Przewidywanie zwycięzcy rundy w grze Counter-Strike: Global Offensive

Autorzy: Piotr Kryczka, Jakub Michalak, Damian Opoka, Adam Ryl

Counter-Strike: Global Offensive

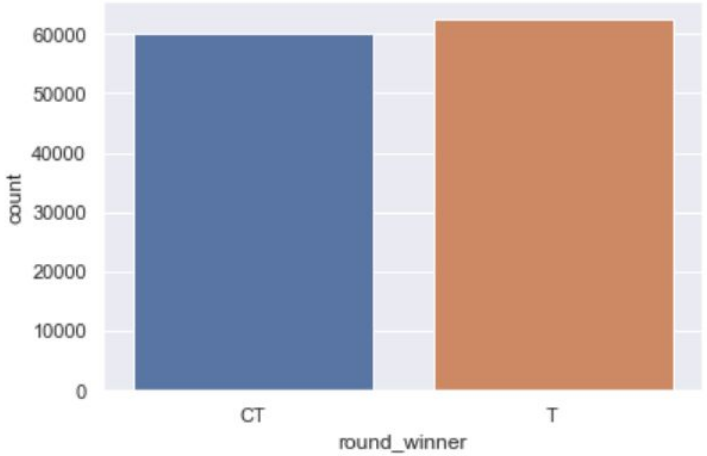
- Wieloosobowa gra typu FPS
- 30 rund trwających po 1 min 55 s
- Drużyny terrorystów i antyterrorystów
- Dane: https://www.kaggle.com/datasets/christianlillelund/csgo-round-winner-classification
- Przewidywanie zwycięzcy rundy

CS:GO - kluczowe elementy rozgrywki

- broń podstawowa - pistolety i karabiny
- broń wspomagająca - granaty
- dodatki - kamizelka, hełm, defuse kit
- zbudowana ekonomia
- żywi gracze, punkty życia
- mapa
- czas pozostały do końca rundy
- podłożenie bomby

Wykorzystane dane

Zbiór danych składa się ze snapshotów rund z około 700 meczów z profesjonalnych turniejów rozgrywanych w 2019 i 2020 roku. Snapshoty, czyli zestawienie pewnych stanów kluczowych elementów rozgrywki, były rejestrowane podczas gry co 20 sekund aż do rozstrzygnięcia danej rundy. Łączna liczba zapisanych snapshotów wynosi **122410**, a każdy rekord zawiera **97** atrybutów.



Przykładowe atrybuty

Zmienna	Definicja	Przykład
time_left	czas pozostały do zakończenia rundy	
ct_score	bieżący wynik drużyny CT	
t_score	bieżący wynik drużyny T	
map	mapa, na której toczy się rozgrywka	np. de_dust2, de_inferno and de_overpass
bomb_planted	czy bomba została podłożona	False = nie, True = tak
ct_health	suma punktów życia drużyny CT	zakres 0-500
t_health	suma punktów życia drużyny T	zakres 0-500
ct_armor	suma punktów pancerza drużyny CT	
t_armor	suma punktów pancerza drużyny T	
ct_money	suma pieniędzy drużyny CT	wartość w USD
t_money	suma pieniędzy drużyny T	wartość w USD
ct_helmets	liczba graczy z hełmem w drużynie CT	
t_helmets	liczba graczy z hełmem w drużynie CT	
ct_defuse_kits	liczba graczy z zestawem do rozbrajania	
ct_players_alive	Liczba żywych graczy w drużynie CT	zakres 0-5
t_players_alive	Liczba żywych graczy w drużynie CT	zakres 0-5
ct_weapon_X	liczba broni X w drużynie CT	np. Ak47, Deagle i UMP45.
t_weapon_X	liczba broni X w drużynie T	np. Ak47, Deagle i UMP45.
ct_grenade_X	liczba granatów X w drużynie CT	np. HeGrenade, Flashbang.
t_grenade_X	liczba granatów X w drużynie T	np. HeGrenade, Flashbang.
round_winner	Zwycięzca rundy	CT = antyterrorysta, T = terrorysta

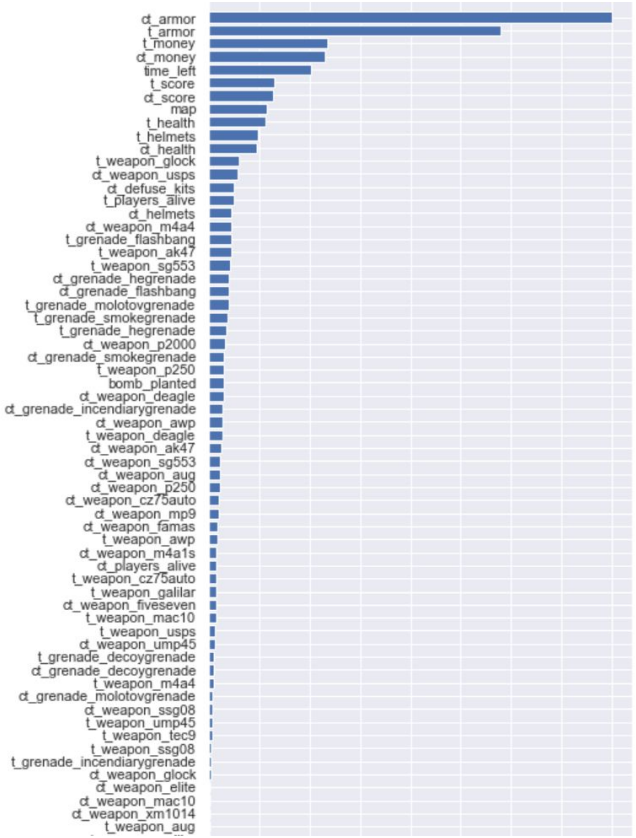
Przetworzenie danych

1. Konwersja wartości tekstowych na liczbowe.
2. Normalizacja danych.
3. Wydzielenie atrybutów decyzyjnych.
4. Wybór atrybutów o największym znaczeniu.
5. Podział danych na treningowe i testowe.

Wybór atrybutów

Wykorzystany został model *Random Forest Regressor*, który dopasowuje szereg klasyfikujących drzew decyzyjnych do różnych podpróbek zbioru danych i wykorzystuje uśrednianie w celu poprawy dokładności predykcyjnej i kontroli nadmiernego dopasowania. Po tej operacji wybrane zostały atrybuty, dla których wartość parametru *feature_importance* była większa od wartości 0,005.

Po tym etapie pozostało 38 atrybutów.



Model MLP

Wykorzystujemy klasyfikator wielowarstwowy MLP ze *scikit-learn* (*MLPClassifier*).

W przeciwieństwie do innych algorytmów klasyfikacji *MLPClassifier* przy wykonaniu zadania klasyfikacji opiera się na podstawowej sieci neuronowej.

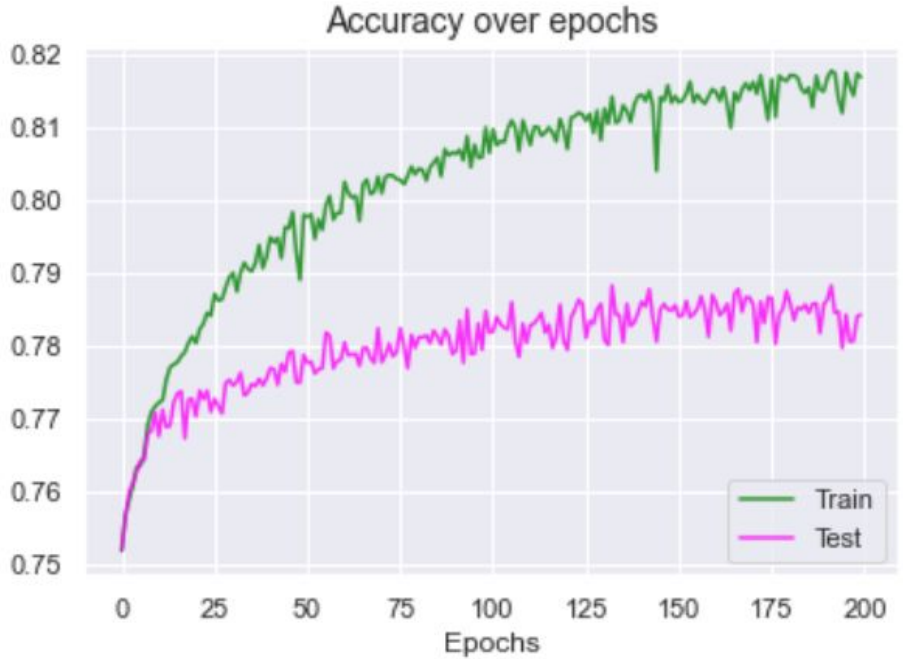
```
nn_clf = MLPClassifier(verbose = True, warm_start=True, max_iter=1)
```

Model MLP - dopasowanie

Następnie przechodzimy do dopasowania modelu do macierzy danych *x_train* i wyniku *y_train* dla dwustu epok, obliczając i zapamiętując wyniki dla zbioru treningowego i testowego.

```
scores_mlp_train = []
scores_mlp_test = []
for i in range(200):
    nn_clf.fit(x_train, y_train)
    scores_mlp_train.append(nn_clf.score(x_train, y_train))
    scores_mlp_test.append(nn_clf.score(x_test, y_test))
    print(f'Iteration {i + 1}, score = {nn_clf.score(x_test, y_test)}')
```

Model MLP - wyniki



Model sieci neuronowej: 0.7842496528061433

Model Regresji Logistycznej

Wykorzystujemy klasyfikator Regresji Logistycznej ze *scikit-learn* (*LogisticRegression*).

Metoda statystyczna pozwalająca na opisanie współzmienności kilku zmiennych przez dopasowanie do nich funkcji.

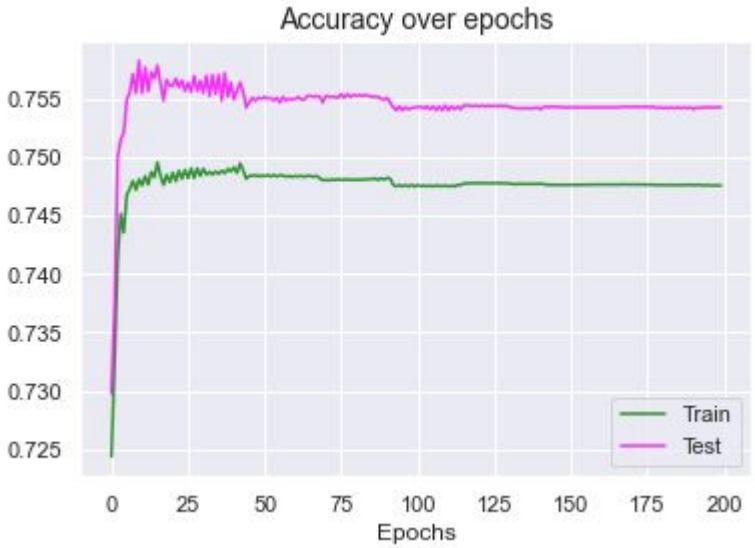
```
log_model = LogisticRegression(verbose=True, n_jobs=-1, warm_start=True, max_iter=1)
```

Model Regresji Logistycznej - dopasowanie

Następnie przechodzimy do dopasowania modelu do macierzy danych *x_train* i wyniku *y_train* dla dwustu epok, obliczając i zapamiętując wyniki dla zbioru treningowego i testowego.

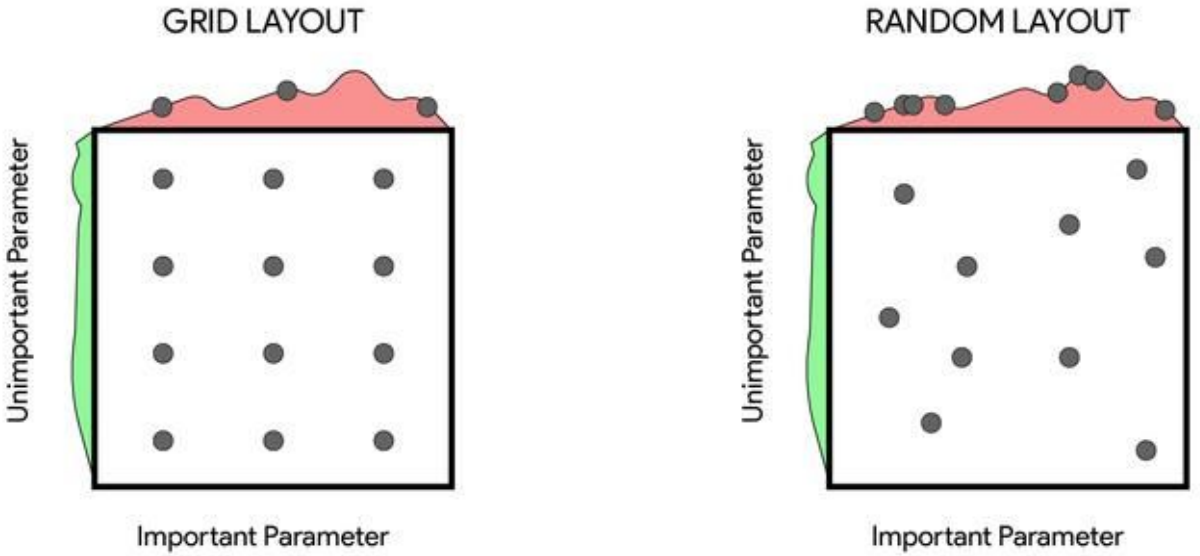
```
scores_log_train = []
scores_log_test = []
for i in range(200):
    log_model.fit(x_train, y_train)
    scores_log_train.append(log_model.score(x_train, y_train))
    scores_log_test.append(log_model.score(x_test, y_test))
    print(f'Iteration {i + 1}, score = {log_model.score(x_test, y_test)}')
```

Model Regresji Logistycznej - wyniki




```
Model regresji logistycznej: 0.7542684421207417
```

Optymalizacja hiperparametrów



Optuna

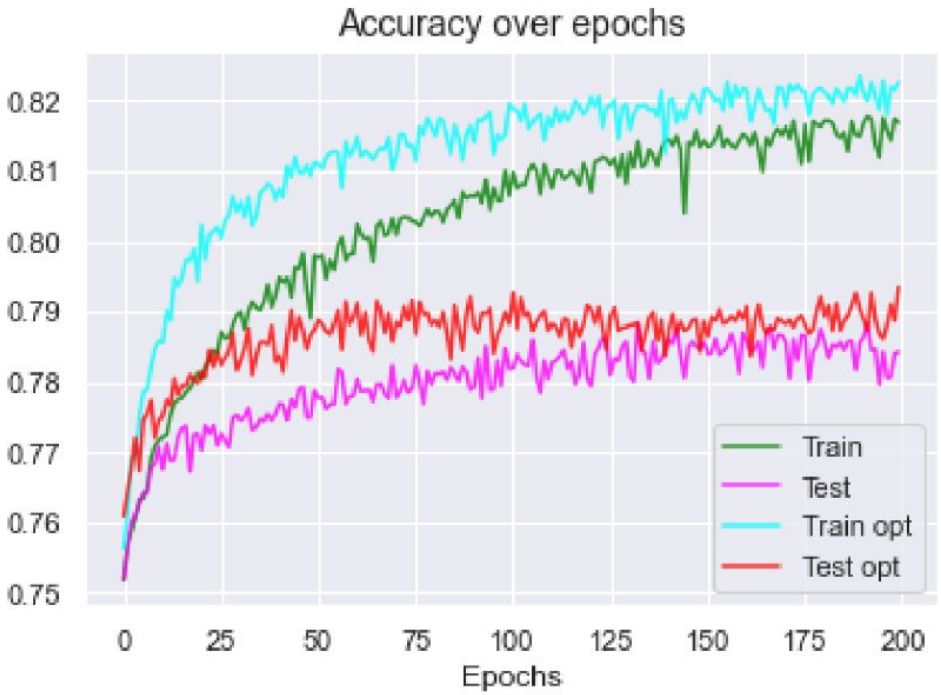
Optuna wykorzystuje historyczny zapis konfiguracji do określenia obiecującego obszaru do przeszukiwania w celu optymalizacji hiperparametrów i w ten sposób znajduje optymalny hiperparametr w rozsądnym czasie przy użyciu algorytmu T*ree-structured Parzen estimators*. Niezbyt obiecujące konfiguracje nie są chętnie eksplorowane. Zalety *Optuny* to m.in.:

- lekka, uniwersalna i niezależna od platformy architektura,
- wydajne algorytmy optymalizacyjne,
- łatwa wielowątkowość,
- ładna wizualizacja.

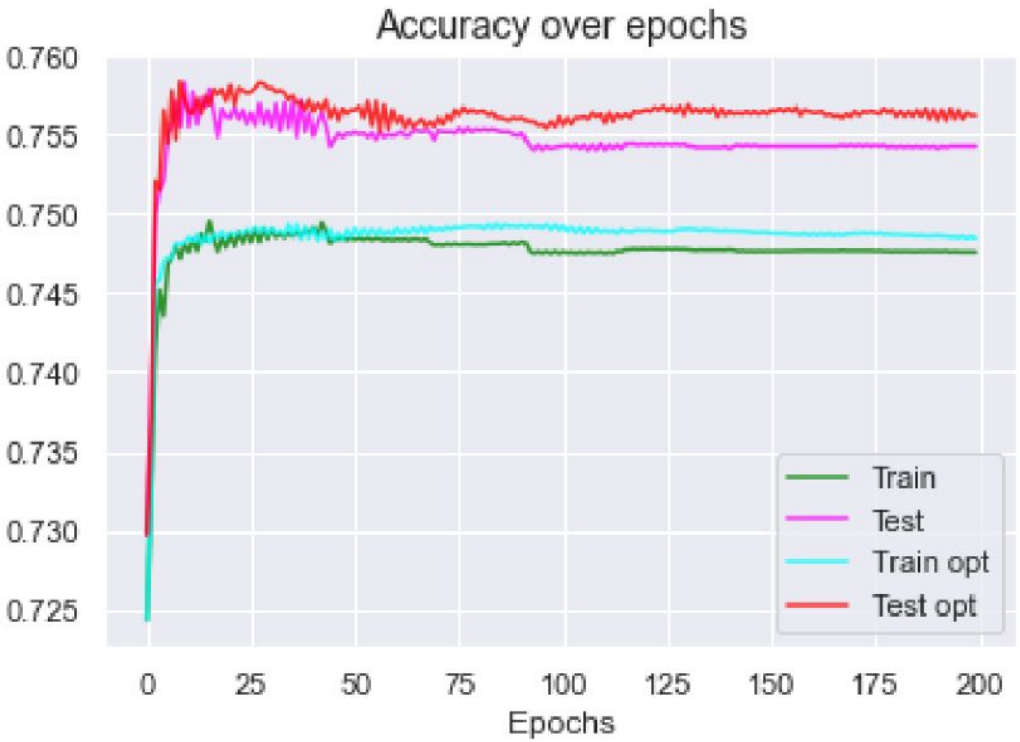
Przykład

```
import optuna
import sklearn.datasets

import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm

def objective(trial):
    activation = trial.suggest_categorical("activation", ["identity",␣
 ↪"logistic", "tanh", "relu"])
    solver = trial.suggest_categorical("solver", ["lbfgs", "sgd", "adam"])
    learning_rate_init = trial.suggest_float("lri", 1e-5, 1e-2, log=True)
    learning_rate = "constant"
    if solver == "sgd":
        learning_rate = trial.suggest_categorical("learning_rate", ["constant",␣
 ↪"invscaling", "adaptive"])
    nn_clf = MLPClassifier(activation=activation, solver=solver,␣
 ↪warm_start=True, max_iter=1,
                           learning_rate=learning_rate,␣
 ↪learning_rate_init=learning_rate_init)
    for i in range(200):
        nn_clf.fit(x_train, y_train)
    return nn_clf.score(x_test, y_test)


study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

Optymalizacja MLP



Optymalizacja RL



Dziękujemy za uwagę :)