COGS108 Final Project Group 17

Predict Dota2 Winner/Loser

Permissions

Place an `X` in the appropriate bracket below to specify if you would like your group's project to be made available to the public. (Note that PIDs will be scraped from the public submission, but student names will be included.)

- [X] YES - make available
- [ ] NO - keep private

Overview

Dota2 is a very popular multi-player video game. The game is played by 5 players vs 5 players and one team would win the game. And of course, every player wants to win the game.

This project aims to define the most influential factors correlated with winning a Dota2 match. The final goal is then to predict the winner and the loser of a match during the early minutes of the game. To answer this question, we need to look at a huge amount of game matches data and analyze the factors to win a match.

In this project, we build a machine learning model that can predict the outcome of Dota2 matches with over 70% accuracy when game has started 5 mins, 10 mins, 15 mins or 20 mins.

Group Member IDs and Names

Futian Zhang

Billy Halim

Dennis J

Qinghong Chen

Research question

What factors influence the outcome of a Dota2 match? Specifically, can we predict the outcome of a Dota2 match when the match has started 5 mins, 10 mins, 15 mins or 20 mins?

Background and Prior Work

Before diving into datasets, we first need to understand our subject, Dota2. We will present 2 introductions to the game, a long one and a short one. The longer one contains more details about the game. The shorter one only contains a few information that matters to our project. We would suggest reading the longer one to get a better understanding of the game. However, if you are run out of time, you can just read the shorter one.

Long introduction start here

Dota2 is a popular multiplayer online battle arena video game developed and published by Valve. The full name is Defense of the Ancients 2. It was published in 2013 and so far, it has attracted millions of the players around the world. Each year, a championship is held for professional players/clubs to compete with each other. Every year, the championship has a prize pool of over $10,000,000 dollars.

Dota2 is played in matches between two teams of five players. Each match is a new start. The past matches have no effect on current matches. Meaning if you start a new match, the only thing that you can carry on for this match is your own skill and experience. For a normal player, when you play the game, you can play alone and the system will match you up with 4 other players to form a team to compete with 5 other players in your opponent side, or you can play with your friends, and the system will fill your team with other players to form a team of 5 and match you up with 5 other players in your opponent side as well. Anyway, each match will start with 5 players vs 5 players. Also, one team is called Radiant and one team is called Dire. The assignment is random.

The goal of the game is to destroy the ancient on the opponent side and protect your ancient. We will explain everything in detial.

Before a match start, the first thing that a player need to do is to select a hero. A hero is a special unit that you control. There are more than 100 heros in the game. Each hero has different abilities and attributes so that they function differently, we will explain this later. Everyone would select a hero so each team would have 5 heros. For professional matches, there is a banning phase, where you can ban the heros that you dislike to forbid the opponent side to select this hero. But in this project we would not consider this. We only need to know that a match start with 5 heros in each side.

Each hero is a unit controlled by players. Each hero has many attributes: health, mana, attack damage, attack speed, etc. We will explain all of them. Note that all the attributes can be increased/decreased by abilities and items.

Health: The health is the value that determine whether a hero is alive or dead. Each hero has a maximum health. Its health cannot exceed this value. Each hero starts with full health in the game, health can be lowered and raised by many methods. For example, being attacked by enemy heros would reduce your health value. Staying in base and rest would recover your health. Most of abilities of the heros are for damaging or recover the health of the heroes. The maximun health can be increased by abilities and items.

Mana: The mana is spent to use abilities. It works the same with health, but your hero would not die if it runs out of mana, it just couldn't use any abilities.

Move speed: The speed that your hero walks around the map.

Basic attack: Each hero can attack any opponent units within its attack range, which is different for all heros. Attack damage would determine the amount of damage that is dealt for this attack and attack speed would determine the length of time between each attack.

Abilities: Each hero normally has four different abilities, three normal abilities and a powerful ultimate ability. Each abilities is unique. You can learn/level up your abilities for each level you gain. Some abilities can deal damage to enemy units or heros, some abilities can increase/decrease the attributes of heros, some can heal friendly units etc.

Respawn: If health value drop to 0, a hero is dead (so is all other units in the game). The dead hero need to wait some amount of time to respawn with full health in the team base. The length of the time determine on many things, and it will get longer as the game time get longer. For example, if your hero died at the start of the game, you only need 10 seconds to respawn, but if your hero died 30 minutes after the game started, you would need about 1 mintes to respawn.

Level: All heroes start with level 1. Then the hero can gain experience by the death of enemy units around the hero. If the hero gain enough experience, it will level up. When a hero level up, its attributes improve and the hero can learn/improve one ability. The higher the level a hero is, the more powerful the hero is. The maximum level is 25.

Item: Each hero can purchase items from stores using gold, gold can be earned by kill enemy units. Items are for improving attributes. Some items have give additonal abilities for heros to use. The better a hero's items are, the more powerful the hero is. A hero can maximumly have 6 items.

Now, the match has begun. The map for Dota2 stays the same for all matches. The map is a square. Radiant team is in left bottom side of the map and Dire team is top right side of the map. There are many component of the map: base, lane, and jungle, etc. We will explain each one of them.

Base: The base is the place for heros to rest and purchase items. Each team has one base. For Radiant, the base is in left bottom corner of the map Each hero would start the game in the team base. For Radiant, the base is in top right corner.

Ancient: The goal of the game is to destroy the enemy ancient. It is located near the base of each team. A team wins by destroy the enemy ancient.

Tower: Each team has many towers located in the map to protect the ancient. Each tower cannot be moved or controlled by players, but it will automatically attack any enemy units with the attack range.

Lane: The map has three lanes, call top, mid and bottom. Each lane has towers to defence.

Minion: Each lane would produce minions. Minions are units that cannot be controlld and constantly move towards the enemy ancients unless enemy units is in range, then the minions would attack enemy units. Killing minions can gain gold and experience.

Jungle: Each rest of the map is the jungle. There are neutral creeps in the jungle that you can kill for gold and experience.

The above explained most of the game. For further details, please refer to https://dota2.gamepedia.com/Dota_2_Wiki (https://dota2.gamepedia.com/Dota_2_Wiki).

Then let us define some terms for the game for this paper to use.

KDA: Three values that determine the performance of a hero. Kill/Death/Assist.

Kill: Damaging enemy heros and drops their health below 0. When the enemy hero dies, the last hero that deals damage to it gains the kill, and its kill count plus one. The kill will reward with gold and experience to enpower heros. More kills means more powerful the hero is.

Death: When the hero's health drop to 0, it dies, and its death count plus one. The dead hero would lose some gold and need to wait for time to respawn. The more death a hero has, the less powerful a hero is.

Assist: When a hero helps a friendly hero to kill enemy hero, it gains assist. The assist also rewards some gold and experience, but not as much as the kill.

Farm: The action of killing enemy minions and creeps to gain gold and experience. Some heros need long farming phase, meaning that it need to kill a lot of minions and creeps to buy good items and have high level to become powerful.

Gank: The action of killing enemy heros to gain gold and experience. Some heros are good at ganking, meaning that they can easily kill enemy heros.

Carrier: The hero that deal high damage.

Support: The hero that supports friendly heros to farm or gank. For example, heros with abilities that can heal friend heros are usually support.

Teamfight: Fights between multiple heros from each side. Usually leads to kill and death of heros.

Long introduction ends here

Short introduction starts here

Dota2 is played with 5 players vs 5 players, one side is called radiant and one side is called dire. Each game is a new start, meaning that eariler games have no effects on later games. The game started with each player select a hero. There are more 112 heros in the game right now. Different heros have different abilities, and different power. In the game, players can kill minions and opponent heros to earn gold and experience. Gold can buy items and experience can level up. Items and level up will empower your hero. So more items and higher level means more powerful. The goal of the game is to destroy the opponent ancient, the structure located in the opponent base. One match usually takes 25-45 mins.

Term explanation:

Radiant: the team that starts from left botton side of map

Dire: the team that starts from right top side of the map

Hero: the character of a player, different heros have different abilities and power

Ability: abilities of a hero are an array of magic that a hero can cast, different heros have different abilities. Abilities can be learned and enpower when the hero levels up

Gold: money to buy items in the game. Note that gold can only be earned by in-game action. Such as killing minions, killing opponent heros, etc.

Experience: enough experience will level your hero up. Note that experience can only be earned by in-game action. Such as minions die around you, opponent heros die around you, etc.

Item: the items will empower a hero. More and better items mean stronger heros.

Level: the higher level a hero is, more powerful the hero is.

Minions: AI controlled soilders or creeps.

Ancient: The goal of the match is to destroy opponent ancient

Short introduction starts here

We are interested in this topic because we are Dota2 players. We all want to win the matches. Therefore, we are curious about what influence the winner/loser of a match.

We want to investigator win prediction of Dota2. According to the experience gained from the game that we have played, we think that the side with gold lead and experience lead (more gold and experience than the opponent side) would win the game. We also think the side with better hero selection would win the game, but this is vague and completely based on our experience, therefore, we want to build a model to predict winner. This is important because Most players want this because this can help them determine which sides is going to win the game. For pro players, this is more important because this analysis help them win matches and earn money.

Many data scientists have done research on dota2. A few will be listed below:

UCSD researchers studied the win prediction of dota2 based on hero selection.
http://jmcauley.ucsd.edu/cse255/projects/fa15/018.pdf (http://jmcauley.ucsd.edu/cse255/projects/fa15/018.pdf)

A group of Stanford professors built a hero recommandation system with machine learning.
http://cs229.stanford.edu/proj2013/PerryConley-HowDoesHeSawMeARecommendationEngineForPickingHeroesInDota2.pdf (http://cs229.stanford.edu/proj2013/PerryConley-HowDoesHeSawMeARecommendationEngineForPickingHeroesInDota2.pdf)

Prior research had been done to study the win prediction of the game based on the hero selection. We would like to study further about win prediction 5 minutes, 10 minutes, 15 minutes, and 20 minutes after the game has started.

Hypothesis

Our primary focus is to determine win prediction based on the game situations. The prediction may be based on hero selection, total gold, experience, skill of the players of the team. We predict that the team with better hero selections, more gold and experience, and better skilled players would win the game. (If you don't understand the terms in this section, please refer to introduction term explanation part.)

We aim to build a machine learning model to predict the winner of a match. However, this is a very challenging topic since Dota2 is a game with great variance. Due to this reason, we would only expect that our model can achieve 70% accuracy.

Dataset

Match data

Dataset name: Dota 2 Matches

Link: https://www.kaggle.com/devinanzelmo/dota-2-matches (https://www.kaggle.com/devinanzelmo/dota-2-matches)

Number of observations: 50000

We will use this dataset for our analysis. This dataset contains 50000 ranked matches collected with OpenDota API by Devin Anzelmo in 2019 December. They are all high-level ranked matches, meaning all players contributed to this dataset must be experienced players and have played the game for hundreds of hours. It contains details about the matches, such as outcome, hero selection, KDA, GPM (Gold per minute), XPM (experience per minute), etc. The information that we will use is the outcome of a match, hero selection, and the hero's gold/experience at certain game time.

Setup

Import all required package

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
sns.set_context('talk')
# Erase warnings
import warnings
warnings.filterwarnings("ignore")

import patsy
import statsmodels.api as sm
import scipy.stats as stats
from scipy.stats import ttest_ind, chisquare, normaltest

from sklearn.svm import SVC
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report, precision_recall_fscore_sup
port

import gc
```

Data Cleaning

Data files: Match.cvs, player.cvs, player_time.cvs. These data files are already "clean", because they were downloaded from game servers, so we don't need to do anything to modify it. We just need to remove some unrelated data. First, we load each dataset and see all the values

In [2]:
```
df_match = pd.read_csv("match.csv")
df_match.head()
```

Out[2]:

	match_id	start_time	duration	tower_status_radiant	tower_status_dire	barracks_status_dire
0	0	1446750112	2375	1982	4	3
1	1	1446753078	2582	0	1846	63
2	2	1446764586	2716	256	1972	63
3	3	1446765723	3085	4	1924	51
4	4	1446796385	1887	2047	0	0

There are many data in this csv. But we don't need all of them. So we remove all the unimportant items, leaves "match_id", "duration", "radiant_win". These values indicates the match_ID, length and winner of the game.

In [3]:
```
df_match = df_match.drop(columns=["start_time","tower_status_radiant","tower_s
tatus_dire","barracks_status_dire","barracks_status_radiant","first_blood_tim
e", "game_mode","negative_votes","positive_votes","cluster"])
df_match.head()
```

Out[3]:

	match_id	duration	radiant_win
0	0	2375	True
1	1	2582	False
2	2	2716	False
3	3	3085	False
4	4	1887	True

In [4]:
```
df_player = pd.read_csv("players.csv")
df_player.head()
```

Out[4]:

	match_id	account_id	hero_id	player_slot	gold	gold_spent	gold_per_min	xp_per_min	kill
0	0	0	86	0	3261	10960	347	362	9
1	0	1	51	1	2954	17760	494	659	1
2	0	0	83	2	110	12195	350	385	
3	0	2	11	3	1179	22505	599	605	
4	0	3	67	4	3307	23825	613	762	2

5 rows × 73 columns

Now, we look at players.csv. As we can see, there are tons of unimportant data, such as account id, gold/experience source (where the hero get gold and experience), and unit order (these columns record how you control your hero and other units, we are not going to use these data because these data are too detailed). So we remove them.

In [5]:
```
df_player = df_player.drop(columns=["account_id",'xp_hero', 'xp_creep', 'xp_ro
shan', 'xp_other',
       'gold_other', 'gold_death', 'gold_buyback', 'gold_abandon', 'gold_sell'
,
       'gold_destroying_structure', 'gold_killing_heros',
       'gold_killing_creeps', 'gold_killing_roshan', 'gold_killing_couriers',
'unit_order_none',
       'unit_order_move_to_position', 'unit_order_move_to_target',
       'unit_order_attack_move', 'unit_order_attack_target',
       'unit_order_cast_position', 'unit_order_cast_target',
       'unit_order_cast_target_tree', 'unit_order_cast_no_target',
       'unit_order_cast_toggle', 'unit_order_hold_position',
       'unit_order_train_ability', 'unit_order_drop_item',
       'unit_order_give_item', 'unit_order_pickup_item',
       'unit_order_pickup_rune', 'unit_order_purchase_item',
       'unit_order_sell_item', 'unit_order_disassemble_item',
       'unit_order_move_item', 'unit_order_cast_toggle_auto',
       'unit_order_stop', 'unit_order_taunt', 'unit_order_buyback',
       'unit_order_glyph', 'unit_order_eject_item_from_stash',
       'unit_order_cast_rune', 'unit_order_ping_ability',
       'unit_order_move_to_direction', 'unit_order_patrol',
       'unit_order_vector_target_position', 'unit_order_radar',
       'unit_order_set_item_combine_lock', 'unit_order_continue'])
df_player.columns
```

Out[5]:
```
Index(['match_id', 'hero_id', 'player_slot', 'gold', 'gold_spent',
       'gold_per_min', 'xp_per_min', 'kills', 'deaths', 'assists', 'denies',
       'last_hits', 'stuns', 'hero_damage', 'hero_healing', 'tower_damage',
       'item_0', 'item_1', 'item_2', 'item_3', 'item_4', 'item_5', 'level',
       'leaver_status'],
      dtype='object')
```

Note that there are still tons of data in here. We are probably not going to use all of them, but they means somethings to the game, so we keep it.

In [6]:
```
df_player_time = pd.read_csv("player_time.csv")
df_player_time.head()
```

Out[6]:

	match_id	times	gold_t_0	lh_t_0	xp_t_0	gold_t_1	lh_t_1	xp_t_1	gold_t_2	lh_t_2	...	xp_
0	0	0	0	0	0	0	0	0	0	0	...	
1	0	60	409	0	63	142	1	186	168	0	...	
2	0	120	546	0	283	622	4	645	330	0	...	
3	0	180	683	1	314	927	9	1202	430	0	...	
4	0	240	956	1	485	1264	11	1583	530	0	...	

5 rows × 32 columns

We have tons of data in this csv too, but this time all of the information is important for us to understand the game situation in some game time, such as 5 minutes after the game started. So we are not going to remove any data. Notes that the player id indicates which player it is, in radiant, it is 0,1,2,3,4 and in dire it is 128,129,130,131,132. For example, gold_t_0 means the total gold player_0 has at some game time. We are not going to change this because this is easy to use.

Now we have tons of data in here. We need to clean up is to remove all matches with leavers. Leavers mean that the player leaves the match before it ends, it usually has significant impact on game result. It is meaningless to analyze matches with leavers because the side with leavers are usually going to lose, even if they have advantage in early game. Therefore, it is meaningless to analyze leaver data.

In [7]:
```
df_player["leaver_status"].value_counts()
```

Out[7]:
```
0    488178
1      7019
2      2598
3      1670
4       535
Name: leaver_status, dtype: int64
```

Anything other than 0 means that the player leaves the game. Therefore, we removes all data other than 0.

We will use a list to contains all match_id with leavers, the remove all rows with match_id in that list.

In [8]:
```
# Record all match_id with leavers
leaver_array = df_player[df_player["leaver_status"] != 0]["match_id"].to_numpy
()
match_drop_row = []
player_drop_row = []
time_drop_row = []

# Loop through the dataframe to get rows that need to be droped
for i in range(0, len(df_match)):
    if df_match.iloc[i,0] in leaver_array:
        match_drop_row.append(i)
for i in range(0, len(df_player)):
    if df_player.iloc[i,0] in leaver_array:
        player_drop_row.append(i)
for i in range(0, len(df_player_time)):
    if df_player_time.iloc[i,0] in leaver_array:
        time_drop_row.append(i)

# Drop the rows
df_match = df_match.drop(match_drop_row)
df_player = df_player.drop(player_drop_row)
df_player_time = df_player_time.drop(time_drop_row)

# Clear data structures
match_drop_row= None
player_drop_row = None
time_drop_row = None
```

In [9]:
```
# Reset their indexes for easier access later.
df_match = df_match.reset_index(drop=True)
df_player = df_player.reset_index(drop=True)
df_player_time = df_player_time.reset_index(drop=True)
df_player.head()
```

Out[9]:

	match_id	hero_id	player_slot	gold	gold_spent	gold_per_min	xp_per_min	kills	deaths	a
0	0	86	0	3261	10960	347	362	9	3	
1	0	51	1	2954	17760	494	659	13	3	
2	0	83	2	110	12195	350	385	0	4	
3	0	11	3	1179	22505	599	605	8	4	
4	0	67	4	3307	23825	613	762	20	3	

5 rows × 24 columns

The result will be 3 clean dataframe, lets us check whether there are missing values in the dataframes.

In [10]:
```
print("There are " + str(len(df_match)) + ' matches in dataframe. Check missin
g values:')
print(df_match.isna().sum())
print(df_player.isna().sum())
print(df_player_time.isna().sum())
```

```
There are 42220 matches in dataframe. Check missing values:
match_id       0
duration       0
radiant_win    0
dtype: int64
match_id         0
hero_id          0
player_slot      0
gold             0
gold_spent       0
gold_per_min     0
xp_per_min       0
kills            0
deaths           0
assists          0
denies           0
last_hits        0
stuns            0
hero_damage      0
hero_healing     0
tower_damage     0
item_0           0
item_1           0
item_2           0
item_3           0
item_4           0
item_5           0
level            0
leaver_status    0
dtype: int64
match_id      0
times         0
gold_t_0      0
lh_t_0        0
xp_t_0        0
gold_t_1      0
lh_t_1        0
xp_t_1        0
gold_t_2      0
lh_t_2        0
xp_t_2        0
gold_t_3      0
lh_t_3        0
xp_t_3        0
gold_t_4      0
lh_t_4        0
xp_t_4        0
gold_t_128    0
lh_t_128      0
xp_t_128      0
gold_t_129    0
lh_t_129      0
xp_t_129      0
gold_t_130    0
lh_t_130      0
xp_t_130      0
gold_t_131    0
```

```
lh_t_131      0
xp_t_131      0
gold_t_132    0
lh_t_132      0
xp_t_132      0
dtype: int64
```

We are finished with Data cleaning. Now we have 42220 matches without leaver. Let us do some analyze here!

DATA ANALYSIS & RESULTS

Our goal is to predict winner at 5 mins, 10 mins, 15 mins and 20 mins.

a) Base on Gold/Experience only

First of all, let us do the easiest analysis. Let us predict the win/lose based on the current gold/experience. In another word, since more gold and more experience means more powerful your hero is, we predict that the side with more gold/experience would win the game.

Let us try 5 mins case with gold. In other words, we predict that at 5 mins, the side with more gold would win the game.

First step is to get all data at 5 mins, so we only want to see data at 5 mins. Select all rows with times = 300, which means 300s, 5 mins.

In [11]:
```
df_player_time_five = df_player_time[df_player_time.times == 300]
df_player_time_five = df_player_time_five.reset_index(drop=True)
df_player_time_five.head()
```

Out[11]:

	match_id	times	gold_t_0	lh_t_0	xp_t_0	gold_t_1	lh_t_1	xp_t_1	gold_t_2	lh_t_2	...	xp_
0	0	300	1056	1	649	1451	13	1810	630	0	...	
1	1	300	798	0	910	1479	20	1348	1118	8	...	
2	2	300	1140	12	1291	1518	14	1795	1103	6	...	
3	3	300	1007	4	1034	1140	9	700	957	9	...	
4	4	300	1739	28	1313	1888	16	2139	1013	11	...	

5 rows × 32 columns

Now we have 42220 matches at 5 mins. Now, we want to add up the gold in both sides to calculate the difference.

In [12]:
```
df_player_time_five["Radiant_gold_t"] = df_player_time_five["gold_t_0"] + df_p
layer_time_five["gold_t_1"] + df_player_time_five["gold_t_2"] + df_player_time
_five["gold_t_3"] + df_player_time_five["gold_t_4"]
df_player_time_five["Dire_gold_t"] = df_player_time_five["gold_t_128"] + df_pl
ayer_time_five["gold_t_129"] + df_player_time_five["gold_t_130"] + df_player_t
ime_five["gold_t_131"] + df_player_time_five["gold_t_132"]
df_player_time_five.head()
```

Out[12]:

	match_id	times	gold_t_0	lh_t_0	xp_t_0	gold_t_1	lh_t_1	xp_t_1	gold_t_2	lh_t_2	...	lh_
0	0	300	1056	1	649	1451	13	1810	630	0	...	
1	1	300	798	0	910	1479	20	1348	1118	8	...	
2	2	300	1140	12	1291	1518	14	1795	1103	6	...	
3	3	300	1007	4	1034	1140	9	700	957	9	...	
4	4	300	1739	28	1313	1888	16	2139	1013	11	...	

5 rows × 34 columns

Now, this dataset becomes more complex, we don't need all other data for this analysis, so we write all gold values to a new dataset. Also, calculate their difference in gold.

In [13]:
```
df_gold_five = df_player_time_five[["match_id","Radiant_gold_t", "Dire_gold_t"
]]
df_gold_five["diff"] = df_gold_five["Radiant_gold_t"] - df_gold_five["Dire_gol
d_t"]
df_gold_five.head()
```

Out[13]:

	match_id	Radiant_gold_t	Dire_gold_t	diff
0	0	5947	6984	-1037
1	1	5919	6740	-821
2	2	5468	4946	522
3	3	5048	5303	-255
4	4	6886	5131	1755

This looks much better! Now we need to get winner/loser data from df_match.

In [14]:
```
df_win = df_match[["match_id", "radiant_win"]]
df_gold_five = pd.merge(df_win, df_gold_five, on="match_id")
df_gold_five.head()
```

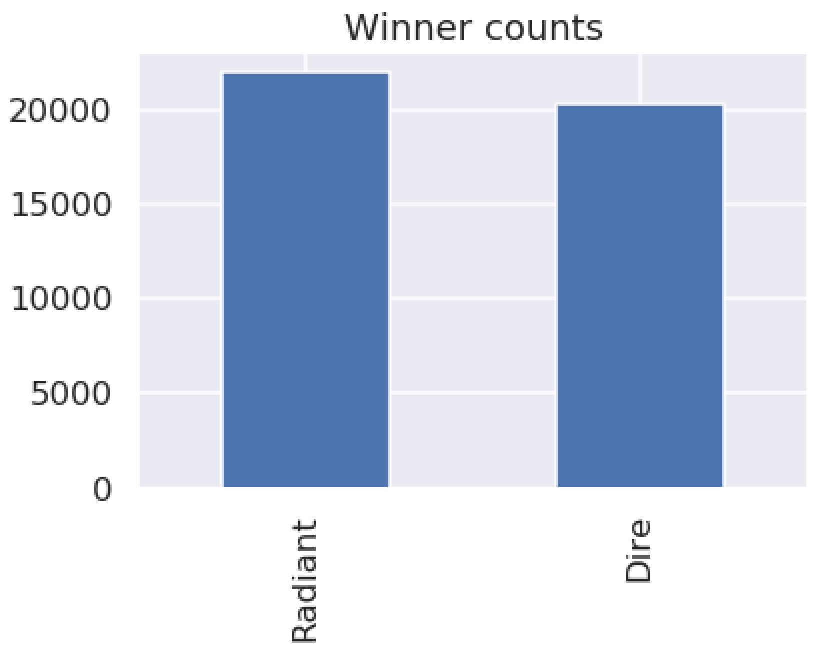
Out[14]:

	match_id	radiant_win	Radiant_gold_t	Dire_gold_t	diff
0	0	True	5947	6984	-1037
1	1	False	5919	6740	-821
2	2	False	5468	4946	522
3	3	False	5048	5303	-255
4	4	True	6886	5131	1755

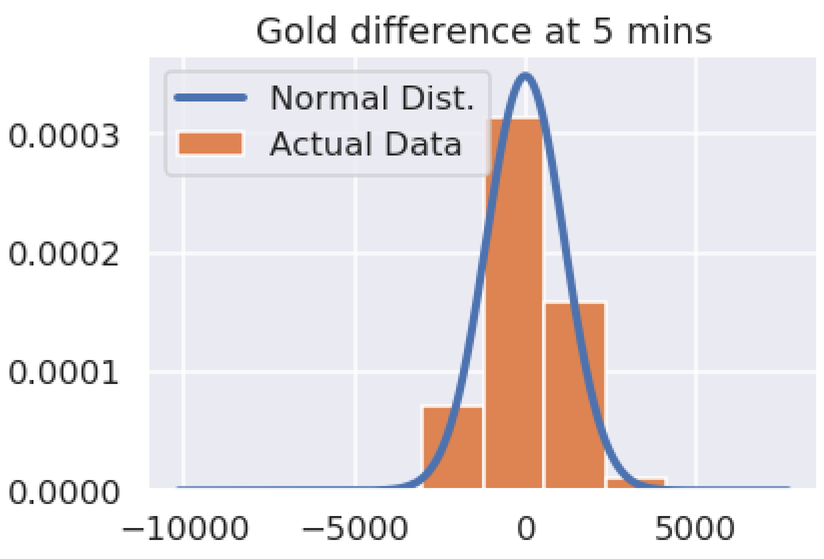
Ok, we will analysis on this dataset. It contains information about winner of the game and total gold on both side at 5 mins. Before we do any analysis, let us take a look at winner/loser and gold diff in graph. We will draw winner counts into a bar graph to see if any side has an advantage over the other side. We will draw the gold diff in a histgram and draw a noraml distrubution line to see if the data is normally distrubuted.

In [15]:
```
winner = df_gold_five["radiant_win"].value_counts()
winner = winner.rename({True: "Radiant", False: "Dire"})
winner.plot(kind="bar",title = "Winner counts")
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6bf59985c0>
```



In [16]:
```
xs = np.arange(df_gold_five["diff"].min(), df_gold_five["diff"].max(), 1)
fit = stats.norm.pdf(xs, np.mean(df_gold_five["diff"]), np.std(df_gold_five["d
iff"]))
# most easily done using matplotlib
plt.plot(xs, fit, label = 'Normal Dist.', lw = 4)
plt.hist(df_gold_five["diff"], density = True, label = 'Actual Data');
plt.title('Gold difference at 5 mins')
plt.legend();
```



Ok, this is what we expected. Radiant and Dire have almost equal chance to win the game, the gold distrubution is a normal distrubution around 0.

Let us change the difference in gold to a bool indicating which side has leading gold. True means radiant is leading and False means dire is leading.

In [17]:
```
def advantage(diff):
    if diff>0:
        return True
    else:
        return False
```

In [18]:
```
df_gold_five["Radiant_adv"] = df_gold_five["diff"].apply(advantage)
df_gold_five.head()
```

Out[18]:

	match_id	radiant_win	Radiant_gold_t	Dire_gold_t	diff	Radiant_adv
0	0	True	5947	6984	-1037	False
1	1	False	5919	6740	-821	False
2	2	False	5468	4946	522	True
3	3	False	5048	5303	-255	False
4	4	True	6886	5131	1755	True

We predict that the side with gold lead would win the match. If radiant is leading at 5 mins and radiant wins the game, or radiant is not leading and lost the game, then our prediction is correct. We put whether our prediction is correct in a new column called "association". True in association means that our prediction is correct, false means that it is wrong. Then calculate how many percent of the matches we have predicted correct.

In [19]:
```
df_gold_five["association"] = df_gold_five["Radiant_adv"] == df_gold_five["radiant_win"]
df_gold_five.head()
```

Out[19]:

	match_id	radiant_win	Radiant_gold_t	Dire_gold_t	diff	Radiant_adv	association
0	0	True	5947	6984	-1037	False	False
1	1	False	5919	6740	-821	False	True
2	2	False	5468	4946	522	True	False
3	3	False	5048	5303	-255	False	True
4	4	True	6886	5131	1755	True	True

In [20]:
```
print("Gold lead at 5 mins has {0:.0f}% chance to win the game".format(df_gold_five["association"].value_counts()[1] / len(df_gold_five)* 100))
```

```
Gold lead at 5 mins has 60% chance to win the game
```

As we can see, this only have 60 percent accuracy, which is not good enough. But the gold lead did show that the team has some advantage, therefore, a higher chance to win the game.

Now, let us try experience and other game time. The code is similar to the above, we just changed gold to experience.

In [21]:
```
# Determine the relationship between experience lead and winning
df_player_time_five["Radiant_xp_t"] = df_player_time_five["xp_t_0"] + df_player_time_five["xp_t_1"] + df_player_time_five["xp_t_2"] + df_player_time_five["xp_t_3"] + df_player_time_five["xp_t_4"]
df_player_time_five["Dire_xp_t"] = df_player_time_five["xp_t_128"] + df_player_time_five["xp_t_129"] + df_player_time_five["xp_t_130"] + df_player_time_five["xp_t_131"] + df_player_time_five["xp_t_132"]
df_xp_five = df_player_time_five[["match_id","Radiant_xp_t", "Dire_xp_t"]]
df_xp_five["diff"] = df_xp_five["Radiant_xp_t"] - df_xp_five["Dire_xp_t"]
df_xp_five = pd.merge(df_win, df_xp_five, on="match_id")
df_xp_five["Radiant_adv"] = df_xp_five["diff"].apply(advantage)
df_xp_five["association"] = df_xp_five["Radiant_adv"] == df_xp_five["radiant_win"]
print("Experience lead at 5 mins has {0:.0f}% chance to win the game".format(df_xp_five["association"].value_counts()[1] / len(df_xp_five)* 100))
```

```
Experience lead at 5 mins has 58% chance to win the game
```

Apparently, experience has less effects on the win prediction than gold. Then let us try other game time. 10mins, 15mins, and 20mins.

In [22]:
```
# First ten mins gold/experience
df_player_time_ten = df_player_time[df_player_time.times == 600]
df_player_time_ten = df_player_time_ten.reset_index(drop=True)

df_player_time_ten["Radiant_gold_t"] = df_player_time_ten["gold_t_0"] + df_pla
yer_time_ten["gold_t_1"] + df_player_time_ten["gold_t_2"] + df_player_time_ten
["gold_t_3"] + df_player_time_ten["gold_t_4"]
df_player_time_ten["Dire_gold_t"] = df_player_time_ten["gold_t_128"] + df_play
er_time_ten["gold_t_129"] + df_player_time_ten["gold_t_130"] + df_player_time_
ten["gold_t_131"] + df_player_time_ten["gold_t_132"]
df_gold_ten = df_player_time_ten[["match_id","Radiant_gold_t", "Dire_gold_t"]]
df_gold_ten["diff"] = df_gold_ten["Radiant_gold_t"] - df_gold_ten["Dire_gold_
t"]
df_gold_ten = pd.merge(df_win, df_gold_ten, on="match_id")
df_gold_ten["Radiant_adv"] = df_gold_ten["diff"].apply(advantage)
df_gold_ten["association"] = df_gold_ten["Radiant_adv"] == df_gold_ten["radian
t_win"]
print("Gold lead at 10 mins has {0:.0f}% chance to win the game".format(df_gol
d_ten["association"].value_counts()[1] / len(df_gold_ten)* 100))

df_player_time_ten["Radiant_xp_t"] = df_player_time_ten["xp_t_0"] + df_player_
time_ten["xp_t_1"] + df_player_time_ten["xp_t_2"] + df_player_time_ten["xp_t_
3"] + df_player_time_ten["xp_t_4"]
df_player_time_ten["Dire_xp_t"] = df_player_time_ten["xp_t_128"] + df_player_t
ime_ten["xp_t_129"] + df_player_time_ten["xp_t_130"] + df_player_time_ten["xp_
t_131"] + df_player_time_ten["xp_t_132"]
df_xp_ten = df_player_time_ten[["match_id","Radiant_xp_t", "Dire_xp_t"]]
df_xp_ten["diff"] = df_xp_ten["Radiant_xp_t"] - df_xp_ten["Dire_xp_t"]
df_xp_ten = pd.merge(df_win, df_xp_ten, on="match_id")
df_xp_ten["Radiant_adv"] = df_xp_ten["diff"].apply(advantage)
df_xp_ten["association"] = df_xp_ten["Radiant_adv"] == df_xp_ten["radiant_win"
]
print("Experience lead at 10 mins has {0:.0f}% chance to win the game".format(
df_xp_ten["association"].value_counts()[1] / len(df_xp_ten)* 100))

# 15 mins gold/experience
df_player_time_15 = df_player_time[df_player_time.times == 900]
df_player_time_15 = df_player_time_15.reset_index(drop=True)

df_player_time_15["Radiant_gold_t"] = df_player_time_15["gold_t_0"] + df_playe
r_time_15["gold_t_1"] + df_player_time_15["gold_t_2"] + df_player_time_15["gol
d_t_3"] + df_player_time_15["gold_t_4"]
df_player_time_15["Dire_gold_t"] = df_player_time_15["gold_t_128"] + df_player
_time_15["gold_t_129"] + df_player_time_15["gold_t_130"] + df_player_time_15[
"gold_t_131"] + df_player_time_15["gold_t_132"]
df_gold_15 = df_player_time_15[["match_id","Radiant_gold_t", "Dire_gold_t"]]
df_gold_15["diff"] = df_gold_15["Radiant_gold_t"] - df_gold_15["Dire_gold_t"]
df_gold_15 = pd.merge(df_win, df_gold_15, on="match_id")
df_gold_15["Radiant_adv"] = df_gold_15["diff"].apply(advantage)
df_gold_15["association"] = df_gold_15["Radiant_adv"] == df_gold_15["radiant_w
in"]
print("Gold lead at 15 mins has {0:.0f}% chance to win the game".format(df_gol
d_15["association"].value_counts()[1] / len(df_gold_15)* 100))

df_player_time_15["Radiant_xp_t"] = df_player_time_15["xp_t_0"] + df_player_ti
me_15["xp_t_1"] + df_player_time_15["xp_t_2"] + df_player_time_15["xp_t_3"] +
df_player_time_15["xp_t_4"]
df_player_time_15["Dire_xp_t"] = df_player_time_15["xp_t_128"] + df_player_tim
e_15["xp_t_129"] + df_player_time_15["xp_t_130"] + df_player_time_15["xp_t_13
1"] + df_player_time_15["xp_t_132"]
df_xp_15 = df_player_time_15[["match_id","Radiant_xp_t", "Dire_xp_t"]]
df_xp_15["diff"] = df_xp_15["Radiant_xp_t"] - df_xp_15["Dire_xp_t"]
df_xp_15 = pd.merge(df_win, df_xp_15, on="match_id")
df_xp_15["Radiant_adv"] = df_xp_15["diff"].apply(advantage)
df_xp_15["association"] = df_xp_15["Radiant_adv"] == df_xp_15["radiant_win"]
print("Experience lead at 15 mins has {0:.0f}% chance to win the game".format(
df_xp_15["association"].value_counts()[1] / len(df_xp_15)* 100))

# 20 mins gold/experience
df_player_time_20 = df_player_time[df_player_time.times == 1200]
df_player_time_20 = df_player_time_20.reset_index(drop=True)

df_player_time_20["Radiant_gold_t"] = df_player_time_20["gold_t_0"] + df_playe
r_time_20["gold_t_1"] + df_player_time_20["gold_t_2"] + df_player_time_20["gol
d_t_3"] + df_player_time_20["gold_t_4"]
df_player_time_20["Dire_gold_t"] = df_player_time_20["gold_t_128"] + df_player
_time_20["gold_t_129"] + df_player_time_20["gold_t_130"] + df_player_time_20[
"gold_t_131"] + df_player_time_20["gold_t_132"]
df_gold_20 = df_player_time_20[["match_id","Radiant_gold_t", "Dire_gold_t"]]
df_gold_20["diff"] = df_gold_20["Radiant_gold_t"] - df_gold_20["Dire_gold_t"]
df_gold_20 = pd.merge(df_win, df_gold_20, on="match_id")
df_gold_20["Radiant_adv"] = df_gold_20["diff"].apply(advantage)
df_gold_20["association"] = df_gold_20["Radiant_adv"] == df_gold_20["radiant_w
in"]
print("Gold lead at 20 mins has {0:.0f}% chance to win the game".format(df_gol
d_20["association"].value_counts()[1] / len(df_gold_20)* 100))

df_player_time_20["Radiant_xp_t"] = df_player_time_20["xp_t_0"] + df_player_ti
me_20["xp_t_1"] + df_player_time_20["xp_t_2"] + df_player_time_20["xp_t_3"] +
df_player_time_20["xp_t_4"]
df_player_time_20["Dire_xp_t"] = df_player_time_20["xp_t_128"] + df_player_tim
e_20["xp_t_129"] + df_player_time_20["xp_t_130"] + df_player_time_20["xp_t_13
1"] + df_player_time_20["xp_t_132"]
df_xp_20 = df_player_time_20[["match_id","Radiant_xp_t", "Dire_xp_t"]]
df_xp_20["diff"] = df_xp_20["Radiant_xp_t"] - df_xp_20["Dire_xp_t"]
df_xp_20 = pd.merge(df_win, df_xp_20, on="match_id")
df_xp_20["Radiant_adv"] = df_xp_20["diff"].apply(advantage)
df_xp_20["association"] = df_xp_20["Radiant_adv"] == df_xp_20["radiant_win"]
print("Experience lead at 20 mins has {0:.0f}% chance to win the game".format(
df_xp_20["association"].value_counts()[1] / len(df_xp_20)* 100))
```

```
Gold lead at 10 mins has 65% chance to win the game
Experience lead at 10 mins has 63% chance to win the game
Gold lead at 15 mins has 69% chance to win the game
Experience lead at 15 mins has 68% chance to win the game
Gold lead at 20 mins has 73% chance to win the game
Experience lead at 20 mins has 73% chance to win the game
```

As we can see, as time increase, gold lead had a greater chance to win the game. However, gold and experience alone are not good predictors. We need to add more variables to it.

b) Add Hero selection to the model

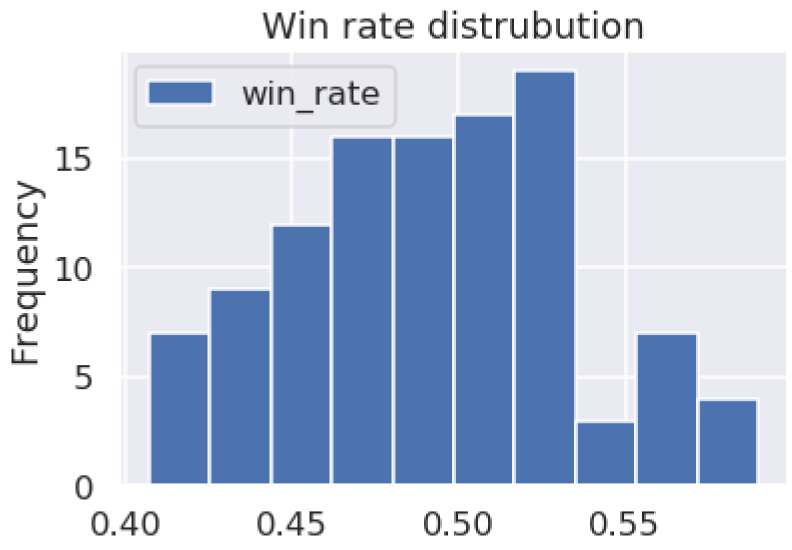
Note: This section largely borrowed the idea from http://jmcauley.ucsd.edu/cse255/projects/fa15/018.pdf (http://jmcauley.ucsd.edu/cse255/projects/fa15/018.pdf)

Some heros just have higher win rate than other heros. So we need to calculate the win rate of each hero and add this factor to our analysis.

First read all match data and get the win rate of each hero. And take a look at the distrubution.

In [23]:
```
# We will store the data in a dictionary whose key will be hero_id, and whose
 items will be a pair of (win count, total count)
heros = {}

# Loop through all matches
for i in range(0,len(df_match)):

    # Get the 10 rows of that match data
    df_hero = df_player[i*10: (i+1)*10]

    # Each hero total count + 1
    for j in range(0,len(df_hero)):
        hero_id = df_hero.iloc[j,1]
        if hero_id in heros.keys():
            (win, total) = heros[hero_id]
            heros[hero_id] = (win, total+1)
        else:
            heros[hero_id] = (0,1)

    # Add the win count to the winner side
    if df_match.iloc[i,2]:
        for j in range(0, 5):
            if df_hero.iloc[j,2] < 100:
                hero_id = df_hero.iloc[j,1]
                (win, total) = heros[hero_id]
                heros[hero_id] = (win+1, total)
            else:
                print("error")
    else:
        for j in range(5, 10):
            if df_hero.iloc[j,2] > 100:
                hero_id = df_hero.iloc[j,1]
                (win, total) = heros[hero_id]
                heros[hero_id] = (win+1, total)
            else:
                print("error")

# Transform (win count, total count) pair into winrate using win/total
heros_winrate = {}
for i in heros.keys():
    heros_winrate[i] = heros[i][0] / heros[i][1]
heros = None
```

In [24]:
```
df_hero = pd.DataFrame.from_dict(heros_winrate, orient='index', columns=["win_
rate"])
df_hero = df_hero.sort_values(by=["win_rate"])
df_hero.plot(kind="hist",title = "Win rate distrubution")

df_hero = None
```



As we can see, hero win rate range from 40% to 59%. There are some difference between the win rate of each hero. Therefore, we need to build this into our model.

Now, we have hero win rate. But also, hero composition is very important. Two heros may have good or bad synergy in the same team. Two heros may form a counter relationship in opposite team. Therefore, we want to calculate all of them.

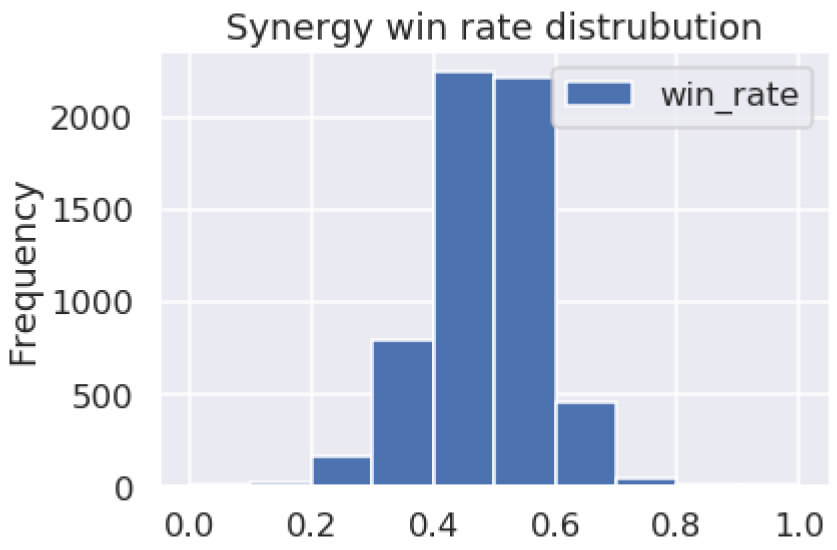
We will use similar approach as the above, count total and win to get win rate.

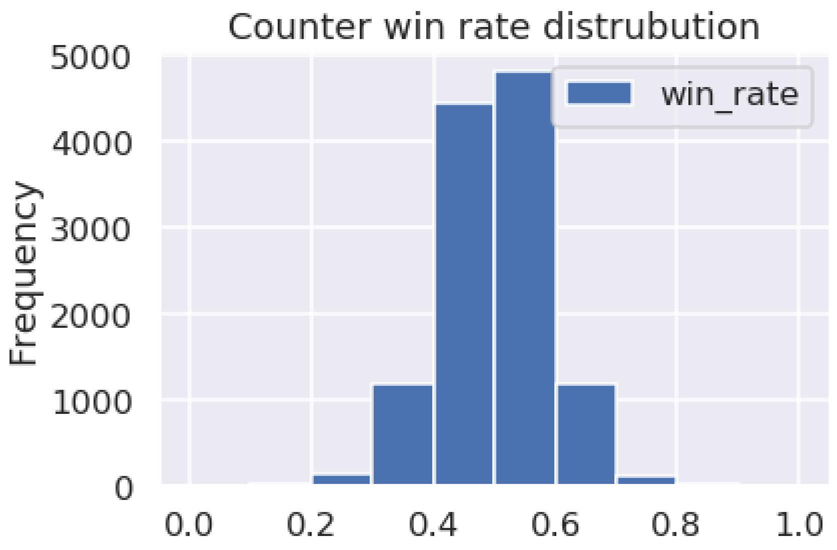
In [25]:
```
heros_synergy = {}
heros_counter = {}

# Loop through all matches
for i in range(0,len(df_match)):
    df_hero = df_player[i*10: (i+1)*10]
    radiant_win = df_match.iloc[i,2]

    # For radiant, all hero composition total count + 1, if radiant win, their
win count + 1
    for j in range(0, 5):
        for k in range(0, 5):
            if j != k:
                hero_j = df_hero.iloc[j,1]
                hero_k = df_hero.iloc[k,1]
                key = sorted([hero_j, hero_k])
                key = str(key[0]) + " " + str(key[1])
                if key in heros_synergy.keys():
                    (win, total) = heros_synergy[key]
                    heros_synergy[key] = (win + int(radiant_win), total + 1)
                else:
                    heros_synergy[key] = (int(radiant_win), 1)

    # The same for dire
    for j in range(5, 10):
        for k in range(5, 10):
            if j != k:
                hero_j = df_hero.iloc[j,1]
                hero_k = df_hero.iloc[k,1]
                key = sorted([hero_j, hero_k])
                key = str(key[0]) + " " + str(key[1])
                if key in heros_synergy.keys():
                    (win, total) = heros_synergy[key]
                    heros_synergy[key] = (win + int(not radiant_win), total +
1)
                else:
                    heros_synergy[key] = (int(not radiant_win), 1)

    # Now, calculate all hero counter situation. all hero counter total count
 + 1 (and the reverse because counter is a mutual relationship)
    # add the counter win count + 1 on the winner side
    for j in range(0, 5):
        for k in range(5, 10):
            hero_j = df_hero.iloc[j,1]
            hero_k = df_hero.iloc[k,1]
            key = [hero_j, hero_k]
            key = str(key[0]) + " " + str(key[1])
            if key in heros_counter.keys():
                (win, total) = heros_counter[key]
                heros_counter[key] = (win + int(radiant_win), total + 1)
            else:
                heros_counter[key] = (int(radiant_win), 1)
            key = [hero_k, hero_j]
            key = str(key[0]) + " " + str(key[1])
            if key in heros_counter.keys():
                (win, total) = heros_counter[key]
                heros_counter[key] = (win + int(not radiant_win), total + 1)
            else:
                heros_counter[key] = (int(not radiant_win), 1)
```

In [26]:
```
# Transform all data into winrate
synergy_winrate = {}
for i in heros_synergy.keys():
    synergy_winrate[i] = heros_synergy[i][0] / heros_synergy[i][1]
heros_synergy = None

counter_winrate = {}
for i in heros_counter.keys():
    counter_winrate[i] = heros_counter[i][0] / heros_counter[i][1]
heros_counter = None
```

Let us plot them out to see if the distrubution of win rate.

In [27]:
```
df_synergy = pd.DataFrame.from_dict(synergy_winrate, orient='index', columns=[
"win_rate"])
df_synergy = df_synergy.sort_values(by=["win_rate"])
df_synergy.plot(kind="hist",title = "Synergy win rate distrubution")
df_synergy = None

df_counter = pd.DataFrame.from_dict(counter_winrate, orient='index', columns=[
"win_rate"])
df_counter = df_counter.sort_values(by=["win_rate"])
df_counter.plot(kind="hist",title = "Counter win rate distrubution")
df_counter = None
```





As we can see, there are extreme cases exists in both synergy and counter cases. Therefore, this is very important to our model as well.

Now, we need to calculate hero score, synergy score and counter score for all matches. For hero score, just add up hero win rates. For synergy score, add up win rate of every possible synergy pair in one team. For counter score, add up win rate of every possible counter pair in a match.

Then we need to put all data into one data frame to run machine learning. Note that we don't need dire counter because it is just the opposite of radiant_counter score.

In [28]:
```
# Prepare the slots for the data
df_match.insert(3,"radiant_score",0)
df_match.insert(4,"dire_score",0)
df_match.insert(5,"radiant_synergy",0)
df_match.insert(6,"dire_synergy",0)
df_match.insert(7,"radiant_counter",0)
```

In [29]:
```
# Loop through all matches
for i in range(0,len(df_match)):
    df_hero = df_player[i*10: (i+1)*10]
    radiant_score = 0
    dire_score = 0
    radiant_synergy = 0
    dire_synergy = 0
    radiant_counter = 0

    # Count radiant and dire hero score
    for j in range(0, 5):
        hero_id = df_hero.iloc[j,1]
        radiant_score = radiant_score + heros_winrate[hero_id]
    for j in range(5, 10):
        hero_id = df_hero.iloc[j,1]
        dire_score = dire_score + heros_winrate[hero_id]
    df_match.iloc[i, 3] = radiant_score
    df_match.iloc[i, 4] = dire_score

    # Count their synergy score and counter score
    for j in range(0, 5):
        for k in range(0, 5):
            if j != k:
                hero_j = df_hero.iloc[j,1]
                hero_k = df_hero.iloc[k,1]
                key = sorted([hero_j, hero_k])
                key = str(key[0]) + " " + str(key[1])
                radiant_synergy = radiant_synergy + synergy_winrate[key]
    for j in range(5, 10):
        for k in range(5,10):
            if j != k:
                hero_j = df_hero.iloc[j,1]
                hero_k = df_hero.iloc[k,1]
                key = sorted([hero_j, hero_k])
                key = str(key[0]) + " " + str(key[1])
                dire_synergy = dire_synergy + synergy_winrate[key]
    for j in range(0, 5):
        for k in range(5, 10):
            hero_j = df_hero.iloc[j,1]
            hero_k = df_hero.iloc[k,1]
            key = [hero_j, hero_k]
            key = str(key[0]) + " " + str(key[1])
            radiant_counter = radiant_counter + counter_winrate[key]
    df_match.iloc[i, 5] = radiant_synergy
    df_match.iloc[i, 6] = dire_synergy
    df_match.iloc[i, 7] = radiant_counter
df_match.head()
```

Out[29]:

	match_id	duration	radiant_win	radiant_score	dire_score	radiant_synergy	dire_synergy	rad
0	0	2375	True	2.574347	2.592252	10.524785	10.411289	
1	1	2582	False	2.455522	2.684388	9.535232	11.359371	
2	2	2716	False	2.504080	2.475058	8.700309	10.032232	
3	3	3085	False	2.454647	2.564438	9.569182	10.565340	
4	4	1887	True	2.531044	2.558996	10.313792	10.631068	

Now, add the total gold and experience columns from section a). We will use this dataframe to build machine learning model. X is all the columns after radiant_win and y is radiant_win.

In [30]:

```
# Put all required data into one dataframe
df_score = df_match[["match_id","radiant_win","radiant_score","dire_score","ra
diant_synergy","dire_synergy","radiant_counter"]]
df_gold_diff = df_gold_five[["match_id", "Radiant_gold_t", "Dire_gold_t"]]
df_xp_diff = df_xp_five[["match_id", "Radiant_xp_t", "Dire_xp_t"]]
df_new_five = pd.merge(df_score, df_gold_diff, on="match_id")
df_new_five = pd.merge(df_new_five, df_xp_diff, on="match_id")
df_score = None
df_gold_diff = None
df_xp_diff = None
df_new_five.head()
```

Out[30]:

	match_id	radiant_win	radiant_score	dire_score	radiant_synergy	dire_synergy	radiant_count
0	0	True	2.574347	2.592252	10.524785	10.411289	12.3458
1	1	False	2.455522	2.684388	9.535232	11.359371	11.3309
2	2	False	2.504080	2.475058	8.700309	10.032232	12.6311
3	3	False	2.454647	2.564438	9.569182	10.565340	11.8980
4	4	True	2.531044	2.558996	10.313792	10.631068	12.2159

Now, we seperate them into training and testing group. We will use 80% of data as training and 20% of data as testing.

In [31]:
```
num_train = int(42220*0.8)
num_test = int(42220*0.2)

df_train = df_new_five[:num_train]
df_test = df_new_five[num_train:]

train_X = df_train[["radiant_score","dire_score","Radiant_gold_t","Dire_gold_
t","Radiant_xp_t","Dire_xp_t","radiant_synergy","dire_synergy","radiant_counte
r"]].as_matrix()
train_y = df_train["radiant_win"].values

test_X = df_test[["radiant_score","dire_score","Radiant_gold_t","Dire_gold_t",
"Radiant_xp_t","Dire_xp_t","radiant_synergy","dire_synergy","radiant_counter"
]].as_matrix()
test_y = df_test["radiant_win"].values
```

In [32]:
```
def train_SVM(X, y, kernel='linear'):
    clf = SVC(kernel=kernel)
    clf.fit(X, y)
    return clf
```

In [33]:
```
Five_clf = train_SVM(train_X, train_y)
```

In [34]:
```
predicted_train_y = Five_clf.predict(train_X)
predicted_test_y = Five_clf.predict(test_X)
print(classification_report(train_y,predicted_train_y))
print(classification_report(test_y,predicted_test_y))
Five_clf = None
```

```
              precision    recall  f1-score   support

       False       0.68      0.65      0.67     16289
        True       0.69      0.72      0.70     17487

    accuracy                           0.69     33776
   macro avg       0.69      0.68      0.68     33776
weighted avg       0.69      0.69      0.69     33776

              precision    recall  f1-score   support

       False       0.69      0.66      0.67      4009
        True       0.70      0.73      0.71      4435

    accuracy                           0.69      8444
   macro avg       0.69      0.69      0.69      8444
weighted avg       0.69      0.69      0.69      8444
```

This takes years to run, but finally we got our result.

As we can see, the score increased a lot, but it is still below our expectation, even it is close. We need to add more things to our model. (We also tried to use hero selection information only and it yields simialr result. We don't think this is good enough)

c) Player skill level

We have ran the machine learning model with gold and experience difference and hero selection. The last thing we wanted to add on our model is the skill of the players. The skill is a vague concept and could hardly be represented by any data. However, we can compare how well a player plays to other players who play the same hero. Using this method, we can get a score about how well a player is. Then we record this score and add it to our machine learning model.

The first thing that we want to do is to see how well other players play this hero. Take 5 mins as an example. We take all data that record the gold/experience at 5 mins for this hero, and calculated mean. Then using these value, we can determine how well a player is at 5 mins. Note that we will only record the matches that wins with that hero because the loser's match doesn't provide much information. Also, we use gold and experience because they are both good indicators of skill level.

First of all, let us record all hero gold/experience at 5 mins. We will use a similar approach as before. We will set up a list of 112 indexes, representing 112 heros, so that we can use hero_id to access this list. Then loop through all matches, add the gold to the total gold of the heros that win, then add the count + 1.

In [35]:
```
hero_gold_five = []
hero_experience_five = []

# Prepare the lists
for i in range(0, 113):
    hero_gold_five.append((0, 0))
    hero_experience_five.append((0, 0))

# Loop through all matches
for i in range(0,len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_five.iloc[i]

    # If radiant wins, add radiant heros' gold at 5 mins to total and count +
 1
    for j in range(0,5):
        if df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_five = df_condition.iloc[2 + j*3]
            xp_five = df_condition.iloc[4 + j*3]
            (total, count) = hero_gold_five[hero_id]
            hero_gold_five[hero_id] = (total + gold_five, count + 1)
            (total, count) = hero_experience_five[hero_id]
            hero_experience_five[hero_id]  = (total + xp_five, count + 1)

    # The same for dire
    for j in range(5,10):
        if not df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_five = df_condition.iloc[2 + j*3]
            xp_five = df_condition.iloc[4 + j*3]
            (total, count) = hero_gold_five[hero_id]
            hero_gold_five[hero_id] = (total + gold_five, count + 1)
            (total, count) = hero_experience_five[hero_id]
            hero_experience_five[hero_id]  = (total + xp_five, count + 1)
```

In [36]:
```
gold_five_stat = []
xp_five_stat = []

# Transform the data into average gold of winners.
for i in range(0, 113):
    if hero_gold_five[i][1] != 0:
        mean = hero_gold_five[i][0] / hero_gold_five[i][1]
        gold_five_stat.append(mean)

        mean = hero_experience_five[i][0] / hero_experience_five[i][1]
        xp_five_stat.append(mean)

    else:
        empty_tuple = ()
        gold_five_stat.append(empty_tuple)
        xp_five_stat.append(empty_tuple)
hero_gold_five = None
hero_experience_five = None
```

We have established a profile of the average skill level of the players of each hero. Since SVC only accept positive numbers, we will use gold/experience divides mean to get a score, representing the performance of a place. Therefore, we need to calculate how each players play at 5 mins for each match and sum them up as the team skill level.

In [37]:
```
def skill_level(hero_id, gold, xp, gold_stat, xp_stat):
    mean = gold_stat[hero_id]
    p_gold = (gold / mean)
    mean = xp_stat[hero_id]
    p_xp = (xp / mean)
    return (p_gold, p_xp)
```

In [38]:
```
radiant_gold_list = []
radiant_xp_list = []
dire_gold_list = []
dire_xp_list = []

# Loop through all matches
for i in range(0, len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_five.iloc[i]

    radiant_gold = 0
    radiant_xp = 0

    dire_gold = 0
    dire_xp = 0

    # Sum up radiant gold/experience skill score
    for j in range(0, 5):
        hero_id = df_hero.iloc[j,1]
        gold_five = df_condition.iloc[2 + j*3]
        xp_five = df_condition.iloc[4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_five, xp_five, gold_five_st
at, xp_five_stat)
        radiant_gold = radiant_gold + p_gold
        radiant_xp = radiant_xp + p_xp

    # The same for dire
    for j in range(5, 10):
        hero_id = df_hero.iloc[j,1]
        gold_five = df_condition.iloc[2 + j*3]
        xp_five = df_condition.iloc[4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_five, xp_five, gold_five_st
at, xp_five_stat)
        dire_gold = dire_gold + p_gold
        dire_xp = dire_xp + p_xp

    radiant_gold_list.append(radiant_gold)
    radiant_xp_list.append(radiant_xp)
    dire_gold_list.append(dire_gold)
    dire_xp_list.append(dire_xp)
```

Now, we have the skill level score for both sides for all matches. We need to plug them into our model and try again. Use the same model as before, add the four new variables that we just calculated. But this time, we will not use total gold/experience as variables because skill level = current_gold / mean_gold. The sum of current gold is total gold. Because the total gold did not divide the factor, it contains less information than skill level and would disrupt our model. Therefore, we only use hero selection and skill level to build our model.

In [39]:
```
df_new_five.insert(4, "radiant_gold_level", radiant_gold_list, True)
df_new_five.insert(5, "radiant_xp_level", radiant_xp_list, True)
df_new_five.insert(6, "dire_gold_level", dire_gold_list, True)
df_new_five.insert(7, "dire_xp_level", dire_xp_list, True)
radiant_gold_list = None
radiant_xp_list = None
dire_gold_list = None
dire_xp_list = None

df_train = df_new_five[:num_train]
df_test = df_new_five[num_train:]

train_X = df_train[["radiant_score","dire_score","radiant_gold_level","radiant
_xp_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy",
"radiant_counter"]].as_matrix()
train_y = df_train["radiant_win"].values

test_X = df_test[["radiant_score","dire_score","radiant_gold_level","radiant_x
p_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy","r
adiant_counter"]].as_matrix()
test_y = df_test["radiant_win"].values

df_new_five.head()
```

Out[39]:

	match_id	radiant_win	radiant_score	dire_score	radiant_gold_level	radiant_xp_level	dire_gol
0	0	True	2.574347	2.592252	5.381976	4.980337	4.
1	1	False	2.455522	2.684388	4.923641	4.164652	4.
2	2	False	2.504080	2.475058	4.767575	5.372707	4.
3	3	False	2.454647	2.564438	4.978273	4.861272	4.
4	4	True	2.531044	2.558996	5.474070	4.766156	4.

In [40]:
```
Five_clf = train_SVM(train_X, train_y)
```

In [41]:
```
predicted_train_y = Five_clf.predict(train_X)
predicted_test_y = Five_clf.predict(test_X)
print(classification_report(train_y,predicted_train_y))
print(classification_report(test_y,predicted_test_y))
Five_clf = None
```

```
              precision    recall  f1-score   support

       False       0.74      0.73      0.73     16289
        True       0.75      0.76      0.76     17487

    accuracy                           0.75     33776
   macro avg       0.74      0.74      0.74     33776
weighted avg       0.74      0.75      0.74     33776

              precision    recall  f1-score   support

       False       0.74      0.72      0.73      4009
        True       0.75      0.77      0.76      4435

    accuracy                           0.75      8444
   macro avg       0.75      0.75      0.75      8444
weighted avg       0.75      0.75      0.75      8444
```

Ahoy! It seems that we have achieved our goal of 70% accuracy rate! Thanks for reading this until so far. Now we just need to do it again with 10 mins and 15 mins (we don't want to do 20 mins because prediction on gold only has already achieved our goal. The code is the same as the above except we use 10 mins and 15 mins data.

In [42]:
```
hero_gold_ten = []
hero_experience_ten = []

for i in range(0, 113):
    hero_gold_ten.append((0,0))
    hero_experience_ten.append((0,0))

for i in range(0,len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_ten.iloc[i]

    for j in range(0,5):
        if df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_ten = df_condition.iloc[2 + j*3]
            xp_ten = df_condition.iloc[4 + j*3]
            (total, count) = hero_gold_ten[hero_id]
            hero_gold_ten[hero_id] = (total + gold_ten, count + 1)
            (total, count) = hero_experience_ten[hero_id]
            hero_experience_ten[hero_id]  = (total + xp_ten, count + 1)

    for j in range(5,10):
        if not df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_ten = df_condition.iloc[2 + j*3]
            xp_ten = df_condition.iloc[4 + j*3]
            (total, count) = hero_gold_ten[hero_id]
            hero_gold_ten[hero_id] = (total + gold_ten, count + 1)
            (total, count) = hero_experience_ten[hero_id]
            hero_experience_ten[hero_id]  = (total + xp_ten, count + 1)

gold_ten_stat = []
xp_ten_stat = []

for i in range(0, 113):
    if hero_gold_ten[i][1] != 0:
        mean = hero_gold_ten[i][0] / hero_gold_ten[i][1]
        gold_ten_stat.append(mean)

        mean = hero_experience_ten[i][0] / hero_experience_ten[i][1]
        xp_ten_stat.append(mean)

    else:
        empty_tuple = ()
        gold_ten_stat.append(empty_tuple)
        xp_ten_stat.append(empty_tuple)
hero_gold_ten = None
hero_experience_ten = None
```

In [43]:
```
df_score = df_match[["match_id","radiant_win","radiant_score","dire_score","ra
diant_synergy","dire_synergy","radiant_counter"]]
df_gold_diff = df_gold_ten[["match_id", "Radiant_gold_t", "Dire_gold_t"]]
df_xp_diff = df_xp_ten[["match_id", "Radiant_xp_t", "Dire_xp_t"]]
df_new_ten = pd.merge(df_score, df_gold_diff, on="match_id")
df_new_ten = pd.merge(df_new_ten, df_xp_diff, on="match_id")
df_score = None
df_gold_diff = None
df_xp_diff = None

radiant_gold_list = []
radiant_xp_list = []
dire_gold_list = []
dire_xp_list = []
for i in range(0, len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_ten.iloc[i]

    radiant_gold = 0
    radiant_xp = 0

    dire_gold = 0
    dire_xp = 0

    for j in range(0, 5):
        hero_id = df_hero.iloc[j,1]
        gold_ten = df_condition.iloc[2 + j*3]
        xp_ten = df_condition.iloc[4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_ten, xp_ten, gold_ten_stat,
xp_ten_stat)
        radiant_gold = radiant_gold + p_gold
        radiant_xp = radiant_xp + p_xp

    for j in range(5, 10):
        hero_id = df_hero.iloc[j,1]
        gold_ten = df_condition.iloc[2 + j*3]
        xp_ten = df_condition.iloc[4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_ten, xp_ten, gold_ten_stat,
xp_ten_stat)
        dire_gold = dire_gold + p_gold
        dire_xp = dire_xp + p_xp

    radiant_gold_list.append(radiant_gold)
    radiant_xp_list.append(radiant_xp)
    dire_gold_list.append(dire_gold)
    dire_xp_list.append(dire_xp)
```

In [44]:
```
df_new_ten.insert(4, "radiant_gold_level", radiant_gold_list, True)
df_new_ten.insert(5, "radiant_xp_level", radiant_xp_list, True)
df_new_ten.insert(6, "dire_gold_level", dire_gold_list, True)
df_new_ten.insert(7, "dire_xp_level", dire_xp_list, True)
radiant_gold_list = None
radiant_xp_list = None
dire_gold_list = None
dire_xp_list = None

df_train = df_new_ten[:num_train]
df_test = df_new_ten[num_train:]

train_X = df_train[["radiant_score","dire_score","radiant_gold_level","radiant
_xp_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy",
"radiant_counter"]].as_matrix()
train_y = df_train["radiant_win"].values

test_X = df_test[["radiant_score","dire_score","radiant_gold_level","radiant_x
p_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy","r
adiant_counter"]].as_matrix()
test_y = df_test["radiant_win"].values
```

In [45]:
```
Ten_clf = train_SVM(train_X, train_y)
```

In [46]:
```
predicted_train_y = Ten_clf.predict(train_X)
predicted_test_y = Ten_clf.predict(test_X)
print(classification_report(train_y,predicted_train_y))
print(classification_report(test_y,predicted_test_y))
Ten_clf = None
```

```
              precision    recall  f1-score   support

       False       0.75      0.74      0.74     16289
        True       0.76      0.77      0.77     17487

    accuracy                           0.76     33776
   macro avg       0.76      0.75      0.75     33776
weighted avg       0.76      0.76      0.76     33776

              precision    recall  f1-score   support

       False       0.76      0.75      0.75      4009
        True       0.77      0.78      0.78      4435

    accuracy                           0.77      8444
   macro avg       0.77      0.76      0.77      8444
weighted avg       0.77      0.77      0.77      8444
```

It seems that 10 mins has a better result than 5 mins. Let us try 15 mins now.

In [47]:
```
hero_gold_15 = []
hero_experience_15 = []
for i in range(0, 113):

    hero_gold_15.append((0,0))
    hero_experience_15.append((0,0))

for i in range(0,len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_15[df_player_time_15["match_id"] == df_match
.iloc[i,0]]
    if len(df_condition) == 0:
        continue

    for j in range(0,5):
        if df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_15 = df_condition.iloc[0,2 + j*3]
            xp_15 = df_condition.iloc[0,4 + j*3]
            (total, count) = hero_gold_15[hero_id]
            hero_gold_15[hero_id] = (total + gold_15, count + 1)
            (total, count) = hero_experience_15[hero_id]
            hero_experience_15[hero_id]  = (total + xp_15, count + 1)

    for j in range(5,10):
        if not df_match.iloc[i,2]:
            hero_id = df_hero.iloc[j,1]
            gold_15 = df_condition.iloc[0,2 + j*3]
            xp_15 = df_condition.iloc[0,4 + j*3]
            (total, count) = hero_gold_15[hero_id]
            hero_gold_15[hero_id] = (total + gold_15, count + 1)
            (total, count) = hero_experience_15[hero_id]
            hero_experience_15[hero_id]  = (total + xp_15, count + 1)

gold_15_stat = []
xp_15_stat = []

for i in range(0, 113):
    if hero_gold_15[i][1] != 0:
        mean = hero_gold_15[i][0] / hero_gold_15[i][1]
        gold_15_stat.append(mean)

        mean = hero_experience_15[i][0] / hero_experience_15[i][1]
        xp_15_stat.append(mean)

    else:
        empty_tuple = ()
        gold_15_stat.append(empty_tuple)
        xp_15_stat.append(empty_tuple)

hero_gold_15 = None
hero_experience_15 = None
```

In [48]:
```
df_score = df_match[["match_id","radiant_win","radiant_score","dire_score","ra
diant_synergy","dire_synergy","radiant_counter"]]
df_gold_diff = df_gold_15[["match_id", "Radiant_gold_t", "Dire_gold_t"]]
df_xp_diff = df_xp_15[["match_id", "Radiant_xp_t", "Dire_xp_t"]]
df_new_15 = pd.merge(df_score, df_gold_diff, on="match_id")
df_new_15 = pd.merge(df_new_15, df_xp_diff, on="match_id")
df_score = None
df_gold_diff = None
df_xp_diff = None

radiant_gold_list = []
radiant_xp_list = []
dire_gold_list = []
dire_xp_list = []

for i in range(0, len(df_match)):
    df_hero = df_player[i * 10: (i + 1) * 10]
    df_condition = df_player_time_15[df_player_time_15["match_id"] == df_match
.iloc[i,0]]
    if len(df_condition) == 0:
        continue

    radiant_gold = 0
    radiant_xp = 0

    dire_gold = 0
    dire_xp = 0

    for j in range(0, 5):
        hero_id = df_hero.iloc[j,1]
        gold_15 = df_condition.iloc[0, 2 + j*3]
        xp_15 = df_condition.iloc[0, 4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_15, xp_15, gold_15_stat, xp
_15_stat)
        radiant_gold = radiant_gold + p_gold
        radiant_xp = radiant_xp + p_xp

    for j in range(5, 10):
        hero_id = df_hero.iloc[j,1]
        gold_15 = df_condition.iloc[0, 2 + j*3]
        xp_15 = df_condition.iloc[0, 4 + j*3]
        (p_gold, p_xp) = skill_level(hero_id, gold_15, xp_15, gold_15_stat, xp
_15_stat)
        dire_gold = dire_gold + p_gold
        dire_xp = dire_xp + p_xp

    radiant_gold_list.append(radiant_gold)
    radiant_xp_list.append(radiant_xp)
    dire_gold_list.append(dire_gold)
    dire_xp_list.append(dire_xp)


df_new_15.insert(4, "radiant_gold_level", radiant_gold_list, True)
df_new_15.insert(5, "radiant_xp_level", radiant_xp_list, True)
df_new_15.insert(6, "dire_gold_level", dire_gold_list, True)
df_new_15.insert(7, "dire_xp_level", dire_xp_list, True)
```

```
df_train = df_new_15[:num_train]
df_test = df_new_15[num_train:]

train_X = df_train[["radiant_score","dire_score","radiant_gold_level","radiant
_xp_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy",
"radiant_counter"]].as_matrix()
train_y = df_train["radiant_win"].values

test_X = df_test[["radiant_score","dire_score","radiant_gold_level","radiant_x
p_level","dire_gold_level","dire_xp_level","radiant_synergy","dire_synergy","r
adiant_counter"]].as_matrix()
test_y = df_test["radiant_win"].values
```

In [49]:
```
F15_clf = train_SVM(train_X, train_y)
```

In [50]:
```
predicted_train_y = F15_clf.predict(train_X)
predicted_test_y = F15_clf.predict(test_X)
print(classification_report(train_y,predicted_train_y))
print(classification_report(test_y,predicted_test_y))
```

```
              precision    recall  f1-score   support

       False       0.77      0.76      0.76     16288
        True       0.78      0.79      0.78     17488

    accuracy                           0.77     33776
   macro avg       0.77      0.77      0.77     33776
weighted avg       0.77      0.77      0.77     33776

              precision    recall  f1-score   support

       False       0.77      0.76      0.77      4009
        True       0.79      0.79      0.79      4433

    accuracy                           0.78      8442
   macro avg       0.78      0.78      0.78      8442
weighted avg       0.78      0.78      0.78      8442
```

Nice work! This is all our analysis. We have successfully build a model that can predict winner at 5 mins, 10 mins, 15 mins at over 70% accuracy.

Ethics and Privacy

This dataset is being distributed only for research purposes and easy-to-use resources, under Creative Commons Attribution-ShareAlike 4.0 International (CC BY-SA 4.0). When downloading this data we agreed to give appropriate credit with the provided link to the license and indicate if changes were made. We also agree to distribute our contributions under the same license as the original if we decide to remix, transform, or build upon the data. In this case, we will attribute the original creator of the dataset, Yasp.co for using the dataset in our project. Yasp is now known as Opendota here are links to their website and GitHub page:

https://www.opendota.com/ (https://www.opendota.com/) the data is used to for this site and it is an easy way to get familiar with it https://github.com/odota/core (https://github.com/odota/core) check here for info especially this wiki page which gives details on the schema.

Furthermore, the dataset may include relevant player statistics in the game, but it does not include very specific personal information of people who contribute to the dataset. And from the original dataset (Opendota or Yasp) people can also opt-out from contributing to the dataset if they decide to unchecked “Expose Public Match Data” option in their Dota 2 options (this is the default). Thus, the dataset is very secure. The dataset contains 50000 ranked ladder matches from the Dota 2 data dump created by Opendota or Yasp. Therefore, we expect that it does not exclude specific groups. We will also list out additional citations of the dataset at the end under the Reference tab. Because the dataset is originally coming from an open-source dataset (Opendota or Yasp), the bias level of the dataset will be very low; however, we will consider any bias that appears in our analysis.

The result of the project has no harm or benefit to the Dota2 community and other related organizations. There is no potential misuse of our project result.

Conclusion and Discussion

We used 50000 Dota2 ranked match data to build a machine learning model to predict the outcome of a match at 5 mins, 10 mins, 15 mins and 20 mins with over 70% accuracy.

In order to build the model, first, we considered the gold/experience difference at current game time. It turns out that if a team is leading gold/experience at 5 mins, they have about 60% chance to win the game. This is certainly not enough. Therefore, we added hero selection to our model. We calculated hero strength, hero synargy and hero counter and put all data into a machine learning model. It turns out to successfully predict 68% matches for 5 mins. Lastly, we considered the skill level of a player according to the performence and calculate the team skill level. The final model can predict correctly 75% matches at 5 mins, 76% at 10 mins and 77% at 15 mins.

So far, we have successfully developed profiles to determine the strength of each hero, hero synargy and hero counter. We also developed profile to determine how well a player is at certain game time. We implement these profile and build a machine learning model. Our machine learning model could predict the outcome of matches with over 70% accuracy. The limitation for our project is that our project is only valid for the patch of the game when the dataset was created. If Dota2 updated its patch, which it is very often, then we need to use new dataset to do analysis again. Another limitation is that we cannot access the history of a player. If we can, we could improve the algorithm for calculating skill level, which will greatly help the prediction success rate.

This project has impacts on gaming society. For Dota2 players, this project could help them determine the chance to win the game when they are playing. Also, this approach could also help analysis other video game as well.