

## CS579 Project 2 weekly sample report

Bhanuja Arekatla(A20449753)

Navya Medarametla(A20442648)

WEEK-4 Date 11/19/2020

### Last week's work:

Given this dataframe we used the `userId`, `movieId` and `rating` to construct a sparse matrix, feeded it into a collaborative filtering based algorithm and generated item recommendations for users.


```
<31647x175434 sparse matrix of type '<class 'numpy.float32'>'
  with 506402 stored elements in Compressed Sparse Row format>
```

### This week's progress:

Next, perform Bayesian Personalised Ranking to provide a user with a ranked list of items. This Ranking optimization criterion involves pairs of items(the user-specific order of two items) to come up with more personalized rankings for each user.

```
from implicit.bpr import BayesianPersonalizedRanking
```

```
np.random.seed(1122)
bpr = BayesianPersonalizedRanking(iterations=100)
bpr.fit(user_item.T.tocsr())
```

100%  100/100 [00:46<00:00, 2.13it/s, correct=92.01%, skipped=0.86%]

We considered the first user as an example to see whether our recommendations are calibrated or not. Once we're familiar with the procedure for one user, we planned to repeat the process for all of the users if we'd like to.

```
16]: # Look at the first user
      user_id = 0
      interacted_ids = user_item[user_id].nonzero()[1]
      interacted_items = [item_mapping[index2item[index]] for index in interacted_ids]
      interacted_items[:10]
      #interacted_ids[66296]
```

16]: [Fogo de Chao]

So, the first user mostly visited to Fogo de Chao and this restaurant has to be in the topn recommendations that must be provided to him.

```
# map the index to Item
reco_items = [item_mapping[index2item[index]] for index, _ in reco]
reco_items[:10]
```

```
[The 3rd Coast,
Mastro's Steakhouse,
The Original Gino's East of Chicago,
La Pasadita,
Simply It,
La Villa Restaurant Pizzeria & Banquet,
Little Bucharest Bistro,
The Art of Pizza,
Sushi X,
Mercat a la Planxa]
```

The user 1 has interacted with restaurant Fogo de Chao however, its are nowhere to be seen in the topn recommendation to the user, hence we can argue based on the output that our recommendation might not be that well calibrated to the user's past interaction.

To scale this type of comparison, we'll now define our calibration metric -KL divergence to compare whether two distributions(interacted distribution and recommended distribution) are similar to each other.

```
compute_kl_divergence(interacted_distr, reco_distr)
```

```
: 6.643856189774724
```

### Generating Calibrated Recommendations(which only optimizes for score, s)

```
class Item:

    def __init__(self, _id, name, categories, score=None):
        self.id = _id
        self.name = name
        self.score = score
        self.categories = categories
```

For a given user, generate the list of items that we can recommend, during this step, we will also attach the recommender's score to each item.

Next, we created an objective function for computing the utility score for the list of recommended items.

Start with an empty recommendation list, loop over the topn cardinality and during each iteration update the list with the item that maximizes the utility function.

Then we compared the calibrated recommendation (which only optimizes for score, s) and the original recommendation and the user's interaction distribution.

lmbda : float, 0.0 ~ 1.0, default 0.5 Lambda term controls the score and calibration tradeoff, the higher the lambda the higher the resulting recommendation will be calibrated.

**Lambda is keyword in Python, so it's lmbda instead.**

```
original reco kl-divergence score: 6.643856189774724  
calibrated reco kl-divergence score: 4.070966521354143
```

Printing out the categories distribution from the calibrated recommendation list shows that this list covers more categories and its distribution closely resembles the distribution of the user's past historical interaction and our quantitative calibration metric, KL-divergence also confirms this. i.e. the calibrated recommendation's KL-divergence is lower than the original recommendation's score.

**But, is there any way that we can minimize the score and optimize the performance using any other ranking metrics? Or does it just result in score drop!**

Lets look into that in the following week..