

**VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA
STRUKOVNIH STUDIJA**

BEOGRAD



PROJEKTNI ZADATAK

Tema: *Predviđanje pobjednika runde u igrici CS:GO*

Predmet: **Meko računarstvo I Veštačka inteligencija**

Datum: *(datum predaje rada)*

Student

Marković Boško

RT-122/21

S A D R Ź A J

1. Uvod.....	3
2. Opis problema koji se rešava	3
3. Rešenje.....	4
3.1. Izbor metode i alata.....	4
3.2. Analiza i vizuelizacija podataka.....	5
3.3. Manipulacija podataka.....	8
3.4. Treniranje modela i analiza rezultata.....	10
Prilog (rešenje).....	12
Reference.....	15

1. Uvod

Ovaj projekat ima za cilj treniranje veštačke neuronske mreže kao i primene algoritama mašinskog učenja kako bi se predvideo ishod rundi u popularnoj igri Counter-Strike: Global Offensive. Veliki broj mečeva se odvija na jako sličnom nivou veština, te kroz analizu obimnih podataka iz prethodnih mečeva, možemo da istražimo različite karakteristike i faktore koji utiču na porednika runde kao i njihovu bitnost na sam ishod runde. Neuronska mreža će biti trenirana na podacima iz stvarnih mečeva sa već poznatim ishodima kako bi se stekla sposobnost da što tačnije klasifikuje timove kao potencijalne porednike ili gubitnike runde na osnovu novih unosa. Cilj projekta je pružiti korisnicima alat koji može unaprediti strategiju i donošenje odluka tokom mečeva CS:GO, pružajući im prednost u igri.

2. Opis problema koji se rešava

Counter-Strike: Global Offensive (CS:GO) je najpoznatija takmičarska pucačina gde se dva tima, Teroristi(T) i Protivteroristi(CT), bore jedni protiv drugih u serijama rundi. Cilj terorista je da postave i detoniraju bombu na jednoj od dve unapred određene lokacije na mapi, dok je cilj protivterorista da spreče postavljanje bombe ili je uspešno deaktiviraju ukoliko je postavljena. Igra se odvija na različitim mapama koje imaju svoje karakteristike i strategijske tačke. Tokom rundi, igrači mogu kupovati oružja, opremu i koristiti taktičke ekipe kako bi ostvarili prednost nad protivničkim timom i osigurali poredbu u rundi. Pored timske igre i brzih reakcija, veliki broj parametara može odlučiti na koju će stranu prevagnuti runda.



Slika 2.1 – Primer poredbenog meča sa rezultatom 16:3 i

prikaz na koji način je svaka runda poredbena.

Na primer ukoliko je u rundi ostalo 4 živa igrača u jednoj ekipi, oni će uvek imati veću prednost od 2 preostala živa igrača u drugoj ekipi. Uprkos tome, ukoliko ta dva igrača brane postavljenu bombu sa jačim oružjem sada su oni u prednosti. Svaka runda sa sobom nosi veliki broj ovakvih informacija gde svaka nosi određenu težinu i bitnost u samom ishodu. Ukoliko pretpostavimo da su u svakom timu igrači sličnih veština, sam ishod runde pa i meča zavisi od ovakvih podataka. Treniranjem neuronske mreže nad određenim podacima sa odgovarajućom težinom mi možemo postići zadovoljavajuću tačnost pa i primenu ovakvog programa u stvarnom svetu (npr. uživo izveštavanje publike E-sport turnira ko ima veću šansu da poredi u rundi koja se odvija). Idealno, naša neuronska mreža bi trebala da na osnovu svih tih bitnijih ulaznih parametara da obavi tačnu binarnu klasifikaciju porednika runde.

3. Rešenje

3.1. Izbor metode i alata

Za projekat je korišćeno nekoliko predefinisanih biblioteka (Slika 3.1). Scikit-learn je upotrebljena zbog svoje jednostavnosti i lakoće treniranja neuronske mreže. Prilikom kreiranja projekta, funkcije te biblioteka su dale najbolji odnos tačnosti i jednostavnosti. Iz te biblioteka korišćeni su:

1. MLPClassifier – višeslojni algoritam korišćen za klasifikaciju izlaza
2. LogisticRegression – statistički algoritam regresije povoljan za binarnu klasifikaciju
3. LabelEncoder – enkoder za prebacivanje string vrednosti u celobrojne
4. RobustScaler – skaliranje podataka radi učenja
5. PCA – radi sažimanja originalnog skupa

Takođe su korišćene Numpy, Seaborn i Matplotlib radi analiziranja i prikazivanja podataka kao i (Google) Drive radi unošenja skupa podataka.

Funkcija random je korišćena radi finalnog testiranja modela.

```
[ ] import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

[ ] from google.colab import drive

drive.mount('/content/gdrive/', force_remount=True)

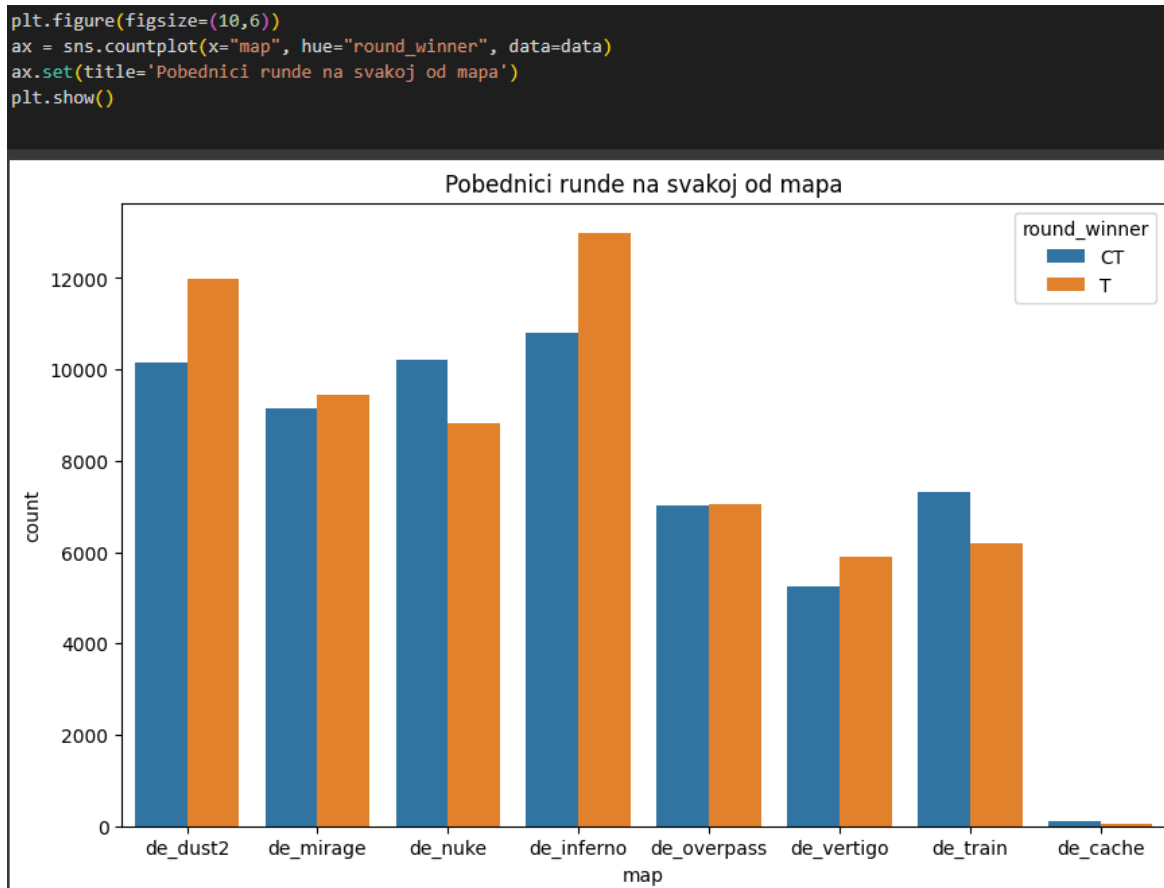
Mounted at /content/gdrive/

[ ] data = pd.read_csv('/content/gdrive/MyDrive/csgo_round_snapshots.csv')
```

Slika 3.1 Importovanje svih neophodnih biblioteka i čitanje .csv fajla u dataframe.

3.2. Analiza i vizuelizacija podataka

Nakon importovanja svih biblioteka kao i učitavanje .csv fajla u dataframe, možemo početi sa analiziranjem nekih podataka. Prilikom odabira skupa podataka, provereno je i utvrđeno da skup podataka nema nijednu null vrednost, stoga možemo krenuti sa analiziranjem podataka. Pre svega možemo videti raspodelu pobeda (Slika 3.2.1) na svakoj od mapa, tu možemo uočiti da se verovatnoće pobede menjaju od mape do mape.



Slika 3.2.1 Pobednici runde na različitim mapama

U podacima takođe možemo videti da je procenat slučajeva gde je bomba postavljena relativno mali, svega 11.58%, ali kasnije možemo videti da to značajno utiče na ishod runde.(Slika 3.2.2)

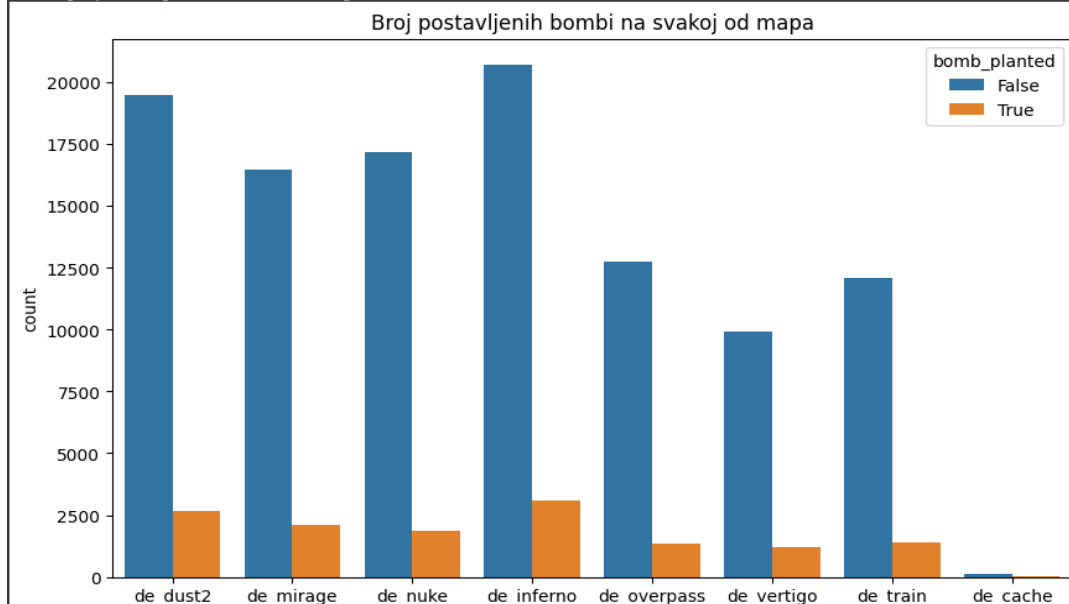
```

postavljenaBomba=0
for i in data["bomb_planted"]:
    if i==True:
        postavljenaBomba+=1
procentatPostavljenihBombi = round(postavljenaBomba/len(data["bomb_planted"])*100, 2)
print(f"Bomba je postavljena u {procentatPostavljenihBombi}% slucajeva.")

plt.figure(figsize=(10,6))
ax = sns.countplot(x="map", hue="bomb_planted", data=data)
ax.set(title='Broj postavljenih bombi na svakoj od mapa')
plt.show()

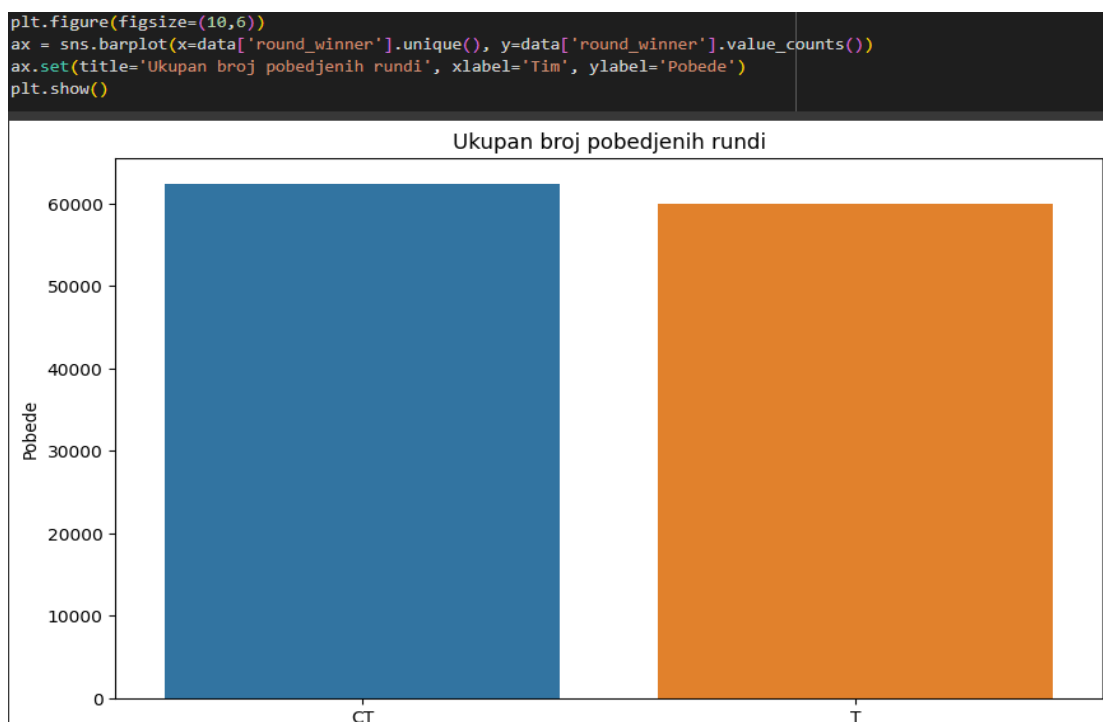
```

Bomba je postavljena u 11.18% slucajeva.



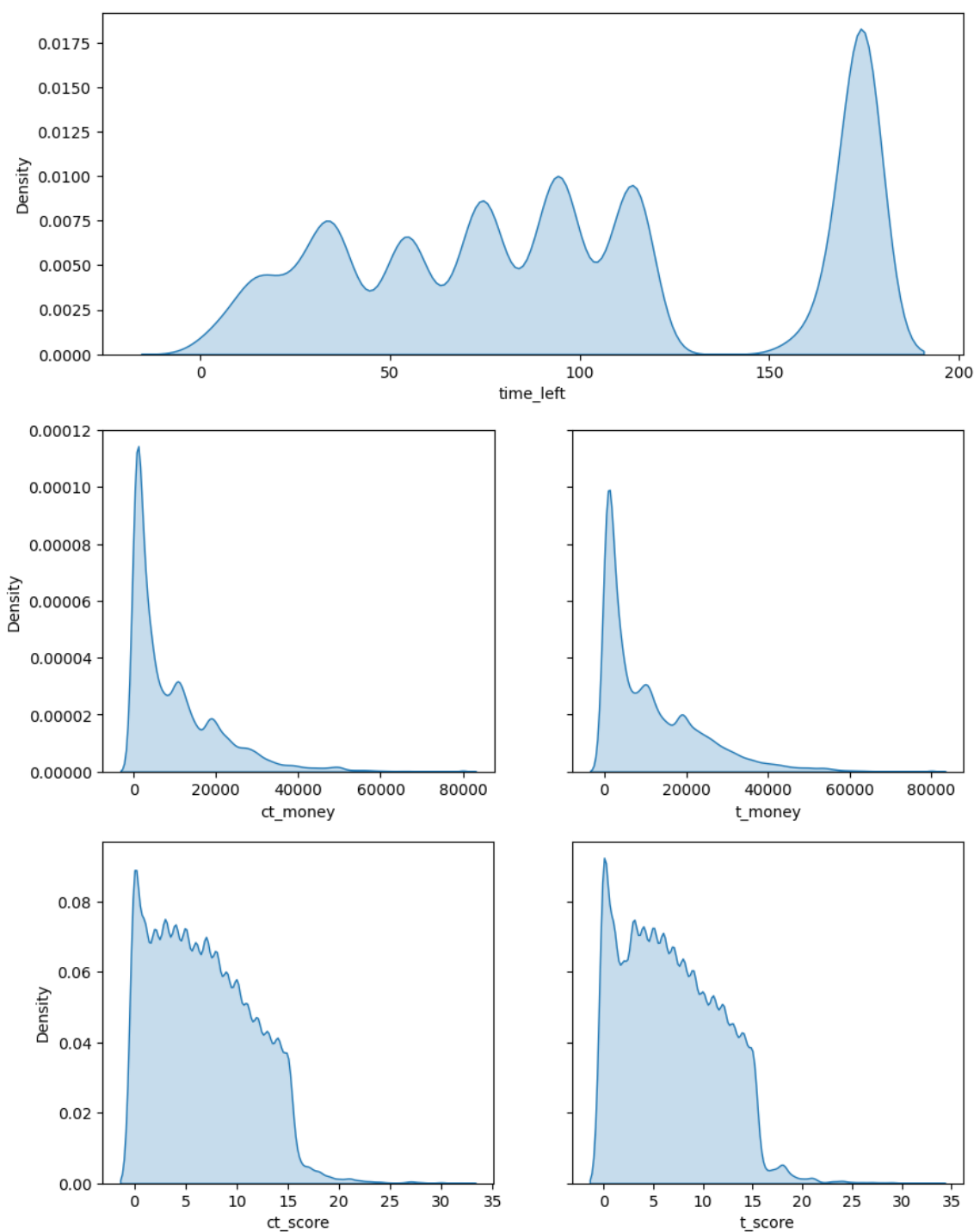
Slika 3.2.2 Broj postavljenih bombi i ukupni procenat u svim rundama.

Skup podataka je poprilično objektivan i reprezentativan situacijama u stvarnom svetu jer je i broj osvojenih rundi skoro pa jednak, sa blagom prednošću za tim CT.(Slika 3.2.3)



Slika 3.2.3 Ukupan broj pobedjenih rundi za svaki tim

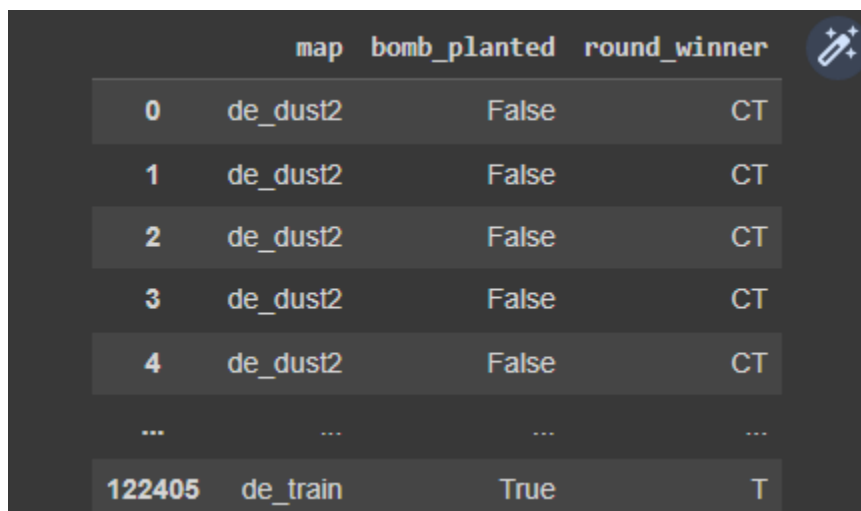
U daljim analizama (Slika 3.2.4), KDE plotom (analogno histogramu) možemo uočiti gustinu raspodele za još neke bitnije podatke – vreme preostalo do kraja runde, ukupan novac po timu kao i trenutni rezultat kada se ta runda odvijala.



Slika 3.2.4 Predstava preostalog vremena, ukupnog novca kao i trenutnog rezultata KDE Plotovima

3.3. Manipulacija podataka

Sada moramo predstaviti kolone koje nisu brojne vrednosti, kao brojne vrednosti. Imamo ih tri i možemo ih videti na Slici 3.3.1.



	map	bomb_planted	round_winner
0	de_dust2	False	CT
1	de_dust2	False	CT
2	de_dust2	False	CT
3	de_dust2	False	CT
4	de_dust2	False	CT
...
122405	de_train	True	T

Slika 3.3.1 Kolone koje nisu predstavljene kao brojne vrednosti u originalnom skupu

Uz pomoć LabelEncodera ćemo kolonu mapa i pobednika predstaviti rednim brojevima, dok ćemo kolonu da li je bomba postavljena konvertovati direktno u Int16 iz Boolean tipa (Slika 3.3.2).

```
[ ] data['bomb_planted'] = data['bomb_planted'].astype(np.int16)

[33] encoder = LabelEncoder()

data['map'] = encoder.fit_transform(data['map'])
mapa_mapa = {index: label for index, label in enumerate(encoder.classes_)}

data['round_winner'] = encoder.fit_transform(data['round_winner'])
mapa_pobednika = {index: label for index, label in enumerate(encoder.classes_)}

[34] print(mapa_mapa)
print(mapa_pobednika)

{0: 'de_cache', 1: 'de_dust2', 2: 'de_inferno', 3: 'de_mirage', 4: 'de_nuke', 5: 'de_overpass', 6: 'de_train', 7: 'de_vertigo'}
{0: 'CT', 1: 'T'}
```

Slika 3.3.2 Kod za konvertovanje „problematicnih“ kolona u celobrojni tip

Nakon ovoga možemo odvojiti originalni data frejm na X i Y skup podataka. X matrica se mora skalirati i za to ćemo koristiti RobustScaler. Izlaz ove komande možemo videti na slici 3.3.3.

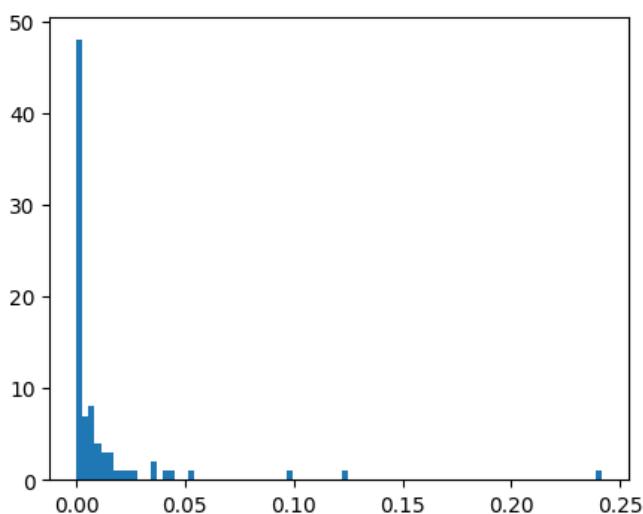
	0	1	2	3	4	5	6	7	8	9	...	86	87	88	89	90	91	92	93	94	95
0	0.715105	-0.857143	-0.857143	-0.666667	0.0	0.000000	0.000000	-1.291096	-1.136054	-0.112782	...	-0.333333	-0.333333	-0.333333	-0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
1	0.545726	-0.857143	-0.857143	-0.666667	0.0	0.000000	0.000000	0.078767	-0.115646	-0.368421	...	-0.333333	-0.333333	-0.333333	0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
2	0.010000	-0.857143	-0.857143	-0.666667	0.0	-0.726667	-0.561798	-0.284247	-0.455782	-0.357143	...	-0.333333	-0.333333	-0.333333	0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
3	-0.168575	-0.857143	-0.857143	-0.666667	0.0	-0.726667	-0.561798	-0.284247	-0.455782	-0.357143	...	-0.333333	-0.333333	-0.333333	-0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
4	0.714837	-0.714286	-0.857143	-0.666667	0.0	0.000000	0.000000	-0.633562	-1.136054	0.966165	...	-0.333333	-0.333333	-0.333333	-0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
...
122405	-0.709837	0.714286	1.142857	1.000000	1.0	-2.000000	-1.449438	-0.623288	0.085034	-0.406015	...	0.000000	0.333333	0.000000	0.000000	0.0	0.0	0.0	-0.5	0.0	0.0
122406	0.714480	0.714286	1.285714	1.000000	0.0	0.000000	0.000000	-0.965753	-0.540816	0.451128	...	0.000000	0.333333	0.000000	-0.333333	0.0	0.0	0.0	-0.5	0.0	0.0
122407	0.178754	0.714286	1.285714	1.000000	0.0	0.000000	0.000000	0.404110	0.479592	-0.323308	...	1.000000	1.000000	0.666667	1.333333	0.5	0.0	0.0	2.0	0.0	0.0

Slika 3.3.3 Skalirana matrica X uz pomoć RobustScalera

U naš PCA model ćemo prvobitno staviti 85 kolona od mogućih 95. Razlog tome je što poslednjih 10 kolona sadrže informacije koliko granata ima svaki od članova tima i te nam kolone ne donose nikakve bitne informacije samom ishodu runde. Te kolone ćemo odmah izbaciti.

```
pca = PCA(n_components=85)
pca.fit(X)

plt.figure(figsize=(5, 4))
plt.hist(pca.explained_variance_ratio_, bins=85)
plt.show()
```



Slika 3.3.4 PCA histogram varijansi svih kolona

Na histogramu komponenti (Slika 3.3.4), možemo uočiti da skoro 50 komponenti ne nosi nikakvu varijabilnost u našem skupu podataka. Cilj je da se uklone ti neželjeni viškovi podataka da bismo mogli da treniramo model manjem skupu podataka koji nosi približnu ako ne i istu količinu informacija.

Bitne kolone ćemo birati uz pomoć funkcije `uzmiKParametara`. Ta funkcija za argument uzima PCA model i razliku maksimalne i željene varijabilnosti u odnosu na originalni skup podataka. Funkcija ide redom po kolonama i sabira njihovu varijabilnost i kad dođe do te

prethodno spomenute razlike, tu se zaustavlja i taj broj kolona vraća. Na primer, ukoliko želimo da naš sažeti skup podataka sadrži 95% varijanse u odnosu na originalni onda ćemo uneti 0.05 kao alpha argument. U ovom slučaju, za 0.05 kao alpha argument, dobijamo najbolji PCA model i on nam vraća 33 kolone. Taj postupak se može videti na slici 3.3.5

```
def uzmiKParametara(pca, alpha):
    ukupnaVarijabilnost = 0

    for odlika, varijabilnost in enumerate(pca.explained_variance_ratio_):
        ukupnaVarijabilnost += varijabilnost
        if (ukupnaVarijabilnost >= 1 - alpha):
            return odlika + 1
    return len(pca.explained_variance_ratio_)

K = uzmiKParametara(pca, 0.05)
X = pca.transform(X)[: , 0:K]
X.shape

(122410, 33)
```

Slika 3.3.5 Funkcija uzmiKParametara i sažimanje originalnog skupa podataka na 33 kolone

3.4. Treniranje modela i analiza rezultata

Pre nego što krenemo sa treniranjem neuronske mreže, moramo odvojiti podatke na trenirajuće i testirajuće. To ćemo odraditi u razmeri 80:20, 80% podataka za treniranje a ostatak za testiranje. (Slika 3.4.1)

```
X_trenirajuci, X_testirajuci, y_trenirajuci, y_testirajuci = train_test_split(X, y, train_size=0.8)
```

Slika 3.4.1 Podela skupova X i Yt na testirajuće i trenirajuće

Sada je samo preostalo da treniramo model neuronske mreže kao i model logističke regresije.

U modelu logističke regresije ćemo ostaviti sve parametre da ostanu na svojim default vrednostima.

U MLPClassifier modelu ćemo naglasiti da solver bude „adam“ – optimizacioni stohastički algoritam za treniranje mreže sa adaptivnim prilagođavanjem stope učenja za svaki parametar, kombinujući procene prvog i drugog momenta gradijenata. Shuffle će biti na true da bi se smanjila pristrasnost jer su svi podaci poređani nekim redosledom (prvo po mapi, osvojenim rundama, po postavljenoj bombi itd...). Konačno, verbose parametar će biti postavljen na true da bismo videli do koje iteracije je stigao model u treniranju (po default-u ide do 200 iteracija) kao i da ispratimo smanjenje funkcije gubitka tokom treniranja modela.

Razlika u tačnosti između ova dva modela varira između 3-4% a ceo proces se može videti na sledećoj slici 3.4.2 :

```

model_log_regr = LogisticRegression()
model_neur_mrez = MLPClassifier(
    solver='adam',
    shuffle=True,
    verbose=True
)

model_log_regr.fit(X_trenirajuci, y_trenirajuci)
model_neur_mrez.fit(X_trenirajuci, y_trenirajuci)

print(f"Model logističke regresije: {round(model_log_regr.score(X_testirajuci, y_testirajuci)*100,2)}%")
print(f"MLP Klasifikator/Model neuronske mreže: {round(model_neur_mrez.score(X_testirajuci, y_testirajuci)*100,2)}%")

Model logističke regresije: 74.59%
MLP Klasifikator/Model neuronske mreže: 78.16%

```

Slika 3.4.2 Deo koda za treniranje modela i ispis konačnih vrednosti

Konačno, možemo testirati „uživo“ ovaj model (Slika 3.4.3). Funkcijom `koJePobedio` se analizira izlaz `predict` funkcije koji može biti 0(CT) ili 1(T). Nasumičnim odabirom bira se 10 rundi iz testirajućeg skupa (između runde 10000 i 20000 radi lepšeg formatiranja izlaza), poredi se izlaz neuronske mreže sa stvarnim rezultatom i taj se izlaz ispisuje.

```

def koJePobedio(x, i):
    pobednikRunde = model_neur_mrez.predict(x)
    if(pobednikRunde==1):
        if(pobednikRunde==y_testirajuci.iloc[i]):
            print(f"Team T je pobedio rundu {i}. (TACNO - taj odgovor je kao i u testirajucem skupu.)")
        else:
            print(f"Team T je pobedio rundu {i}. (NETACNO - taj odgovor NIJE kao u testirajucem skupu.)")
    else:
        if(pobednikRunde==y_testirajuci.iloc[i]):
            print(f"Team CT je pobedio rundu {i}. (TACNO - taj odgovor je kao i u testirajucem skupu.)")
        else:
            print(f"Team CT je pobedio rundu {i}. (NETACNO - taj odgovor NIJE kao u testirajucem skupu.)")

brojac=0
while(brojac<=10):
    indeks = random.randint(10000,20000)
    koJePobedio(X_testirajuci[indeks].reshape(1,-1), indeks)
    brojac+=1

```

Team T	je pobedio rundu 15398.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 10197.	(NETACNO - taj odgovor NIJE kao u testirajucem skupu.)
Team T	je pobedio rundu 19518.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 14984.	(NETACNO - taj odgovor NIJE kao u testirajucem skupu.)
Team CT	je pobedio rundu 18087.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team CT	je pobedio rundu 18916.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 14707.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 19645.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 16100.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 17249.	(TACNO - taj odgovor je kao i u testirajucem skupu.)
Team T	je pobedio rundu 11533.	(TACNO - taj odgovor je kao i u testirajucem skupu.)

Slika 3.4.3 Testiranje modela uživo i poređenje sa tačnim vrednostima iz testirajućeg skupa

Možemo uočiti da ovaj klasifikator u ~80% predviđa dobar izbor prilikom klasifikacije.

Prilog (rešenje)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from google.colab import drive

drive.mount('/content/gdrive/', force_remount=True)
data = pd.read_csv('/content/gdrive/MyDrive/csgo_round_snapshots.csv')

plt.figure(figsize=(10,6))
ax = sns.countplot(x="map", hue="round_winner", data=data)
ax.set(title='Pobednici runde na svakoj od mapa')
plt.show()

# Broj postavljениh bombi od strane T tima na svakoj od mapa
postavljenaBomba=0
for i in data["bomb_planted"]:
    if i==True:
        postavljenaBomba+=1
procenatPostavljenihBombi =
round(postavljenaBomba/len(data["bomb_planted"])*100, 2)
print(f"Bomba je postavljena u {procenatPostavljenihBombi}%
slucajeva.")

plt.figure(figsize=(10,6))
ax = sns.countplot(x="map", hue="bomb_planted", data=data)
ax.set(title='Broj postavljениh bombi na svakoj od mapa')
plt.show()

# Ukupno pobeda
plt.figure(figsize=(10,6))
ax = sns.barplot(x=data['round_winner'].unique(),
y=data['round_winner'].value_counts())
ax.set(title='Ukupan broj pobedjenih rundi', xlabel='Tim',
ylabel='Pobede')
plt.show()
```

```

# Gustina raspodele prema preostalom vremenu
fig, (ax3) = plt.subplots(ncols=1, sharey=True, figsize=(10,4))
sns.kdeplot(data['time_left'], fill=True, ax=ax3)

# Distribucija para po timu ($)
fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True, figsize=(10,4))
sns.kdeplot(data['ct_money'], fill=True, ax=ax1);
sns.kdeplot(data['t_money'], fill=True, ax=ax2);

# Distribucija poena osvojenih rundi
fig, (ax1, ax2) = plt.subplots(ncols=2, sharey=True, figsize=(10,4))
sns.kdeplot(data['ct_score'], fill=True, ax=ax1)
sns.kdeplot(data['t_score'], fill=True, ax=ax2)

# Prebacivanje drugih tipova u int
data.drop(data.select_dtypes(np.number), axis=1)
data['bomb_planted'] = data['bomb_planted'].astype(np.int16)

encoder = LabelEncoder()

data['map'] = encoder.fit_transform(data['map'])
mapa_mapa = {index: label for index, label in
enumerate(encoder.classes_)}
data['round_winner'] = encoder.fit_transform(data['round_winner'])
mapa_pobednika = {index: label for index, label in
enumerate(encoder.classes_)}

# Ispis dobijenih mapa
print(mapa_mapa)
print(mapa_pobednika)

# Razdvajanje skupa podataka na X i Y skupove
y = data['round_winner']
X = data.drop('round_winner', axis=1)

# Skaliranje svake promenljive u tabeli uz pomoc RobustScalera
scaler = RobustScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X)

# Ne uzimamo zadnjih 10 kolona od mogucih 95 jer skoro nebitne za
treniranje mreze (kolicina granata koju poseduje svaki od clanova tima)
pca = PCA(n_components=85)
pca.fit(X)

```

```

# Prikaz histograma PCA
plt.figure(figsize=(5, 4))
plt.hist(pca.explained_variance_ratio_, bins=85)
plt.show()

# Sazimanje na zeljenu varijabilnost
def uzmiKParametara(pca, alpha):
    ukupnaVarijabilnost = 0

    for odlika, varijabilnost in
enumerate(pca.explained_variance_ratio_):
        ukupnaVarijabilnost += varijabilnost
        if (ukupnaVarijabilnost >= 1 - alpha):
            return odlika + 1
    return len(pca.explained_variance_ratio_)

K = uzmiKParametara(pca, 0.05)
X = pca.transform(X)[: , 0:K]
X.shape

pd.DataFrame(X)

# Razdvajanje skupova na trenirajuci i testirajuci 80/20
X_trenirajuci, X_testirajuci, y_trenirajuci, y_testirajuci =
train_test_split(X, y, train_size=0.8)

# Treniranje modela
model_log_regr = LogisticRegression()
model_neur_mrez = MLPClassifier(
    solver='adam',
    shuffle=True,
    verbose=True
)

model_log_regr.fit(X_trenirajuci, y_trenirajuci)
model_neur_mrez.fit(X_trenirajuci, y_trenirajuci)

# Ispis tacnosti modela
print(f"Model logističke
regresije: {round(model_log_regr.score(X_testirajuci,
y_testirajuci)*100,2)}%")
print(f"MLP Klasifikator/Model neuronske moreže:
{round(model_neur_mrez.score(X_testirajuci, y_testirajuci)*100,2)}%")

```

```

# Funkcija za proveru rezultata predict funkcije sa pravim tacnim
vrednostima
def koJePobedio(x, i):
    pobednikRunde = model_neur_mrez.predict(x)
    if(pobednikRunde==1):
        if(pobednikRunde==y_testirajuci.iloc[i]):
            print(f"Team T je pobedio rundu {i}.      (TACNO - taj odgovor
je kao i u testirajucem skupu.)")
        else:
            print(f"Team T je pobedio rundu {i}.      (NETACNO - taj odgovor
NIJE kao u testirajucem skupu.)")
    else:
        if(pobednikRunde==y_testirajuci.iloc[i]):
            print(f"Team CT je pobedio rundu {i}.      (TACNO - taj odgovor
je kao i u testirajucem skupu.)")
        else:
            print(f"Team CT je pobedio rundu {i}.      (NETACNO - taj odgovor
NIJE kao u testirajucem skupu.)")

# Petlja za ispis 10 nasumicnih rundi I njihovih proveravanja u modelu
brojac=0
while(brojac<=10):
    indeks = random.randint(10000,20000)
    koJePobedio(X_testirajuci[indeks].reshape(1,-1), indeks)
    brojac+=1

```

Projekat je pisan u Google Collab okruženju i provereno radi u istom. Projekat je pisan po notebook poljima dok je ovde prekopiran kao jedna celina. [Link za pristup tom fajlu](#). Projekat se može odatle skinuti pa dodati na lični Drive. Drive povezati sa Google Collabom i odatle se može otvoriti na ličnom nalogu.

Reference

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

<https://blog.scope.gg/cs-go-stats-why-is-it-so-important-en/>

https://www.youtube.com/watch?v=Vi3cMKqwGkA&ab_channel=ProGuidesCSGOTips%2CTricksandGuides

Priručnici za laboratorijske vežbe:

Meko računarstvo – M. Pejanović, M. Živanović 2023. god.

Veštačka Inteligencija – M. Pejanović, M. Živanović 2023. god.