Star Rating & Score, does it work well enough to supplement each other?

Evening

Goal

If Star Rating (SR) perfectly represents difficulty, we would have a perfect correlation between SR and Score achieved by players. However, this is not true, and we would like to find out what beatmaps are so-called “overrated” and “underrated”.

Procedures

Processes	**Details**
Data Retrieval	Get all data via osu!API.
Thinning Data	Removal of “extreme and unnecessary” data points (Refer to **Thinning Data**).
Selection of Players and Maps	Stratified sampling of certain player groups. Then using INNER JOIN with available map data.
Removal of Invalid Records	Removal of: - Double Time Scores - Loved Beatmap Scores
Construction of Isontonic Regression	Usage of Weka’s Isotonic Regression Node.
Construction of Polynomial Regression	Usage of Polynomial Regression Learner of Max Degree (3) Node.
Calculation of Prediction Errors	Calculate both prediction errors with polarity.
Output	

Retrieval of players to analyze

Data Retrieval

We retrieve all maps starting from 2015 and retrieve scores from each of these maps for a total of **4621 maps**.

Within each of these maps, we gather results for the **top 100 scores** (limited by osu!API).

Thinning Data

As we are more interested in certain star rating groups, we ignore all data that have:

$$Star\ Rating < 3.0\ ||\ Star\ Rating > 10.0$$

We purposely ignored anything below 3.0 Star Rating as they do not provide interesting insight and doesn't affect results too much. All Top 100 Scores should be close to 1,000,000 in Score in that category.

Selection of Players and Maps

We want to analyze are players that currently have scores in **ranked and loved maps**. This is to ensure that we don't analyze "dead accounts". This has proven to be hard to get a **random sample** as it's hard to avoid "dead accounts" even with recent plays (the API only retrieves a 24h backlog), this method will trim out too many players.

To get the best possible **sample**, we will look at players that have been playing the recent maps, from there, we will generate a **purposive sample**.

By aggregating and counting the number of records the players have in the selected maps, we selected the **top 10%** of the players and **INNER JOIN** (SQL) with the current data.

Removal of Invalid Records

Double Time Records

osu! doesn't provide an updated Star Rating value when you grab it from the API, so it's impossible to calculate.

Loved Records

Loved records greatly skew the data due to some of those maps having little to no player interaction. The regression learners work best if there are a lot of records to estimate to.

Construction of Isotonic Regression Learner

As of **Full Tablet**'s suggestion[1], we will use an Isotonic Regression alongside the Polynomial Regression.

Construction of Polynomial Regression Learner

Using KNIME's Polynomial Regression Learner, we use a **maximum polynomial degree** of 3 as the data points seems to suit it quite well.

Calculation of Prediction Errors

$$\Delta Star\ Rating_{polynomial} = Star\ Rating_{polynomial} - Star\ Rating_{original}$$

$$\Delta Star\ Rating_{isotonic} = Star\ Rating_{isotonic} - Star\ Rating_{original}$$

Output

I think it's best to output the document as .csv and inside Google Sheets[2] as they are easy to view and easy to manipulate by both spectrums of users. PDF seems to be a bit too formal and I have had a lot of trouble creating reports on KNIME, so I'll just avoid that.

[1] https://osu.ppy.sh/community/forums/topics/835809?start=6890991
[2] https://docs.google.com/spreadsheets/d/1XIFd0A34YlQaR_FUfOYt0gAHCfyltCkkndjiTS7ITs0/edit#gid=0

Personal Thoughts

Is Star Rating a good indication of difficulty?

I think Star Rating does an alright job at handling most maps. However, it starts to produce very inconsistent **alpha** (inversely proportionate to star rating) at maps above 5.0 Star Rating. Since star rating is very closely related to maximum density of the chart itself, it's obvious that it will follow a similar trend to the actual difficulty of the chart.

Trends about the outliers

Negative Delta maps

Usually these maps (data points) have characteristics that make it avoid this **"Star Rating Inflation"** by affecting difficulty increasing density. Here are the top 2 factors that contribute to this:

- Difficult SVs
    - Intro + Dualive [Contemplate + Dive]
    - perthed again (yambabom remix) [Extreme]
    - Acorn [Nuts]
    - and much more…
- Unorthodox Patterning
    - The Impulsive State [Entropy_]
    - Finger VIP [Difficult]
    - Ren Ren Ai Ai Cir Cir Cula Cula Tion Tion [Koi Koi]

Positive Delta maps

On the contrary, maps that have high delta because:

- Easier Patterning for high density sections (Jumptrills/Quads)
    - Galaxy Collapse [Cataclysmic Hypernova]
    - iLLness LiLin [HEAVENLY]
    - ETERNAL DRAIN [Eternal]
- Lack of difficult SVs in general

Fix Star Rating?

I think the best way to tackle star rating calculation is to generate a universal formula using deep learning algorithms, but it'll be a long way from now if no one does it. If it does succeed, this doesn't mean we throw away the idea of the current one, it has proven to be "alright", and if anything goes wrong, at least we can still count on what we have right now.

Sample Results

keys	artist	title	version	iso star_rating	poly star_rating	iso star_rating delta	poly star_rating delta	star_rating
9	xi	Happy End of the World	9K Collab Hard	4.622	4.274	+1.380	+1.032	3.241
8	Warak	REANIMATE	Reanimated obj. Kamikaze	5.100	5.047	+1.361	+1.308	3.739
4	TSUNKU	Batting Show Perfect Version	Batter Up!	4.272	4.254	+1.262	+1.244	3.009
9	DJ Mashiro	Prismatic Lollipops	Lv.20	5.224	4.564	+1.247	+0.588	3.977
8	Warak	REANIMATE	Hyper	4.425	4.164	+1.197	+0.936	3.228
9	Horie Yui	PRESENTER (TV Size)	yoshilove's EX	4.645	4.570	+1.165	+1.090	3.480
4	Ekcle	The Impulsive State	Entropy_	4.705	4.523	+1.162	+0.980	3.544
4	So Sus + Konka	Acorn	Hard	4.157	4.123	+1.150	+1.116	3.007
8	Ti7	Love Maker	8K MX	4.467	4.344	+1.148	+1.026	3.318
7	Orange Heart(cv:Honda Mariko) Neptune(cv:Tanaka Rie)	Mousou Katharsis	_UJ's MX	4.173	4.132	+1.104	+1.063	3.069

Table 1. Top 10 Results when sorted by iso_star_rating_delta

keys	artist	title	version	iso star_rating	poly star_rating	iso star_rating delta	poly star_rating delta	star_rating
8	Warak	REANIMATE	Reanimated obj. Kamikaze	5.100	5.047	+1.361	+1.308	3.739
4	TSUNKU	Batting Show Perfect Version	Batter Up!	4.272	4.254	+1.262	+1.244	3.009
4	So Sus + Konka	Acorn	Hard	4.157	4.123	+1.150	+1.116	3.007
9	Horie Yui	PRESENTER (TV Size)	yoshilove's EX	4.645	4.570	+1.165	+1.090	3.480
7	Gotou Mai	Funwari Fuwari	yoshilove's Uryu~	4.042	4.111	+1.013	+1.082	3.029
7	Orange Heart(cv:Honda Mariko) Neptune(cv:Tanaka Rie)	Mousou Katharsis	_UJ's MX	4.173	4.132	+1.104	+1.063	3.069
9	xi	Happy End of the World	9K Collab Hard	4.622	4.274	+1.380	+1.032	3.241
8	Ti7	Love Maker	8K MX	4.467	4.344	+1.148	+1.026	3.318
4	Culprate	Finger VIP	Difficult	4.083	4.170	+0.913	+1.001	3.169
4	sakuraburst	forest of the spirits	insane	4.196	4.179	+1.009	+0.993	3.187

Table 2. Top 10 Results when sorted by poly_star_rating_delta

Annex

Figure 1. Grabs Recent maps from osu!API

```
# Retrieves Maps from osu API
import requests
import json

parameters = {"k": "REDACTED",
              "since": "2018-06-30 11:00:12",
              "m": "3",
              "limit": "500"}

response = requests.get("https://osu.ppy.sh/api/get_beatmaps", params=parameters)
print(response.status_code)

with open('api_responses/response10.json', 'w', encoding='utf-8') as outfile:
    json.dump(response.content.decode('utf-8'), outfile)
```

Figure 2. Grabs specific map data via osu!API

```
# Converts JSON to CSV
import json
import csv

with open('api_responses/out.csv', 'w', newline='') as respcsv:
    csv_file = csv.writer(respcsv)

    for x in range(0, 11):
        with open('api_responses/response' + str(x) + '.json', encoding='utf-8') as resp:
            respJson = json.loads(json.load(resp))

        for map in respJson:
            csv_file.writerow([map["approved_date"],
                               map["beatmap_id"],
                               map["beatmapset_id"],
                               map["difficultyrating"],
                               map["diff_size"],
                               map["hit_length"],
                               map["favourite_count"],
                               map["playcount"],
                               map["passcount"],
                               map["artist"],
                               map["title"],
                               map["version"]])
```

Figure 3. Grabs Top 100 Player Scores from each map

```
# Retrieves Maps from osu API
import requests
import json
import csv
import time

key = open('../../osu_api_key.txt', 'r').read()
ids = open('documents/Selected Maps.csv', 'r').read().split('\n')

for id in ids:
    parameters = {"k": key,
                  "b": id,
                  "m": "3",
                  "limit": "1"}

    response = requests.get("https://osu.ppy.sh/api/get_scores", params=parameters)
    print(response.status_code)

    # Writes to json
    with open('api_responses/map_scores/json/' + id + '.json', 'w', encoding='utf-8') as outfile:
        json.dump(response.content.decode('utf-8'), outfile)

    # Writes to csv
    with open('api_responses/map_scores/csv/' + id + '.csv', 'w', newline='') as respcsv:
        csv_file = csv.writer(respcsv)
        for score in json.loads(response.content):
            csv_file.writerow([score["score_id"],
                               score["score"],
                               score["username"],
                               score["count300"],
                               score["count100"],
                               score["count50"],
                               score["countmiss"],
                               score["maxcombo"],
                               score["enabled_mods"],
                               score["user_id"],
                               score["date"],
                               score["rank"],
                               score["pp"]])

    time.sleep(1) # Prevent spam requests
```

Figure 4. KNIME Data Mining Workflow

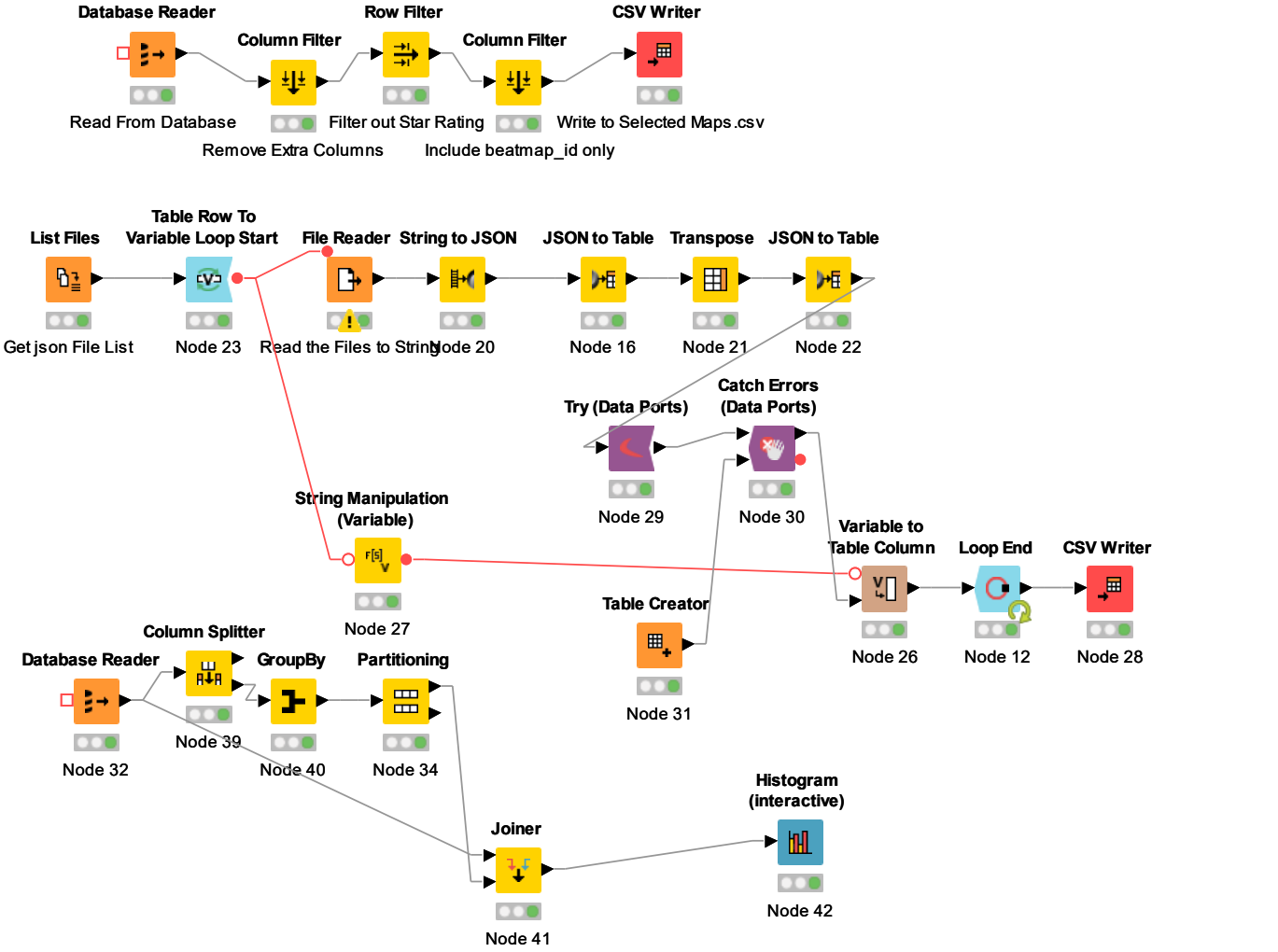

Figure 5. KNIME Data Analysis Workflow