*Želim se*

*zahvaliti svom mentoru doc.dr.sc Marku Đuraševiću na odličnoj suradnji tijekom ove akademske godine. Također bih se želio zahvaliti svojoj djevojci Magdaleni koja mi je bila najveća podrška, te obitelji na potpori kroz ove 3 godine studija.*

# Sadržaj

# 1. Uvod

Digitalni razvoj jedan je od glavnih izazova današnjeg vremena, a taj razvoj nije zaobišao niti nogomet. Napretkom tehnologije i digitalizacijom došlo je do intenzivnog razvoja uređaja za praćenje, prikupljanja podataka i upotrebe iste u trenažnom procesu. Analiza podataka postala je svakidašnji dio gotovo svakoga sporta današnjeg doba, a nogomet nije iznimka tog pravila. Ista je u nogometu postala je važan čimbenik u procesu slaganja momčad i pripremanja taktike sukladno protivniku. Također igračima daje uvid u njihove mane i vrline, čime doprinosi sportskom razvoju pojedinog igrača.

Jedna od popularnijih metrika u današnjem nogometu odnosi se na predviđanje vjerojatnosti za zgoditkom, popularnije *Expected Goals*, skraćeno *xG*. Ova metrika nam omogućuje da evaluiramo timsku izvedbu, te izvedbu pojedinog igrača. Nogomet je igra u kojoj rezultat po utakmici nije stvarno ogledalo onoga što se događalo na terenu, zato se brojni sportski analitičari okreću naprednim modelima kao što su *xG*, koji nam zorno prikazuje kvalitete prilika u kojima se pojedina momčad našla. Ukratko, *xG* daje vjerojatnost da pojedini udarac rezultira zgoditkom, temeljeno na karakteristikama pojedinog udarca. Karakteristike koje model koristi su lokacija igrača na terenu, dio tijela kojim je udarac izvršen, izvorna pozicija igrača te mnoge druge. Primjera radi, identičan udarac s *xG* vrijednosti 0.5 će od 10 pokušaja, 5 puta završiti u mreži. *xG* vrijednost ne odnosi se na određenog igrača, već predviđa vrijednost prosječnog igrača ili ekipe u sličnoj situaciji. Usporedbom *xG* vrijednosti i stvarnog broja postignutih pogodaka možemo donositi zaključke o igračevim sposobnostima ili u krajnjem slučaju o sreći. Igrač koji u kontinuitetu postiže više golova nego što je njegov ukupni *xG*, za njega kažemo da je igrač s iznadprosječnom sposobnosti postizanja pogodaka. Za ekipu čija je *xG* razlika negativna kažemo da je imala manjak sreće ili ima ispodprosječnu sposobnosti postizanja pogodaka. Također možemo razlikovati kvalitetu prilika pojedine ekipe sukladno situacijama iz kojih je udarac započet, otvorena igra, udarac iz kuta, slobodni udarac i brojni drugi.

Cilj ovoga rada je razviti razne modele predviđanja *xG* vrijednosti i usporediti uspješnosti pojedinog modela. Razvijena su dva modela, temeljena na dva pristupa, lo-

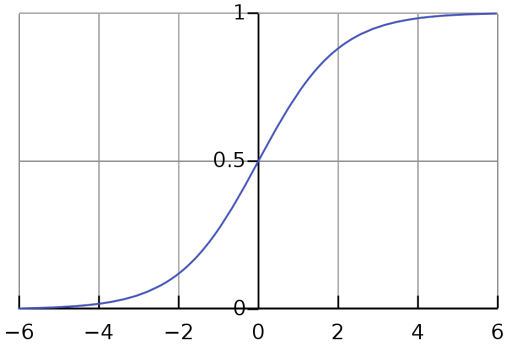
gistička regresija i slučajne šume. U narednim poglavljima opisat ćemo rad navedenih algoritama, te vidjeti proces učenja pojednih modela.

# 2. Modeli

## 2.1. Logistička regresija

Logistička regresija je tip generaliziranog linearnog modela koji predviđa vjerojatnost binarnog ishoda temeljenog na ulaznim podatcima. Naziv logistička dolazi zbog upotrebe logističke funckije, također poznate kao sigmoidna funkcija, koja za ulazne vrijednost vraća vrijednost između 0 i 1. Graf sigmoidne funkcije prikazan je na slici 2.1, dok je sama fukcija definirana na sljedeći način:

$$\sigma(\alpha) = \frac{1}{1 + exp(-\alpha)} \tag{2.1}$$




**Slika 2.1:** Graf sigmoidne funkcije

U našem slučaju $\alpha$ predstavlja linearnu kombinaciju ulaznih parametara zajedno s odgovarajućim koeficijentima. Funkcija kao rezultat vraća vjerojatnost za pojedinu klasu na način da zbroj tih vjerojatnosti iznosi 1.

$$P(1) = \mu \;\; P(0) = 1 - \mu \tag{2.2}$$

Model uči na način da nakon svake iteracije dobivenu predviđevnu vrijednost uspoređuje sa stvarnom vrijednošću i računa pogrešku unakrsne entropije, te ažurira koefi-

cijente ulaznih parametara kako bi se vrijenost entropije sustava minimizirala. Greška unakrsne entropije računa se na sljedeći način.

$$L(y, h(x)) = -y \ln(h(x)) - (1 - y) \ln(1 - h(x)) \tag{2.3}$$

Gdje je $y$ oznaka klase, a *h(x)* vjerojatnost pojave pojedine klase. U našem primjeru *h(1)* predstavljat će vjerojatnost pojave klase s oznakom 1, što će ujedno biti i *xG* vrijednost.

### 2.1.1. Gradijentni spust

Postupak optimizacije izvodi te tehnikom gradijentnog spusta gdje je cilj pronaći minumum fukcije te ažurirati koeficijente ulaznih parametara. Bazira se na spuštanju po najvećoj strmini. Ideja gradijentnog spusta jest da se, krenuvši od neke početne točke (početnog vektora **w**), postepeno "spuštamo" niz površinu funkcije E(**w**|*D*) u smjeru suprotnome od gradijent-vektora u točki **w**. Postupak ponavljamo dok se ne spustimo u točku u kojoj je gradijent jednak nuli ili je barem dovoljno blizu nuli. Formalno, težine **w** u svakoj iteraciji gradijentnog spusta ažuriramo na ovaj način:

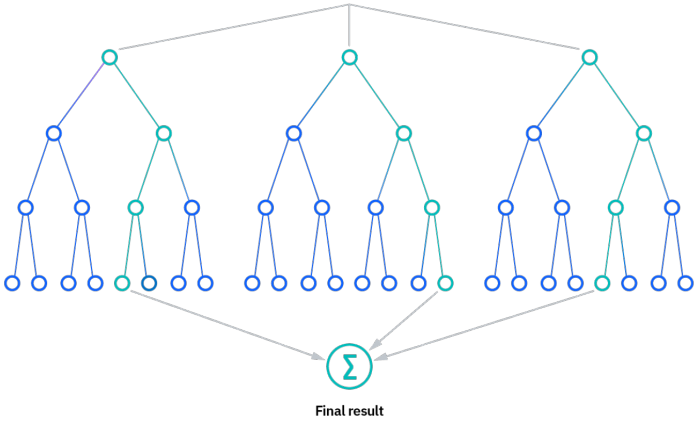
$$\mathbf{w} \longleftarrow \mathbf{w} - \eta \nabla E(\mathbf{w}|D) \tag{2.4}$$

Parametar $\eta$ predstavlja stopu učenja koja određuje veličinu koraka koje radimo spuštajući se prema minimumu. Ako radimo korake u tom smjeru, spustit ćemo se do minimuma.

## 2.2. Slučajne šume

Stabla odluke popularan su način primjene za mnoge probleme strojnog učenja, čiji je cilj stvoriti model koji će predvidjeti vrijednost tražene varijable temeljem određenih ulaznih parametara. Stabla odluke imaju određene nedostatke, nisu pogodna za velike probleme, sklona su prenaučenosti, a mala promjena u skupu za učenje rezultirat će velikim promjenama u generiranju stabla te konačnim odlukama. Cilj ovog pristupa je, umjesto stvaranja jednog modela, izgraditi složeniji model koji se sastoji od više manjih, jednostavnih modela. Umjesto da se skupa za učenje stvori jedno stablo, skup za učenje podijeli se u manje skupove, te se za svaki od njih izgrađuje posebno stablo odluke, a kao rezultat uzima klasu s najvećom frekvencijom pojavljivanja. Ovaj pristup nazivamo slučajnim jer se iz izvornog skupa za učenje izvače nasumični retci po kojima

se gradi svako od stabala, broj redaka će biti jednak izvornom skupu, no retci se smiju ponavljati. U narednom koraku generiraju se stabla odluke, a značajke po kojima ćemo generirati pojedino stablo također biramo nasumično. Stručnjaci tvrde da broj značajki po kojima gradimo stablo treba biti otprilike jednako $\log(n)$, gdje je $n$ broj značajki izvornog skupa. U našem primjeru postotak klasa s oznakom 1, bit će predviđena *xG* vrijednost.




**Slika 2.2:** Skica slučajne šume

# 3. Opis baze

## 3.1. O bazi

Podatci korišteni u procesu učenja i testiranja modela ovog projekta uzeti su iz javno dostupne *Statsbomb* baze podataka. *Statsbomb* je vodeći tvrtka u području nogometne analitike te javno objavljuje kvalitetne podatke, savršene za ovaj projekt. Dio u bazi koji je nama zanimljiv odnosi se na događaje na terenu. Svi događaji na terenu detaljno su označeni i opisani prikladnim parametrima. Dakako, naš fokus je na događajima vezanim za udarce na gol. U narednom dijelu teksta opisat ćemo parametre koji opisuju udarac, zajedno sa svim vrijednostima koje ti parametri mogu poprimiti.

- **under_pressure:** Poprima vrijednost True ili False, sukladno tome je li igrač pod pritiskom.
- **position:** Izvorna pozicija igrača na terenu, svaka pozicija ima svoj jednistveni id.


**Slika 3.1:** Pozicije igrača na terenu

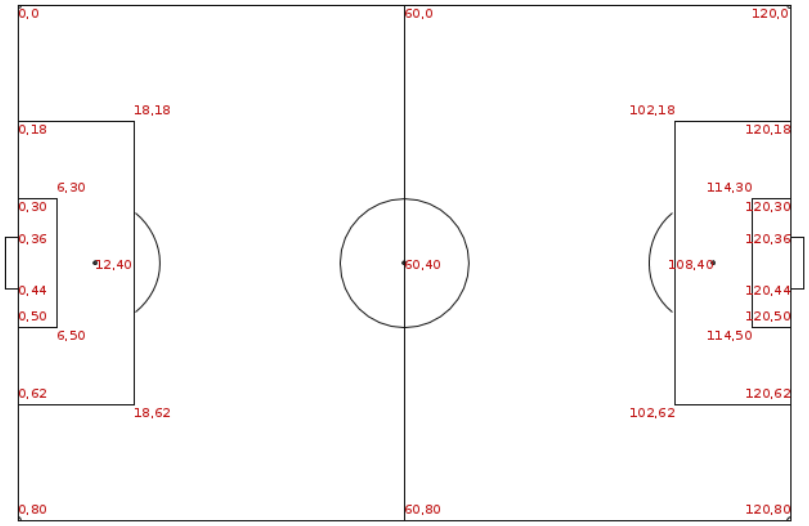
- **play_pattern:** Opisuje tijek igre koji je prethodio udarcu. Vrijednosti koje ovaj parametar može poprimiti su ”Iz udarca iz kuta”, ”Iz protunapada”, ”Iz otvorene igre” i slično.

- **body_part:** Dio tijela kojim je udarac upućen. Udarac može biti upućen lijevom nogom, desnom nogom, glavom ili ostalim dijelom tijela (koljeno, prsa, itd.).

- **technique:** Tehnika kojom je udarac upućen. Tu ubrajamo udarac petom, volej, lob, škarice i slično.

- **type:** Opisuje odakle je udarac upućen. Udarac može biti upućen iz kornera, iz slobodnog udarca, iz kaznenog udarca te iz otvorene igre.

- **location:** Koordinate na terenu odakle je upućen udarac. Ovaj parametar nam služi za izračun udaljenosti do gola, te kut pod kojim se gol vidi gledajući iz te lokacije.




**Slika 3.2:** Koordinate točaka na terenu

## 3.2. Priprema podataka

Baza u sebi sadrži brojne lige i utakmice, no ponajviše se podataka nalazi o španjolskoj ligi, stoga smo upravo Španjolsku odabrali za treniranje i testiranje naših modela. Za početak potrebno je filtrirati utakmice španjolske lige, to smo napravili sljedećim kodom:

```
def filter_leagues(self, dir):
    spain_dir = os.listdir(dir+"matches/"+str(SPAIN_ID))
    for file in spain_dir:
        f = open(dir + "matches/" +
            str(SPAIN_ID) + "/" + file, "r", encoding="utf8")
        data = json.load(f)
```

```
            temp_list = []
            for i in data:
                temp_list.append(i["match_id"])
            spain_league_matches[int(file.split(".")[0])] = temp_list
        return
```

Potom je potrebno iz baze podataka izvući parametre koji su nam bitni za udarce. Iz događaja koji opisuju udarce uzimamo parametre navedene u poglavlju 3.1., zajedno sa stvarnom vrijednosti *xG* te rezultatom udarca.

```
    def extractShots(self, dir, key):
        for match in spain_league_matches[key]:
            f = open(dir + str(match) + ".json", "r", encoding="utf8")
            data = json.load(f)

            for i in data:
                type = i["type"]
                if(type['name'] == "Shot"):

                    useful_shot_info = []
                    shot_info = i["shot"]

                    if("under_pressure" in i):
                        useful_shot_info.append(-1)
                    else :
                        useful_shot_info.append(1)
                    useful_shot_info.append(i["position"]["id"])
                    useful_shot_info.append(i["play_pattern"]["id"])
                    useful_shot_info.append(shot_info["body_part"]["id"])
                    useful_shot_info.append(shot_info["technique"]["id"])
                    useful_shot_info.append(shot_info["type"]["id"])
                    useful_shot_info.append(shot_angle(i["location"]))
                    useful_shot_info.append(distance_to_goal(i["location"]))
                    useful_shot_info.append(shot_info["statsbomb_xg"])
                    if(shot_info["outcome"]["id"] == 97):
                        useful_shot_info.append(1)
                    else:
                        useful_shot_info.append(0)

                    self.shots.append(useful_shot_info)
            f.close()
        return
```

Funkcija shot_angle računa kut koji zatvaraju pravci povučeni iz točke odakle je upućen udarac, do svake od vratica gola. Funckija distance_to_goal računa udaljenost do središta gola u metrima. Obje funkcije definirane su u nastavku.

```
def shot_angle(location):
    x = location[0]
    y = location[1]
    if(x > 60):
```

```
            if(y >= 36 and y <= 44):
                alfa = atan2((y-36), (120-x))
                beta = atan2((44-y), (120-x))
                return (beta + alfa)
            else:
                alfa = atan2((36-y), (120-x))
                beta = atan2((44-y), (120-x))
                return (beta - alfa)
        else:
            if(y >= 36 and y <= 44):
                alfa = atan2((y-36), x)
                beta = atan2((44-y), x)
                return (beta + alfa)
            else:
                alfa = atan2((36-y), x)
                beta = atan2((44-y), x)
                return (beta - alfa)


def distance_to_goal(location):
    x = location[0]
    y = location[1]
    if(x>60):
        return sqrt((120-x)**2 + (40-y)**2)
    else:
        return sqrt(x**2 + (40-y)**2)
```

Pojednine kategoričke varijable mapirat ćemo na njihove jednistvene ključeve, a svaki udarac će u csv (*comma-separated values*) datoteci za treniranje biti reprezentiran jednim retkom u sljedećem formatu:

$$under\_pressure, position, play\_pattern, body\_part, technique, type, angle, distance, statsbomb\_xg, outcome \tag{3.1}$$

Kada opisnike zamjenimo stvarnim vrijednostima, redak u datoteci za treniranje izgleda:

$$1, 24, 4, 40, 93, 87, 0.6875682269849009, 9.068627239003705, 0.45647225, 1 \tag{3.2}$$

# 4. Treniranje

Nakon pripreme i razdvajanja podataka u skup za treniranje i skup za testiranje, slijedi sam proces treniranja. Skup za treniranje sadrži ukupno 10487 udaraca iz 13 različitih sezona, dok skup za testiranje sadrži ukupno 2356 udaraca iz 3 različite sezone. U narednom poglavlju slijedi prikaz procesa treniranja pristupom logističke regresije i slučajnih šuma te usporedba pojednih pristupa.

## 4.1. Logistička regresija

U procesu učenja koristili smo programski jezik Python, te njegovu biblioteku *scikit-learn* unutar koje postoji ugrađeni model *LogisticRegression*. Biblioteka *scikit-learn* nam uvelike olakšava posao jer nakon uvodnog podešavanja parametara modela jedino što je potrebno jest predati podatke za treniranje. Funckije modela *LogisticRegression* koje smo koristili unutar projekta:

- fit(X,Y): Trenira model sukladno predanim parametrima.
- predict(X): Predviđa oznaku klase za dani uzorak X.
- predict_proba(X): Predviđa vjerojatnost pojave pojedine klase za dani uzorak X.
- score(X,Y): Vraća točnost predviđanja klasa za predani uzorak.

Postupak inicijalizacije samog modela učinili smo na sljedeći način:

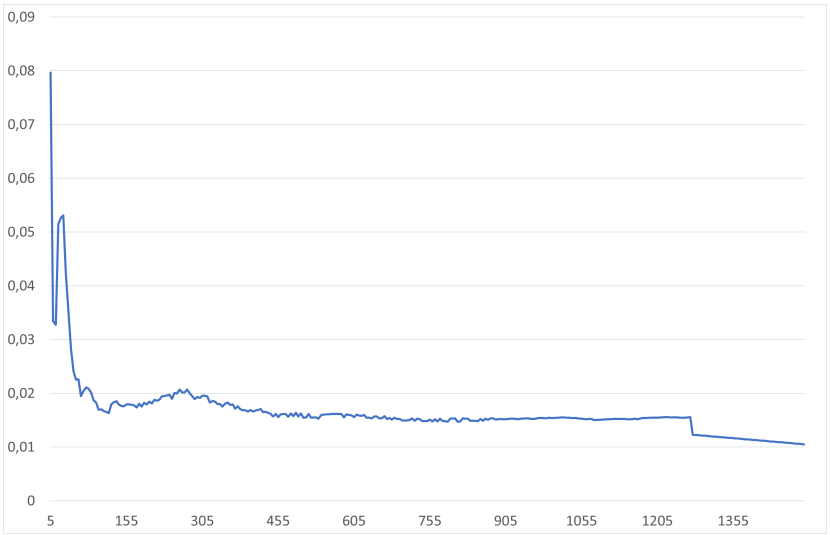
```
def __init__(self):
    self.model = LogisticRegression(max_iter=1000, solver='saga')
```

Argumentom *solver* specificiramo algoritam koji će biti korišten za optimizaciju samog modela. Ispitivanjem svakog od ponuđenih algoritama, "saga" se pokazao najuspješnijim. *Saga* je napredna inkrementalna metoda gradijentog spusta. Dakako, zbog složenosti problema i veličine podatkovnog skupa, morali smo povećati maksimalan broj iteracija potrebnih za konvergenciju na 1000. Postavljanjem ovih parametara, naš model je spreman za učitavanje podataka i učenje.

Treniranje je opisano unutar fukcije train, dok se konkretno učenje modela događa unutar ugrađene funkcije fit.

```
def train(self, file):
    with open(file, mode='r') as f:
        reader = csv.reader(f)
        data = []
        for row in reader:
            data.append(row)
    f.close()

    np_array = np.array(data)
    train_input = np_array[:,:-2]
    train_output = np_array[:, -1]

    self.model.fit(train_input, train_output)
    return
```

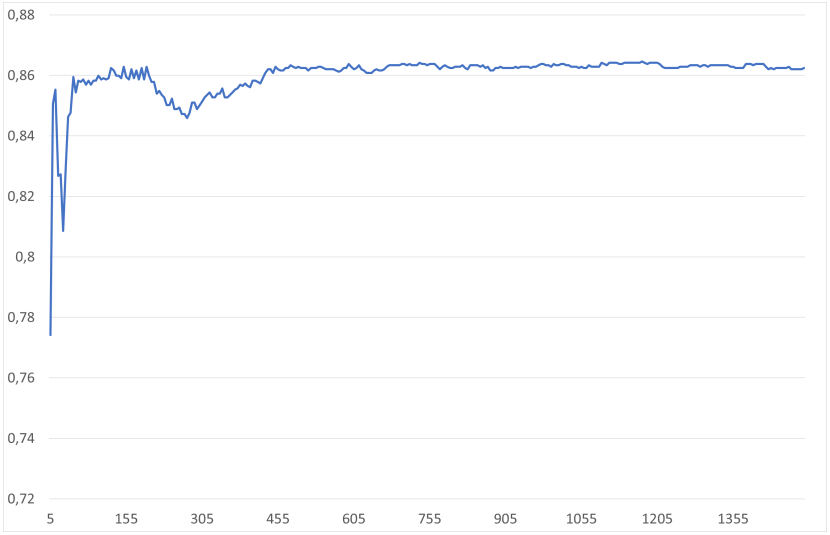
Proces učenja kroz etape možemo prikazati grafom na slici 4.1. Graf je dobiven na način da se računa srednja kvadratna pogreška između predviđenih vrijednosti koje je model izbacio i stvarnih *xG* vrijednosti iz baze. Svaka etapa uzima 5 novih udaraca i na njima uči te ažurira parametre modela. Npr. model u prvoj etapi uzima 5 udaraca na uči, u idućem prolasku uči na 10 udaraca, u idućem 15 i tako dalje. Na horizontalnoj osi je prikazan broj udaraca, a na vertikalnoj osi vrijednost srednje kvadratne pogreške za zadanu populaciju. Model nakon svake od etapa testiramo i računamo pogrešku.




**Slika 4.1:** Graf srednje kvadratne pogreške

Model isprva poprilično loše aproksimira *xG* vrijednost, no kako učenje odmiče, tako se model bolje prilagođava ulaznim podatcima, a aproksimacija *xG* vrijednosti postaje poprilično točna. Kako model napreduje tako se povećava preciznost kojom

predviđa određenu klasu, u narednom grafu možemo vidjeti kako se preciznost povećava kroz etape.




**Slika 4.2:** Graf *score* funkcije

## 4.2. Slučajne šume

U procesu učenja koristili smo programski jezik Python, te njegovu biblioteku *scikit-learn* unutar koje postoji ugrađeni model *RandomForestRegressor*. Biblioteka *scikit-learn* nam uvelike olakšava posao jer nakon uvodnog podešavanja parametara modela jedino što je potrebno jest predati podatke za treniranje. Funckije modela *RandomForestRegressor* koje smo koristili unutar projekta:

- fit(X,Y): Gradi šumu stabala prema skupu za treniranje.
- predict(X): Predviđa vjerojatnost klase za dani uzorak X.
- score(X,Y): Vraća koeficijent determinacije $\mathbf{R}^2$ koji je definiran kao $1 - \frac{u}{v}$ gdje je $u$ suma kvadrata $((y\_true - y\_pred)^2).sum()$, a $v$ je ukupna suma kvadrata $((y\_true - y\_true.mean())^2).sum()$. Najbolji mogući *score* je 1.0, no može biti i negativan jer model može biti proizvoljno lošiji.

Postupak inicijalizacije samog modela učinili smo na sljedeći način:

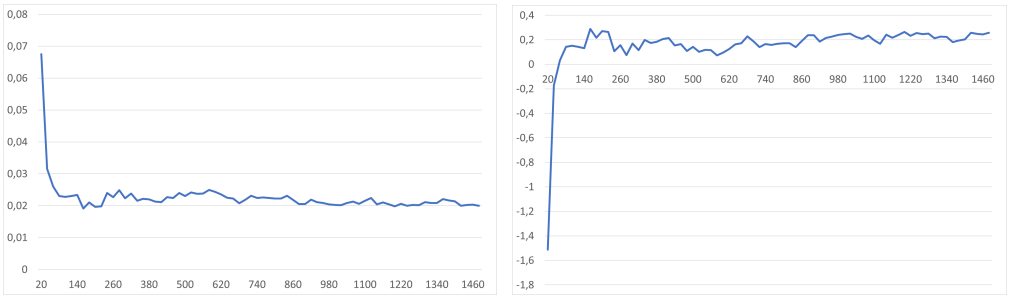
```
def __init__(self):
    self.model = RandomForestRegressor(max_depth=6)
```

Argumentom *max_depth* definirali smo maksimalnu dubinu do koje je potrebno generirati pojedino stablo prije nego li donese odluku. Podešavanjem parametra *max_depth*

dolazi do podrezivanja stabala, čime se povećava entropija listova samog stabla, što pridonosi smanjenju šanse za prenaučenost modela. Ostali parametri ostavljeni su na izvornim vrijednostima. Kriterij za odabir kvalitete raspodjele je srednja kvadratna pogreška (*MSE*). Minimalni broj elemenata potreban za raspodjelu ostavljen je na 2, a minimalan broj elemenata u listu je 1. Postavljanjem ovih parametara, naš model je spreman za učitavanje podataka i učenje. Treniranje je opisano unutar fukcije train, dok se konkretno učenje modela događa unutar ugrađene funkcije fit.

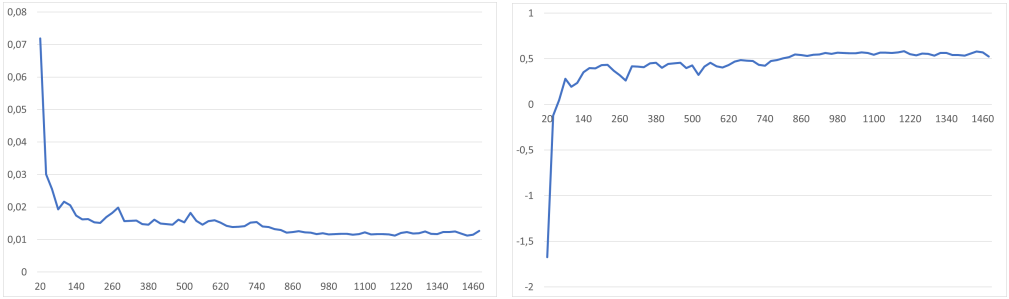
```
def train(self, file):
    with open(file, mode='r') as f:
        reader = csv.reader(f)
        data = []
        for row in reader:
            data.append(row)
    f.close()

    np_array = np.array(data)

    train_input = np_array[:,:-2]
    train_output = np_array[:, -1]

    self.model.fit(train_input, train_output)
    return
```

Proces učenja kroz etape možemo prikazati grafovima na slici 4.3. Lijevi graf na slici 4.3 je dobiven na način da se računa srednišnja kvadratna pogreška između predviđenih vrijednosti koje je model izbacio i stvarnih *xG* vrijednosti iz baze. Ekvivalentno postupku u poglavlju 4.1., svaka etapa iterativno uzima dodatnih 20 udaraca i nad njima generira stabla te ažurira parametre modela. Model nakon svake od etapa testiramo i računamo preciznost, desni graf na slici 4.3 prikazuje napredak preciznosti modela kroz vrijeme. Na horizontalnoj osi je prikazan broj udaraca, a na vertikalnoj osi vrijednost srednje kvadratne pogreške za zadanu populaciju. Prvo je provedeno učenje na modelu bez ograničenja dubine.




**Slika 4.3:** Graf srednje kvadratne pogreške(lijevo) i *score* funkcije(desno) za *max_depth*=0

Na slici 4.4 prikazani su grafovi koji prikazuju napredak učenja kroz etape na modelu koji ima ograničenje dubine na 6.




**Slika 4.4:** Graf srednje kvadratne pogreške(lijevo) i *score* funkcije(desno) za *max_depth*=6

Priloženi grafovi zorno prikazuju bolju aproksimaciju modelom s ograničenom dubinom stabla, za razliku od onoga koji nema ograničenu dubinu. Ukratko, model s ograničenom dubinom rezultira generalno boljim stablima jer su stabla odluke sklona prenaučenosti, što znači da se previše prilagođavaju skupu za treniranje, dok daju loše rezultate na skupu na testiranje. Postavljanjem određene maksimalne dubine stabla ograničavamo rast stabla prije dosezanja savršene klasifikacije na skupu primjera za učenje, čime smanjujemo vjerojatnost za prenaučenost.

# 5. Rezultati

## 5.1. Logistička regresija

Rezultati koje smo dobili primjenom modela logističke regresije postigao je relativno dobre rezultate. Uz parametar *score*, zanimljivo je pogledati kolika je prosječna pogreška modela na testom uzorku. Prosječna pogreška modela na testnom uzorku iznosi 0.074172, što izraženo u postotku iznosi 7.4172%. *Score* modela iznosi 0.863327, što izraženo u postotku iznosi 86.3327%. Uspjeli smo postići da model s vjerojatnošću preko 86% predviđa hoće li upućeni udarac završiti u golu ili ne.

## 5.2. Slučajne šume

Rezultati koje smo dobili primjenom modela slučajnih šuma postigao je razmjerno bolje rezultate od prethodnog modela. Parametar prosješne pogreške modela na testnom uzorku iznosi 0.058517, što izraženo u postotku iznosi 5.8517%. *Score* modela iznosi 0.689828, izraženo u postotku 68.9828%.

## 5.3. Usporedba

Rezultate je nezahvalno uspoređivati zbog različitog načina računanja *score* funkcije. Mogli bismo pomisliti da je model logističke regresije bolji, no uslijed drugačijeg načina računanja *score* funkcije, prikladnije je uspoređivati koliko se predviđene vrijednosti razlikuju od onih stvarnih unutar baze podataka. Uspoređivanjem tih vrijednosti, vidimo da je model slučajnih šuma razmjerno uspješniji i pouzdaniji u procjeni *xG* vrijednosti.

| | Logistčka regresija | Slučajne šume | |
|---|---|---|---|
| | | max_depth=0 | max_depth=6 |
| score | 0.863 | 0.257 | 0.694 |
| srednja kvadratna pogreška | 0.014 | 0.020 | 0.008 |
| stvarna suma xG | 304.945 | 304.945 | 304.945 |
| predviđena suma xG | 333.922 | 357.674 | 337.679 |

**Tablica 5.1:** Prikaz uspješnosti modela

# 6. Zaključak

U sklopu rada implementirana su dva modela za predviđanje vjerojatnosti za zgoditkom na nogometnim utakmicama. Za potrebe ovog rada korištena su pristupi linerne regresije i slučajnih šuma. Nakon detaljnog opisa rada, slijedi opis pojedinog pristupa na teorijskoj razini te načini previđanja, učenja i optimizacije pojedinog pristupa. Za potrebe treniranja i testiranja korištena je *Statsbomb open-data*.

Rezultati koje smo dobili po završetku projekta u skladu su s očekivanjima, čak i bolja. Od dva modela boljim se pokazao model slučajnih šuma. Najizazovnije u ovom projektu bilo je odabrati dobar model, koji će biti podoban za ovakav problem, te nakon toga pronaći najprikladnije parametre koji definiraju sam model. Najmanji prosječnu apsolutnu pogrešku postigli smo modelom slučajnih šuma s inačicom ograničene dubine generiranog stabla na 6.

Sami modeli koriste relativno jednostavne ulazne parametre, stoga ne čudi određeno odstupanje od *xG* vrijednosti koje se koriste u stvarnom životu. Uz parametre koji su korišteni unutar ovog projekta, korisno bi bilo znati pozicije suigrača, pozicije suparničkih igrača, pokrivenost vratara, detaljnije podatke o samome igraču, jer trenutne vrijednosti su procjenjene na prosjeku, a ne na stvarnom igraču u stvarnoj situaciji.

Dodatna poboljšanja temeljena na trenutnim ulaznim parametrima, bismo mogli postići naprednijim modelima za predviđanje, na primjer neuronskim mrežama.

# LITERATURA


StatsBomb: *What Are Expected Goals (xG)?*, svibanj, 2023. https://statsbomb.com/soccer-metrics/

Building Blocks: *DeepxG Tutorial Part 1: Train your own Deep Learning Model to predict Expected Goals (xG)*, studeni, 2022. https://medium.com/mlearning-ai/deep-xg-training-your-own-expected-goals-xg-deep-learning-model-cbb9b9eb5465

Building Blocks*: DeepxG Tutorial Part 2: Train your own Deep Learning Model to predict Expected Goals (xG)*, studeni, 2022. https://medium.com/mlearning-ai/deepxg-tutorial-part-2-train-your-own-deep-learning-model-to-predict-expected-goals-xg-425e4e9636bd

StatsBomb: *open-data* https://github.com/statsbomb/open-data

Jan Šnajder: *Strojno učenje - 6. Logistička regresija*, 2021. https://www.fer.unizg.hr/_download/repository/SU-2020-06-LogistickaRegresija[1].pdf

Jan Šnajder, Bojana Dalbelo Bašić: *Uvod u umjetnu inteligenciju – 10. Strojno učenje*, 2019. https://www.fer.unizg.hr/_download/repository/UI-2020-10-StrojnoUcenje.pdf

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

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

**Predikcija vjerojatnosti za zgoditkom u nogometnim utakmicama**

## Sažetak

Implementirana su dva modela previđanja vjerojatnosti za zgoditkom na nogometnim utakmicama (*Expected Goals*) s razmjerno velikom vjerojatnošću uspješnosti. Modeli su opisani pristupom logističke regresije i slučajnih šuma. Rad opisuje teorijsku podlogu svakog od pristupa, daje opis baze koja je korištena, te definira proces učenja pojedinog modela. Završno, dana su razmatranja dobivenih rezultata te detaljni potencijalni načini poboljšanja programskog rješenja.

**Ključne riječi:** logistička regresija, slučajne šume, xG, nogomet

**Prediction of scoring probability in football matches**

## Abstract

Two models for prediction od scoring probability in football matches are implemented with proportionately big accuracy. Models are described with logistic regression approach and random forests approach. Thesis defines theoretical background of each approach, gives database description, and defines training process of individual model. Finally, further considerations of acquired results and possible future algorithm improvements are given.

**Keywords:** logistic regression, random forests, expected goals, xG, football