SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Dražen Vuk

PRIMJENA METODA STROJNOG UČENJA NA PODATKE INTERNETA STVARI

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dražen Vuk

Matični broj: 0313007389

Studij: Informacijsko i programsko inženjerstvo

PRIMJENA METODA STROJNOG UČENJA NA PODATKE INTERNETA STVARI

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Darko Andročec

Varaždin, Svibanj 2020.

Dražen Vuk

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu će biti objašnjeno kako primijeniti strojno učenje XGBoost i duboko učenje Keras na podacima od parkirnog senzora, a svrha ovog rada je da uz pomoć ova dva spomenuta alata od sirovih vrijednosti senzora kao rezultat dobijemo informaciju je li je parkirno mjesto iznad senzora zauzeto ili slobodno. Koristit će se NBPS parkirni senzor tvrtke Mobilisis d.o.o. Senzor ima dva magnetometra za detekciju metala, u ovom slučaju automobila. Za ovaj rad potrebno je prikupiti sirove podatke od parkirnih senzora i to minimalno 100000 podataka za treniranje modela. U radu biti će objašnjeno kako pripremiti sirove podatke od magnetometra i normalizirati takve podatke za XGBoost i Keras, te kako pripremiti i trenirati model do najboljih mogućih rezultata. Postavljeni cilj detekcije automobila uz pomoć strojnog i dubokog učenja je minimalno 96 posto.

Ključne riječi: strojno učenje XGBoost, duboko učenje Keras, parkirni senzori, magnetometar, priprema podataka, python, scikit-learn

Sadržaj

1. Uvod

Strojno učenje je grana umjetne inteligencije koja omogućuje računalnim sustavima učiti izravno iz primjera, podataka i iz vlastitog iskustva. Da bi računalo moglo obavljati i učiti zadatke inteligentno, strojni sustavi učenja moraju obavljati složene procese učeći sami iz sirovih ili unaprijed pripremljenih podataka, a ne slijedeći unaprijed programirana pravila.

Cilj ovog radu je objasniti primjenu metoda i algoritama strojnog učenja na podacima IoT senzora. Algoritmi koji će biti korištenu i uspoređeni u ovome radu su XGBoost u svrhu dubokog učenja i Keras u svrhu strojnog učenja. Oba alata koriste paket scikit-learn u programskom jeziku Python. Kao rezultat ovog rada je uz pomoć strojnog učenja odrediti zauzeće parkirnih mjesta na parkiralištu. Svako parkirno mjesto ima jedan senzor s magnetometrom koji mjeri magnetno polje oko senzora.

U prvom poglavlju nalaze se informacije o parkirnom senzoru i pozicija senzora na parkiralištu te način slanja podataka do servera i dohvaćanje podataka. Parkirni senzor ima dva magnetometra koji mjere magnetno polje oko senzora i senzor temperature.

U drugom poglavlju biti će objašnjeni problem za koji će se koristiti strojno učenje te će također biti objašnjeni sirovi podaci dobiveni iz parkirnog senzora. Također će biti objašnjeno kako pripremiti sirove podatke iz magnetometra za strojno i duboko učenje.

U trećem govorit će se o usporedbi različitih algoritama, usporedbi dubokog učenja i strojnog učenja.

Strojno učenje je jedno od uistinu korisnih tehnologija u svijetu, gotovo nema korisnika interneta koji svakodnevno ne koristi strojno učenje. Svaki put prilikom upotrebe internetskih tražilica poput Google-a ili Bing-a, jedan od razloga dobrpg funkcioniranja je upravo algoritam učenja koji je naučio kako rangirati web stranice, a implementiran je od strane kompanija kao što su Google ili Microsoft. Facebook isto tako posjeduje aplikacije koje prepoznaju fotografije naših prijatelja, a svaki put kada čitamo našu e-poštu uvijek postoji filter neželjene e-pošte (engl. spam) koji nas oslobađa muke ručnog sortiranja velikog broja e-pošte, a to također radi nekakav algoritam učenja.

2. Metode i tehnike rada

U ovom radi korišteni su TODO

3. NBPS IoT parkirni senzor

NarrowBand parking senzor (NBPS) je autonomni, bežični kompaktni senzor za praćenje zauzetosti parkirnih mjesta što omogućava gradovima lakše upravljanje i izazovima parkiranja. Senzor koristi tehnologiju mjerenja Zemljinog magnetskog polja za otkrivanje prisutnosti vozila na parkirnom mjestu, te su instalirani ispod asfalta svakog parkirnog mjesta i kod bilo kakve jače magnetske promjene senzor šalje putem NBIoT-a mreže sve promjene na server[1]. Paket koji se šalje prema serveru sastoji se od desetak različitih podataka a sam paket je veličine 48 bajtova, podaci koji su bitni za ovaj rad su vrijednosti od oba magnetometra x,y,z i x2,y2,z2 te temperatura za svaki magnetometar. IoT NBPS senzor proizvodi tvrtka Mobilisis d.o.o., a primjer senzora je prikazan na (slici 1).

Slika 1: NBPS senzor tvrtke Mobilisis d.o.o (Izvor: **www.mobilisis.hr**, **2020**)

Trenutni broj instaliranih senzora je više od 3000 komada diljem svijeta te se svi podaci iz senzora skupljaju na serverima. Trenutno ima više od 50 GB podataka na serverima koji su bili korišteni za detekciju automobila. Glavna prednost NBPS senzora je 3-osni magnetometar i sofisticirani algoritam koji osigurava da snijeg, prljavština ili lišće što pokrivaju senzor ne utječu na pouzdanu detekciju. Informacije o zauzetosti parkirnih mjesta mogu se koristiti za navođenje vozača do slobodnog parkirnog mjesta, za povećanje protočnosti prometa i smanjenje onečišćenja prouzročenog prometom. Senzor automatski mijenja status parkirnog mjesta putem NB-IoT mreže na Web platformi (Smart parking Cloud). Trenutna detekcija se nalazi na samome senzoru i dodatna provjera detekcije još se jednom odrađuje na serveru koja se bazira na povijesti svih parkiranja za određeni senzor te na taj način algoritam zaključuje je li mjesto slobodno ili zauzeto. Takvim načinom rada detekcija NBPS senzora je visokih 97 posto točnosti, ovim radom i uz pomoć strojnog i dubokog učenja trebalo bi dokazati da je detekciju automobila možda moguće i povećati.

3.1. Najveći izazov kod detekcije automobila magnetometrom

Najveći izazov kod detekcije automobila magnetometrom se odnosi na kalibraciju magnetometra i problem takozvanoga drifranja magnetskih vrijednosti kod promjene temperature i okoline, kalibracija je postojana samo na temperaturama na kojima je kalibracija odrađena, ali kod promijene temperature magnetometar iz još točno neutvrđenih razloga mijenja svoje mag-

netsko očitavanje i to za više od 150 miligausa, što je jednako promijeni kada je automobil iznad senzora. Također, još je bitno spomenuti da je kod svakog senzora ova promjena očitanja magnetskih vrijednosti u odnosu na temperaturu, drugačija. Temperatura okoline kod magnetskog senzora često prelazi granice od -20 pa sve do 70 stupnjeva celzijusa te na taj način utječe na mjerenje vrijednosti magnetometra. Također bitno je napomenuti da nijedna dva senzora nemaju ista mjerenja niti slične vrijednosti pošto je magnetomerat osjetljiv na rotaciju i poziciju samog senzora. Toj razlici u mjerenju magnetometra, također, pridonose i različite metalne instalacije koje su u blizini senzora, kao na primjer šahtovi, metalne ograde, susjedni automobili koji ne parkiraju prema propisu, nego parkiraju preko crte, metalna prašina koja pada iz automobila, strujni vodovi, transformatori i sve što može utjecati na magnetsku promjenu. Ovaj senzor ima dva magnetometra s dobrim razlogom, ideja kod razvoja senzora bila je da ako postoje dva magnetometra na istom senzoru, to jest na istoj PCB pločici da bi oni trebali imati iste vrijednosti ako automobil nije iznad senzora, a kad automobil dođe iznad senzora svaki magnetometar bi trebao pokazivati drugačije vrijednosti pošto su ta dva magnetometra odmaknuta 15 centimetara jedan od drugog, tu bi detekcija bila očita i trebalo bi samo izračunati deltu između ta dva magnetometra. I to pravilo uistinu se pokazalo dobro na testnom polju, ali na žalost u produkciju varijanta s dva senzora nije dala očekivane rezultate, iako su magnetometri bili kalibrirani nakon par mjeseci magnetometar je promijenio svoja magnetska očitanja u odnosu kakve je mjerio kod proizvodnje i ugradnje na parkirno mjesto. Razlog tome je temperatura koja utječe na vrijednosti magnetometra i drift magnetometra koji se ne može spriječiti. TODO stavi sliku PCB-a ili skicu

3.2. Magnetometar

Magnetometar je uređaj koji mjeri magnetizam, primjer je mjerenje magnetiziranja magnetskog materijala (poput feromagneta). Magnetometri rade na principu mjerenja gustoće magnetskog toka i količine magnetskog toka preko određenog područja. Kompas je jedan takav uređaj koji mjeri smjer magnetskog polja okoline ili zemljino magnetsko polje. Magnetizam varira od mjesta do mjesta zbog razlika u zemljinom magnetskom polju.

Postoje dvije osnovne vrste magnetometra, a to su vektorski magnetometri i apsolutni magnetometri.

Vektorski magnetometri mjere vektorske komponente magnetskog polja i magnitudu vektorskog magnetskog polja. Magnetometri koji se koriste za proučavanje Zemljinog magnetskog polja mogu izraziti vektorske komponente polja u smislu deklinacije (kut između horizontalne komponente vektora polja i magnetskog sjevera) i nagiba (kuta između vektora polja i vodoravna površina). Mana vektorskih magnetometra je ta što su podložni temperaturnom odstupanju i dimenzionalnoj nestabilnosti feritnih jezgara. Oni zahtijevaju niveliranje tj., kalibraciju kako bi dobili informacije o komponentama, za razliku od ukupnih polja (skalarnih) instrumenata.

Apsolutni magnetometri mjere apsolutnu magnitudu ili vektorsko magnetsko polje koristeći unutarnju kalibraciju ili poznate fizičke konstante magnetskog senzora. [6] Relativni mag-

netometri mjere magnitude ili vektorsko magnetsko polje u odnosu na fiksnu, ali ne kalibriranu osnovnu liniju. Također se zove variometers, relativni magnetometri se koriste za mjerenje varijacija u magnetskom polju.

3.2.1. Korištenje magnetometra u svakodnevnom životu

U svakodnevnom životu često susrećemo magnetometre, iako toga nismo ni svjesni. Prvi magnetometar stvorio je Carl Friedrich Gauss, kojeg često nazivaju "princom matematike", i koji je 1833. objavio članak u kojem opisuje novi uređaj koji je nazvao "magnometar" u obliku detektora metala. Magnetometar se nalazi gotovo u svakom mobilnom uređaju, navigaciji, automobilu, detektoru metala u industriji, pronalaženje metalnih predmeta, kompasu, kućnom usisivaču itd.

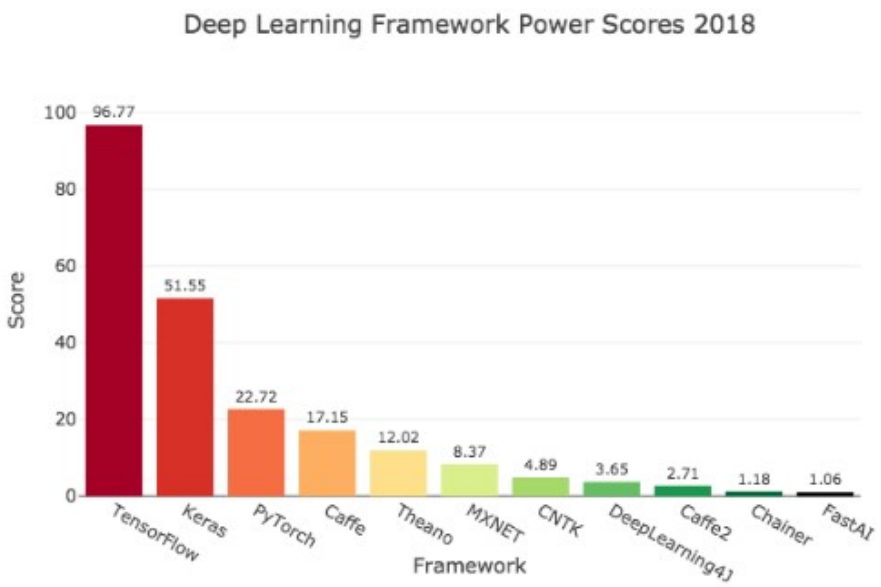
U ovom radu će se koristi vektorski magnetometar oznake TODO.

4. Neuronske mreže Keras

Keras je alat za neuronske mreže koji je napisan u Python-u i može se pokretati preko TensorFlow-a , CNTK-a ili Theano-a . Razvijen je s naglaskom na omogućavanje brzog eksperimentiranja, ali također i za sposobnost da se krene od ideje pa do finalnog proizvoda. Keras omogućuje lako i brzo prototipiranje, modularnost i proširivost, podržava i konvolucijske mreže i ponavljajuće mreže, kao i kombinacije dviju mreža. Mana kod Keras-a je ta da radi samo s Pythonom 2.7-3.6, tako da uvijek treba paziti na verziju Pythona [2].

4.1. Prednosti Keras-a

Keras ima široko prihvaćanje u industriji i istraživačkoj zajednici te ima više od 250 000 korisnika sredinom 2018. godine. Keras je odlično prihvaćen kako u industriji tako i u istraživačkoj zajednici.



Slika 2: Rangiranje deep learning-a na temelju 11 izvora podataka u 7 kategorija (Izvor: **https://keras.io/why-use-keras/**, **2020**)

4.1.1. Keras treniranje modela i postavljanje slojeva

Modeli u Keras-u se definiraju kao slijed slojeva, prvi ili ulazni sloj mora imati iste dimenzije kao i pripremljeni podaci a to se određuje prilikom stvaranja prvog sloja s argumentom "input-dim". Pronalaženje najbolje mrežne arhitekture (broj slojeva, veličina slojeva, aktivacijske funkcije) uglavnom se provodi putem pokušaja i pogreške ali također postoje i alati kao što je "GridSearchCV" koji sugeriraju kako namjestiti razne parametre u svrhu bolje predikcije. Potpuno povezani slojevi definirani su pomoću sloja Dense a u nastavku biti će detaljnije objašnjeni Keras slojevi.

Osnovni koraci za korištenje Keras dubokog učenja: [3].

- Priprema i učitavanje podataka. Ovaj korak samo izgleda jednostavan ali zapravo nije jer treba pripremiti veliki skup "točnih" podataka, obično čovjek mora prolaziti kroz takav skup i detaljno svaki podatak analizirati.
- Definiranje Kerasov-og modela, postavljanje slojeva. Često je najbolja mrežna struktura pronađena postupkom eksperimentiranja i pokušaja pogrešaka. TODO ovaj dio doraditi
- Izgradnja Kerasov-og modela, tu se koriste efikasne numeričke biblioteke kao što su Theano ili TensorFlow. Pomoćni program automatski odabire najbolji način za predstavljanje mreže za obuku i predviđanja koja će se izvržavati na nekom računalu. Za izgradnju modela koristi se funkcija "model.compile(...)".
- Proces obuke Keras modela, izvodi se fiksni broj ponavljanja kroz skup podataka koji se nazivaju "epohe", a trebaju biti specificirane kroz argument funkcije "model.fit", tu je također bitno postaviti broj redaka skupa podataka koji se razmatraju prije ažuriranja težina modela unutar svake epohe, a postavlja se pomoću argumenta batch-size. Ako je ovo finalni model, nakon ovog koraka on se može spremiti u datoteku putem Keras API-ja. Jednom kada postoji spremljen model on se može u bilo kojem trenutku učitati i koristiti za predikciju.
- Procijena Kerasov-og model-a, funkcija se poziva: model.evaluate(X, y), ova funkcija daje informacije koliko je dobro modelirali skup podataka nad kojim je trenirani model, iako ovaj algoritam ne može znati kakav će rezultati imati neki novi podaci. Funkcija model.evaluate() vraća popis od dvije vrijednosti, prvi je gubitak modela na skupu podataka, a drugi je točnost modela na danom skupu podataka.
- Predikcija, ovo je finalni korak i rezultat predikcije. Za korištenje predikcije koristi se funkcija "model.predict(X)" a izlazni rezultat može ovisiti o aktivacijskoj funkciji koja je definirana na izlaznom sloju. Pa tako na primjer ako se koristi sigmoid ili softmax vrijednost će biti između 0 i 1.

U nastavku biti će prikazan kratki primjer s osnovnom funkcionalnošću kako definirati Keras model, izgraditi ga te napraviti predikciju na nekom skupu podataka.

```
#pripremi podatke i spremi ih u varijablu "data".
data = []
#ocekivani rezultat spremiti u varijablu "result".
result = []
#staviti podatke u numpy.array
trainX = numpy.array(data)
#postavi Keras slojeve
model = Sequential()
model.add(Dense(10))
model.add(Dense(1))
#kompajliraj model
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
#izgradi model na temelju podataka
model.fit(trainX, numpy.array(result), epochs=10, batch_size=1, verbose=2)
#napravi predikciju
value = model.predict(numpy.array(testX))
```

4.1.1.1. Odabir i postavljanje slojeva za Keras model

Odabir i broj slojeva za Keras neuronsku mrežu ne može se analitički izračunati te ne postoji jednostavan prijedlog koje slojeve odabrati i koristit za neki skup podataka. U osnovi treba krenuti s načinom pokušaji i pogreške. Potrebno je isprobavanje različitih kombinacija parametara i zadržavanje one s najmanjom vrijednošću gubitka ili boljom preciznošću, ovisno o problemu, trenutno na žalost, ne postoji drugi način podešavanje takvih parametara. U nastavku su opisani osnovni slojevi Keras modela te kratko objašnjeni slojevi koji će se koristiti u ovome radu.

- Sequential, ovaj sloj služi za grupiranje linearnih snopa slojeva te stvaranje modela dubokog učenja gdje se izrađuje instanca klase "Sekvencijal" te stvaraju i dodaju novi slojevi modela. Sequential model je dobar odabir za razvoj modela dubokog učenja u većini situacija, ali ima i određene nedostatke kao što je nemogućnost definiranja modela koji mogu imati više različitih ulaznih izvora, te proizvesti više izlaznih odredišta ili modela koji ponovo koriste slojeve.
- LSTM ponavljajuće neuronske mreže koje su dobar izbor za podatke koji su tipa "vremenske serije". LSTM sloj može se koristiti za modeliranje univarijantnih problema predviđanja vremenskih serija, riječ je o problemima koji se sastoje od jedne serije opažanja i potreban je model kako bi se naučio iz niza prošlih promatranja kako bi se predvidjela sljedeća vrijednost u nizu. LSTM sloj zahtijeva trodimenzionalni ulaz, a LSTM će prema zadanom proizvesti dvodimenzionalni izlaz kao interpretaciju s kraja niza. Ulazni podaci sastoje se od tri dimenzije a to su [uzorci, vremenski koraci, značajke] [4].
- Dense sloj standardni duboko povezani sloj neuronske mreže, te je ujedno najčešći i najčešće korišteni sloj. Dense sloj predstavlja umnožavanje matričnog vektora ako je veličina serije 1 te se tako dobiva m dimenzijski vektor kao izlaz. Svaki neuron prima na ulazu sve neurone u prethodnom sloju, čime je gusto povezan [5].

- Dropout sloj sprečava "prefinjen" model a funkcionira tako da slučajnim odabirom neurona zanemaruje tijekom treninga. Oni se „odbacuju“ nasumično. To znači da je njihov doprinos aktivaciji nizvodnih neurona privremeno uklonjen na prednjem prolazu i nikakvo ažuriranje težine se ne primjenjuje na neuron na stražnjem prolazu. Problem može nastati ako se odbacivanjem neurona ode predaleko te na taj način može loše utjecati na model i kasniju predikciju.
- SimpleRNN sloj je klasa neuronskih mreža koja je moćna za modeliranje podataka o sekvenci poput vremenske serije ili prirodnog jezika. Spada u istu kategoriju kao i sloj LSTM te se najčešće koristi za podatke koji su tipa "vremenske serije"[6].

4.1.2. Podržani operativni sustavi za Keras

- Na iOS-u, putem Appleovog CoreML-a (Kerasovu službenu podršku službeno pruža Apple). Evo tutorijala .
- Na Androidu, putem vremena izvršavanja TensorFlow Android. Primjer: nije Hotdog aplikacija .
- U pregledniku, putem JavaScript-a koje su ubrzavali JavaScript, kao što su Keras.js i WebDNN .
- Na Google Cloudu, putem TensorFlow-posluživanja .
- U podupiraču za Python webapp (poput aplikacije Flask) .
- Na JVM-u, preko uvoza modela DL4J, kojeg je osigurao SkyMind .
- Na Raspberry Pi.

5. XGBoost strojno učenje

XGBoost je optimizirana distribuirana knjižnica za strojno učenje te za povećanje gradijenta, dizajnirana da bude učinkovita, fleksibilna i prenosiva. XGBoost pruža paralelno povećanje stabla (poznato i kao GBDT, GBM) koje brzo i precizno rješava mnoge probleme u znanosti o podacima. XGBoost je open source opće namjene nadzirane metode učenja koja postiže najveću točnost na širokom rasponu skupova podataka u praktičnim primjenama. Prednost korištenja XGBoost je ta što je nakon što su izgrađena pojačana stabla relativno jednostavna za dobivanje rezultata važnosti za svaki atribut. Algoritam za važnost nekog ulaznog parametara daje ocjenu koja pokazuje koliko je svaka značajka bila korisna u izgradnji stabala s pojačanim odlukama unutar modela. Što se više atribut koristi za donošenje ključnih odluka s stablima odluka, to je veća njegova relativna važnost. Ova se važnost izračunava izričito za svaki atribut u skupu podataka, čime se atributi mogu rangirati i uspoređivati. Važnost se izračunava za jedno stablo odluke prema iznosu za koji svaka točka razdvajanja atributa poboljšava mjeru performansi, ponderiranoj s brojem opažanja za koje je čvor odgovoran [7].

5.1. XGBoost model

Implementacija modela podržava karakteristike scikit-learning i R implementacije, s novim dodacima poput regularizacije. Podržana su tri glavna oblika gradijentskog povećavanja: algoritam gradient boosting, stohastičko povećanje gradijenata, regulirano povećanje gradijenta. Model učenja obično se odnosi na matematičku strukturu gdje predviđanje yi izrađuje se iz ulaza xi, u stvarnoj primjeni često to bude linearni model gdje je predviđanje dano s linearnom kombinacija ponderiranih ulaznih značajki.

$$(y) \wedge i = \Sigma j \theta j x_{ij}$$

Vrijednost predviđanja može imati različita tumačenja, ovisno o zadatku, tj. regresiji ili klasifikaciji. Na primjer, može se logistički transformirati kako bi se dobila vjerojatnost pozitivnog razreda u logističkoj regresiji, a može se koristiti i kao rangiranje bodova kada se žele rangirati rezultati.

5.2. XGBoost parametri

Parametri su neutvrđeni dio koji treba naučiti iz podataka. Kod pokretanja XGBoost-a treba postaviti tri vrste parametara: opći parametri, parametri potaknuća i parametri zadataka.

Opći parametri odnose se na to koji pojačivač koristimo za pojačavanje, obično stablo ili linearni model.

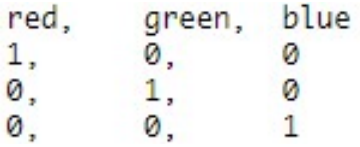
Parametri potaknuća ovise o tome koji ste pojačivač odabrali.

Parametri zadataka učenja odlučuju o scenariju učenja. Na primjer, regresijski zadaci mogu koristiti različite parametre sa zadacima rangiranja [8].

5.3. XGBoost ulazni podaci

XGBoost algoritam za ulazne podatke podržava samo numeričke vrijednosti, iako postoji riješenje kako koristiti XGBoost ako skup ulaznih podataka sadrži tekst, objekte ili binarne vrijednosti. Jednostavna metoda za pretvaranje objekata ili teksta u numerički vektor je "One Hot Encoding". Ovaj izraz potječe iz jezika digitalnog kruga, što znači niz binarnih signala. U ovom se slučaju na cijeli broj može primijeniti jednosmjerno kodiranje na način da se uklanja cjelovita kodirana varijabla i dodaje se nova binarna varijabla za svaku jedinstvenu cjelobrojnu vrijednost.

Kao primjer za One Hot Encoding u nastavku će biti prikazana ulazna varijabla "boja". Ako postoje 3 kategorije ili boje tada su potrebne 3 binarne varijable. Vrijednost "1" stavlja se u binarnu varijablu za boju, a "0" za ostale boje. Ovaj korak prikazan je na (slici 3), a u osnovi će napraviti matricu koristeći zastavice za svaku moguću vrijednost te varijable. Sparse Matrix je matrica u kojoj je većina vrijednosti nula. Suprotno tome, gusta matrica je matrica u kojoj su većina vrijednosti ne-nule [9].



Slika 3: Sparse Matrix kao ulazni podatak za XGBoost (Izvor: **vlastiti podaci**, **2020**)

6. Razrada teme

Cilj ovog projekta je napraviti algoritam i izgraditi model koji će znati za svaki senzor je li iznad senzora automobil ili nije. TODO opiši bolje ovaj dio

6.1. Priprema podataka za strojno učenje

Podaci u ovom rad-u su stvarni podaci parkirnih senzora koji se nalaze po cijelom svijetu. Dobiveni sirovi podaci od parkirnih senzora su vrijednosti od dva magnetometra i temperatura, sve ukupno 7 vrijednosti (x, y, z), (x2, y2, z2) i temperatura. Nad podacima će biti korištena dva alata za predikciju: XGBoost u pogledu strojnog učenja i Keras za neuronske mreže.

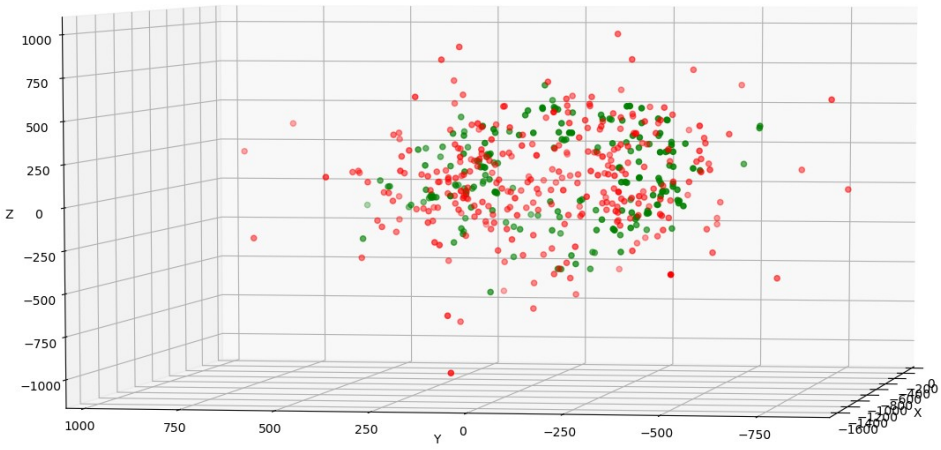
6.2. Problem sirovih podataka

Najveći izazov kod sirovih podataka je da svaki senzor dobiva potpuno drugačije vrijednosti od magnetomera u slučaju kada nema automobila iznad senzora, a taj problem se odnosi najviše na kalibraciju magnetometra i problem takozvanoga drifranja magnetskih vrijednosti kod promjene temperature, starosti senzora i okoline, kalibracija je postojana samo na temperaturama na kojima je kalibracija odrađena, ali kod promijene temperature magnetometar, iz još točno neutvrđenih razloga, mijenja svoje magnetsko očitavanje i to za više od 150 miligausa, što je jednako promijeni kada je automobili iznad senzora. Također je bitno spomenuti da je kod svakog senzora ova promjena očitanja magnetskih vrijednosti u odnosu na temperaturu, drugačija. Također problem mogu biti šahtovi, metalne ograde, susjedni automobili koji ne parkiraju prema propisu, nego parkiraju blizu ili preko crte, metalna prašina koja pada iz automobila, strujni vodovi, transformatori i sve ostalo što može utjecati na magnetsku promjenu. Bilo bi idealno da se sirovi podaci mogu pripremiti tako da svaki senzor kada nema automobila iznad njega da ima slične vrijednosti kao i svi ostali senzori, ali to je na žalost nemoguće zbog orijentacije senzora, i raznih drugih utjecaja.

6.2.1. Primjer sirovih podataka

U ovom poglavlju biti će vizualno prikazani sirovi podaci u 3D - u, korišten alat za vizualizaciju je Python Pyplot. Na (slici 4) su prikazane strane x, y, z i vrijednost magnetometra za sto različitih senzora. Crvena boja označava da je automobil iznad senzora, dok zelena boja označava da iznad senzora nema automobila.

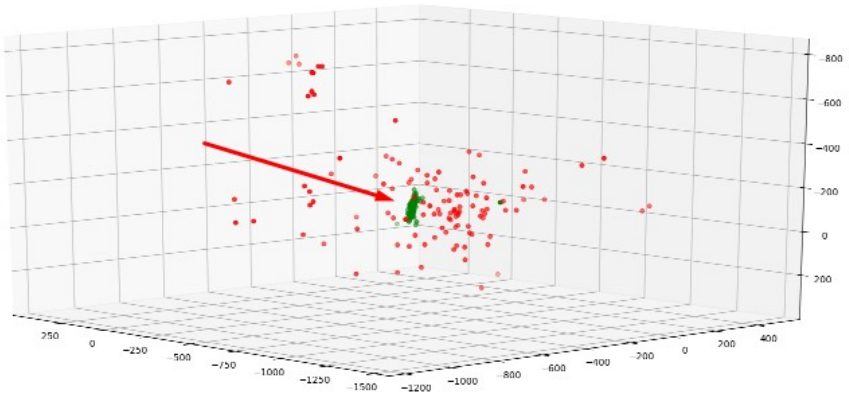
Uz pomoć vizualizacije na (slici 4) nikako se ne može raspoznati iz sirovih podataka kada se automobil nalazi iznad senzora, a kada se ne nalazi. Vizualno je vidljivo da nema nikakvih pravila niti po jednoj osi da se lako može donijeti zaključak o detekciji. Raspon od sto različitih senzora kada automobila nema iznad senzora je po x-osi 150 milligauss-a, po y osi 750 milligauss-a i po z osi 1000 milligauss-a. Kao što je već spomenuto, kada automobil



Slika 4: Sirovi podaci od 100 različitih senzora (Izvor: **vlastiti podaci**, **2020**)

parkira iznad senzora napravi vektorsku promjenu za 150 milligauss-a ako se svaka promijena zbroji vektorski x-os + y-os + z-os.

Ako se za vizualizaciju magnetskih vrijednosti uzme samo jedan senzor tada podaci iz magnetometra imaju smisla i jasno se mogu vidjeti razlike između vrijednosti kada je automobil parkiran iznad senzora ili kada nije. Na (slici 6) jasno se vidi da najviše zelenih imaju sličnu poziciju u 3D prikazu, a to znači da postoji pravilo za svaki pojedinačni senzor, ali ne postoji pravilo koje vrijedi za sve senzore. U ovome slučaju moglo bi se reći da bi klasteriranjem 3D podataka mogli odrediti dali je mjesto slobodno ili zauzeto. Klaster koji bi imao najviše vrijednosti, odnosno 3D točaka ta skupina točaka bila bi referenca, to jest nulta vrijednost koja sugerira da je parkirno mjesto slobodno, prikazano strelicom na (slici 6).

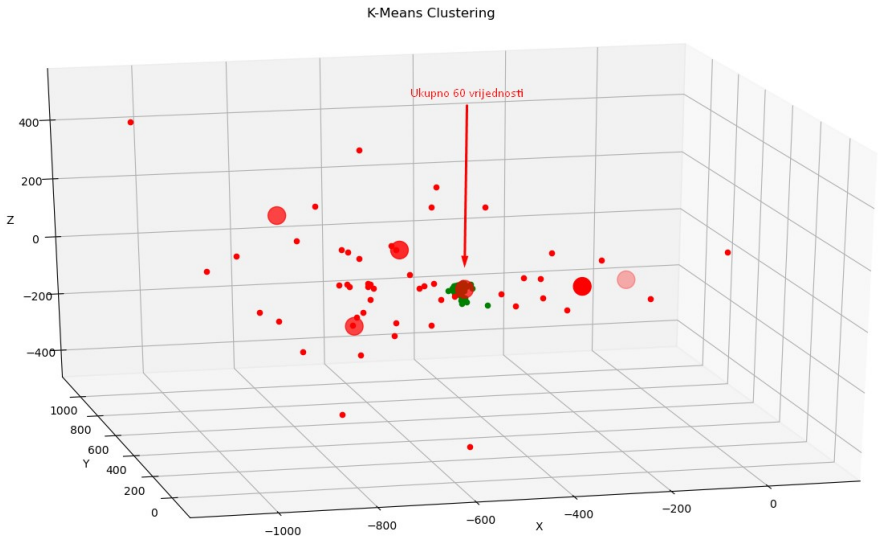


Slika 5: Sirovi podaci iz samo jednog senzora (Izvor: **vlastiti podaci**, **2020**)

6.2.2. Klasteriranje sirovih podataka

Uz pomoć alata K-Means lako se može vidjeti kako su podaci klasterirani, a klaster koji sadrži najviše podataka on je zapravo u 3D prostoru nulta vrijednost ili referenca u situaciji kada se iznad senzora ne nalazi automobil. Lako se može zaključiti da sve točke koje se nalaze u

području nultog klastera, su vrijednosti magnetometra u slučaju kada nema automobila iznad senzora, tj. parkirno mjesto je slobodno. Već ovaj algoritam s klasteriranjem bio bi dovoljan algoritam da ne ne postoji drift magnetskih vrijednosti na koje utječe temperatura, susjedni automobil i podzemni strujni kablovi.



Slika 6: Klasterirani podaci iz jednog senzora, klasteri su označeni s velikim crvenim krugom (Izvor: **vlastiti podaci**, **2020**)

6.3. Korištenje strojnog učenja za detekciju automobila

Za detekciju automobila će se koristiti "Keras" duboko učenje i "Xgboost" za strojno učenje, za oba algoritma će biti detaljno objašnjen, koji su ulazni podaci, kako su pripremljeni ulazni podaci, te koje kombinacije daju najbolje rezultate.

6.3.1. Podaci za strojno učenje

TODO

6.3.2. Pozicija senzora

TODO

6.3.3. Normalizacija podataka za strojno učenje

Normalizacija je tehnika koja se često primjenjuje kao dio pripreme podataka za strojno učenje. Cilj normalizacije je promjena vrijednosti numeričkih stupaca u skupu podataka na uobičajenu ljestvicu, bez narušavanja razlika u vrijednostima. Za strojno učenje svaki skup podataka ne zahtijeva normalizaciju, ali ako značajke imaju različite domete bilo bi dobro koristiti normalizaciju. Na primjer, ako postoji skup podataka koji sadrži dvije značajke, kao što su godine zaposlenika (x1) i plaća zaposlenika (x2), tu se jasno vidi da se godine kreću od 18g do 70g, dok se plaća kreće od 0kn do 20000kn i više. Plaća je oko 1.000 puta veća od godina. Dakle, ove dvije značajke su u vrlo različitim rasponima, te kada se radi daljnja analizu, poput multivarijantne linearne regresije, plaća će više utjecati na rezultat zbog veće vrijednosti. Ali to ne mora nužno značiti da je i važnija kao prediktor. U nastavku biti će prikazani primjer da li uistinu utječe normalizacija za XGBoost strojno učenje i Keras duboko učenje, te rezultat točnosti predikcije. https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029

6.3.3.1. Normalizacija podataka iz parkirnog senzora

U nastavku će biti prikazani primjeri kako normalizirati vrijednosti iz NBPS parkirnog senzora. Vrijednosti za normalizaciju će biti vrijednosti iz dva magnetometra $\{$x1,y1, z1$\}$ i $\{$x2, y2, z2$\}$, temperatura, magnituda izračunata oba senzora i razlika vektora od magnetometra M1 i M2. U nastavku na (slici 7) je prikazana normalizacija korištenjem algoritma "MinMax". MinMax normalizacija je normalizacijska strategija koja linearno transformira x u y = (x-min) / (max-min), gdje su min i max minimalne i maksimalne vrijednosti u X. Kada je x = min, onda je y = 0, a kada je x = max, onda je y = 1. To znači, da je minimalna vrijednost u X preslikana na 0, a maksimalna u X na 1. Dakle, cijeli raspon vrijednosti X od min do max mapiran je u raspon od 0 do 1.

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Slika 7: Formula za normalizaciju ulaznih podataka "Min-Max" (Izvor: **vlastiti podaci**, **2020**)

X1	Y1	Z1	X2	Y2	Z2	Magnituda M1	Razlika vektora
-507	-196	-128	-615	-177	-127	558,43	88
-474	191	130	-452	119	37	527,316	143
-550	-149	-79	-539	-211	-8	575,27	-20
-449	-182	-286	-477	-172	-294	562,60	26
-712	42	351	-833	95	421	794,92	-2

Tablica 1: Prikazuje sirove podatke podatke iz senzora.

U nastavku su prikazani sirovi podaci iz parkirnog senzora. U tablici 1 prikazuju se ne normalizirani podaci, dok tablica 2 prikazuje normalizirane podatke.

6.3.4. Detekcija automobila korištenjem XGBoost algoritma

Za detekciju automobila korištenjem XGBoost strojnog učenja pripremljeno je 100000 podataka, od tih podataka prvih 90000 podataka će biti rezervirani za izradu modela dok drugi dio od 10000 biti će namijenjen za testiranje točnosti strojnog učenja. Ulazni parametri za strojno učenje biti će vrijednosti iz prvog magnetometra (X, Y, Z), vrijednosti iz drugog magnetometra (x, y, z), temperatura, magnituda iz prvog magnetometra, razlika vektora između prvog i drugog magnetometra i zbroj vektora prvog magnetometra. Svaki ulazni parametar i njegova važnost u predikciji biti će prikazan na grafu.

6.3.4.1. Važnost značajki ulaznih parametara i kako ih odabrati

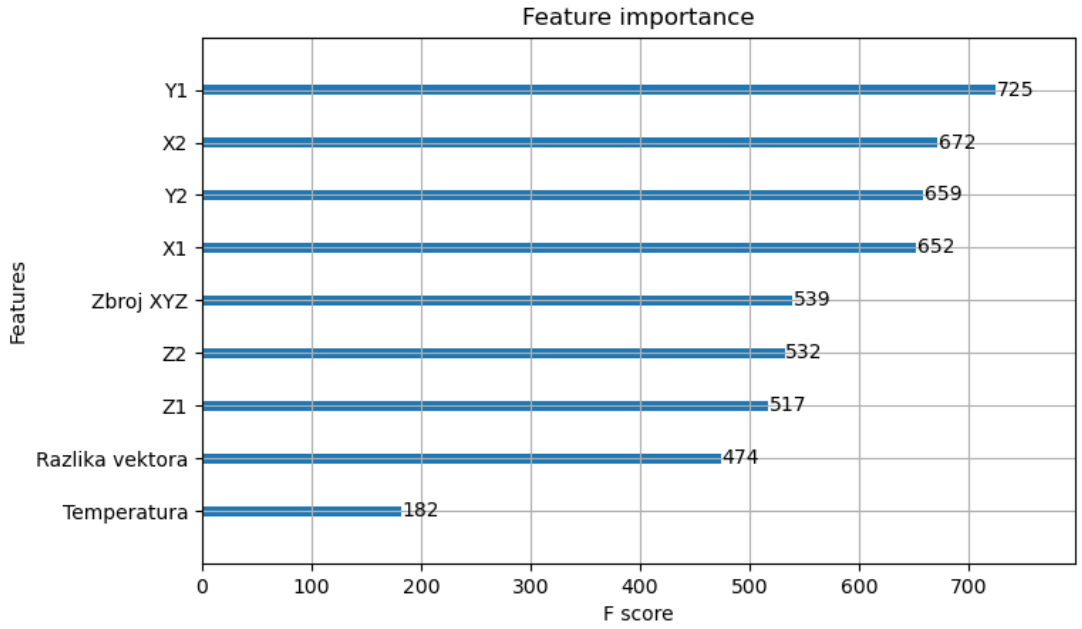
Prednost korištenja ove metode odnosi se na povećavanje gradijenta to jest predikciju strojnog učenja. U nastavku je prikazana tablica 3 s važnostima značajki za problem predik-

U tablici 2 prikazani su podaci iz parkirnog senzora koji su normalizirani u rasponu od 0 - 1 ili jasnije definirano to su podaci jednaki kao i u tablici 1 samo u drugačijem rasponu.

X1	Y1	Z1	X2	Y2	Z2	Magnituda M1	Razlika vektora
0,779	0	0,248	0,572	0,103	0,233	0,116	0,662
0	0,614	1	0	0,927	1	1	0,110
0,904	1	0,653	1	1	0,462	0	1
0,615	0,121	0,324	0,771	0	0,4	0,179	0
1	0,036	0	0,934	0,118	0	0,131	0,285

Tablica 2: Prikazuje normalizirane podatke iz senzora. Raspon vrijednosti 0 - 1

tivnog modeliranja pomoću knjižnice "sklearn.feature-selection" u Pythonu. U osnovi ovaj dio testira model uklanjajući značajke po njihovoj važnosti, ako značajka, to jest ulazni podatak nema utjecaj na predikciju ili još gore ako ima negativan utjecaj na predikciju tada se takva značajka treba ukloniti, a ovaj način omogućuje jednostavno uklanjanje značajki bez upotrebe pokušaja pogreške. Također, ovaj pristup kasnije će se primijeniti i na druge parametre kod treniranja modela kao šro su postavke za "learning-rate", "max-depth" itd. Također na (slici 6) prikazan je grafikon s važnostima za svaki ulazni podatak.



Slika 8: Važnost ulaznih parametara za XGBoost strojno učenje (Izvor: **vlastiti podaci**, **2020**)

Granica (eng. Threshold)	Pozicija	Točnost u postocima
0.042	9	87.86
0.047	8	87.93
0.059	7	87.61
0.060	6	86.94
0.062	5	87.16
0.069	4	85.47
0.071	3	83.98
0.078	2	81.46
0.512	1	79.97

Tablica 3: Tablica prikazuje važnost značajki ulaznih parametara za treniranje modela kod strojnog učenja XGBoost.

U nastavku je funkcija i kod koji je korišten za izračun važnost značajki ulaznih parametara. Prije korištenja ove funkcije treba izraditi XGBoost model i pripremiti ulazne podatke.

```
def provjeri_vaznost_znacajki(model):
    thresholds = sort(model.feature_importances_)
```

```
    for thresh in thresholds:
        # odaberi znacajke koristenjem granica
        selection = SelectFromModel(model, threshold=thresh, prefit=True)
        select_X_train = selection.transform(ULAZNI_PODACI_ZA_MODEL)
        # treniraj model
        selection_model = XGBClassifier()
        selection_model.fit(select_X_train, ULAZNI_PODACI_ZA_MODEL_DETEKCIJA)
        select_X_test = selection.transform(PODACI_ZA_TESTIRANJE)
        #odradi predikciju
        y_pred = selection_model.predict(select_X_test)
        predictions = [round(value) for value in y_pred]
        accuracy = accuracy_score(PODACI_ZA_TESTIRANJE_DETEKCIJA, predictions)
        print("Thresh=%.3f, n=%d, Accuracy: %.2f%%" % (thresh, select_X_train.shape
            [1], accuracy*100.0))
```

6.3.4.2. Treniranje XGBoost modela

Za treniranje XGBoost modela korišteno je 90000 podataka, tj. 90000 parkiranja. U nastavku biti će objašnjeno kako podesiti broj i dubinu stabala odlučivanja kada se koristi gradijentno povećanje s XGBoostom u Python-u. Za traženje parametara koji daju najbolji rezultat korištena je klasa GridSearchCV, acijeli dio koda za traženje najboljih parametara je u nastavku. Traženjem parametara koji daju najbolji rezultat točnost se povećala s 86.86 posto na 89.92 posto, dakle dobiven rezultat je povećan za 3.06 posto.

```
def podesi_broj_stabala_i_najvecu_dubinu():
    model = XGBClassifier()
    n_estimators = [50, 100, 150, 200]
    max_depth = [2, 4, 6, 8]
    print(max_depth)
    param_grid = dict(max_depth=max_depth, n_estimators=n_estimators)
    kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
    grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1,
         cv=kfold, verbose=1)
    grid_result = grid_search.fit(np.array(ULAZNI_PODACI_ZA_MODEL), np.array(
        ULAZNI_PODACI_ZA_MODEL_DETEKCIJA))
    # summarize results
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    params = grid_result.cv_results_['params']
    for mean, stdev, param in zip(means, stds, params):
        print("%f (%f) with: %r" % (mean, stdev, param))
    # plot results
    scores = numpy.array(means).reshape(len(max_depth), len(n_estimators))
    for i, value in enumerate(max_depth):
        pyplot.plot(n_estimators, scores[i], label='depth: ' + str(value))
    pyplot.legend()
    pyplot.xlabel('n_estimators')
    pyplot.ylabel('Log Loss')
    pyplot.savefig('n_estimators_vs_max_depth.png')
```

Pokretanjem koda za pronalaženje najboljih parametara dobivaju se izlazni podaci koji su prikazani u tablici 4. Prema toj tablici jasno se vidi da su najbolji parametri u zadnjoj koloni (-0.217778, max-depth = 8, n-estimators=150), dok zadani ili defaultni podaci u XGBClassifier su max-depth = 4, n-estimators=100, kao što je već spomenuto ovom promjenom točnost je poboljšana za 3.06 posto.

Means	Stds	Parametri
-0.395251	(0.005498)	('max-depth': 2, 'n-estimators': 50)
-0.378443	(0.005259)	('max-depth': 2, 'n-estimators': 100)
-0.368585	(0.005491)	('max-depth': 2, 'n-estimators': 150)
-0.361856	(0.005498)	('max-depth': 2, 'n-estimators': 200)
-0.342592	(0.004927)	('max-depth': 4, 'n-estimators': 50)
-0.317545	(0.005525)	('max-depth': 4, 'n-estimators': 100)
-0.301065	(0.005359)	('max-depth': 4, 'n-estimators': 150)
-0.289758	(0.005015)	('max-depth': 4, 'n-estimators': 200)
-0.294746	(0.005770)	('max-depth': 6, 'n-estimators': 50)
-0.263726	(0.006574)	('max-depth': 6, 'n-estimators': 100)
-0.248994	(0.007177)	('max-depth': 6, 'n-estimators': 150)
-0.251579	(0.005276)	('max-depth': 8, 'n-estimators': 50)
-0.227026	(0.006441)	('max-depth': 8, 'n-estimators': 100)
-0.217778	0.006663)	('max-depth': 8, 'n-estimators': 150)

Tablica 4: Tablica prikazuje najbolju kombinaciju postavka za model kod strojnog učenja XGBoost.

6.3.5. Konačan rezultat korištenja XGBoost strojnog učenja

Uz namještanje XGBClassifier parametara i treniranje modela s 90000 postignut je rezultat od 89,92 posto. Bitno je napomenuti da su podaci s kojima je građen model različiti od podataka s kojima je testirana predikcija. U nastavku su bitniji dijelovi kod-a s kojim je dobivena navedena detekcija automobila od 89.92 posto. Normaliziranje ulaznih podataka nije pomoglo strojnom učenju u predikciji i točnosti detekcije automobila. Ako se trenira model sa samo jednim senzorom tada strojno učenje daje puno bolje rezultate. Točnost se povećava na čak 97.83 posto što je zadovoljavajuća točnost ali s druge strane gotovo je nemoguće trenirati model za svaki pojedinačan senzor, cilj ovog rada je pronaći rješenje koje će dati točnost predikcije za najmanje 98 posto.

6.3.6. Detekcija automobila korištenjem Keras algoritma

Za detekciju automobila korištenjem Keras dubokog učenja pripremljeno je 100000 podataka, od tih podataka prvih 90000 podataka će biti rezervirani za izradu modela, dok drugi dio od 10000 će biti namijenjen za testiranje točnosti.

6.3.6.1. Treniranje Keras modela

Za treniranje Keras modela koristiti će se sirovi podaci iz parkirnog senzora. Dobivene vrijednosti će biti: Magnetometar1 $\{$x1,y1, z1$\}$, Magnetometar2 $\{$x2, y2, z2$\}$, temperatura, magnituda i razlika vektora od magnetometra1 i magnetometra2.

6.3.6.2. Podešavanje parametara za duboko učenje Keras

Za podešavanje parametara korišten je alat "GridSearchCV". GridSearchCV radi na principu da "brute force"-a sve kombinacije za određeni skup podataka i modela[10]. Za najbolje i najoptimalnije vrijednosti GridSearchCV implementira funkcije fit, score, predict, predict-proba, decistion-function i transform [11].

Fino podešavanje parametara

U nastavku su opisani parametri koji služe za fino podešavanje kod dubokog učenje Keras, te kako ih najbolje podesiti da se dobije najbolji rezultat na podacima od parkirnih senzora.

- optimizers - algoritam za optimizaciju koji se koristi za obuku mreže.
- initializer - pojam za koji se statistička distribucija ili funkcija koriste za inicijaliziranje težina. U slučaju statističke distribucije, knjižnica će generirati brojeve iz te statističke raspodjele i koristiti ih kao početne težine. Opcije initializerera su: [’softmax’, ’softplus’, ’softsign’, ’relu’, ’tanh’, ’sigmoid’, ’hard-sigmoid’, ’linear’]
- epochs - koliko puta se čitav skup podataka treninga prikazuje u toku treninga. Neke su mreže osjetljive na veličinu serije, kao što su LSTM ponavljajuće neuronske mreže i Konvolucionarne neuronske mreže.
- batches - je broj obrazaca prikazanih na mreži prije ažuriranja težina. To je ujedno i optimizacija za trening mreže, definiranje broja obrazaca za čitanje odjednom i broj obrazaca u memoriji.
- activation - su matematičke jednadžbe koje određuju izlaz neuronske mreže. Funkcija je priključena na svaki neuron u mreži i određuje treba li ga aktivirati ili ne, na osnovu je li unos svakog neurona relevantan za predviđanje modela. Aktivacijske funkcije također pomažu normalizirati izlaz svakog neurona u rasponu između 1 i 0 ili između -1 i 1 [12].

Uz pomoć klase GridSearchCV za parametar "activation" najbolje dobiveni rezultat je s "softplus", prikaz predikcije u postotku je prikazan u tablici 5, u tablici se vidi da se najbolja točnost od 74,8533 dobiva s parametrom "softplus".

Za parametre batch-size, epochs, init i optimizer također je korištena klasa GridSearchCV te su za najprecizniju detekciju dobiveni sljedeći parametri: ’batch-size’: 20, ’epochs’: 100, ’init’: ’uniform’, ’optimizer’: ’rmsprop’. Za ove rezultate bilo je potrebno 19 sati da GridSearchCV usporedi sve kombinacije i prikaže koje parametre je najbolje upotrijebiti za najveću točnost. Dio rezultata je prikazan u tablici 6.

Točnost u postotku	Parametar
0.710611	’activation’: ’softmax’
0.748533	’activation’: ’softplus’
0.721011	’activation’: ’softsign’
0.742433	’activation’: ’relu’
0.700311	’activation’: ’tanh’
0.727433	’activation’: ’sigmoid’
0.727311	’activation’: ’hard-sigmoid’
0.628556	’activation’: ’linear’

Tablica 5: Prikazuje usporedbu parametara za "activation".

TODO ovo ne valja, prevelika tablica, nepotrebno

Točnost	Greška	Parametar
0.627778	0.046348	'batch': 5, 'epochs': 50, 'init': 'glorot', 'opti': 'rmsprop'
0.647778	0.036952	'batch': 5, 'epochs': 50, 'init': 'glorot', 'opti': 'adam'
0.703333	0.019436	'batch': 5, 'epochs': 50, 'init': 'normal', 'opti': 'rmsprop'
0.736667	0.011967	'batch': 5, 'epochs': 50, 'init': 'normal', 'opti': 'adam'
0.727778	0.031230	'batch': 5, 'epochs': 50, 'init': 'uniform', 'opti': 'rmsprop'
0.730000	0.019437	'batch': 5, 'epochs': 50, 'init': 'uniform', 'opti': 'adam'
0.707778	0.036952	'batch': 5, 'epochs': 100, 'init': 'glorot', 'opti': 'rmsprop'
0.667778	0.037417	'batch': 5, 'epochs': 100, 'init': 'glorot', 'opti': 'adam'
0.733333	0.024845	'batch': 5, 'epochs': 100, 'init': 'normal', 'opti': 'rmsprop'
0.734444	0.019689	'batch': 5, 'epochs': 100, 'init': 'normal', 'opti': 'adam'
0.734444	0.028415	'batch': 5, 'epochs': 100, 'init': 'uniform', 'opti': 'rmsprop'
0.753333	0.009027	'batch': 5, 'epochs': 100, 'init': 'uniform', 'opti': 'adam'
0.711111	0.046081	'batch': 5, 'epochs': 150, 'init': 'glorot', 'opti': 'rmsprop'
0.680000	0.056503	'batch': 5, 'epochs': 150, 'init': 'glorot', 'opti': 'adam'
0.752222	0.027352	'batch': 5, 'epochs': 150, 'init': 'normal', 'opti': 'rmsprop'
0.730000	0.015947	'batch': 5, 'epochs': 150, 'init': 'normal', 'opti': 'adam'
0.734444	0.023675	'batch': 5, 'epochs': 150, 'init': 'uniform', 'opti': 'rmsprop'
0.733333	0.022498	'batch': 5, 'epochs': 150, 'init': 'uniform', 'opti': 'adam'
0.741111	0.026434	'batch': 10, 'epochs': 50, 'init': 'normal', 'opti': 'rmsprop'
0.727778	0.027442	'batch': 10, 'epochs': 50, 'init': 'normal', 'opti': 'adam'
0.730000	0.023986	'batch': 10, 'epochs': 50, 'init': 'uniform', 'opti': 'rmsprop'
0.748889	0.011331	'batch': 10, 'epochs': 50, 'init': 'uniform', 'opti': 'adam'
0.702222	0.029938	'batch': 10, 'epochs': 100, 'init': 'glorot', 'opti': 'rmsprop'
0.697778	0.021257	'batch': 10, 'epochs': 100, 'init': 'glorot', 'opti': 'adam'
0.711111	0.022498	'batch': 10, 'epochs': 100, 'init': 'normal', 'opti': 'rmsprop'
0.736667	0.022388	'batch': 10, 'epochs': 100, 'init': 'normal', 'opti': 'adam'
0.753333	0.019437	'batch': 10, 'epochs': 100, 'init': 'uniform', 'opti': 'rmsprop'
0.722222	0.031427	'batch': 10, 'epochs': 100, 'init': 'uniform', 'opti': 'adam'
0.685556	0.031545	'batch': 10, 'epochs': 150, 'init': 'glorot', 'opti': 'rmsprop'
0.690000	0.028846	'batch': 10, 'epochs': 150, 'init': 'glorot', 'opti': 'adam'
0.722222	0.036004	'batch': 10, 'epochs': 150, 'init': 'normal', 'opti': 'rmsprop'
0.727778	0.034960	'batch': 10, 'epochs': 150, 'init': 'normal', 'opti': 'adam'
0.736667	0.035066	'batch': 10, 'epochs': 150, 'init': 'uniform', 'opti': 'rmsprop'
0.754444	0.021199	'batch': 10, 'epochs': 150, 'init': 'uniform', 'opti': 'adam'
0.728889	0.024191	'batch': 20, 'epochs': 50, 'init': 'normal', 'opti': 'rmsprop'
0.730000	0.009686	'batch': 20, 'epochs': 50, 'init': 'normal', 'opti': 'adam'
0.734444	0.020306	'batch': 20, 'epochs': 100, 'init': 'normal', 'opti': 'rmsprop'
0.723333	0.022055	'batch': 20, 'epochs': 100, 'init': 'normal', 'opti': 'adam'
0.715556	0.032470	'batch': 20, 'epochs': 100, 'init': 'uniform', 'opti': 'rmsprop'
0.746667	0.011440	'batch': 20, 'epochs': 100, 'init': 'uniform', 'opti': 'adam'
0.717778	0.016997	'batch': 20, 'epochs': 150, 'init': 'normal', 'opti': 'rmsprop'
0.743333	0.004157	'batch': 20, 'epochs': 150, 'init': 'normal', 'opti': 'adam'
0.724444	0.016330	'batch': 20, 'epochs': 150, 'init': 'uniform', 'opti': 'adam'

Tablica 6: Prikazuje usporedbu parametara za duboko učenje Keras.

6.3.7. Konačan rezultat korištenja Keras dubokog učenja učenja

Uz pomoć GridSearchCV alata za odabir najboljih parametara, namještanjem slojeva za duboko učenje te treniranjem modela s 90000 parkiranja postignut je rezultat od 79,56 posto. Bitno je napomenuti da su podaci s kojima je treniran model, različiti od podataka s kojima je testirana predikcija. U nastavku je konačan kod s kojim je dobivena navedena detekcija automobila od 79,56 posto. Ako se trenira model sa samo jednim senzorom tada duboko učenje Keras daje puno bolje rezultate, točnost se povećava na čak 97.83 posto što je zadovoljavajuća točnost, ali s druge strane gotovo je nemoguće trenirati model za svaki pojedinačan senzor, cilj ovog rada je pronaći rješenje koje će dati točnost predikcije za najmanje 90 posto i to za sve senzore.

U nastavku je izvorni kod napisan u Python-u koji je korišten za predikciju dali je parkirno mjesto zauzeto ili slobodno. Dijelovi koda mogu se podijeliti u tri razine: čitanje podataka iz CSV datoteke, treniranje modela i testiranje predikcije koja govori je li je parkirno mjesto zauzeto ili slobodno, svaki dio koda biti će ukratko objašnjeni u nastavku.

Funkcija za čitanje podataka "`procitaj_podatke_iz_datoteke()`" čita podatke iz CSV datoteke i sprema ih u memoriju. Podaci spremljeni u CSV datoteci su vrijednosti iz magnetometra, to jest svaka veća magnetska promjena je jedan zapis.

Funkcija za treniranje Keras modela "`pripremi_model()`" služi za treniranje modela ili čitanje postojećeg već treniranog modela s diska.

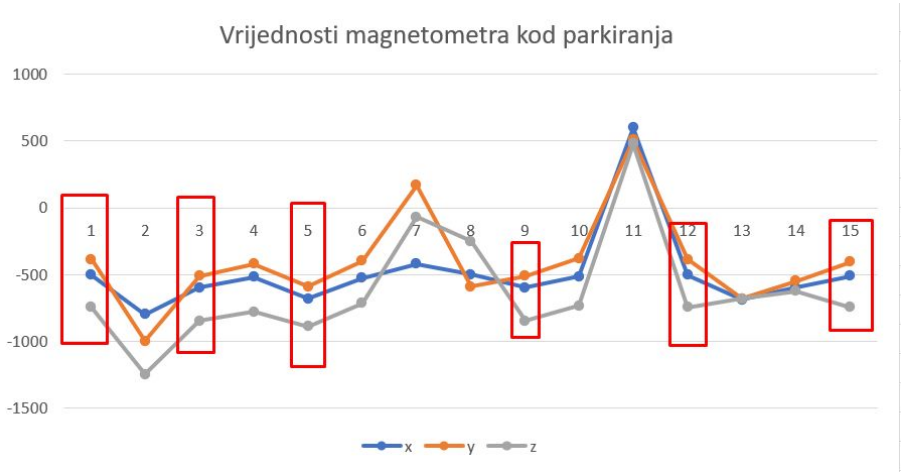
Za predikciju korišteno je duboko učenje Keras a dijelovi koda mo

```
TODO stavi ovdje bitne dijelove koda
```

7. Poboljšanje algoritma s "klasteriranjem" podataka

Za poboljšanje postojećih rezultata koji još uvijek nisu zadovoljavajući a dobiveni su korištenjem Keras i XGBoost-a trebalo bi pripremiti sirove podatke na način da vrijednosti magnetometra kada nema automobila kod svih senzora bude slična bez obzira na njihovu kalibraciju magnetometra, temperaturni drift ili orijentaciju senzora. Ideja je da se napravi algoritam koji je sličan klasteriranju podataka te se primjeni za svaki senzor pojedinačno. A to bi značilo da se svaka magnetska promjena treba čuvati u memoriji ili u bazi podataka te za svaku novu magnetsku vrijednost iz senzora treba tražiti najčešću vrijednost u bazi podataka s točno određenom temperaturom jer temperatura ima veliki učinak na vrijednosti magnetometra. Dobivena najčešća vrijednost za svaku os posebno trebala bi biti zapravo referenca kada automobila nema iznad senzora. Na (slici 9) je prikazana teorija najčešćih vrijednosti za svaki pojedini senzor te koja će u nastavku biti i dokazana.

x	y	z
-506	120	-360
-800	-200	-250
-600	90	-340
-520	100	-360
-680	90	-300
-526	129	-320
-420	590	-240
-500	-90	340
-600	90	-340
-516	135	-355
600	-90	-30
-506	120	-360
-690	10	-3
-600	50	-75
-511	105	-340



Slika 9: Najčešće vrijednosti magnetometra prikazuju da je parkirni senzor slobodan, da nema automobila iznad njega (Izvor: **vlastiti podaci**, **2020**)

Na (slici 9) je prikazan jedan idealan primjer kako izgleda parkiranje i odparkiravanje automobila iznad jednog senzora. Ovo je idealan primjer jer se temperature nisu promijenile pa su vrijednosti magnetometra kada automobila nema iznad senzora uvijek slične koje iznosi po x-osi -506, y-os 120 i z-os -360 milligauss-a dok sve vrijednosti koje su udaljene vektorski od ove točke u 3D prostoru pretpostavlja se da su to vrijednosti kada je automobil iznad senzora.

Priprema podatka na ovaj način osigurava da se sirovi podaci od svih magnetnih senzora pripreme tako da X, Y i Z imaju gotovo iste vrijednosti kada iznad senzora nema automobila. Kao što je već spomenuto na (slici 9) najčešća vrijednost magnetometra iznosi (-506, 120, -360) te ako dobivena nova vrijednost iz parkirnog senzora iznosi na primjer (-606, -64, -60) tada oduzimanjem vektora dobijemo slijedeću vrijednost (-506–606, -120–64, -360–60) => (101, -56, -300) te se na ovom primjeru vidi da X,Y i Z i dalje imaju velike vrijednosti te su daleko od referentne ili najčešće vrijednosti, tu se lako može zaključiti da je iznad ovog senzora parkirani automobil. U teoriji, ako se iz parkirnog senzora dobiju ovakve vrijednosti (-515, -135, -345) te oduzmemo referentnu ili najčešću vrijednost kao u sljedećem primjeru: (-506–515, -120–135, -360–344) => (9, 15, -16) tu se jasno vidi da su vrijednosti male, to jest oduzimanjem

referentne najčešće vrijednosti dobije s vrijednosti blizu nule što znači da iznad senzore nema automobila. Ovakvom metodom znatno će se povećati točnost i omogućiti strojnom i dubokom učenju da izgrade kvalitetniji model koji će rezultirati točnijom predikcijom.

Naravno postoje i negativne strane ovog algoritma, za svaki senzor u bazi podatake ili nekom drugom spremniku mora se čuvati povijest parkiranja da bi se mogla izračunati najčešća ili referentna vrijednost. Na primjer ako se instalira novi senzor on neće imati nikakvu povijest i biti će potrebno napraviti barem tri parkiranja da se algoritam može naučiti koje su vrijednosti najčešće te tek nakon toga moći će koristiti strojno učenje ili neuronske mreže za predikciju.

Za potrebe ovog diplomskog rada povijest podataka svake magnetne promjene od parkirnih senzora spremljena je u CSV datoteku iz koje se podaci uzimaju i izračunavaju za svaki pojedini senzor te za svako pojedino parkiranje. CSV datoteka može se skinuti na GitLab adresi: TODO. Primjer Python koda za traženje najčešćih vrijednosti može se pogledati na GitHub adresi: TODO. U nastavku su samo bitniji dijelovi Python koda za traženje referentne vrijednosti.

Funkcija "SearchByMacAndTemperature()" prvo sortira podatkre prema datumu te traži u setu podataka prijašnje magnetske promjene koje odgovaraju prema MAC-u senzora i koje odgovaraju trenutnoj temperaturi

```
"""
Funkcija vraca n podataka koji zadovaoljavaju MAC adresu senzora i priblizni su
    trazenoj temperaturi
"""
def SearchByMacAndTemperature(mac, temperature, dataset, top):
        tData = []
        #dataset2 = dataset.sort(key = lambda c: c.date)
        for d in dataset:
                if d.mac == mac and abs(temperature - d.temp) < 5 and len(tData) <
                    top:
                        tData.append(d)

        return tData
```

Funkcija SearchForMostCommonValues() traži u setu podataka slične vrijednosti na sličnim temperaturama. Funkcija "SearchForMostCommonValues()" već dobiva set podataka koji sadrži povijest samo za traženi senzor, tj podatke koje je vratila funkcija "SearchByMacAndTemperature()".

```
"""
Funkcija trazi i vraca najcesce vektorske vrijednosti iz danog dataseta
"""
def SearchForMostCommonValues(dataset, limitForSimilarValue):
        for d1 in dataset:
                similarX = 0
                similarY = 0
                similarZ = 0
                for d2 in dataset:
                        if abs(d1.x1 - d2.x1) < limitForSimilarValue:
                                similarX+=1
                        if abs(d1.y1 - d2.y1) < limitForSimilarValue:
```

```
                similarY+=1
            if abs(d1.z1 - d2.z1) < limitForSimilarValue:
                similarZ+=1
        if similarX > maxSimilarX:
            maxSimilarX = similarX
            mostCommonValueX = d1.x1
        if similarY > maxSimilarY:
            maxSimilarY = similarY
            mostCommonValueY = d1.y1
        if similarZ > maxSimilarZ:
            maxSimilarZ = similarZ
            mostCommonValueZ = d1.z
    return VectorData(mostCommonValueX, mostCommonValueY, mostCommonValueZ);
```

7.1. Treniranje Keras modela s transformiranim podacima

Za treniranje Keras modela koristiti će se transformirani podaci iz parkirnog senzora. Za transformiranje podataka koristiti će se već spomenute Python funkcije "SearchByMacAndTemperature()" i "SearchForMostCommonValues()". Dobivene vrijednosti iz tih funkcija će biti razlika vektora (X, Y, Z) za svaki pojedini podatak. Transformacija izgleda ovako: Magnetometar(X, Y, Z) minus "najčešća vrijednost"(X, Y, Z).S obzirom da su ovi podaci tipa "vremenske serije" koristiti će se sljedeće slojevi za neuronsku mrežu. Ovi slojevi i njihov broj neurona nisu odabrani slučajnim odabirom već su korištene najbolje prakse za podatke tipa "vremenske serije" i testirani su s postojećim podacima od parkirnih senzora koji se koriste u ovom radu na način da se dobiju najbolji rezultati u smislu predikcije.

- Sequential za grupiranje grupira linearnih snopa slojeva.
- LSTM ponavljajuće neuronske mreže koje su dobar izbor za podatke koji su tipa "vremenske serije". U ovom primjeru postoje dva sloja LSTM, a svaki je podešen s 50 neurona što u ovom primjeru daje najbolje rezultate.
- Dense s deset neurona
- Dropout sloj da spriječi "prefinjen" model, s ulaznim parametrom 0.3 ili 30 posto
- Dense finalni sloj s jednim neuronom bez parametra activation

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(1, 3)))
model.add(LSTM(50))
model.add(Dense(10))
model.add(Dropout(0.3))
model.add(Dense(1, activation='softplus'))
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
model.fit(trainX, numpy.array(result), epochs=10, batch_size=1, verbose=2)
```

Ovdje je prikazan samo dio kod-a za izgradnju Keras modela dok se cijeli kod nalazi na GitLab adresi: TODO:

7.1.1. Keras predikcija s transformiranim podacima

Uz pomoć alata "GridSearchCV" pronađene su postavke koje daju najbolje rezultate.Postavke koje su bile uspoređivane su optimizers = ['rmsprop', 'adam'], init = ['glorot-uniform', 'normal', 'uniform'], activation1 = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard-sigmoid', 'linear'], epochs = [50, 100, 150] i batches = [5, 10, 20]. GridSearchCV kao najbolje rezultate odabrao je:

- batch-size = 20
- epochs = 100
- init = 'uniform'
- optimizer' = 'rmsprop'

Kao i prije model je treniran s 90000 parkiranja i postignut je rezultat od 97,58 TODO još na tome radim posto. Bitno je napomenuti da su podaci s kojima je treniran model, različiti od podataka s kojima je testirana predikcija u ovom slučaju svi sirovi podaci su transformirani s već spomenutim funkcijama "SearchByMacAndTemperature()" i "SearchForMostCommonValues()". Konačan kod s kojim je dobivena navedena detekcija automobila od 97,58 posto se nalazi na GitHub adresi: TODO. Nakon transformacije podataka sada više nije važno dali se trenira model sa samo jednim senzorom ili s n senzora.

7.2. XGBoost model s transformiranim podacima

Za treniranje XGBoost modela koristiti će se isti podaci koji su bili korišteni i kod Keras-s a to su transformirani podaci iz parkirnog senzora. Za transformiranje podataka koristiti će se već spomenute Python funkcije "SearchByMacAndTemperature()" i "SearchForMostCommonValues()". Dobivene vrijednosti iz tih funkcija će biti razlika vektora (X, Y, Z) za svaki pojedini podatak. Transformacija izgleda ovako: Magnetometar(X, Y, Z) minus "najčešća vrijednost"(X, Y, Z).

7.2.1. XGBoost predikcija s transformiranim podacima

S obzirom da se radi o transformiranim ulaznim podacima koji su različiti od prethodnog primjera gdje se također koristio XGBoost ali s običnim sirovim podacima prvi je korak pronaći najbolje parametre koji će dati najbolju predikciju. Za usporedbu parametara ponovno je korištena "GridSearchCV" klasa te za usporedbu koristili su se argumenti: n-estimators = [50, 100, 150, 200], max-depth = [2, 4, 6, 8].

GridSearchCV kao najbolje rezultate odabrao je:

- max-depth = 8
- n-estimators=150

Kao i prije model je treniran s 90000 parkiranja i postignut je rezultat od 98.57 posto. Bitno je napomenuti da su podaci s kojima je treniran model, različiti od podataka s kojima je testirana predikcija u ovom slučaju svi sirovi podaci su transformirani s već spomenutim funkcijama "SearchByMacAndTemperature()" i "SearchForMostCommonValues()". Konačan kod s kojim je dobivena navedena detekcija automobila od 98.57 posto se nalazi na GitHub adresi: TODO.

8. Konkurentna riješena za detekciju automobila

TODO Konkurencija za detekciju automobila najčešće koristi ultrazvučni senzor.

9. Zaključak

TODO

Popis literature

[1] M. d.o.o, *GridSearchCV za duboko učenje i pronalaženje najboljih parametara*, Dostupno na https://www.mobilisis.hr/nbps-narrowband-parking-senzor (2020/03/18).

[2] K. Developers, *Duboko učenje Keras*, Dostupno na https://keras.io/ (2020/04/13).

[3] C. Hansen, *Osnovni koraci kod korištenja Keras dubokog učenja*, Dostupno na https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (2020/07/13).

[4] ——, *Keras LSTM sloj*, Dostupno na https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/ (2020/07/13).

[5] Keras.io, *Keras Dense sloj*, Dostupno na https://keras.io/api/layers/core_layers/dense/ (2020/06/18).

[6] Tensorflow, *Keras SimpleRNN sloj*, Dostupno na https://www.tensorflow.org/guide/keras/rnn (2020/06/18).

[7] X. Developers, *XGBoost Documentation*, Dostupno na https://xgboost.readthedocs.io/en/latest/ (2020/04/12).

[8] X. Dev, *XGBoost Parameters*, Dostupno na https://xgboost.readthedocs.io/en/latest/parameter.html (2020/04/13).

[9] J. Brownlee, *Priprema podataka za povećanje gradijenta pomoću XGBoost-a*, Dostupno na https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/ (2020/04/13).

[10] C. Hansen, *GridSearchCV za pronalaženje najboljih parametara*, Dostupno na https://scikit-learn.org/stable/modules/generated/sklearn.model-selection.GridSearchCV.html (2020/04/13).

[11] J. Hirko, *Klasifikacije i odabir značajki s XGBoostom*, Dostupno na https://www.aitimejournal.com/@jonathan.hirko/intro-to-classification-and-feature-selection-with-xgboost (2020/05/13).

[12] Missinglink, *Aktivacije neuronske mreže*, Dostupno na https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/ (2020/06/18).

Popis slika

Popis tablica

1. Prilog 1

2. Prilog 2