

# Raport

June 4, 2022

## 1 Przewidywanie zwycięzcy rundy w grze Counter Strike: Global Offensive

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 122411. Część tych rekordów będzie traktowana jako zbiór danych uczących, a pozostała część jako część danych testów. Każdy rekord traktowany jest jako pojedynczy, niezależny element do analizy danych.

### 1.1 Czym jest klasyfikator MLP?

Perceptron wielowarstwowy (MLP) to model sztucznej sieci neuronowej ze sprzężeniem do przodu, który odwzorowuje zestawy danych wejściowych na zestaw odpowiednich danych wyjściowych.

MLP składa się z wielu warstw, a każda warstwa jest w pełni połączona z następną. Węzły warstw to neurony z nieliniowymi funkcjami aktywacji, z wyjątkiem węzłów warstwy wejściowej. Pomiedzy warstwą wejściową a wyjściową może znajdować się jedna lub więcej nieliniowych warstw ukrytych.

### 1.2 Import bibliotek i danych

```
[1]: import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot as plt
import seaborn as sns

[2]: df = pd.read_csv('Data/csgo_round_snapshots.csv')

[3]: pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

```
[4]: df.head()
```

```
[4]:   time_left  ct_score  t_score      map  bomb_planted  ct_health  t_health  \
0    175.00      0.0      0.0  de_dust2      False      500.0      500.0
1    156.03      0.0      0.0  de_dust2      False      500.0      500.0
2     96.03      0.0      0.0  de_dust2      False      391.0      400.0
3     76.03      0.0      0.0  de_dust2      False      391.0      400.0
4    174.97      1.0      0.0  de_dust2      False      500.0      500.0

      ct_armor  t_armor  ct_money  t_money  ct_helmets  t_helmets  \
0         0.0      0.0    4000.0   4000.0         0.0         0.0
1        400.0     300.0     600.0    650.0         0.0         0.0
2        294.0     200.0     750.0    500.0         0.0         0.0
3        294.0     200.0     750.0    500.0         0.0         0.0
4        192.0       0.0   18350.0  10750.0         0.0         0.0

      ct_defuse_kits  ct_players_alive  t_players_alive  ct_weapon_ak47  \
0                0.0                5.0                5.0          0.0
1                1.0                5.0                5.0          0.0
2                1.0                4.0                4.0          0.0
3                1.0                4.0                4.0          0.0
4                1.0                5.0                5.0          0.0

      t_weapon_ak47  ct_weapon_aug  t_weapon_aug  ct_weapon_awp  t_weapon_awp  \
0                0.0            0.0            0.0            0.0            0.0
1                0.0            0.0            0.0            0.0            0.0
2                0.0            0.0            0.0            0.0            0.0
3                0.0            0.0            0.0            0.0            0.0
4                0.0            0.0            0.0            0.0            0.0

      ct_weapon_bizon  t_weapon_bizon  ct_weapon_cz75auto  t_weapon_cz75auto  \
0                0.0            0.0            0.0            0.0
1                0.0            0.0            0.0            0.0
2                0.0            0.0            0.0            0.0
3                0.0            0.0            0.0            0.0
4                0.0            0.0            0.0            0.0

      ct_weapon_elite  t_weapon_elite  ct_weapon_famas  t_weapon_famas  \
0                0.0            0.0            0.0            0.0
1                0.0            0.0            0.0            0.0
2                0.0            0.0            0.0            0.0
3                0.0            0.0            0.0            0.0
4                0.0            0.0            0.0            0.0

      ct_weapon_g3sg1  t_weapon_g3sg1  ct_weapon_galilar  t_weapon_galilar  \
0                0.0            0.0            0.0            0.0
1                0.0            0.0            0.0            0.0
```

2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0

	ct_weapon_glock	t_weapon_glock	ct_weapon_m249	t_weapon_m249	\
0	0.0	5.0	0.0	0.0	
1	0.0	5.0	0.0	0.0	
2	0.0	4.0	0.0	0.0	
3	0.0	3.0	0.0	0.0	
4	0.0	5.0	0.0	0.0	

	ct_weapon_m4a1s	t_weapon_m4a1s	ct_weapon_m4a4	t_weapon_m4a4	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_mac10	t_weapon_mac10	ct_weapon_mag7	t_weapon_mag7	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_mp5sd	t_weapon_mp5sd	ct_weapon_mp7	t_weapon_mp7	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_mp9	t_weapon_mp9	ct_weapon_negev	t_weapon_negev	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_nova	t_weapon_nova	ct_weapon_p90	t_weapon_p90	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_r8revolver	t_weapon_r8revolver	ct_weapon_sawedoff	\
--	----------------------	---------------------	--------------------	---

0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	t_weapon_sawedoff	ct_weapon_scar20	t_weapon_scar20	ct_weapon_sg553	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_sg553	ct_weapon_ssg08	t_weapon_ssg08	ct_weapon_ump45	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_ump45	ct_weapon_xm1014	t_weapon_xm1014	ct_weapon_deagle	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_deagle	ct_weapon_fiveseven	t_weapon_fiveseven	ct_weapon_usps	\
0	0.0	0.0	0.0	4.0	
1	0.0	0.0	0.0	4.0	
2	0.0	0.0	0.0	4.0	
3	0.0	0.0	0.0	4.0	
4	0.0	0.0	0.0	4.0	

	t_weapon_usps	ct_weapon_p250	t_weapon_p250	ct_weapon_p2000	\
0	0.0	0.0	0.0	1.0	
1	0.0	0.0	0.0	1.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	

	t_weapon_p2000	ct_weapon_tec9	t_weapon_tec9	ct_grenade_hgrenade	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_grenade_hegrenade	ct_grenade_flashbang	t_grenade_flashbang	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

	ct_grenade_smokegrenade	t_grenade_smokegrenade	\
0	0.0	0.0	
1	0.0	2.0	
2	0.0	2.0	
3	0.0	0.0	
4	0.0	0.0	

	ct_grenade_incendiarygrenade	t_grenade_incendiarygrenade	\
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	

	ct_grenade_molotovgrenade	t_grenade_molotovgrenade	\
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	

	ct_grenade_decoygrenade	t_grenade_decoygrenade	round_winner
0	0.0	0.0	CT
1	0.0	0.0	CT
2	0.0	0.0	CT
3	0.0	0.0	CT
4	0.0	0.0	CT

```
[5]: df.isnull().sum().sum()
```

```
[5]: 0
```

```
[6]: df.shape
```

```
[6]: (122410, 97)
```

### 1.3 Przetwarzanie danych

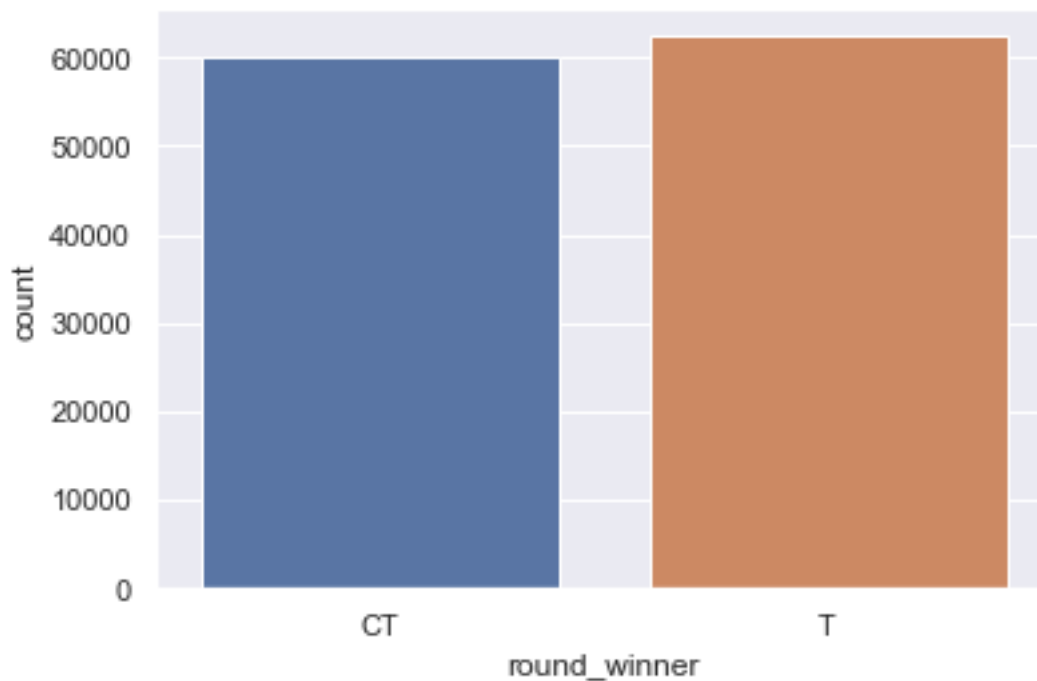
W pierwszej kolejności sprawdzimy licznosc klas.

```
[7]: df['round_winner'].value_counts()
```

```
[7]: T      62406  
     CT      60004  
     Name: round_winner, dtype: int64
```

```
[8]: sns.set_theme(style="darkgrid")  
     sns.countplot(x='round_winner', data=df)
```

```
[8]: <AxesSubplot:xlabel='round_winner', ylabel='count'>
```



Następnie przekonwertujemy wszystkie kolumny na wartości liczbowe.

```
[9]: col = df.drop(df.select_dtypes(np.number), axis = 1).columns  
     col
```

```
[9]: Index(['map', 'bomb_planted', 'round_winner'], dtype='object')
```

```
[10]: lbl = LabelEncoder()  
      for value in col:  
          df[value] = lbl.fit_transform(df[value])
```

```
[11]: df['bomb_planted'] = df['bomb_planted'].astype(np.int16)
```

```
[12]: cols = [f for f in df.columns if f not in ['round_winner']]
```

Dodatkowo każda wartość zostanie znormalizowana.

```
[13]: scaler = RobustScaler()

for value in cols:
    scaler = RobustScaler()
    df[value] = scaler.fit_transform(df[[value]])
```

Na koniec podzielimy dane na wektor danych oraz wektor wyników.

```
[14]: x = df.drop(['round_winner'], axis = 1)
      y = df['round_winner']
```

```
[15]: len(cols)
```

```
[15]: 96
```

Finalnie każdy rekord zawiera 96 atrybutów, które prezentują się w sposób przedstawiony poniżej.

```
[16]: x.head()
```

```
[16]:   time_left  ct_score  t_score      map  bomb_planted  ct_health  t_health  \
0   0.715105 -0.857143 -0.857143 -0.666667         0.0   0.000000  0.000000
1   0.545726 -0.857143 -0.857143 -0.666667         0.0   0.000000  0.000000
2   0.010000 -0.857143 -0.857143 -0.666667         0.0  -0.726667 -0.561798
3  -0.168575 -0.857143 -0.857143 -0.666667         0.0  -0.726667 -0.561798
4   0.714837 -0.714286 -0.857143 -0.666667         0.0   0.000000  0.000000

      ct_armor  t_armor  ct_money  t_money  ct_helmets  t_helmets  \
0 -1.291096 -1.136054 -0.112782 -0.191489        -0.5        -0.6
1  0.078767 -0.115646 -0.368421 -0.395137        -0.5        -0.6
2 -0.284247 -0.455782 -0.357143 -0.404255        -0.5        -0.6
3 -0.284247 -0.455782 -0.357143 -0.404255        -0.5        -0.6
4 -0.633562 -1.136054  0.966165  0.218845        -0.5        -0.6

      ct_defuse_kits  ct_players_alive  t_players_alive  ct_weapon_ak47  \
0         -0.333333                0.0                0.0            0.0
1          0.000000                0.0                0.0            0.0
2          0.000000               -1.0               -1.0            0.0
3          0.000000               -1.0               -1.0            0.0
4          0.000000                0.0                0.0            0.0

      t_weapon_ak47  ct_weapon_aug  t_weapon_aug  ct_weapon_awp  t_weapon_awp  \
0          -0.5            0.0            0.0            0.0            0.0
1          -0.5            0.0            0.0            0.0            0.0
2          -0.5            0.0            0.0            0.0            0.0
3          -0.5            0.0            0.0            0.0            0.0
4          -0.5            0.0            0.0            0.0            0.0
```

	ct_weapon_bizon	t_weapon_bizon	ct_weapon_cz75auto	t_weapon_cz75auto	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_elite	t_weapon_elite	ct_weapon_famas	t_weapon_famas	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_g3sg1	t_weapon_g3sg1	ct_weapon_galilar	t_weapon_galilar	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_glock	t_weapon_glock	ct_weapon_m249	t_weapon_m249	\
0	0.0	0.333333	0.0	0.0	
1	0.0	0.333333	0.0	0.0	
2	0.0	0.000000	0.0	0.0	
3	0.0	-0.333333	0.0	0.0	
4	0.0	0.333333	0.0	0.0	

	ct_weapon_m4a1s	t_weapon_m4a1s	ct_weapon_m4a4	t_weapon_m4a4	\
0	0.0	0.0	-0.5	0.0	
1	0.0	0.0	-0.5	0.0	
2	0.0	0.0	-0.5	0.0	
3	0.0	0.0	-0.5	0.0	
4	0.0	0.0	-0.5	0.0	

	ct_weapon_mac10	t_weapon_mac10	ct_weapon_mag7	t_weapon_mag7	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_mp5sd	t_weapon_mp5sd	ct_weapon_mp7	t_weapon_mp7	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	



3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0

	ct_weapon_mp9	t_weapon_mp9	ct_weapon_negev	t_weapon_negev	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_nova	t_weapon_nova	ct_weapon_p90	t_weapon_p90	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	ct_weapon_r8revolver	t_weapon_r8revolver	ct_weapon_sawedoff	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

	t_weapon_sawedoff	ct_weapon_scar20	t_weapon_scar20	ct_weapon_sg553	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_sg553	ct_weapon_ssg08	t_weapon_ssg08	ct_weapon_ump45	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_ump45	ct_weapon_xm1014	t_weapon_xm1014	ct_weapon_deagle	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_weapon_deagle	ct_weapon_fiveseven	t_weapon_fiveseven	ct_weapon_usps	\
0	0.0	0.0	0.0	0.333333	

1	0.0	0.0	0.0	0.333333
2	0.0	0.0	0.0	0.333333
3	0.0	0.0	0.0	0.333333
4	0.0	0.0	0.0	0.333333

	t_weapon_usps	ct_weapon_p250	t_weapon_p250	ct_weapon_p2000	\
0	0.0	0.0	0.0	1.0	
1	0.0	0.0	0.0	1.0	
2	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	

	t_weapon_p2000	ct_weapon_tec9	t_weapon_tec9	ct_grenade_hegrenade	\
0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	

	t_grenade_hegrenade	ct_grenade_flashbang	t_grenade_flashbang	\
0	0.0	-0.333333	-0.333333	
1	0.0	-0.333333	-0.333333	
2	0.0	-0.333333	-0.333333	
3	0.0	-0.333333	-0.333333	
4	0.0	-0.333333	-0.333333	

	ct_grenade_smokegrenade	t_grenade_smokegrenade	\
0	-0.333333	-0.333333	
1	-0.333333	0.333333	
2	-0.333333	0.333333	
3	-0.333333	-0.333333	
4	-0.333333	-0.333333	

	ct_grenade_incendiarygrenade	t_grenade_incendiarygrenade	\
0	0.0	0.0	
1	0.0	0.0	
2	0.0	0.0	
3	0.0	0.0	
4	0.0	0.0	

	ct_grenade_molotovgrenade	t_grenade_molotovgrenade	\
0	0.0	-0.5	
1	0.0	-0.5	
2	0.0	-0.5	
3	0.0	-0.5	
4	0.0	-0.5	

	ct_grenade_decoygrenade	t_grenade_decoygrenade
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0

### 1.3.1 Wybór atrybutów

W pierwszej kolejności sprawdzimy, które atrybuty mają największy wpływ na klasy.

W tym celu wykorzystamy 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.

```
[17]: rf = RandomForestRegressor(n_estimators=150, n_jobs=-1)
      rf.fit(x, y)
```

```
[17]: RandomForestRegressor(n_estimators=150, n_jobs=-1)
```

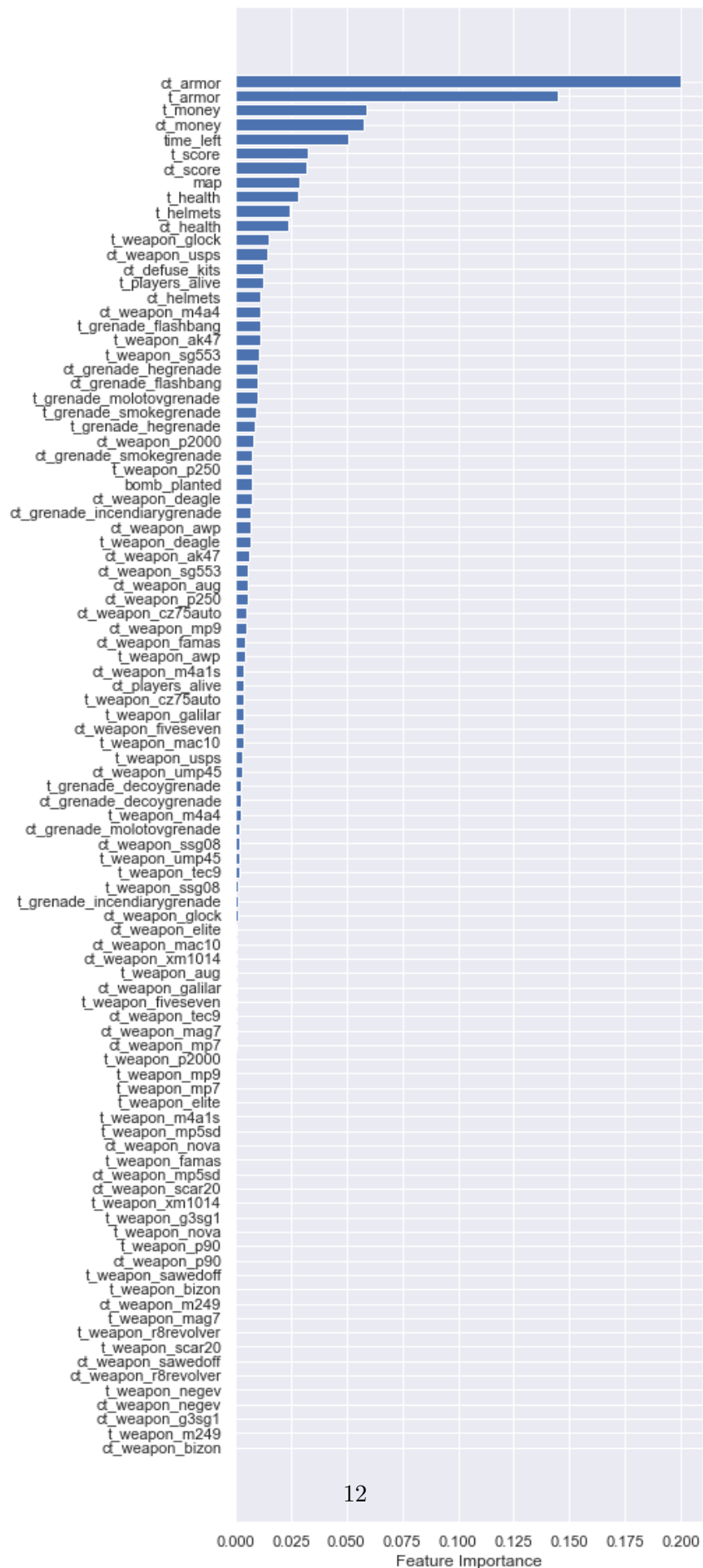
Po stworzeniu i wyuczeniu modelu sortujemy tablicę zawierającą informacje o tym, jak duży wpływ mają konkretne atrybuty w podejmowaniu decyzji.

```
[18]: indexes = rf.feature_importances_.argsort()
```

Wpływ poszczególnych atrybutów został przedstawiony na poniższym wykresie.

```
[19]: plt.figure(figsize=(6,20))
      plt.barh(x.columns[indexes], rf.feature_importances_[indexes])
      plt.xlabel("Feature Importance")
```

```
[19]: Text(0.5, 0, 'Feature Importance')
```



Następnie możemy odrzucić te atrybuty, które mają niski wpływ na podejmowanie decyzji lub nie mają żadnego. W tym celu wybraliśmy te atrybuty, dla których wartość parametru *feature\_importance* jest większa od wartości 0,005.

```
[20]: importances = rf.feature_importances_[indexes]
      indexes = indexes[np.argwhere(importances > 0.005)]
      indexes = np.concatenate(indexes).ravel().tolist()
      columns = x.columns[indexes]
      x = x[columns]
```

W tym momencie możemy sprawdzić liczbę atrybutów pozostałych po selekcji.

```
[21]: x.shape
```

```
[21]: (122410, 38)
```

Mamy 38 atrybutów, które prezentują się jak w poniższej tabeli.

```
[22]: x.head()
```

```
[22]:  ct_weapon_cz75auto  ct_weapon_p250  ct_weapon_aug  ct_weapon_sg553  \
0                0.0            0.0            0.0            0.0
1                0.0            0.0            0.0            0.0
2                0.0            0.0            0.0            0.0
3                0.0            0.0            0.0            0.0
4                0.0            0.0            0.0            0.0
```

```
  ct_weapon_ak47  t_weapon_deagle  ct_weapon_awp  \
0            0.0            0.0            0.0
1            0.0            0.0            0.0
2            0.0            0.0            0.0
3            0.0            0.0            0.0
4            0.0            0.0            0.0
```

```
  ct_grenade_incendiarygrenade  ct_weapon_deagle  bomb_planted  \
0                0.0            0.0            0.0
1                0.0            0.0            0.0
2                0.0            0.0            0.0
3                0.0            0.0            0.0
4                0.0            0.0            0.0
```

```
  t_weapon_p250  ct_grenade_smokegrenade  ct_weapon_p2000  \
0            0.0            -0.333333            1.0
1            0.0            -0.333333            1.0
2            0.0            -0.333333            0.0
3            0.0            -0.333333            0.0
```

4	0.0	-0.333333	1.0	
---	-----	-----------	-----	--

	t_grenade_hegrenade	t_grenade_smokegrenade	t_grenade_molotovgrenade	\
0	0.0	-0.333333	-0.5	
1	0.0	0.333333	-0.5	
2	0.0	0.333333	-0.5	
3	0.0	-0.333333	-0.5	
4	0.0	-0.333333	-0.5	

	ct_grenade_flashbang	ct_grenade_hegrenade	t_weapon_sg553	t_weapon_ak47	\
0	-0.333333	0.0	0.0	-0.5	
1	-0.333333	0.0	0.0	-0.5	
2	-0.333333	0.0	0.0	-0.5	
3	-0.333333	0.0	0.0	-0.5	
4	-0.333333	0.0	0.0	-0.5	

	t_grenade_flashbang	ct_weapon_m4a4	ct_helmets	t_players_alive	\
0	-0.333333	-0.5	-0.5	0.0	
1	-0.333333	-0.5	-0.5	0.0	
2	-0.333333	-0.5	-0.5	-1.0	
3	-0.333333	-0.5	-0.5	-1.0	
4	-0.333333	-0.5	-0.5	0.0	

	ct_defuse_kits	ct_weapon_usps	t_weapon_glock	ct_health	t_helmets	\
0	-0.333333	0.333333	0.333333	0.000000	-0.6	
1	0.000000	0.333333	0.333333	0.000000	-0.6	
2	0.000000	0.333333	0.000000	-0.726667	-0.6	
3	0.000000	0.333333	-0.333333	-0.726667	-0.6	
4	0.000000	0.333333	0.333333	0.000000	-0.6	

	t_health	map	ct_score	t_score	time_left	ct_money	t_money	\
0	0.000000	-0.666667	-0.857143	-0.857143	0.715105	-0.112782	-0.191489	
1	0.000000	-0.666667	-0.857143	-0.857143	0.545726	-0.368421	-0.395137	
2	-0.561798	-0.666667	-0.857143	-0.857143	0.010000	-0.357143	-0.404255	
3	-0.561798	-0.666667	-0.857143	-0.857143	-0.168575	-0.357143	-0.404255	
4	0.000000	-0.666667	-0.714286	-0.857143	0.714837	0.966165	0.218845	

	t_armor	ct_armor
0	-1.136054	-1.291096
1	-0.115646	0.078767
2	-0.455782	-0.284247
3	-0.455782	-0.284247
4	-1.136054	-0.633562

Mając już przetworzone dane, możemy wydzielić z nich zbiór treningowy oraz testowy.

```
[23]: x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y,
↳test_size = 0.1, random_state = 0)
```

```
[24]: x_train.shape
```

```
[24]: (110169, 38)
```

```
[25]: x_test.shape
```

```
[25]: (12241, 38)
```

Jak widać na podstawie powyższych wywołań, mamy 110169 rekordów w zbiorze testowym oraz 12241 w zbiorze treningowym. W tym momencie możemy rozpocząć inicjalizację modelu.

### 1.3.2 Model MLP

W celu inicjalizacji modelu zostanie wykorzystany klasyfikator MLP, zaimplementowany w bibliotece *scikit-learn* jako *MLPClassifier*.

*MLPClassifier* określa wielowarstwowy perceptron. W przeciwieństwie do innych algorytmów klasyfikacji, takich jak klasyfikator wektorów wspierających lub naiwny klasyfikator Bayesa, *MLPClassifier* przy wykonaniu zadania klasyfikacji opiera się na podstawowej sieci neuronowej.

Istotne cechy wielowarstwowego perceptronu MLP w bibliotece *scikit-learn*: - w warstwie wyjściowej nie ma funkcji aktywacji, - w przypadku scenariuszy regresji błąd kwadratowy jest funkcją straty, a entropia krzyżowa jest funkcją straty klasyfikacji, - może pracować z regresją pojedynczych, jak i wielu wartości docelowych, - w przeciwieństwie do innych popularnych pakietów, implementacja MLP w *scikit* nie obsługuje GPU, - nie jest możliwe dostrojenie parametrów, takich jak różne funkcje aktywacji, inicjatory wagi itp. dla każdej warstwy.

```
[26]: nn_clf = MLPClassifier(verbose = True, warm_start=True, max_iter=1)
```

Opcja *verbose=True* powoduje, że na standardowe wyjście będą drukowane wiadomości o postępach w kolejnych iteracjach.

Ustawienie opcji *warm\_start* na *True* oraz *max\_iter* na *1* umożliwia zatrzymanie procesu uczenia na jednej epoce, dzięki czemu można będzie zobaczyć jak szybko uczy się model.

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.

```
[27]: 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)}')
```

```
Iteration 1, loss = 0.48304638
Iteration 1, score = 0.7519810473000572

C:\Users\adasi\AppData\Local\pypoetry\Cache\virtualenvs\csgo-round-prediction-
GhoYBn2B-py3.10\lib\site-
packages\sklearn\normalization\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1) reached and the
optimization hasn't converged yet.
  warnings.warn(

Iteration 2, loss = 0.45531088
Iteration 2, score = 0.7563924515970918
Iteration 3, loss = 0.44944033
Iteration 3, score = 0.759905236500286
Iteration 4, loss = 0.44529239
Iteration 4, score = 0.7610489339106282
Iteration 5, loss = 0.44248140
Iteration 5, score = 0.762601094681807
Iteration 6, loss = 0.44032367
Iteration 6, score = 0.7641532554529858
Iteration 7, loss = 0.43814777
Iteration 7, score = 0.7642349481251531
Iteration 8, loss = 0.43636235
Iteration 8, score = 0.7678294257005147
Iteration 9, loss = 0.43483083
Iteration 9, score = 0.7683195817335186
Iteration 10, loss = 0.43297083
Iteration 10, score = 0.7707703618985376
Iteration 11, loss = 0.43161232
Iteration 11, score = 0.7675843476840127
Iteration 12, loss = 0.43046747
Iteration 12, score = 0.7711788252593742
Iteration 13, loss = 0.42912346
Iteration 13, score = 0.7688097377665224
Iteration 14, loss = 0.42806524
Iteration 14, score = 0.768973123110857
Iteration 15, loss = 0.42679957
Iteration 15, score = 0.7721591373253819
Iteration 16, loss = 0.42575922
Iteration 16, score = 0.7733845274078915
Iteration 17, loss = 0.42454852
Iteration 17, score = 0.7737112980965607
Iteration 18, loss = 0.42377209
Iteration 18, score = 0.7672575769953435
Iteration 19, loss = 0.42276338
Iteration 19, score = 0.7725676006862184
Iteration 20, loss = 0.42196222
Iteration 20, score = 0.7726492933583857
Iteration 21, loss = 0.42093541
```



Iteration 21, score = 0.7702802058655338  
Iteration 22, loss = 0.42009746  
Iteration 22, score = 0.7737929907687281  
Iteration 23, loss = 0.41940261  
Iteration 23, score = 0.7726492933583857  
Iteration 24, loss = 0.41875859  
Iteration 24, score = 0.7737929907687281  
Iteration 25, loss = 0.41804825  
Iteration 25, score = 0.7709337472428723  
Iteration 26, loss = 0.41708242  
Iteration 26, score = 0.7726492933583857  
Iteration 27, loss = 0.41643347  
Iteration 27, score = 0.771668981292378  
Iteration 28, loss = 0.41554629  
Iteration 28, score = 0.7706886692263704  
Iteration 29, loss = 0.41510362  
Iteration 29, score = 0.7748549955069031  
Iteration 30, loss = 0.41458452  
Iteration 30, score = 0.7752634588677396  
Iteration 31, loss = 0.41401973  
Iteration 31, score = 0.7745282248182338  
Iteration 32, loss = 0.41291744  
Iteration 32, score = 0.7750183808512376  
Iteration 33, loss = 0.41213917  
Iteration 33, score = 0.7762437709337472  
Iteration 34, loss = 0.41171740  
Iteration 34, score = 0.7731394493913896  
Iteration 35, loss = 0.41118649  
Iteration 35, score = 0.7735479127522261  
Iteration 36, loss = 0.41098518  
Iteration 36, score = 0.7746916101625684  
Iteration 37, loss = 0.41031233  
Iteration 37, score = 0.7744465321460665  
Iteration 38, loss = 0.40946225  
Iteration 38, score = 0.7754268442120742  
Iteration 39, loss = 0.40883469  
Iteration 39, score = 0.7745282248182338  
Iteration 40, loss = 0.40878372  
Iteration 40, score = 0.7754268442120742  
Iteration 41, loss = 0.40746862  
Iteration 41, score = 0.7768973123110857  
Iteration 42, loss = 0.40769313  
Iteration 42, score = 0.776733926966751  
Iteration 43, loss = 0.40744460  
Iteration 43, score = 0.7746099174904011  
Iteration 44, loss = 0.40648721  
Iteration 44, score = 0.7774691610162568  
Iteration 45, loss = 0.40648453

Iteration 45, score = 0.7764071562780819  
Iteration 46, loss = 0.40593346  
Iteration 46, score = 0.7790213217874357  
Iteration 47, loss = 0.40515185  
Iteration 47, score = 0.7792663998039376  
Iteration 48, loss = 0.40473150  
Iteration 48, score = 0.7749366881790704  
Iteration 49, loss = 0.40467247  
Iteration 49, score = 0.7749366881790704  
Iteration 50, loss = 0.40447075  
Iteration 50, score = 0.7786945510987664  
Iteration 51, loss = 0.40382859  
Iteration 51, score = 0.7776325463605914  
Iteration 52, loss = 0.40312290  
Iteration 52, score = 0.7777959317049261  
Iteration 53, loss = 0.40239195  
Iteration 53, score = 0.7761620782615799  
Iteration 54, loss = 0.40203138  
Iteration 54, score = 0.776733926966751  
Iteration 55, loss = 0.40191788  
Iteration 55, score = 0.7768973123110857  
Iteration 56, loss = 0.40160361  
Iteration 56, score = 0.7817988726411241  
Iteration 57, loss = 0.40117991  
Iteration 57, score = 0.7813087166081203  
Iteration 58, loss = 0.40105949  
Iteration 58, score = 0.7768973123110857  
Iteration 59, loss = 0.40052801  
Iteration 59, score = 0.7778776243770934  
Iteration 60, loss = 0.40006302  
Iteration 60, score = 0.778041009721428  
Iteration 61, loss = 0.40010319  
Iteration 61, score = 0.7804100972142799  
Iteration 62, loss = 0.39934464  
Iteration 62, score = 0.7786945510987664  
Iteration 63, loss = 0.39998656  
Iteration 63, score = 0.7787762437709338  
Iteration 64, loss = 0.39884197  
Iteration 64, score = 0.7788579364431011  
Iteration 65, loss = 0.39863838  
Iteration 65, score = 0.7772240829997549  
Iteration 66, loss = 0.39790748  
Iteration 66, score = 0.7796748631647741  
Iteration 67, loss = 0.39757052  
Iteration 67, score = 0.7796748631647741  
Iteration 68, loss = 0.39748631  
Iteration 68, score = 0.77828608773793  
Iteration 69, loss = 0.39755209

Iteration 69, score = 0.7776325463605914  
Iteration 70, loss = 0.39679115  
Iteration 70, score = 0.7823707213462953  
Iteration 71, loss = 0.39701380  
Iteration 71, score = 0.7777142390327587  
Iteration 72, loss = 0.39639885  
Iteration 72, score = 0.7786945510987664  
Iteration 73, loss = 0.39643356  
Iteration 73, score = 0.7802467118699453  
Iteration 74, loss = 0.39599294  
Iteration 74, score = 0.7785311657544318  
Iteration 75, loss = 0.39548189  
Iteration 75, score = 0.779103014459603  
Iteration 76, loss = 0.39551674  
Iteration 76, score = 0.7823707213462953  
Iteration 77, loss = 0.39542562  
Iteration 77, score = 0.779919941181276  
Iteration 78, loss = 0.39561553  
Iteration 78, score = 0.7768973123110857  
Iteration 79, loss = 0.39457466  
Iteration 79, score = 0.7814721019524549  
Iteration 80, loss = 0.39433387  
Iteration 80, score = 0.7796748631647741  
Iteration 81, loss = 0.39399993  
Iteration 81, score = 0.7811453312637856  
Iteration 82, loss = 0.39380112  
Iteration 82, score = 0.7811453312637856  
Iteration 83, loss = 0.39374429  
Iteration 83, score = 0.7813904092802876  
Iteration 84, loss = 0.39345340  
Iteration 84, score = 0.7803284045421126  
Iteration 85, loss = 0.39325651  
Iteration 85, score = 0.782043950657626  
Iteration 86, loss = 0.39295653  
Iteration 86, score = 0.7812270239359529  
Iteration 87, loss = 0.39293269  
Iteration 87, score = 0.7801650191977779  
Iteration 88, loss = 0.39270894  
Iteration 88, score = 0.7822073360019606  
Iteration 89, loss = 0.39269217  
Iteration 89, score = 0.7813087166081203  
Iteration 90, loss = 0.39190012  
Iteration 90, score = 0.7788579364431011  
Iteration 91, loss = 0.39163829  
Iteration 91, score = 0.7792663998039376  
Iteration 92, loss = 0.39188425  
Iteration 92, score = 0.7834327260844702  
Iteration 93, loss = 0.39180815

Iteration 93, score = 0.7776325463605914  
Iteration 94, loss = 0.39145954  
Iteration 94, score = 0.784984886855649  
Iteration 95, loss = 0.39104062  
Iteration 95, score = 0.7788579364431011  
Iteration 96, loss = 0.39142644  
Iteration 96, score = 0.7789396291152684  
Iteration 97, loss = 0.39033943  
Iteration 97, score = 0.7830242627236337  
Iteration 98, loss = 0.39077416  
Iteration 98, score = 0.7794297851482722  
Iteration 99, loss = 0.39023198  
Iteration 99, score = 0.7847398088391472  
Iteration 100, loss = 0.39030439  
Iteration 100, score = 0.7817988726411241  
Iteration 101, loss = 0.38996142  
Iteration 101, score = 0.7819622579854587  
Iteration 102, loss = 0.38971042  
Iteration 102, score = 0.7839228821174741  
Iteration 103, loss = 0.38956299  
Iteration 103, score = 0.7829425700514664  
Iteration 104, loss = 0.38925026  
Iteration 104, score = 0.7824524140184625  
Iteration 105, loss = 0.38932151  
Iteration 105, score = 0.782289028674128  
Iteration 106, loss = 0.38909313  
Iteration 106, score = 0.7859651989216567  
Iteration 107, loss = 0.38894666  
Iteration 107, score = 0.7804917898864472  
Iteration 108, loss = 0.38857273  
Iteration 108, score = 0.7784494730822645  
Iteration 109, loss = 0.38860105  
Iteration 109, score = 0.7830242627236337  
Iteration 110, loss = 0.38830965  
Iteration 110, score = 0.7804100972142799  
Iteration 111, loss = 0.38777930  
Iteration 111, score = 0.7827791847071318  
Iteration 112, loss = 0.38777608  
Iteration 112, score = 0.7830242627236337  
Iteration 113, loss = 0.38740522  
Iteration 113, score = 0.7836778041009721  
Iteration 114, loss = 0.38765931  
Iteration 114, score = 0.7844947308226452  
Iteration 115, loss = 0.38772907  
Iteration 115, score = 0.7817988726411241  
Iteration 116, loss = 0.38720848  
Iteration 116, score = 0.7828608773792991  
Iteration 117, loss = 0.38716167

Iteration 117, score = 0.7796748631647741  
Iteration 118, loss = 0.38683493  
Iteration 118, score = 0.7812270239359529  
Iteration 119, loss = 0.38733904  
Iteration 119, score = 0.7856384282329875  
Iteration 120, loss = 0.38654546  
Iteration 120, score = 0.7805734825586145  
Iteration 121, loss = 0.38654408  
Iteration 121, score = 0.7794297851482722  
Iteration 122, loss = 0.38647252  
Iteration 122, score = 0.7839228821174741  
Iteration 123, loss = 0.38655580  
Iteration 123, score = 0.7849031941834818  
Iteration 124, loss = 0.38607105  
Iteration 124, score = 0.786291969610326  
Iteration 125, loss = 0.38546065  
Iteration 125, score = 0.7858018135773222  
Iteration 126, loss = 0.38617359  
Iteration 126, score = 0.7810636385916183  
Iteration 127, loss = 0.38562926  
Iteration 127, score = 0.7833510334123029  
Iteration 128, loss = 0.38613696  
Iteration 128, score = 0.7830242627236337  
Iteration 129, loss = 0.38603855  
Iteration 129, score = 0.784984886855649  
Iteration 130, loss = 0.38502028  
Iteration 130, score = 0.7857201209051549  
Iteration 131, loss = 0.38567017  
Iteration 131, score = 0.7807368679029492  
Iteration 132, loss = 0.38498753  
Iteration 132, score = 0.7800833265256106  
Iteration 133, loss = 0.38496486  
Iteration 133, score = 0.7882525937423414  
Iteration 134, loss = 0.38454651  
Iteration 134, score = 0.7843313454783106  
Iteration 135, loss = 0.38478016  
Iteration 135, score = 0.7841679601339759  
Iteration 136, loss = 0.38458143  
Iteration 136, score = 0.7804917898864472  
Iteration 137, loss = 0.38462924  
Iteration 137, score = 0.7857201209051549  
Iteration 138, loss = 0.38450033  
Iteration 138, score = 0.7826974920349644  
Iteration 139, loss = 0.38442705  
Iteration 139, score = 0.7831876480679683  
Iteration 140, loss = 0.38414316  
Iteration 140, score = 0.7844947308226452  
Iteration 141, loss = 0.38414849

Iteration 141, score = 0.786046891593824  
Iteration 142, loss = 0.38424634  
Iteration 142, score = 0.7853933502164856  
Iteration 143, loss = 0.38329087  
Iteration 143, score = 0.7877624377093375  
Iteration 144, loss = 0.38398212  
Iteration 144, score = 0.7838411894453068  
Iteration 145, loss = 0.38360721  
Iteration 145, score = 0.7805734825586145  
Iteration 146, loss = 0.38353255  
Iteration 146, score = 0.784984886855649  
Iteration 147, loss = 0.38328184  
Iteration 147, score = 0.7857201209051549  
Iteration 148, loss = 0.38324438  
Iteration 148, score = 0.784984886855649  
Iteration 149, loss = 0.38310044  
Iteration 149, score = 0.7848215015113145  
Iteration 150, loss = 0.38333210  
Iteration 150, score = 0.7855567355608202  
Iteration 151, loss = 0.38275769  
Iteration 151, score = 0.7840045747896414  
Iteration 152, loss = 0.38282136  
Iteration 152, score = 0.7841679601339759  
Iteration 153, loss = 0.38285296  
Iteration 153, score = 0.7861285842659913  
Iteration 154, loss = 0.38298842  
Iteration 154, score = 0.7839228821174741  
Iteration 155, loss = 0.38281259  
Iteration 155, score = 0.7843313454783106  
Iteration 156, loss = 0.38242250  
Iteration 156, score = 0.7850665795278163  
Iteration 157, loss = 0.38256364  
Iteration 157, score = 0.7867821256433298  
Iteration 158, loss = 0.38239098  
Iteration 158, score = 0.7848215015113145  
Iteration 159, loss = 0.38208493  
Iteration 159, score = 0.7812270239359529  
Iteration 160, loss = 0.38211172  
Iteration 160, score = 0.7869455109876644  
Iteration 161, loss = 0.38223036  
Iteration 161, score = 0.7858835062494894  
Iteration 162, loss = 0.38239856  
Iteration 162, score = 0.7841679601339759  
Iteration 163, loss = 0.38170069  
Iteration 163, score = 0.7847398088391472  
Iteration 164, loss = 0.38215295  
Iteration 164, score = 0.7857201209051549  
Iteration 165, loss = 0.38127229

Iteration 165, score = 0.782043950657626  
Iteration 166, loss = 0.38146434  
Iteration 166, score = 0.7871905890041663  
Iteration 167, loss = 0.38154105  
Iteration 167, score = 0.7877624377093375  
Iteration 168, loss = 0.38115270  
Iteration 168, score = 0.7846581161669798  
Iteration 169, loss = 0.38149591  
Iteration 169, score = 0.7866187402989951  
Iteration 170, loss = 0.38119897  
Iteration 170, score = 0.7866187402989951  
Iteration 171, loss = 0.38085779  
Iteration 171, score = 0.7856384282329875  
Iteration 172, loss = 0.38155405  
Iteration 172, score = 0.7804917898864472  
Iteration 173, loss = 0.38099045  
Iteration 173, score = 0.786046891593824  
Iteration 174, loss = 0.38086916  
Iteration 174, score = 0.783105955395801  
Iteration 175, loss = 0.38113833  
Iteration 175, score = 0.7866187402989951  
Iteration 176, loss = 0.38069056  
Iteration 176, score = 0.7865370476268279  
Iteration 177, loss = 0.38031419  
Iteration 177, score = 0.7802467118699453  
Iteration 178, loss = 0.38102921  
Iteration 178, score = 0.7842496528061433  
Iteration 179, loss = 0.38036198  
Iteration 179, score = 0.7850665795278163  
Iteration 180, loss = 0.38009549  
Iteration 180, score = 0.7875173596928355  
Iteration 181, loss = 0.38050444  
Iteration 181, score = 0.7857201209051549  
Iteration 182, loss = 0.37978025  
Iteration 182, score = 0.7835144187566375  
Iteration 183, loss = 0.37999640  
Iteration 183, score = 0.785229964872151  
Iteration 184, loss = 0.38036370  
Iteration 184, score = 0.784984886855649  
Iteration 185, loss = 0.37994467  
Iteration 185, score = 0.7856384282329875  
Iteration 186, loss = 0.37996105  
Iteration 186, score = 0.7844947308226452  
Iteration 187, loss = 0.37969532  
Iteration 187, score = 0.7857201209051549  
Iteration 188, loss = 0.37961244  
Iteration 188, score = 0.7858018135773222  
Iteration 189, loss = 0.37984843

```

Iteration 189, score = 0.7817171799689567
Iteration 190, loss = 0.37955472
Iteration 190, score = 0.7856384282329875
Iteration 191, loss = 0.38023581
Iteration 191, score = 0.7863736622824933
Iteration 192, loss = 0.37948620
Iteration 192, score = 0.7882525937423414
Iteration 193, loss = 0.37942357
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
Iteration 193, score = 0.7844947308226452
Iteration 194, loss = 0.37933809
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
Iteration 194, score = 0.7846581161669798
Iteration 195, loss = 0.37882986
Iteration 195, score = 0.7796748631647741
Iteration 196, loss = 0.37898010
Iteration 196, score = 0.7842496528061433
Iteration 197, loss = 0.37889837
Iteration 197, score = 0.7805734825586145
Iteration 198, loss = 0.37959929
Iteration 198, score = 0.7806551752307818
Iteration 199, loss = 0.37886890
Iteration 199, score = 0.7840045747896414
Iteration 200, loss = 0.37919533
Iteration 200, score = 0.7842496528061433

```

Otrzymane wyniki zostały zestawione na wykresie.

```

[48]: plt.plot(scores_mlp_train, color='green', alpha=0.8, label='Train')
plt.plot(scores_mlp_test, color='magenta', alpha=0.8, label='Test')
plt.title("Accuracy over epochs", fontsize=14)
plt.xlabel('Epochs')
plt.legend(loc='lower right')
plt.show()

```





Jak widać, wraz z kolejnymi iteracjami dokładność się zwiększa, jednak dla zbioru testowego rezultaty są gorsze niż dla zbioru treningowego.

Ostatecznie, średnia trafność modelu na rozważanych danych testowych jest następująca:

```
[29]: print(f'Model sieci neuronowej: {nn_clf.score(x_test, y_test)}')
```

Model sieci neuronowej: 0.7842496528061433

## 1.4 Regresja logistyczna

Regresja logistyczna to algorytm nadzorowanego uczenia maszynowego używany do problemów z klasyfikacją binarną. Najlepszym sposobem myślenia o regresji logistycznej jest to, że jest to regresja liniowa, ale dla problemów z klasyfikacją. Podstawowa różnica między regresją liniową, a regresją logistyczną polega na tym, że zakres regresji logistycznej jest ograniczony od 0 do 1. Ponadto - w przeciwieństwie do regresji liniowej - regresja logistyczna nie wymaga liniowej zależności między zmiennymi wejściowymi i wyjściowymi.

```
[30]: log_model = LogisticRegression(verbose=True, n_jobs=-1, warm_start=True,
    ↪max_iter=1)
```

Analogicznie jak poprzednio testujemy model dla 200 iteracji.

```
[31]: 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)}')

```

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   1.9s finished

```

Iteration 1, score = 0.7297606404705498

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 2, score = 0.7373580589821093

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 3, score = 0.7499387304958746

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 4, score = 0.7514908912670534

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 5, score = 0.7520627399722245

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.7s finished

```

Iteration 6, score = 0.7549219834980803

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 7, score = 0.7555755248754187

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.6s finished

```

Iteration 8, score = 0.7571276856465975

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

```

Iteration 9, score = 0.7554938322032514

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

```

Iteration 10, score = 0.7582713830569398

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 11, score = 0.7554938322032514

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 12, score = 0.7576995343517686

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 13, score = 0.755657217547586

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 14, score = 0.7572910709909321

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 15, score = 0.7567192222857609

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.1s finished

Iteration 16, score = 0.7578629196961033

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 17, score = 0.7563924515970918

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 18, score = 0.7548402908259129

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 19, score = 0.7566375296135937

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 20, score = 0.7561473735805898

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 21, score = 0.7560656809084225

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 22, score = 0.7567192222857609
```

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 23, score = 0.7559839882362552

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 24, score = 0.7565558369414264

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 25, score = 0.755657217547586

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 26, score = 0.7563924515970918

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 27, score = 0.7555755248754187

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 28, score = 0.7569643003022629

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 29, score = 0.755657217547586

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 30, score = 0.7565558369414264

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 31, score = 0.7554938322032514

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 32, score = 0.7569643003022629

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 33, score = 0.7552487541867494

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 34, score = 0.7570459929744302

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 35, score = 0.7554121395310841  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 36, score = 0.7570459929744302  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 37, score = 0.7548402908259129  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 38, score = 0.7572093783187648  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 39, score = 0.7551670615145821  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.1s finished  
Iteration 40, score = 0.7563924515970918  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 41, score = 0.7549219834980803  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 42, score = 0.7557389102197533  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 43, score = 0.7563924515970918  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 44, score = 0.7555755248754187  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 45, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 46, score = 0.7546769054815783

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 47, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 48, score = 0.7548402908259129

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 49, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 50, score = 0.7550036761702476

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 51, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 52, score = 0.7550036761702476

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 53, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 54, score = 0.7548402908259129

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 55, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 56, score = 0.7546769054815783

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 57, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 58, score = 0.7548402908259129
```

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 59, score = 0.7550036761702476

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 60, score = 0.7549219834980803

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 61, score = 0.7550036761702476

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 62, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 63, score = 0.7549219834980803

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 64, score = 0.7549219834980803

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 65, score = 0.7552487541867494

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 66, score = 0.7552487541867494

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 67, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 68, score = 0.7552487541867494

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 69, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 70, score = 0.7546769054815783

```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 71, score = 0.7552487541867494

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 72, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 73, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 74, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 75, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 76, score = 0.7554121395310841

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 77, score = 0.7550853688424148

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.7s finished

Iteration 78, score = 0.7554121395310841

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 79, score = 0.7551670615145821

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 80, score = 0.7553304468589167

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 81, score = 0.7552487541867494

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 82, score = 0.7553304468589167
```



[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 83, score = 0.7552487541867494

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.5s finished

Iteration 84, score = 0.7553304468589167

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 85, score = 0.7552487541867494

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 86, score = 0.7551670615145821

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 87, score = 0.7551670615145821

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 88, score = 0.7551670615145821

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 89, score = 0.7550036761702476

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 90, score = 0.7549219834980803

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 91, score = 0.7550853688424148

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 92, score = 0.7546769054815783

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 93, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 94, score = 0.7540233641042399

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 95, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 96, score = 0.7540233641042399

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 97, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 98, score = 0.7541050567764072

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 99, score = 0.7541867494485744

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 100, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 101, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 102, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 103, score = 0.7541867494485744

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 104, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 105, score = 0.7541050567764072

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.3s finished

Iteration 106, score = 0.754350134792909

```

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 107, score = 0.7540233641042399

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.1s finished

Iteration 108, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 109, score = 0.7540233641042399

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 110, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 111, score = 0.7540233641042399

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 112, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 113, score = 0.7541050567764072

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 114, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 115, score = 0.7541050567764072

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 116, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 117, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 118, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 119, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 120, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 121, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 122, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 123, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 124, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 125, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 126, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 127, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 128, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 129, score = 0.754350134792909

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 130, score = 0.7544318274650764

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 131, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 132, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 133, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 134, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 135, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 136, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 137, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 138, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 139, score = 0.7541867494485744  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 140, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 141, score = 0.7541050567764072  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 142, score = 0.7542684421207417

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 143, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 144, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 145, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 146, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 147, score = 0.754350134792909

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 148, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 149, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.8s finished

Iteration 150, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 151, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 152, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 153, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 154, score = 0.7542684421207417

```

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 155, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 156, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 157, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 158, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 159, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 160, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 161, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 162, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.3s finished  
Iteration 163, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.3s finished  
Iteration 164, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 165, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 166, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 167, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 168, score = 0.754350134792909  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 169, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 170, score = 0.754350134792909  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 171, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 172, score = 0.754350134792909  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 173, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 174, score = 0.754350134792909  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 175, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.4s finished  
Iteration 176, score = 0.754350134792909  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished  
Iteration 177, score = 0.7542684421207417  
[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.5s finished  
Iteration 178, score = 0.754350134792909



[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 179, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 180, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 181, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 182, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 183, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 184, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 185, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 186, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 187, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 188, score = 0.7542684421207417

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 189, score = 0.7541867494485744

[Parallel(n\_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.  
[Parallel(n\_jobs=-1)]: Done 1 out of 1 | elapsed: 0.2s finished

Iteration 190, score = 0.7542684421207417

```

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 191, score = 0.7541050567764072

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 192, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 193, score = 0.7541867494485744

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 194, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 195, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 196, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.3s finished

Iteration 197, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 198, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 199, score = 0.7542684421207417

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   0.2s finished

Iteration 200, score = 0.7542684421207417

```

W tym przypadku również stworzymy wykres.

```

[47]: plt.plot(scores_log_train, color='green', alpha=0.8, label='Train')
      plt.plot(scores_log_test, color='magenta', alpha=0.8, label='Test')
      plt.title("Accuracy over epochs", fontsize=14)
      plt.xlabel('Epochs')
      plt.legend(loc='lower right')

```

```
plt.show()
```



Jak widać po niewielkiej liczbie iteracji znajdowane jest dość dobre rozwiązanie, po czym następuje niewielki spadek jakości rozwiązania, a następnie stopniowo wyrównuje się do wartości nieco gorszych niż na początku.

Ostatecznie, po wytrenowaniu modelu skuteczność na danych testowych jest następująca:

```
[33]: print(f"Model regresji logistycznej: {log_model.score(x_test, y_test)}")
```

Model regresji logistycznej: 0.7542684421207417

Regresja logistyczna okazała się trochę gorsza od modelu MLP, jednak czas trenowania modelu jest znacząco niższy.

## 1.5 Optimalizacja hiperparametrów

Powyższe algorytmy MLP i LR posiadają wiele konfigurowalnych hiperparametrów. W przypadku MLP są to m.in. funkcja aktywacji `activation`, przyrost uczenia `learning_rate` oraz `solver`. Dla LR są to m.in. `multi_class` oraz `solver`. Oczywiście dokładność modeli zależy od wartości tych hiperparametrów. W celu znalezienia kombinacji dającej najlepszej rezultaty posługujemy się biblioteką `optuna`. Optimalizatory zasadniczo stale zawężają przestrzeń poszukiwań, wykorzystując zapisy sugerowanych wartości parametrów i ocenianych wartości funkcji celu, co prowadzi do optymalnej przestrzeni poszukiwań, w której uzyskuje się parametry prowadzące do lepszych wartości funkcji celu. Domyślnym optymalizatorem jest Tree-structured Parzen Estimator.

### 1.5.1 Optymalizacja Multilayer Perceptron

```
[ ]: 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)
```

Dla algorytmu MLP najlepsze rezultaty są osiągane dla następujących hiperparametrów: 'activation': 'relu', 'solver': 'adam', 'lri': 0.004620094435728503. Skuteczność klasyfikatora wynosi wtedy 0.795.

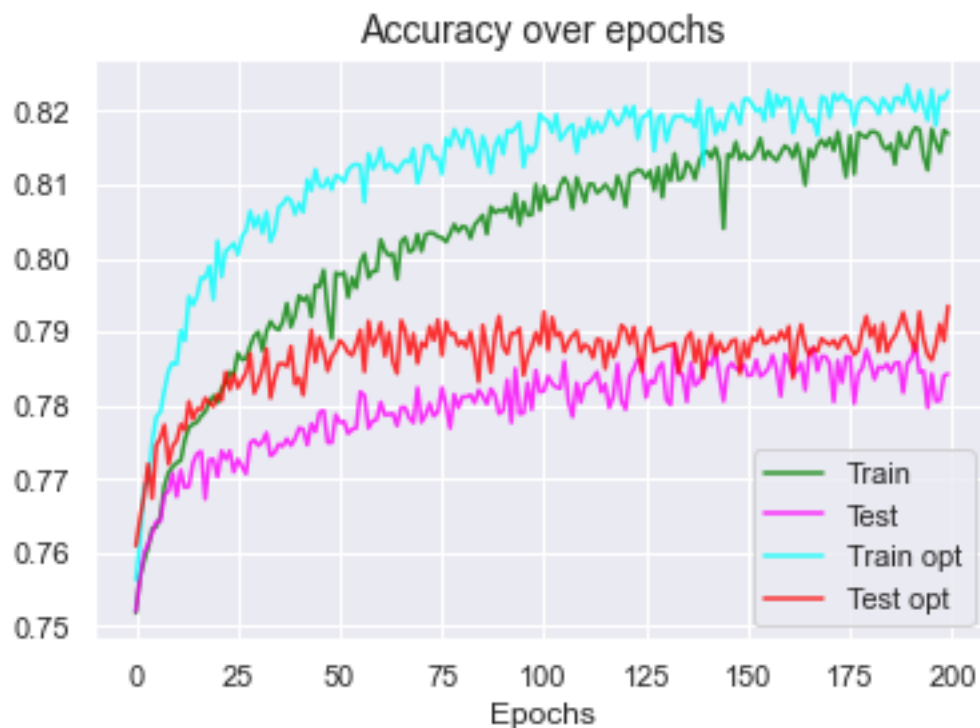
```
[46]: nn_clf_opt = MLPClassifier(warm_start=True, max_iter=1, activation='relu',
↳solver='adam',
                                learning_rate_init=0.004620094435728503)

scores_mlp_train_opt = []
scores_mlp_test_opt = []
for i in range(200):
    nn_clf_opt.fit(x_train, y_train)
    scores_mlp_train_opt.append(nn_clf_opt.score(x_train, y_train))
    scores_mlp_test_opt.append(nn_clf_opt.score(x_test, y_test))

plt.plot(scores_mlp_train, color='green', alpha=0.8, label='Train')
plt.plot(scores_mlp_test, color='magenta', alpha=0.8, label='Test')
```

```
plt.plot(scores_mlp_train_opt, color='cyan', alpha=0.8, label='Train opt')
plt.plot(scores_mlp_test_opt, color='red', alpha=0.8, label='Test opt')
plt.title("Accuracy over epochs", fontsize=14)
plt.xlabel('Epochs')
plt.legend(loc='lower right')
plt.show()
```

C:\Users\adasi\AppData\Local\pypoetry\Cache\virtualenvs\csgo-round-prediction-GhoYBn2B-py3.10\lib\site-packages\sklearn\network\\_multilayer\_perceptron.py:692:  
 ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1) reached and the optimization hasn't converged yet.  
 warnings.warn(



### 1.5.2 Optymalizacja Logistic Regression

```
[ ]: import optuna
import sklearn.datasets

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

```

def objective(trial):
    solver = trial.suggest_categorical("solver", ["newton-cg", "lbfgs",
    ↪ "liblinear", "sag", "saga"])
    if solver == "newton-cg":
        penalty = trial.suggest_categorical("ncg_pen", ["l2", "none"])
    elif solver == "lbfgs":
        penalty = trial.suggest_categorical("lbfgs_pen", ["l2", "none"])
    elif solver == "liblinear":
        penalty = trial.suggest_categorical("l1_pen", ["l1", "l2"])
    elif solver == "sag":
        penalty = trial.suggest_categorical("sag_pen", ["l2", "none"])
    elif solver == "saga":
        penalty = trial.suggest_categorical("saga_pen", ["elasticnet", "l1",
    ↪ "l2", "none"])
    if solver == "liblinear":
        multi_class = trial.suggest_categorical("multiclass1", ["auto", "ovr"])
    else:
        multi_class = trial.suggest_categorical("multiclass2", ["auto", "ovr",
    ↪ "multinomial"])

    l1_ratio = None
    if penalty == "elasticnet":
        l1_ratio = trial.suggest_float("l1r", 0, 1)

    nn_clf = LogisticRegression(solver=solver, warm_start=True, max_iter=1,
    ↪ penalty=penalty,
                                multi_class=multi_class, l1_ratio=l1_ratio)

    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)

```

Dla algorytmu LR najlepsze rezultaty są osiągane dla następujących hiperparametrów: 'solver': 'lbfgs', 'penalty': 'l2', 'multi\_class': 'multinomial'. Skuteczność klasyfikatora wynosi wtedy 0.756.

W przypadku algorytmu MLP następuje wzrost skuteczności klasyfikatora (0.784 -> 0.795). W przypadku LR optymalizacja hiperparametrów następuje nieznaczna poprawa (0.754 -> 0.756). Trenowanie LR trwa zauważalnie krócej - ok. 22 sekundy, a MLP ok. 2,5 minuty.

Na poniższym wykresie przedstawiono porównanie wcześniejszego modelu wraz z zoptymalizowanym modelem.

```

[45]: log_model_opt = LogisticRegression(n_jobs=-1, warm_start=True, max_iter=1,
    ↪ multi_class='multinomial',
                                solver='lbfgs', penalty='l2')

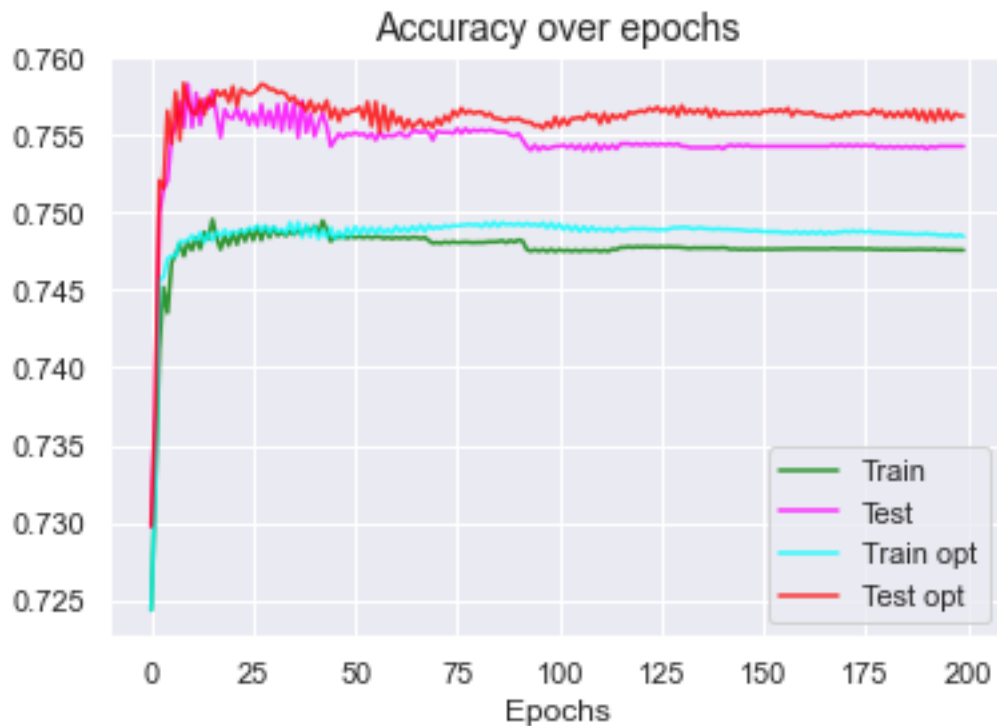
```

```

scores_log_train_opt = []
scores_log_test_opt = []
for i in range(200):
    log_model_opt.fit(x_train, y_train)
    scores_log_train_opt.append(log_model_opt.score(x_train, y_train))
    scores_log_test_opt.append(log_model_opt.score(x_test, y_test))

plt.plot(scores_log_train, color='green', alpha=0.8, label='Train')
plt.plot(scores_log_test, color='magenta', alpha=0.8, label='Test')
plt.plot(scores_log_train_opt, color='cyan', alpha=0.8, label='Train opt')
plt.plot(scores_log_test_opt, color='red', alpha=0.8, label='Test opt')
plt.title("Accuracy over epochs", fontsize=14)
plt.xlabel('Epochs')
plt.legend(loc='lower right')
plt.show()

```



## 1.6 Podział prac

1. Wybór niezbędnych bibliotek - Jakub Michalak
2. Walidacja i wydzielenie danych treningowych - Damian Opoka
3. Analiza i przygotowanie danych - Damian Opoka
4. Definicja modelu - Adam Ryl

5. Weryfikacja krzyżowa - Adam Ryl
6. Dostosowanie hiperparametrów - Jakub Michalak
7. Wygenerowanie wyników - Piotr Kryczka
8. Analiza wyników - Piotr Kryczka