Udacity Data Scientist Nanodegree
Capstone Project Final Report

OpenDota API Data Analysis and Modeling

Xi Zhu
08/03/2021

Outlines

Definition

Overview

Being one of the most famous RTS-styled (RTS: real-time strategy) MOBA-type (MOBA: multi-player online battle arena) competitive team game, DotA has been well-loved by its players all over the world for almost 20 years. Every day, millions of players worldwide enter the battle as one of over a hundred Dota Heroes in a 5v5 team clash (Team *Radiant* and Team *Dire*)[1]. Each player controls a single Hero in a game, a strategically powerful unit with unique abilities and characteristics that can be strengthened as the game continues[2].

Having been a DotA fan myself, since I knew that there was a R package "RDota2" and a python package "dota2," and I could access Dota game data via API, I have always wanted to play with the API data. This project provides a great opportunity for me to make fun with the data and models.

Thus, in this project, I presented a trial to build a model to predict the game results (i.e., which team wins) given the heroes picked by both teams.

In fact, predicting the game result with only heroes would be difficult, as the heroes and characteristics in DotA2 are well-balanced (which is also why the game is popular). Besides

[1] DotA 2; https://www.dota2.com/home?l=english (last visited on Aug. 5, 2021)
[2] DotA 2; https://dota2.fandom.com/wiki/Dota_2 (last visited on Aug. 3, 2021)

the heroes, there would be many different factors that would impact how the game goes, and it is not likely that a specific hero combo would always win over others. However, it is possible to find some imbalance from the data exploration and modeling process, which would lead to meaningful and interesting findings.

The project is inspired by Chengjun Hou's blog "Ten lines of code to predict TI7," in which he performed data extraction and analysis using OpenData API data and R language[3]. The project also borrows some thought from Bill Prin's blog "Python and DotA2: Analyzing Team Liquid's Io success and failure."[4]

Problem Statement

This project aims to create a model that can predict game results from the heroes selected by both game teams. The tasks of this project include:

1. Extract match data from OpenDota API.
2. Train a model that could predict the game result from heroes selected.
3. Interpret the model result.

Metric

Area Under the Curve (AUC)

AUC is a metric commonly used to evaluate the performance of a binary predictive model. The range of AUC is between 0 and 1. Higher AUC means better performance for a predictive model. In other words, a model whose predictions are all wrong has an AUC of 0.0, while one whose predictions are all correct has an AUC of 1.0[5].

To demonstrate the meaning of AUC further, I would like to introduce the ROC curve (receiver operating characteristic curve) briefly. ROC curve is a graph showing the performance of a classification model by summarizing the trade-off between True Positive Rate vs. False Positive Rate at all classification thresholds[6]. Given the ROC curve, AUC further measures the entire two-dimensional area underneath the ROC curve from point (0,0) to point (1,1).

[3] Chengjun Hou; Ten lines of code to predict TI 7; Capital of Statistics, 2017.05.19; https://cosx.org/2017/05/rdota2-seattle-prediction/ (last visited on Aug. 5, 2021)

[4] Bill Prin; Python and DotA2: Analyzing Team Liquid's Io success and failure; Medium, Aug. 13, 2018; https://medium.com/@waprin/python-and-dota2-analyzing-team-liquids-io-success-and-failure-7d44cc5979b2 (last visited on Aug. 6, 2021)

[5] Classification: ROC Curve and AUC; Machine Learning Crash Course; https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (last visited on Aug. 06, 2021)

[6] Jason Brownlee; How to Use ROC Curves and Precision-Recall Curves for Classification in Python; Machine Learning Mastery, Jan. 13, 2021; https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ (last visited on Aug. 6, 2021)

In the project, the AUC score is calculated with Python *sklearn* package[7].

Data

Data Extraction

OpenDota is an open-source Dota data API platform where one could access Dota related data, including match, hero, team, etc.[8] As described in Bill's blog: "…OpenDota actually parses the replay files to pull out all sorts of interesting information, such as the results of team fights and gold advantages at various points in the game."[9] Thus, OpenDota can be an ideal source for me to get Dota match data. My reference for the OpenDota API platform is the OpenDota API doc.[10]

To extract data from the OpenDota API platform, an API key is required. I obtained an API key with my steam account.[11] Another helpful tool is a python package named "dota2."

Get Public Matches

I started with the /publicMatches endpoint to get public matches, which can return 100 randomly sampled public matches for each call.[12] With the following code in the terminal, I could extract and save the data in a *.json* file:

```
curl https://api.opendota.com/api/publicMatches > publicMatches.json
```

Another useful query parameter that can be applied to this request is "less_than_match_id," which could help get matches with a match ID lower than a given value. With this query parameter, I could obtain matches that have match IDs before a certain match. The query could be something like:

```
curl https://api.opendota.com/api/publicMatches?less_than_match_id={A_MATCH_ID} > publicMatches.json
```

[7] Metrics and scoring: quantifying the quality of predictions; https://scikit-learn.org/stable/modules/model_evaluation.html (last visited on Aug. 6, 2021)

[8] OpenDota; https://www.opendota.com/ (last visited on Aug. 7, 2021)

[9] Bill Prin; Python and DotA2: Analyzing Team Liquid's Io success and failure; Medium, Aug. 13, 2018; https://medium.com/@waprin/python-and-dota2-analyzing-team-liquids-io-success-and-failure-7d44cc5979b2 (last visited on Aug. 6, 2021)

[10] OpenDota API doc; https://docs.opendota.com/ (last visited on Aug. 7, 2021)

[11] API key; https://www.opendota.com/api-keys (last visited on Aug. 7, 2021)

[12] OpenDota: Public Matches; https://docs.opendota.com/#tag/public-matches (last visited on Aug. 7, 2021)

As match IDs are numbered according to the match time, I could submit more than 600 queries with different maximum match IDs and obtain more than 60,000 unique matches. This helped to obtain my dataset for this project.

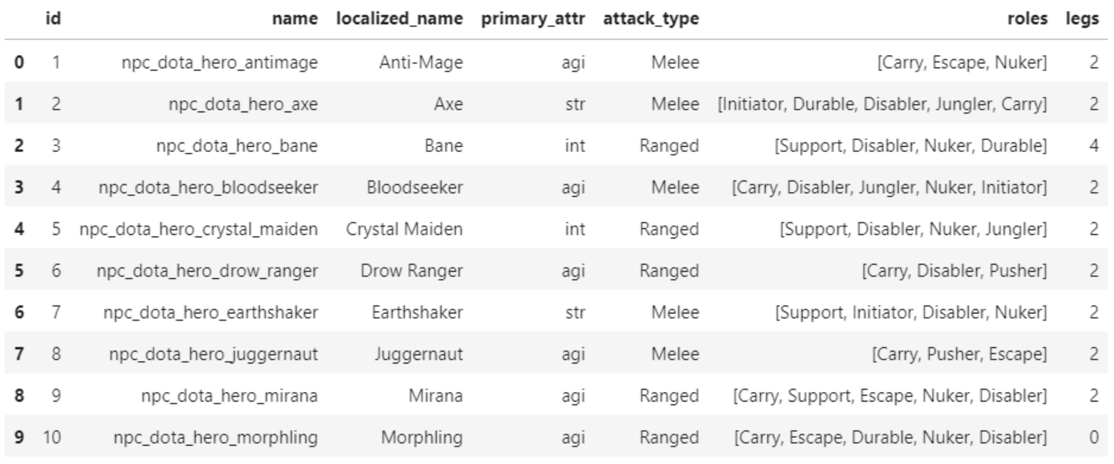
Get Hero Dataset

In the dataset, each hero is represented by a unique integer, named "hero_id." To know more information about each hero, I also obtained the dataset for heroes from OpenDota:

curl https://api.opendota.com/api/teams > teams.json

There are seven attributes in the data frame, where "id" tells the unique integer of each hero, and "localized_name" shows the names of the heroes. "Primary_attr" is the primary attribute for each hero, which is one of the aspects of Strength, Agility, and Intelligence. "Attack_type" tells whether the hero is a Melee or a Ranged. "Roles" shows the roles among nine possible roles that a hero could play in a game. "Legs" is the number of legs of this hero.

A brief view of the hero data frame can be found in Figure 1.



	id	name	localized_name	primary_attr	attack_type	roles	legs
0	1	npc_dota_hero_antimage	Anti-Mage	agi	Melee	[Carry, Escape, Nuker]	2
1	2	npc_dota_hero_axe	Axe	str	Melee	[Initiator, Durable, Disabler, Jungler, Carry]	2
2	3	npc_dota_hero_bane	Bane	int	Ranged	[Support, Disabler, Nuker, Durable]	4
3	4	npc_dota_hero_bloodseeker	Bloodseeker	agi	Melee	[Carry, Disabler, Jungler, Nuker, Initiator]	2
4	5	npc_dota_hero_crystal_maiden	Crystal Maiden	int	Ranged	[Support, Disabler, Nuker, Jungler]	2
5	6	npc_dota_hero_drow_ranger	Drow Ranger	agi	Ranged	[Carry, Disabler, Pusher]	2
6	7	npc_dota_hero_earthshaker	Earthshaker	str	Melee	[Support, Initiator, Disabler, Nuker]	2
7	8	npc_dota_hero_juggernaut	Juggernaut	agi	Melee	[Carry, Pusher, Escape]	2
8	9	npc_dota_hero_mirana	Mirana	agi	Ranged	[Carry, Support, Escape, Nuker, Disabler]	2
9	10	npc_dota_hero_morphling	Morphling	agi	Ranged	[Carry, Escape, Durable, Nuker, Disabler]	0

Figure 1. First ten rows of the hero data frame

Raw Dataset

In the original *.json* file, there are information for seven attributes for each match: “match_id,” “game_mode,” “duration,” “lobby_type,” “radiant_team,” “dire_team,” and “radiant_win.”

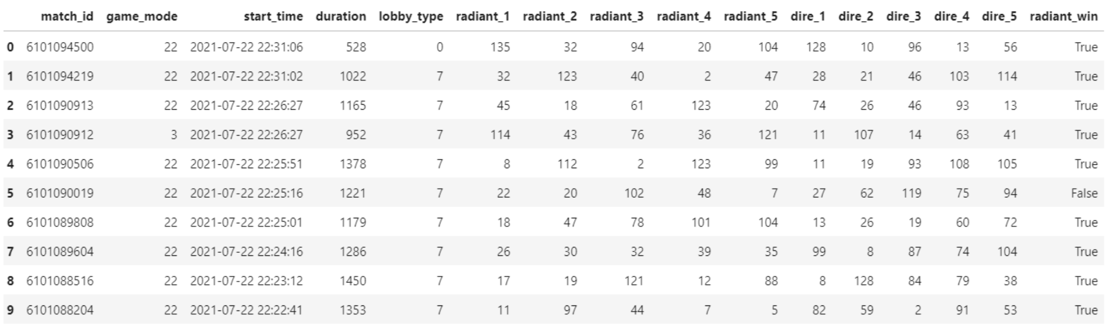
Attribute "match_id" is the match id of each match, which can be seen as the index of the data frame. Attribute "game_mode" refers to the mode of a match. "Duration" is the time length of a match. "Lobby_type" tells the type of lobby, such as public, practice, ranked, etc.[13]

[13] Dota 2 API: Match Details; http://sharonkuo.me/dota2/matchdetails.html (last visited on Aug. 7, 2021)

"Radiant_win" is a binary variable indicating whether team Radiant wins the game or not. This is also the target variable later in our predictive model.

The hero IDs of the heroes that are selected by each team are saved in the attribute "radiant_team" and "dire_team" in strings. To prepare for further analysis, I split the strings by comma and saved the five heroes for each team into five columns.

A brief view of the raw dataset can be found in Figure 2.



	match_id	game_mode	start_time	duration	lobby_type	radiant_1	radiant_2	radiant_3	radiant_4	radiant_5	dire_1	dire_2	dire_3	dire_4	dire_5	radiant_win
0	6101094500	22	2021-07-22 22:31:06	528	0	135	32	94	20	104	128	10	96	13	56	True
1	6101094219	22	2021-07-22 22:31:02	1022	7	32	123	40	2	47	28	21	46	103	114	True
2	6101090913	22	2021-07-22 22:26:27	1165	7	45	18	61	123	20	74	26	46	93	13	True
3	6101090912	3	2021-07-22 22:26:27	952	7	114	43	76	36	121	11	107	14	63	41	True
4	6101090506	22	2021-07-22 22:25:51	1378	7	8	112	2	123	99	11	19	93	108	105	True
5	6101090019	22	2021-07-22 22:25:16	1221	7	22	20	102	48	7	27	62	119	75	94	False
6	6101089808	22	2021-07-22 22:25:01	1179	7	18	47	78	101	104	13	26	19	60	72	True
7	6101089604	22	2021-07-22 22:24:16	1286	7	26	30	32	39	35	99	8	87	74	104	True
8	6101088516	22	2021-07-22 22:23:12	1450	7	17	19	121	12	88	8	128	84	79	38	True
9	6101088204	22	2021-07-22 22:22:41	1353	7	11	97	44	7	5	82	59	2	91	53	True

Figure 2. First ten rows of the raw dataset

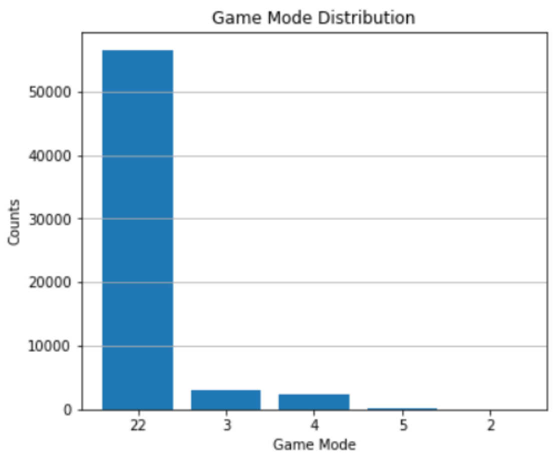
Analysis

Data Exploration

With the data extraction method presented in the previous section, I obtained data for 62,000 matches.

Game Mode and Lobby Type

I first checked game_mode and lobby_type. This is to check whether there are some data points that I need to remove. Since I'm interested in predicting the game results with the ten heroes selected by each team, some of the game modes with fewer than ten players in each match might need to be removed. Some of the lobby types, e.g., the type that refers to exercises, may also not be what I want for model training.



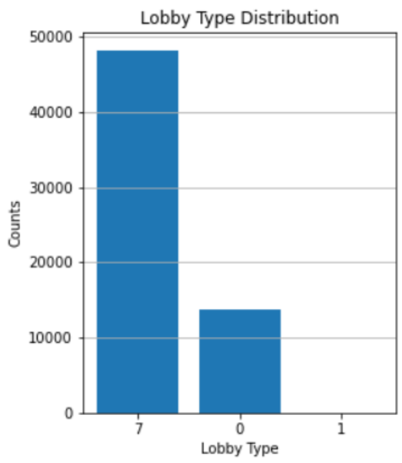


Figure 3. Game Mode and Lobby Type distributions

Figure 3 shows the Game Mode distribution and the Lobby Type distribution. It can be seen that most of the matches (91.2%) have game mode "22: Ranked all pick." Some games are in mode "3: Random draft" or "4: Single draft." Only a few have "5: All random" or "2: Captains mode."[14] All of these types are standard 10-player game modes, and I would keep them in the dataset.

The Lobby Distribution figure in Figure 3 shows that most matches are "0: Public matchmaking" and "7: Ranked." Only 37 among all 62000 games are "1: Practice." For data quality concerns, I would remove these 37 matches from the dataset.

Duration

Attribute "duration" describe how long (in minutes) the matches last. Figure 4 shows the distribution of match durations in the dataset.

[14] The OpenDota Project, dotaconstants; GitHub repository; May 8, 2018; https://github.com/odota/dotaconstants/blob/master/json/game_mode.json (last visited on Aug. 8, 2021)

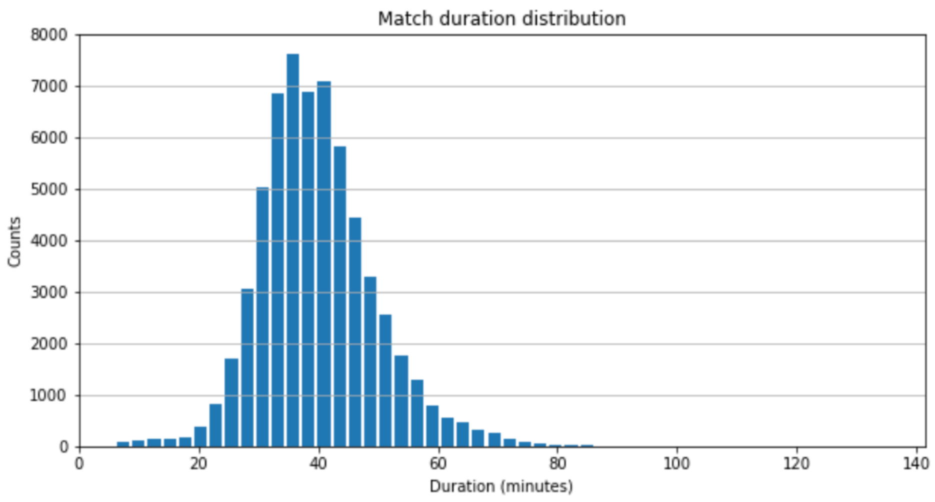


Figure 4. Duration distribution

In general, it is possible that some matches may take less than 20 minutes or take more than 2 hours for DotA games. However, in this project, I removed games that take less than 15 minutes, assuming that these games might have issues or low quality. This is consistent with Chengjun's analysis.[15] After deleting these matches from the dataset, some statistics are summarized in Table 1. It can be seen that most of the matches come to a result between 33 minutes (25%) to 45 minutes (75%).

Table 1. Statistics of match duration

	Count	**Mean**	**Std.**		
	61,603	40.07	9.43		
	Min	**25%**	**50%**	**75%**	**Max**
Duration (minutes)	15.00	33.53	39.00	45.25	135.23

Heroes

After checking the dataset, it could see that all 121 heroes appear in the dataset. Figure 6 shows the frequency of all heroes in the matches in the dataset. The top 5 MOST frequently picked heroes are *Pudge* (No. 14), *Lion* (No. 26), *Invoker* (No. 74), *Juggernaut* (No. 8), *Ogre Magi* (No. 84). The top 5 LEAST frequently picked heroes are *Chen* (No. 66), *Visage* (No. 92), *Lycan* (No. 77), *Beastmaster* (No. 38), and *Lone Druid* (No. 80).[16]

[15] Chengjun Hou; Ten lines of code to predict TI 7; Capital of Statistics, 2017.05.19; https://cosx.org/2017/05/rdota2-seattle-prediction/ (last visited on Aug. 5, 2021)

[16] One funny thing is that among the 5 MOST frequently picked heroes, *Pudge* and *Ogre Magi* are two heroes that are not popular in professional matches. This can be observed in the heroStats.json file data, which includes hero

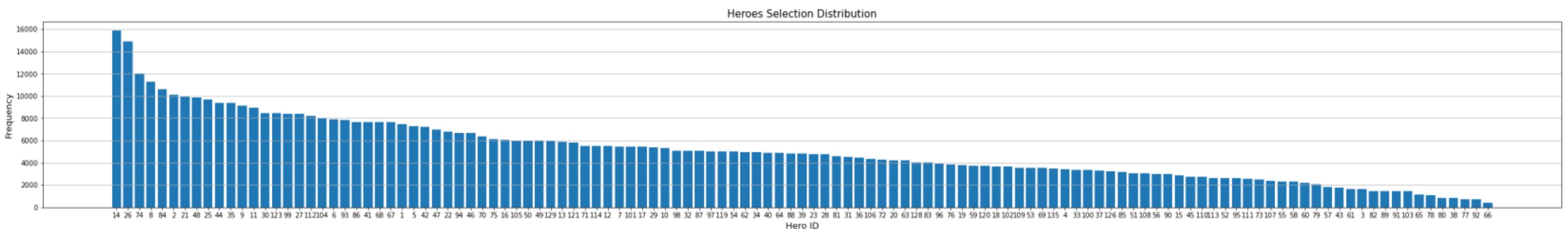


Figure 5. Hero selection distribution

Match Result

The match results are shown in the attribute "radiant_win" in the dataset. From Figure 5, it can be seen that the dataset is balanced, as the number of matches that the Radiant Team wins and the number of matches that the Radiant Team loses are close. Given this, I won't worry too much about the imbalance issue in the model training process.

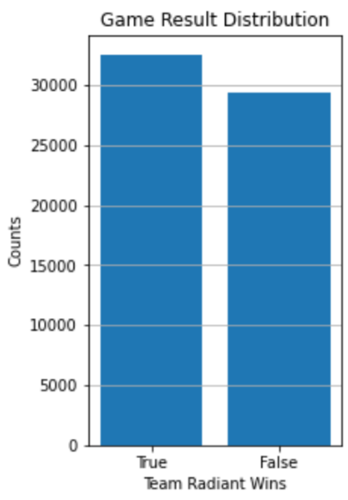


Figure 5. Game result distribution

Start Time

The games in the dataset were all in one day from 2021-07-22 05:53:46 to 2021-07-22 22:31:06. This guarantees that all the matches use the same game version (update patch version).[17]

statistics in recent professional matches, and can also be obtained from OpenDota (use *"curl https://api.opendota.com/api/heroStats > heroStats.json"*).

[17] Patches; Dota 2 Wiki; https://dota2.fandom.com/wiki/Patches (last visited on Aug. 8, 2021)

Modeling

Feature Selection

The features I used in the model included: heroes, the primary attributes of the heroes, roles or the heroes, match duration. How the formats and types of the first three features were modified can be found in the following sections.

Heroes

According to the rule, there are 121 different heroes available for each team (some heroes would be banned in professional matches, but I assume that these won't impact our model here), and each hero can only be selected once in a match. In other words, assume that one player in the radiant team selects Hero Axe (No. 2 in the hero data frame) in a match. Whether in team radiant or team dire, other players can not choose this hero "Axe" again in this match.

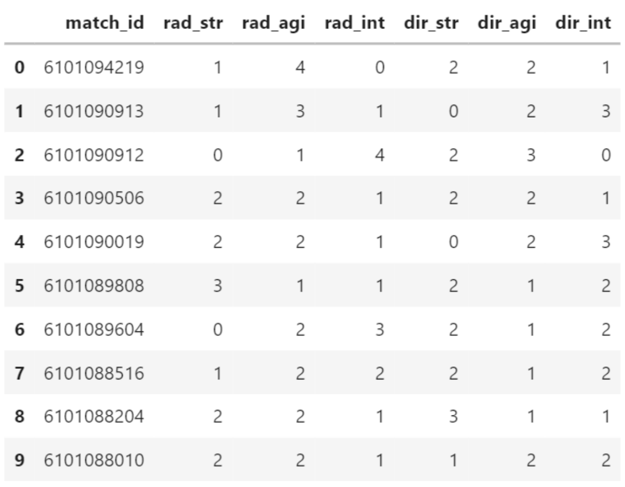
I add 121 features representing the 121 heroes for each team (in total, there would be 242 features). The features are named in a format of "rad_xxx" or "dir_xxx," where the "xxx" represents the hero ids. For each row, i.e., each match, the heroes picked by the two teams would have value "1" while others would be "0". In other words, for each match, there would be ten "1"s in these features.

Let me use a match (match_id = "6101094219") as an example. The team Radiant picked heroes 32, 123, 40, 2, and 47. In this row, columns "rad_32," "rad_123," "rad_40," "rad_2," and "rad_47" will be assigned value "1," while all other columns "rad_xxx" will be 0. Meanwhile, team Dire picked heroes 28, 21, 46, 103, and 144. Thus, in the same row, columns "dir_28," "dir_21," "dir_46," "dir_103," and "dir_144," will be assigned value "1." Other "rad_xxx" or "dir_xxx" columns would be “0.”

Primary Attribute

For each match, I consider the number of heroes with different primary attributes in each team.

For example, in the match "6101094219," the radiant team picked one Strength hero, four Agility heroes, and zero Intelligence heroes. The dire team picked two Strength heroes and two Agility heroes, and one Intelligence hero. The numbers assigned to columns 'rad_str,' 'rad_agi,' 'rad_int,' 'dir_str,' 'dir_agi,' and 'dir_int' would be 1, 4, 0, 2, 2, 1, as shown in Figure 6.



	match_id	rad_str	rad_agi	rad_int	dir_str	dir_agi	dir_int
0	6101094219	1	4	0	2	2	1
1	6101090913	1	3	1	0	2	3
2	6101090912	0	1	4	2	3	0
3	6101090506	2	2	1	2	2	1
4	6101090019	2	2	1	0	2	3
5	6101089808	3	1	1	2	1	2
6	6101089604	0	2	3	2	1	2
7	6101088516	1	2	2	2	1	2
8	6101088204	2	2	1	3	1	1
9	6101088010	2	2	1	1	2	2

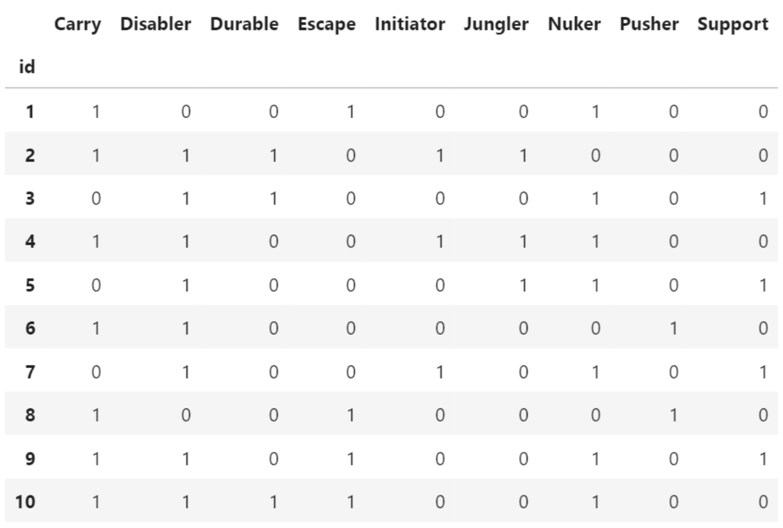
Figure 6. Primary attribute features for the first ten matches

Hero Roles

I encoded Hero Roles in a similar way to the way I dealt with Primary Attribute.

As shown in Figure 1, each hero could have more than one role. Since the match dataset does not tell which role a hero actually takes in a match, in this project, I only considered how many selected heroes could take each role in a team.

I first create a dummy-value hero-roles data frame. Each row is a hero, and each column is a role. There are nine possible roles for all heroes: Carry, Disabler, Durable, Escape, Initiator, Jungler, Nuker, Pusher, Support. The 0-1 values in the data frame indicate whether a hero (row) could play a role (column). See Figure 7 for a brief view of this data frame. For example, for hero number 1 (the first row of the hero-role data frame), the data frame tells that the hero has Carry, Escape, and Nuker roles.



	Carry	Disabler	Durable	Escape	Initiator	Jungler	Nuker	Pusher	Support
id									
1	1	0	0	1	0	0	1	0	0
2	1	1	1	0	1	1	0	0	0
3	0	1	1	0	0	0	1	0	1
4	1	1	0	0	1	1	1	0	0
5	0	1	0	0	0	1	1	0	1
6	1	1	0	0	0	0	0	1	0
7	0	1	0	0	1	0	1	0	1
8	1	0	0	1	0	0	0	1	0
9	1	1	0	1	0	0	1	0	1
10	1	1	1	1	0	0	1	0	0

Figure 7. The first ten rows of the hero-role data frame

With this hero-role data frame, I could calculate how many selected heroes could take each role for a team. For example, in match "6101094219," the radiant team selected heroes 32, 123, 40, 2, and 47. Referring to the hero dataset, for each hero, I could get a list of 0-1 values indicating which roles among all 9 roles this hero could play:

Hero 32: [1,1,0,1,0,0,0,0,0]

Hero 123: [0,1,0,1,0,0,1,0,0]

Hero 40: [0,1,0,0,1,0,1,1,1]

Hero 2: [1,1,1,0,1,1,0,0,0]

Hero 47: [1,1,1,0,1,0,0,0,0]

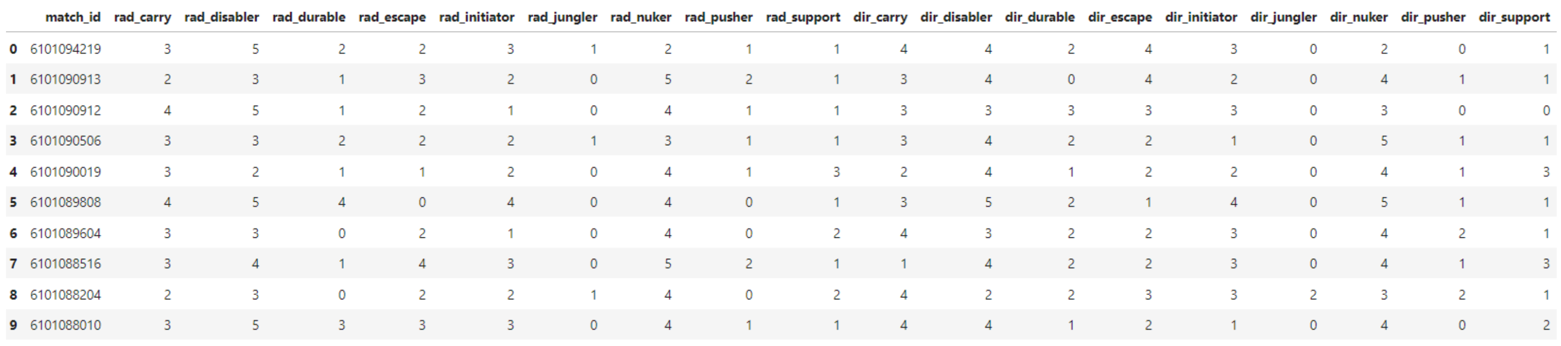
With these five 9-digit lists, I further generated another 9-digit list for the radiant team, by summing up the five lists by column, i.e., summing up the five numbers from the five lists for each role. In this example, we would obtain:

radiant: [3,5,2,2,3,1,2,1,1]

Similarly, for the dire team, I would obtain:

dire: [4,4,2,4,3,0,2,0,1]

As a whole, for match number 6101094219, I would have an 18-digit list for the role attributes. I added these columns to the dataset. Figure 8 shows a brief view of these columns in the dataset.



	match_id	rad_carry	rad_disabler	rad_durable	rad_escape	rad_initiator	rad_jungler	rad_nuker	rad_pusher	rad_support	dir_carry	dir_disabler	dir_durable	dir_escape	dir_initiator	dir_jungler	dir_nuker	dir_pusher	dir_support
0	6101094219	3	5	2	2	3	1	2	1	1	4	4	2	4	3	0	2	0	1
1	6101090913	2	3	1	3	2	0	5	2	1	3	4	0	4	2	0	4	1	1
2	6101090912	4	5	1	2	1	0	4	1	1	3	3	3	3	3	0	3	0	0
3	6101090506	3	3	2	2	2	1	3	1	1	3	4	2	2	1	0	5	1	1
4	6101090019	3	2	1	1	2	0	4	1	3	2	4	1	2	2	0	4	1	3
5	6101089808	4	5	4	0	4	0	4	0	1	3	5	2	1	4	0	5	1	1
6	6101089604	3	3	0	2	1	0	4	0	2	4	3	2	2	3	0	4	2	1
7	6101088516	3	4	1	4	3	0	5	2	1	1	4	2	2	3	0	4	1	3
8	6101088204	2	3	0	2	2	1	4	0	2	4	2	2	3	3	2	3	2	1
9	6101088010	3	5	3	3	3	0	4	1	1	4	4	1	2	1	0	4	0	2

Figure 8. The first ten rows of the role-related columns for both teams in the dataset

Duration

To include duration in the model is a bit tricky. As Chenjun stated in his blog, though this information can only be obtained when the match ends, and adding this variable can be suspected of "predicting the outcome with the result," there would be several reasons to add this feature to the model.[18] For the first, the role of heroes in Dota 2 can change dramatically

[18] Chengjun Hou; Ten lines of code to predict TI 7; Capital of Statistics, 2017.05.19; https://cosx.org/2017/05/rdota2-seattle-prediction/ (last visited on Aug. 5, 2021)

over the game, and different heroes have different strong and weak periods. Thus, it is important to avoid fighting when the opposing team is strong or end the game before it becomes strong. For the second, the duration could be an imaginary feature in prediction, thus makes it possible to obtain the trend of the winning percentage of a hero over time. For the last, the duration itself can not determine the result. Thus, I still decided to include this feature in my model.

The duration is in seconds (not in minutes, as discussed in the previous sections).

LGBM Pipeline

In this project, I use Light Gradient Boosting Model (LGBM) to train and predict the game results.

Target Variable Y

"Radiant_win"

Features X

Heroes

242 hero columns in format of "rad_xx" or "dir_xx"

Hero Primary Attributes

6 variables "rad_str," "rad_agi," "rad_int," "dir_str," "dir_agi," "dir_int"

Hero Roles

18 role variables "match_id", "rad_carry", "rad_disabler", "rad_durable", "rad_escape", "rad_initiator", "rad_jungler", "rad_nuker", "rad_pusher", "rad_support", "dir_carry", "dir_disabler", "dir_durable", "dir_escape", "dir_initiator", "dir_jungler", "dir_nuker", "dir_pusher", "dir_support"

Duration

1 variable "duration"

LGBM Pipeline

The dataset is firstly split into a training set (80% of the data) and a testing set (20% of the data). I use the training set to train the model (with cross-validation), and the test set to test the performance of the trained model.

LGBM has a lot of parameters for tuning. To train the model, I set an LGBM pipeline to train the model and test different parameters for the model.

```python
def model_LGBM():
    pipeline_LGBM = Pipeline([
        ('lgbm', lgb.LGBMClassifier(random_state = 0, n_estimators = 100, max_bin = 200))
    ])

    parameters = {
        'lgbm__max_depth': [2, 3, 5],
        'lgbm__min_data_in_leaf': [30, 40, 50],
        'lgbm__learning_rate': [0.4, 0.5, 0.6],
        'lgbm__num_leaves': [50, 100],
        'lgbm__bagging_fraction': [0.7, 0.8, 0.9],
        'lgbm__feature_fraction': [0.4, 0.45, 0.5],
        'lgbm__bagging_freq': [5, 10, 20],
        'lgbm__lambda_l2': [5, 10],
        }

    cv = GridSearchCV(pipeline_LGBM, param_grid = parameters)
    return cv
```

With the grid search method in cross-validation (GridSearchCV), the best parameters are:

num_iterations:	200
max_depth:	3
n_estimators:	100
min_data_in_leaf:	40
learning_rate:	0.6
num_leaves:	100
bagging_fraction:	0.9
feature_fraction:	0.45
bagging_freq:	5
max_bin:	200
lambda_l2:	10.0

Results & Discussion

Model Performance

After training the model with the training dataset, I applied the model to the testing set to see the result. As stated in the first section, the metric I would use in this project is AUC.

Table 2. Training and testing results

	Training	Testing
AUC	0.70076	0.60341

The results show that AUC for the training set is higher than the AUC for the testing set. This indicates that though I used cross-validation in the model training process, there were still over-fitting issues.

We know that AUC for a perfect model would be 1. A random model would return an AUC of 0.5. The AUC for general models should be between 0.5 and 1. A model with an AUC above 0.8 is considered to have quite good predictive performance, and a value between 0.6 and 0.8 means that the data has a certain degree of predictability. A model with little predictive ability will have an AUC close to 0.5. The AUC values for both training and testing sets show that the data/model has a certain degree of predictability for my model.

Interpretation

As expected, the model performance is poor if only using heroes selected to predict the game results. This is reasonable since a game result should not be purely determined by the players' heroes or how long the game lasts. However, the model may tell me some other information.

I also took a look at the feature importance of the model. Figure 9 shows the distribution of the feature importance. It can be noticed that while most of the features have low importance scores, some features do have higher importance than others.

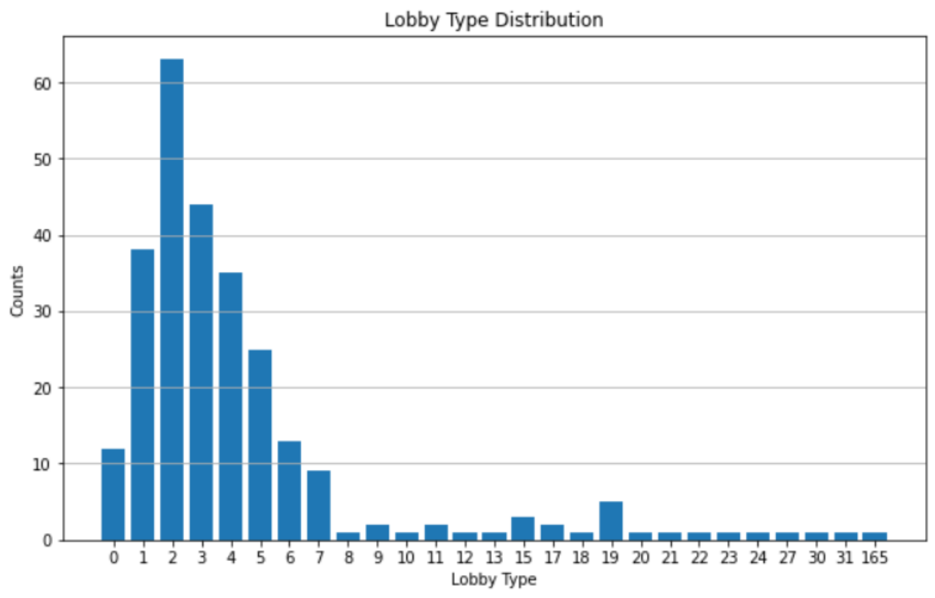


Figure 9. Feature Importance Distribution

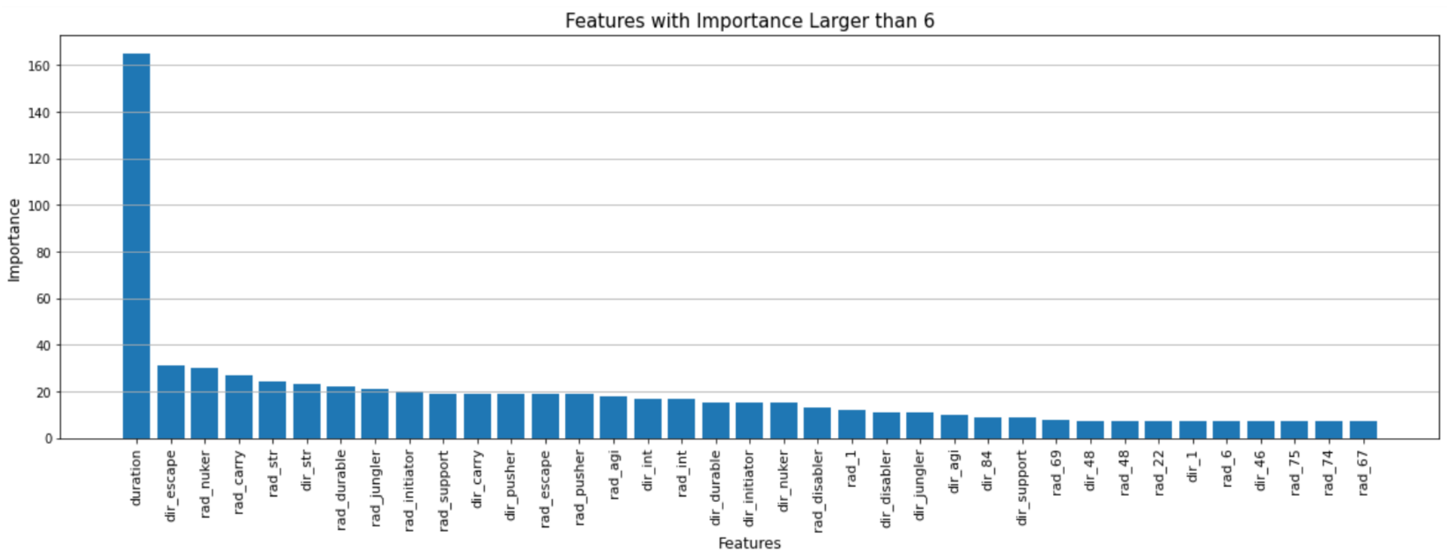


Figure 10. Features with Feature Importance Larger than 6

Let's see what features have importance no lower than 7 (>6). Table 3 shows features and importance scores for features that are not directly related to heroes (I call them "non-hero" features), i.e., primary attributes, roles, and duration. Table 4 summarizes the features that indicate heroes.

Among all features, "Duration" has the highest importance score (the importance is much higher than any other features, as shown in Figure 10). As a feature that could interact and work together with all other features, I would believe duration may contribute the most to the predictive model.

By comparing Table 3 and Table 4 (and also Figure 10), it can also be obviously noticed that almost all non-hero features have higher importance than hero features. While there seems no observable importance difference for hero roles, the "Strength" primary attribute seems to have higher importance than other primary attributes. However, this does not mean that the number of strength heroes itself would impact the game's result. The importance scores tell the predictive ability of variables in the model. It has no direct relationship with whether a single feature has a positive or negative effect on the game's result because the variables in the tree model all affect each other and play a role in the prediction.[19] But from the model, I might feel that a balanced combination of roles or primary attributes would impact the game result more than certain heroes.

Table 3. Feature importance for some non-hero features (importance score no smaller than 7)

Feature	Duration	Dir_escape	Rad_nuker	Rad_carry	Rad_str	Dir_str	Rad_duable

[19] Chengjun Hou; Ten lines of code to predict TI 7; Capital of Statistics, 2017.05.19; https://cosx.org/2017/05/rdota2-seattle-prediction/ (last visited on Aug. 5, 2021)

Importance	165	31	30	27	24	23	22
Feature	Rad_jungler	Rad_initiator	Rad_support	Dir_carry	Dir_pusher	Rad_escape	Rad_pusher
Importance	21	20	19	19	19	19	19
Feature	Rad_agi	Dir_int	Rad_int	Dir_durable	Dir_initiator	Dir_nuker	Rad_disabler
Importance	18	17	17	15	15	15	13
Feature	Dir_disabler	Dir_jungler	Dir_agi	Dir_support			
Importance	11	11	10	9			

Table 4. Feature importance for some hero features (importance score no smaller than 7)

Feature	Rad_1	Dir_84	Rad_69	Dir_48	Rad_48	Rad_22
Importance	12	9	8	7	7	7
Feature	Dir_1	Rad_6	Dir_46	Rad_75	Rad_74	Rad_67
Importance	7	7	7	7	7	7

Let's then take a look at the hero features. Some heroes show higher importance than other heroes. These heroes are:

Hero No. 1: Anti-Mage
Hero No. 48: Luna
Hero No. 84: Orge Magi
Hero No. 69: Doom
Hero No. 22: Zeus
Hero No. 6: Drow Ranger
Hero No. 46: Templar Assassin
Hero No. 75: Silencer
Hero No. 74: Invoker
Hero No. 67: Spectre

Among these heroes, Hero No. 1 *Anti-Mage* and Hero No. 48 *Luna* appear at the forefront of the importance ranking for both teams, indicating that the role of these heroes in improving the predictive ability of the model is stable, regardless of whether they appear in the Radiant side or the Dire side.

Comparing the Dota Hero Meta Statistics for this month[20], we can see that Luna, Ogre Magi, Drow Ranger, Silencer, and Spectre are all heroes that are very commonly picked with high

[20] Hero Meta Statistics; Dota Buff; https://www.dotabuff.com/heroes/meta?view=played&metric=rating_bracket (last visited on Aug. 11, 2021)

winning rates. Meanwhile, Invoker and Anti-mage are two heroes that are widely picked but with comparatively low winning rates. On the other hand, Doom, which has relatively high importance, has a low picking rate and a low winning rate. Even though we can not say that a single hero would definitely increase or decrease the winning probability, these statistics might reflect why they have a higher importance score in the model to some degree.

Discussion

Reflections

The process conducted in the project could be summarized in the following:

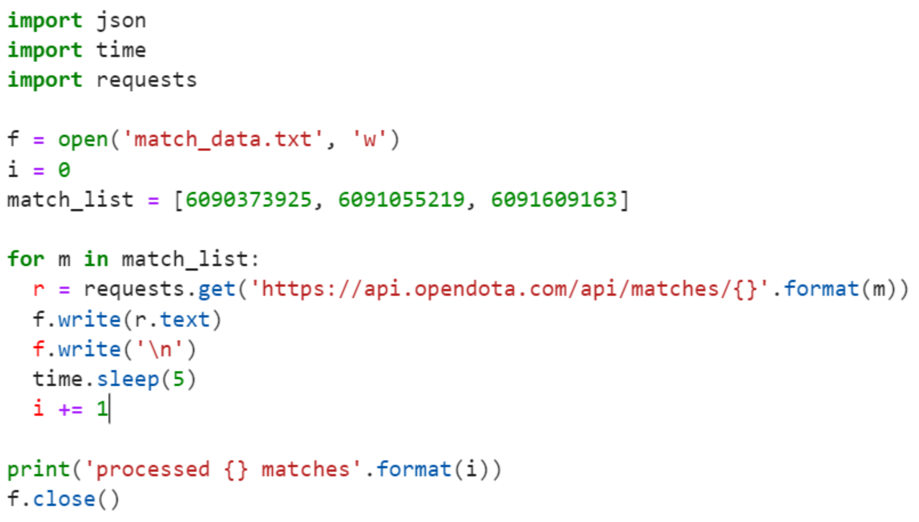
1. Raised a topic and a problem statement
2. Found ways to get relevant data and extracted the data from the API platform
3. Processed the raw dataset, explored the dataset, and created necessary features
4. A binary classifier was trained and tuned using the training set (a subset of the dataset), until a good set of parameters were found
5. Test the classifier using the testing set
6. Interpret the result

I chose this project out of my interest in OpenDota API data and my love for this game. As expected, heroes selected at the beginning of the game usually could not tell whether a team would win or not, especially in professional games when players are more skilled and experienced. Yet, though the model's performance is not good enough, the model itself and the parameters (i.e., importance scores, here in this project) could still tell some stories. In most cases, I would believe a good interpretation of a model would be as important as the model itself and the prediction performance. This project could be an example of this.

Improvement

Besides the heroes (and duration), more data can be accessed using OpenDota. Another more useful modeling approach, which I believe is also currently adopted in the game analysis tool for professional matches, is to predict the winning probability in real-time. This may require much more data than this simple project: the statistics of each hero, heroes' actions, items, etc. Since all this information would change as the game continues, the model needs to update in real-time.

To get more detailed match data, one might need to go to the /maches endpoint of the OpenDota platform. The following code can help (assume that I'm interested in accessing data for match 6090373925, match 6091055219, and match 6091609163):



```
import json
import time
import requests

f = open('match_data.txt', 'w')
i = 0
match_list = [6090373925, 6091055219, 6091609163]

for m in match_list:
  r = requests.get('https://api.opendota.com/api/matches/{}'.format(m))
  f.write(r.text)
  f.write('\n')
  time.sleep(5)
  i += 1

print('processed {} matches'.format(i))
f.close()
```