VSRG Difficulty Calculation with Machine Learning

Evening

2021
May

Contents

Chapter 0

Status of Journal

This journal isn't complete.

Thus, information here may be outdated. If you plan to professionally continue this project or are interested, feel free to contact me.

Part I

Relative Difficulty Estimation

Chapter 1

Preface

I've been meaning to tackle difficulty calculation for the past few months after gaining much knowledge in the field of machine learning. In this journal, I will attempt to use models to predict Relative Difficulty of a map only using data gathered from other maps.

Acknowledgements

All of the data gathered here are from players who've played Ranked/Loved maps, this approach will not work without your contributions.

Chapter 2

Relative Difficulty?

The prominent issue when comparing replays from differing maps, is that players who've played it are different. This make it hard to compare which maps are harder. Plus, even if we found the same players, there's a chance that their improvement would've changed their performance, the comparison will not hold.

Thus, finding the **absolute** difficulty is **difficult**. To find the **absolute** difficulty, we need a player to generate a dataset within days/weeks. Furthermore, the player would be the **benchmark** of the skillset ratings.

I believe that to figure out **absolute** difficulty, we firstly should tackle **relative** difficulty.

2.1 Achieving Relative Map Difficulty

Since we use different maps of different difficulties, we can make it relative via normalization.

Normalization scales any dataset from $[a, b]$ to $[0, 1]$. This allows me to compare the **relative** differences.

This is a big assumption on how difficulty works for simplicity.

2.2 Achieving Relative Player Error

Similar to how we normalize a map, we can also normalize a player error. Take for example:

1. Replay Set A on a easy map, hence better scores
2. Replay Set B on a hard map, hence worse scores

Normalizing both means to find a "common ground" where we're only interested in the harder parts of the map

2.2.1 Other Scales

There are other scaling methods

1. Standardization $(X - \mu)/\sigma$
2. Robust Scaler (Quantile Scaling)

These may prove to be better, however I haven't tested it thoroughly.

Chapter 3

Map Feature Extraction

One of the most important steps in Machine Learning is pre-processing. I'll skip the details on data extraction.

3.1 Elements

As a notation, every time-able object of the map, Normal Note, Long Note Head, Long Note Tail, etc. will be referenced as an element. All element must have a unique offset, thus Long Notes, as a whole, aren't singular elements.

3.2 Neighbours

Neighbours are simply pairing one element to another.

The Notation used is $N_{i \to j}$. Where we're interested in Neighbour relationship(s) from Column i to j.

It's not necessary to include Density or Visual Complexity. It's largely because its a dependent feature on Neighbours, and it's a bet that the model can learn this.

For a good training set, the data should be unique, independent information from the dataset.

3.2.1 Model Acceptable Inputs

Lots of Data Science Jargon here.

Before we jump into how I processed Neighbours, it's vital to understand what are acceptable inputs for the model as it supports the aggregation decisions.

This problem is a multi-input multi-output regression. Hence, for a single data point it would have **n-dimensions** for input features **m-dimensions** for output features.

Note that as convention, most models will accept Python Arrays in the shape of
`[samples, timesteps, features]`.

3.3 Neighbour Distances and Modified Distance

This is simply the offset difference between the Neighbours.

The range of this is (∞, ∞), which isn't ideal as it's unbounded. Since lower absolute values imply a greater difficulty, we transform this as such.

$$\text{Modified Difference}(x) = \Delta(x) = \frac{1}{1 + |x|/\alpha}$$

The notation for Modified Difference is Δ.

α is a correction factor. Higher α implies a smoother decreasing curve on the modification. I recommend plotting it to get a better understanding.

This function dampens the effect of large differences, which implies a low contribution to difficulty.

Polarity of the data isn't useful, hence the absolute.

3.4 Window Binning

Here, binning means to aggregate the features contained within a time-window. For example, binning $(1s, 2s)$ will capture all notes that are contained within. We then aggregate their Neighbours via various methods.

The notation for a Window is W.

Despite this losing data, there is a good reason on aggregation.

When we consider the error of a note, it's usually not just due to itself, instead the difficulty surrounding it. Aggregating it means to smooth out noise from the input and also the output. Thus, any sudden "accidental" errors on the player side can be smoothed out.

3.4.1 Window Binning and Neighbour Distance

In my code, I used Window and Threshold

- **Window** This is the Binning width in milliseconds.
- **Threshold** This how far a note should look for a neighbour.

Mathematically, threshold might not be needed, however, it's not computationally necessary to find far neighbours due to small values.

3.5 Summary of Data

After all of that, we end up with a **Jagged** 2D array for the whole map.

$$\underbrace{\begin{bmatrix} & N_{0\to 0} & N_{0\to 1} & N_{0\to 2} & \dots & N_{0\to n} & \dots & N_{1\to n} & \dots & N_{n\to n} \\ W_0 & \Delta_{0,0,0} & \Delta_{0,0,1} & \Delta_{0,0,2} & \dots & \Delta_{0,0,n} & \dots & \Delta_{0,1,n} & \dots & \Delta_{0,n,n} \\ W_1 & \Delta_{1,0,0} & \ddots & & & & & & & \vdots \\ W_2 & \Delta_{2,0,0} & & & & & & & & \vdots \\ \dots & \vdots & & & & & & & \ddots & \vdots \\ W_m & \Delta_{m,0,0} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}}_{N:I\to I}$$

$$\vdots$$

$$\underbrace{\begin{bmatrix} & N_{0\to 0} & N_{0\to 1} & N_{0\to 2} & \dots & N_{0\to n} & \dots & N_{1\to n} & \dots & N_{n\to n} \\ W_0 & \Delta_{0,0,0} & \Delta_{0,0,1} & \Delta_{0,0,2} & \dots & \Delta_{0,0,n} & \dots & \Delta_{0,1,n} & \dots & \Delta_{0,n,n} \\ W_1 & \Delta_{1,0,0} & \ddots & & & & & & & \vdots \\ W_2 & \Delta_{2,0,0} & & & & & & & & \vdots \\ \dots & \vdots & & & & & & & \ddots & \vdots \\ W_m & \Delta_{m,0,0} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \end{bmatrix}}_{N:J\to J}$$

W_i : Binned Window i

$i \in [0, m]$

m = ceiling(Length of Map/Window Width)

$N_{j\to k}$: Neighbour Column j to k

$j, k \in [0, n]$

n = Keys

$\Delta_{i,j,k}$: Distance Data respectively to their Window and Modified Distance

$N : I \to J$: Neighbour Type I to Neighbour Type J

$\{I, ..., J\}$: {Note, Long Note Head, Long Note Tail, ...}

This is a large array, however, it usually only takes up around 1 million bytes, which is around 1MB per map. It's just that the processing of this data is slower.

$N : I \to J$ is a notation for different types of chart elements. We have additional dimensions for each distinct combination. Note that despite the above equation showing multiple matrices/arrays, they are appended formed as a 2D array.

3.5.1 Jagged Arrays

Jagged arrays are arrays that you can't make into a rectangular array or multi-dimensional arrays.

Take for example:

$$[1, 2], [1, 2, 3], [4, 5] \to \begin{bmatrix} 1 & 2 & \\ 1 & 2 & 3 \\ 4 & 5 & \end{bmatrix}$$

Because of the missing values, this isn't a well-defined multi-dimensional array. We could either replace it with a defined value or a undefined one.

$$[1,2],[1,2,3],[4,5] \rightarrow \begin{bmatrix} 1 & 2 & \text{NaN} \\ 1 & 2 & 3 \\ 4 & 5 & \text{NaN} \end{bmatrix}$$

$$[1,2],[1,2,3],[4,5] \rightarrow \begin{bmatrix} 1 & 2 & 0 \\ 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$$

Both methods will work, however note the consequences of the default values.

```
np.nanmin([1,2,np.nan]) == 1
np.nanmin([1,2,0]) == 0
```

3.5.2 Consequence of Jagged Arrays

Jagged arrays cannot be formed into multi-dimensional arrays, thus incompatible with `np.ndarray`. In order to make these equal, we can either

- Pad: Add zeros such that all arrays are equal in length
- Aggregate: Mean, Median, Max, etc. Anything that reduces a list to a value.

Padding is usually not feasible unless point-wise data is absolutely necessary. The dimensionality of the data will increase due to this and might not fit in the memory, thus infeasible.

Aggregation will **reduce dimensionality**, however, we can at least control its dimensionality expansion.

3.6 Aggregation of Window Binned Distances

Since we're window binning, that means, for all $\Delta_{i \to j}$ located in that window, we group them into a set. Note that modified distances cannot be negative.

$$\Delta_{k,i,j} = \{\Delta_{i \to j} : i \in W_k\}$$

There are several ways to aggregate Δ.

1. Maximum Element
2. Mean
3. Variance
4. Skewness
5. Kurtosis

These are the methods I used. One could exhaustively try others and perform PCA to reduce dimensionality. A precaution is to account for NaNs due to Neighbour Thresholding.

After aggregation, we will yield

$$\Delta_{k,i,j} = \left\{ \begin{array}{l} \text{Max}(\{\Delta_{i\to j} : i \in W_k\}), \\ \text{Mean}(\{\Delta_{i\to j} : i \in W_k\}), \\ \text{Variance}(\{\Delta_{i\to j} : i \in W_k\}), \\ \text{Skewness}(\{\Delta_{i\to j} : i \in W_k\}), \\ \text{Kurtosis}(\{\Delta_{i\to j} : i \in W_k\}) \end{array} \right\}$$

This means, for each Δ, we have an additional axis of size 5.

3.6.1 Shape of Input

The final size of the input for a single map would be

$$\text{Windows} \times \underbrace{\text{Keys} \times \text{Keys} \times \text{Types}}_{Combinations} \times \text{Aggregations}$$

Windows is the first axis and the rest is on the second.

Chapter 4

Player Error Extraction

For this, I extracted all player errors using a custom algorithm in conjunction with the `osrparse` Python library.

4.1 Summary of Data

The data we extract are simply the hit errors for each element in milliseconds.

For each map, I can extract a few replays. Firstly, we Aggregate by **Absolute Averaging**. Then we then find the **relative** error ϵ through normalization.

$$\begin{bmatrix} & \text{Player}_0 & \text{Player}_1 & \dots & \text{Player}_n \\ \text{Element}_0 & \epsilon_{0,0} & \epsilon_{0,1} & \cdots & \epsilon_{0,n} \\ \text{Element}_1 & \epsilon_{1,0} & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ \text{Element}_m & \epsilon_{m,0} & \cdots & \cdots & \epsilon_{m,n} \end{bmatrix} \xrightarrow[\text{Mean}(|\epsilon|)]{\textit{Aggregate}} \begin{bmatrix} & \text{Player}_{\text{Agg}} \\ \text{Element}_0 & \epsilon_{0,\text{Agg}} \\ \text{Element}_1 & \epsilon_{1,\text{Agg}} \\ \vdots & \vdots \\ \text{Element}_m & \epsilon_{m,\text{Agg}} \end{bmatrix}$$

The absolute is required to ensure that late and early errors don't cancel.

Chapter 5

Model Training

5.1 XGB Regression

XGB stands for Xtreme Gradient Boosting. It's a popular set of ensemble models to use for data scientists. I'm interested in the `XGBRegressor` in particular.

I won't go into detail on how it works nor its fine-tuning as I'm not knowledgeable in this.

5.2 Ways of Feeding the Sample

THIS IS A VERY IMPORTANT SECTION

5.2.1 Linear Feeding (Reinforcement)

Initially, I fit the model one map by one, which is a reinforcement learning technique, but one major issue is that when you reinforce, you tend to forget the older data.

Take for example, I have Map A, B, C.

I feed A, B, C respectively, the model will fit to C very well, but A slightly worse.

Consider that example but all the way to Z. A will not tend to be a horrible predictor, but instead, follow the information driven by the last few runs.

5.2.2 Parallel Feeding

With this, we just concatenate the whole thing together and feed it at once. This works fine since there's no many-to-one relationship in `XGBRegressor`.

With this, the model improved **a lot**.

5.3 Evaluation

There are 2 main criterion for a good estimation.

1. Identify the peaks and troughs
2. Match the correct range

Identifying the peaks and troughs is a good indication that the model is working fine. Being able to show the peaks and troughs gives a good idea of map spikes.

Estimation should be as accurate as possible, however, if it's not, we should at least estimate spikes and troughs correctly as those are important features.

I felt that writing too specific of a loss function wouldn't work out too well, Mean Squared Error Loss works fine.

Chapter 6

Results

6.1 Problem With Relativity

One main issue that I had was that, we assumed that the maps difficulties usually revolve around a certain range, that makes the normalization consistent.

An interesting thing about my model is that it works across maps