COLLABORATIVE TEXT FILTERING THE MEAN SQUARES

*Peter J. Bakke, Thomas Brenner, Daniel Horvath & Christian Hansen**

Technical University of Denmark
DTU Compute
Lyngby

1. INTRODUCTION

Collaborative text filtering is one of the most popular and effective approaches for recommender systems. Recommender systems are based on the idea, that given previously collected data about users and their interactions with items, you can predict whether a given user wants to have an interaction with a given item. This is widely used for platforms like Netflix, Amazon, Youtube and news websites. These platforms can increase their profits by being able to predict their consumers interests and showing content relevant for the user.

The purpose of this project is to match two text descriptions of varied lengths. More concretely job applications with job descriptions. The hope is of course, to have a model that matches job applicants with open job positions.

2. RELATED WORK

This section describes the theory behind the project. Also it gives an overview of some methods that are used in our approach.

2.1. Collaborative Filtering

Collaborative filtering is a method used by recommender systems to make predictions about the interest of a given user, by looking at the interest of all (or a subset) users in the system [1, 2]. Naively, if a person *X* and person *Y* are interested in the same item, then they are also more likely to have the same interest in other items. For collaborative filtering, you construct a matrix $R \in \mathbb{R}^{m*n}$ where m is the number of users in the system, and n is the number of items in the system. Every entry r_{ui} is defined as:

$$r_{ui} = \begin{cases} 1, & \text{if user } u \text{ has interaction with item } i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Here if an entry in the matrix has the value 1, it means that there has been an interaction between a user and an item. If an entry has the value 0, than there is no recorded interaction between the user and the item. However it is important to note, that just because a user has interacted with an item, it does not mean that the user likes the item, and similarly just because a user has no interaction with an item, does not mean the user does not like the item [1, 3].

2.1.1. Matrix Factorization

Due to the *Netflix Prize data* competition on website Kaggle[1], Matrix Factorization became a very popular approach to collaborative filtering.

Matrix factorization works by projecting users and items into a shared latent space using a vector of latent features to represent a user or an item. The interaction between a user and an item is then modelled as the inner product of their latent vectors. Although matrix factorization is an effective way of doing collaborative filtering it may fall short in terms of effectiveness due to the simplistic interaction function (the inner product) between the latent factors [1, 2].

One thing to notice is that there are two different types of feedback that can be used as interaction. It can be distinguished between implicit and explicit feedback. Implicit feedback is the classic user interaction. The interaction of a user and an item means not that the user likes or dislikes the item. An example therefore is a user that watches a video or buys a product. Explicit feedback on the other hand tells more precisely whether or not the user actually likes the item, typically in the form of a rating, i.e. Netflixs rating system [4].

2.2. Cold-Start Problem

As previously mentioned collaborative filtering is commonly used for recommender systems, largely due to the efficiency and accuracy. It does however suffer from one problem, also known as the *Cold-Start Problem*. Because recommender systems rely on interactions between items and users, when

*Thanks to our supervisor Alexander.

[1] https://www.kaggle.com/netflix-inc/netflix-prize-data, visited: 2018-10-23

a new item is introduced into the system, the prediction will be off as there is no user interaction with the new item at all [5].

2.3. Word embedding

Word embeddings represents text by numerical values, and perhaps there are different numerical representations for the same text. The reason for using word embeddings is because most deep learning models do not support text or strings [6].

2.3.1. One-hot encoding

One-hot encoding is one method for creating a word embedding. A vector is created with the same length as the used vocabulary. So if the vocabulary consists of 8000 words, then the vector will have size 8000. The vector will have 0 at every position, with the exception of the word currently being looked at. If the word being looked at is *Model* and *Model* has the 6023th position in the vocabulary, then the 6023th position will have a 1 in the vector [7]. This method is also known as localist representation. However one-hot encoding is rarely used because the similarity between words can not be represented. To be more precise there is no way to show the relationship between words.

2.3.2. Word2vec

Word2vec is one model used to create word embeddings. As mentioned before the simplest method is one-hot encoding, but it is rarely used. The Word2vec model uses the idea of generating a method that allows similarity measure such that words that are closely related, have values closer to each other. This allows that a program can find similar words quickly. I.e. the words *dog* and *cat* are closely related, because they are both animals and pets. In one-hot encoding there is no such way to tell whether two words are closely related [8, 9].

3. METHODOLOGY

This section describes the methodology used for this project.

Collaborative Filtering is about matching different entities together. In the first part of the project we will use a public dataset called MovieLens.[2] The MoveieLens dataset consist of 20 million ratings on a scale from 1 to 5, of 27,000 different movies by 138,000 users. Taking outset in the MovieLens dataset the objective is to predict how a specific user will rate a specific movie.

A small subset of the ratings data can be seen in table 1. Here we can see that e.g. the MovieId 4223 (Enemy at the Gates, a World War 1 movie from 2001) has been rated 4.0 by UserId 28 whereas UserId 7 has rated the same movie 2.0. UserId 7 has instead given MovieId 4018 (What Women Want, a comedy/romance from 2000) a rating of 4.0. This indicates that UserId 7 prefers comedy/romance to historical war movies.

Table 1. Five rows from the MovieLens ratings data

UserId	MovieId	Rating	Timestamp
1	151	5.0	964984041
6	151	4.0	845553586
7	4018	4.0	1106712913
7	4223	2.0	1106635742
28	4223	4.0	1242032494

It is worth noting that collaborative filtering could be used to predict other matches as well, e.g. the match between a customer on a web page and whether he is likely to interact with a given recommended product. By predicting what products a given visitor is most likely to interact with the owner of web page can maximize his earnings by showing products most relevant to the visitor.

3.1. General Methodology

In this section we will provide an overview of the general methodology used troughout our analysis.

In general we try maximize the probability of a match between a user and an item given some features:

$$\max p(m|K;\theta) \tag{2}$$

where m denotes the binary variable on whether there is match, K denotes the features, and θ denotes the parameters in the model. In the MovieLens case we could have that K corresponds to the different users (identified by a UserId) and their respective ratings of a specific movie (denoted by MovieId), i.e.:

$$K = \text{list}(\text{UserId}, \text{MovieId}) \tag{3}$$

It is worth noting that if a user has *not* rated or seen a movie it doesn't mean that the user dislikes that particular movie – it could also mean that the user is not aware of the existence of the movie in question.

To predict a users preferences we use

$$p(m) = \sigma(f(\text{UserId}, \text{MovieId})) \tag{4}$$

where $\sigma(\cdot)$ denotes the sigmoid function which has a nice probabilistic interpretation, and $f(\cdot)$ is some function of user and movies. In the most simple case $f(\cdot)$ could be given by the inner product of embeddings of users and movies, e.g.:

$$f(\cdot) = u \cdot m^T \tag{5}$$

[2] https://grouplens.org/datasets/movielens/, visited: 2018-11-01

Here u is an embedding of our user vector and m is an embedding of our movie vector, i.e.

$$u = \text{Embedding}(x_u) \tag{6}$$
$$m = \text{Embedding}(x_m) \tag{7}$$

To build a more advanced method we could use a neural network instead of the inner product, and the model would then be represented by:

$$p(m) = \sigma(NN(\text{u, m})) \tag{8}$$

Where NN denotes a Neural Net.

In order to train our model we need to define a loss function, L:

$$L(m, \hat{m}) \tag{9}$$

Where $\hat{m}$ is the match probability predicted by the model. Based on the magnitude of $\hat{m}$ we can say that the model has predicted a match, e.g. we can classify whether there is a match if the magnitude of $\hat{m}$ is greater than say 0.5.

For our model we define our loss function to be the binary cross entropy loss function, where each loss contribution is given by [10]:

$$L(m, \hat{m}) = -m \log \hat{m} - (1 - m) \log(1 - \hat{m}) \tag{10}$$

To train our model we use an optimizer such as Stochastic Gradient Descent or Adam to search the parameter space for the set of parameters that minimizes the loss function, i.e.:

$$\underset{\theta}{\text{argmin}} L(m, \hat{m}) \tag{11}$$

3.2. Basic Neural Network with numeric features

The basic approach for predicting user ratings is Matrix Factorization. We can extend this idea and change the mapping function $f(\cdot)$. Instead of calculating the dot-product between the features a Neural Network can be used to learn any non-linear function that represents the interaction between a user and a movie.

For creating the NN we need the embedding vectors u and m, which are numerical representations of the features *userId* and *movieId*. These vectors are concatenated and used as input for the NN.

The starting point is a fully connected NN with one hidden layer and ten units. ReLU is used as activation function. The output layer has one unit and represents the predicted user-movie rating. To scale the predicted rating between one (r_{min}) and five (r_{max}) like in the *MovieLens* dataset, following formulation can be used:

$$\hat{r} = \sigma(outputNN) * (r_{\max} - r_{\min}) + r_{\min} \tag{12}$$

During the training process the weights and biases of the NN are optimized regarding the Mean Squared Error(MSE) with Stochastic Gradient Descent(SGD). Following investigations can be done to explore the influence of different changes to the overall prediction:

- change dimensions of word embeddings
- usage of different activation functions
- use Adam Optimizer with momentum
- introducing dropouts
- adding more hidden layers

When implementing more hidden layer the network can be represented as a function of nested functions. The output depends on different linear layers and non-linear activation functions after each layer.This looks like following conceptual structure:

$$\hat{r} = f_X(...f_2(f_1(\text{u, m}))) \tag{13}$$

with f_1 as output of the first layer

$$f_1 = \sigma(u, m) \tag{14}$$

and f_{2-X} for all other layers

$$f_{2-X} = \sigma(W_x^T * f_{x-1} + b_x) \tag{15}$$

where X corresponds to the amount of layers, W and b are the weights and biases of the NN. σ corresponds to the ReLU activation function except to the last layer where a sigmoid function is used. Each hidden layer learns a certain latent structure of the usermovie interaction. The training and optimization follows the same procedure like with one hidden layer.

3.3. CF extended with movie description

With the previous model we achieve a classical implementation of a purely collaborative filtering model. This model exploits the relationship between the movies and users, and embed both of them based on this relationship.

However, when more data is available for the movies, then those data can be used to increase the accuracy of the model, and might increase performance in the case of the cold-start problem as well.

The users u and movies m are again represented by one-hot vectors. These vectors are then embedded by the embedding matrix P and Q.

$$u = P * x_u \tag{16}$$
$$m = Q * x_m \tag{17}$$

Suppose that we have a description for every movie, containing words $w_1, .., w_j$. This description can be embedded either

as bag-of-words or as a paragraph vector using a pretrained embedding matrix. [11, 12].

The embedding of each movie's description is concatenated to the user and movie embeddings into one single vector.

$$s = [u, m, d] \tag{18}$$

The vector s is then fed into the first fully connected layer, a ReLu activation function and then to a second fully connected layer. The output is created with a sigmoid function that is scaled to provide a value between $1..5$.

$$\begin{aligned} &s \rightarrow LinLayer_1 \rightarrow ReLu \rightarrow LinLayer_2 \\ &\rightarrow \sigma() \cdot 4 + 1 \rightarrow \hat{r} \end{aligned} \tag{19}$$

The loss is calculated with the cross entropy loss function.

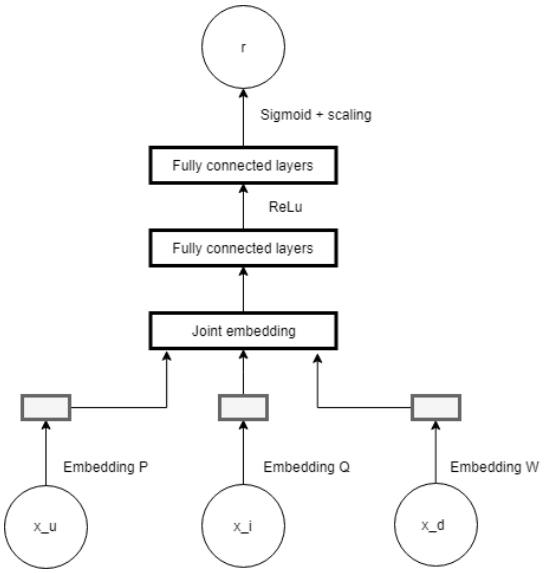

Fig. 1. The model extended with the description embeddings

4. EXPERIMENTAL SETUP

This section will describe our proposed setup.

4.1. Baseline model

As a baseline we will create a matrix factorization model of our data. Another baseline would be the ItemPOP, which ranks items based on the amount of times they have been interacted by users. [13]

4.2. Dataset

The model will be tested on two datasets, the MovieLens dataset (as described in Section 3 and Table 1) [3] and a dataset not yet acquired that will contain job descriptions and job applications.

4.3. Architecture

Figure 1 shows the proposed architecture of the model. The idea is to take multiple embeddings and joining them together, then feed them into multiple fully connected layers in a neural network, before applying the sigmoid and calculating the loss.

4.4. Implementation

The implementation will utilize some (but not limited to) of the following technologies:

- *Python 3.x*
  - *PyTorch* [4]
  - *torchtext* [5]
  - *NumPy* [6]
- *Google Colab*
  - Virtual machines with GPUs.
- *git* - https://github.com/PeterJBakke/TheMeanSquares

5. REFERENCES

[1] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua, "Neural collaborative filtering," *CoRR*, vol. abs/1708.05031, 2017.

[2] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th International Conference on World Wide Web*, New York, NY, USA, 2001, WWW '01, pp. 285–295, ACM.

[3] Guangneng Hu, Yu Zhang, and Qiang Yang, "LCMR: local and centralized memories for collaborative filtering with unstructured text," *CoRR*, vol. abs/1804.06201, 2018.

[4] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua, "Fast matrix factorization for online recommendation with implicit feedback," *CoRR*, vol. abs/1708.05024, 2017.

[5] Jianbo Yuan, Walid Shalaby, Mohammed Korayem, David Lin, Khalifeh AlJadda, and Jiebo Luo, "Solving cold-start problem in large-scale recommendation engines: A deep learning approach," *CoRR*, vol. abs/1611.05480, 2016.

[3] https://grouplens.org/datasets/movielens/, visited: 2018-11-01

[4] https://pytorch.org/, visited: 2018-11-01

[5] https://torchtext.readthedocs.io/en/latest/, visited: 2018-11-01

[6] http://www.numpy.org/, visited: 2018-11-01

[6] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2016, KDD '16, pp. 353–362, ACM.

[7] "Word embeddings: Encoding lexical semantics," https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html, Accessed: 2018-10-31.

[8] "word2vec," http://nbviewer.jupyter.org/github/danielfrg/word2vec/blob/master/examples/word2vec.ipynb, Accessed: 2018-10-31.

[9] Tom Kenter, Alexey Borisov, and Maarten de Rijke, "Siamese cbow: Optimizing word embeddings for sentence representations," *arXiv preprint arXiv:1606.04640*, 2016.

[10] Michael A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[11] Felix Hill, Kyunghyun Cho, and Anna Korhonen, "Learning distributed representations of sentences from unlabelled data," *arXiv preprint arXiv:1602.03483*, 2016.

[12] Quoc Le and Tomas Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

[13] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, Arlington, Virginia, United States, 2009, UAI '09, pp. 452–461, AUAI Press.