Chapter 1

Vertical Scrolling Rhythm Game (VSRG) Score Estimation through Collaborative Filtering

Evening

Rhythm Gaming is a popular gaming genre that uses music to challenge the players' sense of rhythm and visual pattern recognition. Vertical Scrolling Rhythm Game (VSRG), the most popular sub-genre, seeing exponential growth in popularity in current years. However, recommending a suitable in-game level to an audience of a certain skill level is difficult; as game developers face challenges in predicting each level's difficulty due to it's complex unstructured data. To create this level prediction model, researchers require an unbiased ground truth to perform supervised learning on. Thus, to bridge that gap, we propose a collaborative filtering approach to estimate scores which can be used as an auxiliary for estimation of level difficulty estimation.

1 Background of VSRGs

A Vertical Scrolling Rhythm Game (VSRG) is a rhythm game where objects (notes) travel from top-to-bottom or bottom-to-top to hit a receptor. The timing when it lines up with a receptor is usually related to an instrument in the music. In a way, games like these help players mimic the play of the actual instrument.

Players' performances are evaluated by their timing accuracy: the difference between their input timing and the in-game timing when the note lines up. There are several ways to input, either using a keyboard, a pressure-sensitive device, a custom controller, and many more. To limit our scope, we will only consider keyboard inputs, as it's one of the most popular way to play this genre.

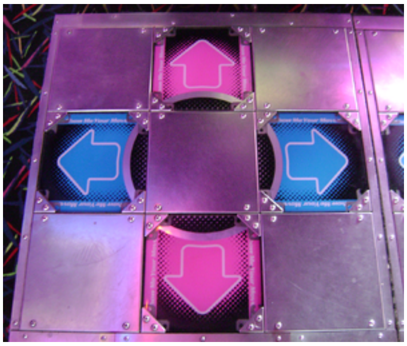

Figure 1: Pressure Sensitive Dance Dance Revolution device to be played with feet. Image from Wikimedia Commons n.d.(a)

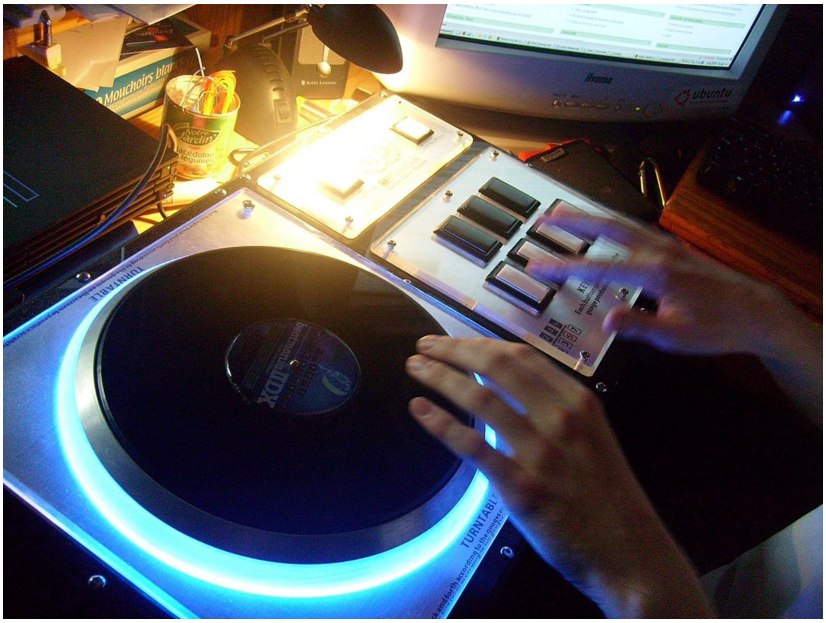

Figure 2: Beatmania IIDX Controller to be played with hands. Image from Wikimedia Commons n.d.(b)

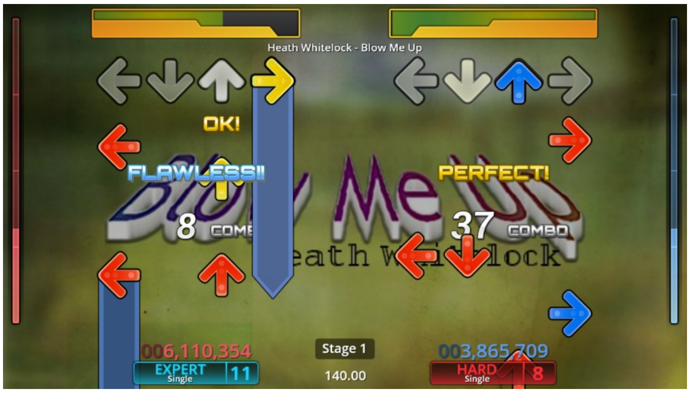

Figure 3: Image of StepMania 5.0.5. Shows gray arrow receptors on top, and red, yellow, blue notes travelling upwards towards the receptor. Image from Wikimedia Commons n.d.(c)

For example, StepMania in Figure 3, one of the more popular old VSRGs.

We observe notes in red, blue, yellow, indicative of their rhythm, travelling upwards to reach the receptor in gray. Upon reaching it, the player is prompted to input correspondingly, where they will be judged on their rhythmic precision.

A beatmap, or map, is the playable file, that spans the entire song. Similar to how you have stages or levels in games like Super Mario, each map corresponds to a stage.

1.1 Research Gap

A high-level study on generic mobile-rhythm games found that difficulty of each map is difficult to estimate Song et al. 2019. Typically, in VSRGs, difficulty of a map is either manually estimated by the creator / publisher of each map, or estimated through features of the map. This is consistent with what Song et al. 2019 found too, where mobile games such as Cytus, Deemo and Lanota has manual difficulty estimates by the creator of the map.

Unfortunately, estimating difficulty is non-trivial, and requires state of the art solutions for best estimates.

An example of a outdated beatmap difficulty estimation is through the density of the map, which is not robust against all approaches of maps. We illustrate with the following example.

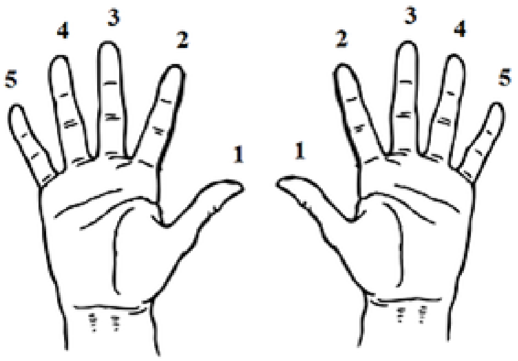

Figure 4: Annotated Finger positions. Image from Simplifying Theory n.d.

In Figure 4, rapidly alternating the index and middle finger (2, 3) is simpler than alternating the ring and pinky (4, 5). This implies that maps with similar density, but different patterns, are capable to be different in difficulty.

1.2 Difficulty Estimation Proposal

We bridge this research gap through estimation of user scores, which then can derive an auxiliary ground truth that future works of beatmap difficulty estimators can evaluate on. The following figure illustrates the motivation of this project.

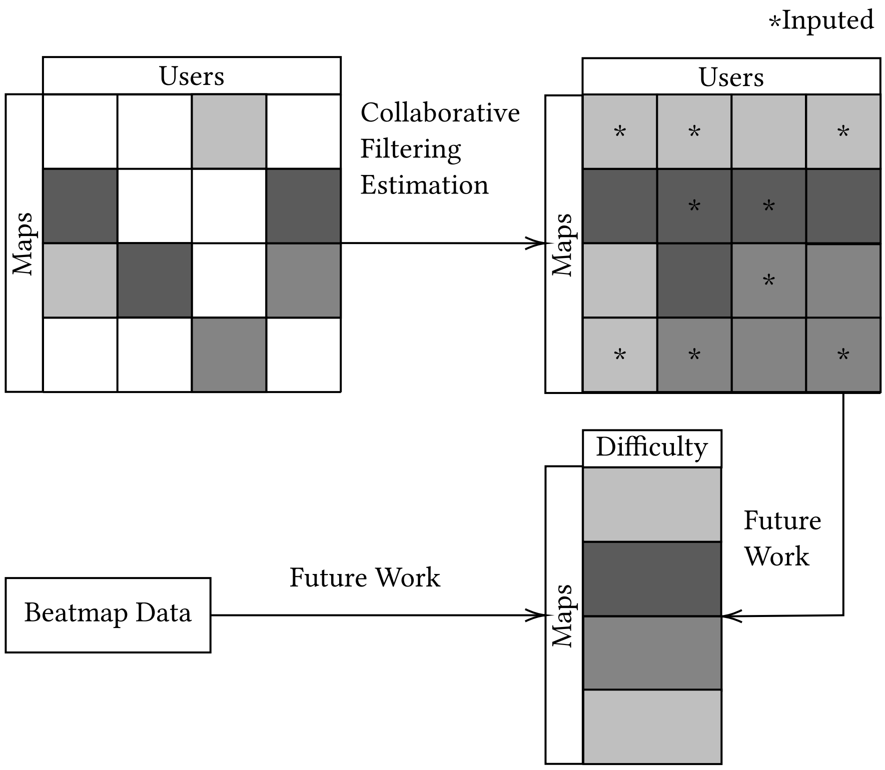

Figure 5: Motivation of this paper

The top left matrix shows the problem in estimating difficulties from current data: There are a significant number of users that haven't played **all maps**. This thus calls for a need to impute these missing scores, in order to create a more structured matrix, which can be used for a ground truth difficulty estimation of maps. This problem draws parallels to common recommendation system problems such as: predicting the next best movie to recommend to a user based on what they liked and disliked.

2 Methods

We yield data from the publicly available data warehouse of osu! *osu!* N.d. at https://data.ppy.sh/ Please do check the licence included before using.

This data consists of all best performance scores by all top 1,000 osu!mania players. A mock sample of 5 rows are shown below to illustrate its structure. Only relevant columns are shown.

	User ID	Score Created On	Map ID	Map Speed	Accuracy
1	123	2022-01-01 00:00:000	111	1	0.9
2	234	2022-02-02 00:00:000	222	0	0.8
3	345	2022-03-03 00:00:000	333	1	0.7
4	456	2022-04-01 00:00:000	444	-1	0.6
5	567	2022-05-01 00:00:000	555	0	0.5

Figure 6: Data Structure of Input Data before pre-processing with mock data

1. Map Speed is calculated from the `enabled_mods` column.
   a) -1: 0.75x Map Speed
   b) 0: 1.0x Map Speed
   c) 1: 1.5x Map Speed
2. Accuracy is calculated from the `count`___ columns. With maximum as 320.

This data is transformed from an SQL file to CSV file through string manipulation, as loading the dump file into MySQL is too slow. You may see the sub-project I created for this SQL Data Processing Repository.

Then, we processed ID inputs for our CF Model. Namely the `uid` and `mid` input. Each `uid` is a unique ID created from the user ID, and the year they set that score. Each `mid` is a unique map ID created from the map ID, and the speed the map was played at.

1. `uid` = User ID + Year of Score
2. `mid` = Map ID + Map Speed

	uid	mid	Accuracy
1	123/2022	111/1	0.9
2	234/2022	222/0	0.8
3	345/2022	333/1	0.7
4	456/2022	444/-1	0.6
5	567/2022	555/0	0.5

Figure 7: Data Structure of Input Data after pre-processing with mock data

We removed any `uid` and `mid` below 50 support to avoid under representation of any ids. Furthermore, we remove scores have have < 500,000 game score (75% accuracy). We then label encode each `uid` and `mid`.

	uid	mid	Accuracy
1	123/2022	111/1	0.9
2	234/2022	222/0	0.8
3	345/2022	333/1	0.7
4	456/2022	444/-1	0.6
5	567/2022	555/0	0.5

	uid	mid	Accuracy
1	0	0	0.9
2	1	1	0.8
3	2	2	0.7
4	3	3	0.6
5	4	4	0.5

Figure 8: Data Structure of Encoded Input Data after pre-processing with mock data

Finally, we quantile-transformed accuracy to a normal distribution before feeding into the model

2.1 Collaborative Filtering Model

We create a model inspired by the Neural Collaborative Filtering Model He et al. 2017. Our model structure is similar.

Input UID	**Input MID**
uid(1)	mid(1)
uid_emb = Embedding(EMB)	mid_emb = Embedding(EMB)
uid_emb(EMB)	mid_emb(EMB)
mul_emb = uid_emb × mid_emb	cat_emb = uid_emb CONCAT mid_emb
mul_emb(EMB)	cat_emb(EMB * 2)

mf_net	**mlp_net**
Linear(EMB, EMB)	Linear(EMB * 2, 512)
Linear(EMB, EMB)	Linear(512, 256)
	Linear(256, 128)
	Linear(128, 64)
	Linear(64, 32)
	Linear(32, MLP_EMB)

net = mf_net(mul_emb) CONCAT mlp_net(cat_emb)
net(EMB + MLP_EMB)
return TanH(Linear(EMB + MLP_EMB, 1)(net) * 2) * 5

Figure 9: Neural Collaborative Filtering Model. Values in brackets indicate the dimensions.Every Linear Layer has a ReLU and DropOut 10% following.

There are a few significant adjustments

1. We used a TanH activation to map to a normal-like distribution
2. We added scalar multiplications to scale up the resulting TanH values. This significantly decreased loss and time to convergence.

We used the Adam optimizer with a One-Cycle Cosine Annealing learning rate scheduler, which resulted a significant reduction in experiments diverging Smith & Topin 2019.

2.2 Importance of Cosine Annealing

During our experiments, a significant amount of them fail to converge within the first few epochs, which then continues to plateau forever. We found that this is mostly by chance, possibly a bad initialization of weights.

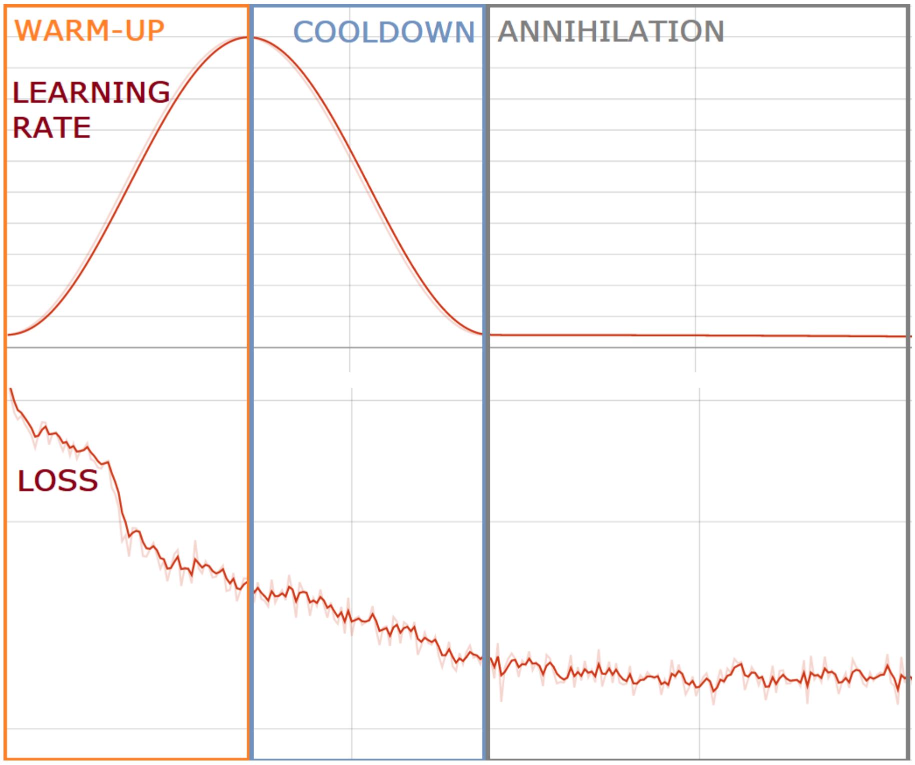

Figure 10: Optimization of V1 Model Loss w.r.t. One Cycle Cosine Annealing LR Scheduler

By using Cosine Annealing, implemented in PyTorch Paszke et al. 2019 as OneCycleLR, we found consistent convergence during the first 2 stages when it cycled through large ranges of LRs. Furthermore, it heavily sped up the learning beyond the plateau, in which the 3rd LR annihilation stage described in Figure 10, will gently tweak settled weights.

2.2.1 Suboptimal sizes of phases in Cosine Annealing

Furthermore, it's important to fine-tune the size of the first 2 phases, as too long of a cycle can waste time. In the following Figure 11, I show the effects of different sizes of the first 2 phases. The wider One Cycles experience loss plateauing due to the large LR, thus converge slower than the smaller LR

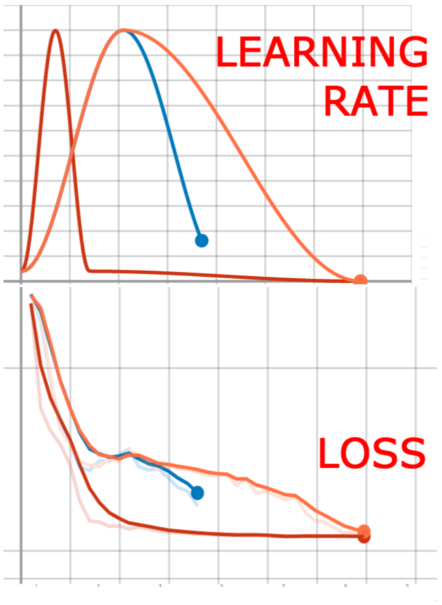

Figure 11: Optimization of V1 Model Loss w.r.t. Different One Cycle Cosine Annealing LR Schedulers

This also further elaborates why I chose a 3 phase annealing instead of the 2 phase proposed by Smith & Topin 2019.

3 Results

We trained the model for all top scores of top 1,000 mania players, with 1.27 million rows. The results are evaluated with a naive train test validation split of 80%, 10%, 10%.

Refer to Figure 9 for usage of parameters EMB, MLP_EMB.

	Parameters		Errors	
Model Name	EMB	MLP_EMB	MAE	RMSE
V1_2022_11	16	8	0.95%	1.56%

Figure 12: Models Trained and Errors

We evaluate the properties of our model in the following sections.

3.1 Model Error

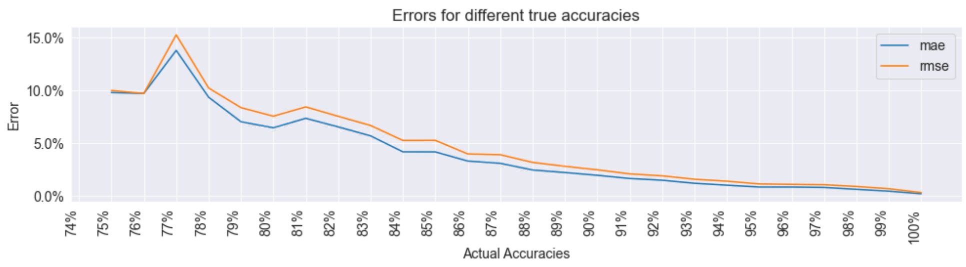

Figure 13: Error Distribution of V1 Model

Next, we do a overview analysis.

3.2 Model Analysis

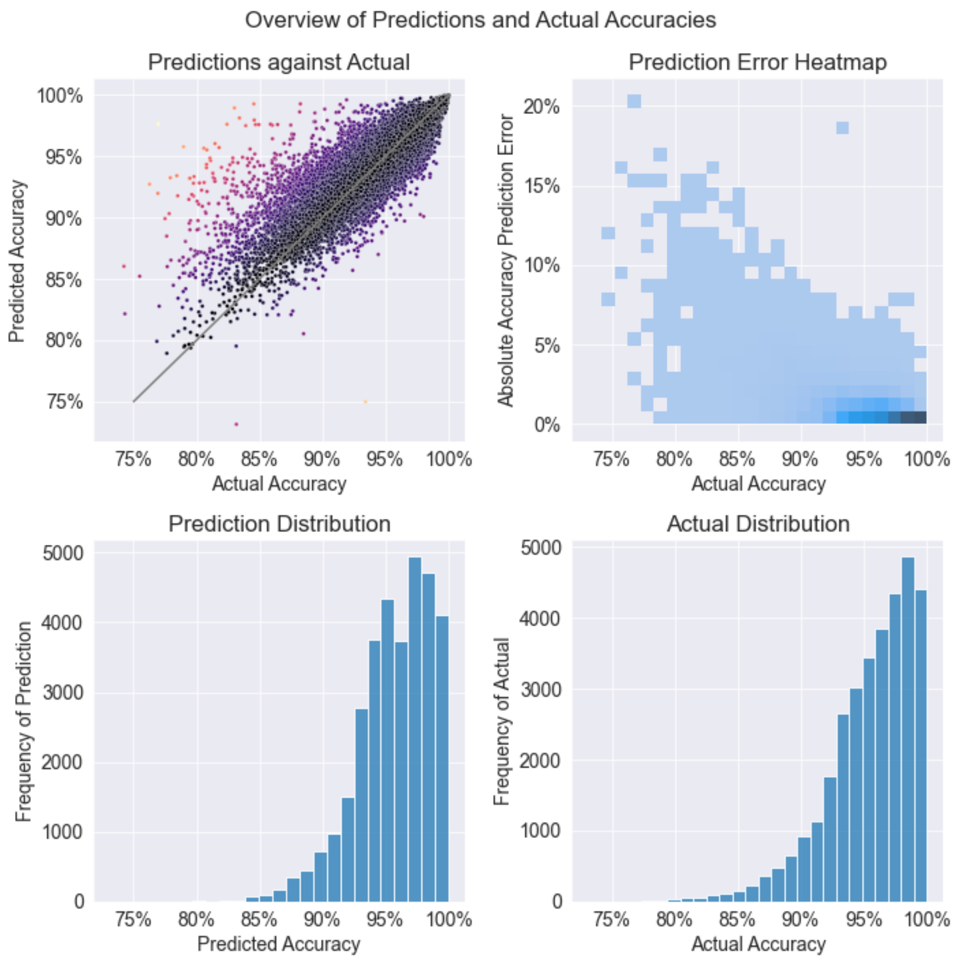

Figure 14: Analysis of V1 Model

4 Discussion

This project is version controlled in a GitHub Repository.

4.1 Model Evaluation

With a 1% for both MAE and RMSE is good in the context of rhythm games, however, it can definitely be improved more.

Previous research by AlphaOsu! *AlphaOsu!* N.d. claims that they achieved 0.74% from the same data source, but focused on a different subset of the data.

1. **Poor Performance on Lower Accuracies**. This is which is to be expected, as we have a large majority of samples from the 90% range, with minorities < 80%. We can improve them through more samples, however, it's difficult as we only have access to best scores.

2. **Overestimation of Performance**. Indicative by many outliers on the top left of the top left plot on Figure 14. This means that our model is prone to heavily overestimate performances, than to heavily underestimate performances. The root cause is unknown, and can be tackled in future work.

4.2 Assumptions & Limitations

We have made some assumptions to make the problem simpler. Those keen to further this research are to note the following.

1. **Assumption: Every score is their Best Score** The model depends on the assumption that the user's best scores are truly their "best". However, it does not always hold. E.g. If players don't perform their best on a map, due to a myriad of reasons, their best score will be lower than expected. This can cause the map's difficulty to be heavily overestimated.

2. **Assumption: Player Performance Year Partitioning** The model assumes users' performance is well partitioned by years. That is, each year, the player can be considered a separate player as their performance changes. However, a player's performance can fluctuate wildly between months. Therefore, when we aggregate over time periods of different performances, it leads to a large skill margin, where a single `uid` cannot be considered constant. On the other hand, through a more granular subset, we're likely to fail to yield enough frequent associations.

3. **Limitation: Map Features** Recall that the model only looks at scores. That means, the model will fail to ignore unwanted features from a map. For example, if a map had visual gimmicks, our model will include the difficulty of these gimmicks into account. Thus it's vital to remove effects of

these gimmicks before input if the end goal is to predict difficulty without gimmicks.

4. **Limitation: Maps of Interest have enough Players** The model assumes that maps that don't meet the minimum number of players (50) aren't important. This thus fails to predict newly published maps. This follows with the first assumption of having all scores being their best; A newly published map won't have players' best score, they are likely to improve on it.

5. **Assumption: Lower Bound Accuracy** The model uses input of > 500,000 score, that is, only a subset of the dataset is used, this assumes that any score below that is not reliable and not used in any use case.

5 Future Work

5.1 Improvements

1. **Correcting Assumptions** While it's best to correct these assumptions, the best direction is to further the research on a high-level as shown in Figure 5, before optimizing its parameters. Getting a rough pipeline out, while keeping components modular will help realise results (though poorly), faster. This can help parallelise research as each component in Figure 5 are swapable.

2. **Upgrading to New CF Models** As of writing this, many of my colleagues have suggested more advanced Neural CF architectures that may be better, for example a paper Rendle et al. 2020, which introspects our current model, and proposes corrections to its flaws. Thanks to foresight, I've intentionally separated the model logic from the rest of the pipeline, thus it's advisable to fork my repository and simply adjust the architecture of the model to achieve a new one.

3. **Magic Constants in CF Model** Not sure why adding some constant multiplications at the end of the module significantly decreases loss (from 1.4% to 1.0% MAE). It's to fix the issue whereby the network was predicting too small of a distribution, however, is there a better way to not have this as additional hyperparameters?

4. **Using a larger dataset** I didn't experiment with the larger 10k dataset as it was too heavy for my computer to convert the MySQL files through

Python. It may be of interest to do so, to expand the reach of users the size of the dataset.

5. **Quantile Transformation** Initially, quantile-transform is used to avoid biasing our training in favour of more common scores, by giving more representation to lower accuracies through transforming its distribution as a normal distribution to be fairer to all ranges of accuracies. However, upon experimentation with a simple standard scale, we found no biasing.

   Despite that, the quantile-transform allowed for a slightly lower loss, thus this paper continued with it. It may be of interest to inspect the best scaling for future works.

5.2 Next Efforts

1. **Using this model** There's many use cases for this model,
   - Predicting beatmap difficulty
   - Players clustering with respect to their skillsets
   - Forecasting match results
   - Better seeding of players
   - Improvement monitoring
2. **Accessibility of model** We have made efforts to decouple the logic of the model such that it's programmatically trivial to predict. However, there's many more ways to make this more accessible, especially with an intuitive UI
3. **Automatic learning of model** The current model must be manually downloaded, trained, and deployed, if these can be all automated, it'll be easier to maintain.

References

AlphaOsu! N.d. https://alphaosu.keytoix.vip/.

He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu & Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

osu! N.d. https://osu.ppy.sh/.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai & Soumith Chintala. 2019. Pytorch: an imperative style, high-performance deep learning library. *CoRR* abs/1912.01703. http://arxiv.org/abs/1912.01703.

Rendle, Steffen, Walid Krichene, Li Zhang & John Anderson. 2020. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th acm conference on recommender systems*, 240–248.

Simplifying Theory. N.d. *Finger notation.* [Online; accessed 11 Jan 2023]. https://www.simplifyingtheory.com/wp-content/uploads/2015/07/finger-notation.png.

Smith, Leslie N & Nicholay Topin. 2019. Super-convergence: very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, vol. 11006, 369–386.

Song, Doo Heon, Kwang Baek Kim & Jong Hee Lee. 2019. Analysis and evaluation of mobile rhythm games: game structure and playability. *International Journal of Electrical and Computer Engineering* 9(6). 5263.

Wikimedia Commons. N.d.(a). *Ddr image.* [Online; accessed 11 Jan 2023]. https://upload.wikimedia.org/wikipedia/commons/thumb/5/5b/Dance_Dance_Revolution_Extreme_arcade_machine_left_side_stage.png/300px-Dance_Dance_Revolution_Extreme_arcade_machine_left_side_stage.png.

Wikimedia Commons. N.d.(b). *Iidx image.* [Online; accessed 11 Jan 2023]. https://upload.wikimedia.org/wikipedia/commons/thumb/2/25/Beatmania_IIDX_DAOPEE_Controller.jpg/1024px-Beatmania_IIDX_DAOPEE_Controller.jpg.

Wikimedia Commons. N.d.(c). *Stepmania image.* [Online; accessed 11 Jan 2023]. https://upload.wikimedia.org/wikipedia/commons/a/ad/StepMania_5.0.5_Demo.jpg.