BonsaiShop ML Task

As described in task description we have to train model that will rank each item for each user. Those items that have higher rating considered to be more relevant for user (in our case relevancy is purchase). Thus we have to optimize our model so that we predict potential purchases well (in our case purchase is <<checkout>> impression) and give them higher score. In order to solve this problem we need robust evaluation metrics to check and compare our models. We need to choose right architecture for our model as required it has to be both accurate and scalable for big amount of data. Below I will describe step by step how I came to my solution and why.

Content:

Evaluation Metrics

AUROC - As it was described in task it is good measurement for this kind of algorithms. I tuned my model for this metric.

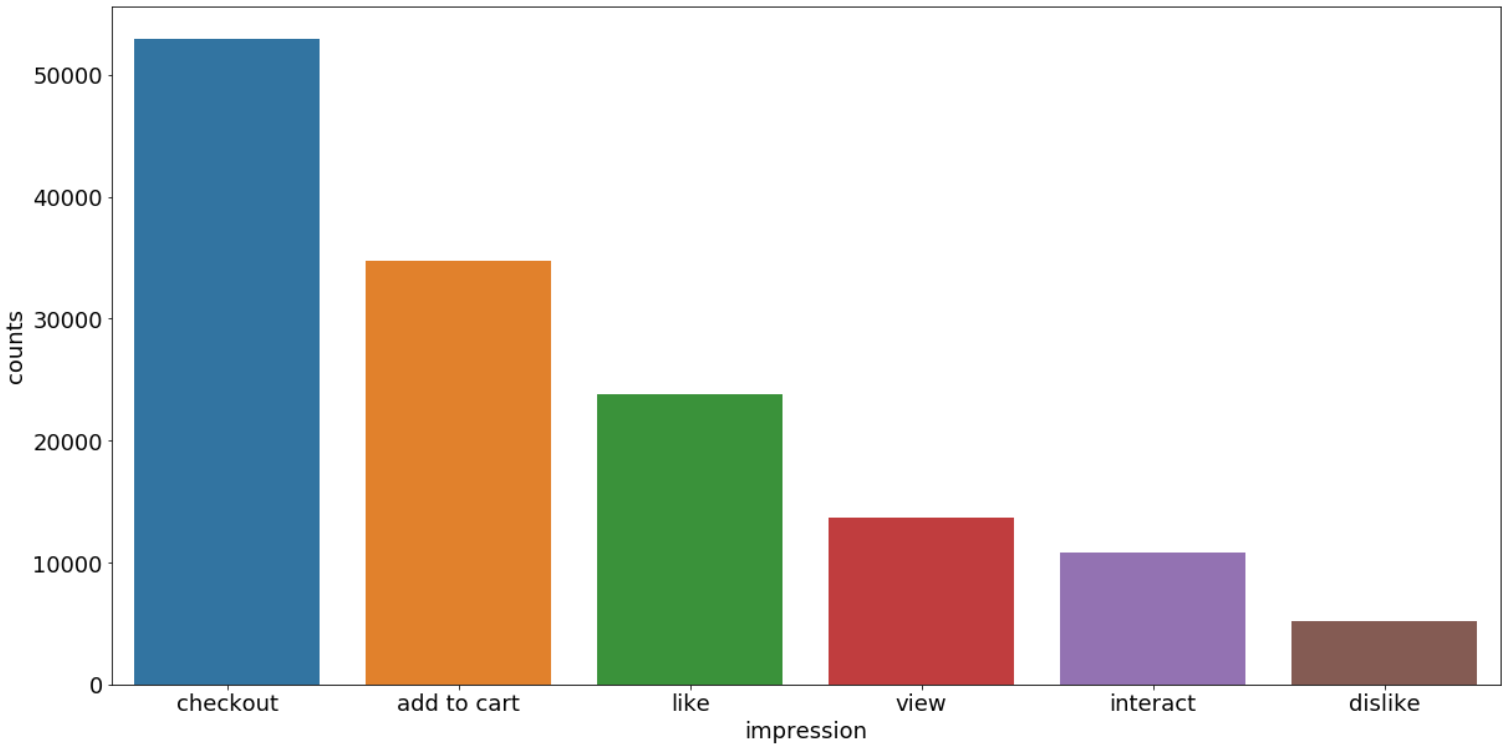
NDCG (normalised discounted cumulative gain) - besides the fact that AUROC is good metric. I personally think NDCG is better choice in case where one need to evaluate ranking system like this. Because this help us better understand how well we ranked those products that user potentially wants to by.

Time base splitting - this type of systems have to be tested by time based splitting, thus data for training and validation have to splitted by time not just by proportion, in order to understand how well your model adopt over time. As I did not have any timestamp for impressions I just made random split of my data.

A/B test - of course offline metrics are good. But we never know that will actually happen in real world with real users, especially when we are playing with such KPIs as amount of purchases per user. Thus after offline evaluating this algorithm have to be tested on some statically significant segments of real users to understand how our KPI's are changing over time.

Data Exploration

This was the trickiest part. In dataset that you provided I had both explicit kinds of data (like, dislike) and implicit kinds of data (view, interaction, add to cart, checkout). Moreover they had very strange distribution for this kind of data. Lets look at histogram below.



As you can see there are more *checkouts*(purchases as I understand) than *views* and *interactions*. And more important there are more *checkout* than *add_to_carts*. This was key point because in order to *checkout* item one has to do *add_to_cart* operation. Then I realized that there is only one *impression* per (user, book) pair. There is no user that has more than one impression on the item for example user could *view* something and then *like* that book. Here I realized that this is some kind of explicit data like obvious ratings (because user could have only one rating for each item). I don't want to look nerd))) but I started to look at public datasets available. There were only few book rating datasets. I found BookCrossing dataset and after a couple of joins I found that dataset is subset of BookCrossing dataset with only explicit ratings.

- Checkout - [9,10]
- Add to cart - [8]
- Like - [7]
- Interactact - [6]
- View - [5]
- Dislike - [1,2,3,4]

As I found implicit feedbacks where thrown away.
After this I have finally understood why my data fits better on explicit type of algorithms. And then I came to my solution.

Model selection and tuning

There are many ways to solve this problems. Nearest Neighborhoods, Neural Network classification, Collaborative filtering with Matrix Factorization and so on. I did not have enough time to test all of this but instead, I concentrated on well know Matrix Factorization with explicit feedback. I choose to train it via alternating least squares (aka ALS). As so with this method it is possible to parallelize training for big amount of data and it also has implementation in Apache Spark. Besides that matrix factorization is pretty good for production workloads we will discuss it later.

I just trained standard Matrix Factorization without using side information like age, gender, movie type and so on. There are some type of MF models that learn prior distribution of data. This is good for cold start users those who don't have enough activities.

Besides side information in case of big amount of data per region, It might be necessary to train recommendation model for each region separately. It is standard practice know in many big companies like Netflix.

I did not used side information because training those additional parameters would require more work and some custom implementation (Of course it is better to test this type of models).

Below I What model I used and how I tuned it.

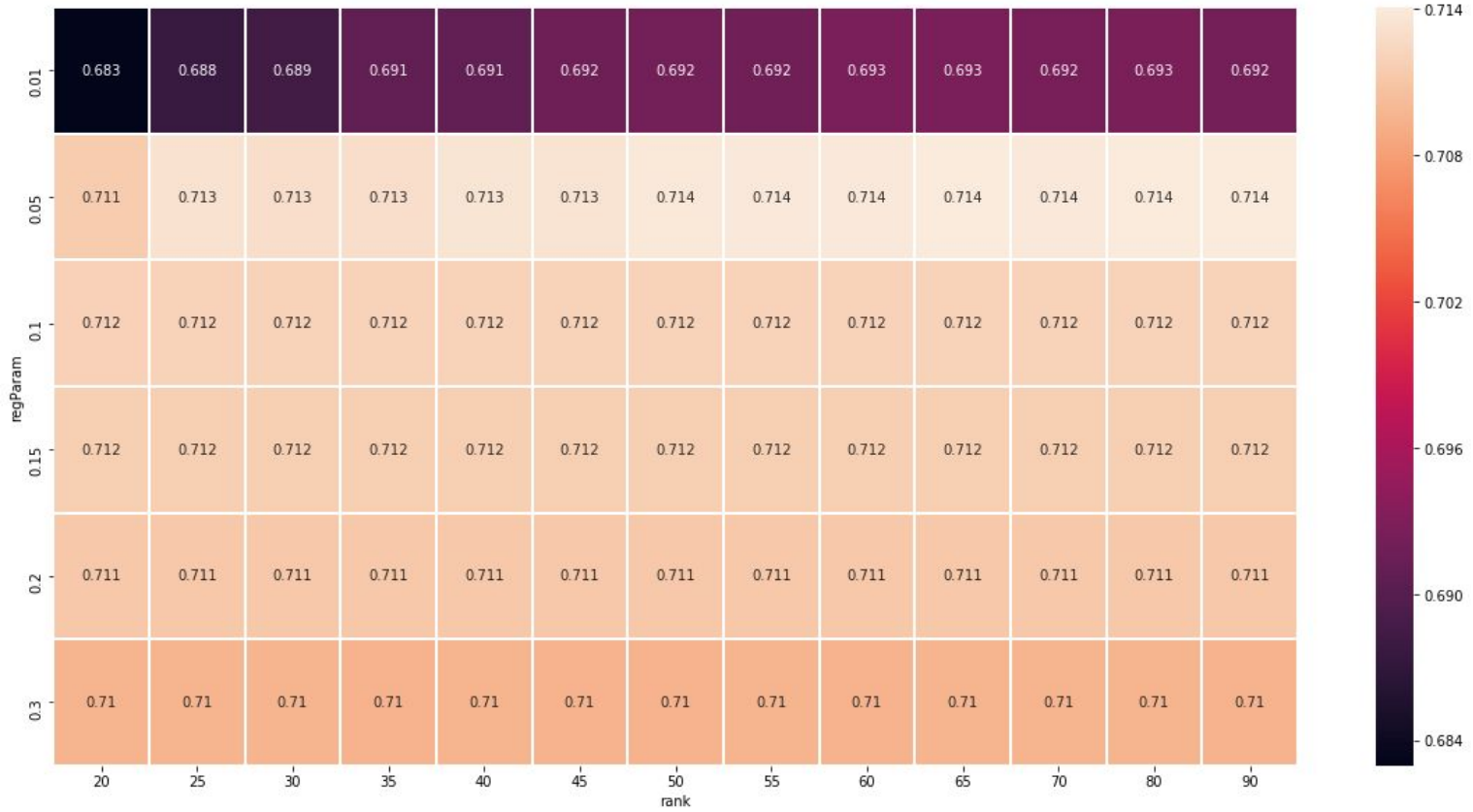
Model: I used spark implementation of ALS matrix factorization with explicit data settings.

Ratings: I assigned score for each impression as below:

- Checkout - 0.95
- Add to cart - 0.8
- Like - 0.7
- Interact - 0.6
- View - 0.5
- Dislike - 0.25

Tuning:

I just made a standard cross validation with many parameters (rank, Lambda) in order to tune my model for better AUROC score. You could find result of cross validation in cross_validation_metadata.csv and in visualization (regParam, rank, auroc_score) below.



Deployment

Matrix Factorization models could be deployed as batch job and also as real time service.

Real time service:

In order to generate ratings for users we have to multiply the user_factors with all items_factos, then sort resulted array.

In this scenario the bottleneck of this algorithm is sorting part. Even though this way we could serve models(rank up to 128) with up to 50M items under 200-300 mls on commodity hardware. Which is pretty big number. And this information could be cached for some time. There are no need to calculate it each time user look at his profile. I give this time metrics based on my previous experience with MF deployment. We are successfully serving it in real time manner.

Batch job:

Even though Matrix Factorization is well adapted for batch jobs. In case where we have 100m items or more it could be pretty hard to calculate score for all users. In this case we have to sample only those user who are active and using app. Because it could take really big time to calculate ratings for all users and all items at once.

Results

Cross validation shows that ***AUROC*** score stack at 0.71 which seems not so well. But ***NDCG*** score was about 0.895 which is very good result for ranking system. I choose rank=25 and lambda=0.05 as it shows similar results to bigger ranks but it will be faster to run on production. But of course this is just.

You could find final rating matrix by link below:

https://drive.google.com/file/d/1uR43xfrQIHLEIceAIhO4mYA7ouCpzYEj/view?usp=sharing

Conclusion

In this work I just worked and optimized one type of algorithm for one metric, but in real world scenario we have to try many types of algorithms and understand which one is better fit for us. And each model that behave well on offline metrics have to be tested as A/B. My experience with my current company has shown that actual problems and bottlenecks of model are found after initial deployment when results of algorithms interact with real users.

Thanks for you time!