JOHN LIU

Team Fight Tactics

An Machine Learning Ranking Project

TM

Background

Developed by Riot Games
Compete with 8 other players
Build the strongest team

Our goal?

Create a model that can predict the Top 4 (Half) of the lobby.

We will be using ranking algorithms to achieve this

Why?

New mode called double up allows players to play in teams of 2.

Best gameplay experience comes from close games

Ranking

Given a QUERY
and a SET of DOCUMENTS
learn a FUNCTION
to SCORE the documents

Ranking

Why not classification?

Could we try and predict the probabilities
of a document being relevant to a query?

Crude but possible.
Misses relationship between
Query-Document
Document-Document

Ranking

Why not Regression?

Could we try and predict
the relevancy score of a document?

Again, crude but possible.
Misses relationship between
Query-Document
Document-Document

Techniques

Point-wise

Go document
by document

Closest to
classification/regression

(Q1, d1) - .9
(Q1, d2) - .2
(Q1, d3) - .5

Pair-wise

Go pair by pair

Compare each pair, which one
should be on top?

(Q, (d1,d2)) - 1
(Q, (d2, d3)) - 0
(Q, (d1, d3)) -1

List-wise

Considers all queries

Try to find an optimal order for
entire list by optimizing metric

Data Acquisition

Where do we get it?

Public Riot Developer API at
https://developer.riotgames.com/

How do we get it?

Using Riot Watcher
https://riot-watcher.readthedocs.io/en/latest/

Which API method to use?

We want a list of matches.
TFT-MATCH-V1

SUMMONER

GET:

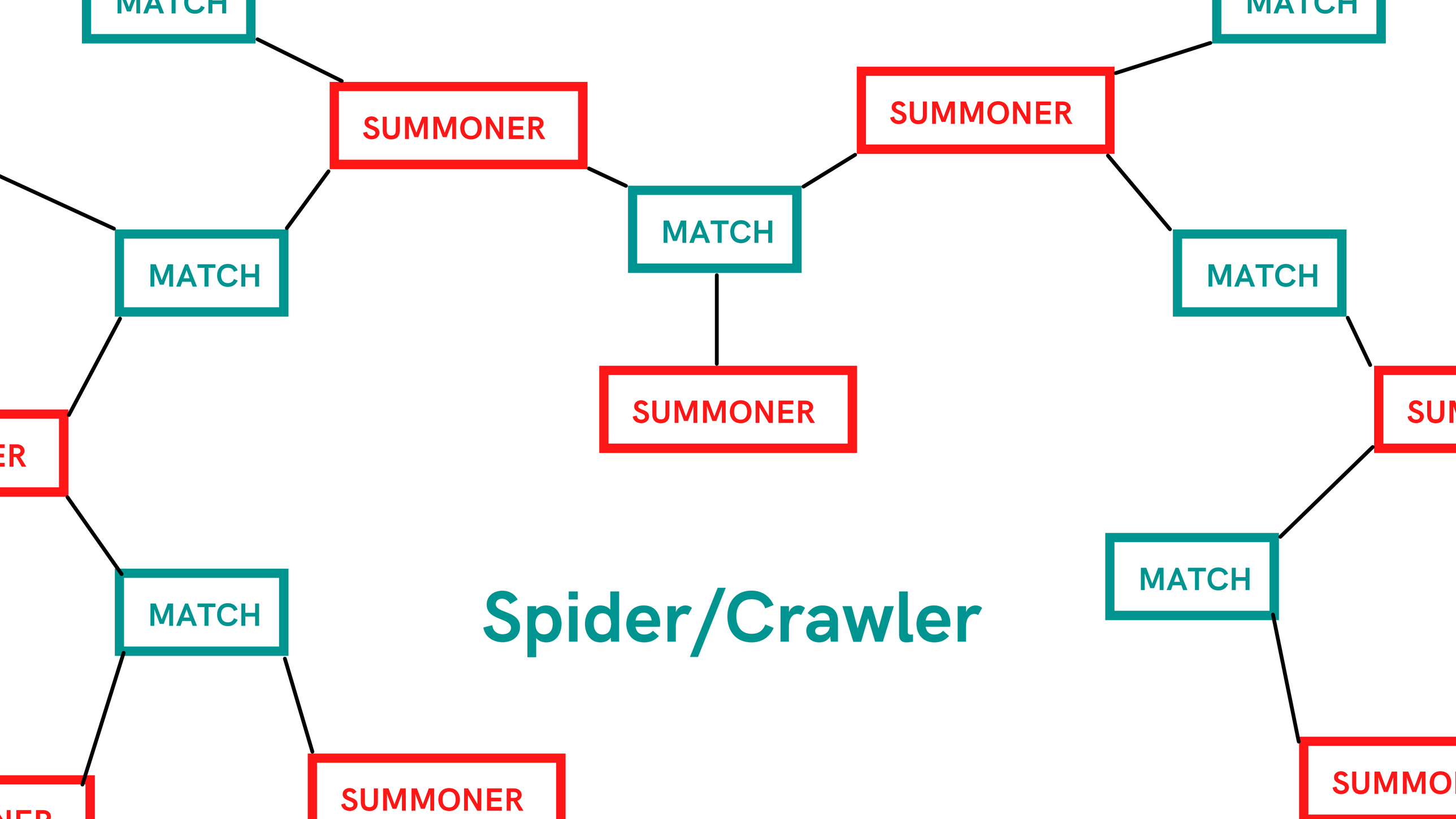
- SUMM NAME
- PUUID
- ENC SUM NAME
- ENC PUUID

MATCH

GET:

- MATCH ID
- PUUID

Spider/Crawler


MATCH
MATCH
SUMMONER
SUMMONER
MATCH
MATCH
MATCH
SUMMONER
MATCH
MATCH
SUMMONER
Spider/Crawler

```
rate = 0

for puuid in puuid_list:
    try:
        match_list = tft.match.by_puuid('americas', puuid, count=matches_per_player)
        rate = check_rate(rate,200)
        for match_id in match_list:
            match=tft.match.by_id('americas', match_id)
            rate =check_rate(rate,200)
            #Version Check
            if 'Version 12.3' in match['info']['game_version']:
                #Double Up Check
                try:
                    match['info']['participants'][0]['partner_group_id']
                    #Need Double Up
                except:
                    match_df = pd.concat({match_id:pd.DataFrame(match['info']['participants'])}, names=['MATCH_ID'])
                    res = pd.concat([res,match_df])
                #Version Mismatch
            else:
                continue
            if res.shape[0] % 1000 == 0:
                if verbose:
                    print("*" * 30)
                    print(f'Reached {res.shape[0]} Rows')
                    print('Saving...')
                    print(f'Current Time: [{time.strftime("%H:%M:", time.localtime())}]')
                res.to_csv('../datasets/match_df.csv', index_label=['MATCH_ID','index'])
```

21 Hours
253,000 Rows
31625 Matches

Cleaning and EDA

Cleaning

Tutorials
Double Up Games
Version

Unpacking

Traits and Units were stored as string interpretation of nested dictionary in list

Meta

What do people think is the strongest traits/units?

`ast.`**`literal_eval`**`(`*`node_or_string`*`)`

Safely evaluate an expression node or a string containing a Python literal or container display. The string or node provided may only consist of the following Python literal structures: strings, bytes, numbers, tuples, lists, dicts, sets, booleans, `None` and `Ellipsis`.

This can be used for safely evaluating strings containing Python values from untrusted sources without the need to parse the values oneself. It is not capable of evaluating arbitrarily complex expressions, for example involving operators or indexing.

Abstract Syntax Tree ast.literal_eval()

Warning: It is possible to crash the Python interpreter with a sufficiently large/complex string due to stack depth limitations in Python's AST compiler.

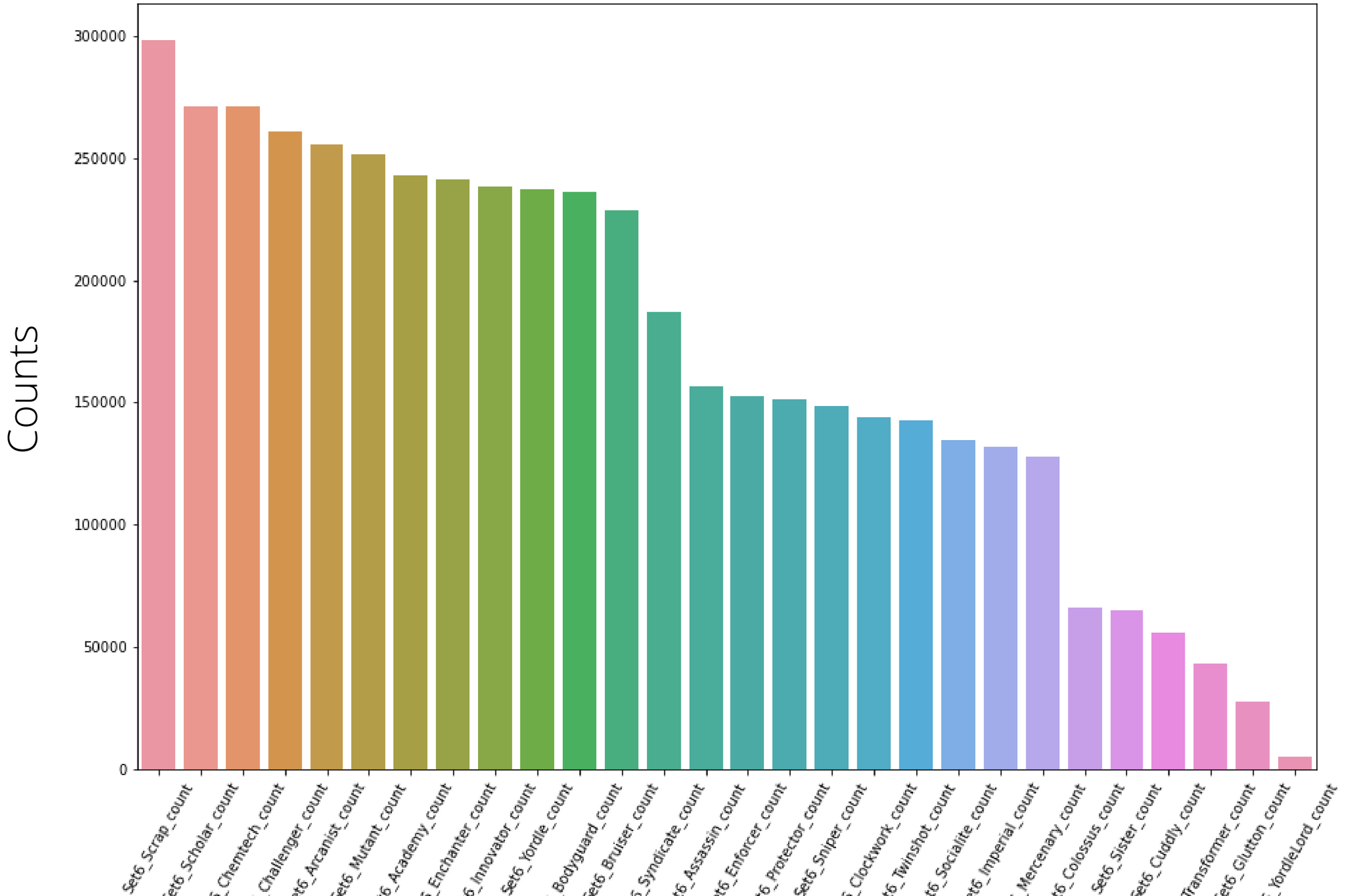
It can raise `ValueError`, `TypeError`, `SyntaxError`, `MemoryError` and `RecursionError` depending on the malformed input.

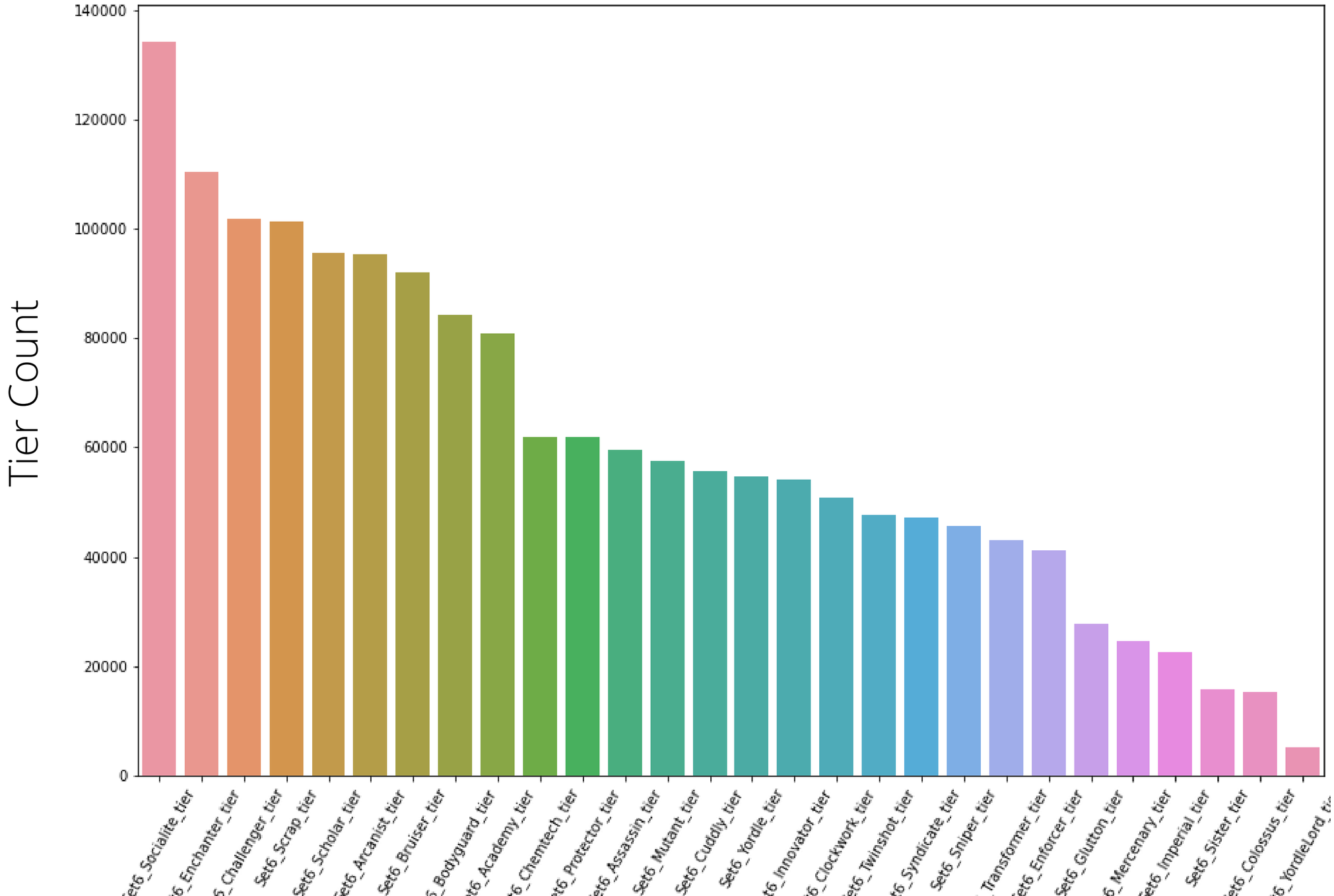
Changed in version 3.2: Now allows bytes and set literals.

Changed in version 3.9: Now supports creating empty sets with `'set()'`.

Changed in version 3.10: For string inputs, leading spaces and tabs are now stripped.

Trait Count


Counts
300000
250000
200000
150000
100000
50000
0
Set6_Scrap_count
Set6_Scholar_count
Set6_Chemtech_count
Set6_Challenger_count
Set6_Arcanist_count
Set6_Mutant_count
Set6_Academy_count
Set6_Enchanter_count
Set6_Innovator_count
Set6_Yordle_count
Set6_Bodyguard_count
Set6_Bruiser_count
Set6_Syndicate_count
Set6_Assassin_count
Set6_Enforcer_count
Set6_Protector_count
Set6_Sniper_count
Set6_Clockwork_count
Set6_Twinshot_count
Set6_Socialite_count
Set6_Imperial_count
Set6_Mercenary_count
Set6_Colossus_count
Set6_Sister_count
Set6_Cuddly_count
Set6_Transformer_count
Set6_Glutton_count
Set6_YordleLord_count


Trait Tier Count
Tier Count
140000
120000
100000
80000
60000
40000
20000
0
Set6_Socialite_tier
Set6_Enchanter_tier
Set6_Challenger_tier
Set6_Scrap_tier
Set6_Scholar_tier
Set6_Arcanist_tier
Set6_Bruiser_tier
Set6_Bodyguard_tier
Set6_Academy_tier
Set6_Chemtech_tier
Set6_Protector_tier
Set6_Assassin_tier
Set6_Mutant_tier
Set6_Cuddly_tier
Set6_Yordle_tier
Set6_Innovator_tier
Set6_Clockwork_tier
Set6_Twinshot_tier
Set6_Syndicate_tier
Set6_Sniper_tier
Set6_Transformer_tier
Set6_Enforcer_tier
Set6_Glutton_tier
Set6_Mercenary_tier
Set6_Imperial_tier
Set6_Sister_tier
Set6_Colossus_tier

Algorithms

RankNet, LambdaRank, and LambdaMART

As with everything

It all comes back to XGB

Evaluation Metric

Normalized Discounted Cumulative Gain

NDCG

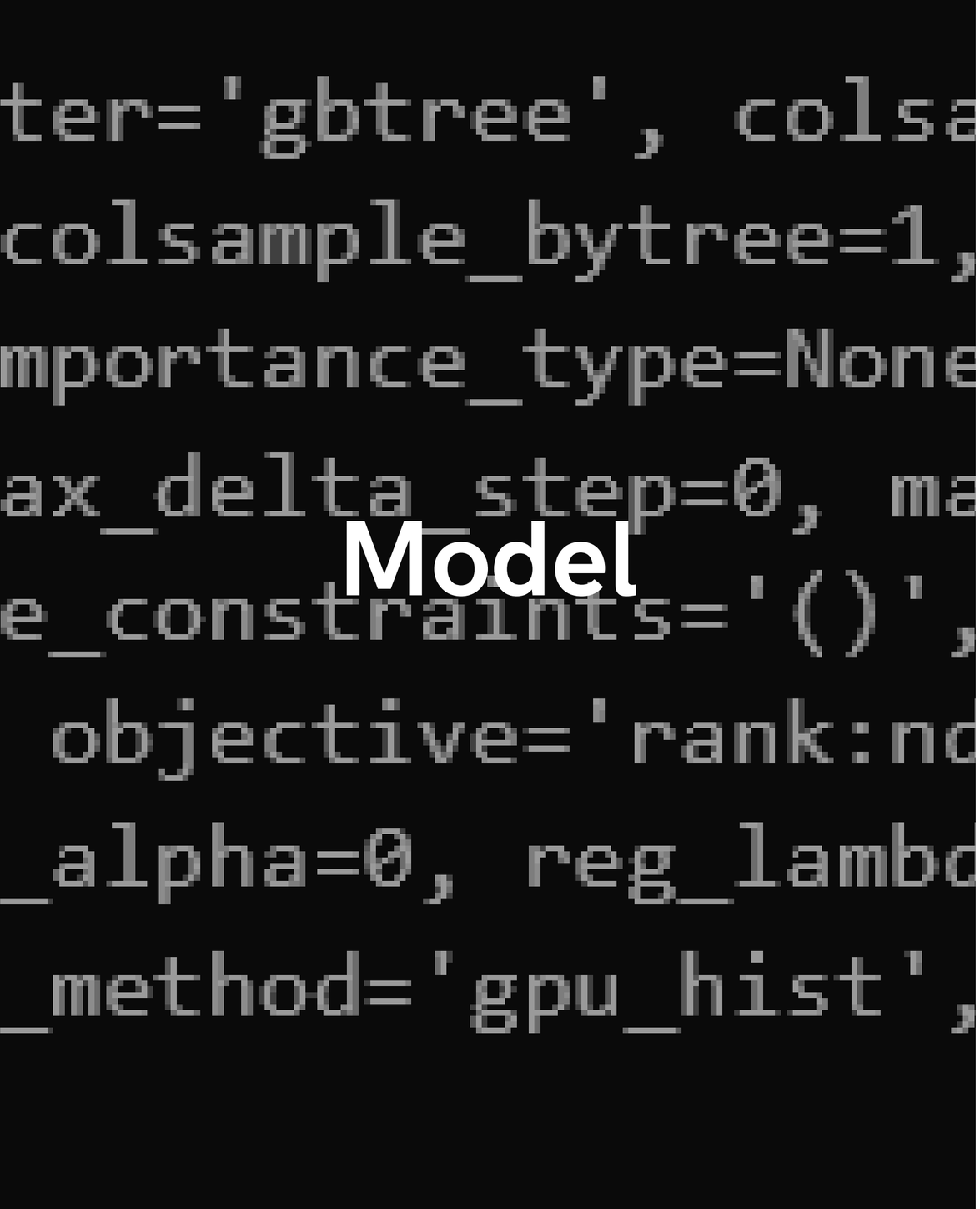
[3,3,2,2,1]

CG = 3+ 3 + 2 + 2 + 1 = 11

$$DCG = \frac{2}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{3}{log_2(3+1)} + \frac{1}{log_2(4+1)} + \frac{2}{log_2(5+1)} \approx 6.64$$

$$iDCG = \frac{3}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{2}{log_2(3+1)} + \frac{2}{log_2(4+1)} + \frac{1}{log_2(5+1)} \approx 7.14$$

NDCG = DCG/iDCG



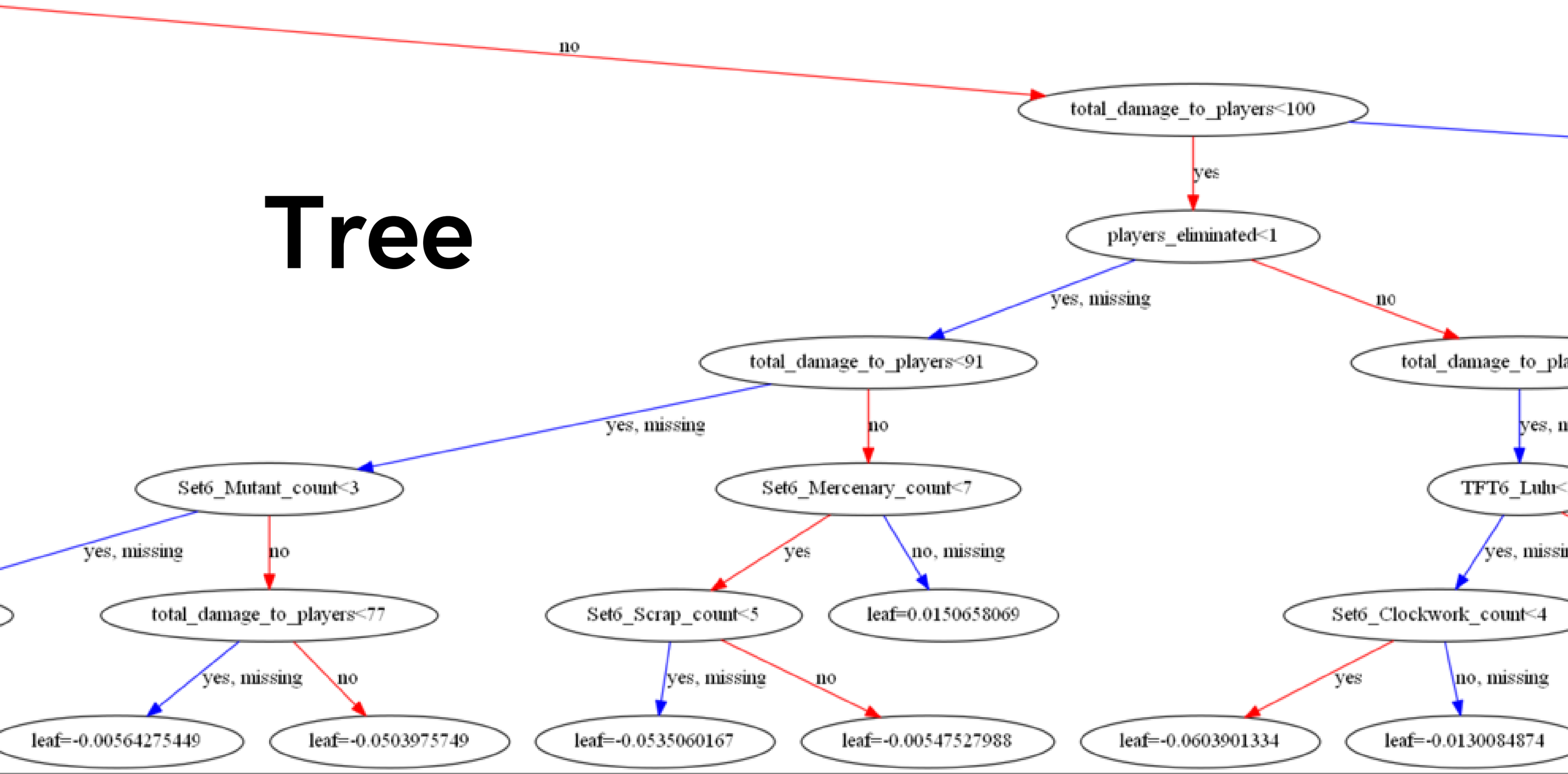
Model

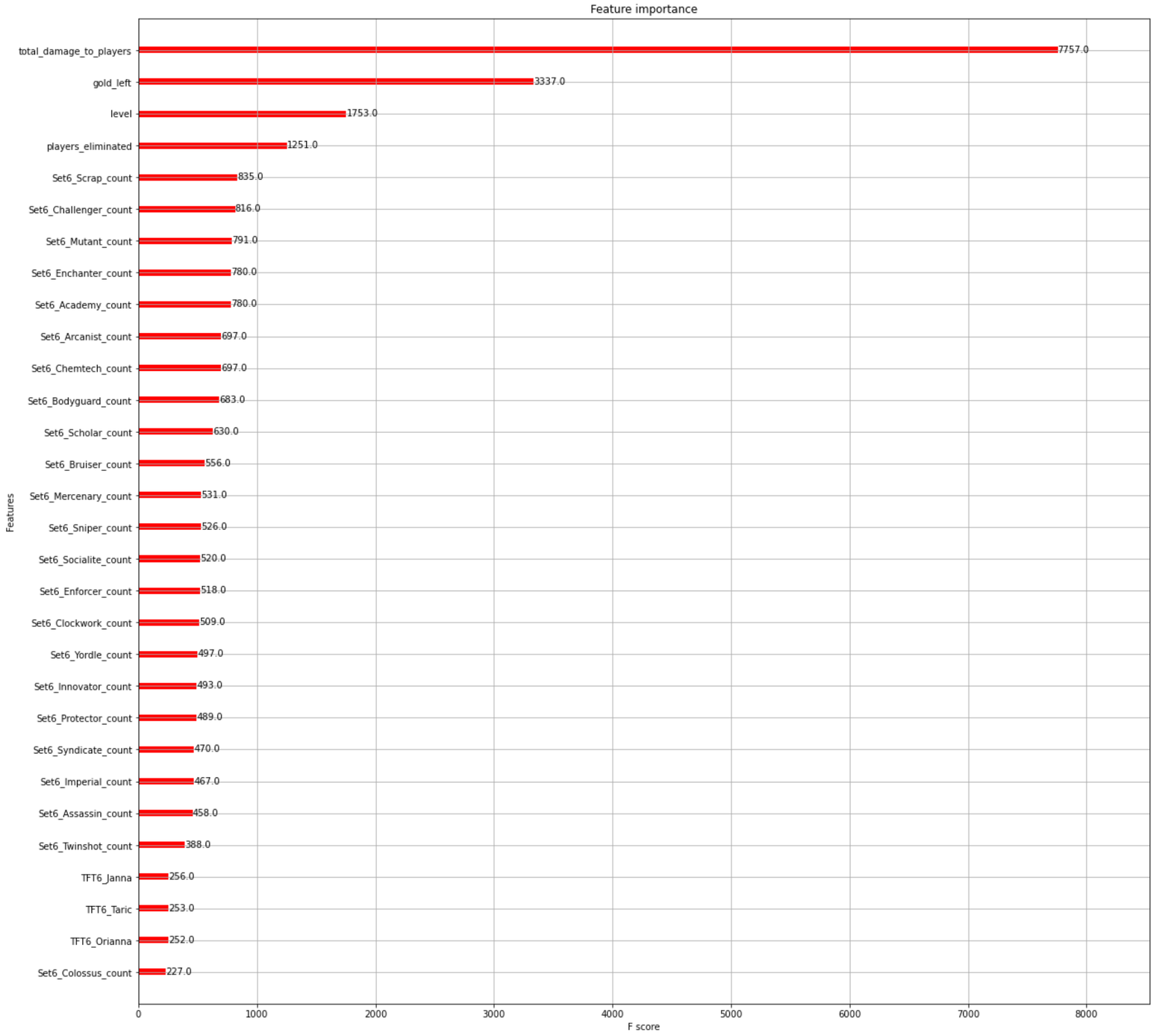
Avg NDCG

Train: 0.9889

Test: 0.9884

Tree


no
total_damage_to_players<100
yes
players_eliminated<1
yes, missing
no
total_damage_to_players<91
total_damage_to_pla
yes, missing
no
yes, n
Set6_Mutant_count<3
Set6_Mercenary_count<7
TFT6_Luh
yes, missing
no
yes
no, missing
yes, missi
total_damage_to_players<77
Set6_Scrap_count<5
leaf=0.0150658069
Set6_Clockwork_count<4
yes, missing
no
yes, missing
no
yes
no, missing
leaf=-0.00564275449
leaf=-0.0503975749
leaf=-0.0535060167
leaf=-0.00547527988
leaf=-0.0603901334
leaf=-0.0130084874



Feature Importance

Next Steps

Incorporate More Data/ Use More Data

Performance too low to be useful. Omitted Unit Items. Only screenshot endgame.

Data Bias

Due to MMR (Match Making Ranking). Seed more Players

Add Frontend

Insights can be used routinely. Accessibility is key.

Thank you!

Ritvik Math
Youtube: Ritvik Math

Sophie Watson
Github: sophwats

Jake and Hank
Github: jellena
Github: hank-butler