Movie Recommendation System

Tiberiu Simion Voicu

Post graduate thesis project, ECS751P

Abstract

Recommender systems aim to provide users with personalized recommendations of items. This helps a lot in filtering irrelevant items from the big collection. It is usually achieved through collaborative filtering. Neural network have proven to be highly effective in natural language processing, computer vision and other areas. Despite this they haven't seen as much use in solving recommendation problems. This is partly because state of the art matrix factorization methods are already very effective for this problem. The system described in this paper will make use of multi layer perceptron networks to solve the recommendation problem using collaborative filtering. The design and implementation details of the recommendation engine will be discussed and the results compared with traditional matrix factorization methods. Furthermore it will also go into detail about the choices that went into creating a scalable overall system and accompanying web application.

Contents

1. Introduction

Nowadays every customer is faced with a multitude of possible choices. Because of this it is very important that the user doesn't have to waste time manually filtering through the items to find relevant ones. Recommender system overcome this by providing personalized recommendations that suit a particular users tastes. They are present in many different websites such as google, amazon and netflix and help highly with user retention and satisfaction. I will focus on the problem of recommending movies and also building the underlying system and infrastructure required to server these recommendations.

1.1. Dataset

The datesets used as basis for the recommendation engine is the movie lens 20m dataset, put together by the Grouplens research group at the university of Minnesota. Harper and Konstan [1]. This is an explicit feedback dataset, where users manually assigned ratings to movies. The characteristics of this dataset are the following.

- 20000263 ratings on a scale of 0.5-5 in 0.5 increments
- 27278 movies having been rated at least once
- 138493 users that have rated at least 20 movies
- 465564 tags about movies

The dataset is split into different files of which I am only using the ratings, movies and links files. They are all available in comma separated value format. The ratings file comprises of values for userId, movieId, rating, and timestamp.1. Timestamp remains unused while the other values are used together to perform collaborative filtering. The movies file is made up of movieId, title and genre. 2 It is used for incorporating genre data into the model. The links file contains movieId, imdbId and tmdbId. 3 This file provides imdbId and tmdbId used in web-scraping scripts to provide content for the demo web-app.

userId	movieId	rating	timestamp
57772	3861	4.0	982690041
122521	1411	2.0	1001957487
22759	46578	4.0	1216540196
92812	1206	2.5	1118792437

Table 1: Rating file

movieId	title	genres
106749	Mr. Morgan's Last Love (2013)	Comedy|Drama
2	Jumanji (1995)	Adventure|Children|Fantasy
91169	Easier with Practice (2009)	Drama|Romance
106879	Fright Night 2: New Blood (2013)	Horror

Table 2: Movie file

movieId	imdbId	tmdbId
6	0113277	949
206	0114805	77350
71466	1174730	28053
96834	2343601	111174

Table 3: Links file

2. Background

The most common types of recommender systems can be split into collaborative filtering and content based. They can be further split into model based, and memory based.

2.1. Content Based

Content based recommendations require a number of features related to an item instead of historic user-item interactions. In the case of movie recommendations these could be year, actors, genres, producers, writers etc. This method relies on calculating similarity between items using using features such as the ones previously stated. The general idea is that if a user likes a given item he will also like items similar to it. An example of content based recommender system might use pearson or cosine similarity. One way of achieving this is by calculating term frequency (TF) and inverse document frequency (IDF) of the items. Then using the vector space model and a choice of similarity metric such as cosine or pearson, we can compare different items.

2.2. Collaborative Filtering

Collaborative filtering (CF) algorithms aims to recommend items to a user by combining the item interactions of a given user with item interactions of all other users. CF can be split into two categories. User-based where the aim is to measure the similarity of a given user and all other users. Item-based where we aim to measure the similarity between the items a given user has

interacted with and other items. The most widely used method of achieving this is through factorization of the very sparse user-item interaction matrix where more than 99% of the entries are missing.

2.3. Matrix Factorization

The idea behind matrix factorization (MF) is to decompose the matrix R containing user-item interactions, into the product of two lower dimensional matrices P of size n X k and Q of size k X m. Matrix P is the users matrix where n is the number of users and k the number of latent factors. The other matrix is the movie matrix with m number of items and the same k latent factors. The resulting matrices are dense and have much lower dimension than the initial matrix R. By choosing a different number of latent factors we can include more or less abstract information about the initial matrix R. MF poses the recommendation problem as an optimization one. Two common metrics used for this are root mean squared error (RMSE) and mean absolute error (MAE). The RMSE and MAE can be calculated as follows given that, e_i is the difference between actual and predicted value of rating i.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2} \tag{1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid e_i \mid \tag{2}$$

Singular value decomposition (SVD) Non-negative matrix factorization (NMF)

Stochastic gradient descent (SGD) and alternating least square (ALS) are two optimization algorithm generally employed to learn a good approximation.

- Set item factor matrix constant and adjust user factor matrix by taking the cost function derivative.
- Set user factor matrix constant and adjust item factor matrix.
- Repeat until convergence.

2.4. Neural Networks & Deep Learning

Neural networks are universal approximators, typically organized in layers of neurons connected to each other through weights and put through an activation function. The number of layers and neurons at each layer also called the depth and width of the network are variable and many different configurations seem to work in practice. The simplest neural network is made up of 3 layers, input, hidden and output.

They can be used for both supervised and unsupervised learning and can be applied to classification and regression problems just as effectively. Deep neural networks (DNN) are NN of more than 3 layers, although in practice and state of the art implementations these are many layers deep and very wide as well. The main benefit of using NNs algorithms is that they don't require manual feature engineering. A big disadvantage is the lack of interpretability for the predictions, due to this NN are generally viewed as black boxes. This means that we are not aware of the 'why' and 'how' did the network product a

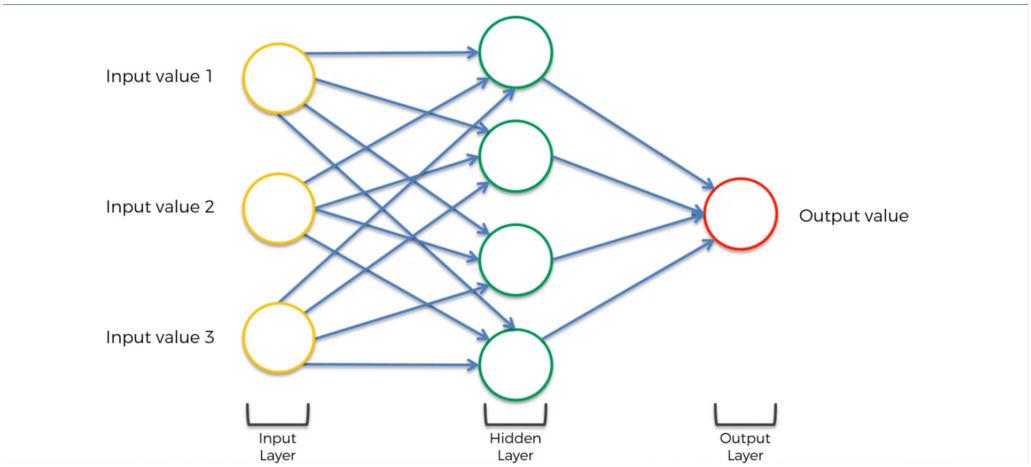


Figure 1: 3 Layer Neural Network

certain output given some inputs. This could be especially detrimental in the context of recommendation systems because the users might wish to know the reasons behind their recommendations. Other memory and similarity based methods provide much more transparency in this regard.

Different classes of NN have emerged in literature including convolutional networks (CNN), autoencoders (AE), recurrent networks (RNN), generative adversarial networks (GAN) in addition of simple fully connected multilayer perceptrons (MLP).

3. Related Work

Strub et al. [2] propose a hybrid recommender system based on auto-encoders

4. System Architecture

4.1. Technologies, tools and frameworks

4.1.1. React

React is a fast, declarative and efficient javascript library for creating web interfaces. It works around the concept of components. They are self-contained and composable blocks of code that encapsulates a part of user interface and its functionality. By putting together multiple small components it's possible to build complex user interfaces (UI). Components can be stateful or stateless. The library provides a virtual-dom similar to the browsers document object model (DOM). They are both node trees that list elements together with attributes and content as objects and properties. Updating the dom is rather slow which is why the virtual-dom is useful for efficiency. It allows react to optimize DOM updates under the hood to only happen when it's necessary.

4.1.2. Docker & Containers

Containers are self contained pieces of code that can be run on any computer and operating system (OS). They contain the code and all of its necessary parts such as libraries, tools and frameworks. They are similar to virtual machines but the main difference is in efficiency and application size. Containers are more efficient and smaller because they run on the same underlying kernel as the operating system as as opposed to virtual machines which runs an entirely different OS.

Docker is tool that allows creating and running containers and managing containers. Docker

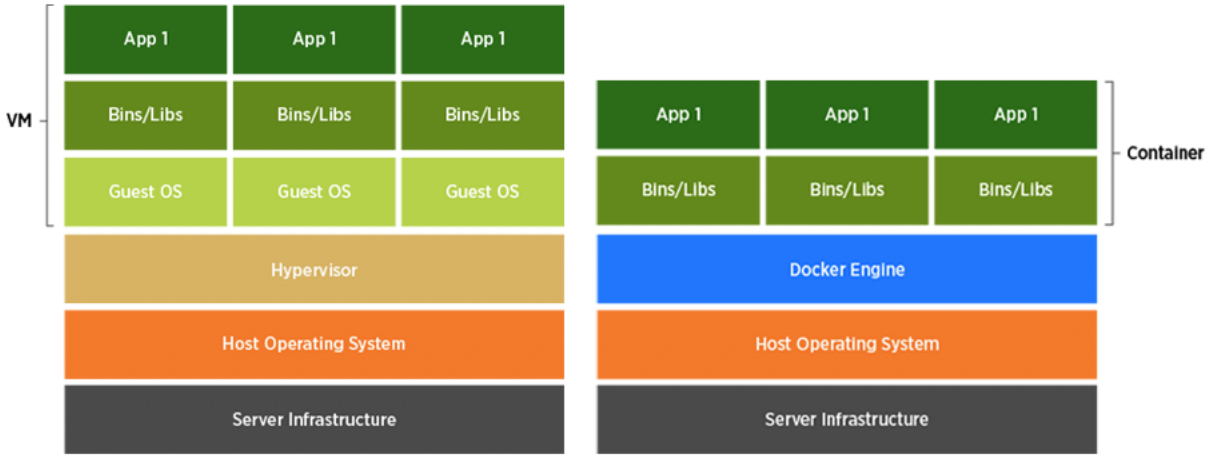


Figure 2: Containers and virtual-machine comparison diagram

4.1.3. Kubernetes

Kubernetes is a container orchestration platform created by google and open sourced in 2014. It allows automation of deployment, scaling and management of containerized applications. It groups the container that make up a multi micro-service application into logical units for easy discovery and management. It's built with scalability in mind

Node Pod Service Gateway Ingress

4.1.4. Mongodb & Mongoose

4.2. Micro-services

Authentication

Gateway

User

Search

Engine

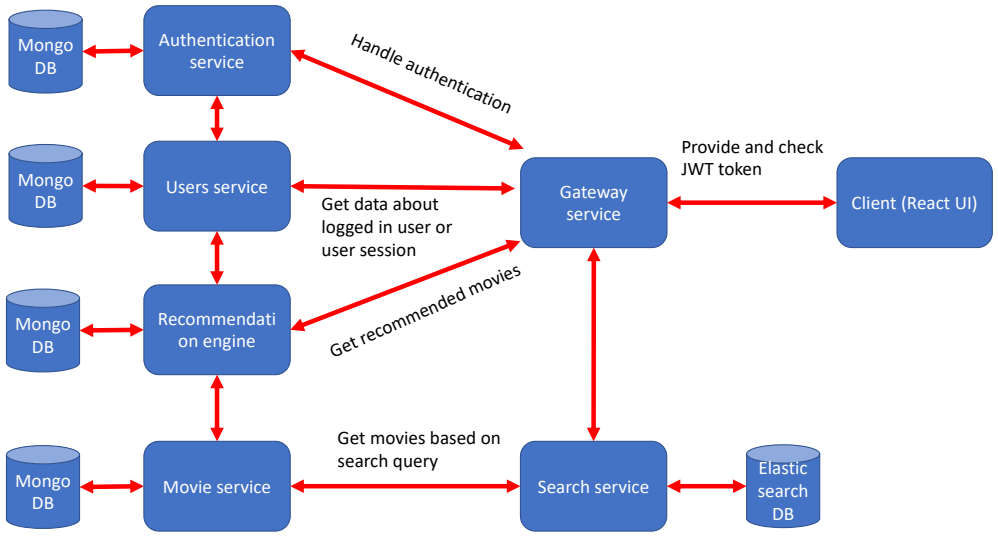


Figure 3: Architecture diagram

4.3. Web application

5. Recommendation Engine

5.1. Embeddings

Sparse to dense representation One hot encoding vs embeddings Word2Vec

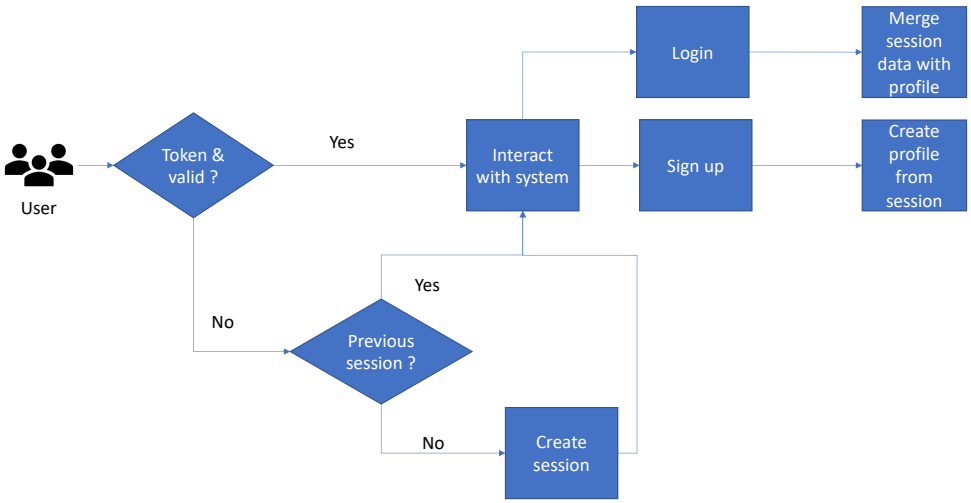


Figure 4: User flow diagram

5.2. Activation function

In a neural network activation functions are mathematical equations applied to each neuron and it determines if it should activate or not based on its inputs. This function must be computationally efficient to calculate and will generally be non-linear. The last part is very important because without non-linearities a NN would just behave like a single-layer perceptron and would not we able to model complex functions. One exception to this is the output layer for a regression NN which will have a linear activation in order to allow the prediction of any real value as.

Early neural networks were using tanh and sigmoid activation functions. Sigmoid also known as logistic function is S shaped and bounded above by 1 and below by 0 Traditional sigmoid 3, tanh 4

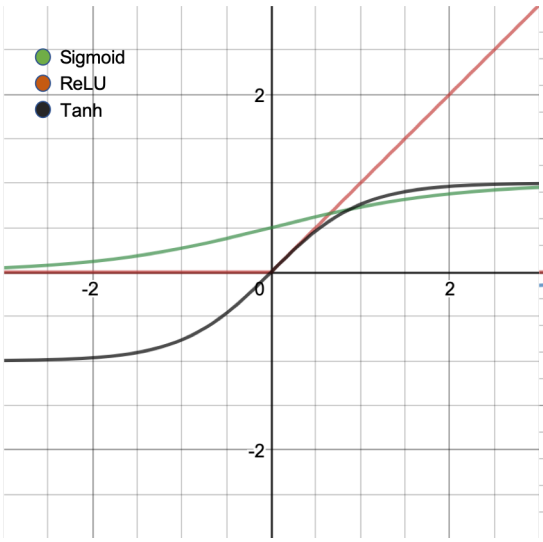
Rectified Linear Unit ReLU 5 A major drawback of using ReLU activations is the "dying ReLU" problem. leaky ReLU 6 comparison with traditional ones comparison between relu and leaky relu

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= f(x)(1-f(x)) \end{aligned} \tag{3}$$

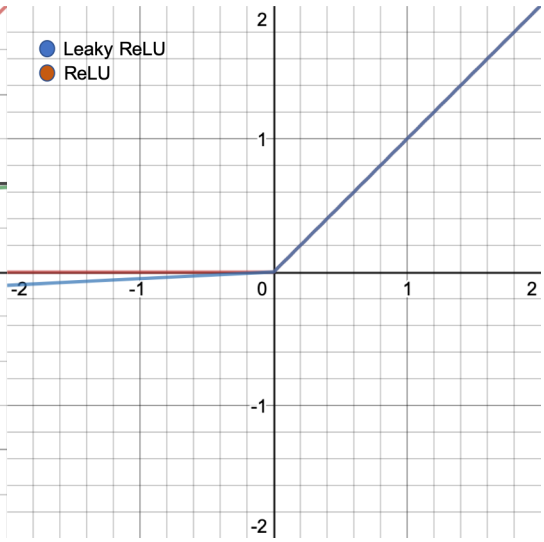
$$\begin{aligned} f(x) &= \frac{2}{1+e^{-2x}} - 1 \\ f'(x) &= 1 - f(x)^2 \end{aligned} \tag{4}$$

$$\begin{aligned} f(x) &= \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \\ f'(x) &= \begin{cases} 1, & \text{if } x \geq 1 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{5}$$

$$\begin{aligned} f(x) &= \begin{cases} x, & \text{if } x \geq 0 \\ 0.01x, & \text{otherwise} \end{cases} \\ f'(x) &= \begin{cases} 1, & \text{if } x \geq 1 \\ 0.01, & \text{otherwise} \end{cases} \end{aligned} \tag{6}$$



(a) Activation functions



(b) ReLU vs Leaky ReLU

5.3. Regularization

Kernel Regularization Activity Regularization Dropout general value 0.5 gives higher RMSE than 0.2

5.4. Optimizer

Optimizers are a very important parameter in NN configuration. Nowadays theres a vast choice of good optimizers. At its core an optimizer is an iterative method of optimizing for a cost function. At each step of the optimization the weights in the NN will be updated based on the negative of the gradient of the cost function. Stochastic gradient descent (SGD) is a variation of GD in that the optimization happens after each training example. In practice this usually implies mini-batches of between 32 and 1024 data points. Batch gradient descent involves updating the weights based on the gradient over the

whole dataset. SGD optimizes based on an approximation of the gradient unlike batch gradient descent. This turns out to be useful as it introduces noise in the network which leads to better generalization. It is also more scalable as the whole dataset does not have to be kept in memory.

$$\theta = \theta - \alpha \Delta_\theta J(\theta; x^{(i)} y^{(i)}) \tag{7}$$

More advanced optimizers are built on top of SGD and include things such as adaptive learning rates and momentum to increase convergence speed and overall stability.

One such algorithm is adaptive moment estimation (Adam). Its widely used in literature and much faster than SGD

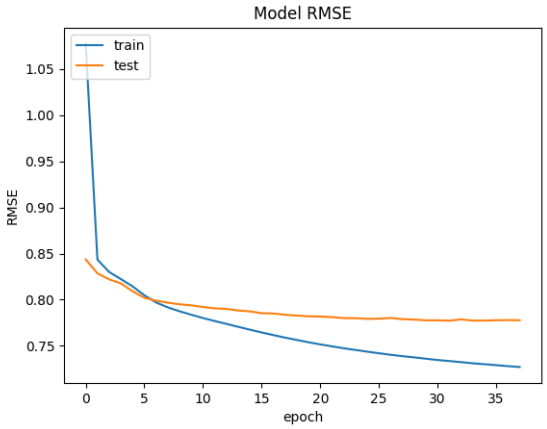
Nesterov accelerated gradient Adaptive Moment Estimation (Adam) Adam can be seen as a combination of RMSProp and momentum. Kingma and Ba [3] Nesterov adaptive moment estimation (NAdam) combines adam with nesterov momentum which improve convergence (cite)

5.5. Serving recommendation & Api

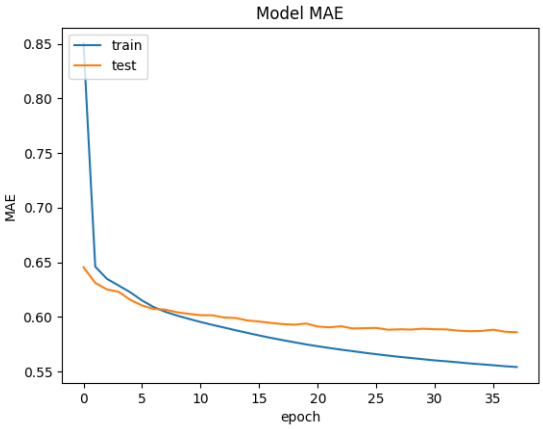
Structure of Api. Create pod that just retrains the model on new ratings. Once finished redeploy pods

6. Evaluation

The evaluation section with results



(a) RMSE plot



(b) MAE plot

Model	RMSE	MAE
SVD	0.8152	0.6190
NMF	0.8246	0.62876
NNCF+genres	0.7763	0.5861

Table 4: Results Comparison (RMSE & MAE)

7. Conclusion

7.1. Future Work

References

[1] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2827872

[2] F. Strub, J. Mary, and R. Gaudel, "Hybrid recommender system based on autoencoders," *CoRR*, vol. abs/1606.07659, 2016. [Online]. Available: http://arxiv.org/abs/1606.07659

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980