Final Project

Due: 5/20/2024

Shimon Greengart

Google Colab: Recommender Systems.ipynb

Different Methods of Building a Recommender System

The two main methods of making a recommender system are Content-Based Filtering and Collaborative Filtering. In both cases, the purpose is to whittle a large number of candidates down to something smaller so that another, more complex algorithm can fine-tune which ones to show using more complex criteria. In our case, the score is the predicted ranking that the user would give the item from 1 to 5 stars.

Content-based filtering works by using features found on both the users and the items, and tries to match similar users to similar items. For each user and item, we create an embedding in the same vector space that describes them. We then find the items most similar to each user, using the dot product or some other metric to show similarity.

Collaborative filtering works by finding similarity not just between users and movies, but between different users by looking at which items they used (or in this case, rated). We do so by generating embeddings based on what users did and what was done to items, and find similarity in a similar way.

The biggest difference between the two versions is what they make their embeddings based on. With content-based, the features often need to be hand-engineered. This means that making a content-based filtering model often requires significant domain knowledge – such as, for example, what about users determines which movies they watch. Collaborative filtering merely requires knowing which users have interacted with which items.

Another difference is that content-based filtering only requires knowing information about *this* user, not any others. This makes it easier to scale. It can also capture niche interests of a user that no one else is interested in. But it can’t expand a user's taste to new things based on what other, similar users are doing, something that collaborative filtering can do.

The biggest flaw of collaborative filtering is the cold start problem. If an item has never been interacted with, it has no embedding and can't be used to make predictions. The same is true with a new user. Various workarounds must be used to generate an embedding.

My Recommender

For my simple recommender, I use a collaborative filtering model to create embeddings for users and movies, and then dot product the embeddings to create a rating. The embedded vectors were of size 8 (I tried making them bigger, but it didn't have a significant effect on RMSE). Afterwards, I added biases for both the user and movie to the result, a number for each user or movie that says how good the movie is or how pessimistic the user is. I found it resulted in a slight improvement.

When training the model, I used an L2 regularizer with a penalty of 1e-4 to limit overfitting. Any higher and it stunted growth, while any lower and it overfit. I have also, instead of testing against the ratings, I centralized them to 0 with the mean of the train data. I considered fully normalizing it, but I didn't because that would mean that my RMSE is being measured at a different scale.

I had a significant problem where beyond a certain point, my model wouldn't even improve its training error. I initially assumed that the issue was the lack of negative feedback as described in the Google course, but adding in some negative examples only reduced performance (probably because we have non-binary ratings). In the end, I think it was something wrong with my preprocessing, because changing my preprocessing fixed it, but I'm not sure what.

I took inspiration when building my model from How to build a Recommendation System: Matrix Factorization using Keras | by Yash Sonar | Medium and Matrix Factorization with Keras for Movielens data. While they did use the movielens dataset, so does every other resource, including the ones you suggested to us. It doesn't look like there is another resource of significant size publicly available to train a model on.

For the hybrid model, I used an ensemble where after getting the result of the simple model, I ran it through a neural network that also sees the genres of that movie so that it can adjust things accordingly. Unfortunately, its validation performance was actually slightly worse than the simple model, implying that just by providing the extra information, I was overfitting to it.

Final Results

In the end, when I tested my model on the test data, it performed surprisingly well, with 0.8668 RMSE on the simple model and 0.8322 RMSE on the hybrid model. This is significantly lower than the validation for those same models, which doesn’t make any sense. I haven’t been able to figure out why this might be. Perhaps I did a particularly bad job with my train-validation split, where my validation data is much more on the extreme side compared to the test data, which was more normal. That’s the only reason I can think of.

Also, even though in validation, my hybrid model performed badly, in test, it is significantly better than the simple model. It seems it is incorporating the genre information to give better recommendations, just as it was supposed to. I’m just not sure why it’s only doing it now.