Detekcija objekata u WiFi polju pomoću strojnog učenja

Nikolić, Nikola

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Nikola Nikolić

DETEKCIJA OBJEKATA U WIFI POLJU POMOĆU STROJNOG UČENJA

Diplomski rad

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER NASTAVNIČKI

Nikola Nikolić

Diplomski rad

Detekcija objekata u WiFi polju pomoću strojnog učenja

Voditelj diplomskog rada: prof. dr. sc. Hrvoje Buljan

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2024.

Sažetak

U radu se istražuje mogućnost i preciznost detekcije ljudi u Wi-Fi elektromagnetskom polju s izvorom unutar prostorije u kojoj su postavljeni detektori. Detektori mjere snagu signala u vremenu te podatke prikupljene od detektora algoritmi strojnog učenja koriste za učenje i predviđanje broja ljudi u prostoriji.

Ključne riječi: Wi-Fi, elektromagnetski valovi, strojno učenje, nadzirano učenje, nenadzirano učenje, logistička regresija, višeklasna logistička regresija, analiza glavnih komponenti, neuronska mreža, slučajna šuma

Detection of objects in WiFi fields with Machine Learning

Abstract

This paper investigates the possibility and accuracy of detecting people in an electromagnetic field with a Wi-Fi source placed inside a room equiped with detectors. The detectors are measuring the power present in a received signal over time. The data gathered from the detectors is used by machine learning algorithms for model training and predicting the number of people occupying the room.

Keywords: Wi-Fi, electromagnetic waves, machine learning, supervised learning, unsupervised learning, logistic regression, multiclass logistic regression, principal component analysis, neural network, random forest

Sadržaj

1 Uvod

Bežična tehnologija u modernom svijetu čini integralni dio svakodnevnog života. Jednu od poznatijih bežičnih tehnologija čini Wi-Fi mreža čija se prisutnost može smatrati učestalom u kućama, radnim organizacijama, te javnim prostorima. Wi-Fi mreže omogućuju razmjenu podataka obližnjih digitalnih uređaja koristeći radio valove te se uglavnom koriste za pristup internetu ili za ostvarenje lokalne mreže, ali osim što služe za prijenos podataka sa jednog uređaja na drugi, radio valovi sadrže puno informacija o prostoru kroz kojeg se šire. U ovom radu će se istražiti mogućnost i preciznost detekcije broja ljudi u zatvorenoj prostoriji iz očitavanja snage signala u vremenu Wi-Fi izvora koji se nalazi u prostoriji koristeći algoritme strojnog učenja i detektore postavljene na rubovima prostorije. Vidjeti ćemo da uz dobru pripremu i obradu podataka prikupljenih sa detektora i dobro ”prilagođene” algoritme strojnog učenja možemo dobiti vrlo precizne informacije o prisutnosti ljudi u Wi-Fi polju. Ključan dio ovog istraživanja čini strojno učenje koje nam omogućava da iz velike količine podataka dođemo do smislenih rezultata. Što imamo veću količinu podataka za algoritme strojnog učenja time dobivamo preciznije rezultate. Na taj način se algoritam može konstantno unapređivati i prilagođavati. Prvo ćemo koristiti algoritam logističke regresije da bismo ispitali mogućnost detekcije ljudi u prostoriji te ćemo nakon toga ispitati preciznost predviđanja točnog broja ljudi pomoću algoritama kao što su neuralna mreža, višeklasna logistička regresija regresija i slučajna šuma. Više o spomenutim algoritmima ćemo saznati u kasnijim poglavljima.

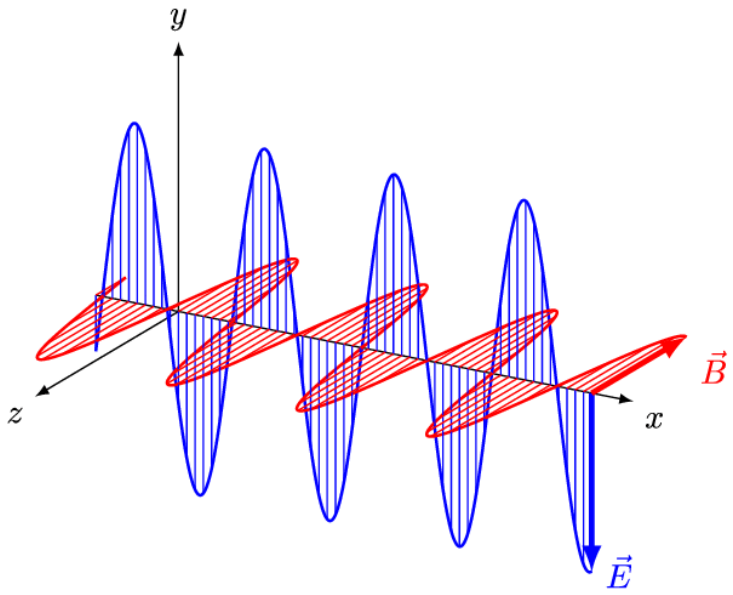
Koncept koji istražujemo može imati razne primjene u stvarnom svijetu, ali možda jedna od najočitijih je u sigurnosti javnih objekata kao što su škole, banke itd. U zadnjih nekoliko godina Wi-Fi tehnologija je iskusila nagli rast zbog svoje efikasnosti i priuštivosti zbog niskih cijena [1]. To čini korištenje Wi-Fi polja za detekciju ljudi odličnim kandidatom za nadziranje objekata umjesto kamera koje imaju veću ulogu u narušavanju privatnosti čovjeka te su skuplje za održavati. Uz sigurnosne sustave bazirane na kamerama često dolazi i osoblje koje mora nadgledavati postavljene kamere. To čini dodatni trošak te bi se umjesto osoblja mogli implementirati algoritmi strojnog učenja za detekciju ljudskog ponašanja u određenim okolnostima.

Ideja predstavljena u ovom radu je daleko od neistražene. Nedavna istraživanja pokazuju da uz procesiranje signala i korištenjem primjerenih struktura dubokog učenja možemo dobiti preciznu estimaciju lokacije čovjeka (ili više ljudi) i pozu ljudskog tijela u prostoru [2], a prva istraživanja u području detekcije ljudi pomoću radio valova su se odvijala još u 2011. godini [3]. No iako je koncept već relativno poznat ne znači da nije zanimljivo zaviriti u ovaj diplomski rad koji bi mogao poslužiti kao uvod u ovo područje istraživanja.

Da bismo otišli u detalje eksperimenta prvo moramo razumjeti fizikalne koncepte koji opisuju elektromagnetske valove i kako medij kroz kojeg se elektromagnetski valovi šire utječe na njih. Iz tih koncepata ćemo dobiti uvid kako snaga signala Wi-Fi polja nam može reći o tome što se događa u prostoriji. Nakon toga ćemo malo detaljnije objasniti korištene algoritme strojnog učenja iza čega će slijediti specifikacije postava eksperimenta, njegova izvedba i priprema podataka te ćemo na kraju diskutirati dobivene rezultate.

2 Elektromagnetski valovi

Elektromagnetski valovi čine jedan od osnovnih aspekata fizike koji igraju ulogu u shvaćanju svemira i tehnologije koju svakodnevno koristimo. Razumijevanje njihovih efekata i njih samih je ključno u raznim znanstvenim i tehnološkim područjima od telekomunikacije i medicinskog snimanja do astronomije i fizike čestica. Oni se razlikuju od valova koje vidimo na moru ili zvučnih valova koji širenjem kroz zrak dolaze do našeg uha. Da bi se širili, ti valovi trebaju neku vrstu medija kroz kojeg bi se propagirali i takve valove nazivamo mehaničkim valovima. Elektromagnetskim valovima nije potreban nikakav materijalni medij da bi prenosili energiju te se sastoje od električnih i magnetskih polja koja osciliraju okomito jedno na drugo (Slika 2.1).



Slika 2.1: Skica elektromagnetskog vala (slika preuzeta sa [4]).

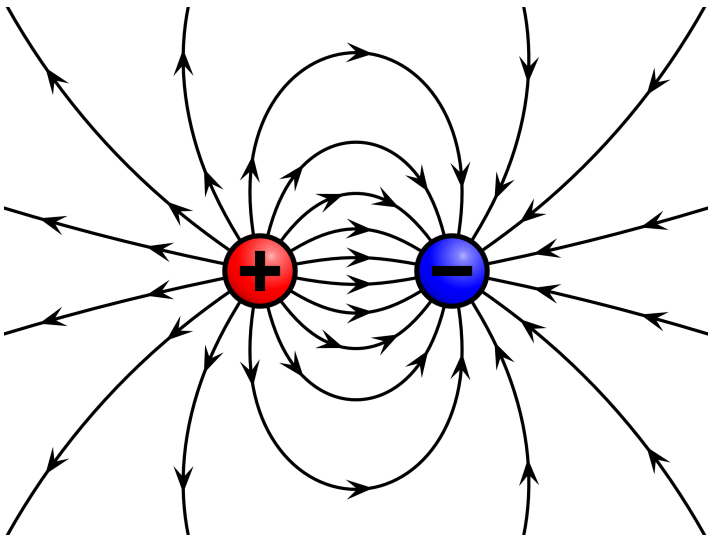
Svaki elektromagnetski val možemo opisati sa pripadajućom frekvencijom, valnom duljinom i energijom koju nosi sa sobom. Brzinu širenja elektromagnetskih valova u vakuumu označavamo sa c i naziva se brzina svjetlosti čija je vrijednost otprilike m/s. Vidljivo svjetlo sadrži valne duljine od 380 do 700 nanometara te čini samo mali dio elektromagnetskog spektra kojeg čine različiti tipovi elektromagnetskog zračenja kao što su radiovalovi, mikrovalovi, infracrveno zračenje, ultraljubičasto zračenje, rendgenske zrake i gamma zrake.

2.1 Električno i magnetsko polje

Na pokusima iz fizike koje su nam profesori i profesorice demonstrirali tokom srednje i osnovne škole primijetili smo da trljanjem plastičnog štapa o komad krzna čini štap električno nabijenim. Ako je predmet električno nabijen to znači da sadrži više elektrona od protona (negativno nabijen predmet) ili sadrži više protona od elektrona (pozitivno nabijen predmet). Dva nabijena tijela međusobno djeluju silom i tu interakciju možemo opisati takozvanim Coloumbovim zakonom kojeg je eksperimentalno utvdio Charles Augustin de Coloumb (jednadžba 2.1).

$$_{12} \quad \text{——} \text{————} \tag{2.1}$$

Sila je privlačna ako su naboji tijela (i) istoimeni, a odbojna ako su naboji suprotnog predznaka. Silu nazivamo električnom silom i djeluje na daljinu na ostala nabijena tijela te njezine utjecaje možemo opisati koristeći koncept polja. Električno polje opisuje utjecaj koji električno nabijeno tijelo ima na ostala nabijena tijela u svojoj blizini. To je vektorsko polje, što znači da u svakoj točki prostora ima svoj iznos, smjer i orijentaciju. Električno polje označavamo sa te njegovo ponašanje u prostoru grafički možemo prikazati sa silnicama (Slika 2.2). Silnice su zamišljene linije koje pokazuju smjer i relativnu jakost električnog polja u različitim točkama u prostoru. Silnice počinju iz pozitivnog naboja kojeg nazivamo i izvorom električnog polja, a završavaju na negativnom naboju ili takozvanim ponorom električnog polja.



Slika 2.2: Silnice električnog polja (slika preuzeta sa [5]).

Osim električne sile, nabijene čestice proizvode i magnetske sile. Susrećemo ih u električnim motorima, mikrovalnim pećnicama, zvučnicima, printerima i tvrdim diskovima. Najpoznatiji poznati primjeri magnetizma su permanentni magneti koji privlače nemagnetizirane željezne objekte i ostale magnete. Usklađivanje magnetne igle u kompasu sa Zemljinim magnetskim poljem je primjer takve interakcije. Ono što čini osnovu magnetizma je međudjelovanje električnih naboja koji su u pokretu. Za razliku od električne sile, koja djeluje na električne naboje micali se oni ili ne, magnetska sila djeluje samo na naboje koji se kreću.

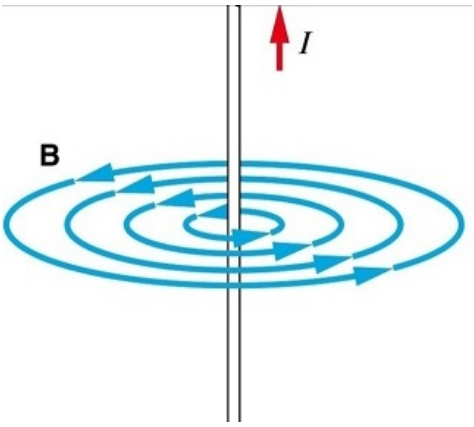
Električna sila se javlja u dva koraka: (1) nabijeno tijelo stvara električno polje u prostoru oko sebe, (2) drugi naboj interagira sa tim poljem. Magnetska sila se također javlja u dva koraka. Prvo, naboj u pokretu ili više naboja u pokretu (električna struja) stvaraju magnetsko polje . Nakon toga drugi naboj (ili više naboja) koji se kreće sa brzinom međudjeluje sa magnetskim poljem i iskusuje magnetsku silu [6] (jednadžba 2.2).

(2.2)

Magnetsko polje u prostoru možemo prikazati koristeći linije magnetskog polja (Slika 2.3). Ideja je ista kao i kod električnog polja. Linije magnetskog polja nam također pokazuju smjer i relativnu jakost magnetskog polja u različitim točkama u prostoru. Na mjestima gdje su linije bliže jedne drugima, polje je jače, a na mjestima gdje su linije udaljenije jedne od drugih, polje je slabije. Za razliku od silnica električnog polja, linije magnetskog polja nam ne pokazuju smjer sile koja djeluje na naboj. Iz jednadžbe 2.2 vidimo da je sila uvijek okomita na magnetsko polje. Također, nemamo izvore i ponore magnetskog polja već linije magnetskog polja počinju i završavaju same u sebe. Razlog tome je činjenica da nemamo dokaze za postojanje izoliranih magnetskih polova, to jest magnetskih monopola.

2.2 Utjecaj raznovrsnih medija na elektromagnetske valove

U prošlom naslovu smo pokazali par značajki električnih i magnetskih polja. I ti koncepti vrijede za polja koja se ne mijenjaju u vremenu kao što je električno polje koje stvara naboj koji miruje ili magnetsko polje stalne struje. Ali kada se polja mijenjaju u vremenu, onda više nisu nezavisna. Iz Maxwellovih jednadžbi (jednadžbe



Slika 2.3: Silnice magnetskog polja (slika preuzeta sa [7]).

2.3, 2.4, 2.5 i 2.6) za prostor u kojem nema slobodnih naboja niti struja možemo vidjeti da magnetsko polje koje se mijenja u vremenu djeluje kao izvor električnog polja. Isto vrijedi i za promjenjivo električno polje koje djeluje kao izvor magnetskog polja.

(2.3)

(2.4)

— (2.5)

— (2.6)

U kratkih par koraka iz gornjih jednadžbi možemo izvesti izraze za valne jednadžbe električnog i magnetskog polja (jednadžbe 2.7 i 2.8) iz kojih vidimo da se ta polja zaista ponašaju kao valovi koji se šire kroz prostor te je takvo ponašanje tih polja ono što čini elektromagnetske valove.

— (2.7)

— (2.8)

Iz jednadžbi 2.7 i 2.8 vidimo da je brzina širenja tih valova kroz vakuum jednaka brzini svjetlosti c.

— (2.9)

U slučaju kada se elektromagnetski val širi kroz neku vrstu medija kao što su voda, zrak ili neko čvrsto tijelo onda se brzina širenja vala mijenja (jednadžba 2.10) te se izraz pod korijenom zamjenjuje sa . Gdje je i . Pošto je relativna permeabilnost ( ) i relativna permitivnost ( ) dielektrika uvijek veća od 1, brzina širenja vala kroz neki medij je uvijek manja od brzine svjetlosti *c*.

(2.10)

Omjer brzine svjetlosti u vakuumu *c* i brzine elektromagnetskih valova u materijalu *v* je poznat u optici kao indeks loma *n* materijala (jednadžba 2.11).

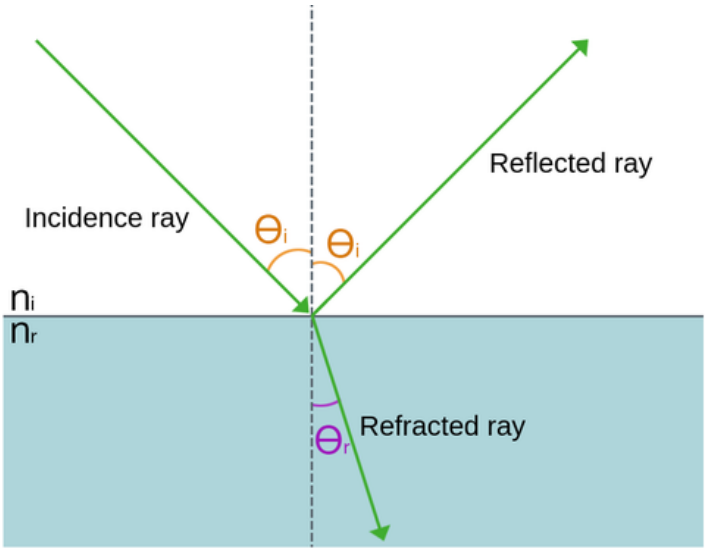
(2.11)

Kada elektromagnetski val upada na granicu između dva sredstva (npr. radio val koji se širi kroz zrak pa naiđe na čovjeka) mogu se dogoditi sljedeće promjene u širenju vala: refleksija i transmisija. Kod refleksije upadni elektromagnetski val se odbija (reflektira) od granice sredstava pod istim kutom pod kojim je došao. Upadni elektromagnetski val se može transmitirati u drugo sredstvo poštujući Snellov zakon (jednadžba 2.12).

(2.12)

Prilikom upada na granicu između dva sredstva elektromagnetski val se može potpuno reflektirati, potpuno transmitirati ili djelomično i reflektirati i transmitirati. Situacija je detaljno prikazana na slici 2.4.

Osim refleksije i transmisije, elektromagnetski valovi mogu biti apsorbirani od strane sredstva kroz kojeg propagiraju. Kod eksperimenta kojeg provodimo u ovom istraživanju radio valovi koje se šire iz Wi-Fi izvora prolaze kroz ljude u prostoriji. Voda ima mogućnost apsorbirati određene frekvencije radio valova te pošto voda čini oko 60% sastava odrasle osobe dio energije koju nose radio valovi je apsorbiran prilikom prolaska valova kroz ljude. Refleksiju i transmisiju valova tokom naleta na osobe također prate efekti kao što su ogib i interferencija valova koji uzrokuju razlike u jačini električnih i magnetskih polja u prostoru. Iz navedenih koncepata možemo pretpostaviti da će postojati razlike u očitavanju jakosti signala u vremenu na detektorima za različit broj osoba u prostoriji.



Slika 2.4: Upad elektromagnetskog vala na granicu između dva sredstva (slika preuzeta sa [8]).

3 Osnovni koncepti strojnog učenja

Strojno učenje, zajedno sa statistikom, su područja koja opisuju kako učiti i napraviti predviđanja o podacima. U današnje vrijeme dostupnost velikim količinama podataka je obilježje moderne znanosti gdje je analiza podataka postala vrlo važna komponenta u različitim područjima kao što su fizika čestica, astronomija, biofizika i kvantno računanje. Uz to, strojno učenje i data science igraju sve važniju ulogu u raznim aspektima moderne tehnologije poput pametnih uređaja i samovozećih automobila. Strojno učenje je grana umjetne inteligencije s ciljem razvoja algoritama koji su sposobni samostalno učiti iz podataka. U načelu, algoritam strojnog učenja mora biti sposoban prepoznati objekte u danom okruženju i predvidjeti ponašanje tog okruženja da bi donio informirane odluke.

Da bismo napravili predviđanje sustava kojeg proučavamo prvo odaberemo neku kvantitetu (interferentni uzorak elektromagnetskih valova) tog sustava koju možemo promatrati koja ovisi o nekim parametrima (brzina elektromagnetskog vala) modela koji opisuje vjerojatnost opažanja za određeni . Onda napravimo eksperiment da dobijemo set podataka (tako zvani ”*dataset*”) koji ćemo iskoristiti da ”prilagodimo” model. Kada govorimo o prilagođavanju modela govorimo o traženju koji daje najbolje objašnjenje za podatke (jednadžba 3.1) [9].

(3.1)

Glavni zadatak algoritama strojnog učenja će uglavnom biti pronalaženje parametara do kojih nije lako doći, ali korištenje parametara koji su blizu ispada jednako učinkovito. Postoje i razni algoritmi koji ne prate predloženu formulu za učenje te ćemo neke od njih istražiti u sljedećem poglavlju.

3.1 Nadzirano i nenadzirano strojno učenje

U našem eksperimentu najviše će nam poslužiti nadzirano učenje (eng. *supervised learning*). Kod nadziranog učenja algoritam je treniran na označenom setu podataka gdje svakom primjeru pridodajemo odgovarajuću oznaku setu . Učenje nazivamo ”nadziranim” jer je algoritam vođen točnim odgovorima (oznakama) tokom treniranja što mu omogućuje da napravi predviđanja na novim, neviđenim podacima. Česti zadaci kod nadziranog učenja su problemi regresije i klasifikacije. Kod regresije cilj je predvidjeti kontinuirane izlazne numeričke vrijednosti kao što su cijene kuća ovisno o odabranim značajkama kuće. Kod klasifikacije cilj je kategorizirati podatke u unaprijed definirane klase ili oznake (npr. klasifikacija slika).

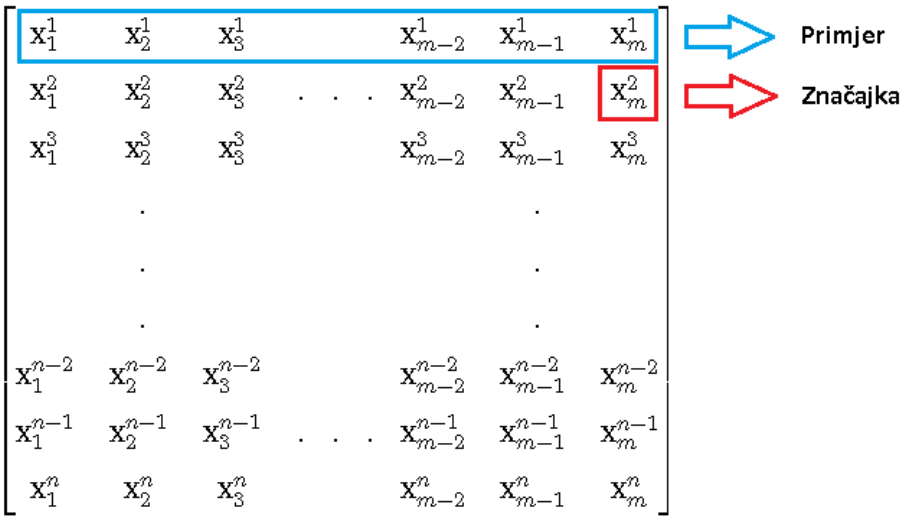
Kod nenadziranog učenja algoritam se trenira na setu podataka gdje nemamo određene vrijednosti ili oznake pridodane podacima (nemamo set). Cilj nenadziranog učenja je pronaći strukture, uzorke ili veze unutar seta podataka bez da vodimo algoritam sa određenim znanjem o priloženim podacima. Vrste nenadziranog učenja uključuju algoritme klasteriranja koji slične podatke grupiraju u klastere te metode smanjivanja dimenzionalnosti u kojima se smanjuje broj značajki koji opisuje set podataka bez gubitka važnih informacija.

Nadzirano i nenadzirano učenje nisu jedini tipovi strojnog učenja, postoje još i podržano i polu-nadzirano učenje ali se njima nećemo baviti u ovom radu. Većina korištenih algoritama u eksperimentu pripadaju nadziranom učenju sa jednim algoritmom nenadziranog učenja koji će nam poslužiti u vizualnoj prikazu našeg seta podataka.

3.2 Postavljanje i rješavanje problema u strojnom učenju

Veliki broj zadataka u nadziranom strojnom učenju započinje sa istim koracima. Prvi korak čini set podataka gdje je matrica čiji svaki redak predstavlja jedan primjer podatka (npr. ako nam se set podataka sastoji od raznih slika da naučimo al-

goritam za kompjuterski vid, jedan redak predstavlja jednu sliku) i svaki stupac predstavlja jednu značajku tog seta (npr. vrijednost prvog piksela slike) koje algoritam strojnog učenja koristi da bi ostvario predviđanja, klasifikacije i ostale analize. Struktura seta podataka se može vidjeti na slici 3.1. Set je vektor u n-dimenzionalnom prostoru gdje je n ukupan broj primjera u setu podataka i sadrži vrijednosti koje model strojnog učenja pokušava predvidjeti koristeći značajke iz ulaznog seta podataka . Svaki odgovara vrijednosti pridodanoj i-tom primjeru iz (npr. ako želimo da naš algoritam predviđa da li je na slici mačka ili pas, onda za svaku sliku u stavljamo odgovarajuću oznaku u).



Slika 3.1: Struktura matrice X.

Drugi korak čini model koji je zapravo funkcija koja ovisi o parametrima . se još naziva hipotezom i ona se koristi za predviđanje izlaznih vrijednosti iz ulaznih vrijednosti . U algoritmima koji koriste linearni model, će uvijek biti funkcija koja je nadogradnja na linearnu kombinaciju parametara i značajki primjera čiju strukturu možemo vidjeti u jednadžbi 3.2,

(3.2)

gdje je uvijek jednak 1.

Zadnji korak čini funkcija gubitka (eng. *loss function*) koja nam omogućuje da prosudimo učinkovitost modela sa opažanjima . Model se uči traženjem vrijednosti parametara koji minimiziraju funkciju gubitka. Najčešće korištena funkcija gubitka je kvadratna pogreška te minimiziranje funkcije gubitka

kvadratne pogreške (jednadžba 3.3) nazivamo metodom najmanjih kvadrata. Primjenu metode najmanjih kvadrata susrećemo u jednom od jednostavnijih algoritama strojnog učenja; linearne regresije.

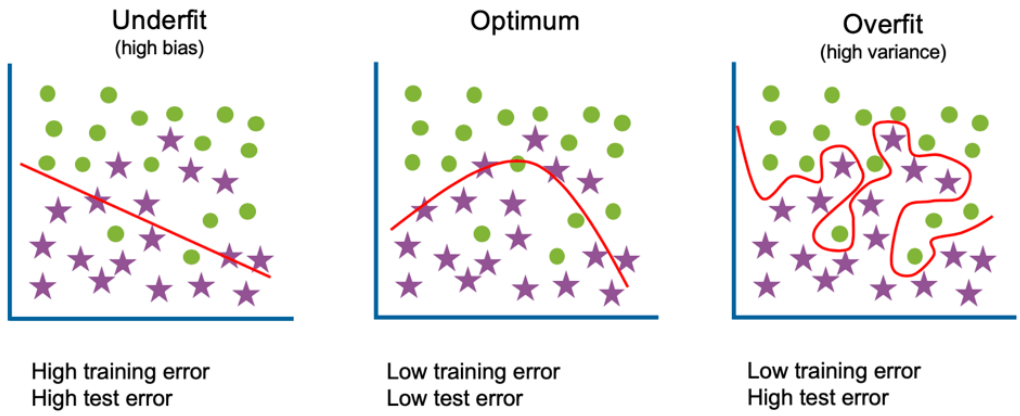
— (3.3)

Naš eksperiment pripada problemu klasifikacije te ćemo koristiti gubitak unakrsne entropije kao funkciju gubitka o kojoj ćemo više reći u četvrtom poglavlju.

Osim koraka s kojima se postiže učenje (treniranje) algoritma bitno je naglasiti korak koji se treba primijeniti na podacima koji dolazi prije samog učenja. Istraživači strojnog učenja prate standardizirani postupak da dobiju modele koji su korisni za predviđanje i dio tog postupka obuhvaća nasumičnu podjelu ulaznog seta podataka na dva međusobno isključiva seta i koje nazivamo set za trening (eng. *training set*) i testni set (eng. *test set*). Tipično se većina podataka odjeljuje u set za trening (oko 90% ili 80%) te se ostatak odjeljuje za testni set. Model se onda trenira koristeći samo podatke iz seta za trening. Učinkovitost algoritma možemo procijeniti gledajući vrijednost funkcije gubitka na setu za trening. Iako nam funkcija gubitka na setu za trening daje informaciju o tome koliko je model dobro prilagođen za priloženi set podataka nas zanima i sposobnost predviđanja algoritma na testnom setu. Za mjerenje učinkovitosti predviđanja je bolje gledati vrijednosti funkcije gubitka na testnom setu . Točnost rezultata modela još možemo procijeniti gledajući njegovu preciznost čija je jednaka omjeru broja točnih predviđanja i broja ukupnih predviđanja te ju izražavamo u postocima.

Treniranje algoritma je često postupak pokušaja i pogreške. Često se možemo susresti sa slučajem gdje istrenirani model ima visoku preciznost na setu za trening, ali neuspijeva dobro predvidjeti ponašanje testnog seta. U tom slučaju kažemo da je model ”preprilagođen” (eng. *overfitting*) što znači da je šum koji se nalazio u podacima seta za trening značajno utjecao na proces učenja. To možemo izbjeći koristeći više podataka pri učenju algoritma, koristeći jednostavniji model, uporabom regularizacije (modifikacije na algoritmima napravljene za smanjivanje preprilagođenosti), koristeći ansamble koji koriste predviđanja više modela da bi dobili konačne rezultate te razne druge metode. Također postoji slučaj u kojem ćemo nakon treniranja dobiti nisku preciznost na setu za trening i na testnom setu. Tada govorimo o ”ne-

dovoljno prilagođenom" modelu (eng. *underfitting*). Model strojnog učenja je tada prejednostavan da primijeti kompleksnije uzorke u setu podataka te je potrebno isprobati kompleksniji model, koristiti veći set podataka, povećati broj iteracija u algoritmu za optimizaciju parametara (više o tome u sljedećem naslovu) i slično. Primjer preprilagođavanja, nedovoljnog prilagođavanja i dobrog prilagođavanja za problem klasifikacije možemo vidjeti na slici 3.2. Za model još možemo reći da ima visoku

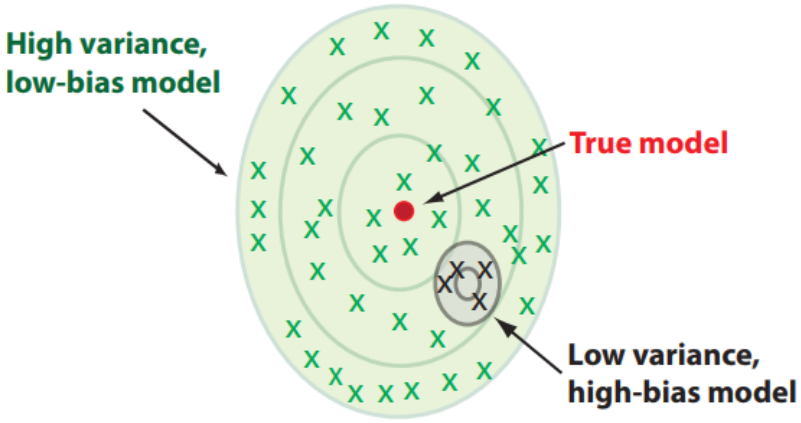


Slika 3.2: Slika prikazuje nedovoljno prilagođavanje, dobro prilagođavanje i preprilagođavanje granice odluke između dvije različite klase podataka (slika preuzeta sa [10]).

varijancu ako dobivamo predviđanja koja se jako razlikuju za različite setove za trening istog uzorka. To je često indikator da je model preprilagođen. Možemo reći i da model ima visoku "pristranost" (eng. *bias*) kada dobivamo predviđanja koja konstantno odstupaju za jednaku vrijednost od pravih vrijednosti. Visoka pristranost se uglavnom javlja ako koristimo prejednostavan model ili imamo krive pretpostavke o ponašanju uzorka. Shemu različitih pristranosti i varijanci modela možemo vidjeti na slici 3.3.

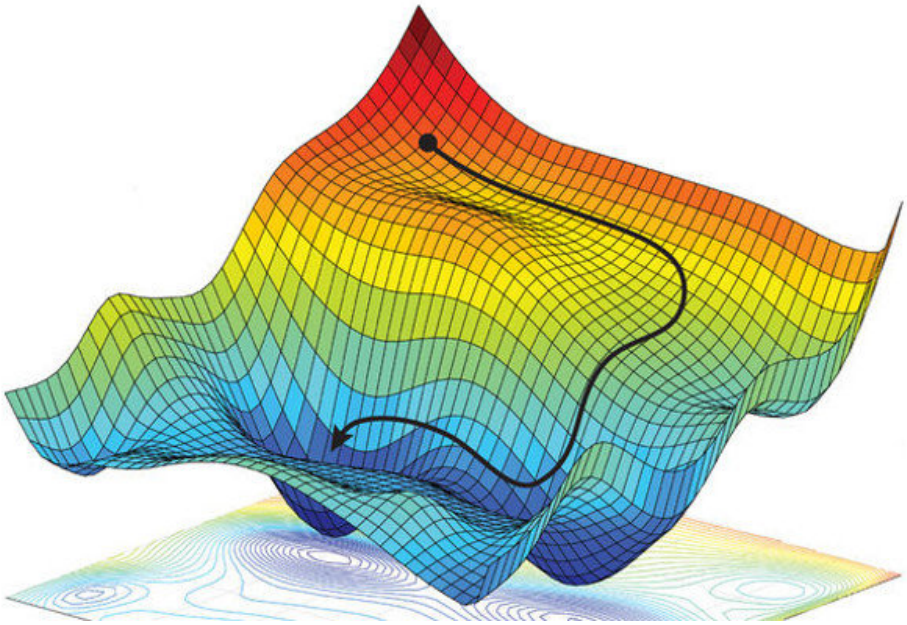
3.3 Gradijentni spust

Rekli smo da se model trenira tražeći vrijednosti parametara koje minimiziraju funkciju gubitka koja nam objašnjava koliko dobro model objašnjava set podataka . Jedna od najučinkovitijih i najkorištenijih metoda koja nam omogućuje dobiti optimalne vrijednosti parametara jest gradijentni spust (eng. *gradient descent*). Ideja je jednostavna; iterativno namještavamo vrijednosti parametara u smjeru gdje je gradijent funkcije gubitka najveći i nega-



Slika 3.3: Slika prikazuje prikaz modela sa visokom varijancom i niskom pristranošću (zeleno) uz model sa niskom varijancom i visokom pristranošću (crno) (slika preuzeta iz [9]).

tivan. Na taj način jamčimo da procedura treniranja algoritma pomiče vrijednosti parametara prema lokalnom minimumu funkcije gubitka (Slika 3.4).



Slika 3.4: Primjer prikaza gradijentnog spusta na funkciji gubitka koja sadrži dva parametara (slika preuzeta sa [11]).

U najobičnijem algoritmu za gradijentni spust mijenjamo parametre na sljedeći način. Inicijaliziramo parametre na neku početnu vrijednost (najčešće na nulu) i iterativno ažuriramo parametre prema jednadžbi 3.4,

(3.4)

gdje je η stopa učenja (eng. *learning rate*) koja nam govori koliko veliki korak želimo napraviti u smjeru gradijenta tijekom i-te iteracije. Odabirom dovoljno male vrijednosti η (uglavnom oko 0.01) omogućujemo konvergiranje prema lokalnom minimumu neovisno od kuda smo počeli. Ukupan broj iteracija je također naš odabir koji ovisi o tome koliku preciznost želimo postići od modela $f(x;\Theta)$. Za algoritme koji koriste linearni model i funkcije gubitka koje ćemo koristiti u ovom istraživanju pravilo ažuriranja j-tog parametara možemo izraziti na sljedeći način (jednadžba 3.5),

$$\Theta_j = \Theta_j + \eta \sum_{i=1}^{n} (y^{(i)} - f_\Theta(x^{(i)}))x_j^{(i)} \tag{3.5}$$

gdje je n ukupan broj primjera.

U stvarnosti često moramo raditi sa veoma ”hrapavim” funkcijama gubitka koje sadrže više lokalnih minimuma te doći do optimalnog minimuma koristeći gradijentni spust jako ovisi o početnim vrijednostima parametara. Iz tog razloga postoje razne inačice gradijentnog spusta čija je svrha izbjeći plitke minimume. Jednu od tih inačica dobivamo dodavajući tako zvanu ”količinu gibanja”, to jest dodatni izraz koji služi kao sjećanje na smjer u kojem smo mijenjali parametre u prošlom koraku te omogućuje izbjegavanje plitkih minimuma (jednadžba 3.6).

$$\begin{aligned} v_i &= \gamma v_{i\ 1} + \eta \nabla_\Theta C(\Theta_i) \\ \Theta_{i+1} &= \Theta_i - v_i \end{aligned} \tag{3.6}$$

U praksi se još često koriste i metode poput ADAM i RMSprop u kojima se za dodatnu optimizaciju regulira i stopa učenja η ovisno o strmini pada funkcije gubitka na mjestu na kojem se nalazimo u prostoru parametara (na područjima gdje je gradijent funkcije mali želimo veći η , dok na područjima gdje je gradijent veliki želimo manji η).

Ako imamo veliku količinu podataka kod metode gradijentnog spusta se javlja još jedan problem. Računanje gradijenata je vremenski skup proces ako imamo veliki set podataka. Stoga umjesto cijelog seta podataka X koristimo manju podskupinu podataka tako zvanu minigrupu (eng. *mini-batch*) za računanje gradijenata. Ako imamo ukupno n podataka i veličina minigrupe je M onda ćemo imati n/M minigrupa. U

tom slučaju pravilo ažuriranja parametara će poprimiti sljedeći izraz (jednadžba 3.7).

(3.7)

Prolazak kroz minigrupa još nazivamo epohom te uporabom ovakve stohastičke aproksimacije koristimo metodu koju nazivamo stohastički gradijentni spust (SGD). Veličina minigrupe također ne smije biti premala jer čini optimizaciju parametara veoma šumovitom što povećava varijancu modela.

4 Korišteni algoritmi

U ovom poglavlju ćemo opisati način rada algoritma logističke regresije, višeklasne logističke regresije, neuronske mreže, slučajne šume i analize glavnih komponenti (PCA). Svi navedeni algoritmi su pisani u programskom jeziku Python koristeći samo biblioteke numpy i nekoliko biblioteka za korištenje objekata korisnih kod određenih izračuna i za grafički prikaz podataka i rezultata. Algoritmi su implementirani korištenjem samo osnovnih biblioteka u svrhu da jasnije možemo vidjeti i razumjeti procedure i izračune potrebne za pravilan rad algoritma.

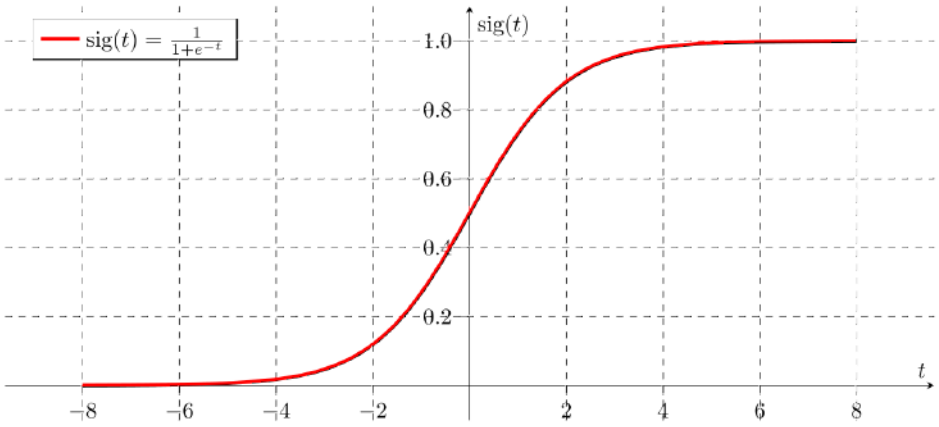
4.1 Logistička regresija

Naš prvi algoritam ćemo objasniti koristeći jednostavan primjer. Recimo da imamo određenu skupinu europskih zečeva i kunića. Želimo napisati algoritam koji će raspoznati da li je životinja europski zec ili kunić. Kunići su inače manje životinje od europskih zečeva, ali ih možemo prepoznati i po tome što imaju manje uši i manje zadnje noge od europskih zečeva. Primjeri u našem označenom setu podataka koji se sastoji od europskih zečeva i kunića će stoga imati dvije značajke: duljinu ušiju i duljinu stražnjih nogu (tablica 4.1).

Algoritam logističke regresije će imati zadatak naučiti kako točno klasificirati primjere navedenih životinja gledajući samo dvije značajke. Da bismo napravili bilo kakva predviđanja moramo imati dobro definiran model . Već smo naveli strukturu funkcije hipoteze u jednadžbi 3.2 što znači da samo moramo odabrati funkciju koja će nam poslužiti u klasificiranju primjera. U logističkoj regresiji koristi

Duljina uha	Duljina stražnje noge	Klasa
6.0	14.0	0
8.5	16.2	1
4.4	14.3	0
7.5	17.5	1
5.2	14.9	0

Tablica 4.1: Tablica prikazuje značajke i pripadajuću klasu prvih pet primjera iz seta podataka koji se sastoji od europskih zečeva (klasa 1) i kunića (klasa 0).



Slika 4.1: Graf prikazuje ponašanje sigmoidne funkcije (slika preuzeta sa [12]).

se tzv. sigmoidna funkcija (jednadžba 4.1),

$$g(x) = \frac{1}{1 + e^{\ x}} \tag{4.1}$$

koja je jako korisna kod problema binarne klasifikacije jer ovisno o vrijednosti linearnog modela $\Theta^T x^{(i)}$ možemo dobiti vjerojatnost točne klasifikacije primjera. Prikaz sigmoidne funkcije možemo vidjeti na slici 4.1. Za vrijednosti funkcije veće ili jednake od 0.5 primjer svrstavamo u klasu 1, u suprotnom primjer svrstavamo u klasu 0.

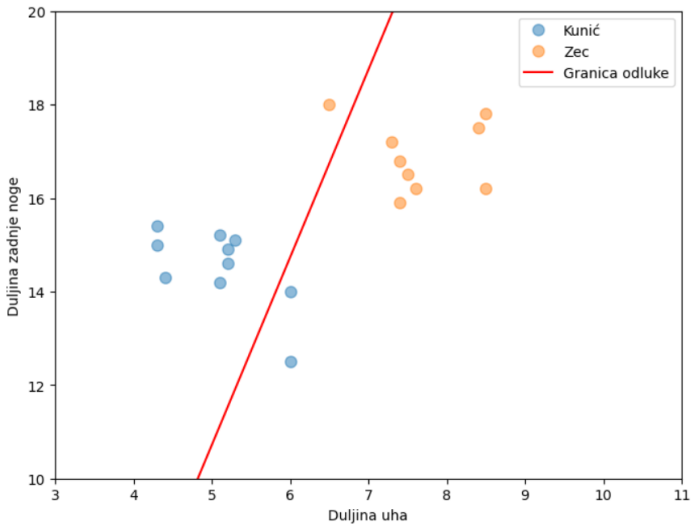
Naravno, naš model nam ne bi bio previše koristan bez parametara optimiziranih za točna predviđanja stoga koristimo metodu gradijentnog spusta s kojom dolazimo do željenih vrijednosti. Optimizaciju pojedinog parametra postižemo koristeći pravilo ažuriranja prema jednadžbi 3.5. Funkcija gubitka koju koristimo za prosuđivanje učinkovitosti algoritma logističke regresije i iz koje smo izveli pravilo za ažuriranje

parametara se zove gubitak unakrsne entropije (jednadžba 4.2),

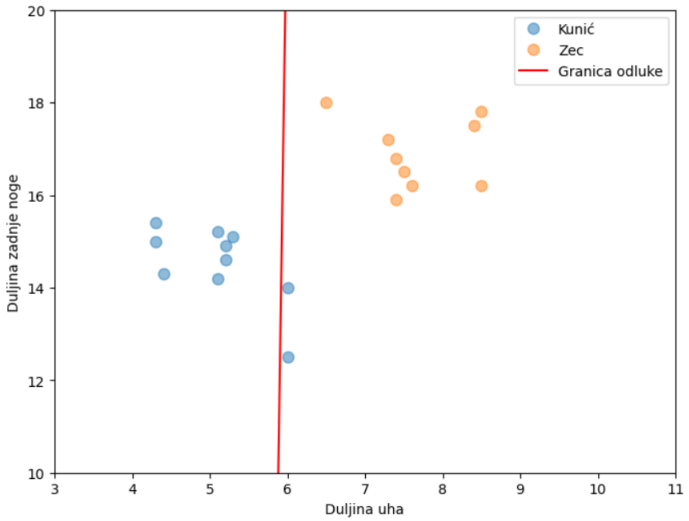
– (4.2)

gdje je ukupan broj primjera. Primijetimo da kod svakog slučaja klasifikacije samo jedan član pod sumom doprinosi iznosu funkcije gubitka; ako se radi o slučaju klase 1 drugi član iščezava i suprotno vrijedi za slučaj klase 0. S ovakvom funkcijom uspijevamo ispitati točnost modela za dva slučaja koristeći samo jedan izraz.

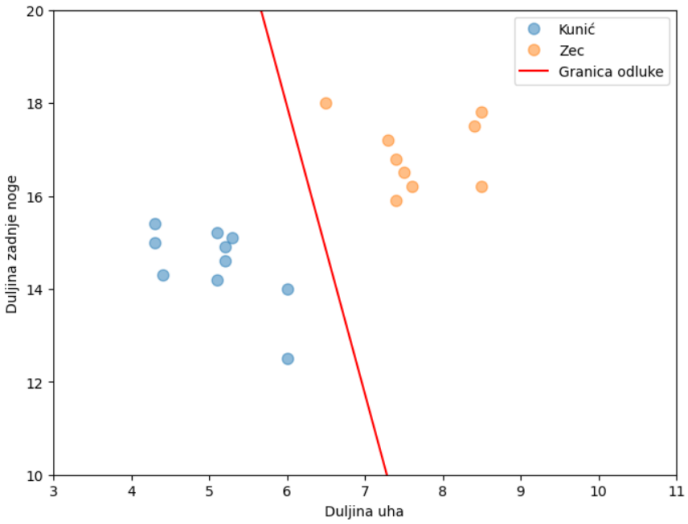
Sada možemo nastaviti sa prikazom principa rada algoritma logističke regresije na primjeru nakon što smo obišli par ključnih koncepata. Nakon učitavanja setova podataka i , algoritam inicijalizira parametre (imamo ih tri u ovom primjeru) i postavlja ih na nulu. Nakon toga parametri se ažuriraju prema jednadžbi 3.7 za određeni broj iteracija koji ovisi o odabiru veličine minigrupe i broja epohi. Svakom iteracijom parametri se pomiču prema optimalnoj vrijednosti za klasifikaciju slučajeva kao što možemo vidjeti na slikama 4.2, 4.3 i 4.4 gdje pravac linearnog modela konvergira prema dobro definiranoj granici između dvije klase. Implementacija logističke regresije u programskom jeziku Python se može vidjeti u dodatku B.



Slika 4.2: Granica odluke nakon 3000 iteracija.



Slika 4.3: Granica odluke nakon 7000 iteracija.



Slika 4.4: Granica odluke nakon 15000 iteracija.

4.2 Višeklasna logistička regresija

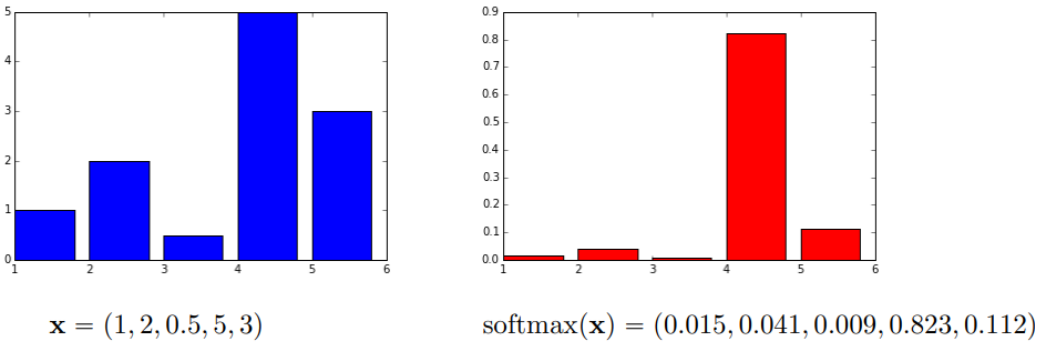
Kod logističke regresije zadatak je bio klasificirati točke podataka u jednu od dviju mogućih klasa. Taj algoritam se zato još naziva i binarna logistička regresija. Ako imamo više od dviju mogućih klasa onda moramo modificirati način na koji zapisujemo oznake podataka tako da sada postaje matrica, gdje je ukupan broj podataka i je ukupan broj mogućih klasa. Za pojedinu točku podataka stavljamo 1 na stupac odgovarajuće klase dok za ostale klase stavljamo nulu. Isti postupak ćemo napraviti i sa parametrima koji se sada zapisuju u matricu gdje je broj značajki te ćemo imati parametara za svaku klasu.

Pošto ćemo imati zasebne parametre za svaku od klasa moramo uzeti model

takav da skalarne umnoške propusti kroz funkciju koja će se pobrinuti da se vjerojatnosti svih klasa zbrajaju ukupno na 1 (jer svaki primjer može pripadati samo jednoj klasi). Funkcija koja radi upravo to se naziva softmax funkcija (jednadžba 4.3).

$$\frac{T}{\frac{k}{T}} \tag{4.3}$$
$$j$$

Za neki ulazni primjer , funkcija softmax uzima vrijednosti za svaku od klasa te ih preslikava u -dimenzijski vektor čije se komponente zbrajaju u 1. Funkcija softmax radi dvije stvari: normalizira sve vrijednosti tako da njihov zbroj bude jednak 1, ali i pojačava veće vrijednosti i smanjuje manje vrijednosti (slika 4.5). Ulazni primjeri su na kraju svrstani u klasu najveće vjerojatnosti.



Slika 4.5: Primjer normalizacije, povećavanja i smanjivanja vrijednosti kod funkcije softmax (preuzeto sa [13]).

Funkciju gubitka koja nam omogućuje optimizaciju algoritma dobivamo iz poopćenja gubitka unakrsne entropije tj. funkcije koju smo koristili kod binarne logističke regresije. Opet koristimo negativan logaritam vjerojatnosti oznaka prilagođen za slučaj kada imamo klasa (jednadžba 4.4),

(4.4)

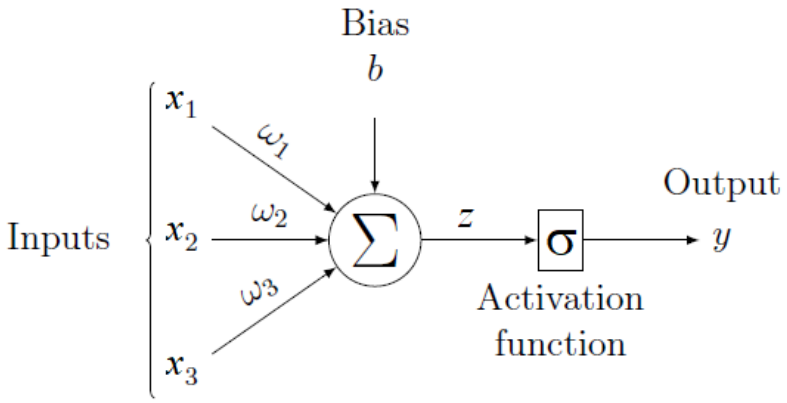
gdje je ukupan broj primjera i ukupan broj klasa. Možemo vidjeti da je logika ove funkcije gubitka ista kao i kod binarne logističke regresije. Ako je oznaka -tog primjera klase jednaka 1, onda želimo da predviđanje modela tog primjera za tu klasu bude visoka jer će tada gubitak biti minimalan. U suprotnom ako je za taj primjer vjerojatnost za klasu mala onda logaritam daje veliki negativan broj koji

pomnožen sa -1 daje velik gubitak.

Pravilo ažuriranja parametara je isto kao i kod binarne logističke regresije te algoritam funkcionira prema istim koracima. Jedina razlika je u tome što imamo više od dviju klasa pa algoritam koristi više od jedne granice odluke. Implementacija višeklasne logističke regresije u programskom jeziku Python se može vidjeti u dodatku C.

4.3 Neuronska mreža

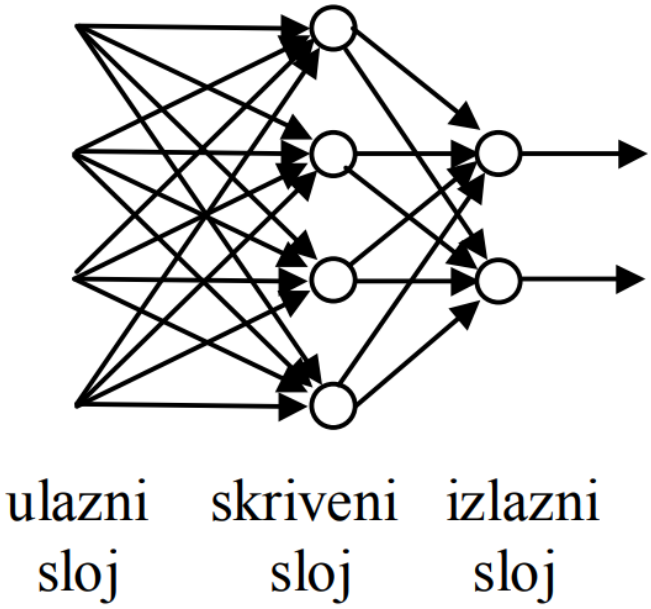
Kao što je višeklasna logistička regresija nadogradnja binarnoj logističkoj regresiji tako možemo neuronsku mrežu gledati kao nadogradnju na višeklasnu logističku regresiju. Neuronske mreže su inspirirane načinom rada mozga prema kojem je napravljena arhitektura algoritma koja se sastoji od raznih slojeva neurona koji su međusobno umreženi te se kroz proces učenja informacije spremaju u vezama između neurona. Konvencija je da parametre u neuronskim mrežama nazivamo težinama (eng. *weights*) te parametar zovemo pristranost (eng. *bias*) . Rekli smo da neuronsku mrežu čine međusobno povezani neuroni (slika 4.6) koji se sastoje od linearnog dijela i aktivacije. Linearni dio čini skalarni umnožak parametara i značajki primjera dok drugi dio čini aktivacijska funkcija poput sigmoide, softmax funkcije, ReLU, itd. Aktivacijska funkcija uvodi nelinearnost pomoću koje jačamo ili gušimo signale koji prolaze kroz neurone ovisno o uvjetima koji su postavljeni. Algoritam binarne logističke regresije možemo gledati kao neuronsku mrežu koja se sastoji od jednog neurona.



Slika 4.6: Shema neurona (preuzeto sa [14]).

Neuroni su svrstani u slojeve gdje slojeve čine neuroni koji nisu međusobno po-

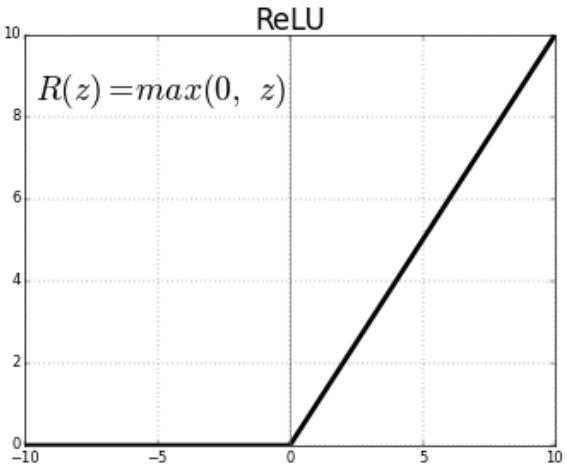
vezani (slika 4.7). Slojevi mogu biti ulazni, izlazni ili skriveni. Ulazni sloj služi za učitavanje podataka te se u njemu ne izvode nikakva računanja. Svaki neuron u ulaznom sloju predstavlja pojedinu značajku ulaznog seta podataka. Izlazni sloj daje izlazne podatke neuronske mreže koji ovisi o vrsti zadatka kojeg mreža treba izvršiti. Kod problema klasifikacije izlazni sloj ima onoliko neurona koliko imamo klasa te se uglavnom koristi funkcija softmax kao aktivacijska funkcija koja svrstava točke podataka u pripadne klase. Algoritam višeklasne logističke regresije možemo gledati kao neuronsku mrežu koja sadrži samo ulazni i izlazni sloj.



Slika 4.7: Shema neuronske mreže (preuzeto sa [15]).

Skriveni slojevi imaju ključnu ulogu u učenju algoritma te većinu informacija potrebnih za donošenje točnih predviđanja sadrže neuroni skrivenih slojeva i njihove veze. Nazivamo ih skrivenim jer nismo sasvim sigurni što se u njima događa ali znamo da sa dovoljno velikim setom podataka otkrivaju kompleksne veze između podataka. Zato neuronske mreže spadaju i pod modele ”crne kutije”. ReLU (eng. *Rectified Linear Unit*) je jedna od najkorištenijih aktivacijskih funkcija u skrivenim slojevima zbog svoje jednostavnosti i učinkovitosti (slika 4.8). Broj skrivenih slojeva kao i broj neurona u njima ovisi o kompleksnosti zadatka i količini računalnih resursa. Za jednostavne probleme nije potrebno više od jednog skrivenog sloja dok bi više slojeva neurona bilo korisno kod kompleksnijih zadataka ali to ujedno zauzima i više računalnih resursa.

Podaci prolaze kroz neuronsku mrežu preko procesa koji se zove ”širenje unapri-



Slika 4.8: Graf pokazuje ponašanje funkcije ReLU (preuzeto sa [16]).

jed" (eng. *forward propagation*) te se kreću od ulaznog sloja prema izlaznom sloju gdje su svrstani u odgovarajuće klase. Iskoristiti ćemo primjer mreže koja se sastoji od dva skrivena sloja i izlaznog sloja da prikažemo proces prolaska matrice (ulazni set podataka) kroz neuronsku mrežu. Sljedeće jednadžbe prikazuju prolazak podataka kroz prvi skriveni sloj,

(4.5)

(4.6)

gdje označava linearni dio a označava aktivaciju sloja. Prolazak kroz drugi skriveni sloj prikazuju sljedeće jednadžbe.

(4.7)

(4.8)

Na kraju još imamo izlazni sloj.

(4.9)

(4.10)

Da bi naša neuronska mreža bila u stanju naučiti nešto iz danih primjera moramo napraviti optimizaciju parametara i iz svakog sloja. Kao i kod prethodnih al-

goritama parametre ćemo optimizirati koristeći metodu gradijentnog spusta, ali da bismo dobili pravilo ažuriranja (jednadžba 3.4) za sve parametre koristi se algoritam ”povratnog širenja” (eng. *backpropagation*) koji ćemo objasniti na istom primjeru mreže. Nakon što prođe kroz proces ”širenja unaprijed” potrebno je napraviti prvu iteraciju ažuriranja parametara prema jednadžbama

———— (4.11)

———— (4.12)

gdje označava sloj kojem parametri pripadaju i je aktivacija zadnjeg sloja tj. izlazni podaci. Koristeći pravilo derivacije složene funkcije možemo raspisati izraze potrebne za optimizaciju parametara naše mreže.

—— —— —— —— (4.13)

—— —— —— —— —— (4.14)

—— —— —— —— —— (4.15)

Možemo primijetiti da je najjednostavnije prvo izračunati gradijente za treći sloj mreže pa iskoristiti izračunate izraze za drugi sloj te isti postupak napraviti i za prvi sloj. Kada su svi gradijenti izračunati parametri se ažuriraju i prolazi ponovo kroz isti postupak sve dok funkcija gubitka nije minimizirana. Algoritam nazivamo ”povratnim širenjem” jer se gradijenti funkcije gubitka kao i pogreške u predviđanjima računaju unazad od zadnjeg sloja prema prvom.

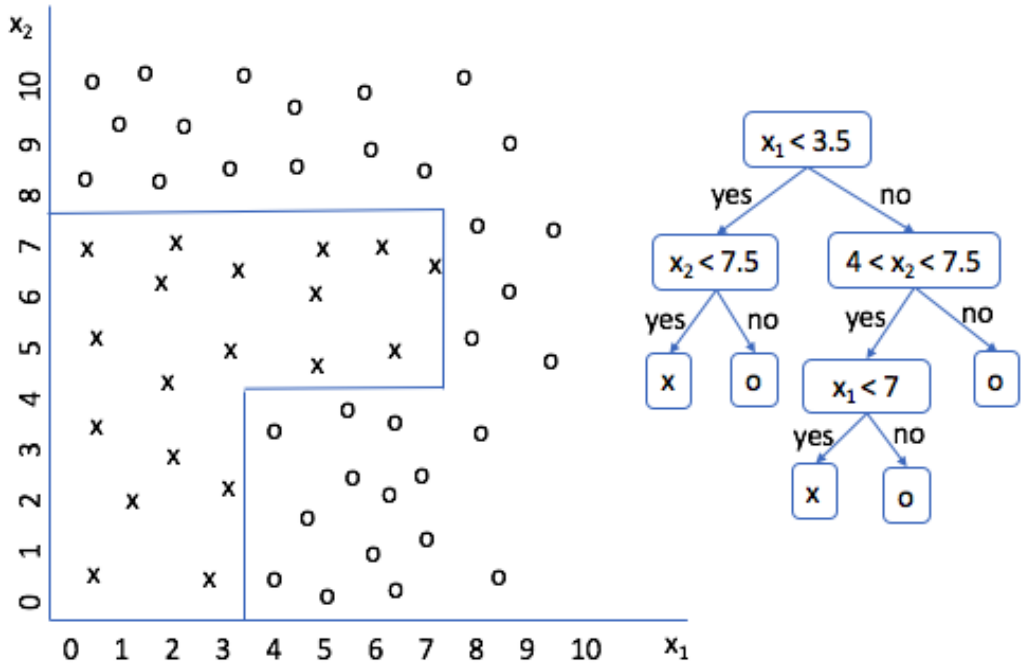
Proces učenja neuronske mreže se uglavnom sastoji od raznih računskih operacija nad matricama te se u praksi za tu svrhu koriste grafičke kartice jer je njihovo procesiranje podataka optimizirano za rad s matricama. Na taj način skraćujemo vrijeme potrebno za treniranje algoritma koje je znatno duže kada se ta računanja odvijaju na procesoru. U programskom jeziku Python biblioteke kao što su Tensorflow omogućuju korištenje grafičke kartice za algoritme strojnog učenja.

Postoje razni tipovi neuronskih mreža od kojih je svaki dizajniran za specifične zadatke i tipove podataka. Mrežu koju smo opisali u ovom poglavlju je standardna višeslojna mreža bez povratnih veza koja se koristi općenito za probleme regresije

i klasifikacije. Konvolucijske neuronske mreže sadrže konvolucijske slojeve koji su specijalizirani za procesiranje slika dok su mreže s povratnim vezama prikladnije za sekvencirane podatke gdje je bitan poredak među podacima te se koriste za prepoznavanje govora. Implementacija dvoslojne neuronske mreže bez povratne veze sa jednim skrivenim slojem u programskom jeziku Python se može vidjeti u dodatku D.

4.4 Slučajna šuma

Ne možemo imati šumu bez stabala, a upravo stablo je struktura podataka koja će nam trebati da izgradimo algoritam slučajne šume. Slučajnu šumu čini više algoritama stabla odluke pa ćemo prvo obići način rada stabla odlučivanja. Algoritam funkcionira na jednostavan način. Prolaskom kroz stablo primjerima su u svakom čvoru postavljeni određeni uvjeti na njihovim značajkama. Ovisno o tome kako njihove značajke zadovoljavaju uvjete primjeri su dovedeni u listove koji predstavljaju regiju pripadne klase. Primjer načina rada stabla odluke je prikazan na slici 4.9.



Slika 4.9: Slika prikazuje graf seta podataka i simboličan prikaz načina rada stabla odlučivanja (preuzeto sa [17]).

Pravi zadatak u izgradnji stabla odluke će biti pronalaženje optimalnih uvjeta za dobivanje točnih predviđanja. Kao što smo rekli stablo rekurzivno particionira prostor podataka postavljajući uvjete na značajke primjera s ciljem da napravi regije u

kojima odgovaraju podaci određene klase. To postižemo na način da tražimo podjelu na roditeljsku regiju te vrijedi

(4.16)

gdje je X set podataka iz , je uvjet i je indeks značajke na kojoj se primjenjuje uvjet . Podskupove u podjeli zovemo regijama djece. Da bismo odabrali optimalne podjele na svakom čvoru moramo definirati gubitak na regiji , odnosno nešto što nam govori koliko primjera smo krivo klasificirali u regiji ili koliko imamo "nečistoće" u regiji. Često korištene funkcije gubitka kod problema klasifikacije su gubitak unakrsne entropije (jednadžba 4.17) i gini indeks (jednadžba 4.18).

(4.17)

(4.18)

Ako imamo ukupno C klasa u nekoj regiji , definiramo kao omjer primjera u regiji R koji pripadaju klasi kao što predlaže jednadžba 4.19,

(4.19)

gdje je broj primjera u regiji koji pripadaju klasi , a ukupan broj primjera u regiji. Možemo vidjeti da je u obje funkcije (4.17 i 4.18) gubitak najmanji kada imamo samo primjere iz jedne klase u regiji, a najveći kada imamo podjednak broj primjera iz svake klase što indicira najveću "nečistoću" u regiji. Na svakom čvoru odabire se podjela koja daje najveću informacijsku dobit (eng. *information gain*) (jednadžba 4.20), to jest najveću razliku između gubitka roditeljske regije i gubitka regija djece.

— (4.20)

U jednadžbi 4.20 predstavlja ukupan broj regija djece, je broj primjera u regiji i N je broj primjera u regiji .

Bez uvođenja neke vrste regularizacije stablo bi moglo rasti primjenjujući navedene korake sve dok nemamo posebnu regiju za svaku točku podataka što možemo

smatrati preprilagođavanjem. Da izbjegnemo taj slučaj možemo koristiti sljedećih regularizacija: definiramo minimalan broj primjera u listovima, definiramo maksimalnu dubinu stabla ili definiramo maksimalan broj čvorova. Stablo odluke je algoritam koji se brzo može istrenirati, ali je sklono preprilagođavanju i uglavnom ima lošu preciznost predviđanja. Dobra vijest je da možemo smanjiti sklonost preprilagođavanju i povećati preciznost predviđanja koristeći ansamble stabala odluke.

U slučajnoj šumi treniramo više stabala odluke da bismo na kraju uzeli srednju vrijednost predviđanja svih stabala za konačno predviđanje. Svako stablo učimo na različitom setu podataka koji je jednake veličine kao i ulazni set ali se sastoji od nasumično odabranih primjera iz seta . se naziva "*bootstrap*" setom. Pošto su bootstrap setovi napravljeni od nasumično odabranih primjera iz imaju puno duplikata primjera i uglavnom sadrže oko dvije trećine primjera od ulaznog seta. Ako želimo imati stabala moramo imati i bootstrap setova. Sa bootstrap setova , , ..., treniramo svaki model sa setom te je konačno predviđanje za primjer jednako srednjoj vrijednosti predviđanja svih modela (jednadžba 4.21).

——————— (4.21)

Ovakav ansambl modela gdje koristimo srednju vrijednost predviđanja više modela koji su naučeni na bootstrap setovima se zove "*bagging*" (eng. *Bootstrap Aggregation*). Primjenom "bagginga" na stablo odluke dobivamo slučajnu šumu. Slučajna šuma ne koristi linearan model što je korisno kod podataka na kojima ne možemo dobro namjestiti linearne granice odluke. Iako slučajna šuma smanjuje varijancu stabla odluke te povećava preciznost predviđanja, također imamo i malo povećanje pristranosti jer svaki bootstrap set ne sadrži sve primjere iz ulaznog seta. Implementacija slučajne šume u programskom jeziku Python se može vidjeti u dodatku E.

4.5 Analiza glavnih komponenti

Analiza glavnih komponenti, skraćeno PCA (eng. *Principal component analysis*), je tehnika često korištena u statistici za vizualizaciju podataka ali spada i pod algoritme nenadziranog učenja. Algoritam ne radi nikakva predviđanja već mu je cilj transformirati visoko-dimenzionalne podatke ulaznog seta u reprezentaciju nižih dimenzija na način da što više očuvamo varijabilnost originalnog seta podataka (mi-

nimiziramo gubitak informacije iz početnog seta). To postižemo preko sljedećih koraka.

Za početak ulazni set X se centrira na nulu. To postižemo tako da izračunamo srednju vrijednost seta X i oduzmemo tu vrijednost od svake točke podatka u X. Nakon toga izračunamo matricu kovarijance centriranog seta X (jednadžba 4.22),

——— (4.22)

gdje je broj točaka podataka. sadrži varijancu svake značajke i kovarijancu svih parova značajki što nam daje informaciju kako promjena vrijednosti u jednoj značajki ovisi o promjeni vrijednosti u drugoj. Ako imamo značajki ima dimenzije . Nakon toga izračunamo svojstvene vektore i svojstvene vrijednosti za matricu kovarijance . Svojstvene vrijednosti dobivamo rješavanjem determinante u jednakosti 4.23,

(4.23)

gdje je jedinična matrica. Pripadne svojstvene vektore dobivamo tako da pojedinačno uvrštavamo sve dobivene svojstvene vrijednosti u jednadžbu 4.24,

(4.24)

Naravno, u praksi sve potrebne izračune za nas napravi računalo. U programskom jeziku Python dovoljno je pozvati jednu metodu iz biblioteke numpy da dobijemo željene svojstvene vrijednosti i svojstvene vektore. Svojstvene vektore onda sortiramo silazno prema svojstvenim vrijednostima. Svojstvene vektore sa najvećim svojstvenim vrijednostima nazivamo glavnim komponentama. Svojstvena vrijednost glavne komponente nam daje informaciju o vrijednosti varijance podataka projiciranih na tu komponentu što znači da su projicirani podaci najviše raširenи na prvoj glavnoj komponenti, pa na drugoj i tako dalje. Ovisno o odabiru dimenzija, na primjer , ulazni set projiciramo na novi prostor definiran sa prvih glavnih komponenti. To postižemo tako da matricu pomnožimo sa matricom transformacije (jednadžba 4.25).

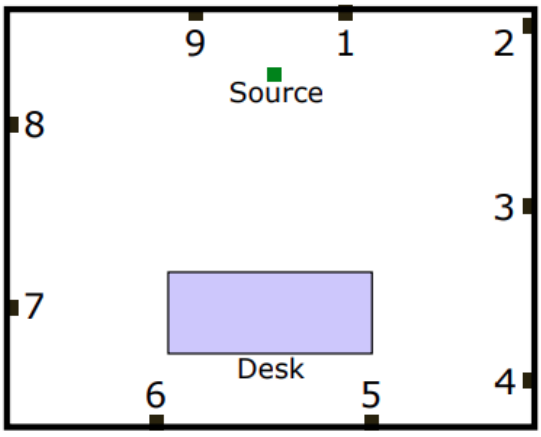
(4.25)

Ako je dimenzija , gdje je broj opažanja i originalan broj značajki, i dimenzija , gdje je broj glavnih komponenti, dobiti ćemo matricu dimenzija što rezultira u nisko-dimenzionalnoj reprezentaciji originalnog seta X.

Analiza glavnih komponenti je korisna za dvodimenzionalni ili trodimenzionalni prikaz visoko-dimenzionalnih podataka na kojem možemo bolje razumjeti cjelokupnu strukturu i odnose među podacima. Osim toga ako imamo jako šumovite podatke PCA može prepoznati i zadržati samo najbitnije uzorke koristeći glavne komponente povezane s visokom varijancom podataka. Na taj način smanjujemo utjecaj šuma na proces učenja drugih algoritama strojnog učenja. Implementacija analize glavnih komponenti u programskom jeziku Python se može vidjeti u dodatku F.

5 Eksperimentalni postav

Kao što smo rekli cilj ovog rada je ispitati mogućnost detekcije prisutnosti ljudi i preciznost detekcije broja osoba u elektromagnetskom polju. Eksperiment je postavljen na način da se izvor Wi-Fi zračenja nalazio unutar zatvorene prostorije. Na rubovima prostorije je postavljeno 9 detektora koji bilježe snagu signala u vremenu. Prikaz postava eksperimenta se može vidjeti na slici 5.1. Specifično, detektori bilježe vrijednosti



Slika 5.1: Slika prikazuje tlocrt eksperimentalnog postava (preuzeto sa [18]).

indikatora snage signala ili RSSI (eng. *Recieved signal strength indicator*) u vremenu. Indikator snage signala je vrijednost koja opisuje snagu signala. Veća vrijednost odgovara većoj snazi ali je vrijednost relativna jer ne postoji definirani odnos između

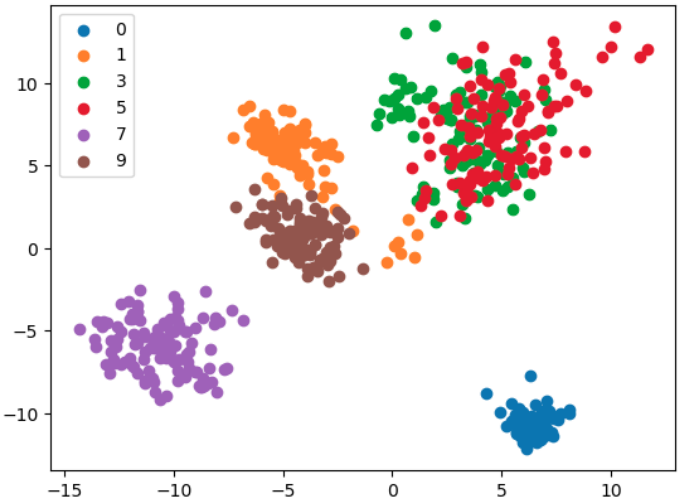
RSSI vrijednosti i snage u Watt-ima pa tako proizvođač Wi-Fi kartice određuje raspon vrijednosti i njezinu osjetljivost. Na RSSI vrijednost utječu razni faktori kao što su udaljenost između odašiljača i primatelja te sredstva koja se nalaze između njih. Ljudska aktivnost utječe na RSSI vrijednosti iz čega slijedi da će se na očitavanjima od detektora moći isto i primijetiti.

Mjerenja su trajala po 20 minuta za 1, 3, 5, 7 i 9 osoba u prostoriji. Uz ta mjerenja također su izmjerene RSSI vrijednosti kada u prostoriji nije bilo ljudi (mjerenje šuma).

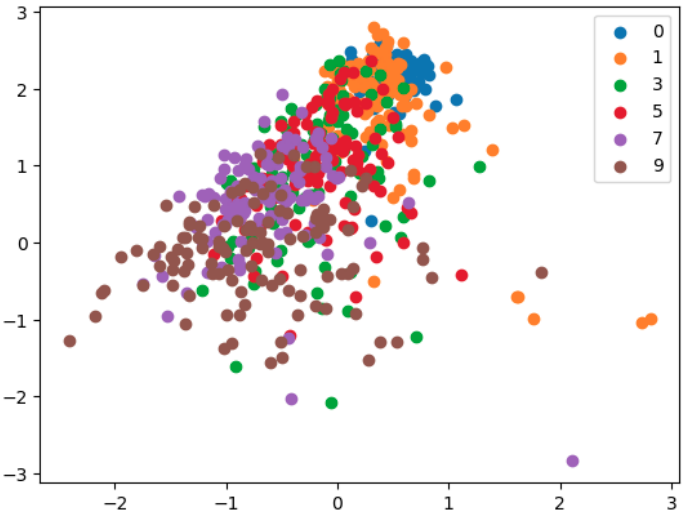
6 Priprema podataka

Izmjerene vrijednosti su zabilježene u 9 različitih ".csv" datoteka, gdje svaka sadrži snimljene vrijednosti pojedinog detektora, za 6 različitih mjerenja (za 0, 1, 3, 5, 7 i 9 osoba u prostoriji). Detektori mjere snagu signala na frekvenciji od 10 očitavanja vrijednosti u jednoj sekundi, što znači da ".csv" datoteke koje sadrže mjerenja od 20 minuta sadrže oko 12,000 redaka RSSI vrijednosti u paru sa pripadajućom vremenskom oznakom. Vremenske oznake zanemarujemo te kao značajke za jednu točku podataka uzimamo srednju vrijednost od 50 uzastopnih RSSI vrijednosti za svaki detektor. Pošto je u svakom mjerenju sudjelovalo 9 detektora imati ćemo ukupno 9 značajki. Ako nam jedan primjer obuhvaća 50 RSSI vrijednosti onda imamo 230 primjera za svako mjerenje te sveukupno 1380 primjera (u svakoj ".csv" datoteci koristimo prvih 11,500 vrijednosti pošto datoteke sadrže 11,500 - 12,000 izmjerenih vrijednosti osim datoteka koje sadrže izmjerene vrijednosti kada u prostoriji nije bilo ljudi). Naš set podataka na kraju čini matrica od 1380 redaka i 9 stupaca zajedno sa vektorom koji sadrži pripadajuću oznaku za svaki primjer (oznaka 0 kada nema ljudi, oznaka 1 za jednu osobu u prostoriji, itd.). Uz taj set podataka napravljena su još dva dodatna seta: u kojem su se kao značajke koristile standardne devijacije 50 uzastopnih RSSI vrijednosti i u kojem su se kao značajke koristile realne vrijednosti prvih 5 koeficijenata Fourierovog razvoja dobivenog iz 50 uzastopnih RSSI vrijednosti. Oba seta imaju isti broj primjera kao i set , ali dok set ima 9 značajki kao i set , set ima 45 značajki pošto koristimo 5 različitih vrijednosti da bi opisali izmjerene podatke svakog detektora. Koristeći algoritam analize glavnih komponenti dobivamo 2D prikaz setova gdje potencijalno

možemo uočiti uzorke i veze između podataka te klasteriranje podataka istih klasa (slike 6.1, 6.2 i 6.3). Priprema seta koristeći biblioteku pandas u programskom jeziku Python se može vidjeti u dodatku A.



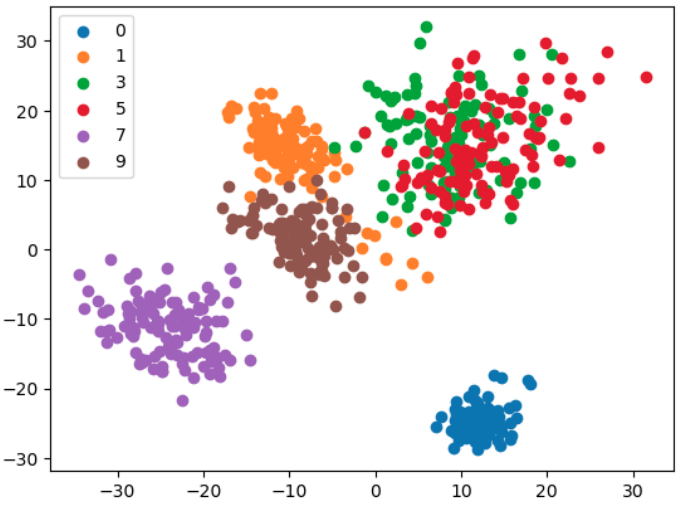
Slika 6.1: Slika prikazuje 2D prikaz točaka podataka i pripadne klase za set .



Slika 6.2: Slika prikazuje 2D prikaz točaka podataka i pripadne klase za set .

7 Rezultati i rasprava

Za prilagođavanje algoritama korištena je metoda -struke unakrsne provjere. Kod -struke unakrsne provjere set podataka se podijeli na jednakih podsetova gdje se podsetova koristi za treniranje algoritma, a preostali set se koristi kao testni set. Postupak se ponavlja puta na način da se svaki put koristi različit podset za



Slika 6.3: Slika prikazuje 2D prikaz točaka podataka i pripadne klase za set .

evaluaciju te se uspješnost algoritma evaluira usrednjavanjem preciznosti algoritama na testnom setu preko svih iteracija. Uspješnost algoritama navedenih u ovom radu je evaluirana za i . Primjenom ove metode uspijevamo na određeni način iskoristiti cijeli set podataka za treniranje i za evaluaciju. Primjeri seta podataka su nasumično pomiješani prije podjele na podsetove. Rezultati mjerenja za i su prikazani u tablicama 7.1 i 7.2. Iz dobivenih rezultata možemo vidjeti da su

Set podataka	LR	VLR	NM	SŠ
	100%	95%	96%	96%
	100%	90%	91%	93%
	95%	68%	72%	77%

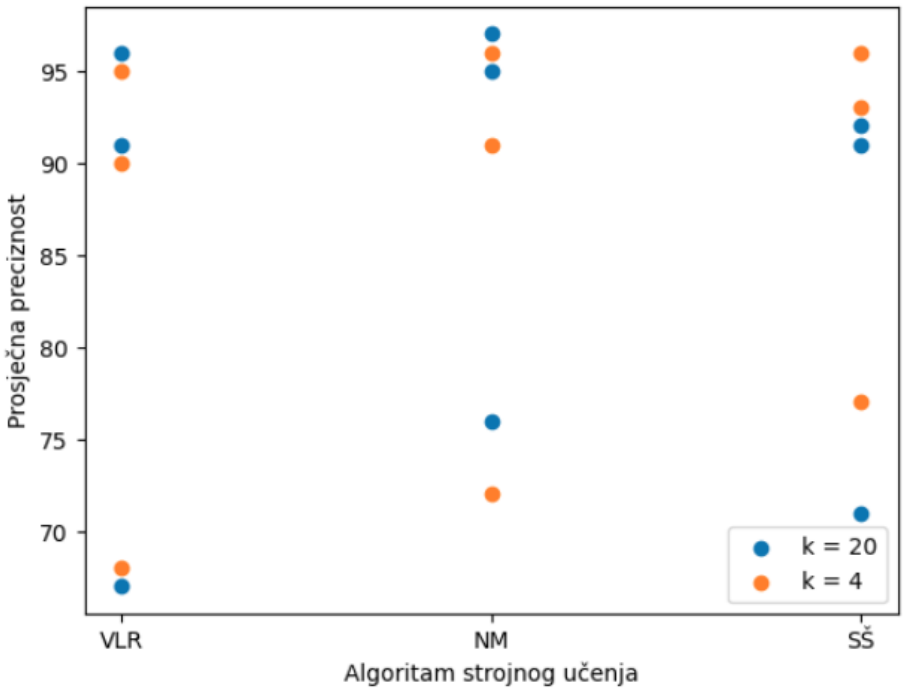
Tablica 7.1: Tablica prikazuje prosječnu preciznost detekcije točnog broja ljudi u prostoriji za podjelu.

Set podataka	LR	VLR	NM	SŠ
	100%	96%	97%	92%
	100%	91%	95%	91%
	95%	67%	76%	71%

Tablica 7.2: Tablica prikazuje prosječnu preciznost detekcije točnog broja ljudi u prostoriji za podjelu.

algoritmi višeklasne logističke regresije (VLR) i neuronske mreže (NM) bili uspješniji pri većoj vrijednosti dok je algoritam slučajne šume (SŠ) dobivao bolje rezultate za manju vrijednost. Za razliku od ostalih algoritama slučajna šuma pripada statičnim

modelima što znači da jednom kada se nauči na određeni set podataka model se ne može ažurirati. Stabla koja algoritam izgradi za svaki podset se nikada ne mijenjaju već se samo dodavaju nova stabla za svaki sljedeći podset. Očekivano je onda da će algoritam imati veću uspješnost za manju vrijednost pošto iz istog slijedi da će podsetovi biti veći i time će pojedina stabla imati više podataka za donošenje odluke. Prosječne preciznosti pojedinog modela se mogu vidjeti na grafu prikazanom na slici 7.1.



Slika 7.1: Graf prikazuje prosječne preciznosti algoritama za dvije različite podjele seta podataka.

Prema grafičkim prikazima do kojih smo uspjeli doći koristeći analizu glavnih komponenti (slike 6.1, 6.2 i 6.3) možemo vidjeti da se uzorci seta znatno razlikuju od uzoraka ostala dva. Samom usporedbom grafičkog prikaza seta sa prikazom ostala dva seta možemo zaključiti da će korištenje standardne devijacije davati najlošije rezultate. Za razliku od setova i , u setu ne možemo vidjeti odvojeno klasteriranje različitih klasa već su sve klase većinom međusobno pomiješane u jedan veliki klaster što algoritmima otežava posao kada moraju svrstati primjer u određenu klasu na temelju vrijednosti značajki. Algoritmi ipak uspijevaju u većini slučajeva točno predvidjeti klasu primjera pošto povlače granice odluke u prostoru sa devet dimenzija (jer ulazni set originalno ima 9 značajki) gdje je razlika između dviju različitih klasa uočljivija, ali u usporedbi sa rezultatima na setovima

i jasno je da je i u višim dimenzijama algoritme teško prilagoditi podacima. Iz redukcije dimenzionalnosti seta i nije očito koji set bi davao bolje rezultate pošto grafovi prikazuju veoma slične uzorke, ali možemo uočiti da su točke podataka za pojedinu klasu manje raspršene kod seta što čini posao modela lakšim u nalaženju optimalne granice odluke. Iz grafova također možemo vidjeti da su RSSI vrijednosti prikupljene za 3 osobe i 5 osoba u prostoriji veoma slične što se vidi i u krivim predviđanjima algoritama od kojih se velika većina (u puno slučajeva i sva kriva predviđanja) sastoji od klasifikacije primjera u klasu 5 umjesto klase 3 i obrnuto.

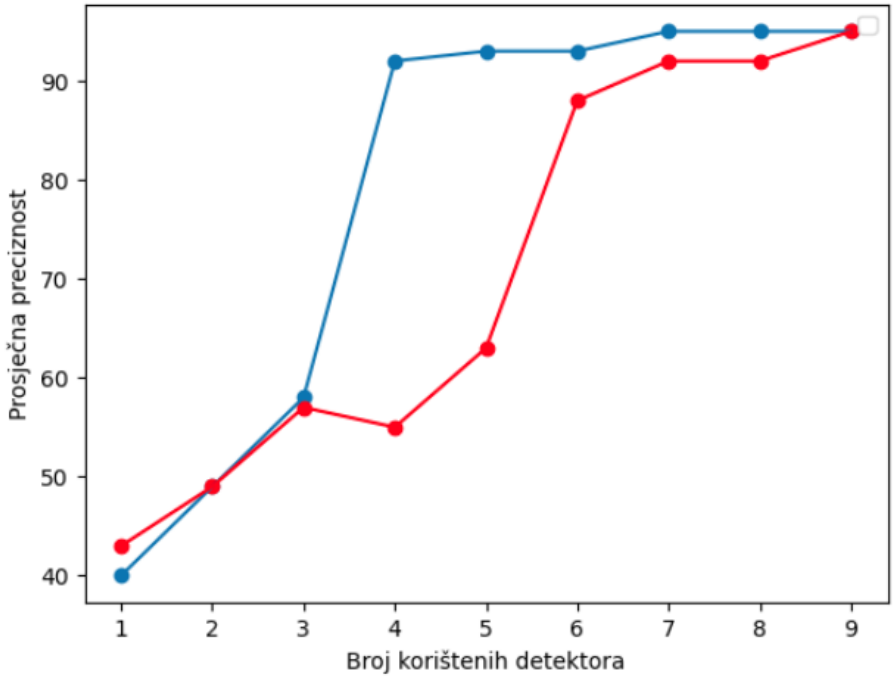
Nakon istraživanja preciznosti pojedinih algoritama proučavalo se i kako preciznost detekcije broja osoba u prostoriji ovisi o broju korištenih detektora. Detektori su numerirani brojevima od 1 do 9 (ESP1, ESP2, ..., ESP9) te se nalaze na različitim mjestima u prostoriji (slika 5.1). Za početak se mjerila preciznost predviđanja točnog broja osoba u prostoriji koristeći samo podatke izmjerene od jednog detektora. Rezultate istraživanja prikazuje tablica 7.3. Ponašanje preciznosti ovisno o broju korištenih

Detektor	Prosječna preciznost
ESP1	40%
ESP2	33%
ESP3	38%
ESP4	76%
ESP5	44%
ESP6	34%
ESP7	41%
ESP8	43%
ESP9	43%

Tablica 7.3: Tablica prikazuje prosječnu preciznost detekcije točnog broja ljudi u prostoriji za pojedini detektor.

detektora se može vidjeti na grafu prikazanom na slici 7.2. Naravno, vrijednosti se mogu mijenjati za različite kombinacije detektora. Korištenjem podataka od različitih kombinacija dvaju detektora dobivena je najveća prosječna preciznost od 89% za detektore ESP1 i ESP4. Za kombinaciju od tri detektora dobivena je najveća prosječna preciznost od 92% za detektore ESP1, ESP4 i ESP7. Lako možemo vidjeti da je detektor ESP4 imao ključnu ulogu u detekciji točnog broja osoba u prostoriji što se može uočiti i u naglom porastu preciznosti plave i crvene krivulje na grafu (slika 7.2). Također, preciznosti kombinacija od tri ili dva detektora koje nisu uključivale ESP4

su poprimale vrijednosti između 40% i 70%. Na temelju dobivenih rezultata mogli bi zaključiti da su se aktivnosti osoba tijekom mjerenja većinom odvijale u blizini detektora ESP4 ili na spojnici između izvora i detektora ESP4. Navedene preciznosti u ovom dijelu istraživanja dobivene su koristeći algoritam višeklasne logističke regresije. Podaci su pripremljeni na isti način kao i set osim što se koristilo više srednjih vrijednosti dobivenih iz 50 uzastopnih RSSI vrijednosti kao značajke pojedinog detektora da postignemo veću dimenzionalnost podataka pri korištenju malog broja detektora. Sa većim brojem značajki model lakše prepoznaje razliku između dva primjera različitih klasa kada imamo više informacija o njima. Prosječna preciznost je dobivena uporabom k-strukom unakrsnom provjerom za .



Slika 7.2: Graf prikazuje ovisnost preciznosti detekcije točnog broja osoba o broju korištenih detektora. Plava krivulja prikazuje kombinacije detektora od 1 do 9 dok crvena krivulja prikazuje kombinacije od 9 do 1.

8 Zaključak

U radu se istražuje mogućnost i preciznost detekcije ljudi u Wi-Fi elektromagnetskom polju s izvorom unutar prostorije u kojoj su postavljeni detektori. Detektori zapisuju snagu signala u vremenu u obliku RSSI vrijednosti. U radu su opisani fizikalni koncepti koji čine ovakvu vrstu detekcije mogućom zajedno sa algoritmima strojnog učenja koji omogućuju realizaciju detekcije. Nakon toga je uspješnost algoritama ispitana za različite načine pripreme podataka i za različite kombinacije detektora korištenih za prikupljanje podataka.

Preciznost od 97% je najveća točnost predviđanja broja ljudi u prostoriji koja je ostvarena u ovom radu sa algoritmom neuronske mreže koja je implementirana u Pythonu koristeći uglavnom samo biblioteku numpy i za optimizaciju parametara je implementirana osnovna verzija gradijentnog spusta. Svi algoritmi su za set davali zadovoljavajuće rezultate. Za još preciznije rezultate predlaže se uporaba naprednijih metoda optimizacije parametara kao što su ADAM i RMSprop te korištenje biblioteka specijaliziranih za strojno učenje kao što su pytorch i tensorflow. Nadalje, iz korištenih ulaznih setova možemo vidjeti da su algoritmi veoma osjetljivi na način pripreme podataka te je odabir prikladne reprezentacije prikupljenih podataka ključno za proces učenja algoritma. Također, s obzirom na prirodu eksperimenta mogli bi se postići još bolji rezultati korištenjem neuronske mreže sa povratnom vezom koja je specijalizirana za rad na sekvenciranim podacima.

Dodaci

Dodatak A Priprema podataka

```
X = np.zeros([1380, 9])
y = []
class_start = 0

for n in tqdm([0, 1, 3, 5, 7, 9]):
    folder_path = 'C:/Users/Nikola/Downloads/19_10_2022/'+str(n)+'_O'
    dataframes = []

    for filename in os.listdir(folder_path):
        if filename.endswith('.xlsx'):
            file_path = os.path.join(folder_path, filename)
            df = pd.read_excel(file_path)
            df=df.drop(["time"], axis=1)
            dataframes.append(df)

    detector = 0
    for dt in dataframes:
        whole_table = dt.to_numpy()
        sample_start = 0
        for i in range(class_start, class_start + 230):
            if(detector == 0):
                y.append(n)

            X[i][detector] = np.mean(whole_table[sample_start : sample_start + 50])

            if (n == 0):
                sample_start = np.random.randint(210000)
            else:
                sample_start += 50

        detector += 1

    class_start += 230

y = np.array(y)
```

Dodatak B Implementacija algoritma Logistička regresija

```
class LogisticRegression:
    def __init__(self, learning_rate=0.1, n_epochs=2000, b_size = 30):
        self.lr = learning_rate
        self.n_epochs = n_epochs
        self.batch_size = b_size
        self.w = None
        self.b = None

    def _init_weights_bias(self, X):
        n_features = X.shape[1]
        self.w = np.zeros(n_features)
        self.b = 0

    def _hypothesis(self, x):
        return 1/(1 + np.exp(-(np.dot(x, self.w) + self.b)))

    def _get_gradients(self, X, y):
        dw = (1/X.shape[0])*np.dot(X.T, (y - self._hypothesis(X)))
        db = (1/X.shape[0])*np.sum(y - self._hypothesis(X))
        return dw, db

    def _update_weights_bias(self, dw, db):
        self.w += self.lr * dw
        self.b += self.lr * db

    def fit(self, X, y):
        self._init_weights_bias(X)
        n_samples = X.shape[0]
        n_batches = n_samples // self.batch_size

        for epoch in range(self.n_epochs):
            shuffled_indexes = np.random.permutation(n_samples)
            X = X[shuffled_indexes]
            y = y[shuffled_indexes]

            for batch in range(n_batches):
                start = batch * self.batch_size
                end = start + self.batch_size
                X_batch = X[start:end]
                y_batch = y[start:end]

                dw, db = self._get_gradients(X_batch, y_batch)
                self._update_weights_bias(dw, db)

    def predict(self, X):
        estimate = np.dot(X, self.w) + self.b
        prediction = np.sign(estimate)
        return np.where(prediction == -1, 0, 1)

    def calc_accuracy(self, y_hat, y):
        correct_predictions = sum(p == a for p, a in zip(y_hat, y))
        total_predictions = len(y_hat)

        accuracy = correct_predictions / total_predictions
        return accuracy

    def get_params(self):
        return self.w, self.b
```

Dodatak C Implementacija algoritma Višeklasna logistička regresija

```
class SoftmaxRegression:
    def __init__(self, learning_rate=0.01, n_epochs=5000, b_size = 30):
        self.lr = learning_rate
        self.n_epochs = n_epochs
        self.batch_size = b_size
        self.w = None
        self.b = None

    def _init_weights_bias(self, X, y):
        n_features = X.shape[1]
        self.classes = max(y) + 1
        self.w = np.zeros([self.classes, n_features])
        self.b = np.zeros(self.classes)

    def _make_hot(self, y):
        Y = np.zeros([len(y), max(y) + 1])
        Y[np.arange(len(y)), y] = 1
        return Y

    def _hypothesis(self, X):
        e = np.exp(np.dot(X, self.w.T) + self.b)
        return e/np.sum(e, axis = 1, keepdims = True)

    def _get_gradients(self, X, Y):
        dw = np.dot((Y - self._hypothesis(X)).T, X)
        db = np.sum((Y - self._hypothesis(X)), axis = 0)
        return dw, db

    def _update_weights_bias(self, dw, db):
        self.w += self.lr * dw
        self.b += self.lr * db

    def fit(self, X, y):
        self._init_weights_bias(X, y)
        Y = self._make_hot(y)
        n_samples = X.shape[0]
        n_batches = n_samples // self.batch_size

        for epoch in range(self.n_epochs):
            shuffled_indexes = np.random.permutation(n_samples)
            X = X[shuffled_indexes]
            Y = Y[shuffled_indexes]

            for batch in range(n_batches):
                start = batch * self.batch_size
                end = start + self.batch_size
                X_batch = X[start:end]
                Y_batch = Y[start:end]

                dw, db = self._get_gradients(X_batch, Y_batch)
                self._update_weights_bias(dw, db)

    def predict(self, X):
        return np.argmax(self._hypothesis(X), axis=1)

    def calc_accuracy(self, y_pred, y):
        correct_predictions = sum(p == a for p, a in zip(y_pred, y))
        total_predictions = len(y_pred)

        accuracy = correct_predictions / total_predictions
        return accuracy
```

Dodatak D Implementacija algoritma Neuronska mreža

```
class NeuralNetwork:
    def __init__(self, learning_rate = 0.01, n_epochs=3000, b_size = 30,*, plot=False):
        self.lr = learning_rate
        self.n_epochs = n_epochs
        self.batch_size = b_size
        self.loss_values = []
        self.plot = plot

    def _init_params(self, X, y):
        self.w1 = 0.1*np.random.randn(50, X.shape[1])
        self.b1 = 0.1*np.random.randn(1, 50)
        self.w2 = 0.1*np.random.randn(max(y) + 1, 50)
        self.b2 = 0.1*np.random.randn(1, max(y) + 1)

    def _ReLU(self, Z):
        return np.maximum(0, Z)

    def _dReLU(self, Z):
        return Z > 0

    def _softmax(self, Z):
        return np.exp(Z)/np.sum(np.exp(Z), axis=1, keepdims=True)

    def _make_hot(self, y):
        Y = np.zeros([len(y), max(y) + 1])
        Y[np.arange(len(y)), y] = 1
        return Y

    def _forward_prop(self, X):
        Z1 = np.dot(X, self.w1.T) + self.b1
        A1 = self._ReLU(Z1)
        Z2 = np.dot(A1, self.w2.T) + self.b2
        A2 = self._softmax(Z2)
        return Z1, A1, Z2, A2

    def _back_prop(self, Z1, A1, Z2, A2, X, Y):
        n = len(y)
        dZ2 = A2 - Y
        dw2 = 1/n * np.dot(dZ2.T, A1)
        db2 = 1/n * np.sum(dZ2, axis = 0)
        dZ1 = np.dot(dZ2, self.w2)*self._dReLU(Z1)
        dw1 = 1/n * np.dot(dZ1.T, X)
        db1 = 1/n * np.sum(dZ1, axis = 0)
        return dw1, db1, dw2, db2

    def _update_params(self, dw1, db1, dw2, db2):
        self.w1 -= self.lr*dw1
        self.b1 -= self.lr*db1
        self.w2 -= self.lr*dw2
        self.b2 -= self.lr*db2

    def fit(self, X, y):
        self._init_params(X, y)
        Y = self._make_hot(y)

        n_samples = X.shape[0]
        n_batches = n_samples // self.batch_size

        for epoch in range(self.n_epochs):
            shuffled_indexes = np.random.permutation(n_samples)
            X = X[shuffled_indexes]
            Y = Y[shuffled_indexes]

            if(self.plot):
                Z1, A1, Z2, A2 = self._forward_prop(X)
```

```
            self.loss_values += [self._calc_loss(Y, A2)]

        for batch in range(n_batches):
            start = batch * self.batch_size
            end = start + self.batch_size
            X_batch = X[start:end]
            Y_batch = Y[start:end]

            Z1, A1, Z2, A2 = self._forward_prop(X_batch)
            dw1, db1, dw2, db2 = self._back_prop(Z1, A1, Z2, A2, X_batch, Y_batch)
            self._update_params(dw1, db1, dw2, db2)

    if(self.plot):
        self._plot_loss()

def predict(self, X):
    Z1, A1, Z2, A2 = self._forward_prop(X)
    return np.argmax(A2, axis=1)

def _calc_loss(self, Y, Y_hat):
    epsilon = 1e-15  # Small value to avoid division by zero
    Y_hat = np.clip(Y_hat, epsilon, 1 - epsilon)  # Clip probabilities to avoid log(0)
    loss = -np.sum(Y * np.log(Y_hat)) / len(Y)
    return loss

def _plot_loss(self):
    iterations = list(range(1, len(self.loss_values) + 1))
    plt.plot(iterations, self.loss_values, marker='o')
    plt.title('Loss Function Value Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss Value')
    plt.grid(True)
    plt.show()

def calc_accuracy(self, y_hat, y):
    correct_predictions = sum(p == a for p, a in zip(y_hat, y))
    total_predictions = len(y_hat)

    accuracy = correct_predictions / total_predictions
    return accuracy
```

Dodatak E Implementacija algoritma Slučajna šuma

```
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None,*,label=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.label = label

    def is_leaf_node(self):
        return self.label is not None


class DecisionTree:
    def __init__(self, min_samples_split=10, max_depth=7, n_features=None):
        self.min_samples_split=min_samples_split
        self.max_depth=max_depth
        self.n_features=n_features
        self.root=None

    def fit(self, X, y):
        self.n_features = X.shape[1] if not self.n_features else min(X.shape[1],self.n_features)
        self.root = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_feats = X.shape
        n_labels = len(np.unique(y))

        # check the stopping criteria
        if (depth>=self.max_depth or n_labels==1 or n_samples<self.min_samples_split):
            leaf_label = self._most_common_label(y)
            return Node(label=leaf_label)

        feat_idxs = np.random.choice(n_feats, self.n_features, replace=False)

        # find the best split
        best_feature, best_thresh = self._best_split(X, y, feat_idxs)

        # create child nodes
        left_idxs, right_idxs = self._split(X[:, best_feature], best_thresh)
        left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth+1)
        right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth+1)
        return Node(best_feature, best_thresh, left, right)


    def _best_split(self, X, y, feat_idxs):
        best_gain = -1
        split_idx, split_threshold = None, None

        for feat_idx in feat_idxs:
            X_column = X[:, feat_idx]
            thresholds = np.unique(X_column)

            for thr in thresholds:
                # calculate the information gain
                gain = self._information_gain(y, X_column, thr)

                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_threshold = thr

        return split_idx, split_threshold


    def _information_gain(self, y, X_column, threshold):
```

```
        # parent entropy
        parent_entropy = self._entropy(y)

        # create children
        left_idxs, right_idxs = self._split(X_column, threshold)

        if len(left_idxs) == 0 or len(right_idxs) == 0:
            return 0

        # calculate the weighted avg. entropy of children
        n = len(y)
        n_l, n_r = len(left_idxs), len(right_idxs)
        e_l, e_r = self._entropy(y[left_idxs]), self._entropy(y[right_idxs])
        child_entropy = (n_l/n) * e_l + (n_r/n) * e_r

        # calculate the IG
        information_gain = parent_entropy - child_entropy
        return information_gain

    def _split(self, X_column, split_thresh):
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        return left_idxs, right_idxs

    def _entropy(self, y):
        hist = np.bincount(y)
        ps = hist / len(y)
        return -np.sum([p * np.log(p) for p in ps if p>0])


    def _most_common_label(self, y):
        counter = Counter(y)
        label = counter.most_common(1)[0][0]
        return label

    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x in X])

    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.label

        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)


class RandomForest:
    def __init__(self, n_trees = 20, max_depth = 7, min_samples_split = 30, n_feature = None):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.n_features = n_feature
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(max_depth = self.max_depth,
                                min_samples_split = self.min_samples_split,
                                n_features = self.n_features)
            X_sample, y_sample = self._bootstrap_samples(X, y)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def _bootstrap_samples(self, X, y):
        n_samples = X.shape[0]
        idxs = np.random.choice(n_samples, n_samples, replace=True)
```

```
        return X[idxs], y[idxs]

    def _most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(predictions, 0, 1)
        predictions = np.array([self._most_common_label(pred) for pred in tree_preds])
        return predictions

    def calc_accuracy(self, y_hat, y):
        correct_predictions = sum(p == a for p, a in zip(y_hat, y))
        total_predictions = len(y_hat)

        accuracy = correct_predictions / total_predictions
        return accuracy
```

Dodatak F Implementacija algoritma Analiza glavnih komponenti

```
class PCA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        # mean centering
        self.mean = np.mean(X, axis=0)
        X = X -  self.mean

        # covariance
        cov = np.cov(X, rowvar = False)

        eigenvalues, eigenvectors = np.linalg.eig(cov)

        # sort eigenvectors
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvectors = eigenvectors[:, idxs]

        self.components = eigenvectors[:, :self.n_components]

    def transform(self, X):
        # projects data
        X = X - self.mean
        return np.dot(X, self.components)
```

Literatura

[1] Ma, Yongsen, "Improving Wifi Sensing And Networking With Channel State Information" (2019). *Dissertations, Theses, and Masters Projects.* William & Mary. Paper 1593091976. http://dx.doi.org/10.21220/s2-dwgg-3j27 .

[2] Geng, J., Huang, D., & De la Torre, F. (2022). DensePose From WiFi. https://arxiv.org/abs/2301.00250.

[3] Y. Yuan, C. Qiu, W. Xi and J. Zhao, Crowd Density Estimation Using Wireless Sensor Networks, Seventh International Conference on Mobile Ad-hoc and Sensor Networks, pp. 138-145 (2011).

[4] https://edutorij-admin-api.carnet.hr/storage/extracted/1872721/elektromagnetski-val.html

[5] https://en.wikipedia.org/wiki/Electric_field

[6] Young, H. D., Freedman, R. A., Ford, A. L.,, Sears, F. W. (2012). *Sears and Zemansky's University Physics: with Modern Physics*. San Francisco: Pearson Addison-Wesley. ISBN: 9780321696861 0321696867 9780321696854 0321696859 9780321762184 0321762185 9780321741264 0321741269 9780321696229 0321696220

[7] https://physics.stackexchange.com/questions/712884/what-would-be-the-shape-of-the-magnetic-field-around-a-current-carrying-wire-

[8] https://theory.labster.com/snell-law/

[9] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, David J. Schwab, A high-bias, low-variance introduction to Machine Learning for physicists, Physics Reports, Volume 810, 2019, Pages 1-124, ISSN 0370-1573, https://www.sciencedirect.com/science/article/pii/S0370157319300766.

[10] https://www.javatpoint.com/overfitting-in-machine-learning

[11] https://poissonisfish.com/2023/04/11/gradient-descent/

[12] https://en.wikipedia.org/wiki/Sigmoid_function

[13] https://www.fer.unizg.hr/_download/repository/SU1-2021-P07-LogistickaRegresija2.pdf

[14] https://www.cosmos.esa.int/web/machine-learning-group/neural-network-introduction

[15] https://www.fer.unizg.hr/_download/repository/01-Uvod-1s.pdf

[16] https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI

[17] https://www.jeremyjordan.me/decision-trees/

[18] Jukić, D., Domazet, S., Ivanko, A., Raca, D., Nikolić, S., Knežević, M., ... & Buljan, H. (2023). Determining the presence and the number of people by using a Wi-Fi signal. https://arxiv.org/abs/2308.06773v1.