CNN Data Intelligence: MLOps Eng Take Home Project

Patrick Spencer

## Considerations with the data

There are about 50k users and about 1500 articles.

The data looks to be within one week in January of 2021. Because of this, we will not limit the queries to a certain time frame. If the data wasn’t limited to this week we would add in seasonality aspects to our recommender, for example, only recommend the most popular articles in the past week.

## How the Recommender Works

The recommender is split into two different recommendation paradigms: a cold start and an item-item collaborative filtering model which uses past user history to find articles similar to the ones the user has already visited.

### Cold Start

A common problem in recommendations is the cold start problem. How do we make recommendations if we don’t know a user’s preference? There are many ways to do this but we will just focus on the most popular articles in the past week. We could be biased to one specific category though so we will grab the most popular article from each category. We will limit the number of recommendations to 5.

### Enhanced Recommendations

If we know a user’s past history then we can use that information to recommend something the user might be more interested in. There are many ways to do this but the two classical models are user-user and item-item collaborative filtering. In this problem, the items are articles.

We do not have information to create a similarity score between users but we do have enough information to create a similarity score between the items. We use a TFIDF vectorizer to compare article titles. We will then use singular value decomposition (SVD) to create an item-item collaborative filter model. The SVD model will come from the python library ‘Surprise’.

Whenever we know the user’s history we will do the following:

1. Look at each user’s history
2. For each article in the user’s history, grab the 5 most similar articles based on the original article’s headline. Use the similarity scores from the singular value decomposition to rank the articles.
3. Grab the 5 most similar articles.

## Enhanced recommendations Metrics

We use 5 fold cross validation with root mean square and mean average errsor to get the following, offline scores, which seem reasonable for a first version of the recommender:

'test\_rmse': array([0.88737601, 0.88714016, 0.88722031, 0.88736068, 0.88746047])

'test\_mae': array([0.88217455, 0.88187476, 0.88196833, 0.88211519, 0.88221476])

On average the RMSE score is about 88% and the MAE is 88%

## How do we score the user’s articles?

We need a rough score for how the user ‘feels’ about an article. For this we will count up the number of times a user views an article in a week and divide it by the number of articles the user has seen during the week.

## Optimizations

Right now the recommendations are a bit slow. They take about 1500 ms to make a recommendation api call. This is because it has to load the precomputed data frames and sort through the data. We can optimize this with a dedicated data store like AWS DynamoDB.

If we had more information about the user’s preference we could make a user-user collaborative filtering model and include that in the model output along with the output of the item-item recommendations.

## How is the api built

The API is built in python using Flask, Pandas, Surprise, and Scikit-learn. I have mad documentation about how to set up the api in README.md file along with the package.

## Sample Article Outputs

See the README.md for sample outputs.