Gliwice, 26.12.16

League of Legends - Predict

Nowoczesne Języki Programowania

Projekt końcowy

Piotr Jendrzyca
Informatyka, MS
Sem. VII 2016/2017

Spis treści:

1. Założenia projektu

Projekt to aplikacja webowa oparta na popularnym frameworku Django napisanym w Pythonie, która ma za zadanie przewidywać wynik trwającego meczu w grze League of Legends za pomocą naiwnego klasyfikatora Bayesowskiego na podstawie wcześniej zebranych danych. Dane zostały pobrane wykorzystując ogólnodostępne API do danych z gry jakie udostępnia producent gry - Riot Games.

2. User stories

1) Jako użytkownik, chcę mieć możliwość podania nazwy przywoływacza oraz serwera, żeby sprawdzić jakie szanse ma dany zespół na wygraną na podstawie informacji ile gier przywoływacz zagrał daną postacią, stosunku wygranych gier na tej postaci oraz drużyny w jakiej grał (niebieskiej lub czerwonej)
2) Jako użytkownik, chcę mieć podgląd na kluczowe statystyki w bieżącym meczu, żeby móc je odnieść do przewidywanego wyniku
3) Jako użytkownik, chcę być szczegółowo informowany o błędach, żeby móc ich uniknąć w przyszłości

3. Naiwny klasyfikator Bayesa

W projekcie wykorzystany został naiwny klasyfikator Bayesowski, bazujący na twierdzeniu Bayesa ponieważ nadaje się on szczególnie dobrze do problemów z wieloma wymiarami na wejściu. Mimo prostoty tej metody, często daje ona efekt lepszy niż inne, bardziej złożone metody klasyfikujące.

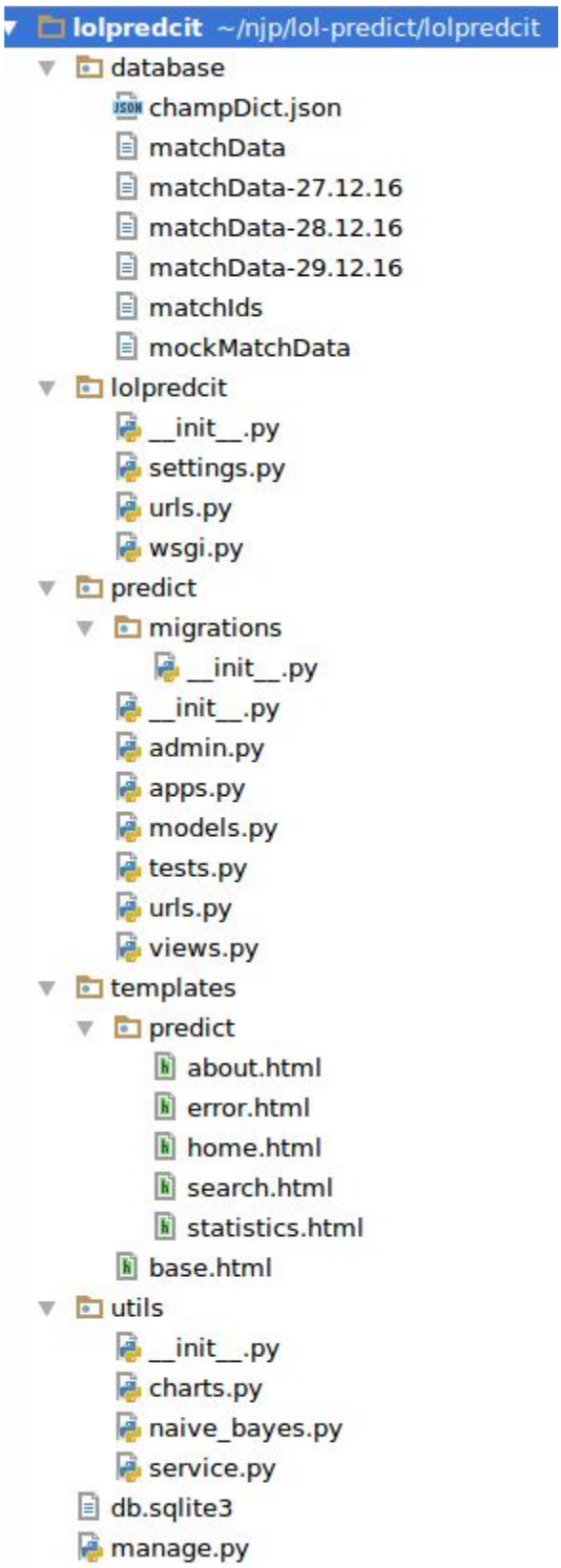
Naiwny klasyfikator to taki klasyfikator Bayesa, dla którego zakłada się, że cechy X1,X2,X3...Xn są wzajemnie niezależne. Taki klasyfikator wskazuje na daną klasę na podstawie zaobserwowanego wektora cech x. Algorytm w zagadnieniach dotyczących sztucznej inteligencji jest wykorzystywany do tzw. nadzorowanego uczenia się.

W aplikacji wykorzystana została implementacja Naiwnego klasyfikatora Bayesa z pakietu scikit-learn (http://scikit-learn.org/) oparta na rozkładzie naturalnym Gaussa. Głównym kryterium wyboru właśnie tego algorytmu była jedna z cech rozkładu naturalnego - jeśli jakaś wielkość jest sumą lub średnią bardzo wielu drobnych losowych czynników, to niezależnie od rozkładu każdego z tych czynników jej rozkład będzie zbliżony do normalnego.

Możemy wyodrębnić kolejne kroki jakie zostały przedsięwzięte w celu "wytrenowania klasyfikatora":

1) Wczytanie danych z pliku CSV
2) Podzielenie danych na dwa zbiory - treningowy i testowy. Zbiór treningowy został wykorzystane do "uczenia" klasyfikatora, testowy z kolei do weryfikacji jak dobrze algorytm radzi sobie z klasyfikacją.
3) Wywołanie funkcji z pakietu scikit-learn **fit**(X, y[, sample_weight]), gdzie X to tablica wartości treningowej zawierająca wektory cech x1,x2..xn, a y to treningowa tablica klas (0 - zwycięstwo drużyny czerwonej, 1 - zwycięstwo drużyny niebieskiej)
4) Sprawdzenie dokładności klasyfikacji przez wywołanie funkcji z pakietu scikit-learn **score**(X, y[, sample_weight]) gdzie X to tablica wartości testowej zawierająca wektory cech x1,x2..xn, a y to testowa tablica klas (0 - zwycięstwo drużyny czerwonej, 1 - zwycięstwo drużyny niebieskiej)

4. Struktura projektu



Aplikacja podzielona jest na kilka modułów. Głównym modułem jest ‘lolpredict’, który zawiera ustawienia i drogi przekierowywania adresów. Moduł ‘predict’ z kolei jest odpowiedzialny za renderowanie wszystkich stron projektu. Korzysta on z klas i metod zapisanych w pomocniczym module ‘utils’ - odpowiedzialnym za tworzenie naiwnego klasyfikatora Bayesowskiego oraz wykonywanie zapytań do udostępnianego przez Riot API gry League of Legends.

5. Opis zaimplementowanych metod i klas

1) Klasa Player - reprezentująca gracza

```
def __init__(self, summoner_name=None, champion_id=None,
team=None, winrate=0, total_games=0, avatar_url=None)
```

Metoda inicjalizująca instancje klasy Player. Tworzy ona gracza na podstawie podanych parametrów:

- summoner_name - nazwa przywoływacza; nickname gracza używany w grze
- champion_id - ID bohatera jakim gra przywoływacz
- team - ID teamu w jakim się znajduje gracz (100 - drużyna niebieska, 200 - drużyna czerwona)
- winrate - stosunek zwycięztw do rozegranych meczów danym bohaterem
- total_games - liczba wszystkich gier danym bohaterem
- avatar_url - adres url miniaturki rysunku bohatera

2) Klasa ServiceException - klasa ma za zadanie wyłapywanie błedów podczas zapytań do API League of Legends oraz wyświetlanie informacji czym dany błąd był spowodowany

```
def __init__(self, error_code):
```

Metoda inicjalizująca instancję klasy, przekazywany jest numer błędu

```
def __str__(self):
```

Metoda przedstawia obiekt w postaci ciągu znaków z odpowiednim komunikatem ze słownika `_errors`

3) Klasa RiotService - klasa, która dostarcza zbiór metod niezbędnych do komunikacji z API League of Legends

```
def __init__(self, region):
```

Metoda inicjalizująca instancję klasy, należy podać region z jakiego chcemy wyszukiwać interesujące nas dane.

```
def get_current_game(self, summoner_id):
```

[Zapytanie API] Metoda zwracająca dane aktualnej gry dla podanego ID przywoływacza w postaci obiektu JSON. Wyrzuca wyjątek 404 Game Not Found jeśli gracz o podanym ID nie jest obecnie w grze.

```
def create_player_list(self, current_game):
```

Metoda tworząca listę graczy typu Players. Zainicjalizowane pola to summoner_id, champion_id oraz team. Jako argument należy podać obiekt JSON bieżącego meczu.

```
def get_summoner_id(self, ummoner_name):
```

[Zapytanie API] Metoda zwraca ID przywoływacza o podanym nicku

```
def get_champion_winrate(self, summoner_id, champion_id):
```

[Zapytanie API] Metoda zwraca stosunek zwycięstw do rozegranych meczów oraz ilość wszystkich rozegranych meczy danym bohaterem.

```
def get_match_by_id(self, match_id):
```

[Zapytanie API] Metoda zwraca informacje dotyczące podanego w argumencie meczu w postaci obiektu JSON.

```
def get_summs_and_champs_from_match(self, match_id=None,
match=None):
```

Metoda zwraca listę ID graczy oraz ID bohaterów z podanego w argumencie meczu. Jeśli zamiast meczu przekazane zostało ID wtedy wywoływana jest dodatkowo metoda get_match_by_id

def create_match_database(self, summoner_name, count=1000)

[Zapytanie API] Tworzy listę unikalnym ID meczów przeszukując historię meczów podanego w argumencie gracza. Następnie z listy uzyskanych meczy pobiera szczegółowy informacje o jednym, tworzy listę graczy których to historia meczów jest kolejno pobierana. Kroki zostaną powtarzane, aż do uzyskania odpowiedniej ilości meczy podanej w argumencie, domyślnie jest to tysiąc.

def get_data_from_current_match(self, current_match):

Metoda przyjmuje dane bieżącego meczu jako argument zwraca słownik składający się z tablicy danych gotowych do przekazania dla klasyfikatora Bayesowskiego oraz listę obiektów klasy Player z uzupełnionymi wartościami avatr_url, total_games, winrate, champion_id oraz summoner_name.

def get_data_from_match(self, match):

Metoda przyjmuje dane historycznego meczu jako argument i zwraca tablicę danych gotową do przekazania dla klasyfikatora Bayesowskiego

def create_champ_database(self):

[Zapytanie API] Metoda tworzy lokalnie baze danych bohaterów zawierającą słownik z ID bohatera oraz pełną nazwą bohatera i adresem url do miniaturki rysunku bohatera.

def create_stats_database(self, ids_path, out_path):

Metoda tworzy lokalną bazę danych w postaci pliku CSV. Jako argument należy podać plik zawierający ID meczów, z których chcemy stworzyć bazę jak również nazwę pliku wyjściowego. Baza jest tworzona w postaci: Champion_id, winrate, total_games, team x10 dla każdego gracza, na końcu podawany jest wynik meczu 0 - oznacza wygraną niebieskiej drużyn, 1 - czerwonej.

*Metody oznaczone jako [Zapytanie API] mogą wyrzucać wyjątek typu ServiceException.

4) Plik naive_bayes

```
def _read_lines(trainingset_path):
```

Metoda wczytująca dane z pliku CSV zawierającego dane z poszczególnych meczów. Zwraca tablicę dwuwymiarową składającą się z liczb zmiennoprzecinkowych reprezentujących dane z meczu. Każdy mecz to jeden wiersz macierzy

```
def _handle_data(data, path=os.getcwd() + '/database/matchData',
split_ratio=.7):
```

Metoda wczytująca dane z pliku CSV. Parametr data determinuje jakie konkretnie dane mają być wczytane

```
def _split_to_test_and_training_sets(X, Y, ratio):
```

Metoda dzieli zbiór cech X i zbiór klas Y na zbiory testowe i zbiory treningowe, według przekazanego współczynnika ratio (0,1)

5) Klasa Data, dziedzicząca po Enum używana do inicjalizacji klasy NBClassifier - wskazuje jaki zestaw danych ma przyjąć klasyfikator do "uczenia się". Możliwe opcje to:

- Default - Id bohatera, współczynnik zwycięstw, ilość gier, id drużyny dla każdego z 10 graczy
- Summarized - zsumowane współczynniki zwycięstw obu drużyn (np. [2.2, 1.5])
- Multiplied - suma wymnożonych przez siebie współczynników zwycięstw oraz ilości gier

6) Klasa NBClassifier - inicjalizacja klasyfikatora i trenowanie klasyfikatora Bayesa

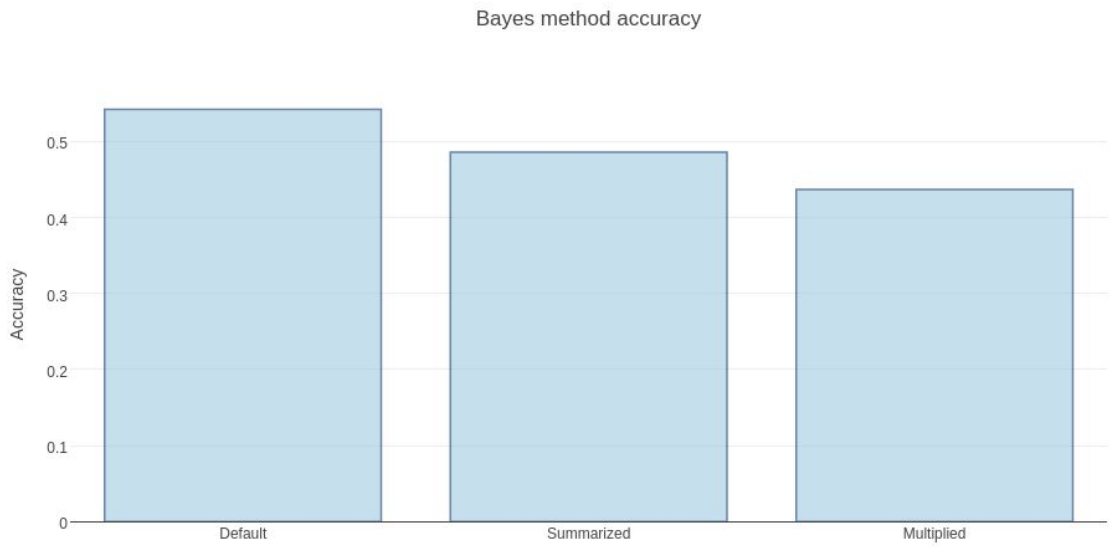
7) Plik charts.py - zawiera metody odpowiedzialne za rysowanie wykresów i przekazywanie ich do widoków. Dostępne metody:

- `def accuracy_chart()` - rysuje wykres dokładności klasyfikatorów (0,1)
- `def split_chart()` - rysuje wykres obrazujący w jaki sposób dane zostały podzielone na zbiór treningowy i testowy

6. Analiza uzyskanych wyników

Udało się pobrać do analizy około 500 meczy (taka ilość ze względu na ograniczenia zapytań do API od strony producenta). Dane zostały podzielone tak jak pokazane na rysunku

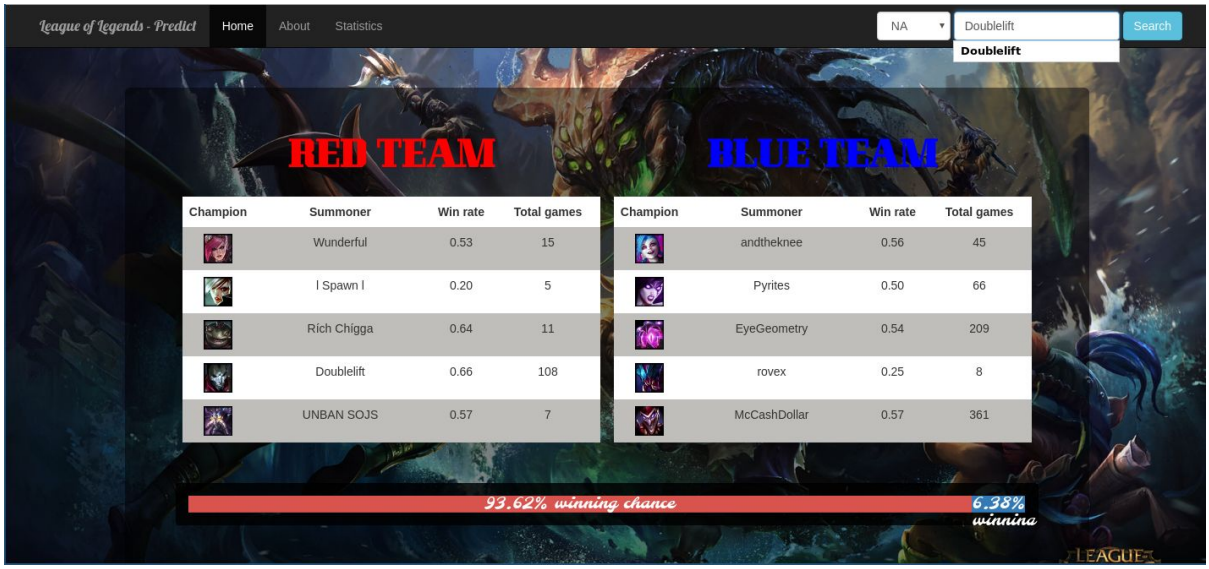
Po wytrenowaniu klasyfikatorów otrzymane zostały następujące wyniki:



Można zauważyć że system działa najlepiej dla danych typu DEFAULT, lecz i tak nie jest to zadowalający wynik. Przyczyną takiego stanu rzeczy jest najprawdopodobniej liczba gier - 500 to zdecydowanie za mało - oraz fakt iż dopiero rozpoczął się nowy sezon w grze League of Legends i często otrzymywane wyniki dla współczynnika zwycięstw oraz ilości gier były równe 0.

7. Działanie aplikacji

Gdy szukany przywoływacz jest w grze:



Gdy szukany przywoływacz aktualnie nie gra lub gdy wprowadzony nick jest nieprawidłowy: