Movie recommender system

23 - Nov - 2018

Arturo Alam Téllez Villagómez A01205569
Fermín Alejandro Moreno Cornejo A01209665
Sistemas Inteligentes
ITESM Campus Querétaro

Justification

On-demand providers face big challenges to keep their customers. One of them is to provide them with personalized suggestions so that they keep consuming their products/services. This is why they invest a lot of resources to deliver better recommendation systems.

For this challenge we chose to use collaborative filtering, which is a technique for recommendation systems where its purpose is to make predictions based other users data, under the assumption that if user A has a similar opinion to user B, then it is more likely that for a different issue, A will have a similar opinion to B than any other random user.

For this problem we chose two algorithms and compared their performance, the algorithms chosen were K-nearest-neighbors (KNN) and Single Value Decomposition (SVD).

Documentation

Instructions

To run this project you will need Anaconda package that gathers the most common libraries and IDEs for data science and artificial intelligence development using this link.

https://www.anaconda.com/download/#linux

Select to download the python 3.x version

Locate your downloaded file in your terminal and execute the following instruction.

```
bash Anaconda-latest-Linux-x86_64.sh
```

This will start running the installation script

After it finishes, you will need to install the surprise library that sits on top of sci-kit learn running the following command.

```
conda install -c conda-forge scikit-surprise
```

Once downloaded, locate yourself in the directory `song_recommendation`, and run the following instruction:

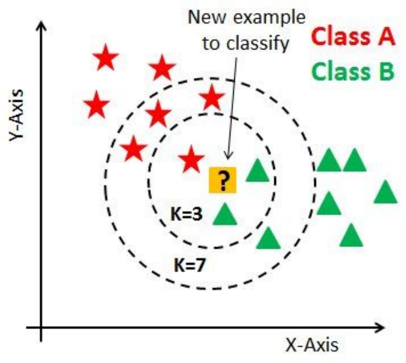
```
jupyter-notebook
```

Copy the link it gives you on any browser, and select the file `*.ipynb`

For our project we used the MovieLens 1m dataset which is an industry standard for the evaluation of recommendation systems, it is highly similar to the one used in the Netflix challenge, it consists in 1 million ratings from 6000 anonymous users on 4000 movies.

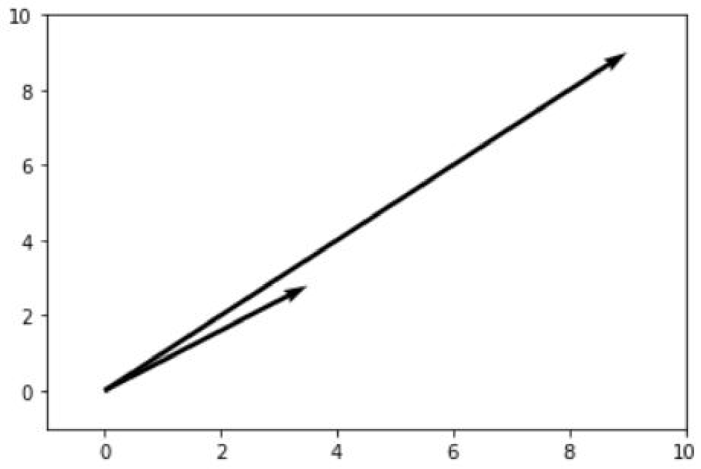
K Nearest Neighbors

KNN is a classification algorithm used to determine the class of an item based on the similarity of the k nearest neighbors. It can be exemplified here:



We used this approach to predict the rating of a movie determined on the similar movie ratings of the nearest movies around it.

To measure similarity, the cosine rule states that the distance between two vectors is not going to be determined on their coordinates euclidean distance, but the orientation in which they are directed. The difference can be seen in this example:



Cosine similarity is a measure of orientation, not magnitude.

If this vectors are orthogonal, meaning the angle between them is 90° then, it's similarity is 0 because cos(90) = 0. This is highly relevant to our case because we can restrict our vector to non-negative values between 1 and 5 which correspond to cosine similarities between 1 and 0.

How is this going to work?

Having a "u x m" matrix being u the number of users and m the number of movies, each element in the matrix represents how a user rated a movie. Each rating is expected to be a value between 1 and 5, and we have 0 as a non-rated movie by that user.

What the algorithm will do is to predict the rating of a movie that the user hasn't seen, based on what other users have rated that movie by selecting the nearest users similar to him that have seen the movie doing a weighted average based on similarity and rating.

This weighted average will be the estimation on what rating would the user give to that movie.

Results

We can measure the root mean squared error (RMSE) which is the deviation from square root of the mean of the difference between the predicted value and the observed/actual value. The lower the value is, the better the model.

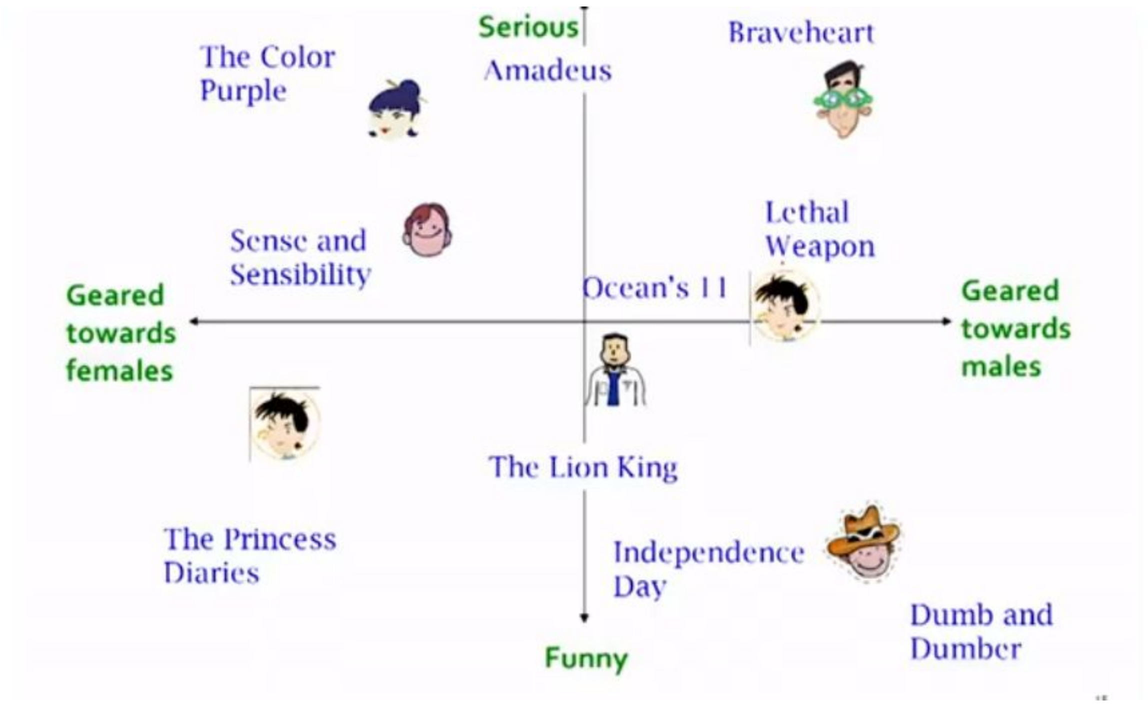
The resulting error for this model is:
RMSE: 0.9832
0.9831581787887709

Single Value Decomposition

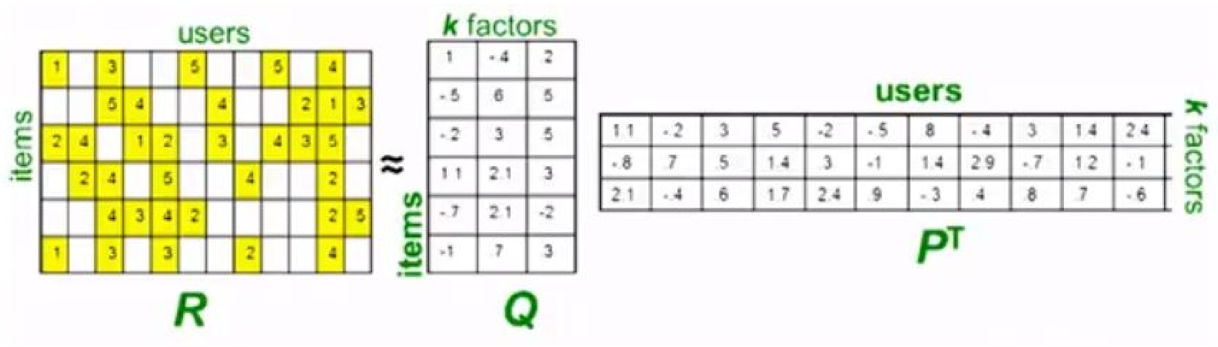
This algorithm was used by Yehuda Koren in the "Bellkor's Pragmatic Chaos" team which led them to win the Netflix Challenge in 2009, the challenge consisted on finding a better recommendation system than the one that Netflix already had by reducing their RMSE by 10% this was only achieved by the team 3 years after the contest started. Latent factor models differ from neighbour models because they attempt to analyze the user-item interaction through factors either inferred or gathered explicitly.

How does it work?

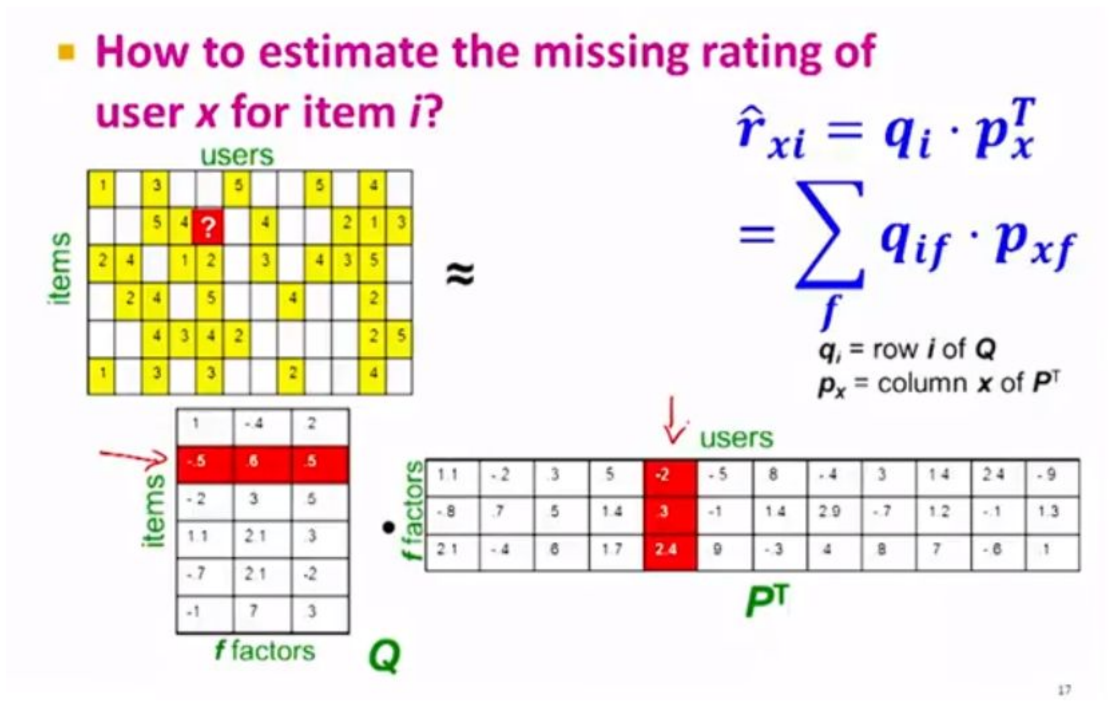
The algorithm does what is called matrix factorization, also known as matrix decomposition, it is a tool for mapping users and movies into a k-factor space.



Every item is represented by a vector q and every user by a vector p, and the elements of p are characterizations of the level of interest that the user has of corresponding movies.



Then by doing the dot product of both vectors, an approximation to the actual or missing rating can be estimated.


How to estimate the missing rating of user x for item i?
users
items
≈
$\hat{r}_{xi} = q_i \cdot p_x^T$
$= \sum_f q_{if} \cdot p_{xf}$
q_i = row i of Q
p_x = column x of P^T
users
f factors
items
f factors
Q
P^T
17

A thing that made this algorithm take the prize home was that it takes into account inferred factors, these are not explicit features in data such as user preferences by genre, searching history, purchase history, and more. SVD allows us to combine both features, the ones of the sparse matrix that provide faithful data, and the more dense matrix with less reliable information, but still helpful to make better predictions.

Result:

RMSE: 0.9450

0.9449797499293449

Conclusions

From our research on both algorithms, we expected SVD to be better and it is, having a lower RMSE. If a inferred factor matrix was given it would be possible to improve the model, but that would be work for the future, outside this project's scope.

References

Bao, Z. (2015). MOVIE RATING ESTIMATION AND RECOMMENDATION. Retrieved November 12, 2018, from http://cs229.stanford.edu/proj2012/BaoXia-MovieRatingEstimationAndRecommendation_FinalWriteup.pdf

Byström, H. (n.d.). Recommendations from user ratings. Retrieved November 12, 2018, from http://cs229.stanford.edu/proj2013/Bystrom-MovieRecommendationsFromUserRatings.pdf

Gower, S. Netflix Prize and SVD. April 18, 2014. Retrieved from http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf

Leskovek, Rajaraman. Standford University. April 13, 2016. Latent Factor Recommender System | Stanford University. Retrieved from https://www.youtube.com/watch?v=E8aMcwmqsTg&feature=youtu.be.

Leskovek, Rajaraman, Ullman. Standford University. April 13, 2016. Finding the Latent Factors | Stanford University | Stanford University. Retrieved from https://www.youtube.com/watch?v=GGWBMg0i9d4

Perone, S. (2013, September 12). Cosine Similarity for Vector Space Models. Retrieved November 10, 2018, from http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/

Vernier. (2018, July 5). Mean Squared Error and Root Mean Squared Error. Retrieved November 1, 2018, from https://www.vernier.com/til/1014/