is that adding biases gives our model more numbers to tune, and in this sense, it's strictly increasing its "capacity". This alone is a good enough reason to believe adding biases will at least increase our accuracy on the training set (and possibly on the validation set, depending on how much we're already overfitting).

2. Some movies are, on average, rated significantly higher or lower than others. Per-movie biases are a simple way for our model to account for the relative goodness or badness of movies. Our biases get added at the very end, our model has a lot less flexibility in how to use them. And this can be a good thing. At a high level, we're imposing a prior belief - that some movies are intrinsically better or worse than others.

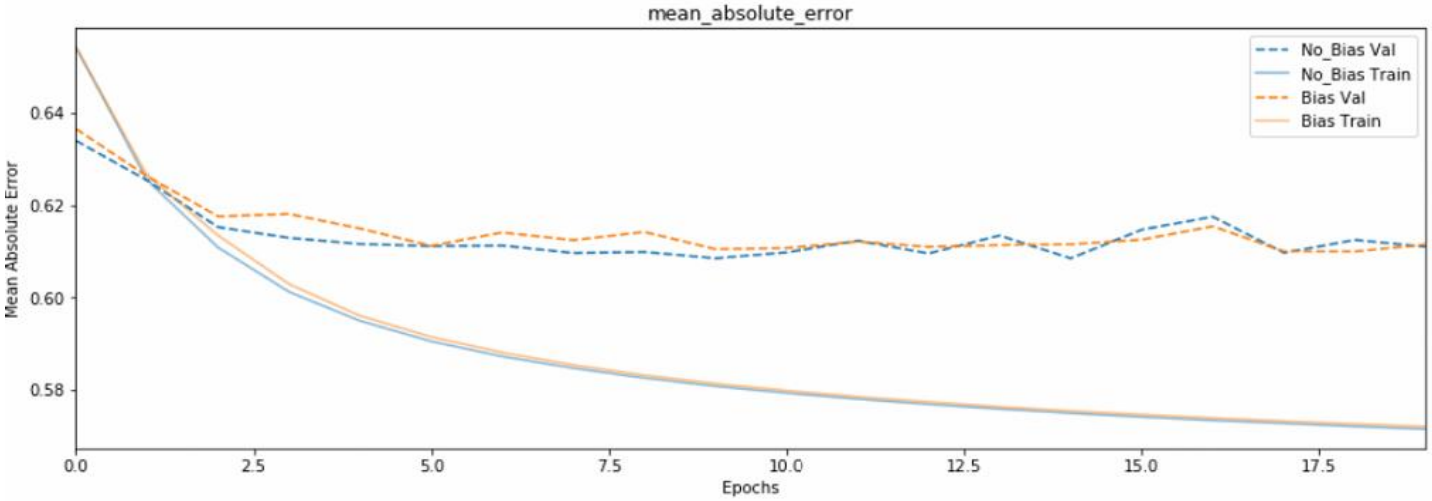


Figure: error comparison between big user and small movie embedding sizes

So, adding biases weren't the huge win we might have hoped for, but it still seems worth testing our hypothesis about how bias values will be distributed among movies.

Inspecting learned bias values:

We check whether our model with learned biases has any effect on movie ratings:

1. **Movies with largest biases:**

movieId	title	genres	year	bias	n_ratings	mean_rating
25658	Always for Pleasure	(no genres listed)	1978	1.809693	1.0	5.0
19857	Octopus, The (Le poulpe)	Comedy|Crime|Thriller	1998	1.651022	2.0	5.0
25235	No Distance Left to Run	Documentary	2010	1.636505	1.0	5.0
21822	One Small Hitch	Comedy|Romance	2013	1.543762	3.0	4.5
15679	For Neda	Documentary	2010	1.528790	5.0	4.5

2. **Movies with smallest biases:**

movieId	title	genres	year	bias	n_ratings	mean_rating
24674	Fugly!	Comedy	2013	-2.895638	1.0	0.5
26673	20 Years After	Drama|Fantasy|Sci-Fi	2008	-2.666615	1.0	0.5
22421	Sand Sharks	Comedy|Horror|Sci-Fi|Thriller	2011	-2.468503	4.0	1.0
15755	Urban Menace	Action|Horror	1999	-2.277655	2.0	0.5
15407	Magic Man	Crime|Mystery|Thriller	2009	-2.131873	3.0	2.0

3. **Plot distinguishing relationship between movie biases and its ratings:**

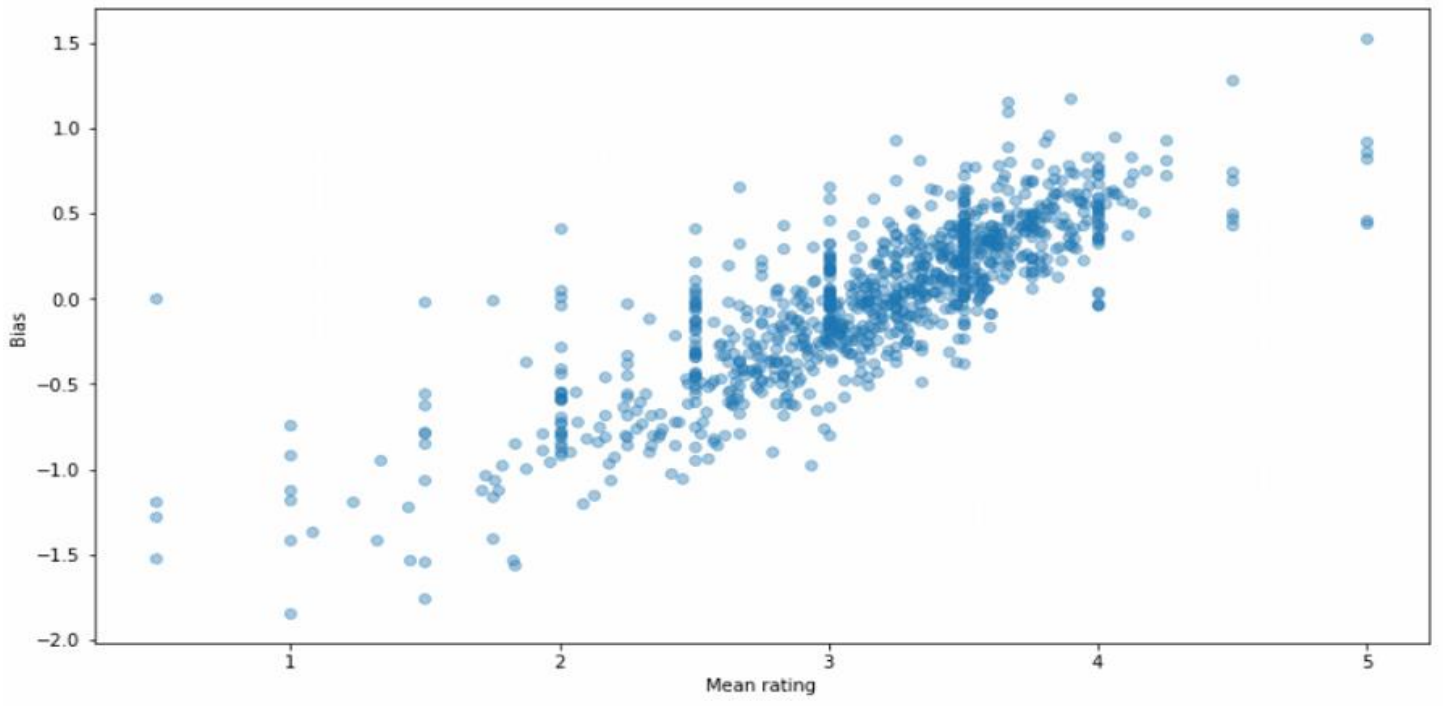


Figure: Scatter plot between movie biases and ratings

From the scatter plot above and with the list of our highest and lowest bias movies, do our model's learned biases agree it's per movie-based biases effect on ratings? Whether there is any positive or negative pattern between them.

It is clearly visible that yes, model has learned biases and as expected we could see that model's ratings and bias values are positively proportional to each other. Highest biased movies have highest ratings.

3.2 Implementing Matrix Factorization architecture:

As we have seen that the deep neural network model has a very complex structured architecture, and which takes huge training time and tends to overfit. Hence, we consider **Matrix factorization architecture**, which is a simpler one that can be a very good thing! Sometimes a simple model will converge quickly to an adequate solution, where a more complicated model might overfit or fail to converge.

MF architecture is nothing, but it takes **userId, movieId** as input units and considers its embeddings which are later send to **dot product** layer to get the predicted rating as an output rather than concatenating the embeddings.

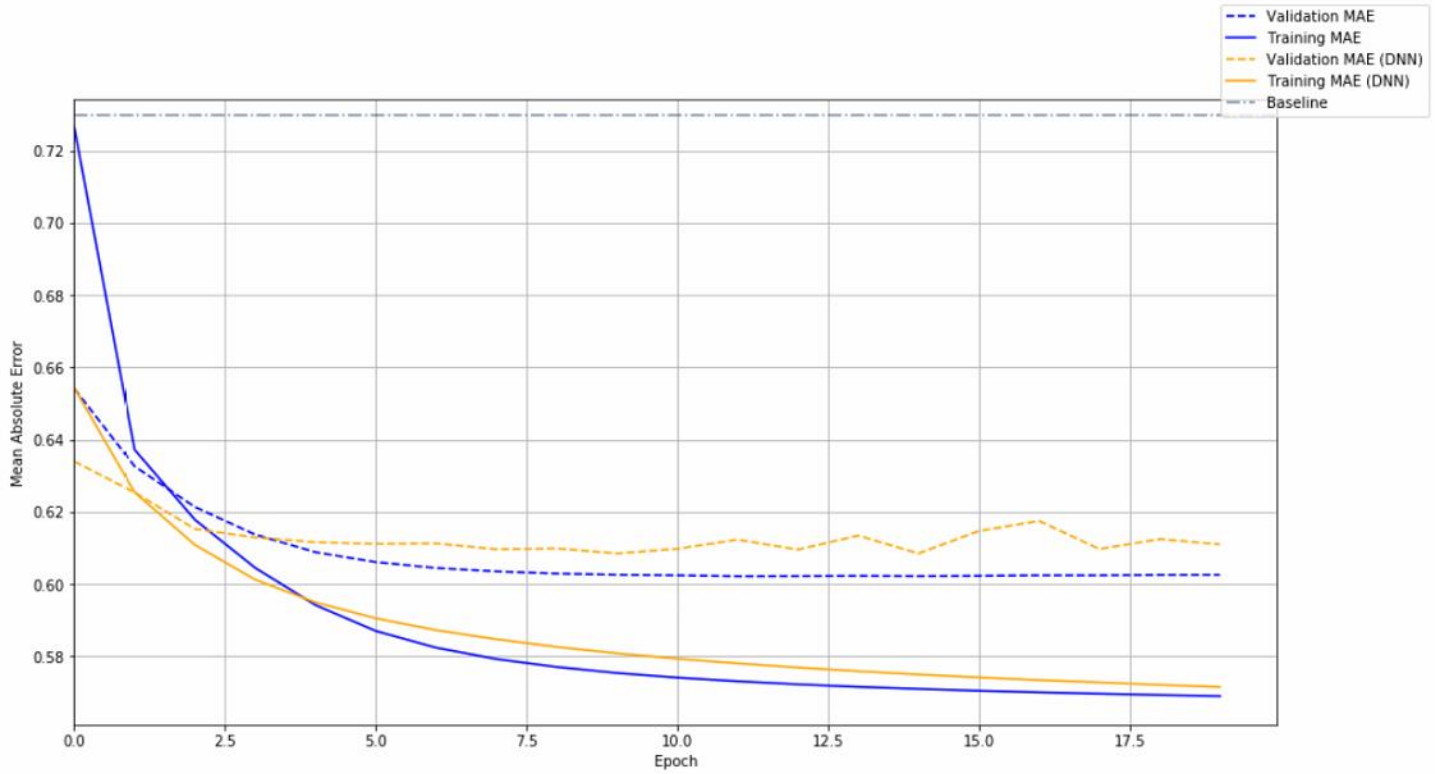


Figure: comparison between Matrix factorization model and DNN model

Our new, simpler model Matrix Factorization (in blue) is looking pretty good.

However, even though our embeddings are small, both models suffer from some obvious overfitting. That is, the error on the training set the solid lines is significantly better than on the unseen data.

I consider that Matrix Factorization architecture (simple model) is better than DNN model in terms of 'validation loss'.

3.2.1 Generating Recommendations: we check how our model performs in predicting the ratings a user would give to some set of movies.

The following table shows a user is taken and a set of 5 movies as shown our model predicts the ratings the user might give to those movies.

	movieId	title	predicted_rating
366	366	Naked Gun 33 1/3: The Final Insult	3.295616
3775	3775	The Naked Gun: From the Files of Police Squad!	4.330486
3776	3776	The Naked Gun 2 1/2: The Smell of Fear	3.827621
5347	5347	Lilo & Stitch	4.216752
10138	10138	The Sisterhood of the Traveling Pants	3.189022

Suppose we're interested in the somewhat more open-ended problem of **generating recommendations**. i.e. given some user ID and some number k, we need to generate a list of k movies we think the user will enjoy.
The most straightforward way to do this would be to calculate the predicted rating this user would assign for *every movie in the dataset*, then take the movies with the k highest predictions.

	movieId	title	genres	year	key	n_ratings	mean_rating	predicted_rating
1233	1233	Evil Dead II (Dead by Dawn)	Action|Comedy|Fantasy|Horror	1987	Evil Dead II (Dead by Dawn)	7788	3.772085	6.419149
2932	2932	Re-Animator	Comedy|Horror|Sci-Fi	1985	Re-Animator	1979	3.476503	6.205729
2374	2374	The Texas Chainsaw Massacre	Horror	1974	Texas Chainsaw Massacre, The (1974)	3026	3.211335	6.200533
4438	4438	The Return of the Living Dead	Comedy|Horror|Sci-Fi	1985	The Return of the Living Dead	1081	3.369103	6.097945
1213	1213	Dead Alive (Braindead)	Comedy|Fantasy|Horror	1992	Dead Alive (Braindead)	2576	3.722632	6.045256

We notice that, when we look at the movies with the highest (or lowest) predicted scores: our model is predicting values outside the allowable range of 0.5-5 stars. For the purposes of recommendation, this is no problem: we only care about ranking movies, not about the absolute values of their predicted scores. But this is still an interesting problem to consider. We see how we could prevent our model from incurring needless errors by making predictions outside the allowable range.

Solutions to this problem:

1. One simple solution would be **limiting our recommendations to movies with at least n ratings**. This feels inelegant, in that we must choose some arbitrary cut-off, and any reasonable choice will probably exclude some good recommendations. It would be nice if we could consider popularity in a 'smoother' way. On the other hand, this is very simple to implement, and we don't even need to re-train our model. We implement a function which will recommend the best movies which have at least some minimum number of ratings by **thresholding**.

	movieId	title	genres	year	key	n_ratings	mean_rating	predicted_rating
1233	1233	Evil Dead II (Dead by Dawn)	Action|Comedy|Fantasy|Horror	1987	Evil Dead II (Dead by Dawn)	7788	3.772085	5.372667
1213	1213	Dead Alive (Braindead)	Comedy|Fantasy|Horror	1992	Dead Alive (Braindead)	2576	3.722632	5.367653
18811	18811	The Cabin in the Woods	Comedy|Horror|Sci-Fi|Thriller	2012	The Cabin in the Woods	1757	3.672738	5.288554
723	723	Cemetery Man (Dellamorte Dellamore)	Horror	1994	Cemetery Man (Dellamorte Dellamore)	1122	3.506239	5.274037
1189	1189	Army of Darkness	Action|Adventure|Comedy|Fantasy|Horror	1993	Army of Darkness	12469	3.725319	5.225542

The predicted rating in the table above is comparatively shows good rating scores than to the recommendations made by function without thresholding, but as we see the rating score range crossed 5 which is not wanted, so we would consider other methods.

2. We train our model by minimizing a loss function. In this case, that's the squared difference between our model's predicted rating and the actual rating. **L2 regularization** adds another term to our model's loss function - a "weight penalty". Now our model must balance making accurate predictions while keeping

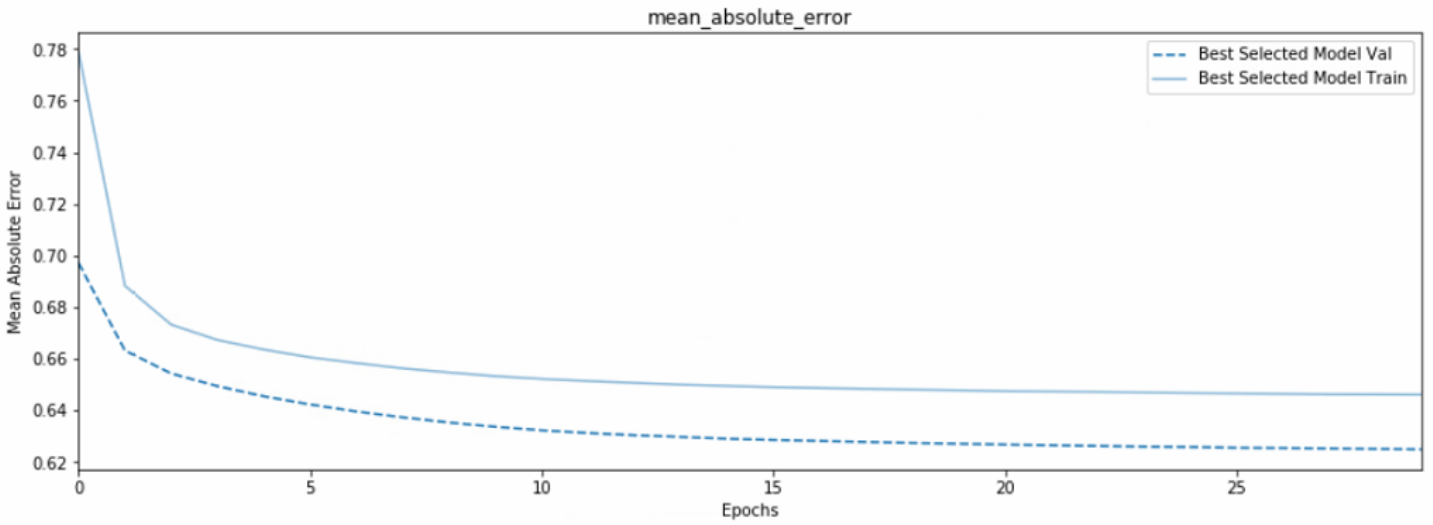
embedding weights not too big. We call this a form of regularization, meaning it's expected to reduce **overfitting** to the training set.

So here comes a question that is **How? And what does this have to do with our obscure recommendation problem?**

Even if a movie has only a single rating in the dataset, our model will, in the absence of regularization, try to move its embedding around to match that one rating. However, if the model has a budget for movie weights, it's not very efficient to spend it on improving the accuracy of one rating out of 20,000,000. Popular movies will be worth assigning large weights. Obscure movies should have weights close to 0.
If a movie's embedding vector is all zeros, our model's output will always be zero (regardless of the user embedding vector).

Recall that an output value of 0 for our model corresponds to a predicted rating equal to the overall average in the training set (around 3.5 stars). This seems like a reasonable behavior to tend toward for movies we have little information about. Hence our final model with best hyperparameters will be having:

1. matrix factorization architecture
2. embedding size = 8
3. embedding L2 penalty
4. dropout (applied to embedding vectors)



Our best model gives best 5 recommendations to a user when given an userId to the recommendation function we have implemented.

	movieId	title	genres	year	key	n_ratings	mean_rating	predicted_rating	l2_predicted_rating
1233	1233	Evil Dead II (Dead by Dawn)	Action|Comedy|Fantasy|Horror	1987	Evil Dead II (Dead by Dawn)	7788	3.772085	6.419149	4.284125
2932	2932	Re-Animator	Comedy|Horror|Sci-Fi	1985	Re-Animator	1979	3.476503	6.205729	4.156466
2374	2374	The Texas Chainsaw Massacre	Horror	1974	Texas Chainsaw Massacre, The (1974)	3026	3.211335	6.200533	4.394135
4438	4438	The Return of the Living Dead	Comedy|Horror|Sci-Fi	1985	The Return of the Living Dead	1081	3.369103	6.097945	4.181310
1213	1213	Dead Alive (Braindead)	Comedy|Fantasy|Horror	1992	Dead Alive (Braindead)	2576	3.722632	6.045256	4.158440

We see that our best model with matrix factorization architecture, biases included, with L2 regularization on embeddings gives us the best predicted rating values with keeping in pace with the mean ratings. Hence our best model is proved to propose best possible recommendations to a user with ‘n’ movies when passed a userId to recommendation method.

3.3 Exploring embeddings using Genism:

Gensim's is a topic modelling library it's docs and many of its class and method names refer to word embeddings. While the library is most frequently used in the text domain, we can use it to explore embeddings of any sort. Using Genism tools, we explore trained embeddings on our best model.

1. We can find **most similar movies of same genres**. "**kv"** is an instance of **WordEmbeddingsKeyedVectors** with our model's movie embeddings and the titles of the corresponding movies, by thresholding ratings of the movie to be greater than 100.

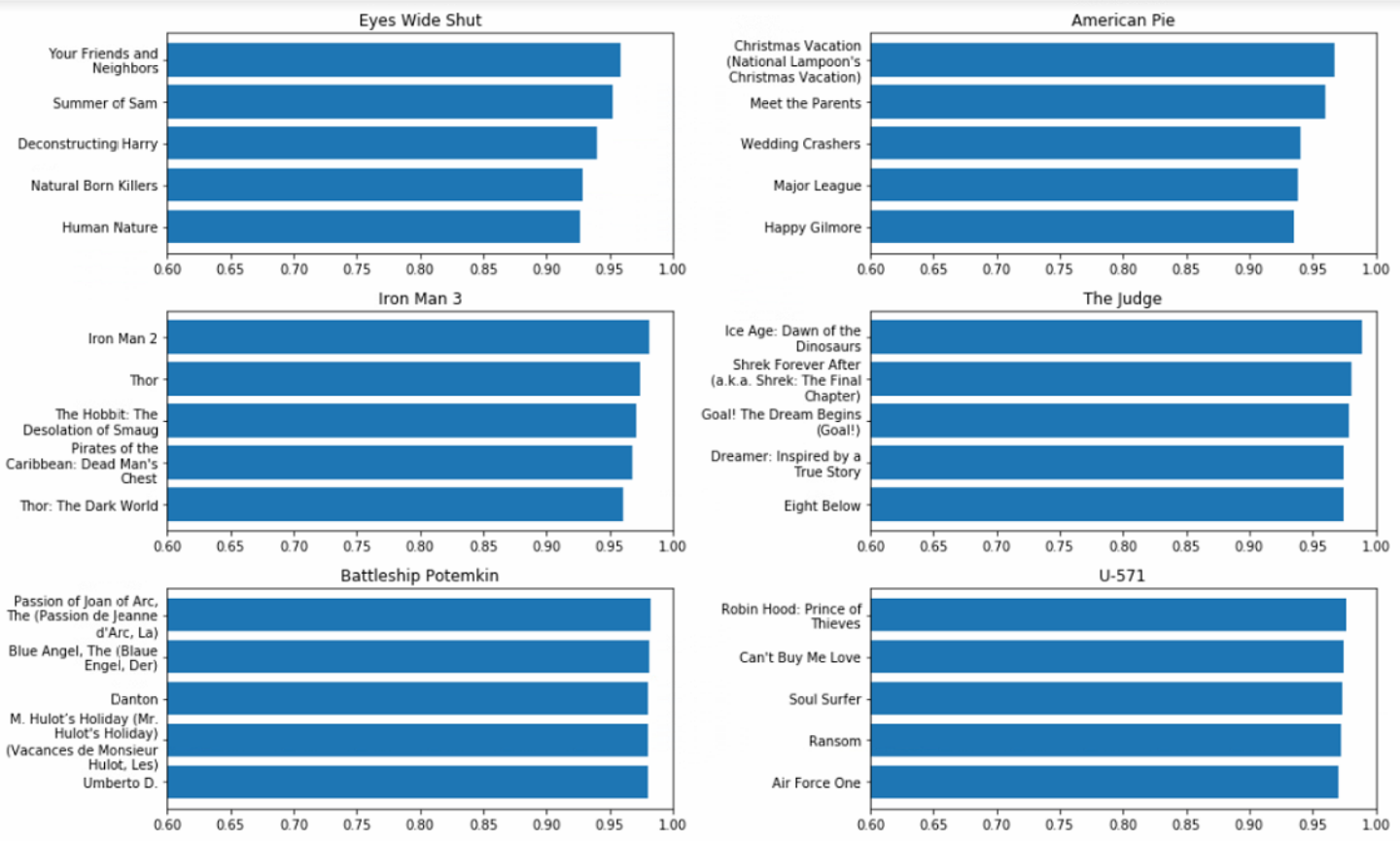
We use `kv.most_similar` method on a few of your favourite movies.

```
# Note: you should always pass 'key' attribute of movies dataset to find similar genre movies
kv.most_similar('Titanic (1953)')
```

```
[("My Best Friend's Wedding", 0.9632135629653931),
 ('Beaches', 0.9585211277008057),
 ('Mrs. Doubtfire', 0.9567484259605408),
 ('The River Wild', 0.9447559118270874),
 ('Homeward Bound: The Incredible Journey', 0.9379501342773438),
 ('Titanic (1997)', 0.9370995163917542),
 ('Ghost', 0.9368044137954712),
 ('My Girl', 0.9362284541130066),
 ('Sleepless in Seattle', 0.9305105209350586),
 ("You've Got Mail", 0.928631603717804)]
```

The above list of movies are the most similar movies to **"Titanic (1953)"** movie.

2. **With genism we can also look at the closest neighbors for a few more movies from a variety of genres:**



Thriller/mystery, Romantic comedy, superhero science fiction, silent drama historical, Action-comedy drama, full action genres, our embeddings manage to nail a wide variety of cinematic niches!

3.4 *Visualizing embeddings using t-SNE:*

Visualizing data in 1 or 2 dimensions is easy - but it's not clear how to visualize embeddings which are 8-dimensional. t-SNE is a dimensionality reduction algorithm which is often used for visualization. It learns a mapping from a set of high-dimensional vectors, to a space with a smaller number of dimensions (usually 2), which is a good representation of the high-dimensional space.

t-SNE tries to make sure that if high-dimensional vectors u and v are close together, then map(u) and map(v) are close together in the 2-d mapping space.

One practical application of visualizing trained embeddings with t-SNE is understanding what information about the embedded entities our model has (and hasn't) learned. This can give us some intuition about how our model works, what latent features it thinks are useful, whether adding certain additional data explicitly might improve the model's accuracy, and so on.

Several Visualizations are made as an experiment to see how different embeddings of data variables effect ratings.

1. Whether Our Embeddings Were Sensitive to Genre:

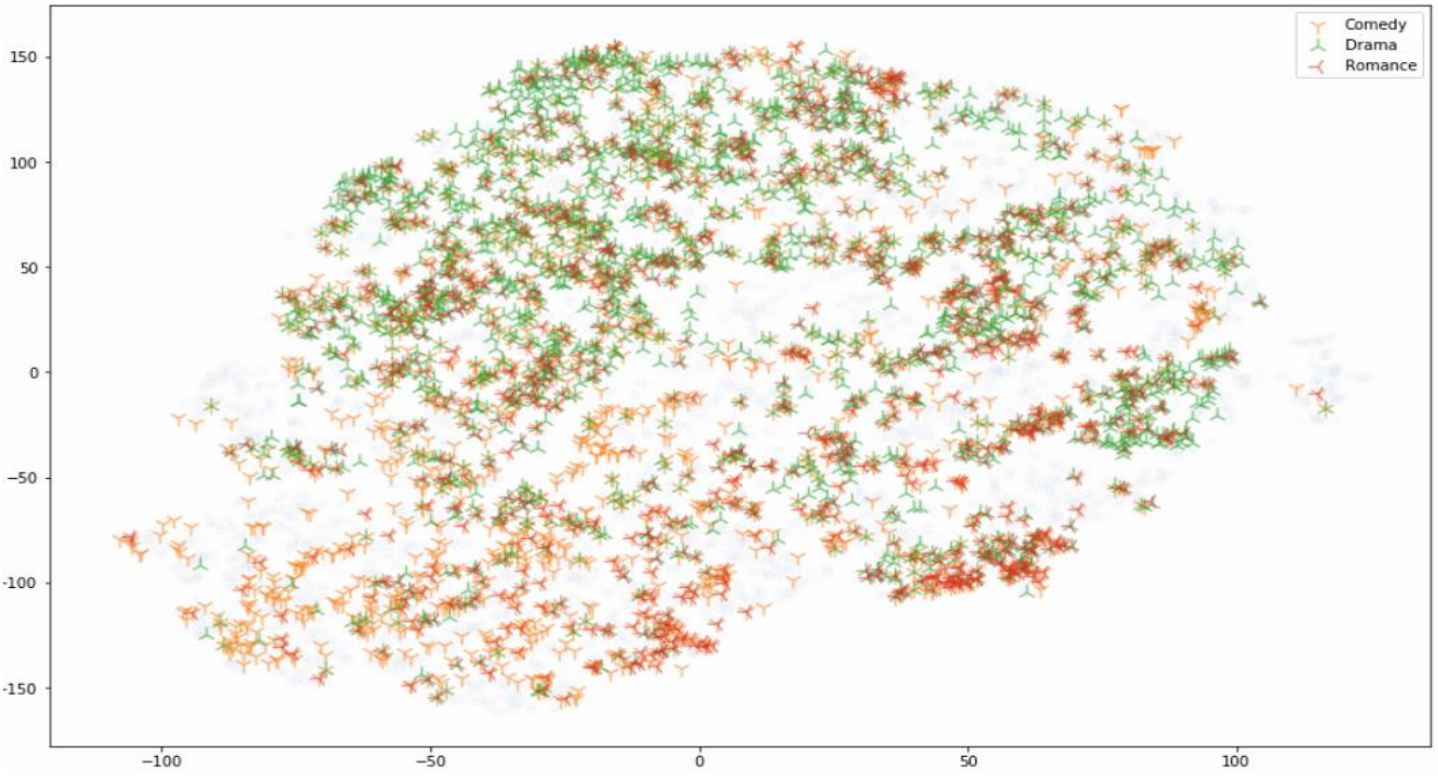
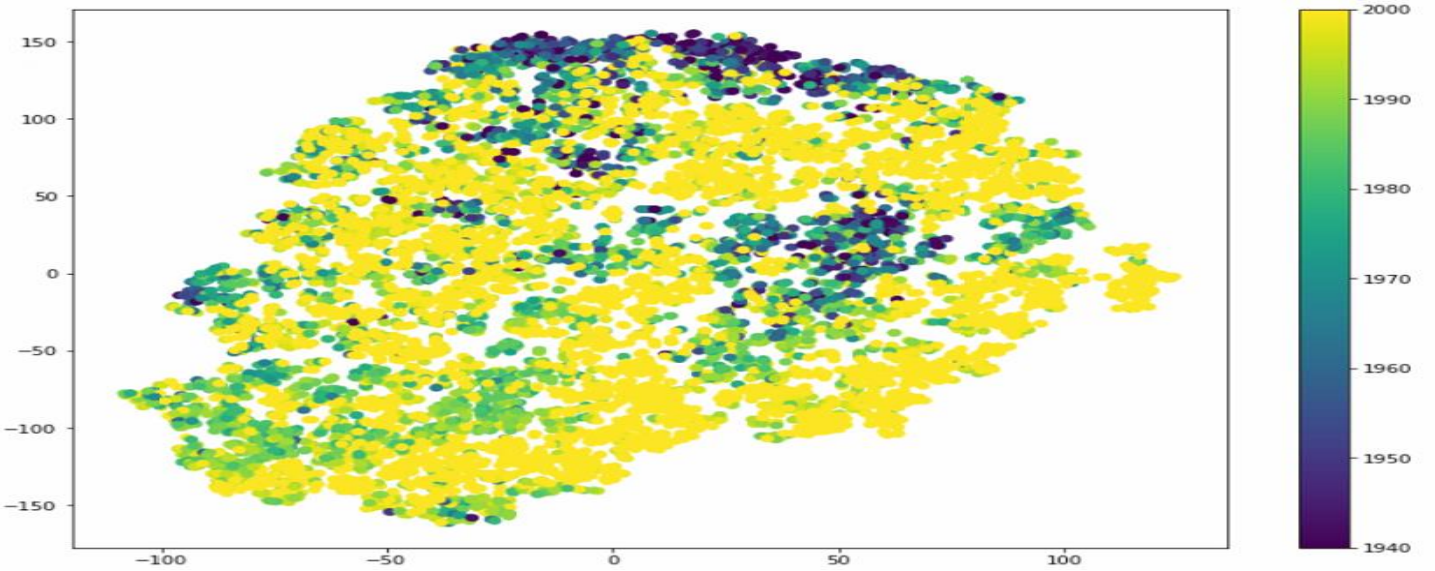


Figure: scatter plot of movie embeddings of different genres

This is an awesome example of structure at the largest scale. Dramas are mostly in the upper-right half, and comedies are mostly in the other half (with romances having a more spread-out, bursty distribution).

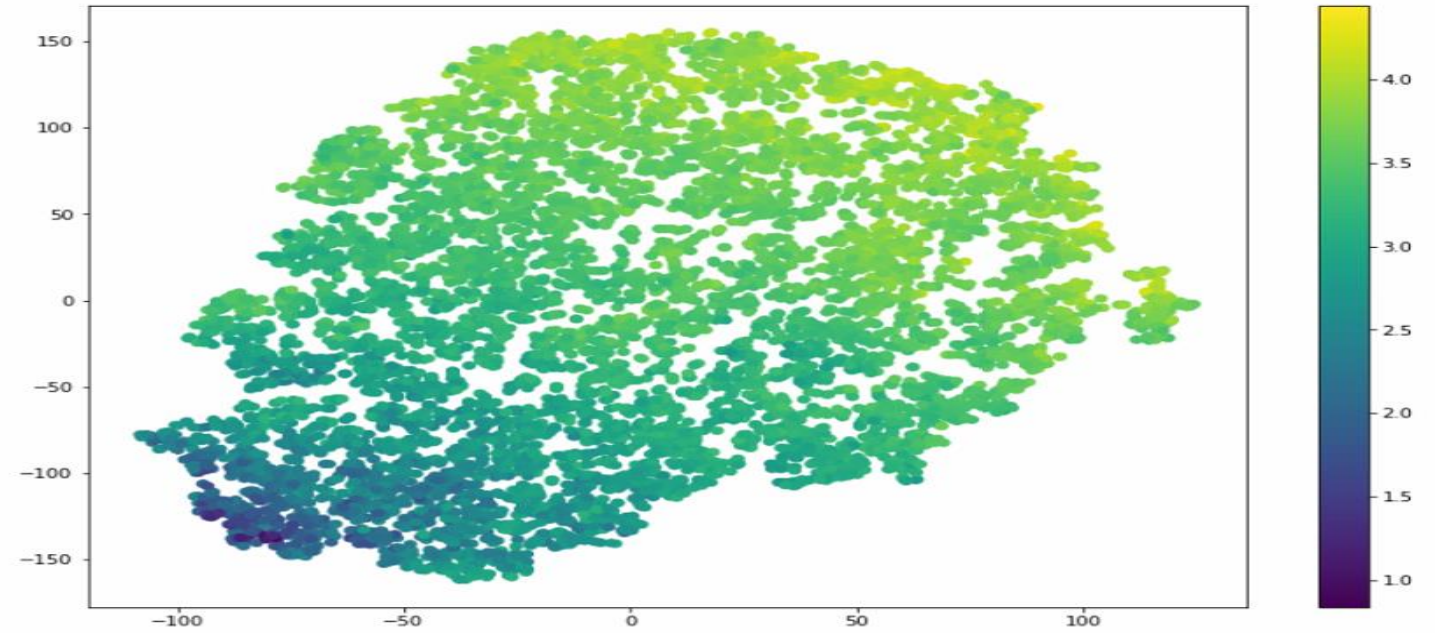
2. Identify patterns in our 2-d embedding space when we group or filter by movie metadata.

- **Based on Release year:** creating a scatter plot of our movie embeddings where a movie's color is set according to its **'release year of movie'**. To check whether there is a global pattern to the distribution of mean rating



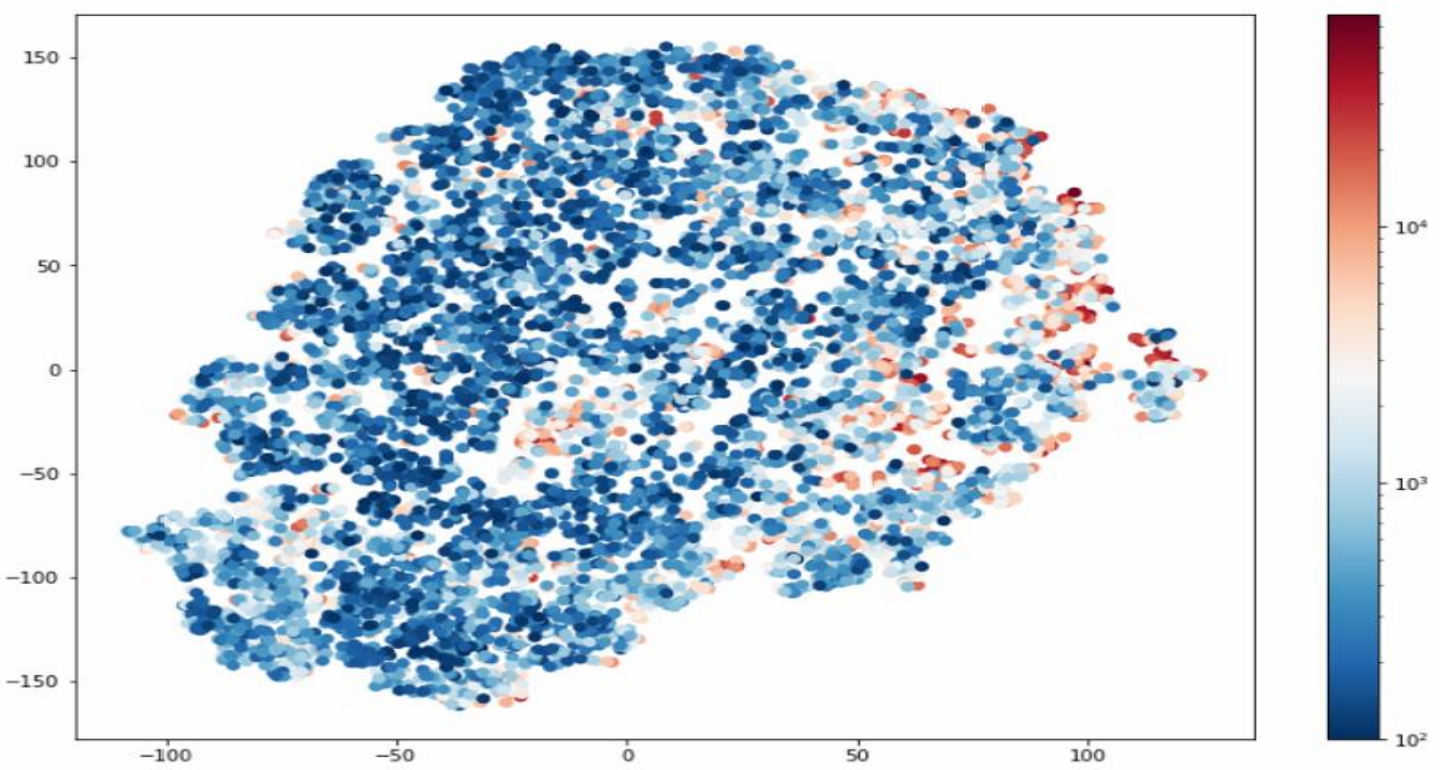
The distribution of year of release does seem to follow some distinct gradients, but the pattern is not global.

- **Based on Average rating:** creating a scatter plot of our movie embeddings where a movie's color is set according to its **'mean rating'**. To check whether there is a global pattern to the distribution of mean rating



Unlike with year of release, there seems to be a clear global pattern here: average rating tends to increase moving from left to right.

- **Based on number of ratings per movie:** Scatter plot showing whether our embeddings reflect the number of ratings we have in the dataset for each movie.



With number of ratings I don't see any pattern either local or global. It seems that there are large number of ratings given by the user below 10k but still you can see the higher number of ratings is not either in a pattern from left to right or right to left.

4. Model Evaluation with metrics:

4.1 Benchmark Model: During my research under this project and its dataset, I found the following two methods which could be considered as benchmark models for the problem statement:

4.1.1 MovieLens 20M using Factorization Machine.

a. They got Mean Absolute Error: **0.60**, and

b. Mean Squared Error: **0.80**.

The research results can be found: https://support.treasuredata.com/hc/en-us/articles/360001260847-MovieLens-20m-Rating-Prediction-using-Factorization-Machine

4.1.2 MovieLens 20M using Autoencoders.

a. They got Mean Squared Error(MSE): **0.81**.

The research results can be found: https://arxiv.org/pdf/1606.07659.pdf

The following table consists of the evaluation metric scores of models we trained based on different constraints and hyperparameter tuning:

Model	**Training MAE (Mean Absoluter error)**	**Validation MAE**	**MSE (Mean Squared Error)**	**Epochs**
DNN model without bias (8d- embedding size)	0.5735	0.6142	0.6513	8
DNN model with bias (8d- embedding size)	0.5735	0.6142	0.6490	8
Matrix Factorization (8d embeddings) Model without bias and regularization	0.5613	0.6071	0.6345	20
Matrix Factorization (8d embeddings) Model with bias and regularization **(Best model)**	0.6421	0.6829	0.6249	30

We see that best model's MAE is larger than other models but still we consider that model as best because as we have discussed in the previous sections, the best model with all it's constraints and hyperparameters results best predictions for movie ratings and movie recommendations. Even though it has slightly larger MAE value I consider it as best model.

Comparing the above metric values to the benchamark models, our model's Validation **MAE** is almost similar to the proposed model, wheras Mean Squared Error value for benchmark models is high when compared to the models designed and implemented using embeddings for movie rating prediction and movie recommendations. Hence, we could say that our model is performing well by reducing loss occurred to a huge extent. **Benchmark model's MSE is :0.81** whereas our **best model's MSE is : 0.6249.** Hence we succeeded in reducing loss.

5. Conclusion and Future Work:

In this project I have concerntrated more on prediction of movie ratings taking user and movie id's as inputs and using embeddings to build a model, explored embeddings using genism, and visualizations of embeddinga using t-SNE to observe any pattern between embeddings and movie meta data. My main goal of this project is to make use of embeddings to build a simple recommendation system. As a result my best model outperformed the benchmark models with less **MSE** value(**0.6249**). I think I have succeeded in implementing my vision and solving the problem I stated.

As a future work there is no end to explore data to extract best features by diving deep into the methods I have used like **genism**, with **t-SNE** visualization of embeddings we could do many more like visualizing one or more variables you find by joining our MovieLens dataset with other movie datasets. For example, imdb-data has some interesting features we're missing including box office revenue, runtime, and director.

6. References:

https://towardsdatascience.com/collaborative-filtering-and-embeddings-part-2-919da17ecefb

https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

https://medium.com/deep-systems/movix-ai-movie-recommendations-using-deep-learning-5903d6a31607

https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

https://www.kaggle.com/kanncaa1/recommendation-systems-tutorial

https://www.kaggle.com/rajmehra03/cf-based-recsys-by-low-rank-matrix-factorization

https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/

https://medium.com/swlh/what-makes-a-hit-film-faa77f501ec7