Myles Tyberg

Machine Learning Final report

Recommender Systems overview

There are multiple ways of building a recommender system and for that reason each path needs to be analyzed depending on the dataset chosen.

Recommender Systems can be split up into many categories but for this project I will look two popular approaches:

- *Content based filtering* - This uses similarity between items which users pick to recommend new items. This approach does not need user preferences such as ratings or reviews. Amazon uses this approach to recommend new items based on browser history.
- *Collaborative filtering* - This uses a user's preference such as a rating of a movie and chosen embeddings to find the hidden relationship for their choice. Netflix is an example of this. If you like a certain movie and someone with similar preferences wants a recommendation, Netflix can recommend a movie using your preferences.
    - *Collaborative filtering also has two sub categories*:
        - *User based* - This uses features based on user attributes i.e some user has a similar demographic and they rate a movie as 5, you are also more likely to rate that movie the same.
        - *Item based* - This uses item metadata as the latent vectors i.e you rate comedy movies generally higher, you are more likely to want to watch another comedy.

Embedding

Both these approaches are similar in that they use embedding space, this is a system which creates a vector space for entities in a dataset. We need these vectors for these entities as they give a way to map discrete features to a sequence of continuous numbers. With this transformation the vectors can be used to create relationships between each other by using a model to learn them.

Similarity measures

In order for the model to learn the relationships between the embeddings we need a way to calculate how similar they are. We can do this using a few different ways:

- *Cosine similarity* - This takes the pairs factor and measures how similar they are based on the cosine of the angle between the vectors (embeddings) in a higher dimensional space. These embeddings will be plotted on a multidimensional space and each dimension will represent an embedding, we can then use the cosine similarity to find embeddings with similar orientations. This type of similarity has an advantage over other

measurements such as the euclidean distance. As when using the measurement we don't base it on distance but rather the angle, this means even if they are far in distance, they can still be similar in their orientation. In terms of our data for movies this basically means we are more likely not to focus on recommending the most popular movies but this can cause problems as we might recommend unpopular movies.

- *Euclidean distance (L2 Norm)* - This takes a pair of embeddings and measures the distance between them. This is a good measurement as if we don't want popular movies to always be recommended. It does this by not placing too much weight on the magnitude of the vector (norm).

- *Dot Product* - This measurement would be great if we always wanted to recommend the most popular movies on netflix as it places a heavy emphasis on the magnitude of the vector.

Matrix Factorization (Collaborative filtering)
We have now spoken about the different components which go into making a simple embedding model, Matrix Factorization. Matrix factorization is classified as a collaborative filtering model as it uses user ratings to recommend new movies. In our case we will be embedding user ID and move ID's to learn the hidden relationship between these features. To do this we need some type of label which tells us if we are learning the relationship between latent vectors. This will be the rating for a movie from users. We use one of the similarity measures above to calculate the relationship between our embeddings and use some type of minimization function to reduce the difference. This will create a model which will be able to recommend movies once trained.

- *Advantages:*
    - *Domain Knowledge* - Content based filtering relies on domain knowledge whether from user or metadata of content, collaborative filtering does not need this.
    - *Hidden relationships* - The model is able to find hidden relationships and thus discover new interests.

- *Disadvantages:*
    - *Cold start problem* - This model will have a problem with making a new recommendation when it hasn't seen the user/item in its training dataset.
    - *Performance* - The performance of the model decreases as data becomes more sparse, this means that this type of model cannot be scaled to be used on larger datasets.

Vector space model (Content based filtering)
Similarly to the collaborative filtering model we create embedded vectors, these are based on either user data or metadata from a movie in the case of netflix. To create the model we take these vectors and calculate the similarity score using one of the formulas discussed above. It's a very simple model to make but needs more domain information to create it.

- *Advantages:*
    - *Transparency* - It's easy to see how we are finding the movie we are recommending in content based, but with collaborative filtering, we are trying to find hidden features.
    - *No cold start* - New items can be suggested without the need for a lot of ratings

- *Disadvantages:*
    - *Specialized* - The model only has the ability to expand on existing users' interests.
    - *Content analysis* - Depending on how much data you have the model may not be able to discriminate items precisely and therefore the model will be imprecise.

Hybrid Models and Deep learning Models

- *Deep learning Matrix Factorization* - We can use deep learning in an approach similar to matrix factorization, where instead of using some type of calculation to find a prediction like the dot product, we can use dense layers and the model will be able to find better predictions. Deep matrix factorization takes the embedded items and concatenates them then inputting them through a feed forward neural network. This network is then trained with parameters of the NN and the concatenated embedding, this now allows the model to find non linear relationships between embeddings whereas matrix factorization can only find linear relationships between features.
    - The advantage of this method is that we may not have the problem of the cold start anymore as the model should be able to handle new queries easier.
    - However using this approach does have a disadvantage as we lack transparency in why certain movies would be recommended.

- *Hybrid Model with deep learning* - This is another approach to a recommender system using deep learning. Here like matrix factorization we use embedding data, but we also combine it with metadata, such as user or movie information. This combines both the ideas of Collaborative filtering and content based filtering. The metadata is first vectorized into a sparse vector. The model can now use the vectorized dataset and embeddings to predict recommendations better.
    - The hybrid model has the advantages of both filtering methods as it is a combination of both. This means it doesn't have a cold start problem and learns hidden non linear relationships.

Reasoning for choice of algorithm

Simple Recommender

Myles Tyberg

For the simple recommender I chose to use Matrix factorization and minimized the cost function with stochastic gradient descent. The reason I built my model this way was due to these reasons:

- I chose matrix factorization as I wanted to see how a model with limited feature data would do, and then compare it to my hybrid. Matrix factorization also is a simple model and doesn't rely on any domain knowledge which meant I didn't need to include any metadata. This model also finds hidden features between embeddings which could discover new relationships between movies and users.
- I used stochastic gradient descent as the minimizer due to its advantages when it comes to missing data[1] and scaling to larger datasets.

Hybrid Model

For the hybrid model, I used a deep hybrid system using movie metadata which was vectorized using tf-idf . I chose this model due to these reasons:

- A deep hybrid model used the best of both filtering methods, as it includes Collaborative filtering and content based filtering.
    - We don't have the problem of the cold start, which means we will be able to recommend movies easily even if we haven't seen all the embedded data in our training.
    - The model will be able to find hidden relationships between embeddings more precisely as we can use the extra metadata for this.
    - As this model is non linear we can find non linear relationships.
- I could have used user information for the metadata but I chose movie information, the reason for this is during my data exploration I tried to find correlations between different different features and rating, I found that user data had a high linear correlation between ratings but using a non linear correlation the genre of the movie had a high score. I wanted to explore this path as NN are the perfect models for finding non linear relationships.

[1] https://stats.stackexchange.com/questions/201279/comparison-of-sgd-and-als-in-collaborative-filtering/203330

Myles Tyberg

Analysis of implementation

The hybrid model outdid my matrix factorization model . This is probably due to the extra data we included in the hybrid model, the extra information gave us the benefits of content based filtering which used metadata for better results. . However there ways we could still of improve our hybrid score:

- User metadata might have been better for the hybrid model, as there might be a better relationship between user information and ratings.
    - There is a way to include both user and movie metadata, this might give interesting results.
- The hybrid model is a neural network and therefore might take longer to learn hidden relationships as its looking non linear connections. Because of this I might need to increase the epochs. I did try to test this theory out and I got a better RMSE score with my test/train validation set but when I tested the u1 test data the RMSE score was worse than the lower epoch training. The increase of epoch data may have overfit on the training data.
    - It might be a good idea to increase epochs and also add more regularization to our model, therefore giving more time to learn hidden connections without overfitting the model.
- The model may be too simple, we are trying to learn a complex nonlinear function, there might not be enough nodes/layers to learn it. Increasing the complexity may increase the RMSE.