

The heartfelt story of me building a League of Legends win interpreter for hard-stuck Silver II players (100% not me)

——INTRODUCTION: Why?

League of Legends is a 5v5 team-based game where each player can select a champion (out of 154 options) and role (top, jungle, mid, adc/bot, support/bot). The game's competitive scene is vibrant and global, with a highly *not-toxic* ranked match-making option where players can compete amongst themselves and climb the "ELO ladder" to rank up. I wanted to see if a machine learning model could answer the common question players might have about their matches: "what did I do this game that helped or hurt my chances of winning, and how much did each thing that happened matter?"

Before I go any further, please try it out yourself @ <http://liamisaacs.com/league> – note that the algorithms sweat and grind to compute, so in the beta stages the algo does kind of crash if more than 2 people are using it 🤖

This blog post will be more 🌟journey-oriented🌟 and less in the nitty-gritty with 🐍robot snek🐍 (python3, I prefer to refer to it this way tbh), but please feel free to look at the code here <https://github.com/yi-ye-zhi-qiu/metis-project3> .

——THE CODING PROCESS CONCEPTUALLY

In terms of making stuff, I like to start at the end, then go to the beginning, kind of oscillate between those two quickly, end up in the middle, then walk a straight path to the finish line. Not everyone has this non-linear process, and it doesn't help in reconstructing linear narratives, but it does make my story more legible and relate-able to people who themselves are working on creating stuff. In that frame of mind, let's start at the end.

Note to self: Start by ignoring machine learning because that's too hard & figure out how that will fit in later

Machine-learning algorithms come from patterns, but they are taught by data, and data is created by a human, at least ideologically speaking – in this vein, algorithms inherit bias and variance, prejudice and perspective. So, it doesn't make a lot of sense to try to "figure out the data" by running models on it; that will just end up justifying opinion without any self-awareness of what that opinion really is.

I thought it'd make more sense if I tried to (with sloppy code) "sketch" an image of what I wanted before ironing it out with algorithmic finesse.

——CODE SLOPS

- Code slop 1: player data

The first slop of code is getting a player's "data". The technique here is to dump data in a dictionary into an array, like

```
```python
from riotwatcher import LolWatcher, ApiError
class game_info():
 def __init__(self, api_key, name, region, game_id):
 self.api_key = api_key
 self.name = name
 self.region = region
 self.game_id = game_id
 watcher = LolWatcher(self.api_key)
 self.user = watcher.summoner.by_name(region, name)

 def match_data(self):
 watcher = LolWatcher(self.api_key)
 self.matches =
watcher.match.matchlist_by_account(self.region,
self.user['accountId'])
 self.this_match = watcher.match.by_id(self.region,
self.game_id)

 def get_data():
 n = []
 for row in m['participants']: #for player in match
 m_row = {}
 m_row['kills'] = row['stats']['kills']

 #so on and so forth for all the data you want
```
```

We can then do something like

```
```python
@app.route('/league', methods=["GET", "POST"])
def riot_api_call():
 name = 'Doublelift'
 region = 'na1'

 user = watcher.summoner.by_name(region, name)
 matches = watcher.match.matchlist_by_account(region,
user['accountId'])
 game_ids = []

 game_amount = 3
 for i in range(game_amount):
 game_ids.append(matches['matches'][i]['gameId'])

 dfs = {}
 for game_id in game_ids:
 dfs[game_id] = game_info(api_key, name, region,
game_id).match_data()
```
```

...

- Code slop 2: displaying player data

Ignoring how we will apply machine learning, the next stage is how we want to show this stuff. Knowing that will help guide how we can apply an algorithm.

After we create a form page like this

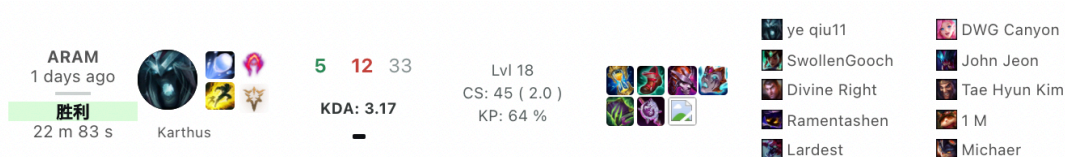
```
```python
<form id="riot-api-form" action="{{ url_for('riot_api_call') }}"
method="post">
 <input dataset placeholder="Doublelift..." id="input-box"
class="presearch-input" name="presearch-input" type="text">
</form>
```
```

And let's say that riot_api_call loads the match data onto a separate html

```
```python
def riot_api_call():
 #
 return render_template('public/separate.html', dfs = dfs)
```
```


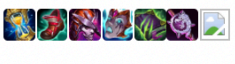

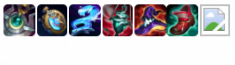





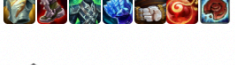







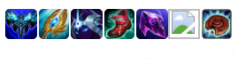

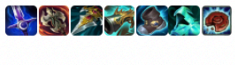



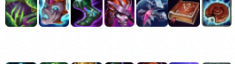

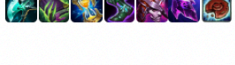
Where do we go from there?

——THE WEB APP: How do Web Apps work?



Web design focuses on reconstruction. When we design a web app for an experience taken from somewhere else, we re-tell the story of that experience; for instance, ordering Papa Johns through their app is just a brief version of the old days where you had to GO and SEE Papa John 🤔

In our context, a League of Legends Web App for displaying a game statistic is re-telling what happened that game, without having to play it. OP.gg is really interesting because it splits the re-told story of a game as 2 flavors: (1) a brief summary; and (2) “advanced stats” or in-depth review.

| 胜利 | | Tier | 1-10 | KDA | 英雄伤害 | 眼 | CS | 装备 |
|--|--------------|----------|-------------|-----------------------|------------------|----------------------|-----------------|---|
|  | ye qiu11 | no judge | 1-10
1st | 5 12 33
3.17, 64% | 18469 | 0
vis
score: 0 | 45
2.0/min. |  |
|  | SwollenGooch | no judge | 1-10
1st | 4 7 40
6.29, 74% | 12343 | 0
vis
score: 0 | 18
0.8/min. |  |
|  | Divine Right | no judge | 1-10
1st | 30 11 23
4.82, 89% | 63169 | 0
vis
score: 0 | 81
3.5/min. |  |
|  | Ramentashen | no judge | 1-10
1st | 13 9 27
4.44, 67% | 23778 | 0
vis
score: 0 | 102
4.5/min. |  |
|  | Lardest | no judge | 1-10
1st | 7 9 28
3.89, 59% | 17571 | 0
vis
score: 0 | 44
1.9/min. |  |
| <div> <div>  0  0  4 </div> <div> <div>Kills</div> <div>59</div> </div> <div> <div>Gold</div> <div>79689</div> </div> </div> <div> <div>  48  0  2 </div> <div> <div>Gold</div> <div>76196</div> </div> </div> | | | | | | | | |
| 失败 | | Tier | lr score | KDA | dmg to champions | wards | CS | items |
|  | DWG Canyon | 别判断人 | 1-10
1st | 0 12 34
2.83, 70% | 11450 | 0
vis
score: 0 | 28
1.2/min. |  |
|  | John Jeon | 别判断人 | 1-10
1st | 8 13 15
1.77, 47% | 15681 | 0
vis
score: 0 | 34
1.5/min. |  |
|  | Tae Hyun Kim | 别判断人 | 1-10
1st | 16 12 14
2.5, 62% | 32472 | 0
vis
score: 0 | 123
5.4/min. |  |
|  | 1 M | 别判断人 | 1-10
1st | 14 6 21
5.83, 72% | 34589 | 0
vis
score: 0 | 45
2.0/min. |  |
|  | Michaer | 别判断人 | 1-10
1st | 10 17 18
1.65, 58% | 28829 | 0
vis
score: 0 | 33
1.4/min. |  |

In taking inspiration from op.gg, I chose to display advanced stats very much the same way.

In our re-telling of the narrative, I chose not to focus on Tier or a score of 1st-10th, both of which OP.gg has. On one hand, it makes sense that Tier is a good intuition for win/loss, since if you are matched up against a player 5x your rank, you will most likely lose. On the other hand, knowing that is not always a helpful piece of information: more often than not, it's used as "I lost because this player was Diamond and I'm Silver", not as "I lost because the Diamond player did ___ to beat me." An algorithmic linear regression (lr) score of 1-10 based off features of the game has the same effect, you reflect on yourself thinking: "Oh I got 1st, I do not know why but I will just do more of that" or "I got last what a useless game". Both of these factors are judgements that do not help you critique yourself.

There are two glaring problems with this sort of analysis: (1) you reduce your "story of a game" to KDA, dmg to champions, wards, CS and items; and (2) you have no idea how much these features actually matter – that's what the linear regression score on OP.gg tries to do,

but since it comes across as a black-box model with no coefficients, it's not entirely doing that.

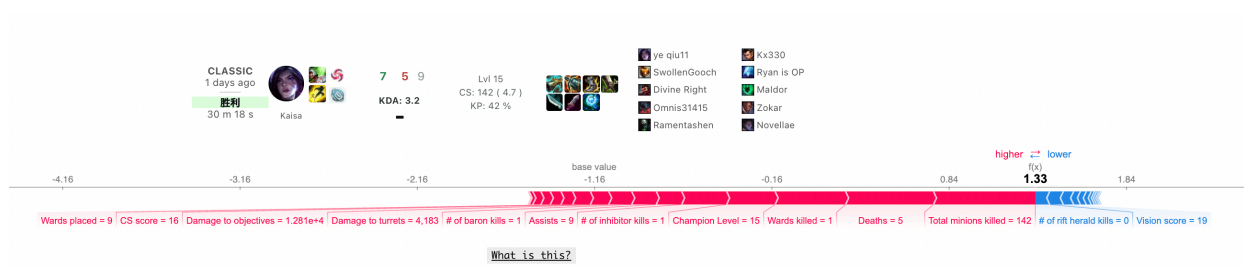
——THE MACHINE LEARNING MODEL: and the struggles of not having data “judge” the player

I don't say that liam.gg is a real answer to this, but it might be a good way of re-framing game data as vehicles of self-improvement. The idea behind the algorithm I'm about to show you isn't efficiency, elegance, it's just the humble desire to make a machine learning algorithm that isn't about judging people for no reason.

Having just made another app that judges movies for no reason <http://liamisaacs.com/liamometer>, I started off on a bit of the wrong foot. There's some raunchy fun about accidentally writing an algorithm that for weeks keeps scoring 50 shades of grey abnormally high, but this is because you immediately think of “how stupid this algorithm is being, how come it doesn't rate Stephenie Meyer's ~~crazy sexual fantasies~~ Twilight books' movies as high?” But there are plenty of algorithms in society which use black-box (or linear regressions but with redacted coefficients) allure to justify ~~random~~ systemic prejudices, like misogyny, like racism. In the context of giving a player feedback, giving just a black-box result is not the answer here: you cannot replace a murk with a “murk backed up by data”. This means no XGBoost, no RandomForest.

The immediate attraction here is to use a logistic regression model, one that mathematically can deduce what and by how much a set of features mattered to a given result (like win/loss). Before doing that, again arises this cold-start problem: what's the use in that if the data is not visual, pliable and transparent?

I was not paid by Microsoft to say this but SHAP feels like the answer here. The ability to display a force plot of features, to see what's pushing the model's result back & forth, is what will be helpful in telling a player “I'm not saying do this or do that but this is what might have done that.”



Now we can see the algorithm's opinion about this game. 5 deaths? Kind of is good not to die so much this game. Maybe should've taken rift herald, or bought less wards. Looks like vision control might have been valuable.

—ENDING WITH THE BEGINNING: Data Collection

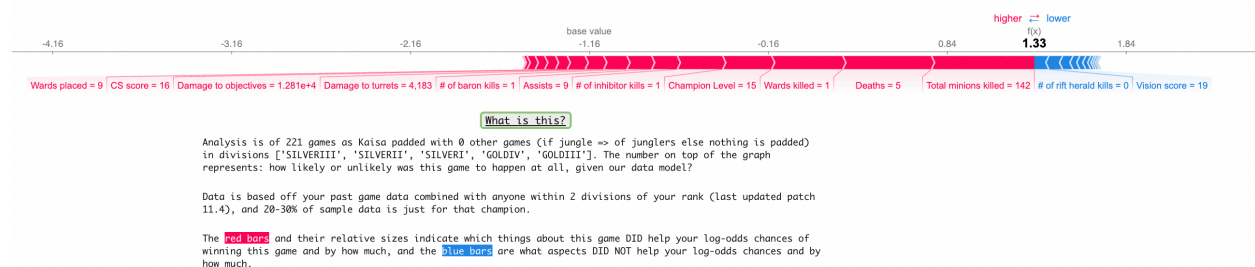
Having started at the end, let's end at the beginning. I ended this project doing data collection. Yes, that's right. Sorry it's possible I might still be insecure about this, so I have to self-affirm and flex 🐼🐼

The input to our model is crucially important. Although I built it initially just with some random dataset with 100k Korean challenger games, I quickly realized: (1) there's no way games at the highest level of play will be helpful to a person in NA in Silver II; and (2) sure, 100k games is great, but how much do I gotta over-sample user games to really personalize these results? 🐟 (I like this fish it looks like it's sighing)

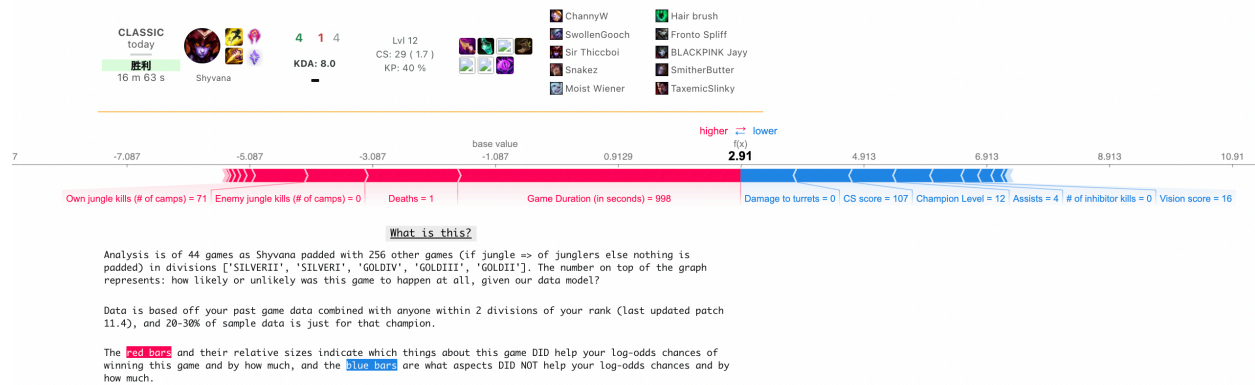
After talking with Vinny, one of my awesome professors (since Liam's stats ... not so good) and I thought some kind of collaborative filtering would be helpful. I walked away thinking "what the hell is collaborative filtering", but it means grouping the player with players like them. So, this goes all the way back to the start of this essay when we're talking about League of Legends is 5v5. You can pick one of 154 champions and a role (top/jungle/mid/adc/support). You are ranked in a division by tier (IRONIV, SILVERII, for example). These are all ripe categories for filtering, but need a healthy dataset that has that information.

I will spare you the details, but I spent around a week constantly running some code to get data (~1000 games) for each division by tier. I uploaded the dataset to Kaggle here <https://www.kaggle.com/liamisaacs/leagueoflegends-ranked-na-soloq-data-patch-114>. The data collection code is here https://github.com/yi-ye-zhi-qiu/metis-project3/blob/main/data_generation.ipynb.

From there, we can just "filter" the input data to the model for each game we want a SHAP plot for.

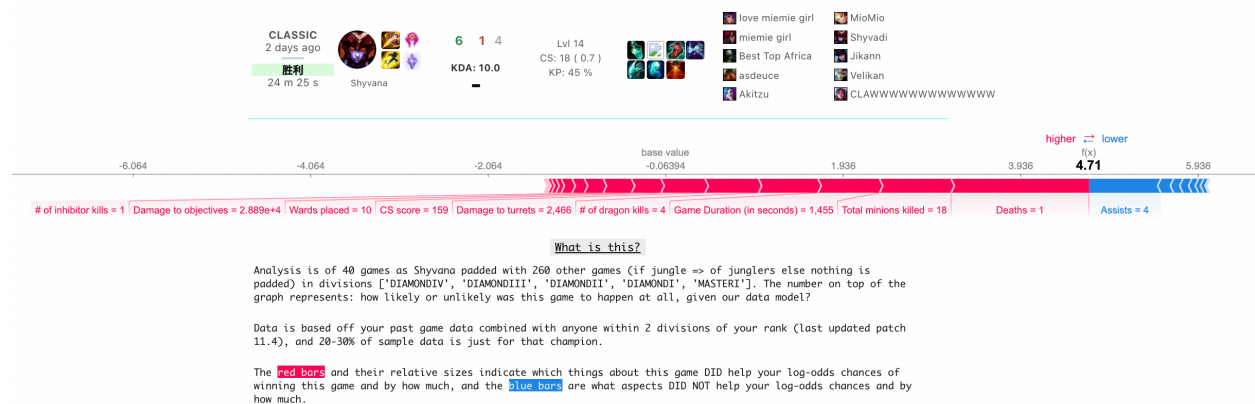


This is for that same Kaisa game: we filter as 221 games of Kaisa in the divisions ['SILVERIII', 'SILVERII', 'SILVERI', 'GOLDIV', 'GOLDIII'] assuming the player is ranked as Silver I. We over-sample recent matches from the player in that calculation from non-Kaisa games to capture the player's general play-style.



Since filtering for jungle is so easy (the API breaks for anything non-jungle, future releases of liam.gg will find a way...), we can for jungler's games "pad" them with enough to hit a sample size of 300. This is for one of my friends in Gold IV.

What's interesting is comparing this for games of a Diamond 2 player who plays a lot of the same champion, Shyvana.



What's funny is that the surface-layer statistics seem pretty similar – same runes, similar KDA, KP – but it's interesting to see how different the SHAP plots look.

IN SUMMARY

What this model does well

- (1) works well for mid to low elo junglers playing meta champions (where's there's a healthy amount of data);
- (2) the human interface is readable and clean;
- (3) can take into account and quantify how much a feature matters in a given win/loss;
- (4) can take into account a given player's play style and rank;
- (5) can provide pretty good feedback – ward less, take more dragons, that sort of thing.

Drawbacks of this model

- (1) Data volume is quite low – the more data, the better – and for smaller samples the model is just going to overfit;
- (2) Data collection pipeline requires a lot of maintenance (part of this is a drawback to API rate limits more than the model);
- (3) Data model takes a long time to run and has the capacity for only a few people to use at a time (due to rate limits but also general model design – the project might be better use to cache user data in a database or something)
- (4) ●●●●WE NEED A DARK MODE●●●●

—— THANK YOU

Thank you for reading. Please find the app here <http://liamisaacs.com/league>

Interested in working together?

Please contact me @ liamnisaacs@gmail.com OR contribute to the repository! <https://github.com/yi-ye-zhi-qiu/metis-project3>

Want to see more of my work?

Please see some more of my work @ liamisaacs.com