NIRS: Neural-based Recommender System with a focus on Interpretability (DAT410, assignment 8)

Luca Modica, Hugo Manuel Alves Henriques e Silva
Chalmers University of Technology, Göteborg
{modica, hugoalv}@chalmers.se

1 Introduction

Recommender systems represent one of the main branches of research applied to companies. They are faced with various challenges to make the best recommendations for any user: we need to take into account suitable representations of them and the items themselves given the available data, with the goal to let the model learn the user-item interaction in a proper and meaningful way. But with good data representation It is highly desirable to know the reason behind why a certain product is recommended to a certain user, given both of their characteristics. This also for transparency of the system and to have useful insight to improve the model even further.

To face this challenges, in this work we present NIRS (Neural Interpretable Recommender System): a recommender system model based on the original neural collaborative filtering, with enhanced features to try to mitigate overfitting and model different types of data. Our method aims to be as interpretable as possible: after a prior data analysis and after generating model recommendations for a sample of users, we will investigate why a recommendation is made for a specific user. Some of the techniques we are going to use are the following:

- **Users and products representation**;
- **Word embeddings and their analysis**;
- **Neural Collaborative Filtering Architecture**;
- **Interpretability of a recommendation**.

In the first part of the report, we will describe our data and all the pre-processing steps we applied to perform our experiments on. To present a scenario as close to the real-world as possible, the dataset chosen for our experiments is "Amazon Review Data (2018)" dataset, which will be presented more in detail in the related section.

Secondly, we will go into detail about how we decided to represent a user and a product and how we intend to extract the features that will be used later on in the model. We will then go into technical details about the architecture of our model and how it accomplishes the recommendation, alongside all the experimented techniques to improve the prediction performance even further.

The fourth part will be dedicated to the numerical results and performance achieved when testing our method.

Next, we will consider the possible limitations of our methodologies for recommending a product to a user, such as the cold-start approach and scalability.

Lastly, we conclude with our ethical considerations about using such a system in the realm of user recommendations.

2 Data

As mentioned above, for our recommender system we worked with the famous Amazon Review dataset (2018) as use case: It includes both variegated user reviews information (ratings, text, helpfulness votes) and products metadata (from descriptions to a dictionary of product features), making it perfect for experimentation with models to generate and justify recommendations (Ni et al., 2019). In particular, in our work we choose a specific subset of this large dataset.

- We chose a k-core dense subset of reviews, such that each of the users and items have k reviews each.
- We limit the investigation to Office product category, which still represents a product section that can be meaningful repreent variegated users' tastes.

For our task, we want to find meaningful representations of both users and products. Therefore, a lot of meaningful data pre-processing steps were made, as It will be described below. Note, that, for the sake of the experimentation, not all of the features from both the reviews and products metadata will be used. Also, for a better visualization, the data will be represented in `json` as in the original file format.

2.1 Review Data

The format of each review, with the features considered, is the following.

```
{
    "overall": 5.0,
    "reviewTime": "01 1, 2018",
    "reviewerID": "AUI6WTTT0QZYS",
    "asin": "5120053084",
    "reviewerName": "Abbey",
    "reviewText": "I now have 4 of \
        the 5 available colors of \
        this shirt ... ",
    "summary": "Comfy, \
        flattering, discreet--highly \
        recommended!",
    "unixReviewTime": 1514764800
}
```

Where:

- `reviewerID`: user id
- `reviewerName`: reviewer username
- `asin`: product id
- `overall`: the numerical rating (a number form 1 to 5) of the user to the product
- `summary`: summary of the review
- `reviewText`: the text of the review written by the user
- `reviewTime` and `unixReviewTime`: time of the review, both in string and unix timestamp format.

Alongside the features described, we create a new feature that represents the difference between the review date and the most recent review's date. Also, empty review dates are assigned to the oldest date found within all reviews.

2.2 Product Data

The data format of the product metadata is shown below.

```
{
  "asin": "0000031852",
  "title": "Girls Ballet Tutu \
    Zebra Hot Pink",
  "feature": [
    "Fits girls up to a size 4T",
    "Hand wash / Line Dry",
    ],
  "description": "This tutu is great \
    for dress up play for your \
    little ballerina.",
  "salesRank": {"Toys \& Games": 211836},
  "brand": "Coxlures",
  "categories": ["Sports \& Outdoors"]
}
```

On the products side, we have a wide variety of features. From its `description`, `title` and `features` such as size or materials, to its `brand` and the date they were published. Furthermore, despite all products being office ones, some of them have other categories as their main one.

Firstly, we will filter out products without descriptions, titles or features. In case they do not contain a published date, we assign that feature to the oldest date found. Moroever, we create a new feature that represents a value from 0 to 1 of how old a product is compared to the newest one. For products without a brand, we assign that brand to a default value of ”Unknown”. Lastly, we one-hot encode the main category feature.

3 Methodologies

In this section, we will describe our approach to represent users and products for this particular task, alongside the architecture used for the model used to make recommendations.

3.1 Data representation

Even though we are using a subset of the Amazon Review dataset, the data are considerably high-dimensional, very sparse and especially not numeric: this may represent a real challenge for building a recommender system, from the expressiveness of the data from which it is learning, to the computational power needed. To address these issues, our approach was to use embeddings: they allow us to represent high-dimensional data in a lower-dimensional space and more dense vectors. These properties will allow the representation of the feature in a numerical and reasonable way for the model, letting it learn the user-item interaction and, for example, the semantic meanings of textual information.

Most of the features we have available in the datasets are textual: therefore, in order to have some numerical representation for them, we rely on text embedding algorithms. The model we experimented with was **Word2Vec**: It is a popular language model used to generate text data representation, by capturing semantic meaning and relationships between words; this, by using both Continuous Bag of Words (CBOW) approach (optimize for predicting the target word in its contexts words) or Skip-gram approach (optimize for predicting context words given a single one). Attempts to use pre-trained models, such as **Distilled-BERT**, which is a simplified version of the pre-trained model BERT that can create text embeddings by extracting meaningful features from the text data with attention mechanisms were made. However, due to time restrictions and computational burden, we had to rely on a simpler model such as Word2Vec for feature extraction. On the other hand, for interoperability measurements, we still used the results obtained from Distilled-BERT

It is relevant to have a user and a product representation when considering a recommender system. In order to make an accurate recommendation, we need to know exactly how a user is described, and how every product is described and assess if the representations make a good match. Below we will describe the representation of the user in terms of the embeddings.

3.1.1 User Representation

Users are firstly represented by their identification number. In addition, as the goal is to find the best user representation possible, we take all of a user's reviews and their respective summaries and create text embeddings for them, using the previously referred algorithms. For each review, we will average the review text embedding with the summary embedding. Then, to obtain a final representation of all of the history given by the reviews of a user, we average all of a user's reviews' embeddings.

3.1.2 Product Representation

Products also have their first representation with their identification number. In the same way as the reviews, text embeddings are created for the products' textual features, which are: Description, Features, Brand and Title. For every product, we average the tensors obtained from the embeddings of the previous features to obtain a single representation.

3.2 Model architecture

In our work, to best capture relationships with complex data such as the embeddings previously described, we employ a modified version of Neural Collaborative Filtering (NCF). Based on the architecture described in the original paper, it represents a state-of-the-art approach in the research branch of recommender systems: its main key strength is to combine the advantages of traditional collaborative filtering approaches (for example SVD) with the ones from the expressiveness of a neural network (that is, learning from arbitrary data), with the goal to generate much more accurate recommendation results (He et al., 2017).

A Neural Collaborative Filtering is able to achieve high accuracy and level of data expressiveness with 2 main components.

- Generalized Matrix Factorization (GMF): similar to the traditional Matrix Factorization technique in Collaborative Filtering, this part is responsible for generating the users' and items' latent factors and capturing the linear relationships between them, by doing element-wise multiplication of their embeddings.

- Multi-Layer Perceptron (MLP): by using the same starting embeddings, this component will capture instead the non-linear interaction between users and items. This is done thanks to a fully connected (FC) neural network, which will learn more complex and abstract representations than matrix factorization. In our experimentation, alongside the user and the items vector also their related text embeddings are part of the input of the FC component: this will allow us to leverage textual information in the dataset, capture semantic similarities and have performances that go beyond the explicit user-item interactions. To be able to use the text embeddings, a linear layer is applied to each of them: this is done to bring the embeddings to the same latent space of the user and item ones.

The resulting vectors from both parts (GMF and MLP) are then concatenated to what in literature is called Neural Matrix Factorization (NeuMF), which in our case represents a single output linear layer. An overview of the baseline architecture of the NCF is shown below.

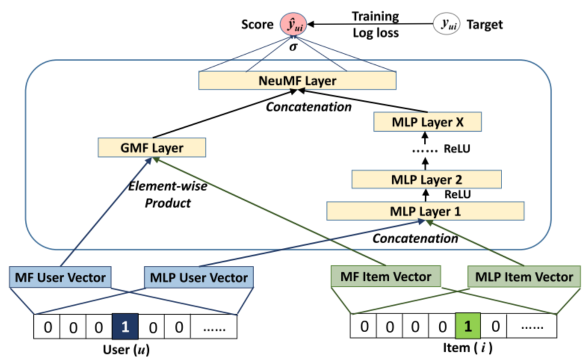


Figure 1: Neural Collaborative Filtering (NCF) architecture, proposed in the original paper.

Compared to the architecture proposed by Xiangnan He et al., our enhancements focus especially on the MLP network.

- We adopted a 5 hidden layer tower architecture, where the number of units is halved by passing each layer by passing from the input to the output. The layers, in order, have the following size: $1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64$; they were chosen with the goal of capturing as many patterns as possible from high dimensional data, such as the dataset used

- With the exception of the last layer of the MLP network component, for each layer a Dropout and a Batch Normalization one are applied. The purposes are, respectively, to handle overfitting situations and help in the algorithm convergence.

- The output of the last NeuMF layer is normalized in the range $[1, 5]$, for the a more accurate a consistent output of the regressor.

- Lastly, all the embedding layers (for the embeddings used both for the GMF and the MLP part) are initialized with a normal distribution. The decision came from a heuristic observation of the model performance.

4 Experiments

In this section, we will present the first results of our approach, based the described data and the presented approach. To train the model, the following parameters have been used:

- number of epochs

- GMF embedding dimensions: 300

- MLP embedding dimensions: 256

- text embedding dimensions: 256

- optimizer: Adam

- learning rate: 0.001

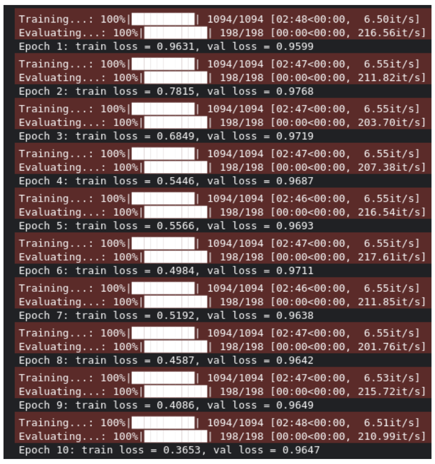
- loss function: RMSE

- dropout rate: 0.2.

Note that, inside the training session, a learning rate scheduler (`ReduceLROnPlateau`) is used to adapt the learning rate based on the validation loss values, through the epochs. Lastly, the test set used follows the leave-one-out strategy: for each user, we remove the most recent review from the dataset to construct the test one.

4.1 Results

As can be seen by the images below, the model's learning, despite showing decreasing values for training loss, the validation score tends to stay constant throughout the whole learning training phase. The results are based on using the user and item latent and the user/item text embedings



```
Training...: 100%|██████████| 1094/1094 [02:48<00:00,  6.50it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 216.56it/s]
Epoch 1: train loss = 0.9631, val loss = 0.9599
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 211.82it/s]
Epoch 2: train loss = 0.7815, val loss = 0.9768
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 203.70it/s]
Epoch 3: train loss = 0.6849, val loss = 0.9719
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 207.38it/s]
Epoch 4: train loss = 0.5446, val loss = 0.9687
Training...: 100%|██████████| 1094/1094 [02:46<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 216.54it/s]
Epoch 5: train loss = 0.5566, val loss = 0.9693
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 217.61it/s]
Epoch 6: train loss = 0.4984, val loss = 0.9711
Training...: 100%|██████████| 1094/1094 [02:46<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 211.85it/s]
Epoch 7: train loss = 0.5192, val loss = 0.9638
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.55it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 201.76it/s]
Epoch 8: train loss = 0.4587, val loss = 0.9642
Training...: 100%|██████████| 1094/1094 [02:47<00:00,  6.53it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 215.72it/s]
Epoch 9: train loss = 0.4086, val loss = 0.9649
Training...: 100%|██████████| 1094/1094 [02:48<00:00,  6.51it/s]
Evaluating...: 100%|██████████| 198/198 [00:00<00:00, 210.99it/s]
Epoch 10: train loss = 0.3653, val loss = 0.9647
```

Figure 2: Neural Collaborative Filtering (NCF) architecture, proposed in the original paper.

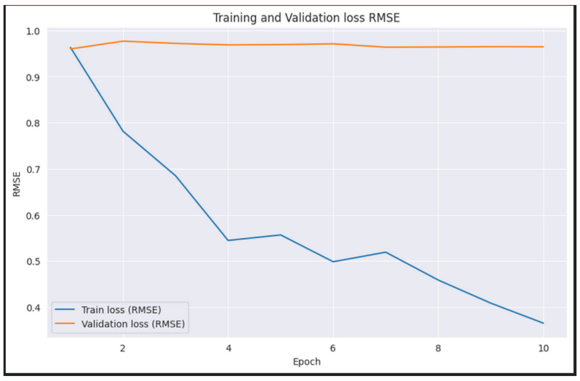


Figure 3:

One possible reason for this unaltered validation loss could lie in the user representation. To represent a user we used his identification number alongside an average of all his review history. The issue with averaging all of the review history of a user is that we fail to take into account each review's aspects, which can be: The recency of the review, the rating and the length of the text. Without taking other properties into consideration, and simply averaging one's reviews, we will have similar representations for different users, even when users may differ a lot from each other. Therefore, the model may fail to learn the different patterns for the users, due to all representations being similar. This is one crucial improvement to be made in future work.

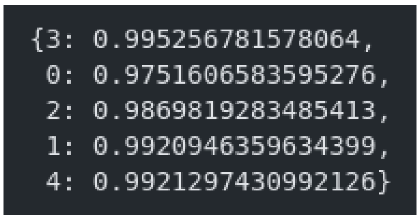
4.2 Interpretability

Many experiments were conducted in terms of interpretability. With the model's input features being a user and a product's representations, we tried to find a way of showing similarities between different users and different products.

In terms of metrics, we decided to utilize the cosine similarity between users' representations, which consisted of the text embeddings of all their reviews and the respective summaries. Users with similar representations will display a cosine similarity close to 1, whereas users whose representations differ significantly we display values closer to 0. Furthermore, we also relied on clustering algorithms to corroborate the previous metric. We separated users into 5 clusters and calculated cosine similarities between 2 users of each cluster and, the obtained results did demonstrate high values for this metric.

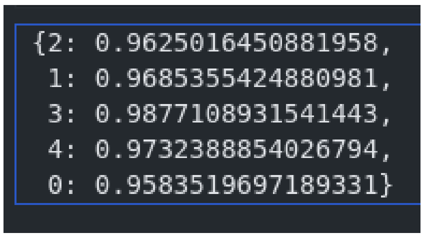
The same exact method was applied to products and their representations, where cosine similarities were calculated for all products and clusters were also built.

These metrics help us understand which users and products are similar, something that is undoubtedly useful when making recommendations. Naturally, similar users will rate the same product in a similar way.



```
{3: 0.995256781578064,
 0: 0.9751606583595276,
 2: 0.9869819283485413,
 1: 0.9920946359634399,
 4: 0.9921297430992126}
```

Figure 4: Cosine similarity between 2 instances of users of each cluster



```
{2: 0.9625016450881958,
 1: 0.9685355424880981,
 3: 0.9877108931541443,
 4: 0.9732388854026794,
 0: 0.9583519697189331}
```

Figure 5: Cosine similarity between 2 instances of products of each cluster

5 Limitations

While the proposed recommender system showed a potential solution to have accurate prediction and a form of explanation of such product recommendations (even for large-scale datasets), our approach can still have limitations. Down below we will now describe the main ones.

- **Data sparsity**. In a relatively large dataset such as the one used for our experimentation, there is a tendency to have a really high data sparsity: that is, many users, interacted with a small subset of items. Even though architectures such as NCF can learn sophisticated patterns, they could still struggle to provide reliable recommendations with datasets that contain limited interactions.
- **Cold-Start Problem**. Since our model learning is mainly based on the user-item interaction, it may have issues in recommending items to a user that has no interaction history. In our approach, even though we tried to alleviate this problem by adding text embeddings to the model learning, performances can still be limited for new users or new items.
- **Interpretability limitations**: by incorporating the text embeddings and doing a comprehensive analysis with a cosine similarity measurement, we were able to give a form of explanations of the recommendations from a such sophisticated model. However, for a system based on a neural network architecture, the interpretability can still be limited. The model's inner workings, instead of specific influence from specific text embeddings, can represent a challenge in terms of explainable predictions.

6 Conclusion

In this work, we presented NIRS, an enhanced Neural Collaborative Filtering with text embeddings for more accurate and explainable recommendations. To have a scenario that is as close to a real-world deployment as possible, we decided to analyze and use the Amazon Review Dataset to show not only potential good results in terms of performance and interpretability, but also its limitations.

Related to the interpretability we tried to achieve during the development of the system, one of our main motivations behind this topic is the ethics of using such a system. Different ethical issues may arise by using a recommender of this kind, as it will be described below.

- An NCF model can have **bias and fairness issues**, especially in datasets such as the used which may contain information such as user demographics, item categories or also sentiment in the review text.
- Manipulating user and item textual embeddings may raise privacy concerns, since the model has access to content written by the users themselves. It's crucial in this case to have a good data protection regulation and give the users the decision to allow their own data to be manipulated, in respect of their own privacy.

A system of such kind can have a wide area of improvement. As for future directions of our work, we are planning on improving the accuracy by handling a large data sparsity and the cold start problem, alongside more advanced metric measurements such as HitRate@K or Precision@K. We also want to try to keep our focus on the interpretability enhancements: we can use more advanced text embedding techniques, from BERT to GPT, which can capture even more meaningful information with attention mechanism; this alongside post-hoc analysis, with LIME and SHAP frameworks.

References

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering.

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. pages 188–197.