Final Project - Data Streaming Algorithms

Stav Yosef, Daniel Sabba

Today, our world needs models to make predictions about everything, and in our case recommend user's preferences.
We can predict using a variety of methods, some of them are simple such as average completion and content-based filtering and some of them are more complicated like neural collaborative filtering and even hybrid systems.
Over this project, we used 2 different datasets, the first one is containing 1 million ratings and it's used for the methods: average, content-based filtering, and matrix factorization using SGD, the second dataset is containing 1 million ratings but the training process is by positive and negative instances, used for the last part of this project, neural collaborative filtering systems.

Project layout:

Matrix completion

- Fill sparsity by avg ranking
- Content base recommendation
    - Approach one
    - Approach two

Matrix factorization using SGD

- Different values of latent dimension

Neural collaborative filtering

- GMF
    - Weighted GMF
- NLP
- MLP
    - MLP with RELU & MSE

To evaluate our experiments, we used some know metrics.
RMSE - Root Mean Squared Error.
MRR - The mean reciprocal rank is a statistic measure for evaluating any process that produces a list of possible responses to a sample of queries, ordered by probability of correctness.
For example, suppose we have the following three sample queries for a system that tries to translate English words to their plurals. In each case, the system makes three guesses, with the first one being the one it thinks is most likely correct:

Query	Proposed Results	Correct response	Rank	Reciprocal rank
cat	catten, cati, **cats**	cats	3	1/3
tori	torii, **tori**, toruses	tori	2	1/2
virus	**viruses**, virii, viri	viruses	1	1

Given those three samples, we could calculate the mean reciprocal rank as (1/3 + 1/2 + 1)/3 = 11/18 or about 0.61.

NDCG - normalized Discounted cumulative gain (nDCG) is a rank score similar to MRR
nDCG measures the usefulness, or gain, of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom, with the gain of each result discounted at lower ranks.

Matrix completion
In this section, we are examining three methods – average filling and content base recommendation and matrix factorization.
While the former method time is very fast and doesn't require much processing, this approach may fit in various scenarios when fast calculation needed and accuracy is less important, there are two ways to fill the missing values with average, each missing value [UserID,ItemID] if filled with either1. the average rating that the user gave to all the items or 2.the average rating that the movie got from all users.

As we can see in our experiment –

	rmse	mrr5	ndcg5	mrr10	ndcg10
0	3.639923	0.76747	0.76747	0.775621	0.775621
1	3.218173	0.59591	0.59591	0.583813	0.583813

Where the first row is the first method and the second row is the second.
We can see that with an average rating that the user gave to all the items is less efficient, we might believe that there are few reasons for this behavior – popular

movies have better ratings and therefore the predicted value should be high above the average – same for lame movies.

On the other hand, the MRR score is better in the first approach, we believe that this comes from user behavior, when a user tends to rank high or low, he will probably continue with this trend.

The second method – collaborative filtering is a stable approach, were to predict the ratings each user will give to each movie, we consider any prior information – in our case its every movie and its genre. Each movie can be tagged with several genres, there are two ways to predict the rating.
We can construct a user profile matrix, each row will be a user and each column will be the weighted average of the movie profile he rated – "user preferences", in this way, there is only a way to recommend movies to a user, but we can't predict the actual rating.

The second way is the method we implemented, the similarity score between two movies is calculated by computing the similarity between the movie profiles of each movie pair. The predicted rating a user will give to a candidate item is calculated by the rating the user gave to K most similar items to the candidate item. The recommended movies are those with the highest predicted rating.
To find K most similar items, we construct a movie profile matrix, where each row is a movie ID and each column is a specific genre, the values filled are the TF-IDF of the genres, this helps us to understand which genre for each movie is common or "strong".

We find K most similar items by cosine similarity of each movie (that is rated) and the movie we want to recommend, the rating that is predicted is the average of those similar movies.

This approach gave us better results.

rmse	mrr	ndcg5	ndcg10
1.006745	0.848648	0.004889	0.006079

The latter method - matrix factorization is the best method we checked in this section, MF produces the best results but takes longer to train.
A summary on matrix factorization, each user can be described by *k* *attributes* or *features*. For example, feature 1 might be a number that says how much each user likes sci-fi movies. Each item (movie) can be described by an analogous set of *k* attributes or features. To correspond to the above example, feature 1 for the movie might be a number that says how close the movie is to pure sci-fi. Each item (movie) can be described by an analogous set of *k* attributes or features. To correspond to the above example, feature 1 for the movie might be a number that says how close the movie is to pure sci-fi.
There are many graphs in the notebook, we'll present the summary of the results.

Test Matrix Size - 2000

Latent Dimension	RMSE / Iteration	MRR / Iteration	NDCG / Iteration
Dimension - 20	0.820 / 98	0.99829 / 93	0.570 / 48
Dimension - 50	0.821 / 95	0.99812 / 90	0.572 / 11
Dimension - 100	0.820 / 96	0.99829 / 73	0.555 / 8

Test Matrix Size - 20% of total ratings (200K ~)

Latent Dimension	RMSE / Iteration	MRR / Iteration	NDCG / Iteration
Dimension - 20	0.80 / 94	0.99843 / 83	0.553 / 48
Dimension - 50	0.80 / 96	0.99848 / 72	0.573 / 13
Dimension - 100	0.80 / 97	0.99816 / 97	0.563 / 41

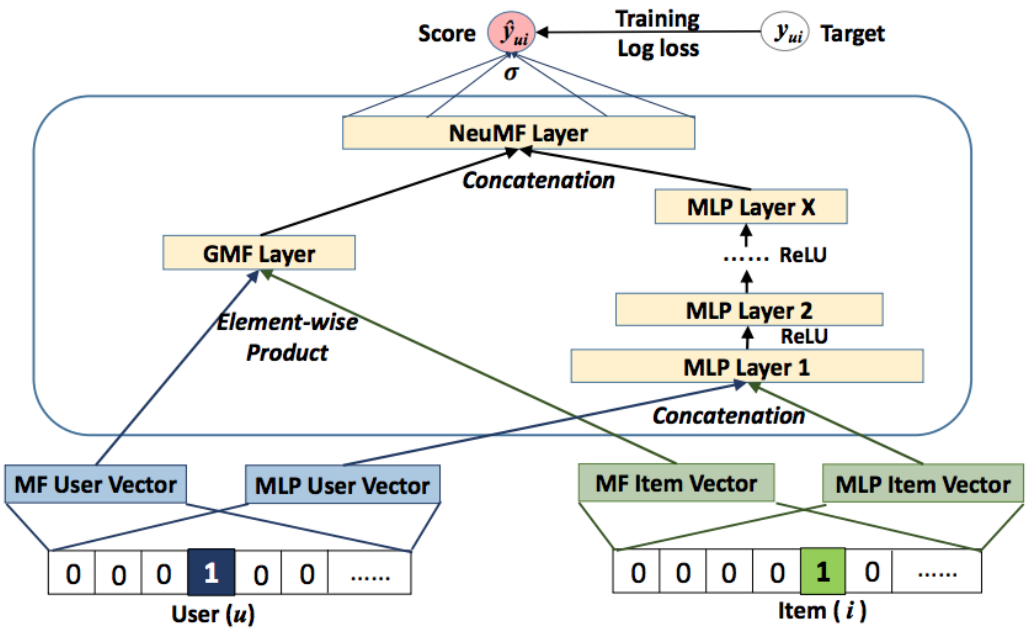
As we can see we achieve the best results with MF but there is a tradeoff we mentioned earlier, it takes more time to train a model than the first two methods.
The first methods it's instant, the second can take a couple of minutes and MF takes 90 seconds per iteration (with evaluation per iteration, without taking 20-sec ~)
Also, we can see there is not a big difference between small and big latent dimensions therefore, we'll use the 20 dimensions option.
This is crucial, we are reducing a lot of memory usage from none-latent dimension to using only 20.
The last thing we can see is there is not a big difference between test matrix size of 2000 to 20% of total ratings (slightly better results to 20% "team", around 2 percent).

Let's review what we did using the neural collaborative filtering approach.
First of all, we need to understand the structure of the dataset.
ml-1m.train.rating contains almost 1 million ratings (user ID, item ID, rating, timestamp)
ml-1m.test.rating contains 6040 (as the number of users) ratings (same format as the training dataset)
ml-1m.test.negative contains 6040 rows and each row is built with 1 positive instance and 100 negative instances.

Using this kind of dataset limits us with producing RMSE results as before so we'll evaluate using MRR and NDCG metrics and with the "Revenue" metric which we will explain shortly.

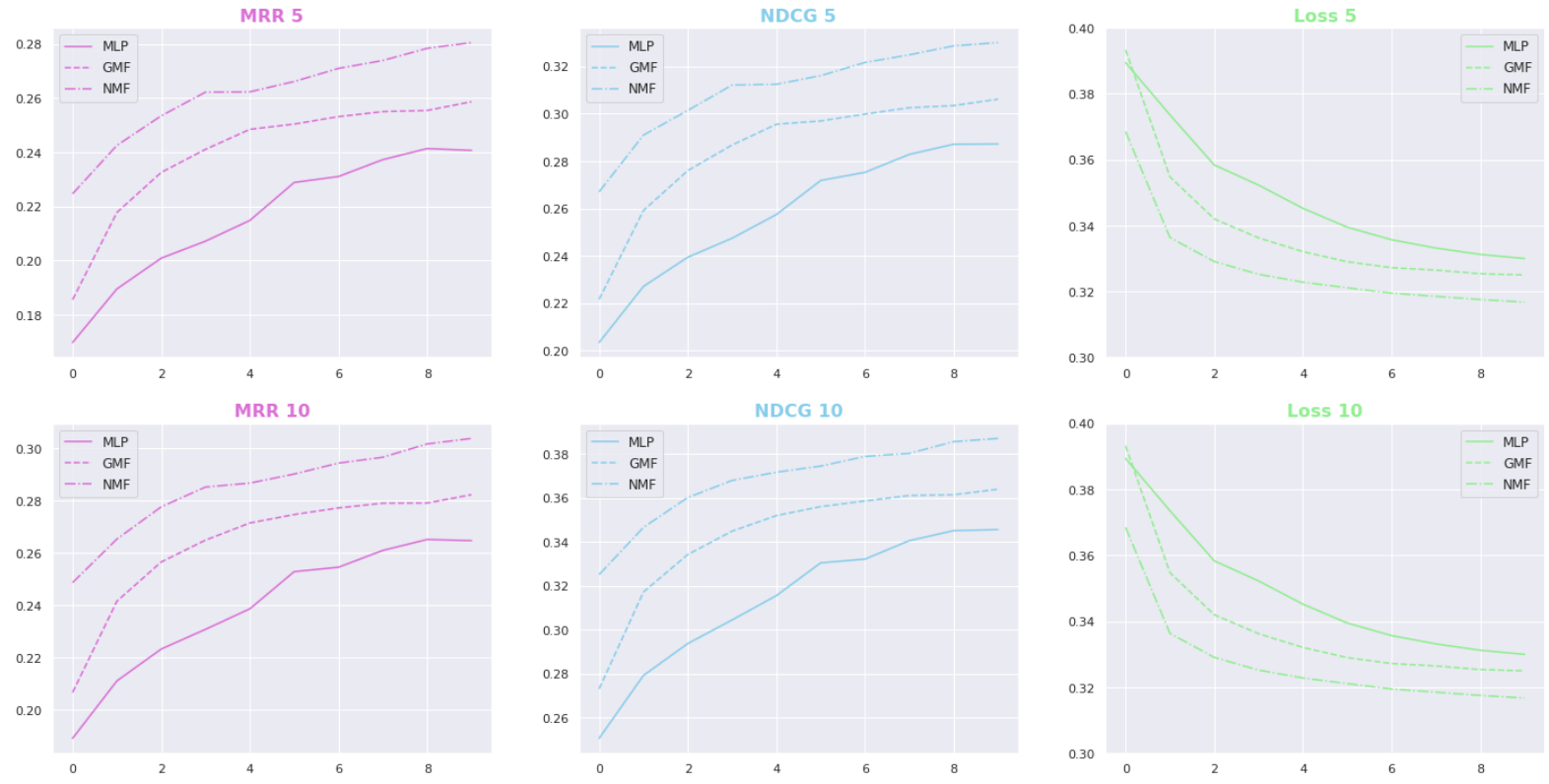
We used 3 different approaches:

1. GMF - Generalized Matrix Factorization. GMF applies linear kernel to model user-item interactions.
2. MLP - Multi-Layer Perceptron. MLP uses multiple neural layers to the nonlinear interactions.
3. NMF - NeuMF, fusion of GMF & MLP.



Let's dive right into the results we got from the models, we used cut-offs 5 and 10.

All Models Comparison

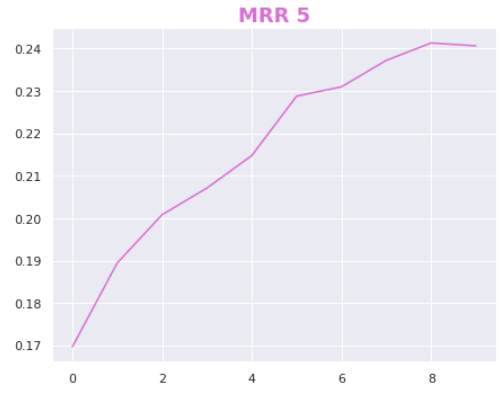


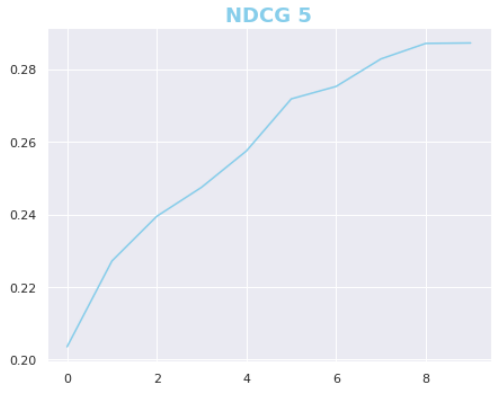
The graph above reflects best the scores of our experiment, although MLP scores are lower than GMF where GMF with identity activation function and edge weights as 1 is MF, we do believe that with further learning (epochs) the MLP will outperform the GMF because MLP provides flexibility and non-linearity.

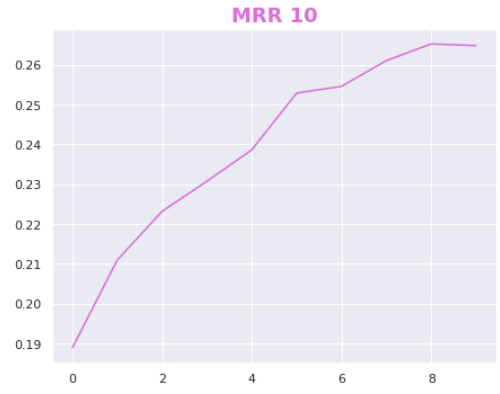
We can also see that NeuMF performs best - which is make sense because of his mix of the two approaches (as we can see in the picture down below). Furthermore, looking at the learning process of each approach, MLP (and by so - NeuMF) learning curve is more smoothed than GMF which implies that the learning process is slow (make sense since we have more weights than GMF).

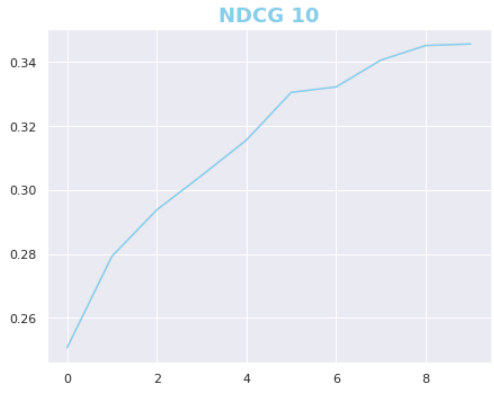
We even tried to change the activation function of the last of MLP from sigmoid to RELU and the loss function from binary cross-entropy to MSE.

Model - MLP

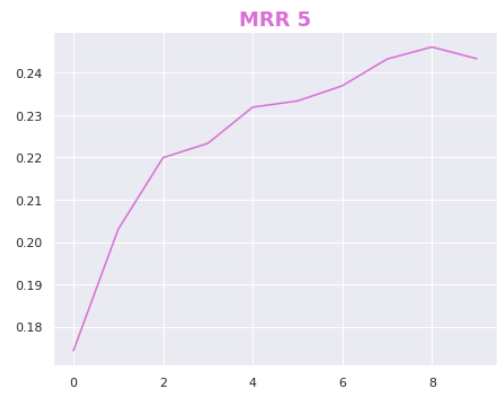


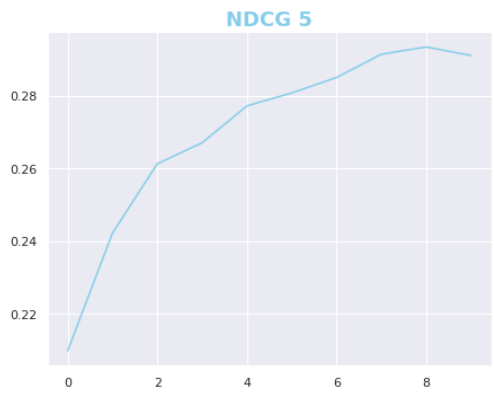


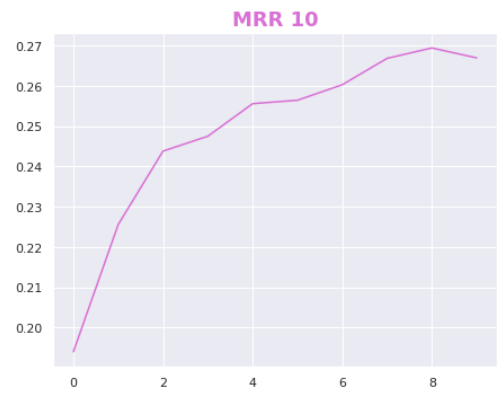


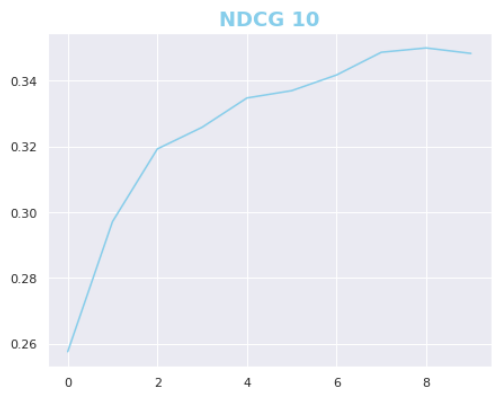


Model - MLP - RELU & MSE

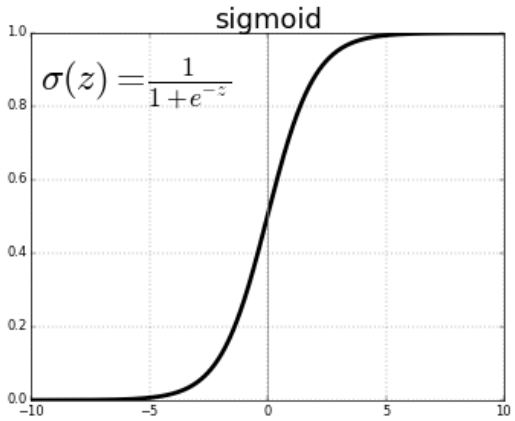


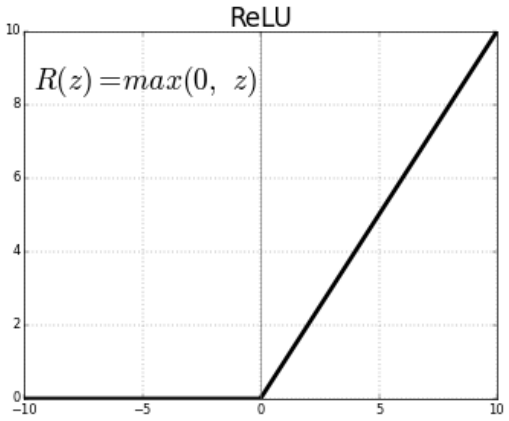






As we can see above the results when we are using MSE & RELU are better than sigmoid & binary cross-entropy (we can see significant improvement in loss plot).
Cross-entropy (or softmax loss, but cross-entropy works better) is a better measure than MSE for classification, because the decision boundary in a classification task is large (in comparison with regression). MSE doesn't punish misclassifications enough but is the right loss for regression, where the distance between two values that can be predicted is small.
Cross-entropy is used as a loss function when the final layer has a sigmoid or softmax activation since cross-entropy is defined over a probability distribution - which is what these two activation functions output.
Mean squared error is used for RELU and linear activation functions, since it is a more natural fit for such a situation. The outputs of RELU and Linear activation functions aren't probabilities.



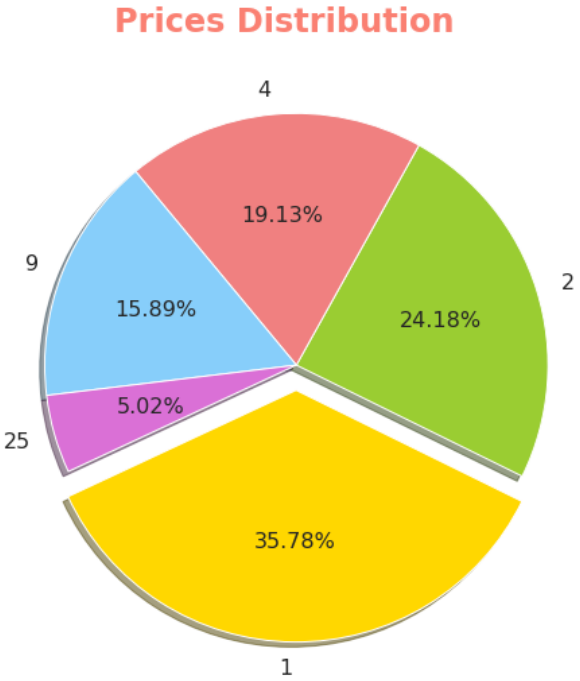


RELU advantages:

1. Not vanishing gradient.
2. RELU tend to give better results (better convergence) than Sigmoid.
3. It is easier to compute than Sigmoid because RELU just needs to pick max(0,x) and not perform expensive operations as in Sigmoid.

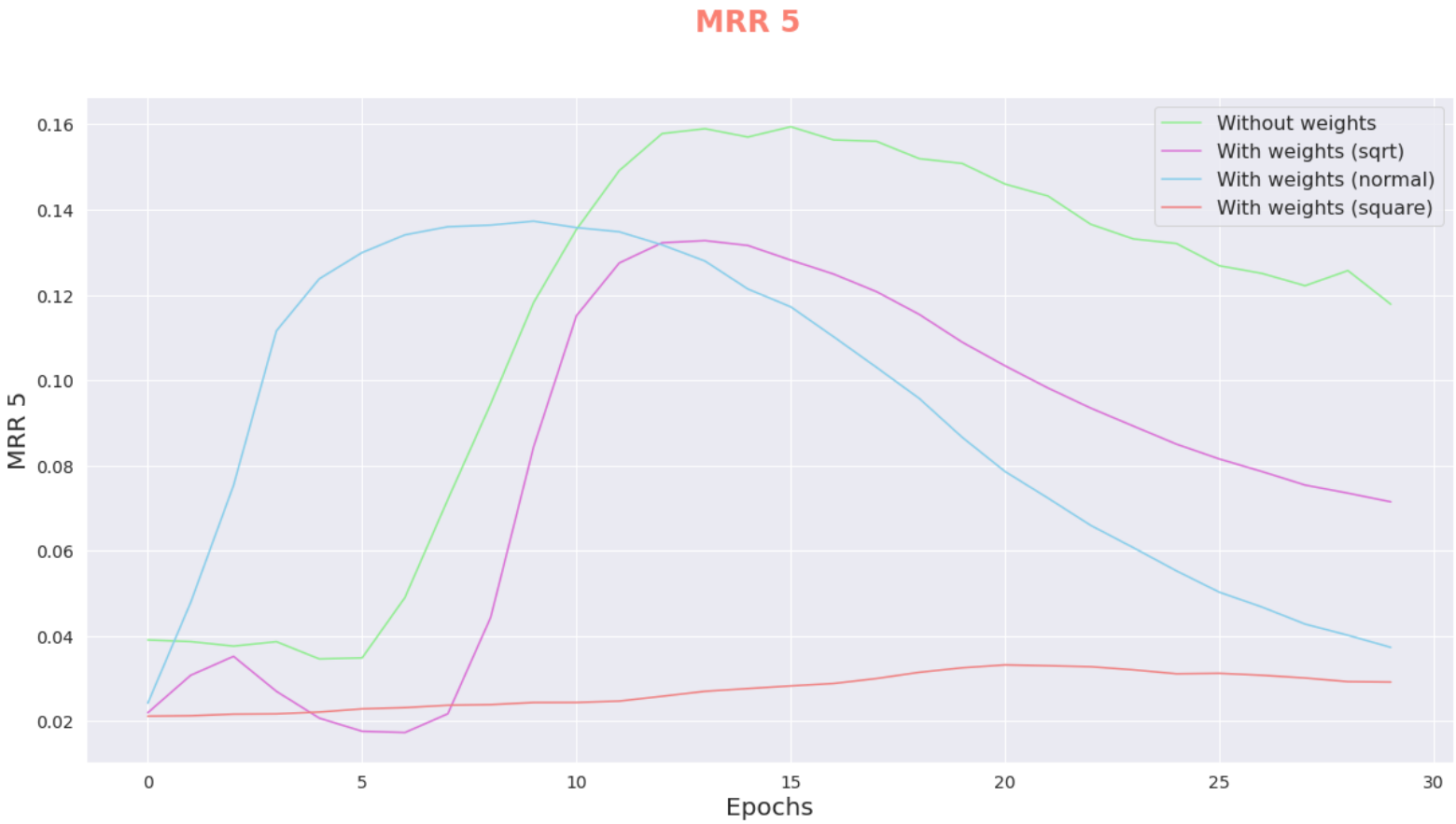
MSE is for regression problems, while cross-entropy loss is for classification ones.
In conclusion, in our situation RELU & MSE will perform better theoretically and practically.

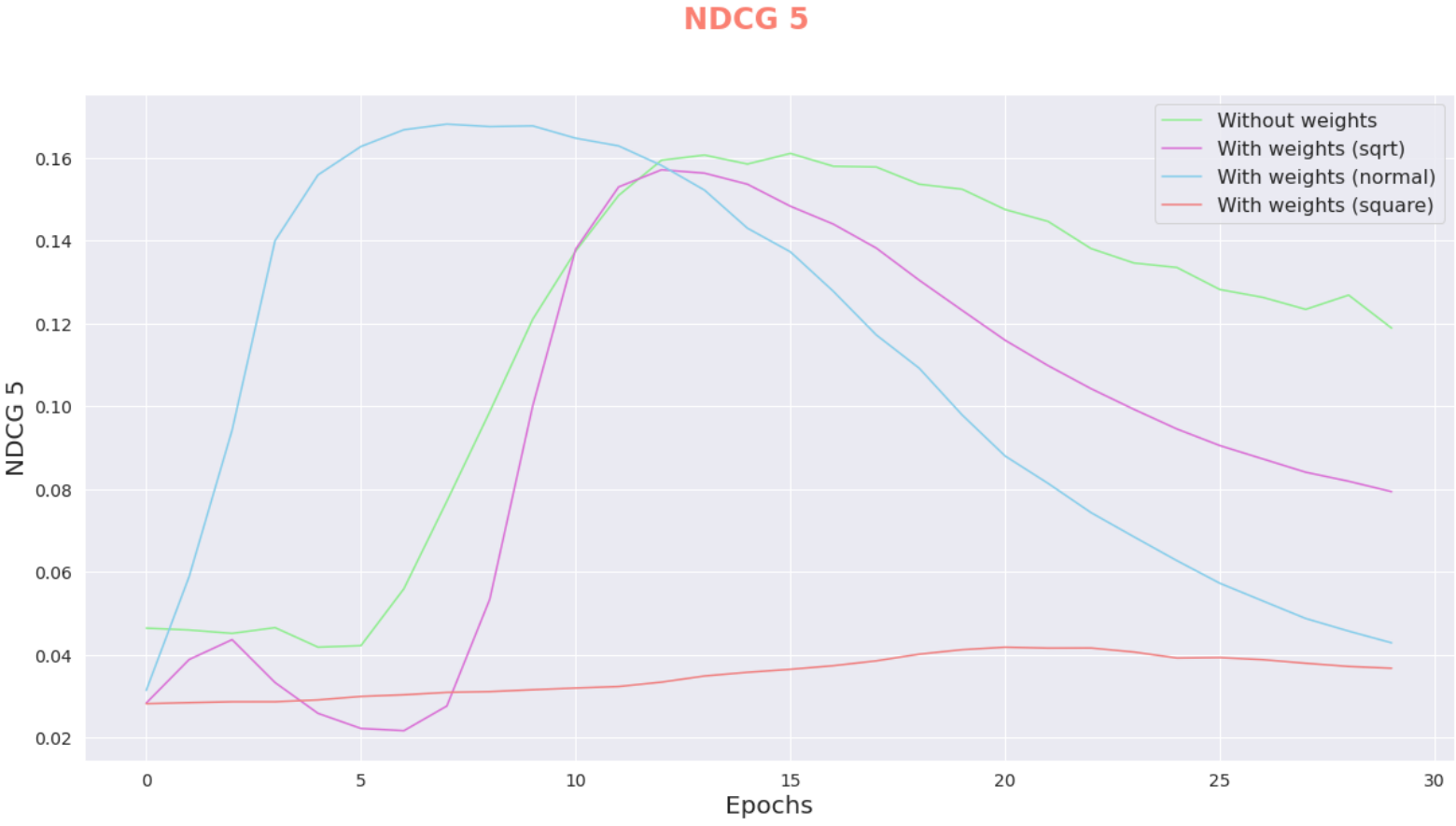
Let's roll back to the point we talked about a metric called "Revenue".
We wanted to draw a scenario not only predict item by other ratings but also give each item a weight, for example, if we have an e-commerce website and we want to recommend the items that will produce our website the highest revenue, therefore we need to give each item a price. (item_priceold2.csv)
Note - We decided to run our experiments with the GMF model



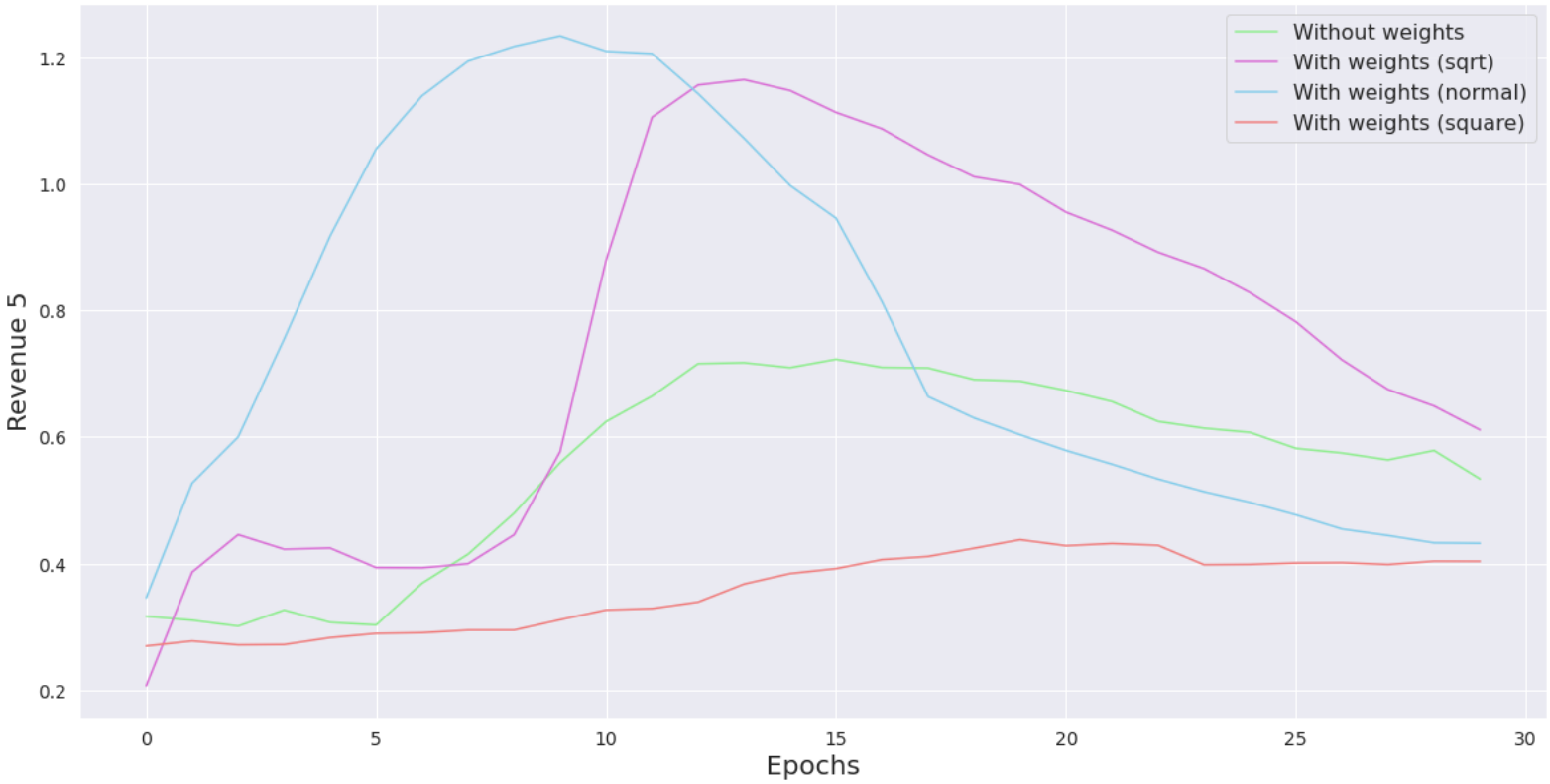
As we can see in the plots above price distribution is 1, 2, 4, 9, and 25.
The most common price is 1 and the least is 25, 35.78%, and 5.02% respectively.
To increase the revenue, we'll need to recommend more on the items with a price of 25 and less on items with a price of 1. (In total there are 3706 items)

Next, we had to think about how to set the weights for the items. Just give them their prices or something different.
We developed 3 models, the first one is "just" their prices as the weights.
The second way we square root their prices and the third way is to square their prices as the weights.

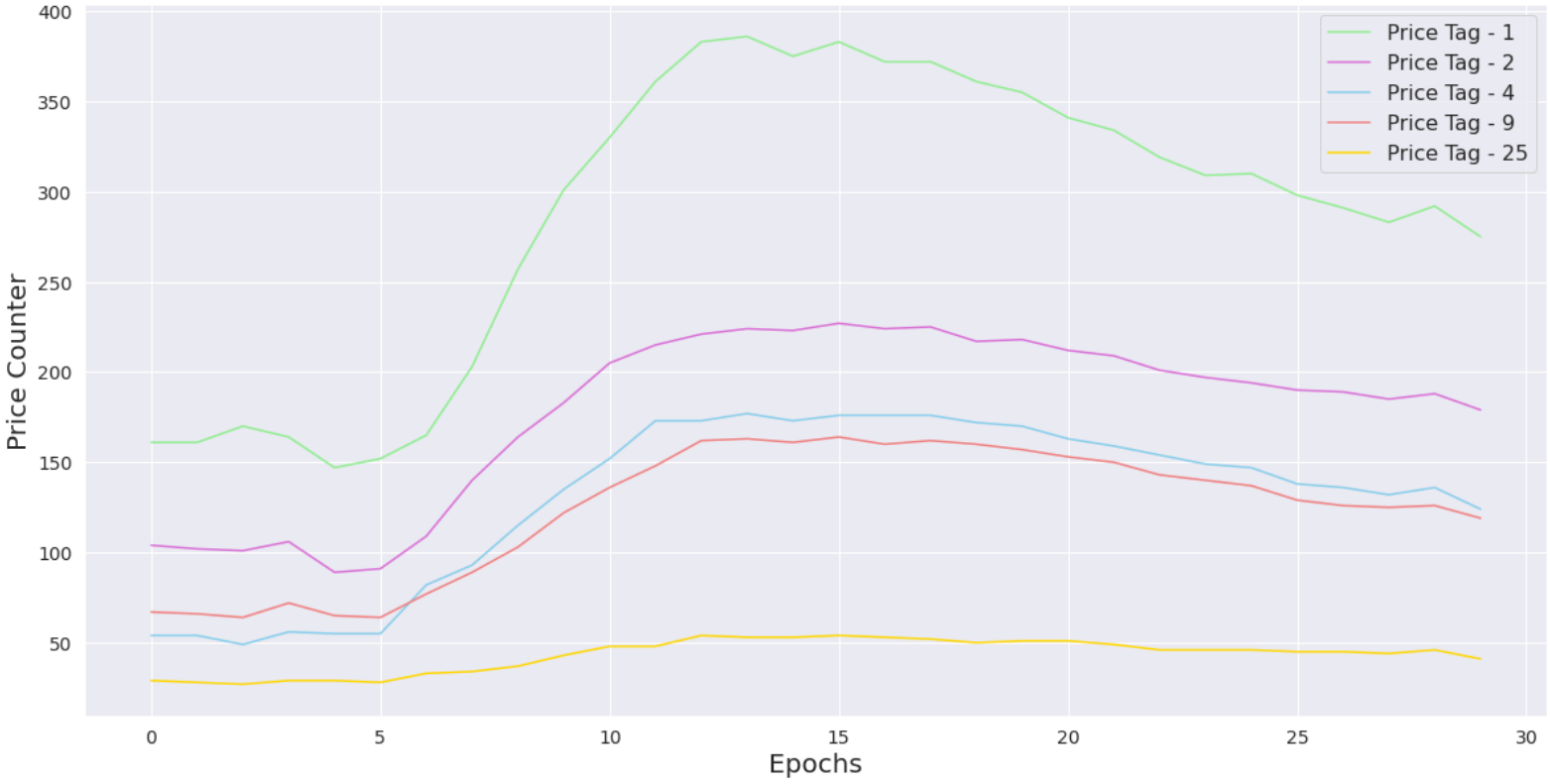




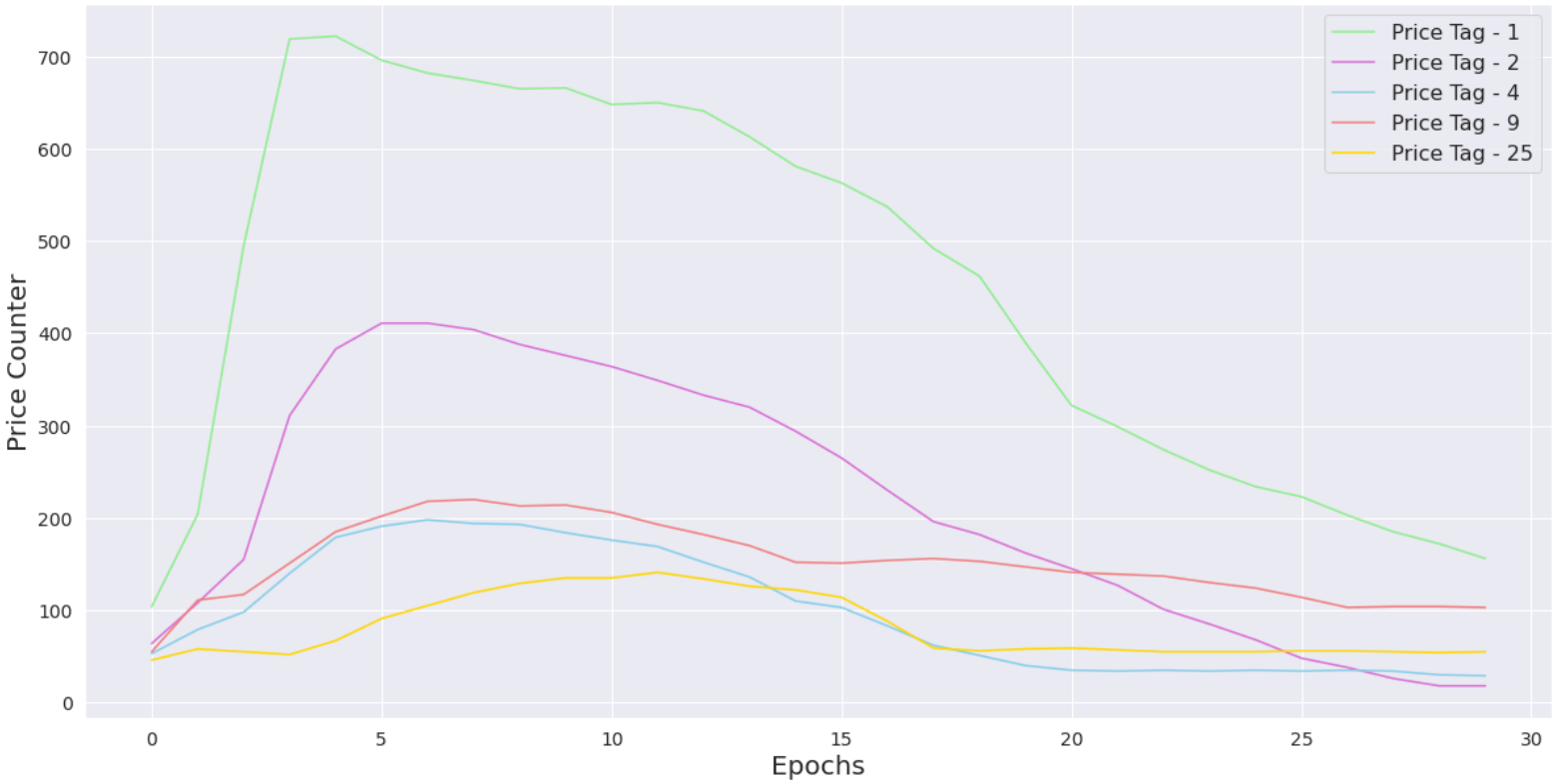
Revenue 5



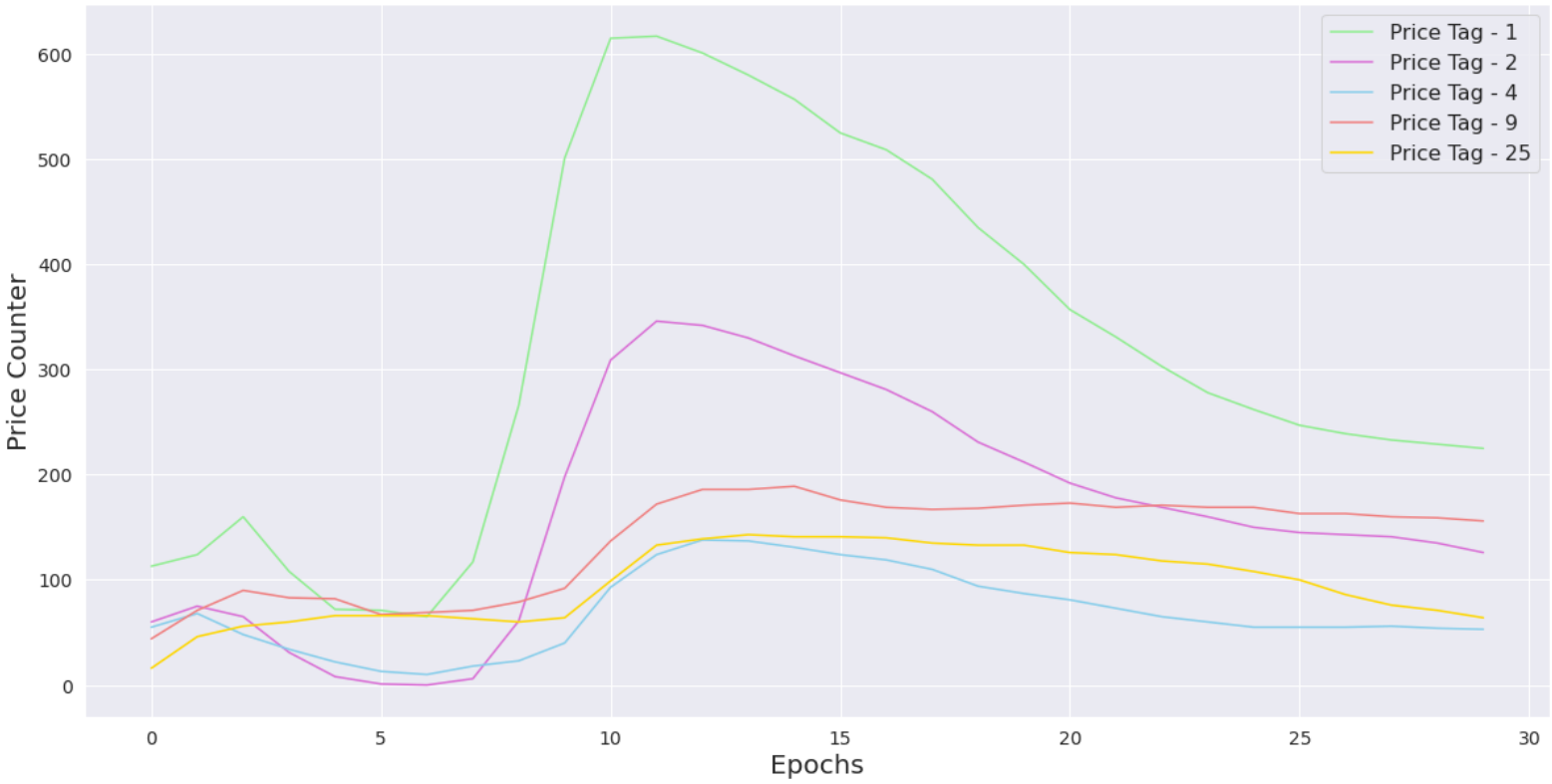
Items Distribution - Without weights



Items Distribution - With weights (normal)



Items Distribution - With weights (sqrt)



Let's start with the revenue, as we can see all models have healthy learning till epoch 10~15. We can see weighted models winning against the model without weights. The most profitable model is the model with normal weights. All the models have great progress through the 15 epochs. The best way to see changes is through revenue distribution. Let's compare the model without weights with the model with the normal weights.

Price 1 - 370~ vs 650~ **Price 2** - 225~ vs 375~ **Price 4** - 175~ vs 180~
Price 9 - 160~ vs 200~ **Price 25** - 50~ vs 125~

We can see the model with the normal weights is winning in every aspect, especially in price tag 25, over 100% growth. In the models with the weights, we can see clearly that items with a price of 1 & 2 are getting "crashed" with the times they occurred. Another interesting factor we can see is the metric revenue with ranking where the model with sqrt weights is taking the first place from the model with the normal weights.

For Us, we were surprised that the model with square weights is pretty much the worse against all 3 other models in every aspect (revenue & item distribution). All 4 models have their advantages and disadvantages, the model without weights is fitting x30 times faster than the models with the weights. the number of negative instances per positive instance is taking a crucial part if we will use only 1 the models will be trained much faster but the results we are achieving are not so good and if we take too many negative instances it's can train the model in the wrong way (as I experienced). Batch size & learning rate have their part also, too big a batch size can take over all the RAM.

In conclusion, there are many factors we need to consider and, every parameter we choose has its advantages and disadvantages which leads to a trade-off between the time of training to the accuracy of the model and much more aspects.