MOVIES POPULARITY PREDICTION

Hila Shoshan 207931106 **Avi Botershvili 203522743**

Abstract

Movies are one of the most prominent means of entertainment. The widespread use of the Internet in recent times has led to large volumes of data related to movies being generated and shared online. People often prefer to express their views online in English as compared to other local languages.

We used the 5000 movies dataset of [1]TMDB. The dataset consists of genres, rating, popularity, budget, production companies and overview etc of a movie across multiple languages, mostly English.

The popularity of a movie can be identified by its budget, its runtime, its language and many other features. Some are more influential and some less so.
In addition, the movie's overview also play a crucial role in the movie's popularity. While reading the overview/ synopsis, even if the other data attracted the viewer, she now finally decides whether she wants to see the movie or not.
Though the popularity of a movie may depend on multiple factors like the performance of actors, screenplay, direction etc but in most of the cases, its good enough to predict it with the features as mentioned above.

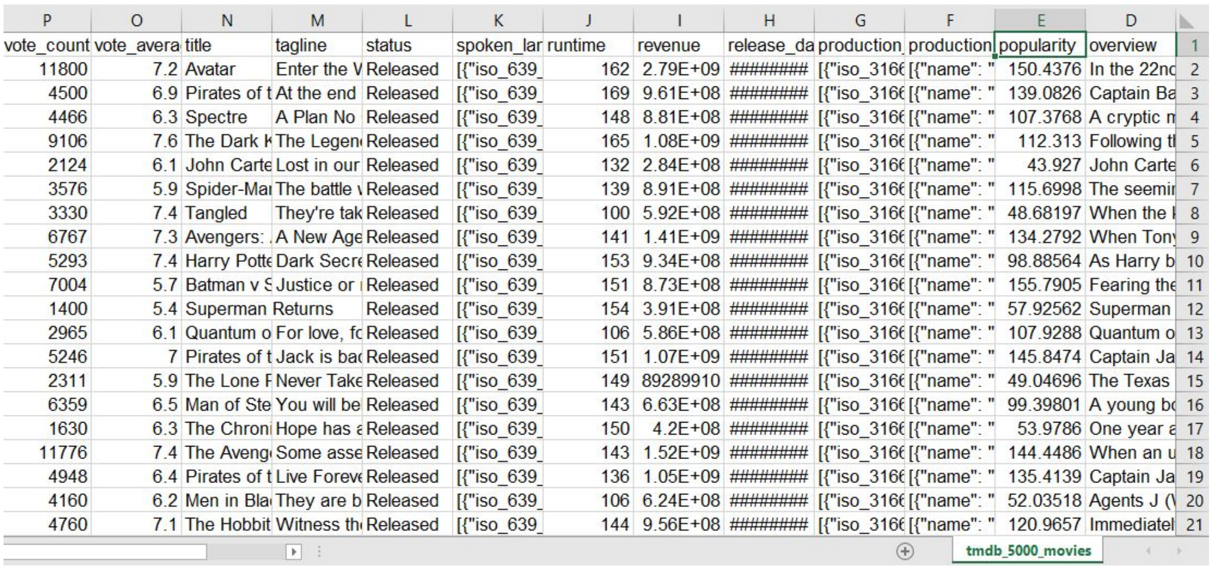
In this work, we provide various model architectures that can be used to predict the popularity of a movie across various features present in our dataset.

1 Introduction

In this paper, we propose multiple deep-learning based methods to predict the popularity of a movie based on its overview, budget, genres etc..
As the amount of data present online increases exponentially day by day, we have reached a point where a human cannot comprehend all of it in a meaningful manner due to its sheer size.

The Movie Database (TMDb) is a community built movie and TV database. It contains various [1] information about many movies. Every piece of data has been added by dating back to 2008 (information from https://www.themoviedb.org/about).

This lead to work on automated recommender systems. The main issue with these kinds of methods is that not all the information is present online and all the information present need not be correct.
Automated movie popularity prediction have a lot of applications. Popular movies are offered as "hot" on movie sites and software such as Netflix, In order to attract as many viewers as possible.
Recommending movies based on how popular the film will be would result in a proper recommendation system. But the main problem here is that people do not often tend to rate the movie they watch, so these sites do not always know what to recommend them, thus automated popularity prediction would be of great help for recommendation systems.
It can be assumed that people will usually be drawn to movies that everyone is talking about, and recommending, and we want to recognize these movies even before they are released.



P	O	N	M	L	K	J	I	H	G	F	E	D	
vote_count	vote_avera	title	tagline	status	spoken_lar	runtime	revenue	release_da	production_	production	popularity	overview	1
11800	7.2	Avatar	Enter the V	Released	[{"iso_639_	162	2.79E+09	#########	[{"iso_3166	[{"name": "	150.4376	In the 22nc	2
4500	6.9	Pirates of t	At the end	Released	[{"iso_639_	169	9.61E+08	#########	[{"iso_3166	[{"name": "	139.0826	Captain Ba	3
4466	6.3	Spectre	A Plan No	Released	[{"iso_639_	148	8.81E+08	#########	[{"iso_3166	[{"name": "	107.3768	A cryptic m	4
9106	7.6	The Dark K	The Legen	Released	[{"iso_639_	165	1.08E+09	#########	[{"iso_3166	[{"name": "	112.313	Following tl	5
2124	6.1	John Carte	Lost in our	Released	[{"iso_639_	132	2.84E+08	#########	[{"iso_3166	[{"name": "	43.927	John Carte	6
3576	5.9	Spider-Mai	The battle v	Released	[{"iso_639_	139	8.91E+08	#########	[{"iso_3166	[{"name": "	115.6998	The seemir	7
3330	7.4	Tangled	They're tak	Released	[{"iso_639_	100	5.92E+08	#########	[{"iso_3166	[{"name": "	48.68197	When the k	8
6767	7.3	Avengers: .	A New Age	Released	[{"iso_639_	141	1.41E+09	#########	[{"iso_3166	[{"name": "	134.2792	When Tony	9
5293	7.4	Harry Potte	Dark Secre	Released	[{"iso_639_	153	9.34E+08	#########	[{"iso_3166	[{"name": "	98.88564	As Harry b	10
7004	5.7	Batman v S	Justice or	Released	[{"iso_639_	151	8.73E+08	#########	[{"iso_3166	[{"name": "	155.7905	Fearing the	11
1400	5.4	Superman	Returns	Released	[{"iso_639_	154	3.91E+08	#########	[{"iso_3166	[{"name": "	57.92562	Superman	12
2965	6.1	Quantum o	For love, fc	Released	[{"iso_639_	106	5.86E+08	#########	[{"iso_3166	[{"name": "	107.9288	Quantum o	13
5246	7	Pirates of t	Jack is bac	Released	[{"iso_639_	151	1.07E+09	#########	[{"iso_3166	[{"name": "	145.8474	Captain Ja	14
2311	5.9	The Lone F	Never Take	Released	[{"iso_639_	149	89289910	#########	[{"iso_3166	[{"name": "	49.04696	The Texas	15
6359	6.5	Man of Ste	You will be	Released	[{"iso_639_	143	6.63E+08	#########	[{"iso_3166	[{"name": "	99.39801	A young bc	16
1630	6.3	The Chroni	Hope has a	Released	[{"iso_639_	150	4.2E+08	#########	[{"iso_3166	[{"name": "	53.9786	One year a	17
11776	7.4	The Aveng	Some asse	Released	[{"iso_639_	143	1.52E+09	#########	[{"iso_3166	[{"name": "	144.4486	When an u	18
4948	6.4	Pirates of t	Live Foreve	Released	[{"iso_639_	136	1.05E+09	#########	[{"iso_3166	[{"name": "	135.4139	Captain Ja	19
4160	6.2	Men in Bla	They are b	Released	[{"iso_639_	106	6.24E+08	#########	[{"iso_3166	[{"name": "	52.03518	Agents J (\	20
4760	7.1	The Hobbit	Witness th	Released	[{"iso_639_	144	9.56E+08	#########	[{"iso_3166	[{"name": "	120.9657	Immediatel	21

picture 1: picture of the our dataset, as a csv file.

2 Related Work

Work has been done in related areas in the past. Basu et al. (Basu et al., 1998) propose an inductive learning approach to predict user preferences.

Rasheed et al. (Rasheed and Shah, 2002) present a method to classify movies on the basis of audio-visual cues present in previews which contain important information about the movie.

Chin-Chia Michael Yeh et al. (Yeh and Yang, 2012) concerns the development of a music codebook for summarizing local feature descriptors computed over time.

Aida Austin et al. (Austin et al., 2010) created a database of film scores from 98 movies containing instrumental (non-vocal) music from 25 romance, 25 drama, 23 horror, and 25 action movies. Both pair-wise genre classification and classification with all four genres was performed using support vector machines(SVM) in a ten-fold cross-validation test.

Muhammad Hassan Latif et al. (Latif et al. 2016) predicted movies popularity using machine learning techniques.

Vasu Jain et al. (Vasu Jain et al. 2013) predicted movie success using sentiment analysis of tweets.

And there are also recent studies, for example Wen Bai et al. (Wen Bai et al. 2020) presented a predicting movie popularity via cross-platform feature fusion.

And many others.

Required background: Linear Regression Model, Early Stopping, Lasso and Ridge Regularization, Multi-Layer Perceptron, Tokenization, RNN, LSTM.

---

3 First Part of Work: Learn about the data

We used *pandas* library to read the dataset (that given as an csv file).

First we learned about our label, the *popularity* column.

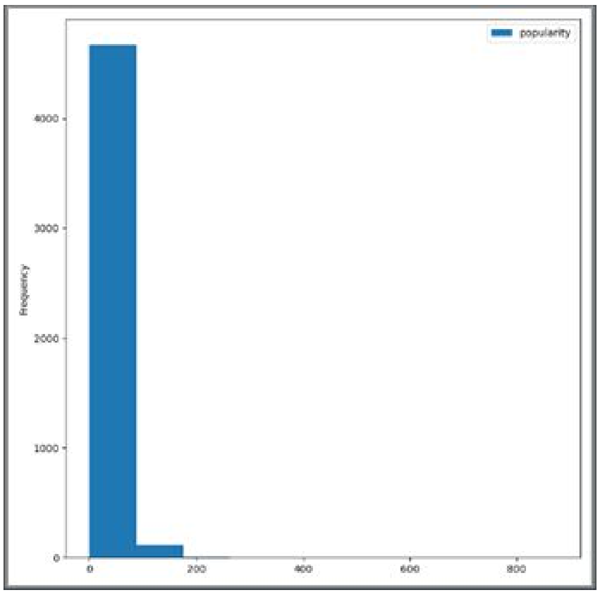
popularity is measured as the cumulative number of star ratings.

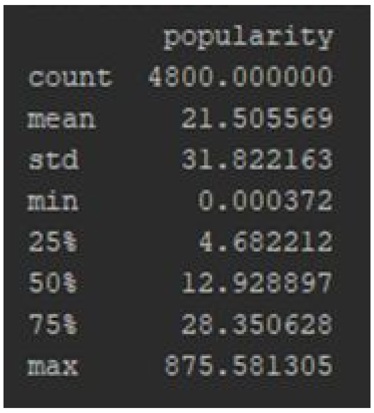
The popularity value for movies builds with data like number of votes for the day, number of views for the day, number of users who marked it as a "favourite" for the day, number of users who added it to their "watchlist" for the day, number of total votes. (this information is taken from TMDB API).

Let's see some statistics & a histogram that describes the data distribution:

[2]

[2] The statistics and histogram were done using *pandas* and *matplotlib* python libraries.





```
        popularity
count  4800.000000
mean     21.505569
std      31.822163
min       0.000372
25%       4.682212
50%      12.928897
75%      28.350628
max     875.581305
```

On the histogram above we can see that most of the data on the dataset has 0-100 popularity value. Therefore, the model probably learns to predict better lower results, and has higher error on large popularity value.

We checked and saw there are no null-values on the label.

Then we took all the columns, those mentioned in the project proposal, and choose what to leave and what to drop.

Finally we choose 'runtime', 'production_companies', 'genres', 'original_language', 'production_countries', 'release_date', 'vote_count', 'vote_average', 'budget', and 'overview' to be our features. While the last one used only on the RNN (last) part.

We sorted the *vote_count* column and saw that the movies with the best vote averages were only reviewed one or two times! So we defined a threshold of 10 on vote counts, and later we deleted the movies that did not meet the threshold requirement.

About *original_language*: we saw that almost all the movies have "English" on this column, so we've decided to divide all the languages to two groups: English or not English. That would add one new column to the dataset: isEnglish, instead of full of columns for each language, using one-hot-encoding method.

We dropped out revenue as a feature, as this is something revealed a while after a movie comes out.

We looked for nulls, and saw that 'release_date' and 'runtime' together have only 3 rows with None value, so we drop these rows.

In addition, we drop any values with a runtime of zero, because it's not possible that it's the real value.

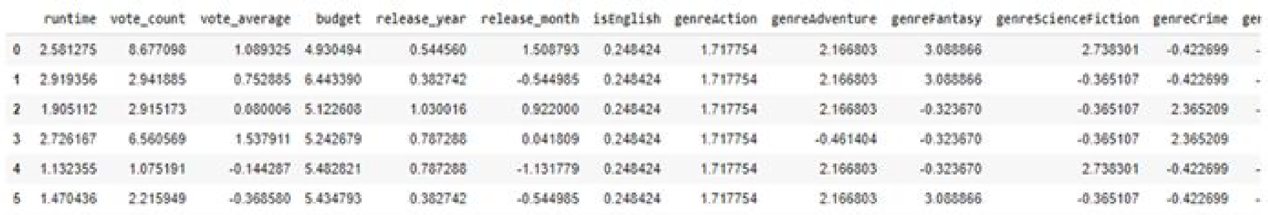
4 Second Part: Arranging the data

We removed the unwanted rows mentioned before, and defined the dataset with the relevant features and rows as *df_x*, and the label defined as *df_y.*

Then we took care of the non-numeric columns, such as:

- release_date column split into two new columns: release_year and release_month.
  We assumed that the release day does not affect popularity at all, while the year and the month does. (maybe in later years, more people had technological means and so they saw more movies. Also, it is possible that in some months people tend to watch more movies than in other months – in winter for example).
- The *original_language* column has been replaced with a binary column 'isEnglish'.
- For the additional categorical columns, production_companies, genres and production_countries, it was a little different because each entry had a list of some genres/companies/countries, and not just one value.
  So we implemented a method that took the relevant values from each list, created new columns where each example has '1' in a column if the value it represents was in her list, and '0' else.

In the end we normalized *df_x*:



	runtime	vote_count	vote_average	budget	release_year	release_month	isEnglish	genreAction	genreAdventure	genreFantasy	genreScienceFiction	genreCrime	ge
0	2.581275	8.677098	1.089325	4.930494	0.544560	1.508793	0.248424	1.717754	2.166803	3.088866	2.738301	-0.422699	-
1	2.919356	2.941885	0.752885	6.443390	0.382742	-0.544985	0.248424	1.717754	2.166803	3.088866	-0.365107	-0.422699	-
2	1.905112	2.915173	0.080006	5.122608	1.030016	0.922000	0.248424	1.717754	2.166803	-0.323670	-0.365107	2.365209	-
3	2.726167	6.560569	1.537911	5.242679	0.787288	0.041809	0.248424	1.717754	-0.461404	-0.323670	-0.365107	2.365209	
4	1.132355	1.075191	-0.144287	5.482821	0.787288	-1.131779	0.248424	1.717754	2.166803	-0.323670	2.738301	-0.422699	-
5	1.470436	2.215949	-0.368580	5.434793	0.382742	-0.544985	0.248424	1.717754	2.166803	3.088866	-0.365107	-0.422699	-

Then we split the dataset into train set (80%) and test set (20%), and started running the models on it.

5 Running a simple Linear Regression

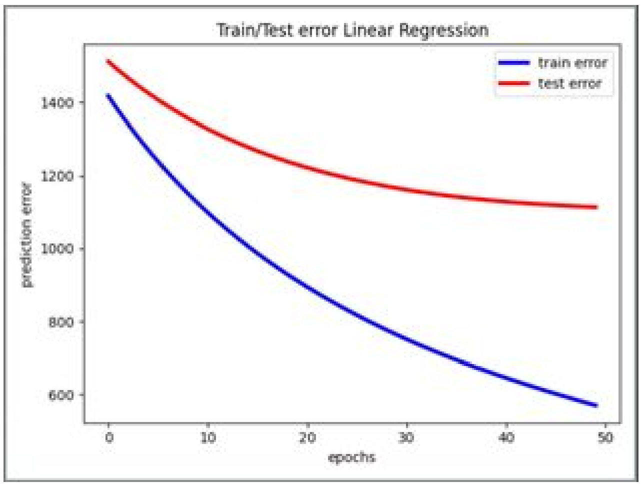
We implemented a linear regression model using *tensorflow* library.

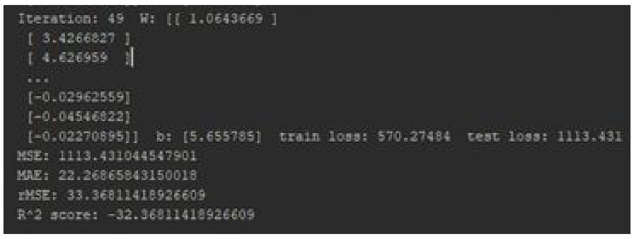
We used learning rate alpha = 0.001, MSE loss function, and initialized the weights and bias to zeros.

In addition, we used batches of size 120, and 50 epochs.

The minimal loss value (in the end of the training) was 570.27484 for train, and 1113.431 for test.

Here a graph that represents the Train/Test error (computed while training the model):



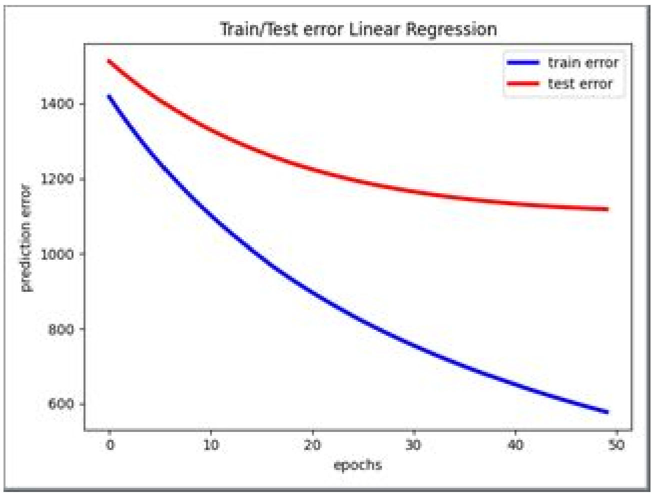


```
Iteration: 49  W: [[ 1.0643669 ]
 [ 3.4266827 ]
 [ 4.626959  ]
 ...
 [-0.02962559]
 [-0.04546822]
 [-0.02270895]]  b: [5.655785]  train loss: 570.27484  test loss: 1113.431
MSE: 1113.431044547901
MAE: 22.26865843150018
rMSE: 33.36811418926609
R^2 score: -32.36811418926609
```

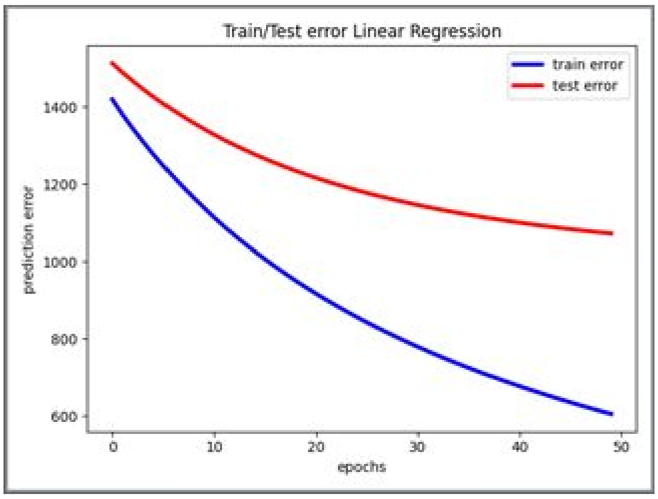
Than we add regularization, and got the following results:

Lasso: train loss is 605.5994, and test loss is 1072.196

Ridge: train loss is 577.85767, and test loss is 1117.633



Lasso

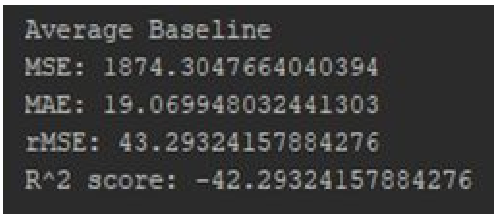


Ridge

6 Average Baseline for comparison

We create a function that computes the average on train data, and use it as prediction.

Than we used our compute_error function to compute the MSE and MAE when using the average prediction, and got the following results:



```
Average Baseline
MSE: 1874.3047664040394
MAE: 19.069948032441303
rMSE: 43.29324157884276
R^2 score: -42.29324157884276
```

We can see that Linear Regression is better than that average prediction.

---

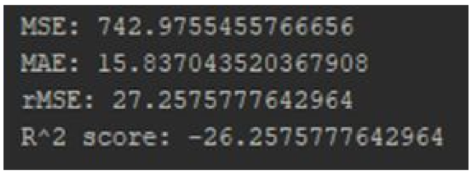
7 Multi-Layer Perceptron with 1 hidden layer

We implemented NLP using hidden layer with 50 neurons, initialize random weights, b1=0.1, b2=0.

We also used relu as activation function, and MSE function as loss.

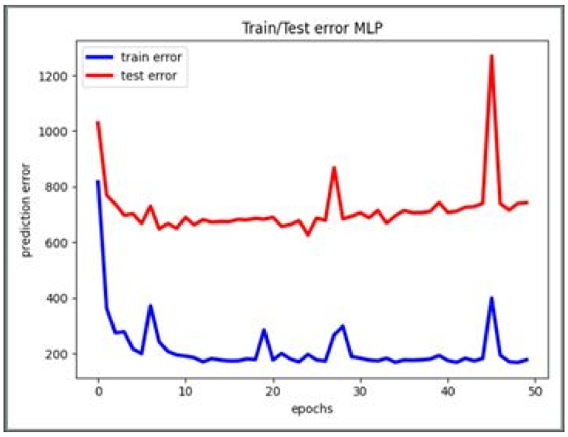
First we used batches of size 120, and 50 epochs.

Results:



```
MSE: 742.9755455766656
MAE: 15.837043520367908
rMSE: 27.2575777642964
R^2 score: -26.2575777642964
```

Plot:



The MSE here is better (smaller) than the average baseline, and also than the linear regressor.

8 Neural Network with 2 hidden layers

On hidden layer 1 and hidden layer 2 there are 100 and 50 neurons respectively.

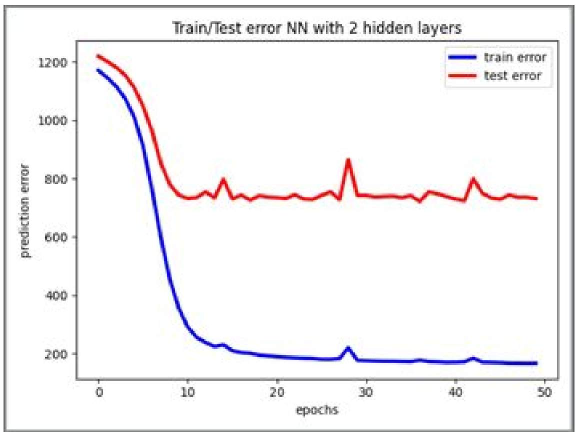
Weights: Initialized to random ones.

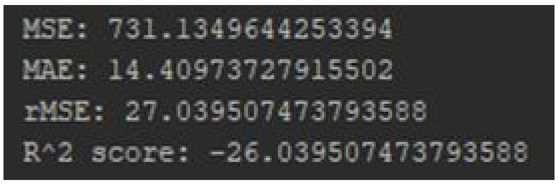
Biases: determined as 0.1 (b1=b2=b3=0.1)

Loss function: MSE

Activation function: relu

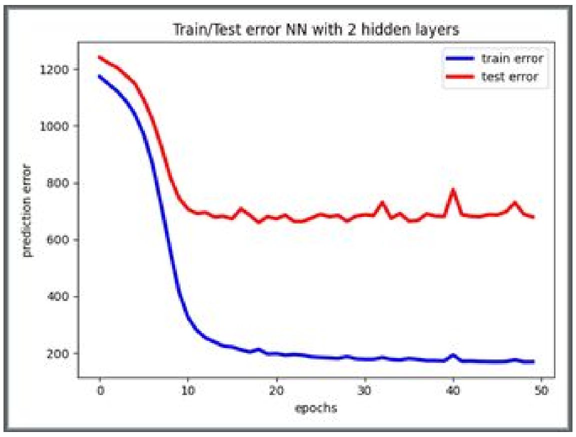
Learning rate using in gradient decent optimizer: 0.0001 (we first tried alpha=0.001, but it was too big so the loss has "jumped" and finally its value has reached to infinity).

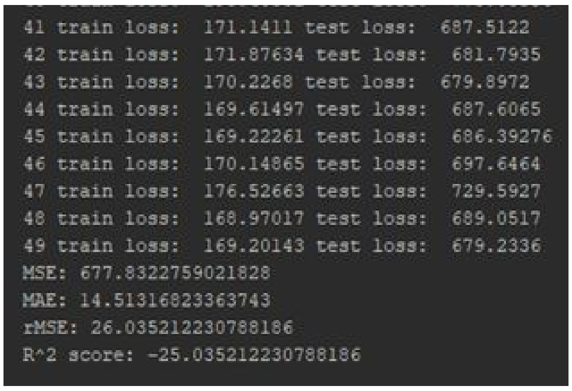




```
MSE: 731.1349644253394
MAE: 14.40973727915502
rMSE: 27.039507473793588
R^2 score: -26.039507473793588
```

with Lasso regularization:





```
41 train loss:  171.1411 test loss:  687.5122
42 train loss:  171.87634 test loss:  681.7935
43 train loss:  170.2268 test loss:  679.8972
44 train loss:  169.61497 test loss:  687.6065
45 train loss:  169.22261 test loss:  686.39276
46 train loss:  170.14865 test loss:  697.6464
47 train loss:  176.52663 test loss:  729.5927
48 train loss:  168.97017 test loss:  689.0517
49 train loss:  169.20143 test loss:  679.2336
MSE: 677.8322759021828
MAE: 14.51316823363743
rMSE: 26.035212230788186
R^2 score: -25.035212230788186
```

- Because of the asymmetrical label, the model probably has higher error on large popularity value, what can cause large MSE, which is relatively sensitive to errors on large number.
- But the other error calculation – MAE, is much smaller!
- Also the NN results are all better than both Linear Regression Model and Average Baseline, so it's ok.

---

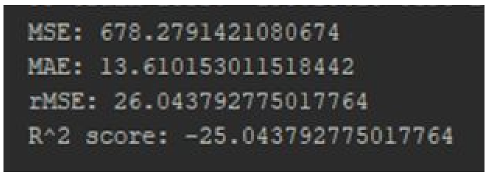
9 Implements early stopping to prevent overfitting

We split the data again to create a validation set too [now we have train size = 60%, test size = 20%, and validation size = 20%].

We add to train_NN function option of early_stopping.

If it asked, then in the end of each epoch, we check if there was an improvement in the last 10 epochs (the default require_improvement). If not, we stop the training and return results.

In fact, it did not improve our algorithm, and we got similar results:



```
MSE: 678.2791421080674
MAE: 13.610153011518442
rMSE: 26.043792775017764
R^2 score: -25.043792775017764
```

---

10 Implements RNN with LSTM

In this part of the work, we focused on the *overview* data and *popularity* data. The overview is a sequential column (textual - words in a sequence), so it can fit very well into the RNN network.

First we created a vocabulary, consisting of all the words in the whole dataset's examples overviews.

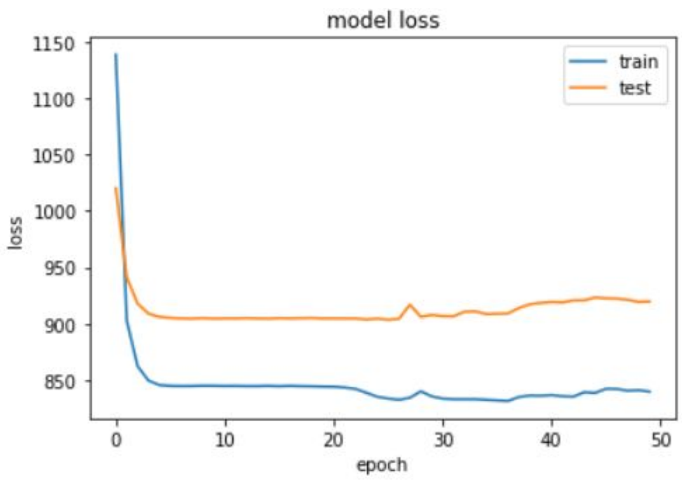
Then we used word embedding to represent each word.
These words/ vectors used as features of the RNN model.

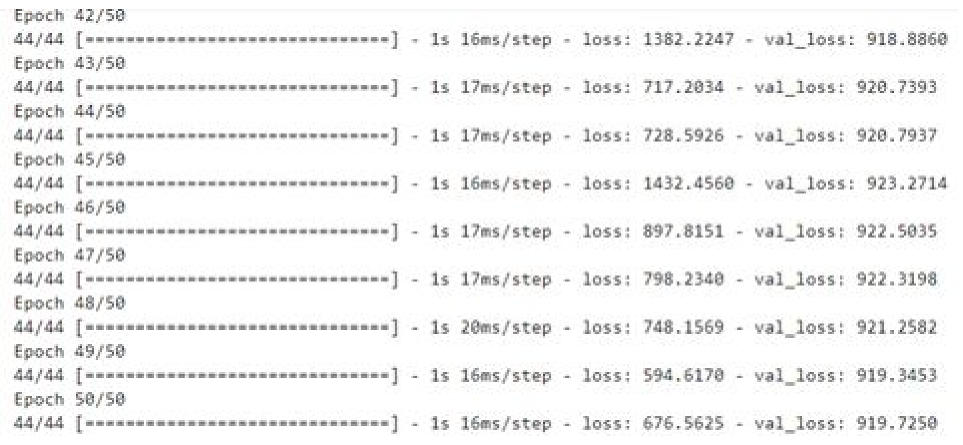
We used a regression analysis with Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) to build a model that doesn't only consider the individual words, but the order they appear in.

This leads to a powerful model for making predictions.

The activation function we used was relu, learning rate alpha = 0.001, Loss function: MSE (Mean Squared Error), batch size of 60, 50 epochs.

The results:





```
Epoch 42/50
44/44 [==============================] - 1s 16ms/step - loss: 1382.2247 - val_loss: 918.8860
Epoch 43/50
44/44 [==============================] - 1s 17ms/step - loss: 717.2034 - val_loss: 920.7393
Epoch 44/50
44/44 [==============================] - 1s 17ms/step - loss: 728.5926 - val_loss: 920.7937
Epoch 45/50
44/44 [==============================] - 1s 16ms/step - loss: 1432.4560 - val_loss: 923.2714
Epoch 46/50
44/44 [==============================] - 1s 17ms/step - loss: 897.8151 - val_loss: 922.5035
Epoch 47/50
44/44 [==============================] - 1s 17ms/step - loss: 798.2340 - val_loss: 922.3198
Epoch 48/50
44/44 [==============================] - 1s 20ms/step - loss: 748.1569 - val_loss: 921.2582
Epoch 49/50
44/44 [==============================] - 1s 16ms/step - loss: 594.6170 - val_loss: 919.3453
Epoch 50/50
44/44 [==============================] - 1s 16ms/step - loss: 676.5625 - val_loss: 919.7250
```

---

11 Summary of results and conclusions

Our Linear Regression was better than the average baseline.

All the models of the Neural Networks using 'runtime', 'production_companies', 'genres', 'original_language', 'production_countries', 'release_date', 'vote_count', 'vote_average' and 'budget' as features, were the best.

We concluded that predict popularity by all those features, using a MLP model is better than predict the popularity only by ‘overview’, with RNN model.

[3]

[3] Our Project include the code on GITUB:
https://github.com/HilaShoshan/moviesPopularityPrediction.git