ppshift machine learning

In this document, we will be discussing methods of obtaining a credible way of classifying difficulty in VSRG maps. We will first establish what makes a map difficult, then we build from there!

Part I
Preface

1 Define Difficulty

What makes a map difficult, what is a difficult map? Could it be the following? The map was difficult because of ...

1. Failing
2. Combo Breaks
3. High Stamina Requirement
4. Low Accuracy

We discuss all of these scenarios and we will choose one to tackle, possibly integrate the other options into our calculations in the future.

Failing The most significant way that we can readily control if players fail is via **Health Drain** in which most VSRGs will implement. However, this value is inconsistent and will not provide useful information on higher **Health Drain** values due to lack of players passing certain maps.

Combo Breaks Combo Breaks analysis is another method that isn't consistent, whereby chokes can be random, creating too much noise on higher skill plays. Combo breaks mainly can only determine the **hardest** points on the map, it doesn't depict a difficulty cure.

High Stamina Requirements While stamina is a good way to look at difficulty, it can readily be derived from accuracy, which is conveniently what we'll be looking at next

Low Accuracy This is the best way to look at difficulty, because not only it gives us a figure, it tells us the story and correlation between **accuracy** and **patterning**. This will be the main focus of the document.

1.1 Comparing Accuracy in Replays to Patterns

Details aside, how can we describe the impact of patterns on replays, what story does the replay tell us about the pattern?

Consider this...

1. $Player_1$ plays $Pattern_A$ and $Pattern_B$
2. $Player_1$ achieves $Accuracy_A > Accuracy_B$
3. Considering $Accuracy_A$ and $Accuracy_B$ are independent events
4. We deduce $Pattern_A < Pattern_B$ in difficulty

It is a simple idea to pitch, but we will have to dive into more details on how we can "teach" a machine this concept and "learn" from it!

2 Machine Learning

A tangent from this article, I will briefly talk about what is machine learning.

Making a Prediction Machine Learning is all about making a prediction, by looking at already curated data. *We show the someone 100 pictures of rabbits and explained to them that they are rabbits, can they tell if the next picture of an animal is a rabbit?*

Same idea we have here, we show the machine hundred of beatmaps, we tell the machine how difficult all of them are (according to score regression). Can the machine predict the difficulty of the next one? The answer is yes, but it won't be accurate!

2.1 Valid input

Think about a neural network, you've most likely seen one, it's circle(s) connected to more circle(s), in the end, there will be circle(s) that tell you something. Each circle represents a **neuron**.

Each neuron holds a numeric value (between 0 and 1), we need to put a value in each of these neurons. Ignoring Timing Points and Scroll Speed Changes, how do we squeeze a beatmap into these neurons?

Hit Object Neurons Remember that a neuron will only work with numeric values, what do you want to have in the neuron that represents a Hit Object? Putting either the **offset** or **column** will not work because both are vital!

Column Neurons It is a possible idea, to have a neuron for **each millisecond**, then put **column value** on those neurons that match the **offset**. We run into a glaring issue where input neurons will stretch to the **hundred thousands**, computational power required on this scale would be too high. If we decide to **bin** the offsets (binning is the act of grouping things together to reduce the number of fields) to **100ms** steps, it'll still be in the thousands.

2.1.1 Subdivide the Issue

Does it have to be the whole map?

It doesn't! That's what we are doing for this research, we don't look at the whole map, we look at parts of it, the inner mechanisms, the **patterns**! Sure, even though the whole map must be present to calculate difficulty but if we break it into **2 distinct procedures**, it's easier to visualize and control the system.

1. Patterns to Difficulties
2. Difficulties to Map Rating

Patterns to Difficulties In the following sections we will be discussing how we can task the machine to learn what patterns are easy and which are harder

Difficulties to Map Rating Using the **difficulty** values defined previously, we can find out how to use that to reliably give the map a rating. Do we rate it by the average, maximum or median? We will discuss those later.

2.2 Expected output

Just like the example in **valid inputs**, we have to tell the machine that **that picture is of a cat**, else it doesn't know what is and what isn't.

In this case, we can relate back to *Accuracy* (Player), where it is the most reliable source of difficulty rating as explained earlier.

One good source of this data is to grab replays (not scores!), because we can then analyse small parts of it in order to match with the input.

2.2.1 Differing and Multiple Accuracies

However, there are two problems:

Multiple Players There will be multiple replays, there isn't a replay that is all-encompassing (it'll be too biased!). So we need to create a middle-ground for these data.

Different Players Not everyone will perform similarly for every beatmap, there will be discrepancies. We can't just take the accuracy as a raw value and run a function through them, we will have to make use of relative accuracies instead of raw accuracies before finding the "ideal" replay.

2.3 Recap

To sum it up

1. Grab beatmap from server
2. Grab replay from server
3. Convert beatmap into simpler bite-sized patterns
4. Convert replay to a list of accuracies for each note
5. For each pattern, there's an expected accuracy
6. Teach that concept to the machine and create a model
7. Predict accuracies with the model
8. Give the map a rating according to the expected accuracies