Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku i informatiku

Biljana Malinović

Neki metodi za klasifikaciju tačaka u trodimenzionalnom prostoru

-MASTER RAD-

Mentor: dr Ivana Štajner - Papuga

Novi Sad, 2018.

SADRŽAJ

UVOD

Sistem koji je baziran na veštačkoj inteligenciji jeste onaj sistem koji ima mogućnost da shvati svoje okruženje i da samostalno, ili od strane uticaja okruženja, sprovede određene akcije koje dovode do najveće verovatnoće kompletnog i optimalnog rešavanja specificiranog cilja tog sistema. Ovakvi sistemi su sistemi koji su sposobni da uče iz priloženog skupa podataka ili od strane prethodnih algoritama i iskustava. Ove akcije učenja i samostalnog rešavanja problema ustvari predstavljaju inteligenciju u mašinama i uklanjaju potrebu da se mašine dizajniraju za rešavanje eksplicitnih problema određenim algoritmima.

Sistemi bazirani na veštačkoj inteligenciji postaju sve korisniji što je svet kompleksniji. U poslednje dve decenije postignut je neverovatan napredak u poljima računarskih nauka i veštačke inteligencije. Discipline kao što su *Deep Learning,* Fazi Logika i *Data Mining,* i inteligentni agenti kao što su IBM-ov *Watson* ili Apple-ov *Siri*, predstavljaju osnovu za sisteme koji pružaju servise koji moraju biti zasnovani na nekom obliku inteligencije i kreativnosti. Razvojem ovih sistema i disciplina, sve je manji broj kompanija koje mogu da opstanu bez inteligentnih sistema koji će optimizovati njihove poslove ili minimizovati njihove troškove.

Cilj ovog rada jeste upoznavanje sa osnovnim i često korišćenim metodologijama i algoritmima u oblastima veštačke inteligencije i njihovim primenama u stohastičkim neprekidnim modelima.

U prvoj glavi dat je prikaz razvoja veštačke inteligencije kroz istoriju, u cilju razumevanja značaja veštačke inteligencije u svetu kao i izvanrednog potencijala za poboljšanje i rešavanje problema u praksi. Literatura korišćena u ovom poglavlju je [13].

U drugoj glavi dat je prikaz klasifikacionih pocedura i regresije. To su postupci koji predstavljaju osnovne metode u domenu veštačke inteligencije. U okviru regresije definisana je linearna i nelinearna regresija. Više o ovim temama se može naći u [1], [4] ,[8], [9], [10] , [13] i [17].

U trećoj glavi predstavljen je „naivni“ Bejz klasifikator (eng. „naive“ Bayes). Dati su mogući modeli ovog klasifikatora kao i način na koji se konstruiše. Objašnjena je uslovna verovatnoća, Bejzova teorema i uslovna nezavisnost koja čini ovaj klasifikator „naivnim”. U okviru ovog poglavlja prikazani su i slučajevi u praksi gde je pogodno koristiti ovaj model klasifikatora. Za izradu ovog poglavlja korišćena je literatura [1], [10], [11],[13] ,[16] ,[18].

Četvrta glava se bavi problemom k-najbližih suseda (eng. *K-Nearest Neighbors*), skraćeno *KNN,* metodologijom za klasifikaciju. Predstavljena je ideja samog algoritma k najbližih suseda kao i značaj parametra k, i efikasnosti i brzine ovog algoritma, zajedno sa načinima za računanje distanci ulaznog podatka za klasifikovanje i korišćenje algoritma u svrhu regresije. Detaljnije objašnjenje ovih tema može se naći u [3], [4], [6],[13], [14], [15].

Peta glava je vezana za veštačke neuronske mreže (eng. artificial neural networks), po mnogima osnova veštačke inteligencije. Ovo poglavlje opisuje kako je došlo do ideje kreiranja veštačkih neuronskih mreža, kao i same komponente i strukturu veštačkih neuronskih mreža. Objašnjen je koncept nadgledanog (eng. *supervised*) i nenadgledanog (eng. *unsupervised*) učenja sistema koji je baziran na veštačkim neuronskim mrežama i objašnjen je algoritam ulančavanja unazad (eng. *backpropagation*]) za učenje neuronskih mreža koji je zasnovan na parcijalnim, diferencijalnim jednačinama. Ovo poglavlje je napisano na osnovu [5], [7], [9], [13].

U šestoj glavi ilustrovan je postupak klasifikovanja tačaka u trodimenzionalnom prostoru pomoću prethodno navedenih metoda: „naivni“ Bejz, klasifikatorom k-najbližih suseda i neuronskim mrežama. Tom prilikom korišćen je programski paket MATLAB*. Dat je prikaz odgovarajućih komandi i procedura, kao i originalnih skripti napisanih upravo za potrebe ovog rada.

*LICENCA ZA MATLAB 577200, MATLAB&SIMULINK R2009

1. ISTORIJA VESTAČKE INTELIGENCIJE

Sredinom dvadesetog veka započeti su prvi projekti sa ciljem da se konstruiše veštački mozak koji ima mogućnost učenja i rezonovanja. Kako bi ovakva veštačka inteligencija postala realnost, potrebna je inteligencija i materijalni entitet koji bi sadržao i koristio inteligenciju u svrhu rešavanja problema. Moderni digitalni električni računar je bio odabran kao entitet koji ima najveću verovatnoću da ispuni ovaj cilj i demonstrira inteligenciju. Prvi operabilni računarski sistem nastao je za vreme drugog svetskog rata 1940. godine, napravljen od strane tima Alana Tjuringa (*Alan Turing*), u svrhu dešifrovanja nemačkih poruka. 1941. godine nastao je prvi računar koji je bio programabilan, sredinom dvadesetog veka nastao je i prvi računar opšte namene ENIAC, a 1945. napisan je *Plankakul,* prvi programski jezik visokog nivoa. 1952. godine proizveden je IBM 701, prvi računar koji je doneo profit svom proizvođaču. Ovo je značajan momenat u istoriji razvijanja veštačke inteligencije jer je započelo razvijanje i masovno korišćenje kako računarskih sistema tako i softvera i servisa koji su vezani za ove sisteme. Svakom generacijom računarskog hardvera sistemi postaju sve brži, sve je više memorijskog prostora dostupno, a cene korišćenja ovih sistema opadaju. Ovakvi sistemi, sa mogućnostima rapidnog računanja, rešavanja kompleksnih skupova instrukcija i skladištenja ogromnih količina informacija predstavljaju odličan izbor za nošenje i korišćenje inteligencije i oponašanje ljudskog mozga. Izlaganje ovog poglavlja oslanja se na literaturu iz [13].

Rad koji je generalno prihvaćen kao prvi rad vezan za implementaciju sistema baziranog na veštačkoj inteligenciji je urađen od strane Vorena Mekuloha (*Warren McCulloch*) i Voltera Pitsa (*Walter Pitts*) 1943. Godine. Njihov rad je bio baziran na istraživanjima iz polja : psihologije i neurologije o tome kako neuroni funkcionišu u ljudskom mozgu, informacione teorije o opisivanju digitalnih signala koji su uključeni ili isključeni („*sve ili ništa*“ signali), propozicione logike i Tjuringove teorije računanja, videti [13].

Predstavili su strukturu veštačke neuronske mreže u kojoj je svaki neuron karakterisan signalom koji govori da li je neuron u uključenom ili isključenom stanju. Ova stanja predstavljala su ekvivalent propozicije koju jedan neuron daje, koja je direktno vezana za propozicije koju je taj neuron dobio kao

stimulaciju od strane neurona koji su u vezi sa njim. Ovim neuronskim mrežama pokazali su da je moguće izračunati vrednost bilo koje funkcije koja se može izračunati korektnom raspodelom neurona u mreži. Takođe su pokazali da je moguće promeniti ishod proračuna već postojeće arhitekture neuronske mreže promenom vrednosti težina na konekcijama između neurona, a samim tim i mogućnost učenja neuronskih mreža.

Njihov rad bio je osnova uspešnog razvitka veštačke inteligencije od 1952. godine do 1969. godine. Iako računari u to doba nisu bili hardverski i softverski moćni kao današnji računari, naučnici su uspešno razvijali programe koji su imali mogućnost da samostalno određuju poteze u partiji šaha. 1956. godine u Dartmautu (*Dartmouth*) organizovana je konferencija i radionica za naučnike koji su bili zainteresovani ovim naukama, videti [13]. Na ovoj konferenciji upoznali su se naučnici sa raznih univerziteta i razmenjivali su ideje i implementacije svojih inteligentnih sistema. Ova konferencija nije proizvela nova saznanja ili nove izume vezane za veštačku inteligenciju, ali jeste dovela do zvaničnog osnivanja veštačke inteligencije kao akademske discipline.

Glavni problem sa kojim su se suočavali svi naučnici i inženjeri koji su se bavili računarskim naukama i veštačkom inteligencijom jeste rešavanje problema bilo koje opštosti korišćenjem mašina koje implementiraju pretrage. Ove mašine bile bi snabdevene elementarnim informacijama iz mnogih naučnih disciplina i pokušavale bi da spajaju činjenice ovih disciplina kako bi našle potpuno rešenje određenog problema. Veliki izuzetak ovakvog pristupa jeste što se pretrage nisu razvijale porastom kompleksnosti problema – bile su dovoljne za rešavanje problema koji su bili bliski snabdevenim informacijama. Jedini način rešavanja ove prepreke jeste sužavanje skupa informacija, tako da skup što detaljnije opisuje jednu naučnu disciplinu, predstavljajući eksperta te discipline. Ovakvi sistemi nazivani su sistemi bazirani na znanju (eng. *Knowledge Based Systems - KBS*) ili ekspertski sistemi. Glavni atributi ekspertskih sistema su korišćenje *baza znanja* koja je formirana od činjenica iz kojih se mogu formirati nove, složenije činjenice i veze između činjenica i korišćenje pravila uz pomoć kojih ekspertski sistem obrađuje i testira istinitost ulaznih teorija. Ekspertski sistemi bili su veoma uspešni u periodu između 1969. godine i 1979. godine u poljima hemije, medicine, geologije kao i lingvistike. Ekspertski sistemi kao što su MYCIN, DENDRAL i PROSPECTOR imali su veliki značaj kako za razvoj veštačke inteligencije tako i za rešavanje praktičnih problema u realnom svetu,videti [13]. Uspešni rezultati ekspertskih sistema kreirali su zlatno doba veštačke inteligencije u periodu od 1980. godine do 1990. godine, u kom su velike firme koristile ove sisteme kako bi

uštedele resurse koji bi inače bili potrebni za upošljavanje ljudskih eksperata u svrhu rešavanja problema i donošenja kritičnih odluka.

Od 1990. godine ispostavilo se mnogo isplativije da se vrše istraživanja na postojećim teorijama u polju veštačke inteligencije nego da se radi na novim izumima i teorijama, kao i da je bolje da se eksperimentisanje sa inteligentnim sistemima vrši na aplikacijama u realnom svetu nego na zatvorenim i kontrolisanim primerima. Unapređenje tehnologija i uređaja sa kojima se može implementirati veštačka inteligencija je dovelo do znatnog napretka u poljima prepoznavanja slika, konkretno prepoznavanje pisanih slova. Prepoznavanje slika po karakteristikama slike ili po sličnosti slika, opažanja i pronalaženja oblika u slikama trenutno se koriste u raznim realnim aplikacijama na masovnom tržištu. Pored prepoznavanja slika, ekstenzivno je rađeno i na prepoznavanju audio zapisa ljudskog govora. Inicijalno je dosta ovog proučavanja sprovedeno na ekskluzivno odabranim slučajevima. Međutim, istraživači su do početka 21. veka počeli da pristupaju problemu prepoznavanja govora pretežno koristeći skriveneMarkovljeve modele. Ovi modeli su bazirani na opširnoj i jakoj matematičkoj teoriji, što je inženjere oslobodilo ekstenzivnih zadataka istraživanja i formiranja teorija i dokazivanja postupaka. Pored ovoga, trening sistema za prepoznavanje govora je mogao da se vrši nad velikim skupom realnih zabeleženih razgovora u audio formatu. Sličan napredak nije beležen samo u softveru već i u hardveru u obliku robotike i računarske vizije. Ovo je dovelo do mogučnosti samostalnog prikupljanja informacija i trening skupova od strane mašina, bez potrebe za obeležavanjem i snabdevanjem podataka uz pomoć čoveka.

2. KLASIFIKACIJA I REGRESIJA

Računarska inteligencija zavisi kako od načina učenja računarskog sistema, tako i od problema za koji se traži opšte rešenje, a i od samog skupa podataka koji mogu biti prikupljeni u svrhu učenja računarskog sistema. Skupovi podataka namenjeni za učenje računarskog sistema su obično glomazni i vrlo retko struktuirani na način na koji se mogu korektno i brzo interpretirati od strane inženjera koji proizvodi sistem. Ovakvim podacima inženjeri moraju pristupati sa raznim algoritmima mašinskog učenja, koji se na visokom nivou mogu podeliti u grupe nadgledanih (eng. *supervised* [13]) algoritama za učenje

i nenadgledanih (eng. *unsupervised* [13]) algoritama za učenje. Implementacija konkretnog problema kojim se ovaj rad bavi zahteva takve podatke da se može sprovesti nadgledano učenje i iz tog razloga se ovaj rad neće baviti temom nenadgledanih algoritama za učenje sistema baziranih na veštačkoj inteligenciji.

Veliki deo učenja sistema baziranih na veštačkoj inteligenciji sprovodi se algoritmima nadgledanog učenja. Ovi algoritmi visoko zavise od ljudske intervencije, iz razloga što su potrebni trening podaci koji su *obeleženi* (eng. *labeled*, videti [4], [13]). Ovi obeleženi trening podaci sastoje se od ulaza *X* i očekivanog rezultata *Y* za prosleđeni ulaz *X*. Tehnike ovih algoritama uključuju linearne i logističke regresije, binarne i multi-klasne klasifikacije, metod potpornih vektora, stabla odlučivanja i slično. Ovi algoritmi se dalje mogu grupisati u regresione probleme i klasifikacione probleme.

U daljim poglavljima slede objašnjenja pojmova klasifikacije i regresije. Data su objašnjenja klasifikacionih i regresionih modela, u cilju razumevanja opšteg značaja klasifikacije i regresije u praksi. Dato je i objašnjenje pojma preteranog podudaranja podataka (eng. overfitting, videti [13]).

2.1. KLASIFIKACIONE PROCEDURE

Klasifikacija se bavi problemom obeležavanja novog ulaznog podatka tako da najbliže odgovara određenoj klasi na osnovu svojih atributa, videti [9]. Ova metodologija proizvodi diskretne rezultate (ulazni podatak pripada ili ne pripada klasi) i zavisi isključivo od klasifikacija koje se nalaze u zadatom trening skupu.

Sledi originalni primer koji ilustruje jedan slučaj klasifikacije:

Primer 2.1. Data je klasa ljudi koji imaju plavu boju očiju i klasa ljudi koji imaju zelenu boju očiju. Ukoliko novi ulazni podatak predstavlja osobu sa braon bojom očiju, ta osoba može biti klasifikovana samo u jednu od dve već postojeće klase. Ona će biti klasifikovana kao osoba sa zelenom bojom očiju, jer su crvena, zelena i plava komponenta braon boje mnogo sličnije čistoj zelenoj boji nego plavoj boji.

U primeru 2.1, ulazni podatak predstavlja osoba sa braon bojom očiju, a matematička forma ovog ulaznog podatka data je trojkom crvene, zelene i plave boje koje čine boju očiju osobe. Klasifikacije čine skup osoba sa plavom bojom očiju i skup osoba sa zelenom bojom očiju, a obučavajući skup je celokupan skup ljudi, i sa plavim i sa zelenim očima. Ulazni podatak se tako poredi sa svakom osobom iz obučavajućeg skupa i klasifikuje se na osnovu klase osobe kojoj je najsličniji.

Postoji mnoštvo razloga iz kojih bi modelovali klasifikacionu proceduru u cilju rešavanja određenog problema. Klasifikacione procedure koriste se kod uređivanja dokumenata na osnovu digitalno pročitanih atributa dokumenata, tako da lošije učitani slučajevi ostanu čoveku da ručno reši uređivanje. Banke koriste klasifikacione procedure prilikom odluke da li treba određenim klijentima dati kredit, jer mašina nema bias ka nekim atributima klijenta i neće sagledati te atribute prilikom računanja odluke. U medicini se klasifikacione procedure koriste za davanje dijagnoze, jer je u nekim slučajevima bitno izbeći komplikovane operacije, a moguće je odrediti dijagnozu na osnovu eksternih simptoma, videti [13].

Klasifikacione mašine, naravno, imaju i svoje slabosti, videti [4]. Jedna od ovih slabosti je sama preciznost klasifikatora kao i brzina donošenja odluke. Klasifikatori zavise kako od korektnosti obučavajućeg skupa tako i od obilnosti obučavajućeg skupa. Preciznost klasifikatora se meri proporcijom tačnih odgovara klasifikatora, mada je u nekim slučajevima potrebno i sagledati konkretne netačne predikcije klasifikatora, jer su neke pogrešne procene mnogo ozbiljnije od drugih. Rešavanje ovog problema može negativno uticati na brzinu samog klasifikatora. Veći trening skup sa više atributa podataka će davati bolje predikcije, ali će imati negativan uticaj na brzinu donošenja odluka. U nekim slučajevima gde je moguća ljudska intervencija prilikom rešavanja problema, mnogo je bolje imati klasifikator koji je u 85% slučajeva tačan, a da je 200 puta brži od klasifikatora koji je u 98% slučajeva tačan, videti [9].

U nastavku rada sledi definicija pojma izlaza klasifikatora, koji se dobija korišćenjem određenih karakteristika ulaznog podatka koji je odabran za klasifikaciju. Neka su $\vec{x} = (x_1, x_2, \dots x_n)$ i $\vec{w} = (w_1, w_2, \dots w_n)$ vektori iz prostora $\mathbb{R}^n$, pri čemu je x ulazna vrednost koju treba klasifikovati, a w prethodno dat težinski vektor koji odgovara određenoj klasifikaciji. U tom slučaju izlazna vrednost klasifikatora je $y \in \mathbb{R}$ što je prikazano u sledećoj definiciji:

Definicija 2.1.[1] *Za prethodno definisane ulazne vrednosti $\vec{x}$ i težine klasifikatora $\vec{w}$ i datu funkciju f koja konvertuje skalarni proizvod vektora u odgovarajuće vrednosti u zavisnosti od klasifikacija, izlaz klasifikatora definiše se sa:*

$$y = f(\vec{w} \cdot \vec{x}) = f(\sum_j w_j x_j).$$

Funkcija f nije ista za sve vrste klasifikacija i obično se bira funkcija koja najbolje opisuje klasifikaciju. Često se za klasifikacionu funkciju f koristi sigmoidna funkcija, $f(x) = \frac{1}{1+e^{-x}}$ koja nalazi primenu u neuronskim mrežama o čemu će biti više reči u istoimenom poglavlju.

Vektor težina $\vec{w}$ upravo predstavlja parametar koji treba da se *nauči* od strane sistema koristeći dati skup obučavajućih podataka. Ovakvi klasifikatori se mogu podeliti na osnovu načina na koji određuju parametar $\vec{w}$ i to na *generativne* i na *diskriminacione* modele, što se može videti u [10].

Generativni klasifikacioni modeli su modeli koji se sastoje od tražene klasifikacije *y,* i od posmatranog podatka *X* i njihove međusobne uslovne verovatnoće, videti [10].

U ovakvim modelima parametar *X* uglavnom predstavlja neprekidne vrednosti, a parametar *y* diskretne vrednosti, upravo iz razloga što je poznata svaka moguća klasifikacija iz obučavajućeg skupa, videti [10]. Generativni modeli se koriste u slučajevima kada nam je potrebno da saznamo kako su određeni podaci dobijeni ili generisani i pogodni su za dopunjavanje nedostataka u određenom skupu podataka, za generisanje novih podataka koji inicijalno nisu opaženi ili za kompresiju podataka, opisivajući funkciju koja generiše podatke iz skupa.

Diskriminacioni modeli služe u obrnute svrhe od generativnih modela i opisuju kako se tražene klasifikacije *y* dobijaju od poznatih podataka. Ne vrše modelovanje osnovne raspodele verovatnoća, već vrše direktno ocenjivanje posteriornih verovatnoća, videti [10].

Problemi koji se rešavaju pomoću klasifikacije se svrstavaju u probleme višeklasne klasifikacije i binarne klasifikacije, videti [4],[9]. Binarna klasifikacija je zadatak čiji obučavajući skup sadrži tačno dve moguće klase u

koje se može svrstati ulazni podatak. Ove klase su obično u obliku istinitosnih vrednosti (jeste/nije zaražen, jeste/nije kriv itd.).

Koju god strategiju odaberemo, svaka koristi određeni algoritam za učenje, kako bi se našla najbolja veza (ili *model*) koja se može koristiti za opisivanje ulaznih podataka za učenje, a i za određivanje klasifikacije novih ulaznih podataka. Ključna karakteristika algoritma za učenje jeste generalizacija podataka.

Obučavanje klasifikacionog modela obavezno zahteva skup obučavajućih podataka, videti [10]. Svi podaci u obučavajućem skupu imaju zadate vrednosti atributa klasifikacija koje se klasifikuju i svaka instanca klasifikacionog skupa je obeležena tako da se tačno zna koji rezultat se očekuje od klasifikacionog modela. Pored ovog skupa sa obučavajućim podacima, uključuje se i jedan skup podataka za testiranje efektivnosti klasifikacionog modela. Ovaj *test-skup* takođe ima vrednosti za svaki atribut u skupu i ima obeležene instance koje se očekuju da klasifikacioni model predvidi. Razlika između ova dva skupa je što se prvi, obučavajući skup, koristi za donošenje zaključaka od strane klasifikacionog modela. Test skup se koristi samo za evaluaciju modela i ne ulazi u samu fazu donošenja zaključaka klasifikacionog algoritma.

Evaluacija klasifikacionog modela bazira se na prebrojavanju korektno i pogrešno predviđenih rezultata. Ovi 'pogodci' i 'promašaji' se zapisuju tabelarno u matricu konfuzije (eng. *confusion matrix*). Tabela 2.1 predstavlja matricu konfuzije za jedan binaran problem klasifikacije, gde se podaci mogu klasifikovati kao 0 ili kao 1. Po definiciji 2.1, ovo znači da je izlaz klasifikatora definisan na prostoru {0, 1}. U tabeli 2.1, svaki podatak f_{ij} predstavlja broj podataka iz klase *i* koji su klasifikovani u klasu *j*. Tako bi f_{01} predstavljalo broj podataka iz klase 0 koji su pogrešno klasifikovani u klasu 1. Po definiciji 2.1, f_{00}, f_{01}, f_{10} *i* f_{11} predstavljaju izlaze *y* klasifikatora.

		Predviđena klasifikacija	
		Klasifikacija 1	*Klasifikacija 0*
Tačna Klasifikacija	*Klasifikacija 1*	f_{11}	f_{10}
	Klasifikacija 0	f_{01}	f_{00}

Tabela 2.1. Matrica konfuzije za binarni klasifikacioni problem.

U slučaju konfuzione matrice u tabeli 2.1, broj pogođenih klasifikacija je $f_{00} + f_{11,}$ a broj promašaja je $f_{10} + f_{00}$.

Konfuziona matrica ne služi samo za procenu koliko dobro neki klasifikacioni model predviđa klasifikacije na osnovu obučavajućeg skupa, već se koristi i za izvlačenje određenih proračuna kao što su preciznost klasifikatora i verovatnoća greške klasifikatora.

$$Preciznost = \frac{Broj\ pogođenih\ klasifikacija}{Ukupan\ broj\ predviđanja} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \qquad (2.1)$$

$$Verovatnoća\ greške = \frac{Broj\ pogrešnih\ predviđanja}{Ukupan\ \ broj\ predviđanja} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}} \qquad (2.2)$$

Jedan od najjednostavnijih i najpoznatijih klasifikacionih algoritama je algoritam stabla odlučivanja (eng. Decision Tree, videti [13], [9]). Stabla odlučivanja rešavaju problem klasifikacije tako što za jedan ulazni podatak odgovaraju na sekvencu pitanja. Svaki put kada se da odgovor na jedno pitanje vezano za atribute podatka, postavlja se novo pitanje, sve dok se ne dođe do zaključka o klasifikaciji ulaznog podatka. Ova serija pitanja se može predstaviti u obliku stabla koje se sastoji od čvorova i usmerenih grana. Stablo odlučivanja se sastoji od sledećih komponenti:

1. Početni čvor koji nema ni jednu granu koja vodi do ovog čvora (*koren stabla odlučivanja)*. Ovaj čvor može, a ne mora, da sadrži izlazne grane koje vode do ostalih čvorova unutar stabla,
2. Interne čvorove koji sadrže tačno jednu ulaznu granu, a imaju dve ili više izlaznih grana, i
3. Terminalne čvorove (*listove stabla odlučivanja)*, koji sadrže tačno jednu ulaznu granu i nemaju izlaznih grana.

U stablima odlučivanja, listovi stabla sadrže obeležje klasifikacije. Svi interni čvorovi stabla odlučivanja predstavljaju stanja u kojima se postavlja pitanje za ulazni podatak. Nakon što je stablo odlučivanja konstruisano, jednostavno se može doći do odgovarajuće klasifikacije ulaznog podatka. Počevši od korenskog čvora, testiramo ulazni podatak (postavljamo pitanje za podatak) i na osnovu rezultata pratimo grane stabla odlučivanja. Ovo putovanje po granama nas vodi ili do sledećeg internog čvora na kojem se vrši novo testiranje, ili do terminalnog čvora, koji određuje klasifikaciju ulaznog podatka. Ovaj algoritam je prilično jednostavan za implementaciju, pod uslovom, naravno, da imamo izgrađeno stablo odlučivanja.

Problem kod konstrukcije stabla odlučivanja predstavlja činjenica da se na osnovu jednog datog skupa atributa može konstruisati eksponencijalno mnogo stabala odlučivanja. Neka stabla su mnogo preciznija od drugih, dok neka stabla odlučivanja proizvode rezultate u mnogo kraćem vremenskom roku. Ovi problemi kod konstrukcija stabala odlučivanja doveli su do razvoja algoritama specijalizovanih upravo za konstrukciju stabala u relativno prihvatljivom vremenskom roku. Algoritmi su takvi da često odabiraju više atributa i kreiraju više čvorova za rast stabla tako što prave lokalne optimizacije za odabir atributa i kriterijuma tih atributa na osnovu kojih će se deliti čvorovi. Najpoznatiji od ovih algoritama je Hantov Algoritam (eng. *Hunt's Algorithm*).

Pretpostavimo da nam je dat skup obučavajućih podataka D sa skupom klasifikacija $y = \{y_1, y_2, \dots y_c\}$ i pretpostavimo da je D_t skup obučavajućih podataka na čvoru t u stablu odlučivanja. Hantov algoritam je predstavljen sa sledeća dva koraka:

1. Ukoliko svi obučavajući podaci na čvoru t pripadaju klasi y_t (ukoliko su svi podaci isti), onda čvor t predstavlja list stabla odlučivanja koji obeležavamo sa y_t.
2. Ukoliko skup podataka na čvoru t ne može da se svrsta u jednu klasifikaciju, mora se definisati kriterijum po kom će se skup podeliti u manji skup podataka. Za svaki ishod provere kriterijuma sa jednim podatkom iz obučavajućeg skupa, kreira se novi čvor stabla i na svaki čvor se prosleđuju određeni podaci iz skupa podataka. Nakon ovog koraka, ponavlja se korak 1. za sve nove čvorove stabla odlučivanja.

Hantov algoritam će raditi ako u skupu obučavajućih podataka svaka kombinacija vrednosti atributa ima jedinstvenu klasu. Ovo pravilo je previše strogo kako bi se Hantov algoritam uspešno koristio u praksi. Postoji mogućnost da u koraku broj 2 čvor stabla bude prazan tj. ni jedan podatak iz trening skupa ne ispunjava uslove koji su potrebni da završi u čvoru. Ako dođe do ovog slučaja, čvor koji nema obučavajućih podataka svrstava se u istu klasifikaciju kao čvor koji ima najviše podataka, a pripada istom roditeljskom čvoru kao i čvor bez podataka. Drugi problem jeste ako u jednom čvoru t svi podataka imaju iste vrednosti za svoje atribute, a različito su klasifikovani, onda nije moguće podeliti čvor t. U ovom slučaju se čvor t ne deli već se pretvara u list, sa onom klasifikacijom koja se najviše pojavljuje u obučavajućem skupu na čvoru t.

Rešavajući ove probleme, potrebno je dati odgovor na pitanje kako se obučavajući skup podataka treba podeliti. Podela obučavajućeg skupa na više

novih čvorova podrazumeva da algoritam pruža metod za odabir uslova koje atributi obučavajućeg skupa moraju da ispune kako bi se klasifikovali u neku klasu – kako bi prešli u novi čvor. Pored samog određivanja uslova, mora postojati način da se odredi koliko je taj uslov dobar. Preveliko stablo takođe predstavlja potencijalni problem. Iz ovog razloga mora postojati način da se odredi kada je potrebno zaustaviti rast stabla.

Algoritmi koji se tiču rasta stabla moraju se prilagoditi različitim tipovima podataka. Binarni atributi se najlakše dele pošto ovi atributi mogu imati jednu od dve vrednosti (obično *tačna* ili *netačna)*. Deljenje diskretnih vrednosti se vrši na dva različita načina. Binarnim deljenjem se u jedan čvor izdvajaju podaci koji ispunjavaju jedan kriterijum, a u drugi čvor ostali podaci i taj drugi čvor se dalje može razvijati po istom atributu. Drugi način deljenja diskretnih vrednosti je razvijanje svih čvorova od jednog nadređenog čvora i u ovom slučaju možemo imati više od 2 deteta za jedan čvor. Ako atributi imaju neprekidne vrednosti, takođe se mogu podeliti na ova dva načina.

2.2. REGRESIJA

Jedan od osnovnih pojmova statistike predstavlja obeležje, odnosno osobina koja se ispituje. Obeležje predstavlja slučajnu promenljivu, odnosno promenljivu datu na populaciji, tj. skupu svih mogućih elemenata, sa vrednostima u skupu svih mogućih ishoda posmatranog fenomena. Kako je prilikom istraživanja gotovo nemoguće prikupiti podatke za celokupnu populaciju, ono se vrši na uzorku čija precizna definicija sledi :

Definicija 2.2.[8] Neka se na populaciji $\mathcal{E}$ posmatra obeležje X. Prost slučajan uzorak obima n za obeležje X je n-torka nezavisnih slučajnih promenjivih $(X_1, X_2, \ldots, X_n)$ od kojih svaka ima istu raspodelu kao i obeležje X.

U statistici pronalaženjem statističkih veza između više pojava (obeležja) bavi se regresiona analiza. Ona nalazi široku primenu kako u ekonomiji i privredi, tako i u drugim prirodnim naukama kada treba odrediti međusobnu zavisnost među posmatranim pojavama, kao na primer predviđanje kretanja cena akcija

na berzi, uticaj inteligencije na postignuća pojedinca, veza između kvaliteta vode i upotrebe zemljišta……

Problem opisivanja ovakvih veza svodi se na pronalaženje modela koji povezuje jednu ili više zavisnih promenljivih $Y_1, Y_2, \ldots, Y_n$ sa jednom ili više nezavisnih promenljivih $x_1, x_2, \ldots, x_n$ pomoću neke funkcionalne zavisnosti. Oblik ove funkcionalne zavisnosti najčešće je nepoznat, pa ostaje na istraživaču da izabere onu koja je po nekom kriterujumu najbolja [8].

Prema vrsti veze između zavisne i nezavisnih varijabli, regresija može da bude:

- linearna
- nelinearna (kvadratna, polinomna, eksponencijalna…..).

Linearna regresija predstavlja najjednostavniji regresioni model koji razmatra vezu između jedne zavisno promenljive Y koja je slučajna veličina i jedne nezavisne promenljive x koja je deterministička.

Definicija 2.3.[8] *Neka su Y i ε slučajne promenljive, x neslučajna promenljiva, a μ linearna funkcija od x. Linearni model se definiše sa:*

$$Y = \mu(x) + \varepsilon \tag{2.3}$$

Funkcija μ se naziva deterministički deo modela, a ε se naziva greška ili rezidual i predstavlja slučajni deo modela koji se ne može meriti. Promenljiva ε sadrži u sebi sve one slučajne faktore koji se ne mogu kontrolisati i nisu obuhvaćeni posmatranjem, a utiču na vrednosti promenljive Y.

Definicija 2.4.[8] *Neka su $q_1, q_2, \ldots q_p$ poznate funkcije. Najopštiji oblik funkcije μ koji zavisi od p+1 nepoznatih parametara može se zapisati kao:*

$$\mu(x) = \beta_0 + \beta_1 q_1(x) + \beta_2 q_2(x) + \cdots + \beta_p q_p(x), \tag{2.4}$$

Da bi se ocenili nepoznati parametri $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$, mora se koristiti uzorak. Za fiksiranih n vrednosti $x_1, x_2, \ldots, x_n$ nezavisno promenljive x, određuju se vrednosti promenljive Y. Na taj način se dobija n parova $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$ koji čine uzorački model:

$$\begin{matrix} Y_i = \mu(x_i) + \varepsilon_i \\ E(\varepsilon_i) = 0 \end{matrix}, \qquad i = 1,2, \dots ., n \tag{2.5}$$

gde su greške ε_i , $i = 1,2, \dots, n$ takve da je $cov(\varepsilon_i, \varepsilon_j) = \sigma_{ij}$, $i, j = 1,2, \dots n$ tj. greške su autokorelirane.

Mogu se staviti i dodatne pretpostavke na reziduale: da su međusobno nezavisni, da imaju normalnu raspodelu, homoskedastičnost (da su im disperzije jednake) $D(\varepsilon_i) = \sigma^2$, $i = 1,2, \dots n$, heteroskedastičnost (ako su im disperzije različite) ili da su nekorelirane tj. da je $E(\varepsilon_i \varepsilon_j) = 0$.

Određivanje parametara u funkciji μ u modelu vrši se tako da se pomoću nje najbolje opiše veza između promenljivih Y i x , odnosno da funkcija μ što manje odstupa od registrovanih vrednosti promenljive Y. Jedna od mogućih mera odstupanja funkcije μ od vrednosti promenljive Y je suma kvadrata odstupanja:

$$F = \sum_{i=1}^{n} (Y_i - \mu(x_i))^2 = \sum_{i=1}^{n} {\varepsilon_i}^2 \tag{2.6}$$

Ocene parametara se dobijaju tako da se traži minimum funkcije F. Ova metoda se naziva *metoda najmanjih kvadrata* i ocene dobijene njom su najefikasnije u klasi svih linearnih centriranih ocena. Dakle, one su nepristrane i imaju najmanju disperziju. Pored ove metode, mogu se koristiti još neke za procenu parametara kao što su: metoda maksimalne verodostojnosti , BLUE metoda itd.

U situacijama u kojima funkcionalan odnos između zavisne promenljive Y i nezavisne promenljive x ne može biti adekvatno aproksimiran linearnim odnosom koriste se modeli *nelinearne regresije*. Ukoliko je nelinearan model linearan po svim nepoznatim parametrima, jednostavnom smenom se svodi na linearan model. U slučaju da postoji bar jedan parametar po kome je regresioni model nelinearan izračunavanje ocena parametara je komplikovanije. Sa razvojem računara je došlo do intenzivnije primene nelinearnih modela i do ispitivanja osobina ocena njihovih parametara.

Nelinearni regresioni modeli linearni po nepoznatim parametrima su na primer:

- $Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_k x^k + \varepsilon$,
- $Y = \beta_0 + \frac{\beta_1}{x} + \varepsilon$,
- $Y = \beta_0 + \beta_1 cosx + \beta_2 sinx + \varepsilon$,

dok su nelinearni po nepoznatim parametrima sledeći modeli:

- $Y = \frac{\alpha}{1+e^{\beta-\gamma x}} + \varepsilon$, (logistički)
- $Y = \alpha - \beta e^{-\gamma x^{\delta}} + \varepsilon$, (Weibull-ov)
- $Y = \frac{\beta_1 x_2 - \frac{x_3}{\beta_5}}{1+\beta_2 x_1 + \beta_3 x_2 + \beta_4 x_3} + \varepsilon$, (Hougen Watson-ov).

Vrednost predviđanja zavisne promenljive izračunata na osnovu nelinearnih modela je pristrasna, a stepen pristrasnosti zavisi od veličine koja se naziva suštinska (istinska) nelinearnost. Kod nelinearne regresije, prilikom korišćenja metode najmanjih kvadrata dobijaju se nelinearni oblici parametara. U slučaju kada je prva vrsta nelinearnosti prihvatljiva, postojanje druge vrste nelinearnosti može da utiče da je konvergencija ocena parametara ka stvarnim vrednostima spora. Korišćenjem metode najmanjih kvadrata dobija se sistem jednačina koje nemaju jedinstveno rešenje. Za rešavanje takvih sistema potrebno je napraviti početne procene parametara, te ih zatim odrediti metodom sukcesivnih aproksimacija. Više o ovoj temi može se naći u [17].

2.3. PRETERANO PODUDARANJE PODATAKA

Do sada smo videli da je za učenje klasifikacionih ili regresionih modela potreban obučavajući skup podataka, obeležen tako da model zna kada je pogrešio i za koliko je pogrešio, kako bi korigovao svoje sledeće predikcije. Trening podaci mogu služiti i da model na osnovu njih svaki put donese odluku kada se to traži od njega, bez da uči ili da pamti određene parametre izvedene iz podataka, videti [4], [13]. Ako nam je potreban skup podataka koji pomažu modelu da donese odluku logično je zaključiti da je bolje da imamo što

veći skup podataka kako bi model što preciznije došao do odluke. Međutim, ovo nije uvek tačno jer može doći do slučaja kada je model toliko naučen da novi podaci, koji malo odstupaju od poznatih rešenja, a i dalje mogu da se svrstaju u poznate klasifikacije, bivaju odbačeni kao da ne pripadaju tim klasifikacijama , videti [13].

U praksi su najbolji klasifikacioni modeli oni koji, pored dobrog klasifikovanja obučavajućih podataka, mogu i precizno klasifikovati podatke koje nikada nisu opažali. Za ovaj proračun služe greške obuke i generalizacije, gde greške obuke ukazuju na procenat pogrešno klasifikovanih trening podataka, a greške klasifikacije opisuju koliko često klasifikacioni model greši prilikom klasifikacije podataka koje još nije video. Za modele koji mnogo više greše u proceni klasifikacija novih podataka se kaže da u preteranoj meri traže podudarnost između obučavajućih i novih podataka.

Slučaj kada klasifikacioni model pravi greške dok je trening u toku naziva se slabo podudaranje podataka (eng. *underfitting*, videti [13]). Ovo je zbog toga što model jednostavno nije upoznat sa dovoljno podataka iz kojih može izvoditi bilo kakve zaključke. Iz tog razloga ne može ni obučavajuće podatke, a ni nove podatke korektno da klasifikuje. Kako broj viđenih slučajeva raste, tako se smanjuju greške treninga i greške klasifikacije. Novi problem nastaje kada ovaj broj viđenih obučavajućih podataka poraste toliko da u jednom trenutku greške generalizacije počinju da se povećavaju, a greške obučavanja nastavljaju da se smanjuju. Ovaj problem predstavlja preterano podudaranje podataka (eng. *overfitting*, videti [13]). Overfitting i underfitting modela predstavljaju najveće probleme u treningu sistema baziranih na inteligentnim sistemima i postoji velik broj mogućih objašnjenja zašto se ovi problemi javljaju, što pomaže prilikom selekcije obučavajućeg skupa.

Osnovni razlog zašto može doći do underfitting-a modela jeste zbog mogućih smetnji u skupu obučavajućih podataka, videti [4], [13]. Ove smetnje ne moraju biti pogrešno očitani podaci sa uređaja, već je dovoljno da se klasifikacija određenog trening podatka pogrešno obeleži. Ove slučajeve je gotovo nemoguće izbeći, i najčešće je minimalna moguća greška, koju neki sistem baziran na veštačkoj inteligenciji može da proizvede, upravo greška koju je proizveo pogrešno unet obučavajući podatak.

Do overfitting-a može doći čak kada ima previše ili premalo obučavajućih podataka. Nedostatak raznolikosti u obučavajućem skupu može dovesti do odličnih rezultata za trening, odnosno izuzetno male greške učenja, ali samim tim što nema različitih slučajeva koje sistem može videti, on neće biti u

mogućnosti da precizno klasifikuje slučajeve koji mu nisu poznati i greška generalizacije će biti veća , videti [13].

3. KLASIFIKATORI BAZIRANI NA BEJZOVOM PRAVILU

Tomas Bejz (*Thomas Bayes*) bio je engleski statističar i filozof, rođen u Hertfordšajeru (*Herthfordshire*) na jugu Engleske. 1719. godine započeo je studije logike i teologije u Škotskoj na Univerzitetu u Edinburgu. Tek kasnije u životu je počeo da se bavi teorijama verovatnoće i teorema po njegovom imenu, Bejzova teorema, izneta je 1763. godine, dve godine nakon njegove smrti.

Bejzovi procesi klasifikacije bave se problemima klasifikacije korišćenjem Bejzovog pravila, videti [13]. Pretpostavimo da imamo problem u kojem moramo aproksimirati nepoznatu funkciju $f: X \rightarrow Y$ odnosno odrediti klasifikator, a to je $P(X|Y)$, uzimajući u obzir da je Y slučajna promenljiva koja prima vrednosti iz skupa {tačno, netačno}, a X je vektor koji se sastoji od n atributa sa vrednostima iz skupa {tačno, netačno} zapisan na sledeći način:

$$X = (X_1, X_2, X_3 \dots X_n)$$

gde je X_i slučajna promenljiva koja uzima vrednost iz skupa {tačno, netačno}, odnosno i-ti atribut skupa X. Ono što treba primetiti jeste da se kao klasifikator ovde zapravo koristi uslovna verovatnoća, što će u nastavku biti objašnjeno u sklopu metode „naivni“ Bejz.

U cilju razumevanja ovog problema, u daljim poglavljima dato je objašnjenje Bejzovog pravila, kao i uslovne verovatnoće (videti [16]). Takođe je dat i opis Bejzovog klasifikatora, „naivni“ Bejz (eng. „naive“ Bayes, videti [1], [10], [11]).

3.1. USLOVNA VEROVATNOĆA I BEJZOVO PRAVILO

Definicija 3.1.[12] *Neka je Ω neprazan skup. Podskup $\mathcal{F}$ partitivnog skupa $\mathcal{P}(\Omega)$ je σ-polje (σ-algebra) nad Ω (skupom svih mogućih ishoda) ako važe uslovi:*

i. $\Omega \in \mathcal{F}$,
ii. *ako* $A \in \mathcal{F}$, *onda* $\bar{A} \in \mathcal{F}$,
iii. *ako* $\{A_i\}_{i\in\mathbb{N}} \subseteq \mathcal{F}$, *onda* $\bigcup_{i=1}^{\infty} A_i \in \mathcal{F}$.

Definicija 3.2.[12] *Neka je Ω skup svih elementarnih događaja i $\mathcal{F}$ je σ-polje nad Ω. Funkcija $P: \mathcal{F} \mapsto [0,1]$ se zove verovatnoća na prostoru $(\Omega, \mathcal{F})$ ako zadovoljava uslove*

i. $P(\Omega) = 1$,
ii. *Ako* $\{A_i\}_{i\in\mathbb{N}} \subseteq \mathcal{F}$, $A_i \cap A_j = \emptyset,\ i \neq j,\ i,j = 1,2,\ldots\ldots,$ *onda* $P(\sum_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$.

Definicija 3.3.[12] *Prostor verovatnoća je uređena trojka $(\Omega, \mathcal{F}, P)$, gde je Ω skup svih elementarnih događaja, $\mathcal{F}$ je σ-polje nad Ω, a P je verovatnoća nad $(\Omega, \mathcal{F})$.*

Definicija 3.4.[11] *Neka je* $(\Omega, \mathcal{F}, P)$ prostor verovatnoća i neka su A, B $\epsilon\, \mathcal{F}$ *dva događaja pri čemu je verovatnoća događaja A veća od 0 (P(A) > 0). Verovatnoća da će se desiti događaj B, ukoliko imamo dokaz o dešavanju događaja A naziva se uslovna verovatnoća i data je sa:*

$$P(B|A)$$

U slučaju kada događaji A i B ne zavise jedan od drugog, uslovna verovatnoća događaja B u odnosu na događaj A svodi se na verovatnoću događaja B:

$$P(B).$$

Definicija 3.5.[11] *Ukoliko su događaji A i B zavisni jedan od drugog, onda verovatnoća da se desi i događaj A i događaj B (presek ova dva događaja) definisana je sa:*

$$P\left(A \bigcap B\right) = P(A) \cdot P(B|A)$$

Definicija 3.6.[11] *Neka su A i B događaji, tako da postoji verovatnoća da se desi događaj A ($P(A) \neq 0$). Formalna definicija uslovne verovatnoće dešavanja događaja B pod uslovom da se desio događaj A data je sa:*

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

Primer 3.1.1.[16] U igri sa kartama u kojoj igrač mora izvući dve karte iste klase. U špilu ima 52 karte i 4 moguće klase, dakle 13 karata po klasi. Pretpostavimo da je igrač prvo izvukao kartu u klasi srca. Igrač mora ponovo izvući kartu koja je klase srca kako bi pobedio. S obzirom da je jedno srce već izvučeno, sada ima 12 srca u špilu, koji ima 51 kartu. Ovo svodi uslovnu verovatnoću na:

$$P(druga\ karta\ srce|prva\ karta\ srce) = \frac{12}{51}$$

Na ovaj način moguće je izračunati uslovnu verovatnoću preseka više od dva događaja. Kada je reč o više od dva događaja, potrebno je uračunati uslovne verovatnoće svih prethodnih događaja. Ukoliko imamo događaje *A, B* i *C*, presek ovih događaja se može dobiti kao [16]:

$$P\left(A \bigcap B \bigcap C\right) = P(A) \cdot P(B|A) \cdot P(C|A \cap B)$$

Još jedan značajan način za izračunavanje uslovne verovatnoće dat je Bejzovom formulom (kada imamo dva događaja).

Teorema 3.1. (Bejzova Teorema)[11] *Ukoliko imamo dva događaja A i B tako da postoji verovatnoća da će se desiti događaj B (P(B) > 0), uslovna verovatnoća događaja A ako se desio događaj B može se predstaviti kao:*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Ukoliko posmatramo problem klasifikacije, Bejzovu teoremu možemo formulisati na sledeći način.

Teorema 3.2. [16] *Ukoliko imamo observaciju X, verovatnoća da observacija X pripada klasi C može se predstaviti kao:*

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

Bejzovo pravilo deluje jednostavno, ali je veoma korisno u praksi u slučajevima kada posedujemo dobre procene jedne uslovne verovatnoće i dve nezavisne verovatnoće događaja i potrebno nam je da sračunamo još jednu uslovnu verovatnoću. Bejzovo pravilo se koristi često u medicinskoj dijagnozi, gde su nam date uslovne verovatnoće uobičajenih veza simptoma, a potrebno je da odredimo dijagnozu.

Primer 3.1.2. [13] Lekar zna da meningitis ima simptom krutosti vrata u, recimo, 50% slučajeva. Uz ove informacije, lekar takođe zna i nezavisne činjenice kao što su da jedan od 50.000 pacijenata ima meningitis, i jedan od 20 pacijenata može imati krutost vrata. Ako je M slučaj u kom pacijent ima meningitis, a K slučaj u kom pacijent ima krut vrat, sledi:

$$P(K|M) = 0.5$$
$$P(M) = 1/50000 = 0.00002$$
$$P(K) = 1/20 = 0.05$$
$$P(M|K) = \frac{P(K|M) \cdot P(M)}{P(K)} = \frac{0.5 \cdot 0.00002}{0.05} = 0.0002$$

Ovo implicira da možemo očekivati da jedan u 5000 pacijenata koji ima krut vrat ima i meningitis. Ovde je bitno primetiti meningitis snažno implicira krutost vrata (50% slučajeva), verovatnoća meningitisa u pacijentu sa krutim vratom ostaje mala, upravo zbog nezavisne verovatnoće krutosti vrata u bilo kom pacijentu.

Iz ovog jednostavnog primera se može postaviti pitanje zašto bi imali uslovnu verovatnoću u jednom smeru a ne u drugom? U primeru 3.1.2, recimo da doktor zna da svaki od 5000 pacijenata sa krutosti vrata ima meningitis, i nema potrebu da koristi Bejzovo pravilo. Ukoliko dođe do iznenadne epidemije meningitisa, raste verovatnoća događaja da pacijent ima meningitis, odnosno raste vrednost za $P(M)$. Doktor koji je dobio znanje da svaki pacijent u 5000 pacijenata ima krutost vrata i meningitis, neće znati kako da popravi svoje zapažanje, a doktor koji je ovu verovatnoću dobio iz računanja $P(M|K)$ na osnovu ostale tri verovatnoće, videće da rast $P(M)$ proporcionalno utiče na rast vrednosti $P(M|K)$. Ovakvo korišćenje direktnog praktičnog znanja ili znanja koje je bazirano na konkretnom modelu je krucijalno za razvoj i za uspeh probabilističkih sistema u realnom svetu, videti [13].

Imajući u vidu način za izračuvanje verovatnoće meningitisa ukoliko pacijent ima krutost vrata, pretpostavimo da nas u isto vreme interesuje da li je pacijent istegao vrat ukoliko ima krutost vrata, dato sa:

$$P(I|K) = \frac{P(K|I)P(I)}{P(K)}$$

i pretpostavimo da je verovatnoća da pacijent ima krutost vrata jer je istegao vrat $P(K|I) = 0.8$ i da je verovatnoća da je određeni pacijent istegao svoj vrat $P(I) = 0.001$. Ukoliko moramo izračunati verovatnoću meningitisa u pacijentu i istegnutog vrata, ukoliko pacijent ima krut vrat, nama nije potrebna verovatnoća krutosti vrata u pacijentu $P(K)$:

$$\frac{P(M|K)}{P(I|K)} = \frac{\frac{P(K|M)P(M)}{\cancel{P(K)}}}{\frac{P(K|I)P(I)}{\cancel{P(K)}}} = \frac{P(K|M)P(M)}{P(K|I)P(I)} = \frac{0.5 \cdot 0.00002}{0.8 \cdot 0.001} = \frac{1}{80} = 0{,}0125$$

što implicira da je veća šansa da je pacijent istegao vrat nego da ima meningitis, ukoliko ima krut vrat, za čak 80%. U medicini se ovakav proračun ne koristi upravo jer konačan ishod dovodi do totalno drugačijeg načina lečenja pacijenta i potrebne su što egzaktnije mere proračuna verovatnoća. Potrebno je uzeti u obzir skup ostalih mogućih simptoma. Skup simptoma koji nisu meningitis je, naravno, preglomazan za korišćenje direktno, i iz tog slučaja se ovaj skup koristi kao negirana verovatnoća da pacijent ima meningitis $\overline{M}$:

$$P(M|K) = \frac{P(K|M)P(M)}{P(K)}$$

$$P(\overline{M}|K) = \frac{P(K|\overline{M})P(\overline{M})}{P(K)}$$

S obzirom da su događaji suprotni, jedan od događaja M i $\overline{M}$ mora da se desi, imamo da je:

$$P(M|K) + P(\overline{M}|K) = 1$$

i možemo ovo zapisati kao:

$$\frac{P(K|M)P(M) + P(K|\overline{M})P(\overline{M})}{P(K)} = 1$$

$$P(K|M)P(M) + P(K|\overline{M})P(\overline{M}) = P(K)$$

Ovaj proces naziva se proces normalizacije. Proces normalizacije tretira verovatnoću u imeniocu Bejzove formule kao normalizujuću konstantu. Na ovaj način uklanjamo uopšte potrebu za verovatnoćom u imeniocu Bejzove formule. Imajući vrednost za $P(M|K)$ i uzimajući u obzir $P(M|K) + P(\overline{M}|K) = 1$, možemo izračunati $P(\overline{M}|K)$ i na taj način izbeći zavisnost od korišćenja $P(K)$.

3.2 BEJZOV KLASIFIKATOR U PRAKSI

Bejzovo rezonovanje verovatnoćama korišćeno je u praksi od 1960-ih godina. Ova vrsta veštačke inteligencije i klasifikacije se najčešće koristila u medicini, kada je za takav proračun postojalo dovoljno velik skup podataka kako bi se napravila što egzaktnija procena, videti [13]. U početku su performanse algoritma bile loše i imale česte promašaje. Najveći razlog ovih promašaja je upravo nedovoljan broj podataka za dobre procene, kao i loša teorijska osnova

modelovanih uslova i događaja. Nije bilo konkretnih načina da se opišu veze između događaja ili uzroci određenih događaja i sistemi su morali da rukuju ogromnim količinama podataka. U međuvremenu su se razvijali i razni drugi načini za klasifikaciju ulaznih podataka na osnovu Bejzovog pravila. Pored primera u medicini, Bejzov algoritam koristio se i u klasifikaciji dokumenata, gde je razvijen algoritam „naivni“ Bejz, koji do sada predstavlja jedan od baznih algoritama u klasifikaciji teksta. „Naivni“ Bejz je jedan od najefikasnijih algoritama induktivnog učenja za mašinsko učenje.

Glavna ideja klasifikacionog algoritma „naivni“ Bejz jeste kompletna nezavisnost atributa klasa prilikom računanja verovatnoće da podataka X pripada nekoj od klasa iz skupa klasa $Y = (Y_1, Y_2, Y_3 \dots Y_n)$, videti [10]. „Naivni“ Bejz klasifikator je izvanredno skalabilan klasifikator, kako će se videti dalje u radu, i u određenim poljima predstavlja ozbiljnu konkurenciju drugim mnogo kompleksnijim klasifikatorima kao što su neuronske mreže ili mašine podržavajućih vektora. Način na koji se ovaj klasifikator konstruiše je izuzetno jednostavan za većinu programskih jezika. Suština je u korišćenju vektora atributa koji predstavljaju ‘telo’ klasifikatora, a kao rezultat ovaj klasifikator daje verovatnoće pripadnosti ulaznog podatka za skup za koji se podatak testirao. Skalabilnost sistema leži upravo u ovim vektorima atributa, kao i u konačnom rezultatu. U većini slučajeva nije potrebno kreirati kompleksne strukture podataka u cilju konstruisanja klasifikatora „naivni“ Bejz, već je ovo moguće izvesti primitivnim podacima (realni brojevi, karakteri) kao i osnovnim programskim iskazima i programskim petljama.

Pored ovoga, skalabilnost klasifikatora „naivni“ Bejz je i u odabiru konačnog rezultata. Klasifikacije neuronskim mrežama ili klasifikatorom k najbližih suseda podrazumevale bi iscrpljivanje čitavog skupa obučavajućih podataka, što ne mora uvek biti cilj, videti [13]. Nekada nam može biti dovoljno da izračunamo verovatnoću da jedan podatak pripada nekoj određenoj klasi Y_k. U ovim slučajevima „naivni“ Bejz ima veliku prednost, jer bi iskoristio podskup skupa obučavajućih podataka vezanih za traženu klasu. Ovaj klasifikator se često koristi u režimu u kojem se trening podaci za svaku klasifikaciju koriste iznova, odnosno, treniranje klasifikatora se vrši na svaki zahtev klasifikacije. Ovakav pristup se koristi u slučajevima kada nije velik obučavajući skup, mada je moguće parametrizovati klasifikator i na taj način čuvati vrednosti obučavanja klasifikatora za određene instance klasa koje se uzimaju iz skupa poznatih klasifikacija *Y.*

U metodi „naivnog“ Bejza se kao klasifikator zapravo koristi uslovna verovatnoća. Pretpostavimo da imamo problem u kojem moramo

aproksimirati nepoznatu funkciju $f: X \rightarrow Y$, drugim rečima $P(Y|X)$, pod pretpostavkom da Y uzima vrednosti iz skupa {tačno, netačno}, a da X predstavlja vektor od n promenljivih $X = (X_1, X_2, X_3, \dots X_n)$, gde X_i predstavlja vrednost iz skupa {tačno, netačno} i-tog atributa vektora X. Korišćenjem Bejzovog pravila, $P(Y = y_i|X)$ možemo predstaviti na sledeći način, [18]:

$$P(Y = y_i|X = x_k) = \frac{P(X = x_k|Y = y_i)P(Y = y_i)}{\sum_j P(X = x_k|Y = j)\, P\big(Y = y_j\big)}$$

U ovom zapisu y_m predstavlja m-tu moguću vrednost Y, a x_k predstavlja k-tu moguću vrednost vektora vrednosti X. Sumiranje vrednosti u imeniocu sa desne strane jednakosti je urađeno nad svim mogućim *validnim* vrednostima slučajne promenljive Y. Jedan od mogućih načina da se efikasno nauči (aproksimira) $P(Y|X)$ je upravo da se iskoriste obučavajući podaci kako bi mogli da procenimo vrednost $P(X|Y)$ i $P(Y)$. Koristeći ove procene možemo aproksimirati $P(Y|X = x_k)$ za svako novo x_k koristeći Bejzovo pravilo, videti [10].

Ovakvim pristupom potrebno je odrediti odgovarajući broj obučavajućih podataka kako bi došli do dovoljno poverljivih procena verovatnoća za $P(X|Y)$ i $P(Y)$. Pretpostavimo da je 100 nezavisno dobijenih podataka obučavajućeg skupa dovoljno kako bi se dovoljno precizno procenila verovatnoća za $P(Y)$. Uzimajući u obzir koliko parametara je potrebno za procenu $P(Y)$ kada Y uzima vrednosti iz skupa {tačno, netačno}, a X predstavlja vektor promenljivih koje svoje vrednosti takođe uzimaju iz skupa {tačno, netačno}, nije teško shvatiti da je potrebno mnogo više podataka obučavajućeg skupa za efektivnu procenu verovatnoće $P(X|Y)$, ili u našem slučaju [18]:

$$\theta_{ij} \equiv P\big(X = x_i\big|Y = y_j\big)$$

gde i uzima jednu od 2^n mogućih vrednosti jer se mora uzeti jedna od dve moguće vrednosti za svaki od n atributa vektora X, a j uzima dve moguće vrednosti, što dalje implicira da je potrebno proceniti 2^{n+1} parametara.Kako bi izračunali tačan broj potrebnih parametara za neko fiksirano$j$, suma θ_{ij} za svako $i = 1, 2, 3 \dots n$ mora biti jednaka 1:

$$\sum_{i=1, j=z}^{n} \theta_{ij} = 1$$

gde z predstavlja fiksiranu vrednost iz skupa {tačno, netačno}. Imajući ovo u vidu, za bilo koju $y_j$vrednost i za bilo koju x_i vrednost od mogućih 2^n, potrebno je sračunati $2^n - 1$ nezavisnih parametara. S obzirom da je *Y* vrednost iz skupa {tačno, netačno}, znači da moramo izračunati $2 \cdot (2^n - 1)$ ovih θ_{ij} parametara. Ovo je izuzetno nepraktično koristiti u realnim sistemima s obzirom da skup vrednosti vektora *X* je nečesto skup realnih brojeva, a *Y* je retko kada vrednost iz skupa {tačno, netačno}. Upravo zbog ove neodređenosti u praktičnim implementacijama gde ne postoji pravilo iz kakvog skupa vrednosti će *X* i *Y* dobijati svoje vrednosti, potrebno je pronaći način da se redukuje kompleksnost izračunavanja parametara za aproksimaciju funkcije *f.*

Definicija 3.7.[10] *Za date skupove slučajnih promenljivih X, Y i Z kažemo da je X* ***uslovno nezavisna*** *od Y za dato Z, ako i samo ako je vernovatnoća raspodele koja određuje X nezavisna od vrednosti Y za dato Z:*

$$(\forall i, j, k) P\big(X = x_i \big| Y = y_j, Z = z_k\big) = P(X = x_i | Z = z_k)$$

Koristeći definiciju 3.7 uslovnih nezavisnosti i Bejzovog pravila, možemo konstruisati algoritam za klasifikator „naivni“Bejz. Ukoliko nam je cilj da naučimo $P(Y|X)$,gde je $X = (X_1, X_2, X_3, \dots X_n)$, algoritam „naivni“ Bejz se vodi pretpostavkom da je svako X_i uslovno nezavisno od svakog drugog X_k za dato *Y,* a isto tako je nezavisno i od svakog drugog podskupa vrednosti X_k za bilo koje dato *Y.* Pretpostavimo da imamo slučaj kada *X* uzima vrednosti iz skupa vrednosti {tačno, netačno} (predstavljene kao *X* i $\overline{X}$), onda imamo da je:

$$P(X|Y) = P\big(X, \overline{X}\big|Y\big) = P\big(X\big|\overline{X}, Y\big)P\big(\overline{X}\big|Y\big) = P(X|Y)P\big(\overline{X}\big|Y\big)$$

Iz ovog konkretnog slučaja možemo zapisati i opštiji slučaj kada *X* sadrži *n* mogućih atributa od kojih svi zadovoljavaju pretpostavku uslovne nezavisnosti [18]:

$$P(X_1, X_2, X_3, \dots X_n, |Y) = \prod_{i=1}^{n} P(X_i|Y) \qquad (3.1)$$

Pod pretpostavkom da je *Y* bilo koja diskretna promenljiva i da atributi X_1, X_2, X_3, … X_n predstavljaju bilo kakve diskretne ili realne promenljive, možemo obučiti klasifikator tako da dobijemo moguće vrednosti *Y* za svako novo *X*. U saglasnosti sa Bejzovim pravilom, verovatnoća da će *Y* uzeti svoju *k*-tu moguću vrednost je data sa:

$$P(Y = y_i|X_1, X_2, X_3, \dots X_n) = \frac{P(Y = y_k)P(X_1, X_2, X_3, \dots X_n|Y = y_k)}{\sum_j P(Y = y_j)P(X_1, X_2, X_3, \dots X_n|Y = y_j)} \quad (3.2)$$

gde suma u imeniocu predstavlja sumu za svaku moguću y_j vrednost iz skupa *Y.*

Uzimajući da su vrednosti X_i uslovno nezavisne od *Y,* koristeći jednačinu (3.1), jednačinu (3.2) možemo zapisati kao:

$$P(Y = y_i|X_1, X_2, X_3, \dots X_n) = \frac{P(Y = y_k)\prod_i P(X_i|Y = y_j)}{\sum_j P(Y = y_j)\prod_i P(X_i|Y = y_j)} \quad (3.3)$$

što nam daje fundamentalni oblik jednačine za klasifikator „naivni" Bejz. Ova jednačina nam daje način da izračunamo verovatnoću da će *Y* uzeti bilo koju vrednost iz svog zadatog skupa vrednosti za bilo koju novu vrednost $X^{novo} = (X_1, X_2, X_3, \dots X_n)$, uz date raspodele za $P(Y)$ i $P(X_i|Y)$ iz skupa podataka za obučavanje algoritma. Jednačinu (3.3) je moguće zapisati u sledećem obliku ukoliko je potrebno pronaći klasifikaciju sa najvećom verovatnoćom [18]:

$$Y \leftarrow \max_{y_k} \frac{P(Y = y_k)\prod_i P(X_i|Y = y_j)}{\sum_j P(Y = y_j)\prod_i P(X_i|Y = y_j)} \quad (3.4)$$

što se dodatno može skratiti pošto imenilac ne zavisi od y_k:

$$Y \leftarrow \max_{y_k} P(Y = y_k)\prod_i P(X_i|Y = y_j) \quad (3.5)$$

3.2.1. „Naivni“ Bejz sa diskretnim vrednostima

Ukoliko se na ulazu klasifikatora „naivni“ Bejz pojavljuje n atributa sa vrednostima X_i tako da svako X_i uzima jednu od J mogućih diskretnih vrednosti, a vrednost izlaza klasifikatora Y uzima jednu od K mogućih vrednosti, onda je cilj učenja predstavljen aproksimacijom dva skupa parametara gde je prvi dat sa [18] :

$$\theta_{ijk} \equiv P\left(X = x_{ij} \middle| Y = y_k\right) \tag{3.6}$$

a drugi, za prethodne verovatnoće po Y, je dat u obliku:

$$\pi_k \equiv P(Y = y_k) \tag{3.7}$$

U prvom skupu parametara vršimo aproksimaciju za svaki ulazni parametar X_i, za svaku njegovu moguću vrednost x_{ij} i za svaku moguću vrednost y_k za izlaz Y. Ovih parametara ima $n \cdot J \cdot K$, od kojih ima $n \cdot (J - 1) \cdot K$ nezavisnih parametara. U drugom skupu parametara tražimo ukupno K parametara, od kojih su $K-1$ nezavisni parametri, videti [18].

Ovi parametri mogu se proceniti metodom maksimalne verodostojnosti, računanjem relativne frekvencije različitih događaja unutar skupa obučavajućih podataka.

Za zadati skup obučavajućih podataka D i za funkciju $\gamma(x)$ koja određuje broj podataka koji zadovoljavaju uslov x, ocene maksimalne verodostojnosti za skup parametara θ_{ijk} date su sa:

$$\hat{\theta}_{ijk} = \hat{P}\left(X = x_{ij} \middle| Y = y_k\right) = \frac{\gamma\left(X_i = x_{ij} \wedge Y = y_k\right)}{\gamma\left(Y = y_k\right)} \tag{3.8}$$

Ukoliko ne postoje primeri koji zadovoljavaju funkciju γ, znači da će verovatnoća biti jednaka nuli, videti [18]. Kako bi se ovakvi slučajevi izbegli, uvodi se promenljiva l koja predstavlja simulirane primere koji se nisu u realnosti desili, ali se mogu desiti i zadovoljavaju γ i jednako su rasprostranjeni u skupu mogućih vrednosti od X_i.

$$\hat{\theta}_{ijk} = \hat{P}\left(X = x_{ij} \middle| Y = y_k\right) = \frac{\gamma\left(X_i = x_{ij} \wedge Y = y_k\right) + l}{\gamma(Y = y_k) + l \cdot J} \tag{3.9}$$

l·J predstavlja broj simuliranih pridodatih primera.

Ocene maksimalne verodostojnosti za π_k su:

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\gamma(Y = y_k)}{|D|} \tag{3.10}$$

tj.

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\gamma(Y = y_k) + l}{|D| + l \cdot K} \tag{3.11}$$

gde $|D|$ predstavlja kardinalnost obučavajućeg skupa D, K broj različitih vrednosti koje slučajna promenljiva Y može da uzme i l predstavlja jačinu uticaja pridodatih podataka na posmatrani skup D.

3.2.2 „Naivni" Bejz sa neprekidnim vrednostima

Koristeći jednačine (3.3) i (3.5) moguće je kontruisati „naivni" Bejz klasifikator za slučajeve u kojima ulaz X_i ima neprekidne vrednosti, videti [18]. Uobičajeno je u slučajevima kada se radi sa neprekidnim ulaznim vrednostima da se za svaku moguću diskretnu vrednost y_k od Y pretpostavi da je raspodela svake neprekidne vrednosti X_i Gausova raspodela, i da je definisana srednjom vrednošću i standardnom devijacijom koje su specifične za X_i i y_k.

Napomena. Kao što znamo, slučajne promenljive mogu biti diskretnog i apsolutno neprekidnog tipa. Gausova raspodela $\mathcal{N}(m,\sigma^2)$ je raspodela verovatnoća apsolutno neprekidne slučajne promenjive X i njena funkcija gustine je data sa: $\varphi_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}}$, $x\epsilon\mathbb{R}$, $\mathrm{m}\epsilon\mathbb{R}$, $\sigma > 0$.

Da bi obučili klasifikator, za neprekidne vrednosti X_i, moramo najpre izvršiti ocenu srednje vrednosti i standardne devijacije:

$$\mu_{ik} = E[X_i | \, Y = y_k] \tag{3.12}$$

$$\sigma_{ik}^2 = E[(X_i - \mu_{ik})^2 | Y = y_k] \tag{3.13}$$

za svaki mogući atribut X_i kao i za svaku moguću vrednost y_k.Ovih parametara ima 2·n·K i svaki parametar mora zasebno da se proceni. Kao i u slučaju sa diskretnim vrednostima, potrebno je proceniti i prethodne verovatnoće po Y:

$$\pi_k = P(Y = y_k) \tag{3.14}$$

Ovim dobijamo Gausov „naivni" Bejz klasifikator, videti [18].

U ovakvom modelu takođe je moguće, kao i u modelu sa diskretnim ulaznim vrednostima, izvršiti procenu parametara pomoći metode maksimalne verodostojnosti:

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta\left(Y^j = y_k\right) \tag{3.15}$$

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j \left(X_i^j - \hat{\mu}_{ik}\right)^2 \delta\left(Y^j = y_k\right) \tag{3.16}$$

$$\hat{\sigma}_{ik}^2 = \frac{1}{\left(\sum_j \delta(Y^j = y_k)\right) - 1} \sum_j \left(X_i^j - \hat{\mu}_{ik}\right)^2 \delta\left(Y^j = y_k\right) \tag{3.17}$$

U formuli (3.12) se j koristi za obeležavanje j-*t*og podatka u obučavajućem skupu podataka, a $\delta(Y = y_k)$ ima vrednost 1 ukoliko je $Y=y_k$, a nula u suprotnom (ima ulogu da odabere samo one slučajeve u kojima je $Y=y_k$).

4. NEPARAMETRISANE PROCEDURE KLASIFIKACIJE

U prethodnom poglavlju prikazan je način da se konstruiše klasifikacioni model pod pretpostavkom da postoji funkcija koja zavisi od određenih parametara koja mapira ulazne podatke na poznate klasifikacije. Problemi kojima se klasifikacija bavi nemaju uvek rešenje u obliku parametrizovane funkcije čiji parametri moraju da se obuče, već je moguće da funkcija koja vrši mapiranje ulaza na izlazne klasifikacije ne zavisi ni od kakve raspodele, tj. nije parametrizovana.

Naredna poglavlja posvećena su objašnjenjima procedura klasifikacije koje nisu parametrizovane, kao što su Parzenovi Prozori (eng. Parzen Windows, [6]), klasifikacija pomoću histograma (videti [6]) i metodologijom k-najbližih suseda (eng. K-Nearest Neighbors, *KNN,* videti [4], [6], [14]).

4.1. PROCENA GUSTINE HISTOGRAMIMA

Najjednostavniji način procene gustine je upravo uz pomoć histograma, videti [6]. Uzorak koji služi za klasifikovanje novih podataka se deli u disjunktne kategorije, i procena gustine se vrši tako što se broji koliko podataka pripada svakoj kategoriji. Neka C_m predstavlja m-tu kategoriju i neka h predstavlja širinu kategorije, i neka $\gamma(X_i \in C_m)$ predstavlja ukupan broj podataka iz skupa $X = (X_1, X_2, X_3, \dots, X_n)$ koji pripadaju odgovarajućoj kategoriji C_m širine h. U ovom slučaju, odgovarajuća procena gustine data je sa [6]:

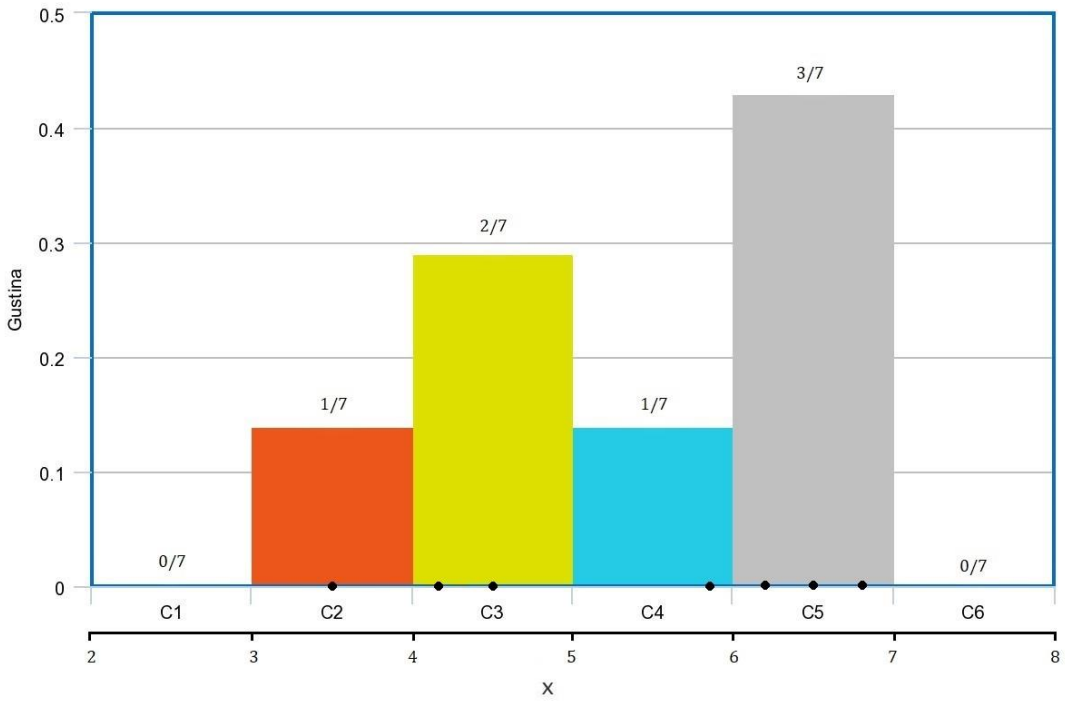
$$\hat{f}(x) = \frac{\gamma(X_i \in C_m)}{n \cdot h} = \frac{k}{n \cdot h} \tag{4.1}$$

za svaku kategoriju $x \in C_m$ pod uslovom da je m prirodan broj. U formuli (4.1) $f(x)$ predstavlja pravu gustinu (koja je najčešće nepoznata), a $\hat{f}(x)$ predstavlja procenu prave gustine.

Primer 4.1.[6]Neka je zadat skup od sedam podataka

$$\{X_1 = 3.5\,, X_2 = 4.2\,, X_3 = 4.5\,, X_4 = 5.8, X_5 = 6.2, X_6 = 6.5\,, X_7 = 6.8\,\}$$

observacijom nekih događaja i neka je startna pozicija histograma $X_0 = 0$, a širina histograma h = 1.



SLIKA 4.1. PRIMER HISTOGRAMA SA UZORKOM OBIMA n=7

Ako pristupamo problemu pomoću histograma, potrebno je definisati parametre širine histograma h i početak histograma X_0. U ovom slučaju sa 6

klasifikacija, veliki problem upravo predstavlja činjenica da su klase *C1* i *C6* prazne. U praktičnim primerima sa mnogo više klasa, ako obučavajući skup nema dovoljno podataka, dosta klasifikacija ostaće prazno. Ovakav pristup retko se koristi u praktičnoj primeni, ali može poslužiti za vizuelizaciju odabranih slučajeva koji imaju mali broj kategorija.

Još jedan nedostatak ovog pristupa je početna pozicija kategorija koja ima velik značaj u formiranju slike histograma, videti [6]. Jedan način da se ovaj problem izbegne je pomeranjem histograma. Ovako pomeren histogram naziva se *srednje pomeren histogram (eng. average shifted histogram ASH,* videti [6]*)*. Ovakvim pristupom dobijamo bolju vizuelizaciju histograma, bolju aproksimaciju vrednosti, a vreme izvršavanja i računanja je gotovo isto kao i za klasičan pristup.

Algoritam srednjeg pomeranja histograma služi primarno za izbegavanje nedostataka za procenu gustine pomoću histograma usled nasumičnog odabiranja vrednosti startne pozicije histograma X_0. Ovakvo ocenjivanje uzima više klasičnih histograma sa različitim početnim pozicijama i usrednjava njihove vrednosti. Kolekcija od m histograma se konstruiše, od kojih svaki histogram ima širinu kategorije h_i, ali svaki ima pomerenu startnu vrednost. Zatim računamo srednju vrednost ovako konstruisanih histograma:

$$\hat{f}_{ASH}(x) = \frac{1}{m}\sum_{i=1}^{m} \hat{f}_i(x) \tag{4.2}$$

4.2. GENERALNA NEPARAMETRISANA PROCENA GUSTINE

Pretpostavimo da imamo uzorak veličine n: $X_1, X_2, X_3, \ldots, X_n$, gde su X_i, i=1,….,n nezavisne i imaju jednaku raspodelu sa funkcijom gustine p(x) koja je nepoznata. Potrebno je pomoću ovih n odabranih podataka izabrati tačku x i izvršiti procenu gustine u njoj. Ako R predstavlja d-dimenzionu hiperkocku oko tačke x sa dužinom stranice h i zapreminom V = h^d, tada *je* verovatnoća da će podatak za trening upasti u region R, videti [6]:

$$P = Pr[x \epsilon R] = \int_R p(x)\, dx \tag{4.3}$$

Poznato je da je verovatnoća da će *k* od ovih *n* podataka upasti u *R* data binomnom raspodelom:

$$\Pr(k; n, P) = Pr(X = k) = \binom{n}{k} P^k (1 - P)^k \tag{4.4}$$

Znajući osobine ove raspodele poznato je da je srednja vrednost i varijansa razmere *k / n* data sledećim jednačinama:

$$\begin{aligned} E\left(\frac{k}{n}\right) &= P \\ Var\left(\frac{k}{n}\right) &= \frac{P(1-P)}{n} \end{aligned} \tag{4.5}$$

Kada $n \to \infty$ tada i varijansa teži nuli, i može se očekivati da će srednja frakcija tačaka koje upadaju u okviru *R* biti dobra procena verovatnoće *P*:

$$P \cong \frac{k}{n} \tag{4.6}$$

Ukoliko se pretpostavi da je prostor *R* mali, toliko da gustina *p(x)* ne varira previše unutar prostora *R*, onda se integral (4.3) može predstaviti sa:

$$P = \int_R p(x)\, dx \cong p(x) \cdot V \tag{4.7}$$

gde *V* označava zapreminu regiona *R.*

Ovim dobijamo:

$$p(x) \cong \frac{k}{n \cdot V} \tag{4.8}$$

Konvergencija u verovatnoći jednačine (4.8) ka pravoj gustini f(x) data je sa:

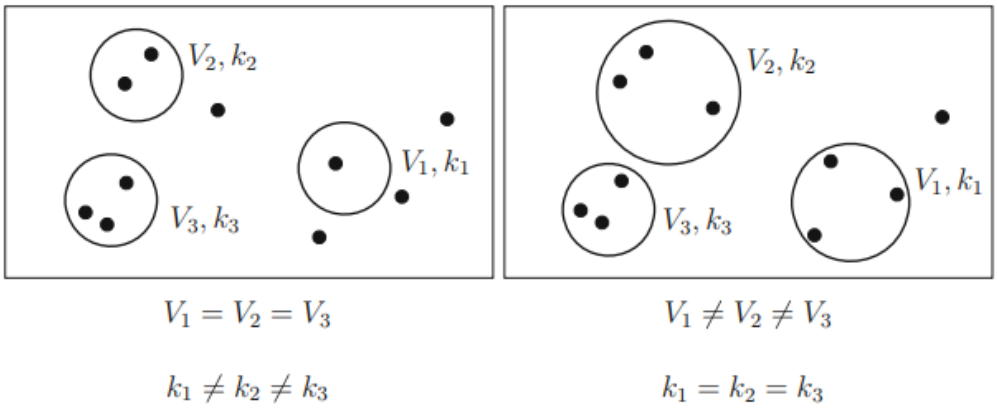
$$f(x) = \lim_{n\to\infty} p(x) = \lim_{n\to\infty} \frac{k}{nV} \tag{4.9}$$

Potrebni uslovi za konvergenciju dati su sa:

$$\begin{gathered} \lim_{n\to\infty} V_n = 0 \\ \lim_{n\to\infty} k_n = b \\ \lim_{n\to\infty} \frac{k_n}{n} = 0 \end{gathered} \tag{4.10}$$

U praksi je *n* uvek predodređena konstantna vrednost, videti [6]. Kako raste vrednost *n* i kako se vrednost *V* smanjuje, tako je i procena gustine sve bolja i preciznija. Ovo naravno vodi ka tome da se smanji *V* na minimalnu vrednost kako bi procena *f(x)* bila maksimalno precizna. Međutim, ovo nije najpametnije uraditi jer bi tada u region *R* upalo 0 podataka za trening. Upravo ovo vodi do problema pronalaženja najboljeg *V*, tako da se uzme dovoljno velik obučavajući skup podataka, ali da region ne bude prevelik kako ne bi narušio procenu [6].

Ovo podešavanje vrednosti *V* dovodi do 2 metodologije u neparametrisanoj proceni gustine. Počevši od formule (4.8), koja predstavlja tzv. naivno ocenjivanje gustine, moguće je fiksirati vrednosti za *V* ili *k.* Ako je V=h (tj.d=1), tada se jednačina (4.8) svodi na jednačinu (4.1) iz definicije histograma. Fiksirajući *V* na određenu vrednost, dobijamo metodologiju *Parzenovih Prozora* (eng. *Parzen Windows,* videti [6]), iz koje određujemo *k* iz datih podataka. Fiksirajući vrednost *k* dobijamo metodologiju nazvanu *K-Najbližih Suseda* (eng. *K-Nearest Neighbors*, videti [4], [6], [13), odakle se određuje zapremina *V* (ili klasifikacija C, u slučajevima mašinskog učenja i klasifikacije).



SLIKA 4.2. PARZENOVI PROZORI (LEVO) I K-NAJBLIŽIH SUSEDA (DESNO)

4.3. PARZENOVI PROZORI

Pod pretpostavkom da je region *R* hiperkocka sa širinom stranice *h* i da čitav region *R* obuhvata svih *k* podataka, očigledno je da je zapremina *V* ove hiperkocke data sa $V=h^d$, gde je *d* dimenzija prostora problema koji posmatramo. Centrirajući region *R* u nekoj tački *x* i prebrojavanjem podataka u regionu *R,* možemo izvršiti procenu nepoznate gustine *f(x)* u tački *x.* Ovo je analitički dato sa:

$$K(u) = \begin{cases} 1, & |u_i| \leq 1/2, \quad \forall i = 1,2 \dots d \\ 0, & inače \end{cases} \tag{4.11}$$

Ova funkcija predstavlja *kernel funkciju* ili *funkciju prozora (eng. kernel function*, videti [6]*).* Njen prikaz u jednoj i dve dimenzije u sklopu jedinične hiperkocke je dat na slici 4.1. Parzenov prozor predstavlja jediničnu kocku koja je konstruisana pomoću funkcije prozora i postavljena sa centrom u tački *x.* Prebrojavanje tačaka *k* u prostoru hiperkocke sa širinom stranice *h*, i centrom u *x,* može se iskoristiti:

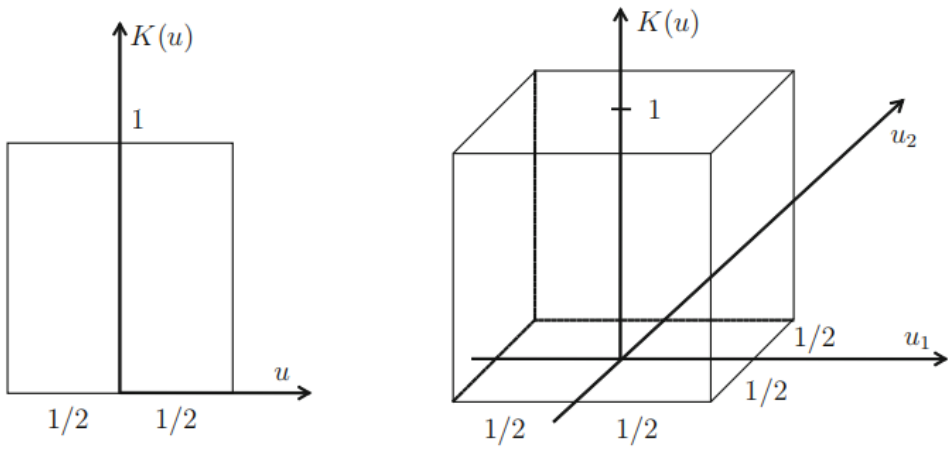
$$k = \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{4.12}$$

za ulazni uzorak veličine n: $X_1, X_2, \dots, X_n$. Uzimajući u obzir (4.8), moguće je formulisati analitički oblik aproksimacije gustine $f(x)$ sa:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{4.13}$$

Metod histograma i Parzenovog prozora predstavlja vrlo sličan pristup u estimaciji gustine sa razlikom u činjenici da je startna pozicija kod drugog pristupa zadata sa ulaznim podacima. Sada funkciju (4.9) možemo zapisati u sledećem obliku [6]:

$$K\left(\frac{x - X_i}{h}\right) = \begin{cases} 1, & |x - X_i| \le h/2, \quad \forall i = 1,2 \dots d \\ 0, & inače \end{cases} \tag{4.14}$$



SLIKA 4.1. JEDINIČNE HIPERKOCKE U JEDNOJ DIMENZIJI I DVE DIMENZIJE

4.4. K-NAJBLIŽIH SUSEDA

Metoda najbližih suseda dodeljuje neklasifikovanu tačku u klasu najbližeg skupa prethodno klasifikovanih tačaka. Literatura korićena u ovom poglavlju je [4], [6], [14]. Prvu formulaciju pravila k-najbližih suseda i analizu njegovih osobina izvršili su Fiks i Hodžs (*Fix and Hodges)* 1951. godine, videti [3].

U ovoj metodologiji koristi se fiksirana vrednost za k a određuje se vrednost zapremine V. Procena gustine se vrši sa [6]:

$$p(x) = \frac{k}{n \cdot V} = \frac{k}{nc_d R_k^d(x)} \tag{4.15}$$

gde R_k^d predstavlja rastojanje između tačke x na osnovu koje se vrši procena i njenog k-tog najbližeg suseda, a c_d je zapremina jedinične sfere sa d dimenzija koja je data sa [6]:

$$c_d = \frac{\pi^{d/2}}{(d/2)!} = \frac{\pi^{d/2}}{\Gamma(d/2+1)} \tag{4.16}$$

Problem kod metodologije k-najbližih suseda je što je metodologija osetljiva na lokalne smetnje u obučavajućem skupu, videti [6].

U mašinskom učenju i klasifikaciji, k-najbližih suseda predstavlja 'lenju' metodologiju (eng. *Lazy Learning technique*, videti [4]), ili metodologiju baziranu na primerima. Nadimak 'lenja' metodologija dobija iz činjenice što se učenje ne vrši pre nego što se izvrši klasifikacija, već u trenutku kada je klasifikacija potrebna odnosno kada je potrebno konkretan podatak klasifikovati. Bazirana je na primerima jer direktno zavisi od podataka iz obučavajućeg uzorka podataka.

Klasfikacija se vrši u dva koraka, videti [4]. Prvi korak predstavlja određivanje najbližih suseda određenog podatka pomoću nekih mera distance atributa podatka, a drugi korak predstavlja određivanje konkretne klasifikacije podatka na osnovu broja pronađenih suseda.

Pretpostavimo da imamo obučavajući uzorak D koji ima n podataka na osnovu kojih klasifikator može da vrši klasifikaciju. Ovi podaci imaju atribute koji uzimaju vrednosti iz skupa F i svaki numerički atribut je normalizovan tako da uzima vrednosti u opsegu $[0, 1]$. Svaki podatak u obučavajućem skupu je obeležen tako da se zna kojoj klasifikaciji pripada, sa $y_i \in Y$. Potrebno je klasifikovati nepoznati podatak q. Moguće je izračunati udaljenost između q i svakog $x_i \in D$ na sledeći način [4] :

$$d(q, x_i) = \sum_{f \in F} w_f \delta\left(q_f, x_{if}\right) \tag{4.17}$$

Što se tiče određivanja udaljenosti između traženog podatka i jednog podatka iz obučavajućeg skupa postoji mnogo načina da se nađe ova udaljenost, od kojih je jedan vrlo jednostavan:

$$\delta(q_f, x_{if}) = \begin{cases} 0, & f\ je\ diskretna\ vrednost\ i\ q_f = x_{if} \\ 1, & f\ je\ diskretna\ vrednost\ i\ q_f \neq x_{if} \\ |q_f - x_{if}|, & f\ je\ kontinualna\ vrednost \end{cases} \quad (4.18)$$

Susedi se biraju na osnovu željenog načina, a nakon toga biramo klasifikaciju za novi podatak q, a najjednostavniji način jeste da se odabere većina 'glasova' od pronađenih suseda.

Naravno nije uvek najbolje odabrati klasu koja ima najviše suseda traženom podatku. U određenim slučajevima se bližim susedima pojačavaju rezultati računanja distanca tako da imaju bolji uticaj u glasanju za najbolju klasifikaciju [4]:

$$G(y_i) = \sum_{c=1}^{k} \frac{1}{d(q, x_c)^n} 1(y_j, y_c) \quad (4.19)$$

Iz ovoga vidimo kako klasifikacija y_i dobija glas od suseda x_c u obliku količnika od 1 i distance do suseda x_c. $1(y_i, y_j)$ vraća 1, ako se klase poklapaju, a 0 u suprotnom. U ovoj jednačini se obično parametar n postavlja na 1, međutim povećavajući ovu vrednost možemo smanjiti uticaj daljih suseda.

Kako bi videli kako se k-najbližih suseda može koristiti u praksi, potrebno je definisati sličnost i blizinu klasifikacija i načine na koje se mogu računati, na osnovu vrednosti klasifikacija, a za to nam je potrebna sledeća definicija.

Definicija 4.1.[15] *Metrika nad skupom X je funkcija d:* $X \times X \to [0, \infty)$ *takva da su za svako* $x, y, z \in X$ *ispunjeni sledeći uslovi:*

- $d(x, y) \geq 0$; nenegativnost,
- $d(x, y) = 0$ *ako i samo ako je* $x = y$; identitet
- $d(x, y) = d(y, x)$; simetrija
- $d(x, z) \leq d(x, y) + d(y, z)$; trougaona nejednakost

Ipak, moguće je konstruisati takav model klasifikatora k-najbližih suseda koji podržava određenu karakteristiku, a da ona nije prava metrika, ali takav klasifikator bi imao problema sa optimizacijom rada.

Definicija 4.2. [15] Minovski metrika (eng. *Minowski Distance Metric*) je distanca reda p između n-dimenzionih tačaka x=(x_1, x_2, … x_n) i y=(y_1, y_2, … y_n), čija je opšta formula data sa:

$$d(x,y) = \left(\sum_{i=1}^{n} |(x_i - y_i)^p| \right)^{\frac{1}{p}} \tag{4.20}$$

L1 Minovski distanca za p=1 predstavlja *Menhetnovu distancu* (eng. *Manhattan Distance)* ili 1-normu, a L2 Minovski distanca za p=2 predstavlja *Euklidovo rastojanje (eng. Euclidian Distance)* ili 2-normu. Obično se ne koriste veće vrednosti od 2 za parametar *p*, ali kada se koriste onda je u svrhu dobijanja efekta da se doda veća težina na atribute koji se najviše razlikuju.

Još jedna bitna Minovski distanca je L_∞, a to je Čebiševa distanca (eng. *Chebyshev Distance*, videti [4]).

$$d(x,y) = \max_{i \in \{1,2 \ldots n\}} |x_i - y_i| \tag{4.21}$$

Ovo predstavlja distancu u dimenziji kada su dva podatka najdalja, odnosno kada se najviše razlikuju.

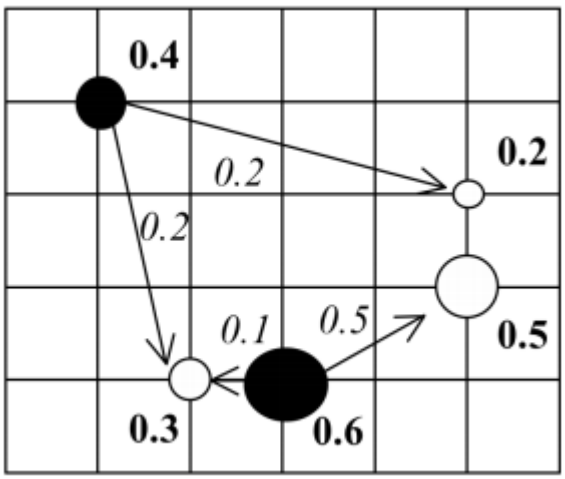
Opšta formula za Minovski distancu predstavlja metriku koja se može iskoristiti u klasifikatoru k-najbližih suseda sa proizvoljnim brojem atributa koji se koriste za određivanje rastojanja između klasifikovanog i novog podatka. Klasifikator k-najbližih suseda se često koristi u domenu digitalne obrade slika. Slike bi se pretvorile u nijanse sive boje, a klasifikator bi poslužio u pronalaženju sličnosti u histogramima sive boje. Histogram *H* sa *N* nivoa sive boje gde je h_i broj piksela koji upadaju u interval predstavljen sa *i*. Upravo ova promenljiva *h* predstavlja vektor atributa za klasifikaciju. Pored korišćenja Minovski distance za klasifikaciju ulaznog piksela, može se koristiti i *Kulibak-Lajbler divergencija (Kullback-Leibler)* ili χ^2 *statistika.*

$$d_{KL}(H,K) = \sum_{i=1}^{N} h_i \log\left(\frac{h_i}{k_i}\right)$$
$$d_{\chi 2}(H,K) = \sum_{i=1}^{N} \frac{h_i - m_i}{h_i} \tag{4.22}$$

gde H i K predstavljaju dva histograma, h i k predstavljaju odgovarajuće vektore klasifikacija histograma, a m_i predstavlja $\frac{h_i+k_i}{2}$.

Dva bitna načina za procenu daljine su Distanca Pomeranja Zemlje*(eng. Earth Mover Distance – EMD,* videti [4]*)* i *Razlika Bazirana na Kompresiji (eng. Compression Based Dissimilarity – CBD,* videti [4]*).*

Pretpostavimo da imamo dve raspodele nekih podataka. Kada bi prvu raspodelu gledali kao skup rupa, a drugu raspodelu kao zemlju, EMD upravo predstavlja minimalan napor da se sve rupe ispune zemljom. EMD je mera koja se bazira na obeležjima, a ne na histogramima. *Obeležje* (eng. *signature,* videti [4]) u ovoj meri predstavlja skup od j grupa $\{s_j = m_j, w_{m_j}\}$ tako da svako m_j predstavlja vektor koji opisuje vrednosti grupe j a w_{mj} predstavlja frakciju podataka koji upadaju u tu grupu: $\{s_j = m_j, w_{m_j}\}$.Ovo grupisanje možemo posmatrati kao kvantizaciju jedne slike tako da je ona predstavljena vrednostima grupa sa težinama.



SLIKA 4.3 DISTANCA IZMEĐU JEDNE SLIKE SA 2 (CRNE) GRUPE I JEDNE SLIKE SA 3 (BELE) GRUPE

Na slici 4.2 predstavljene su dve grupe od koji se prva, crna, grupa sastoji od dve tačke koje imaju težine 0.4 i 0.6 a druga grupa, bela, se sastoji od tri tačke sa težinama 0.2, 0.5, i 0.3. Pretvaranje prve grupe u drugu grupu predstavlja pomeranje težina, tako da se 0.2 od grupe sa težinom 0.4 pomeri ka grupi koja ima težinu 0.3 i još jednom ka grupi koja ima težinu 0.2. Na ovaj način je bela grupa koja ima težinu 0.2 ispunjena. Od grupe sa težinom 0.6, možemo uzeti 0.1 i zajedno sa onih 0.2 koje smo uzeli od grupe sa težinom 0.4, tako da ispunimo belu grupu sa težinom 0.3. Preostalih 0.5 od grupe sa prethodnom težinom od 0.6 možemo iskoristiti kako bi ispunili belu grupu sa težinom 0.5. Sumiranjem proizvoda ovih težina koje su pomerene (0.2, 0.2, 0.1, 0.5) i udaljenosti za koje su te težine pomerene, dobijamo kalkulaciju za *EMD*, i potrebno je još minimizovati ovaj rad pomeranja težina. Ako imamo slike obeležene sa *S* i *Q*:

$$S = \left\{m_{j,} w_{m_j}\right\}_{j=1}^{n} \tag{4.23}$$

i

$$Q = \left\{p_k, w_{p_k}\right\}_{k=1}^{r} \tag{4.24}$$

mi tražimo koliko je potrebno napora da se prebaci iz jedne slike u drugu sliku za zadati šablon *F*:

$$N(S, Q, F) = \sum_{j=1}^{n} \sum_{k=1}^{r} d_{jk} f_{jk} \tag{4.25}$$

gde je d_{jk} distanca između grupa m_j i p_k, a f_{jk} predstavlja tok između m_j i p_k tako da je cena minimalna. Određivanje toka koji zahteva najmanje napora može se naći rešavanjem problema linearnog programiranja, što daje sledeću jednačinu:

$$N(S, Q) = \frac{\sum_{j=1}^{n} \sum_{k=1}^{r} d_{jk} f_{jk}}{\sum_{j=1}^{n} \sum_{k=1}^{r} f_{jk}} \tag{4.26}$$

Razlika Bazirana na Kompresiji ili *CBD* se može demonstrirati na primeru upoređivanja dva kompresovana digitalna dokumenta, videti [4]. Ukoliko dva dokumenta imaju visok stepen sličnosti, to znači da će zbir veličina

kompresovanih dokumenata biti za malu vrednost manja od veličine jednog kompresovanog dokumenta od ta dva.

Ova metrika bazirana je na uslovnoj Kolmogorovljevoj kompleksnosti:

$$d_{K_v}(x,y) = \frac{K_v(x|y) + K_v(y|x)}{K_v(xy)} \tag{4.27}$$

Ovde $K_v(x|y)$ predstavlja najkraću dužinu koja je potrebna programu da izračuna x kada je y unešeno kao ulaz u program, a $K_v(xy)$ je najkraća dužina koja je potrebna programu da se izračuna spajanje y i x . Ovu ideju možemo konkretizovati izrazom:

$$d_C(x,y) = \frac{C(x|y) + C(y|x)}{C(xy)} \tag{4.28}$$

u kojem je $C(x)$ veličina datoteke x nakon kompresije, a $C(x|y)$ je veličina datoteke x nakon što je izvršena kompresija na isti način kao što je kompresovan dokument y.

$$d_{NC}(x,y) = \frac{C(xy) - \min(C(x), C(y))}{max(C(x), C(y))} \tag{4.29}$$

Jednačina (4.26) može definisati ukoliko pretpostavimo da je $K_v(x|y) \approx K_v(xy) - K_v(y)$ i predstavlja normalizovanu distancu kompresije.

Ove metrike su pogodne za korišćenje u određenim slučajevima kada se radi sa multimedijalnim podacima kao što su slike i dokumenti. One su takođe doprinele identifikaciji problema u klasifikaciji metodologijom najbližih suseda, što je inače skupo izvršavanje računice. Zbog ovoga je potrebno smanjiti obučavajući skup tako da se ne izgubi previše bitnih podataka, već izbaciti takve podatke koji dovode do šuma ili podatke koji previše odstupaju od distribucije. Pored skraćivanja obučavajućeg skupa, moguće je i smanjivanje dimenzije problema tako da se ne uračunavaju atributi koji nisu od prevelikog značaja u obučavajućem skupu.

4.4.1. REDUKCIJA DIMENZIJA PODATAKA

Redukovanje dimenzija predstavlja bitan deo u obradi multimedijalnih podataka kao što su slike i digitalni dokumenti, videti [4]. Prvi vid redukovanja dimenzija jeste odbacivanje smetnji u obučavajućem skupu (eng. *noise reduction*). Ovo su podaci koji ne spadaju u raspodelu obučavajućeg skupa podataka, ili podaci koji su ubačeni u trening skup zbog greške uređaja ili laboranta koji je sakupljao podatke. Još jedan način redukcije podataka jeste redukcija broja atributa koji se uzimaju u obzir prilikom klasifikacije.

Redukcija broja atributa za razmatranje je proces poznatiji pod nazivom *Feature Selection.* Redukcija atributa ne mora značiti ignorisanje, već može i kompletan skup atributa da se izbaci iz obučavajućeg skupa. Kada koristimo klasifikator *k-najbližih-suseda* postoji više razloga zbog kojih bi nam poslužila redukcija atributa [4]:

1. Za neke metrike distanci, sam korak klasifikacije duže traje što je više atributa,
2. Šumovi u skupu sa obučavajućim podacima se uzimaju kao legitimni podaci koji će isto uticati na klasifikaciju kao i oni.
3. U opštem slučaju, što više atributa ima na osnovu kojih se može vršiti klasifikacija, to je veća šansa da primeri u obučavajućem skupu deluju slično, što će negativno uticati na određivanje klase.

Glavne operacije prilikom redukcije podataka su pretraga atributa i istraživanje sadržaja podataka. Ova istraživanja se dele na strategije filtriranja podataka i na strategiju obavijanja podataka.

Strategije filtriranja podataka obuhvataju izbacivanje nerelevantnih atributa iz skupa atributa modela podataka za obučavanje klasifikatora, pre nego što se pređe u fazu obuke klasifikatora. Pre bilo kakvog učenja i filtriranja se čitav skup podataka analizira kako bi se pronašli oni podaci koji su najbitniji za opisivanje modela podataka u trening skupu. Ovaj proces se može vršiti rekurzivno, nakon čega odabrani skup atributa prolazi u fazu obučavanja klasifikatora. Strategije obavijanja podataka vrše predikciju preciznosti ocenjivanja koju klasifikator može dati za određeni podskup skupa atributa. Na osnovu ovih procena, ova metodologija traži maksimalno preciznu procenu i rekurzivno vrši ponovnu procenu, sa dodatno smanjenim skupom atributa.

4.4.2. STRATEGIJE FILTRIRANJA

Glavna karakteristika strategija filtriranja atributa predstavlja način na koji se boduje koliko je dobra klasifikacija nakon odabira novih atributa, videti [4]. Informacioni Dobitak (eng. *Information Gain - IG)* je možda najpopularniji kriterijum za odabir atributa i predstavlja meru ukupne količine podataka kojom određeni atribut doprinosi obučavajućem skupu. Ukoliko imamo skup obučavajućih podataka *D*, *IG* se definiše kao ukupno očekivano smanjenje neuređenosti koje se može postići particijom skupa *D* korišćenjem atributa *f:*

$$IG(D,f) = Ent(D) - \sum_{v\ \epsilon\ skup\ vrednosti(f)} \frac{|D_v|}{|D|} Ent(D_v) \qquad (4.30)$$

gde D_v predstavlja podskup obučavajućeg skupa *D* u kojem atribut *f* ima vrednost *v. Ent(x)* je funkcija koja daje broj koji označava meru neuređenosti u prosleđenom skupu podataka *x:*

$$Ent(x) = \sum_{i=1}^{c} -p_i \log_2 p_i \qquad (4.31)$$

u kojoj je *c* broj klasifikacija u skupu podataka a p_i je proporcija *i*-te klase.

Još jedan poznatiji kriterijum filtriranja podataka je *odnos kvota (eng. Odds Ratio – OR,* videti [4]*),* koji računa količnik verovatnoće da će se neki atribut *f* pojaviti u određenoj klasi *c* i verovatnoće da se isti taj atribut ne pojavi u toj klasi.

$$OR(f,c) = \frac{P(f|c)}{P(f|\overline{c})} \qquad (4.32)$$

Za odabir atributa vrši se rangiranje *OR* vrednosti, gde visoke vrednosti ukazuju na atribute koji snažno utiču na predikciju klase.

Iako je filtriranje vrlo efektivan način za odabir atributa prilikom treniranja klasifikatora, postoji problem nezavisnosti atributa prilikom filtriranja. Svaki atribut u skupu podataka se izolovano razmatra i to dovodi do redundantnosti odabira atributa, ili do ignorisanja zavisnosti atributa. Ovo znači da ukoliko dva atributa imaju vrlo visoke ocene za određivanje klasifikacije, može doći do

redundantnosti jednog od ta dva atributa ukoliko se gledaju kao skup atributa, odnosno, uz prisustvo prvog atributa nije potrebno razmatrati drugi atribut. Slično ovome, ukoliko dva atributa imaju vrlo niske ocene za predikciju klasifikacije, neće biti odabrani u finalni model klasifikatora, a kada bi se zajedno razmatrali imali bi jak uticaj na preciznost klasifikatora.

4.4.3. TEHNIKE OBAVIJANJA

Ključna karakteristika tehnika obavijanja je što performanse čitavog klasifikatora određuju kako trebaju atributi da se odabiraju – tehnika *obavija* klasifikator u proces odabira atributa. Problemi koje su imale strategije filtriranja kao što je individualnost atributa su rešene u ovoj strategiji jer se klasifikator posmatra kao jedna celina, sa svim vezama između atributa. Međutim, sama ova činjenica da se čitav klasifikator koristi za testiranje znači da se mora konstruisati više klasifikatora i isti skup provera izvršiti više puta, što je računski izuzetno skupo. Ukoliko imamo skup podataka koji imaju po p atributa, postavlja se pitanje da li će se uzeti atribut ili ne. To znači da je potrebno konstruisati 2^p slučajeva, i za svaki izvršiti celokupnu proceduru testiranja i treniranja modela kako bi se dobile sve predikcije modela.

Popularne metode testiranja modela su selekcija unapred i eliminacija unazad, videti [4]. U selekciji unapred, najpre se odbace svi atributi i vrši se evaluacija modela za po jedan atribut po modelu. Kada se pronađe najbolji rezultat, atribut se čuva, i u taj skup se dodaje po još jedan atribut od preostalih i tako se rekurzivno vrši odabir atributa. U eliminaciji unazad je suprotan proces: počinje se od svih atributa u inicijalnom modelu i odbacuju se atributi nakon svakog testiranja. Ovi procesi se mogu završiti kada dođe do stagniranja rezultata ili narušavanja rezultata, a može se i završiti ako se ne ispune dovoljno dobri rezultati u poboljšanju (za veliki model, dodavanje atributa zbog poboljšanja procene za na primer 0.5% se ne isplati zbog kompleksnosti i vremena izvršavanja predikcije).

I tehnika filtriranja i obavijanja spadaju u tehnike koje će uzeti što više i uvek preferirati da uzimaju velik broj atributa zbog manjeg unapređenja u performansama modela, zbog ovoga se često nazivaju 'pohlepne' strategije.

Zbog svoje jednostavne implementacije, kao i zbog svoje skalabilnosti i adaptivnosti za različite slučajeve, klasifikator k-najbližih suseda može se iskoristiti u svrhe rešavanja širokog spektra problema, videti [4], [6], [14]. Sam proces implementacije i klasifikacije je vrlo transparentan i jednostavno se može naučiti ili objasniti. Postoje određene tehnike uklanjanja šuma iz

skupa obučavajućih podataka koje se mogu primeniti samo na ovaj klasifikator. Još jedna prednost ovog klasifikatora je to što se može lako objasniti razlog iz kog je određeni izlaz dobijen, upravo analizom suseda dobijenog rezultata.

Iako je ovaj način klasifikacije toliko jednostavan za implementaciju i razumevanje, postoje i određene mane ovog klasifikatora. Glavni problem klasifikatora je upravo što se sve procene i proračuni vrše u toku izvršavanja, ne postoji faza u kojoj se čuvaju određena stanja klasifikatora. S obzirom da čitav skup podataka koji je korišćen za trening učestvuje u određivanju finalne predikcije, znači da nepotrebni ili netačni podaci podjednako utiču kao i korektni podaci, što može dovesti do ozbiljnog pada efektivnosti klasifikatora i iz tog razloga je potrebno filtrirati skup obučavajućih podataka.

5. VESTAČKE NEURONSKE MREŽE

Neuronska mreža sačinjena je od *čvorova* koji su spojeni *vezama*. Ove veze sadrže numeričke vrednosti koje nazivamo *težinama.* Upravo te težine predstavljaju način na koji možemo trajno čuvati informacije o stanju neuronske mreže i učenje neuronske mreže se vrši upravo ažuriranjem ovih težina. Čvorovi su postavljeni tako da čine slojeve, gde su određeni slojevi povezani sa spoljašnjim sistemom i čine ulazne i izlazne slojeve, dok ostali slojevi formiraju takozvani *skriveni* sloj između ulaznih i izlaznih slojeva neuronske mreže. Neuronska mreža se obučava tako što se ažuriraju vrednosti težina na vezama između neurona, u cilju da se vrednosti na ulaznom sloju transformišu kako prolaze kroz neuronsku mrežu tako da se na izlaznom sloju dobiju očekivane vrednosti obučavajućeg skupa.

Svaki čvor ima skup ulaznih veza koje sadrže težine, skup izlaznih veza sa težinama, kao i način da izračuna svoju *aktivacionu vrednost* za sledeći trenutak u vremenu u zavisnosti od vrednosti na ulaznim vezama i svoju trenutnu aktivacionu vrednost. Ideja jeste da se izbegne briga o čitavoj neuronskoj mreži i o svim proračunima u celini, već da svaki čvor u neuronskoj mreži obavlja lokalne proračune na osnovu susednih ulaznih

vrednosti. Konstruisanje neuronske mreže podrazumeva donošenje odluke o tome koliko neurona je potrebno koristiti i kako poređati neurone u slojeve.

Naredna poglavlja detaljnije opisuju veštačke neuronske mreže, osnovne delove koji čine neuronske mreže kao što su perceptroni (eng. *Perceptron*, videti [13]), i algoritme ulančavanja unapred (eng. *feed forward*, videti[13]), i ulančavanja unazad (eng. *backpropagation*, videti [13]). Naredna poglavlja bazirana su na [5], [7], [9], [13].

5.1. OSNOVNI KONCEPTI NEURONSKIH MREŽA

Kao što je prethodno pomenuto, svaka neuronska mreža u osnovi sadrži neurone i veze između neurona, videti [13]. Svaki neuron vrši lokalni proračun tako što dobije vrednosti iz ulaznih veza (ulazni signali) i rezultat proračuna propagira kroz svaku od svojih izlaznih veza. Ovaj rezultat se često naziva aktivaciono *stanje* ili aktivacioni *nivo,* videti [7] .Konačna vrednost aktivacionog stanja direktno zavisi od vrednosti na ulazima neurona i od težina na svakom ulazu neurona. Račun je podeljen u dva koraka ([5], [13]):

1. Prvi korak predstavlja korak izračunavanja linearne vrednosti in_i , što podrazumeva množenje vrednosti na ulaznim vezama neurona sa težinama na istim vezama i sumiranje dobijenih rezultata.
2. Drugi korak je korak izračunavanja nelinearne vrednosti aktivacione funkcije neurona, g. Aktivaciona funkcija može imati različit oblik u zavisnosti od konkretnog problema. Iako se aktivaciona funkcija uzima shodno problemu, obično kada se odabere aktivaciona funkcija ona se koristi u čitavoj neuronskoj mreži, odnosno svaki neuron računa svoj aktivacioni nivo a_i koristeći istu aktivacionu funkciju**.**

Ukupna linearna vrednost za ulazne vrednosti je:

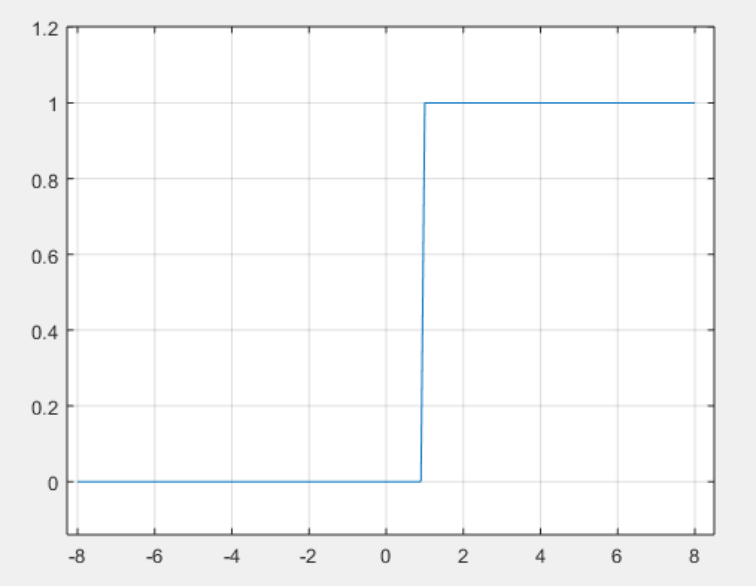
$$in_i = \sum_j w_{j,i} a_j \tag{5.1}$$

gde je $w_{i,j}$ težina na vezi od neurona *j* do neurona *i,* a a_j predstavlja aktivacionu vrednost *j*-tog neurona. Nakon ove linearne računice dolazi korak računanja aktivacionog nivoa neurona koristeći odabranu aktivacionu funkciju *g (*[13]*):*

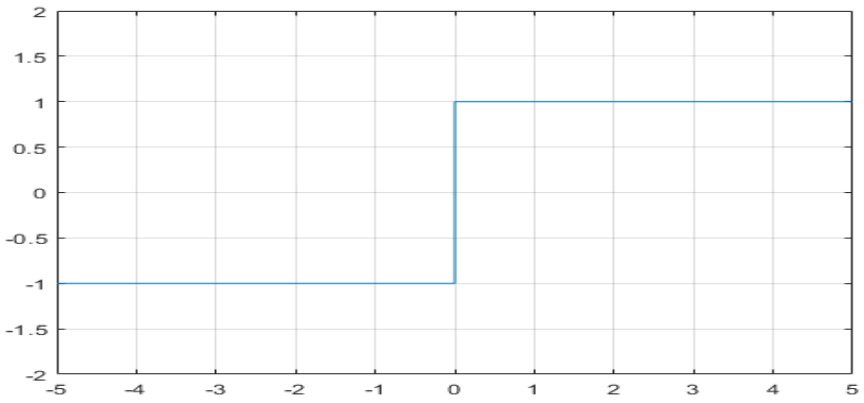
$$a_i \leftarrow g(in_i) = g\left(\sum_j w_{j,i} a_j\right) \tag{5.2}$$

Arhitekture neuronskih mreža (*strukture* neuronskih mreža) zavise od broja slojeva i od broja neurona po sloju. Arhitektura je gotovo uvek ista za neurone, ali se često menja *model* neurona. Model neurona zavisi od aktivacione funkcije *g.* Česte aktivacione funkcije su *Hevisajdova* (*Oliver Heaviside*) ili *Step* funkcija (slika 5.1), funckija *Znaka* (eng. *Sign function)* (slika 5.2) i funkcija *Sigmoid* (slika 5.3), videti [13]. Step funkcija je od ove tri najbliža ljudskom neuronu, upravo jer sadrži granicu koja označava koliko neuron treba da se 'uzbudi' da bi se aktivirao – propustio vrednost 1. Dakle, ova granica je minimalna potrebna suma proizvoda težina sa odgovarajućim ulazima kako bi neuron prosledio vrednost 1. Često se ova funkcija dopunjuje sa dodatnim parametrom koji predstavlja težinu aktivacije. Na ovaj način se neuronska mreža mnogo jednostavnije uči jer više nije potrebno brinuti se o graničnoj vrednosti *t* aktivacione funkcije već samo o težinama neurona, jer ubacujemo dodatni ulazni parametar a_0 koji ima konstantnu vrednost -1. Ovo je dato jednačinom (5.3), gde dodatna težina w_0 koja je vezana za ovu konstantnu vrednost a_0 igra ulogu granične vrednosti *t,* pod uslovom da je $w_{0,i}a_0$=-*t* ([13]).

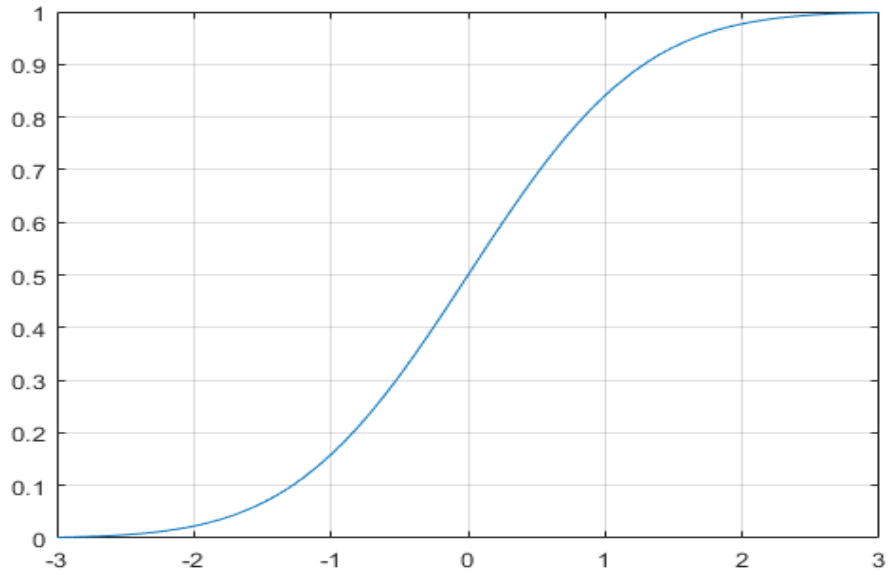
$$a_i = step_i\left(\sum_{j=1}^{n} w_{j,i} a_j\right) = step_0\left(\sum_{j=0}^{n} w_{j,i} a_j\right) \tag{5.3}$$



SLIKA 5.1. HEVISAJDOVA *STEP* FUNKCIJA



SLIKA 5.2. *SIGN* FUNKCIJA



SLIKA 5.3.*SIGMOID* FUNKCIJA

Primer 5.1.[13] Step, sign i sigmoid funkcije:

$$step_i(x) = \begin{cases} 1 \; za \; x \geq t \\ 0 \; za \; x < t \end{cases}$$

$$sign_i(x) = \begin{cases} 1 \; za \; x > 0 \\ -1 \; za \; x < 0 \end{cases} \tag{5.4}$$

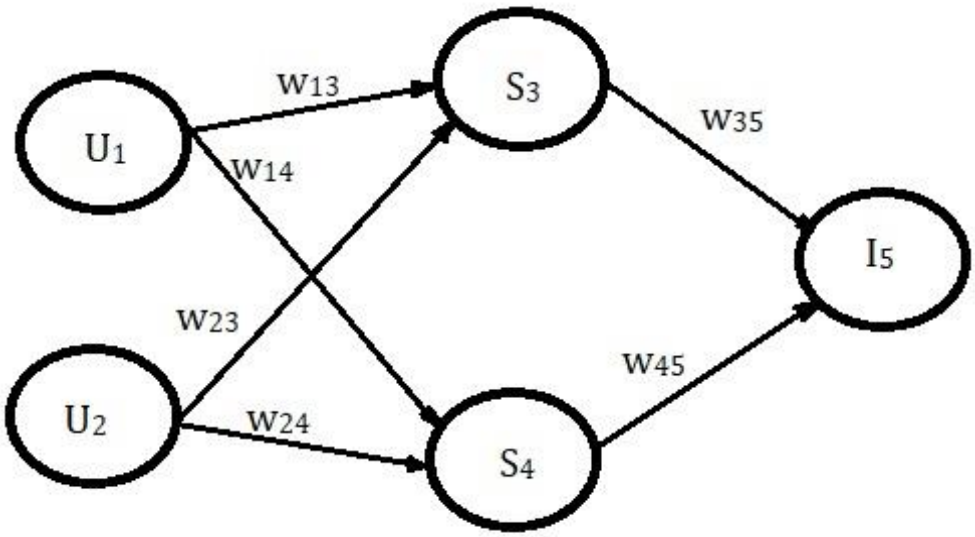
$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

5.2. STRUKTURE NEURONSKIH MREŽA

Imajući u vidu da neuronske mreže nemaju fiksirane aktivacione funkcije i nemaju konstantan broj neurona ili slojeva, lako je zaključiti da postoji širok spektar mogućnosti konstruisanja neuronskih mreža gde svaki način služi za rešavanje nekog određenog skupa problema.

Glavna podela neuronskih mreža je po tome da li neuroni interno čuvaju svoje proračune, odnosno svoja stanja ili ne. U zavisnosti od čuvanja stanja, neuronske mreže se dele na jednosmerne (eng. *feed-forward)* neuronske mreže i na rekurzivne (eng. *recurrent*) ili kružne neuronske mreže, videti [13]. U jednosmernim neuronskim mrežama veze između neurona su jednosmerne i nema nikakvih ponavljanja ili preplitanja smerova. U rekurzivnim neuronskim mreža ove veze mogu biti postavljene proizvoljno i mreža se može sastojati od proizvoljnog broja veza između neurona, u bilo kom smeru.

Neuronske mreže su u većini slučajeva poređane u slojeve, odnosno sačinjene su od nivoa ili grupacija neurona. U jednosmernim neuronskim mrežama, svaki neuron je direktno povezan sa svim neuronima u narednom sloju, ne postoje veze koje vode od jednog neurona u smeru ka prethodnom sloju i isto tako ne postoje veze koje ignorišu sledeći sloj i idu ka sloju nakon prvog sledećeg sloja jednog neurona.



SLIKA 5.4 – NEURONSKA MREŽA SA JEDNIM SKRIVENIM SLOJEM

Na slici 5.5 data je predstava jedne dvoslojne jednosmerne neuronske mreže. Ova mreža je dvoslojna jer neuroni na početku mreže (obeleženi slovom *U)* služe kao ulazni neuroni koji prosleđuju ulaze mreže ka sledećem sloju.

Kod jednosmernih neuronskih mreža, prethodni račun na neuronima u mreži ne igra nikakvu ulogu u računanju izlaza u narednoj iteraciji sa drugačijim

ulazima – ne postoji interna predstava stanja, sve informacije koje mreža može da upamti (sva *memorija* mreže) može biti predstavljena samo težinama na vezama neuronske mreže. Ovakva organizacija neuronske mreže omogućava jednostavan računski tok, ali sprečava kompleksno memorisanje rezultata mreže. Jednosmerna neuronska mreža ne može predstavljati realnu sliku ljudskog mozga, jer bi to bilo ekvivalentno osobi koja nema mogućnost kratkoročnog pamćenja, već refleksno reaguje na sve nadražaje iz svoje okoline.

Upravo zbog ovoga je bilo potrebno implementirati rekurzivne neuronske mreže. Ove rekurzivne neuronske mreže su i dalje vrlo eksperimentalne i upravo zbog raznovrnosti veza između neurona mogu izazvati nestabilno ponašanje i često se ponašaju kao stohastički sistem. Pored ovoga, kompleksne veze između slojeva i neurona otežavaju neuronskoj mreži da izvrši proračune od ulaza do izlaza, a samim tim je i učenje mreže otežano. Sa druge strane, kružne neuronske mreže imaju mogućnost memorisanja stanja upravo kroz propagaciju rezultata na prethodne slojeve. Ovo omogućava konstrukciju kompleksnijih inteligentnih agenata, i modelovanje i simulaciju sistema sa pamćenjem stanja.

Najranije i možda najbolje proučene rekurzivne mreže su Bolcmanove mašine (eng. *Boltzmann Machines)* i Hopfildove mreže (eng. *Hopfield Networks*), videti u [13].

Hopfildove mreže imaju simetrične težine i sve veze su bidirekcione, što znači da svi neuroni u mreži igraju ulogu i ulaznih neurona i izlaznih neurona. Hopfildove mreže takođe implementiraju *sign* funkciju kao aktivacionu funkciju na svim neuronima i mapiraju sve izlaze na vrednosti u skupu $\{-1, 1\}$. Često se koriste za prepoznavanje ili generisanje slika. Pretpostavimo da je obučavajući skup sastavljen od slika i da se ulazni podaci sastoje od piksela jednog dela neke slike. Kao rezultat, Hopfildova mreža bi proizvela sliku iz obučavajućeg skupa koja sadrži region koji je najsličniji prosleđenom ulazu u mrežu. Ovo se zove asocijativno učenje, ili asocijativna mreža, jer mreža vraća rezultat koji se najbolje uklapa u nešto što je mreža već imala šanse da vidi, odnosno nauči.

Bolcmanove mašine predstavljaju stohastičke Hopfildove mreže, koje imaju skrivene slojeve neurona. Kao Hopfildove mreže, Bolcmanove mašine imaju mogućnost internog čuvanja stanja. Bolcmanove mašine kao aktivacione funkcije koriste stohastičke funkcije.

Na slici 5.4, neuroni obeleženi slovom *U* predstavljaju ulazne neurone, neuroni obeleženi slovom *I* su izlazni neuroni, a neuroni obeleženi sa slovom *S* su *skriveni* neuroni. Ulazni neuroni ne sadrže aktivacionu funkciju, niti imaju težine na ulazima, jer nemaju ulazne veze. Njihove izlazne vrednosti zavise isključivo od okruženja u kojem se neuronska mreža nalazi – ulazne vrednosti prosleđuje okruženje ili korisnik, laborant. Broj izlaznih vrednosti neuronske mreže je dat brojem neurona na izlaznom sloju neuronske mreže, pošto svaki neuron kao izlaz daje tačno jednu vrednost. Neuroni u skrivenom sloju ne mogu biti direktno pogođeni iz spoljašnjeg sveta, već samo od strane neurona na ulaznom sloju ili od neurona u nekom od prethodnih skrivenih slojeva ako takvi postoje. Ukoliko neuronska mreža ne sadrži skrivene slojeve, ta mreža predstavlja strukturu *perceptrona* (eng. *perceptron)*. Ovi perceptroni imaju mnogo lakši proces učenja pošto nema potrebe korigovati vrednosti težina na vezama koje su spojene za skrivene neurone. Međutim, perceptroni imaju ograničen broj mogućih stvari koje mogu predstaviti na izlazu. Neuronske mreže koje imaju jedan ili više skrivenih slojeva predstavljaju *višeslojne* neuronske mreže. Višeslojna neuronska mreža na slici 5.4 se može predstaviti sledećom jednačinom ([13]):

$$\begin{aligned} a_5 &= g\left(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4\right) \\ &= g\left(w_{3.5} g\left(w_{1,3} a_1 + w_{2,3} a_2\right) + w_{4,5} g\left(w_{1,4} a_1 + w_{2,4} a_2\right)\right) \end{aligned} \tag{5.5}$$

pod pretpostavkom da je aktivaciona funkcija g za svaki neuron ista, a a_i predstavlja izlaznu vrednost na neuronu i. S obzirom da su težine neuronske mreže parametri koji se koriguju u procesu obučavanja neuronske mreže, obučavanje mreže predstavlja proces nelinearne regresije.

Neuronska mreža na slici 5.4 je dobra za određene probleme, ali je potrebno korigovati njenu strukturu neurona i slojeva, pa čak i aktivacionih funkcija, kako bi mogli drugi problemi da se rešavaju sa njom, videti [13]. Ako konstruišemo mnogo manju neuronsku mrežu nego što je potrebno za određeni problem, možemo doći do problema da naša mreža uopšte nema mogućnost da prikaže funkciju koju pokušavamo da pronađemo (*npr.* nemogućnost prikazivanja nelinearne funkcije zbog nedostatka parametara). Sa druge strane, neuronska mreža može biti toliko velika, da u jednom trenutku prestaje da generalizuje slučajeve, odnosno, za slučajeve koje nije učila neće moći da formira nov izlaz koji bi bio tačan već bi kreirala neki postojeći izlaz koji je prethodno naučila iz obučavajućeg skupa. Ova dva

problema, respektivno, predstavljaju *underfitting* i *overfitting* probleme, videti [13].

5.2.1. PERCEPTRON

Kao što je već rečeno, neuronska mreža koja se sastoji samo od sloja ulaznih neurona i sloja sa jednim izlaznim neuronom naziva se perceptron. Perceptron je jednosmerna neuronska mreža koja je prvi put izučavana 1950-ih godina pošto je predstavljala jedinu strukturu neuronske mreže koja je proizvodila najbolje uspehe što se tiče obuke neuronske mreže. Perceptron se često koristi kako bi se sastavila perceptronska mreža, što je ustvari neuronska mreža koja se sastoji od jednog ili više ulaznih neurona i dva ili više izlaznih neurona, bez skrivenih slojeva. U ovakvoj topologiji neuronske mreže, svaki izlazni neuron je nezavisan od ostalih, i svaka težina na vezama ima uticaj na tačno jedan izlaz, što naravno takođe važi i za perceptron sa jednim neuronom u izlaznom sloju neurona ([13]):

$$O = Step_0\left(\sum_j w_j I_j\right) = Step_0(w \cdot I) \tag{5.6}$$

U formuli 5.6, O predstavlja izlaznu vrednost perceptrona, a w_j predstavlja težinu na vezi od ulaznog neurona j sa ulaznom vrednošću I_j do izlaza O.

Jednostavna struktura perceptrona ograničava njegove mogućnosti toliko da je u mogućnosti da reši jedino one probleme koji su linearno separabilni, videti [7], [13]. Ovo se može predstaviti pronalaženjem težina za rešavanje iskaza *disjunkcije* (*ili*), *konjunkcije* (*i*), *negacije* i *alternacije (ekskluzivno ili*). Pretpostavimo da imamo perceptron sa dva ulazna neurona i jednim izlaznim neuronom i aktivacionom funkcijom *step* na izlaznom neuronu(jedan ulazni neuron za slučaj negacije). Oba ulaza neurona prosleđuju istinitosne vrednosti izlaznom neuronu koji određuje konačnu vrednost iskaza, tako da logički tačno ima vrednost 1, a logički netačno ima vrednost 0. Pod ovakvom

pretpostavkom, konfiguracija ovog perceptrona, to jest težine na vezama perceptrona i granična vrednost aktivacione funkcije mogu biti sledeće ([13]):

- Za disjunkciju, vrednosti na vezama mogu biti $w_1 = 1, i\, w_2 = 1$, a granična vrednost $t = 1.5$
- Za konjunkciju, vrednosti na vezama mogu biti $w_1 = 1, i\, w_2 = 1$, a granična vrednost $t = 0.5$
- Za negaciju, vrednost na vezi može biti $w_1 = 1$, a granična vrednost $t = -0{,}5$.

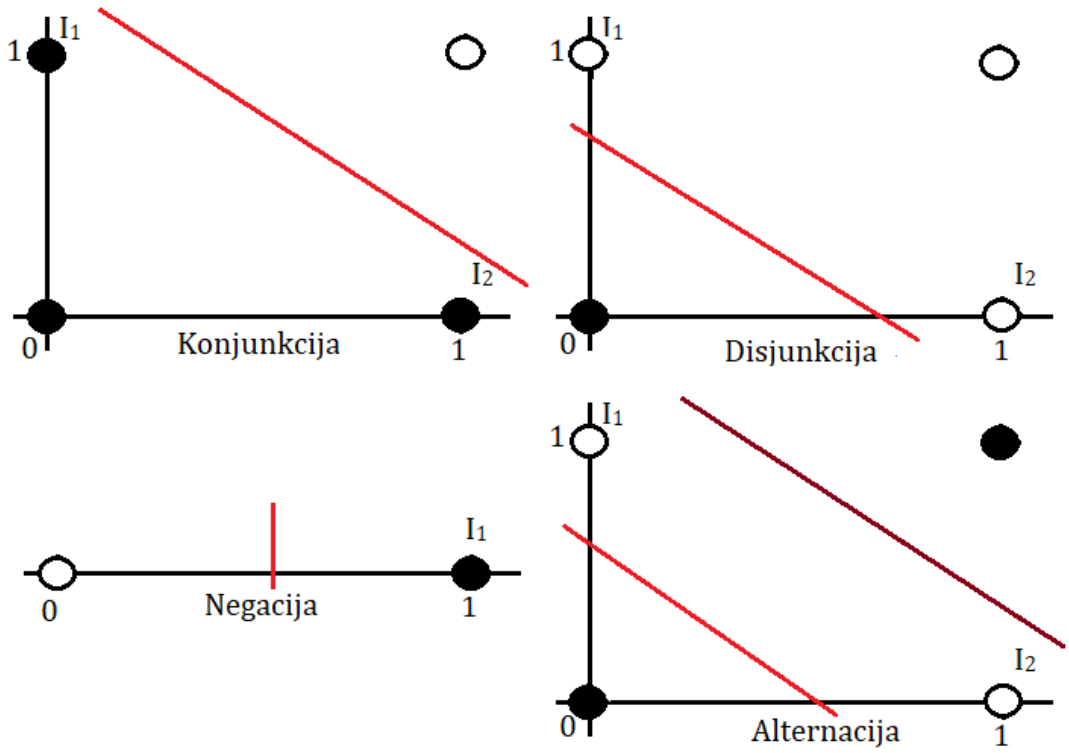
Možemo proslediti bilo koju od 4 kombinacije ulaza za proveru rada ovog perceptrona koristeći:

$$f(U_1, U_2) = U_1 \cdot w_1 + U_2 \cdot w_2,$$
$$i\ \ O(f) = \begin{cases} 1\ za\ f(U_1, U_2)\ \geq t \\ 0\ za\ f(U_1, U_2) < t \end{cases} \tag{5.7}$$

odnosno u slučaju negacije:

$$f(U_1) = U_1 \cdot W_1\,, \qquad i\ \ O(f) = \begin{cases} 1\ za\ f(U_1)\ \geq t \\ 0\ za\ f(U_1) < t \end{cases} \tag{5.8}$$

Problem nastaje kada je potrebno rešiti alternaciju koristeći jedan perceptron. Ovaj problem predstavljen je grafički slikom 5.5.



SLIKA 5.5. GRAFICI ZA DISJUNKCIJU, KONJUNKCIJU, NEGACIJU I ALTERNACIJU

Na slici 5.5 [13], vidi se da za disjunkciju, dobijamo vrednosti 1 (tačno – *true,* obeleženo belim kružićem) za sve kombinacije osim kombinacije koja ima 0 na oba ulaza. Kod konjunkcije, dobijamo vrednosti 0 (netačno – *false,* obeleženo crnim kružićem) za sve kombinacije osim kombinacije koja ima 1 na oba ulaza. Alternaciju je potrebno razdvojiti sa dve linije, što se ne može uraditi uz pomoć perceptrona, čija je funkcija data funkcijom 5.6.

Činjenica da perceptron može rešiti samo linearno separabilne probleme znači da ne možemo da se oslanjamo na perceptrone kada je potrebno rešavati praktične probleme, iz prostog razloga što nema puno linearno separabilnih problema u praksi, videti [7], [13]. Naravno, kada je problem linearno separabilan, može se koristiti perceptron. Dobra karakteristika perceptrona je što postoji algoritam koji omogućava da se perceptron koriguje tako da može da reši bilo kakav linearno separabilan problem. Kako bi ovaj algoritam bio uspešno implementiran, potreban je dovoljno veliki obučavajući skup.

Perceptron sa n ulaza incijalno ima nasumične vrednosti za težine na svim vezama, videti [13]. Ove vrednosti se obično uzimaju na intervalu $[-1, 1]$. Svaki ulazni skup podataka ima skup ulaznih vrednosti i očekivanu izlaznu vrednost.

Sa ovim informacijama perceptron pokušava da svoje težine koriguje tako da što više ovih očekivanih izlaznih vrednosti 'pogodi', odnosno, da dobije vrednost na izlazu onakvu kakvu je ulazni skup prosledio. Kako bi perceptron konvergirao ka ovim tačnim vrednostima i optimalnom skupu težina za te očekivane vrednosti, potrebno je više puta ponoviti prosleđivanje ulaznih podataka perceptronu. Skup podataka za obuku se često deli na pravi obučavajući skup i skup za validaciju. Pravi obučavajući skup služi za korigovanje vrednosti na težinama, a skup za validaciju služi kako bi se proverila tačnost perceptrona i često se obučavajući skup deli tako da pravi obučavajući skup sadrži 69% podataka, a skup za validaciju 31% podataka. Jedno korišćenje čitavog pravog obučavajućeg skupa i skupa za validaciju naziva se *epoha* i trening perceptrona se često sastoji iz više epoha. Konkretno je za perceptron pravilo za korigovanje težina prilično jednostavno ([13]):

$$E = T - I \tag{5.9}$$

gde E predstavlja ukupnu grešku koju je perceptron napravio proizvodeći rezultat I za traženi rezultat T. Ukoliko za E dobijemo vrednost 0, to znači da je perceptron doneo korektnu pretpostavku. Ako je rezultat negativan, znači da je potrebno smanjiti rezultat I, a ako je rezultat pozitivan, znači da je potrebno povećati I kako bi konačna greška bila 0. Imajući ovo u vidu, lako je zaključiti da povećanje u težinama na vezama koje su spojene sa ulazima koji daju pozitivne vrednosti će povećati celokupni rezultat perceptrona, a povećanje na vezama koje su spojene sa ulazima koji proizvode negativne vrednosti znači da će konačni rezultat biti manji. Sa ovim se može napisati sledeća jednačina:

$$w \leftarrow w_j + \alpha \cdot U_j \cdot E \tag{5.10}$$

gde w_j predstavlja težinu na vezi j, U_j je vrednost sa ulaznog neurona j, E je celokupna greška koju je perceptron napravio, a α predstavlja konstantu koja diktira u kolikoj meri će se korigovati težina perceptrona. Ova funkcija se koristi za korigovanje svake od n težina perceptrona, u toku obučavanja neuronske mreže. Prvi put kada je dokazano da ova procedura može korigovati težine perceptrona tako da on ima mogućnost rešavanja bilo koje linearno separabilne funkcije, naučnicima je bilo neverovatno da ovako jednostavna procedura može toliko doprineti polju veštačke inteligencije. Ovo je trajalo do 1969. godine kada su Marvin Minski (*Marvin Minsky)* i Sejmor Papert (*Seymour Papert*) u svojoj knjizi *Perceptrons* demonstrirali ograničenja perceptrona.

5.2.2. VIŠESLOJNE JEDNOSMERNE NEURONSKE MREŽE

Perceptroni, iako ograničeni na linearno separabilne probleme, su mnogo doprineli razvoju veštačke inteligencije kao nauke. Međutim, upravo to ograničenje je sprečavalo razvoj perceptrona. Bilo je potrebno preći na neki drugačiji model veštačkog mozga koji koristi neurone. Neuronske mreže koje imaju više slojeva između ulaznog sloja neurona i izlaznog sloja neurona postojale su kao ideja još kasnih 1950-ih godina (videti [13]), ali nikada nisu zaživele kao i algoritmi klasifikacija ili perceptroni. Veliki problem višeslojnih neuronskih mreža bio je implementacija najbitnijeg algoritma neuronske mreže: obuka neuronske mreže.

Glavni problem obuke višeslojne neuronske mreže je bio određivanje uticaja neurona u međuslojevima na konačni rezultat. Potreban je bio algoritam za računanje koliko svaki neuron u svakom sloju deluje na neurone u sledećim slojevima, kako bi mogle, na osnovu greške izlaznog sloja, da se koriguju njihove težine.

Najpopularnije rešenje problema učenje neuronske mreže je bio algoritam propagacije greške unazad kroz slojeve neuronske mreže, ili, *back-propagation,* videti [5], [13]. Ovaj algoritam je prvi put osmišljen 1969. godine, a prvi put je ozbiljno uzet u razmatranje kao rešenje problema obuke neuronske mreže tek 1980-ih godina. Zbog kompleksnosti algoritma, računarski sistemi početkom 1970-ih godina nisu bili dovoljno napredni za implementaciju backpropagation algoritma.

Učenje neuronske mreže koristeći backpropagation algoritam je proces sličan procesu učenja perceptrona, videti [13]. Na isti način se započinje učenje, neuroni na ulaznom sloju prosleđuju vrednosti iz okruženja neuronske mreže na neurone u skrivenim slojevima. Vrednosti se transformišu u skladu sa težinama na odgovarajućim vezama. Suštinski se za svaki neuron u skrivenom sloju vrši ista kalkulacija kao što bi se vršila za jedan perceptron: ulazne vrednosti u neuron se množe sa težinama na vezama, rezultati se sumiraju i suma se propušta kroz aktivacionu funkciju neurona. Ova aktivaciona vrednost predstavlja ulaz na sledeći sloj neurona. Ovo se vrši za svaki sloj, dok se ne proizvede vektor vrednosti na izlaznom sloju. Kada dobijemo izlaz neuronske mreže, gleda se koliko je neuronska mreža pogrešila. Ukoliko je vektor

dobijenih vrednosti neuronske mreže jednak očekivanim rezultatima, ništa se u neuronskoj mreži neće promeniti. Međutim, ukoliko je došlo do greške, potrebno je korigovati vrednosti težina neurona. Postavlja se par pitanja: kojih težina, kojih neurona, kolika je ta greška i za koji sloj? Ovaj skup pitanja se može odgovoriti deljenjem greške na učesnike u proizvodnji te greške, odnosno, greška se koristi kako bi se korigovale vrednosti na izlaznim neuronima kao da korigujemo perceptrone, a propagacija greške do neurona u prethodnim slojevima podrazumeva deljenje ove greške na sve neurone koji su doprineli grešci. Ovo je za perceptrone lako izvršiti jer imamo jedan neuron (sam perceptron), a veze koje su povezane sa njim su direktno veze iz prethodnog, ulaznog sloja. U višeslojnim neuronskim mrežama ova propagacija unazad se može definisati sledećom formulom ([13]):

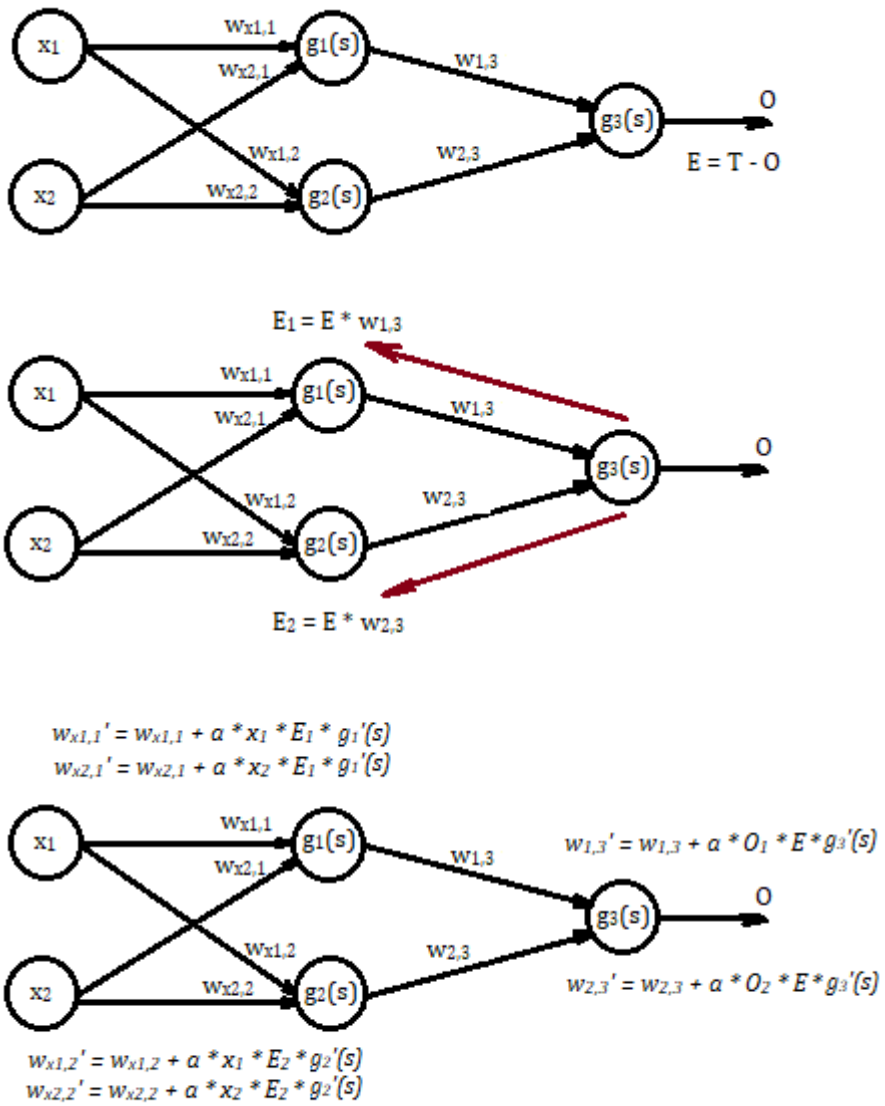
$$w_{j,i} \leftarrow w_{j,i} + \alpha \cdot a_j \cdot E_i \cdot g'(U_i) \tag{5.11}$$

E_i je greška na izlaznom neuronu i izražena preko T_i-O_i, a $g'(U_i)$ rezultat koji proizvodi izvod aktivacione funkcije neurona, za prosleđeni ulaz U_i na neuronu.

Za ažuriranje težina na vezama od ulaznog sloja neurona pa sve do izlaznog sloja neurona potrebno je definisati koliko su skriveni slojevi neuronske mreže doprineli ukupnoj grešci neuronske mreže. Ukupna greška jednog neurona mora se podeliti po težinama koje su povezane za njega i tako propagirati unatrag kroz mrežu. Propagacija greške se može zapisati na sledeći način ([13]):

$$\Delta_j = g'(U_j) \sum_j w_{j,i} \cdot E_i \cdot g'(U_i) \tag{5.12}$$

Ovaj algoritam se svodi na incijalno računanje greške za izlazne neurone. Ove greške se prosleđuju u prethodne slojeve tako što svaka greška E koja mora da se prosledi na neuron J vezom I se najpre množi sa težinom na vezi I. Neuron J će sumirati sve proizvode grešaka sa težinama i vršiti dalje kalkulacije za nove propagacije grešaka. Tek nakon što se greška prosledi od jednog neurona do drugog, može se izvršiti korigovanje težina. Na slici 5.6 je prikazan ovaj algoritam na jednostavnoj neuronskoj mreži sa jednim skrivenim slojem.



SLIKA 5.6. *BACKPROPAGATION* ALGORITAM

Na slici 5.6 vidi se da se u toku korigovanja težina koristi ulazna vrednost neurona. Prilikom implementacije neuronske mreže, dobra je ideja ove vrednosti čuvati na svakom neuronu prilikom proračuna predikcije i izlaza *O*, kako bi posle bilo brže izvesti backpropagation algoritam.

Backpropagation algoritam može se interpretirati pomoću metoda opadajućeg gradijenta (eng. *Gradient descent*, videti [13]). U slučaju opadajućeg gradijenta, gradijent se posmatra na površini ukupne greške – površina koja opisuje

greške za svaki podatak koji se može propustiti kroz funkciju koja je sastavljena od težina.

Čitava neuronska mreža je jedna funkcija koja zavisi od težina na vezama između neurona. Za određene vrednosti ovih težina u jednom momentu, data nam je jedna tačka na prostoru grešaka. Na toj tački posmatra se nagib površine koju formira svaka težina – parcijalni izvod površine u odnosu na svaku težinu. Dakle, gleda se koliko se ova površina menja ukoliko se za mali broj promeni težina. Promenama u težinama, težimo da pomerimo tačku koju proizvodi neuronska mreža ka što dubljem delu površine. Pomoću ovog algoritma, višeslojne veštačke neuronske mreže postaju sve bliže imitaciji ljudskog mozga, omogućavajući paralelizaciju obrade podataka i, ovim algoritmom, paralelno korigovanje težina.

Za backpropagation algoritam potreban nam je način na koji ćemo računati grešku koju je neuronska mreža napravila na izlaznom sloju. Za proračun greške E koristi se suma kvadriranih vrednosti na izlaznim neuronima, zbog toga što se vrlo jednostavno izračunava izvod ove funkcije ([13]):

$$E = \frac{1}{2}\sum_{i}(T_i - I_i)^2 \tag{5.13}$$

gde I_i predstavlja izlaznu vrednost i-tog neurona, a T_i predstavlja tačnu, ili očekivanu vrednost i-tog neurona.

Za slučaj perceptrona, I_i bi glasilo:

$$E = \frac{1}{2}\sum_{i}\left(T_i - g\left(\sum_{j} w_{j,i}a_j\right)\right)^2 \tag{5.14}$$

Gde je $w_{j,i}$ vrednost težine na vezi od neurona j do neurona i, a a_j predstavlja izlaznu vrednost neurona j. Za slučaj dvoslojne neuronske mreže imali bi:

$$E = \frac{1}{2}\sum_i \left(T_i - g\left(\sum_j w_{j,i} g\left(\sum_k w_{j,k} a_k \right) \right) \right)^2 \tag{5.15}$$

Dalja proširivanja na slojeve podrazumeva zapis ulaznih vrednosti u obliku sume proizvoda ulaznih vrednosti i težina na vezama od ulaznih vrednosti do neurona. a_x ne zavisi od težina narednog sloja. Konkretno se na formuli (5.15) može videti da svaki ulaz *i,j* zavisi od tačno jedne težine $w_{i,j}$ što znači da ukoliko diferenciramo E, po nekoj od težina, ostali proizvodi će se posmatrati kao konstante i moći će se skratiti.

Ukoliko bi (5.14) diferencirali po težini $w_{i,j}$ dobili bi ([13]):

$$\begin{aligned} \frac{dE}{dw_{j,i}} &= -a_j(T_i - I_i) g'\left(\sum_j w_{j,i} a_j \right) \\ &= -a_j(T_i - I_i) g'(U_i) \end{aligned} \tag{5.16}$$

gde je U_i ulazna vrednost aktivacione funkcije *i*-tog neurona. Pošto je cilj da se minimizuje ukupna greška *E*, potrebno je sa manjim vrednostima korigovati težine i to u suprotnom smeru kretanja gradijenta. Potrebno je uzeti u obzir aktivacionu funkciju *g*, pošto se u diferenciranju *E* traži i izvod od *g*. S tim u vidu, ne smemo uzeti funkciju kao što je Hevisajdov step signal kao aktivacionu funkciju neuronske mreže. Često se za aktivacionu funkciju uzima sigmoidna funkcija .

Višeslojne neuronske mreže su dobre za rešavanje kontinualnih problema, ali nisu dovoljno dobro proučene, videti [13]. U praksi se dosta problema može rešiti sa relativno malo težina, ali je teško proceniti strukturu, odnosno topologiju mreže, jer i dalje ne postoji jasan način na koji se topologija mreže određuje. Učenje mreže predstavlja potencijalno najsporiji deo neuronske mreže, jer brzina učenja zavisi od broja ulaznih obučavajućih podataka. Ovaj parametar efektivnosti neuronskih mreža se može poboljšati boljim hardverskim komponentama, kao i implementacijom paralelnih backpropagation algoritama.

Za razliku od klasifikacije pomoću algoritma k-najbližih suseda, višeslojne veštačke neuronske mreže imaju relativno dobru toleranciju na smetnje u obučavajućem skupu. Veći problem je što iako neuronska mreža dobro

pretpostavi klasifikaciju, ili dobro izračuna tačku u raspodeli ili nizu, čak i oni koji su dizajnirali topologiju mreže neće znati zašto je taj rezultat *ispao* dobar. Neuronske mreže svoje odluke baziraju na osnovu težina (brojeva), a neki algoritmi koriste stabla sa događajima i kriterijumima.

U svetu veštačke inteligencije se konstantno radi na rešavanju ovih problema, ili makar na njihovom ublažavanju. Neuronske mreže će i pored ovih problema uvek igrati ključnu ulogu u razvoju veštačke inteligencije u svetu, videti [13].

6. OPIS SPECIFIKACIJE I IMPLEMENTACIJA

U ovoj glavi dat je pregled komandi i procedura napisanih u programskom paketu MATLAB, za rešavanje zadatka klasifikovanja tačaka u trodimenzionalnom prostoru. Kroz ovaj zadatak proveravamo kako rade prethodno opisane metode: „naivni“ Bejz, klasifikator k-najbližih suseda i veštačke neuronske mreže. U tu svrhu su predstavljene i originalne skripte napisane upravo za potrebe ovog rada. Kako je za potrebe istraživanja teško doći do stvarnih istorijskih podataka, za ilustraciju su korišćeni automatski generisani podaci .

Klasifikacija zahteva obilan skup podataka koji se može koristiti kao referenca za model koji će vršiti samu klasifikaciju ili kao obučavajući skup podataka za treniranje modela. Iz tog razloga, skup tačaka koji se koristi za obučavanje neuronske mreže, i kao referenca za klasifikator k-najbližih suseda i „naivni“ Bejz, je generisan od strane programa. Generisanje ovog skupa podataka može se kontrolisati unutar programa, povećavajući ili smanjujući obučavajući skup tačaka.

Implementacija sadrži dva skupa tačaka, od kojih jedan skup služi za obuku neuronske mreže koja će vršiti klasifikaciju i kao referenca za druge klasifikatore. Drugi skup tačaka predstavlja skup nad kojim je potrebno izvršiti klasifikaciju tačaka.

MATLAB (laboratorija za matrice –*Matrix Laboratory)* je odabran iz razloga što sadrži sve potrebne alate za implementaciju projekata vezanih za računarsku matematiku. Matlab se smatra i kao programski jezik visokog nivoa koji je odličan za vizuelizaciju podataka i rezultata, kao i za razvoj aplikacija. Takođe sadrži i okruženje namenjeno za lakšu manipulaciju podataka, kao i za bolje snalaženje po izvornim kodovima i strukturi podataka od strane korisnika. MATLAB se koristi u gotovo bilo kojoj sferi računarske matematike, od kojih je najčešće upotrebljen za linearnu algebru, statistiku, obradu i analizu podataka, numeričke proračune, transformacije, fitovanje, algebarske jednačine i nejednačine i slično.

Konkretan zadatak je implementiran koristeći takozvane *nativne* funkcije MATLAB-a, odnosno, funkcije koje ne pripadaju stranim proizvođačima, već one koje bilo koji korisnik može koristiti po instalaciji programskog paketa MATLAB. Postoje razne biblioteke funkcija koje služe kako za rešavanje matematičkih problema tako i za rešavanje problema veštačke inteligencije, ali implementacija ovog konkretnog zadatka koristeći gotove implementacije od drugih proizvođača nije od velikog značaja, jer je ključni izvorni kod skriven od korisnika.

6.1. OBELEŽAVANJE PODATAKA

Za klasifikaciju se unutar programskog rešenja retko koriste prava obeležja klasa. Ovo znači da se unutar programskog koda neće koristiti, na primer, obeležje *CRVENA* za klasifikovanje tačaka koje ispunjavaju uslove da budu klasifikovane u klasu *CRVENIH* tačaka, već će se koristiti brojčana obeležja, na primer, broj 6 za klasu *CRVENIH* tačaka. S obzirom da se ove transformacije iz obeležja (labela) u brojeve vrši na dosta mesta u programu, potrebno je najpre navesti metode unutar direktorijuma *OBRADA_OBELEŽJA*.

Metoda `desifruj_klasu_boja` služi za preuzimanje tekstualnog oblika klasifikacije, na osnovu brojčanog oblika klasifikacije. Mapiranja broj – tekst nisu urađena po nekom određenom redosledu, niti je bilo potrebe da se definiše redosled za ova mapiranja. Jedino što je potrebno je kada se jednom definiše mapiranje broj – tekst, da se ovo mapiranje ponovi za sva drugačija mapiranja klasa, na primer, broj – koordinate. Funkcija

`desifruj_klasu_boja` prolazi kroz sve moguće brojeve koji su uzeti za klasifikacije i povratnoj vrednosti `rgb` dodeljuje tekstualne vrednosti na osnovu promenljive `broj`. Tako, u slučaju da promenljiva `broj` ima vrednost 3, promenljiva `rgb` imaće vrednost 'Zelena', jer je za brojčanu klasu 3 određenja klasifikacija *ZELENIH* tačaka.

```
function [ rgb ] = desifruj_klasu_boja( broj)
if broj == 1
rgb = 'Plava';
elseif broj == 2
rgb = 'Narandzasta';
elseif broj == 3
rgb = 'Zelena';
elseif broj == 4
rgb = 'Roza';
elseif broj == 5
rgb = 'Crvena';
elseif broj == 6
rgb = 'Zuta';
elseif broj == 7
rgb = 'Ljubicasta';
elseif broj == 8
rgb = 'Braon';
elseif broj == 9
rgb = 'Svetlo Plava';
end
end
```

IZLISTAK 6.1. IZVORNI KOD FUNKCIJE `DESIFRUJ_KLASU_BOJA`.

Funkcija `desifruj_klasu_rgb` služi pretežno za crtanje 3D prikaza tačaka u prostoru, pre klasifikacije i nakon klasifikacije. Ova metoda funkcioniše po istom principu kao i funkcija za preuzimanje tekstualnog oblika klasifikacije. Ulazni parametar funkcije je promenljiva `klasa`, brojčana vrednost klase, a na izlazu se preuzima vektor vrednosti `rgb`. Ovaj vektor predstavlja količinu crvene, zelene, i plave nijanse u klasifikaciji (*RGB – Red, Green, Blue)*.

```
function [ rgb ] = desifruj_klasu_rgb( klasa)
if klasa == 1
        rgb = [0, 0, 1];
elseif klasa == 2
        rgb = [1, 0.5, 0];
elseif klasa == 3
        rgb = [0, 1, 0];
```

```
elseif klasa == 4
        rgb = [1, 0.4, 0.6];
elseif klasa == 5
        rgb = [1, 0, 0];
elseif klasa == 6
        rgb = [1, 1, 0];
elseif klasa == 7
        rgb = [0.5, 0, 0.5];
elseif klasa == 8
        rgb = [0.6,0.2,0.1];
elseif klasa == 9
        rgb = [0, 1, 1];
end
end
```

IZLISTAK 6.2. IZVORNI KOD FUNKCIJE `DESIFRUJ_KLASU_RGB`

Funkcija `desifruj_klasu_xyz` služi za preuzimanje opsega koordinata tačaka na osnovu klasifikacije. Unutar ove funkcije definisane su maksimalne i minimalne vrednosti opsega, promenljive `maxx` i `minn` respektivno, kao i promenljiva `offset` koja služi isključivo za lepši grafički prikaz tačaka u prostoru.

```
function [min_max] = desifruj_klasu_xyz(broj)
    maxx = 10;
    minn = -10;
    offset = 1;
if broj == 1
        min_max = [0 + offset, maxx; 0 + offset, maxx; 0 +
offset, maxx];
elseif broj == 2
        min_max = [0 + offset, maxx; 0 + offset, maxx; minn, 0
- offset];
elseif broj == 3
        min_max = [0 + offset, maxx; minn, 0 - offset; 0 +
offset, maxx];
elseif broj == 4
        min_max = [0 + offset, maxx; minn, 0 - offset; minn, 0
- offset];
elseif broj == 5
        min_max = [minn, 0 - offset; minn, 0 - offset; 0 +
offset, maxx];
elseif broj == 6
        min_max = [minn, 0 - offset; minn, 0 - offset; minn, 0
- offset];
elseif broj == 7
        min_max = [minn, 0 - offset; 0 + offset, maxx; minn, 0
- offset];
elseif broj == 8
        min_max = [minn, 0 - offset; 0 + offset, maxx; 0 +
offset, maxx];
elseif broj == 9
        min_max = [minn, maxx; minn, maxx; minn maxx];
```

```
end
end
```

IZLISTAK 6.3. IZVORNI KOD FUNKCIJE `DESIFRUJ_KLASU_XYZ`

Ova funkcija dakle vraća matricu koja predstavlja granice za svaku klasifikaciju. Svaki red u toj matrici je jedna koordinata. Prva kolona matrice predstavlja minimalnu vrednost za koordinatu, a druga kolona maksimalnu. Tako, za slučaj kada je promenljiva `broj` jednaka 8, dobijamo koordinate za tačku koja se nalazi u sledećim granicama:

- Za x: od -10 do -1
- Za y: od 1 do 10
- Za z: od 1 do 10

Lako se vidi da se promenom parametra `offset` menja ukupni prostor između klasifikacija, tako da ako želimo veći razmak između grupa tačaka, potrebno je povećati promenljivu `offset`, a u suprotnom je potrebno smanjiti promenljivu `offset`.

6.2. GENERISANJE PODATAKA

Kao što je već navedeno, kako bi klasifikacija bila uspešna potrebno je posedovati dovoljno opširan skup podataka sa kojima se mogu novi podaci upoređivati, ili kako bi se model veštačke neuronske mreže mogao obučiti za klasifikaciju. Skup podataka za obuku neuronske mreže i za referencu ostalih modela, kao i skup podataka nad kojim će modeli vršiti klasifikaciju kreirani su od strane implementiranih funkcija.

Funkcije koje vrše generisanje podataka za skupove nalaze se u direktorijumu *GENERISANJE_PODATAKA* unutar direktorijuma programskog rešenja.

To su funkcije *generisi_nepoznate_uzorke* , *generisi_skup_podataka* i *generisi_slucajan_broj.*

Generisanje podataka strogo zavisi od funkcije *generisi_slucajan_broj*. Ova metoda uzima dva parametra, *minn* i *maxx* koji služe za definisanje opsega generisanja nasumičnog broja. Funkcija se može iskoristiti na sledeći način:

```
slucajan_broj = generisi_slucajan_broj(-10.5, 125.85);
```

IZLISTAK 6.4. KORIŠĆENJE FUNKCIJE `GENERISI_SLUCAJAN_BROJ`

U ovom slučaju, promenljiva `slucajan_broj` bi imala realnu vrednost koja se nalazi u opsegu od -10.5 i 125.85.

Metoda najpre proverava da li je prosleđeno manje od dva ulazna parametra upoređivajući parametar *nargin* sa 2. Parametar *nargin* je parametar koji postoji u svakoj funkciji koja se definiše u MATLAB-u. Ukoliko je broj prosleđenih parametara manji od 2, znači da nisu definisani parametri za minimalan i maksimalan slučajan broj. U ovom slučaju se za opseg uzimaju brojevi od 0 do 1. Funkcija je tako napisana da se u daljoj implementaciji zadatka na lak način može koristi u slučaju da je potrebno generisati slučajan broj između 0 i 1.

```
function [ res ] = generisi_slucajan_broj(minn, maxx)
if nargin < 2
        minn = 0;
        maxx = 1;
end
```

IZLISTAK 6.5. PROVERA KOLIKO JE PARAMETARA PROSLEĐENO U FUNKCIJU `GENERISI_SLUCAJAN_BROJ`

U ostatku funkcije se koristi MATLAB-ova ugrađena funkcija za generisanje realnog nasumičnog broja između 0 i 1, `rand` kako bi se dobio inicijalni slučajan broj koji će se dalje prebaciti u prosleđeni opseg. Funkciji `rand` prosleđuje se parametar 1 kako bi se dobio rezultat u obliku matrice sa jednim redom i jednom kolonom, odnosno, skalar .Rezultat se čuva unutar promenljive `res` i dalje se koristi formula:

$$y = (max - min) \cdot x + min \tag{6.1}$$

kako bi se parametar `res` prebacio u zadati opseg. U formuli (6.1) x predstavlja `res`,a y se upisuje u istu promenljivu `res`. Na kraju funkcije, koristi se metoda `round` kojoj se prosleđuju parametri `res`i broj 2. Ovo znači da promenljiva `res` treba da se zaokruži na dve decimale. Povratna vrednost

metode `round` čuva se ponovo u promenljivoj `res`, i taj rezultat se vraća kao rezultat izvršavanja funkcije `generisi_slučajan_broj`.

```
    res = rand(1);
    res = (maxx - minn) .* res + minn;
res = round(res, 2);
end
```

IZLISTAK 6.6. OSTATAK FUNKCIJE GENERISI_SLUCAJAN_BROJ

Generisanje uzorka nepoznatih podataka (uzorak podataka koji *treba* da se klasifikuju nakon obuke) implementirano je u funkciji `generisi_nepoznate_uzorke`. Najpre se definiše obim uzorka nepoznatih podataka koji se generiše po klasi uz pomoć promenljive `n_nepoznatih`. Dalje je potrebno generisati koordinate nepoznatih tačaka. Ovo je implementirano kombinacijom metoda za generisanje nasumičnih brojeva i za određivanje minimalnih i maksimalnih granica koordinata za klasifikaciju. Najpre se iskoristi metoda `desifruj_klasu_xyz` kako bi dobili minimalne i maksimalne granice za x,y i z koordinate tačke iz zadate klase `klasa`. Ovim minimalnim i maksimalnim granicama možemo generisati nasumične brojeve pomoću funkcije `generisi_slucajan_broj`. Minimalne i maksimalne vrednosti su sačuvane u promenljivoj `mm_klase` i iz te promenljive uzimamo po redovima granice za koordinate. Ovim postupkom dobijamo promenljivu `red_podataka`. Ova promenljiva sadrži 3 kolone, gde svaka kolona ima vrednost za jednu od svake koordinate- prva kolona ima vrednost za x koordinatu, druga kolona za y i treća za z. Promenljiva `red_podataka` ubacuje se u matricu `nepoznate_klase` na poslednji red, promenljiva `nepoznate_klase` predstavlja matricu koja se proširuje svakim generisanjem nove tačke. Ova procedura ponavlja se `n_nepoznatih` puta, za svaku od 8 klasifikacija.

```
function [ nepoznate_klase ] = generisi_nepoznate_uzorke()
    n_nepoznatih = 10;
    nepoznate_klase = [];
for i = 1 : n_nepoznatih
for klasa = 1 : 8
            mm_klase = desifruj_klasu_xyz(klasa);
            red_podataka = [
                generisi_slucajan_broj(mm_klase(1,          1),
mm_klase(1, 2))
                generisi_slucajan_broj(mm_klase(2,          1),
mm_klase(2, 2))
                generisi_slucajan_broj(mm_klase(3,          1),
mm_klase(3, 2))
            ]';
            nepoznate_klase = [nepoznate_klase; red_podataka];
end
end
```

```
end
```

IZLISTAK 6.7. IZVORNI KOD FUNKCIJE `GENERISI_NEPOZNATE_UZORKE`

Generisanje obučavajućeg skupa podataka je urađeno na sličan način kao i generisanje nepoznatih ulaznih podataka. Generisanje obučavajućeg skupa podataka implementirano je u metodi `generisi_skup_podataka`. Ova metoda uzima dva parametra, minimalan i maksimalan broj podataka po jednoj klasifikaciji. Ukoliko ovi parametri nisu postavljeni, za minimalan broj uzoraka uzeće se broj 250, a za maksimalan broj podataka uzeće se 500. Inicijalan skup podataka dat je promenljivom `skup_podataka` i u tu promenljivu dodavaće se objekti koji predstavljaju skupove obučavajućih podataka po klasifikaciji. Za svaku od 8 klasa se vrši sledeća procedura:

- Pomoću metode `desifruj_klasu_xyz` preuzimamo minimalne i maksimalne vrednosti za x, y i z koordinate,
- Generišemo slučajan broj koji govori koliko podataka treba generisati za datu klasu,
- Kreiramo promenljivu `kord` koja se sastoji od 3 kolone, svaka kolona sadrži vrednost za jednu od x, y i z koordinata,
- Ovu promenljivu dodajemo u poslednji red promenljive `xyz`
- Kreiramo promenljivu `klasa_obj` koja ima polje `xyz` u koje upisujemo skup tačaka `xyz`
- Dodajemo novi objekat `klasa_obj` na kraj promenljive `skup_podataka`

Ova procedura kreira 8 objekata, od kojih svaki objekat ima polje `xyz` koje predstavlja koordinate tačaka za trening. Ovakvom organizacijom podataka lako se može pristupati određenoj tački iz tražene klasifikacije. Unutar ove metode pozvana je i metoda `generisi_nepoznate_uzorke`, tako da ova metoda napravi i podatke za obuku klasifikacionih modela, kao i podatke koje je potrebno klasifikovati.

```
function [ skup_podataka, nepoznati_uzorci ] =
generisi_skup_podataka(min_uzoraka, max_uzoraka)
if nargin < 2
        min_uzoraka = 250;
        max_uzoraka = 500;
end
    skup_podataka = [];
for klasifikacija = 1:8
        minmax_klase = desifruj_klasu_xyz(klasifikacija);
        broj_uzoraka = generisi_slucajan_broj(min_uzoraka,
max_uzoraka);
        xyz = [];
for uzorak = 1:broj_uzoraka
            kord = [generisi_slucajan_broj(minmax_klase(1,1),
minmax_klase(1,2)), generisi_slucajan_broj(minmax_klase(2,1),
```

```
minmax_klase(2,2)),   generisi_slucajan_broj(minmax_klase(3,1),
minmax_klase(3,2))];
            xyz = [xyz; kord];
end
        klasa_obj.xyz = xyz;
        skup_podataka = [skup_podataka, klasa_obj];
end

    nepoznati_uzorci = generisi_nepoznate_uzorke();
end
```

IZLISTAK 6.8. IMPLEMENTACIJA METODE `GENERISI_SKUP_PODATAKA`

6.3. GRAFIČKI PRIKAZ REZULTATA

U svrhu prikaza rezultata metodologija koristi se skup funkcija u direktorijumu *PRIKAZ_PODATAKA.*

Osnovna metoda za prikaz podataka jeste `nacrtaj_tacku`, metoda koja uzima koordinate tačke koja treba da se iscrta, klasifikaciju tačke kako bi se znalo kojom bojom treba obojiti tačku na grafičkom prikazu, i dodatni parametar koji govori da li je prosleđena tačka bila nepoznata, pa je sada klasifikovana ili je poznata od početka simulacije. Koristeći parametar `klasifikacija` i metode `desifruj_klasu_rgb`, dobijamo potrebnu boju ivica tačke kao i same tačke u obliku vektora koji sadrži vrednosti za crvenu, zelenu i plavu nijansu. Ukoliko je tačka bila nepoznata, (parametar `nova_klasa` postavljen na vrednost *true*), onda će ivice tačke dobiti svetlo plavu boju. Veličina tačke za prikaz se postavlja na 25 i koristi se Matlabova ugrađena funkcija `scatter3`, za iscrtavanje tačke u prostoru. Metodi se prosleđuju koordinate tačke, a zatim se prosleđuju parametri za dekoraciju tačke.

```
function   []   =   nacrtaj_tacku(koordinate,   klasifikacija,
nova_klasa)
    boja = desifruj_klasu_rgb(klasifikacija);
    boja_ivica = desifruj_klasu_rgb(klasifikacija);
if nova_klasa
        boja_ivica = [0.0, 1.0, 1.0];
end
    velicina_tacaka = 25;
    scatter3(koordinate(1), koordinate(2), koordinate(3), ...
        velicina_tacaka, ...
'MarkerEdgeColor', boja_ivica, ...
'MarkerFaceColor', boja);
end
```

IZLISTAK 6.9. IMPLEMENTACIJA METODE NACRTAJ_TACKU

Metoda `nacrtaj_zonu` služi za iscrtavanje prostora u kojem se nalaze podaci za trening i za klasifikaciju, tako da se mogu bolje uočiti rezultati simulacije. Funkcija uzima jedan parametar, a to je niz karaktera koji će biti iscrtan iznad prostora kao naslov grafika. Ukoliko nema prosleđenih parametara, naslov neće biti iscrtan. U funkciji su definisane konstantne vrednosti za granice prostora, promenljivama `min_ravan` i `max_ravan`. Prostor se nalazi između vrednosti -10 i 10 po svim osama. Crtanje ravni vrši se definisanjem vrednosti za četiri tačke koje čine ravan. Na primer, ravan koja pokriva z osu, definisana je sa tri vektora:

1. Vektor `z_ravan_x_koord` koji ima vrednosti -10, 10, 10 i -10
2. Vektor `z_ravan_y_koord` koji ima vrednosti -10, -10, 10, 10 i
3. Vektor `z_ravan_z_koord` koji ima vrednosti 0, 0, 0, 0

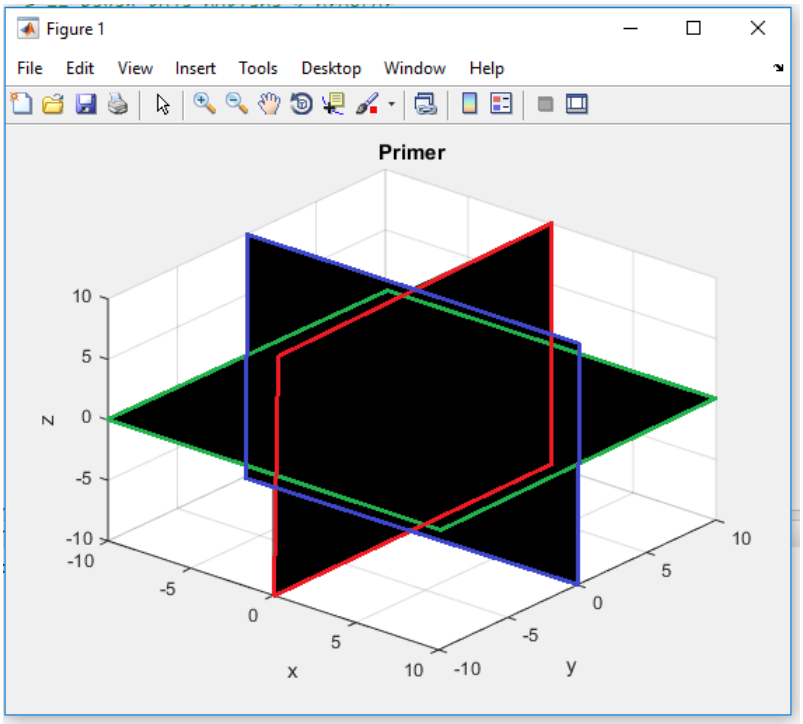
Na ovaj način, ravan će biti konstruisana od tačaka (-10, -10, 0), (10, -10, 0), (10, 10, 0) i (-10, 10, 0) za koordinate (*x, y, z*). Za ravan je definisana crna boja i vektori, kao i promenljiva `color` za boju se prosleđuju Matlabovoj ugrađenoj funkciji `patch` koja vrši konkretno crtanje ravni na grafiku. Na kraju crtanja grafika, beleže se ose koristeći metode `xlabel`, `ylabel` i `zlabel`. Pogled na grafik se horizonalno rotira za 40 stepeni suprotno od smera kretanja kazaljki na satu, i pogled se podiže za 35 stepeni od koordinatnog početka.

```
function [] = nacrtaj_zonu(naslov)
if nargin == 0
        naslov = '';
end
    min_ravan = -10;
max_ravan =  10;
    z_ravan_x_koord   =   [min_ravan,   max_ravan,   max_ravan,
min_ravan];
    z_ravan_y_koord   =   [min_ravan,   min_ravan,   max_ravan,
max_ravan];
    z_ravan_z_koord = [0, 0, 0, 0];
    y_ravan_x_koord   =   [min_ravan,   max_ravan,   max_ravan,
min_ravan];
    y_ravan_y_koord = [0, 0, 0, 0];
    y_ravan_z_koord   =   [min_ravan,   min_ravan,   max_ravan,
max_ravan];
    x_ravan_x_koord = [0, 0, 0, 0];
    x_ravan_y_koord   =   [min_ravan,   max_ravan,   max_ravan,
min_ravan];
    x_ravan_z_koord   =   [min_ravan,   min_ravan,   max_ravan,
max_ravan];
color = 'black';
    patch(z_ravan_x_koord,   z_ravan_y_koord,   z_ravan_z_koord,
color);
    patch(x_ravan_x_koord,   x_ravan_y_koord,   x_ravan_z_koord,
color);
```

```
patch(y_ravan_x_koord,     y_ravan_y_koord,     y_ravan_z_koord,
color);

    xlabel('x'); ylabel('y'); zlabel('z');
grid; view(40, 35); title(naslov);
end
```

IZLISTAK 6.10. IZVORNI KOD FUNKCIJE NACRTAJ_ZONU



SLIKA 6.1. VIZUELNI PRIKAZ REZULTATA POZIVA FUNKCIJE NACRTAJ_ZONU (CRVENI, PLAVI I ZELENI OKVIRI DODATI RADI JASNIJEG PRIKAZA).

6.4. IMPLEMENTACIJA METODOLOGIJA KLASIFIKACIJA

Direktorijum *METODOLOGIJE* sadrži sve funkcije i definicije klasa potrebnih za izvršavanje klasifikacija ulaznih podataka tačaka. Metodologija k-najbližih suseda i metodologija klasifikatora „naivni" Bejz se nalaze u zasebnim datotekama, a sve funkcionalnosti vezane za neuronske mreže, zbog svoje kompleksnosti, implementirane su u više različitih datoteka.

Metodologija k-najbližih suseda data je funkcijom `k_najblizih_suseda`, koja kao ulazne parametre uzima skup svih obučavajućih podataka `skup_podataka` i jedan konkretan podatak koji treba da se klasifikuje `uzorak_xyz`. U metodi se najpre definišu promenljive za pronađenu klasifikaciju podatka, `klasa`, i distanca koja je najbliža jednom obučavajućem podatku, `distanca`. Promenljiva `klasa` se postavlja na vrednost 1, što znači da je inicijalna klasifikacija novog podatka šifrirana sa 1 (što se kasnije dešifruje metodama navedenim u poglavlju 6.1), a distanca je postavljena na beskonačno. Ova metodologija je implementirana za 1-n suseda, što znači da se uzima klasifikacija koja je jednaka klasifikaciji prvog najbližeg suseda ulaznom podatku. Dužina promenljive `skup_podataka` je upravo 8, onoliko koliko ima i klasa, i za svaku tu klasu se vrši računanje distanca za svaki podatak iz klase. Dakle, za svaku klasu i za svaki podatak se najpre preuzimaju koordinate obučavajućeg podatka, i vrši se izračunavanje distance od ulaznog podatka do obučavajućeg podatka funkcijom `izracunaj_distancu`, koja je definisana unutar iste datoteke kao i funkcija `k_najblizih_suseda`. Izračunavanje distance vrši se funkcijom:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{6.1}$$

Ukoliko je nova izračunata distanca manja od vrednosti zabeležene u promenljivoj `distanca`, onda se beleži klasifikacija i promenljiva `distance` dobija ovu novu vrednost. Izlazne vrednosti funkcije su `distance` od ulaznog podatka do najbližeg obučavajućeg podatka, koordinate novog podatka, i obeležje klasifikacije novog podatka.

```
function [klasa, distanca, xyz] = k_najblizih_suseda(skup_podataka, uzorak_xyz)
```

```
xyz = uzorak_xyz;
    klasa = 1;
    distanca = Inf;
for klasa_idx = 1 : length(skup_podataka)
for koordinate_idx = 1 : size(skup_podataka(klasa_idx).xyz, 1)
            xyz_klase                                                =
skup_podataka(klasa_idx).xyz(koordinate_idx, :);
izracunata_distanca       =       izracunaj_distancu(xyz_klase,
uzorak_xyz);
if izracunata_distanca < distanca
                klasa = klasa_idx;
                distanca = izracunata_distanca;
end
end
end
end

function [distanca] = izracunaj_distancu(xyz1, xyz2)
    distanca = sqrt(sum(power(xyz1 - xyz2, 2)));
end
```

IZLISTAK 6.11. SADRŽAJ DATOTEKE K_NAJBLIZIH_SUSEDA.M

Metodologija klasifikatora „naivni" Bejz implementirana je unutar datoteke *naivni_bajes.m*, u kojoj su takođe implementirane pomoćne metode vezane za tu metodologiju. Funkcija `naivni_bajes` za ulazne parametre uzima sve podatke za obučavanje i ulazni podatak koji je potrebno klasifikovati. Prvo se definiše promenljiva `sve_klase_i_verovatnoce` koja sadrži proračune verovatnoća da ulazni podatak pripada određenoj klasi. Za svaku poznatu klasifikaciju u ulaznom skupu podataka, preuzimaju se koordinate svih tačaka posebno, kao i ukupna količina podataka zadate klase. Za svaku koordinatu se računaju srednje vrednosti, koje se kao vektor prosleđuju metodi `izracunaj_devijaciju` koja izračunava standardnu devijaciju koristeći koordinate podataka iz trening uzorka i izračunate srednje vrednosti. Ove devijacije se dalje koriste u kombinaciji sa srednjim vrednostima i koordinatama ulaznih podataka kako bi se izračunale Gausove distribucije. Dobijene distribucije se množe, i proizvod se čuva kao rezultujuća verovatnoća za jednu klasifikaciju. Na kraju proračuna verovatnoća za svaku klasifikaciju, uzima se najveća verovatnoća i klasifikacija za tu verovatnoću, što se i vraća kao povratna vrednost funkcije.

```
function [klasa, verovatnoca, xyz] = naivni_bajes(klase,
xyz_uzorak)
    sve_klase_i_verovatnoce = [];
for klasa_id = 1:length(klase)
        xyz_klase = klase(klasa_id).xyz;
```

```
        [kolicina_uzoraka, ~] = size(xyz_klase);
x = xyz_klase(:, 1);
        y = xyz_klase(:, 2);
        z = xyz_klase(:, 3);
        xu = xyz_uzorak(1);
        yu = xyz_uzorak(2);
        zu = xyz_uzorak(3);
        srednje_vrednosti      =      [sum(x)/kolicina_uzoraka,
sum(y)/kolicina_uzoraka, sum(z)/kolicina_uzoraka];
        devijacije          =          [izracunaj_devijaciju(x,
srednje_vrednosti(1)),                  izracunaj_devijaciju(y,
srednje_vrednosti(2)),                  izracunaj_devijaciju(z,
srednje_vrednosti(3))];
        distribucije        =        [gausova_distribucija(xu,
srednje_vrednosti(1), devijacije(1)), gausova_distribucija(yu,
srednje_vrednosti(2), devijacije(2)), gausova_distribucija(zu,
srednje_vrednosti(3), devijacije(3))];
verovatnoca = prod(distribucije);
        sve_klase_i_verovatnoce   =   [sve_klase_i_verovatnoce;
klasa_id, verovatnoca];
end
    verovatnoca = max(sve_klase_i_verovatnoce(:, 2));
    xyz = [xu, yu, zu];
    [klasa, ~] = find(verovatnoca == sve_klase_i_verovatnoce);
end

function      [dev]      =      izracunaj_devijaciju(vrednosti,
srednja_vrednost)
    dev = sqrt(sum(power(vrednosti - srednja_vrednost, 2)) /
length(vrednosti));
end

function                    [gausov_res]                    =
gausova_distribucija(vrednost_atributa,
srednja_vrednost_atributa, devijacija_atributa)
    numerator_exponent     =     -power((vrednost_atributa     -
srednja_vrednost_atributa), 2);
    denumerator_exponent = 2 * power(devijacija_atributa, 2);
    moj_e = exp(numerator_exponent/denumerator_exponent);
    denumerator = sqrt(2 * pi * power(devijacija_atributa,
2));
    gausov_res = moj_e / denumerator;
end
```

IZLISTAK 6.12. FUNKCIJA NAIVNI_BAJES

Programski najkomplikovanija implementacija od ove tri metodologije jeste implementacija veštačkih neuronskih mreža. Za implementaciju veštačkih neuronskih mreža potrebno je najpre definisati klasu koja će sadržati osnovnu strukturu i funkcionalnosti neurona. Ovo je definisano unutar klase `Neuron`. Klasa `Neuron` sadrži polja koja opisuju težine na vezama neurona, broj ulaza neurona, kao i pomoćna polja koja će čuvati vrednosti za propagiranu grešku neurona, poslednju predikciju koju je neuron napravio (poslednji izračunati izlaz neurona), kao i poslednji skup ulaza koji su prosleđeni neuronu. Prilikom

kreiranja neurona, prosleđen je parametar koji govori koliko ulaza neuron treba da ima i toliko puta se generišu slučajne vrednosti za inicijalne težine na vezama neurona. Dodatno se generiše još jedna slučajna vrednost za bias neurona. Funkcija `forward_pass` dobija vektor ulaznih vrednosti prosleđenih od prethodnog sloja neurona. Ove ulazne vrednosti se množe sa odgovarajućim težinama veza sa kojih su prosleđene vrednosti, a ti proizvodi se sumiraju. Suma se dalje koristi kako bi se izračunala vrednost sigmoidne funkcije, koja je odabrana kao aktivaciona funkcija neurona. Pre korišćenja ulaznih vrednosti, neuron pamti vrednosti koje su prosleđene, i nakon proračuna vrednosti sigma funkcije, predikcija se memoriše.

```
function [res] = forward_pass(obj, ulazne_vrednosti)
            obj.poslednji_ulazi = [ulazne_vrednosti, 1];
            [r, c] = size(ulazne_vrednosti);
if c < r
                ulazne_vrednosti = ulazne_vrednosti';
end
            ulazne_vrednosti = [ulazne_vrednosti, 1];
            suma_vrednosti     =     sum(ulazne_vrednosti     .*
obj.tezine);
            res = 1 / (1 + exp(-suma_vrednosti));
            obj.poslednja_predikcija = res;
end
```

IZLISTAK 6.13. FUNKCIJA FORWARD_PASS

Klasa `Neuron` takođe ima mogućnost korigovanja težina i izračunavanja greške za prosleđivanje greške unazad kroz neuronsku mrežu. Backpropagation algoritam je implementiran kroz više datoteka, a u klasi `Neuron` je implementirano u funkciji `preuzmi_backprop_greske`. Ova funkcija množi svaku svoju težinu sa izračunatom deltom greške, koja je postavljena izvan klase `Neuron`. Korigovanje težina vrši se za čitav vektor težina neurona, unutar funkcije `koriguj_tezine`.

```
function [backprop_greske] = preuzmi_backprop_greske(obj)
            backprop_greske = [];
for tezina = obj.tezine
                backprop_greske  =  [backprop_greske,  tezina  *
obj.delta_greska];
end
            backprop_greske(end) = [];
end
function [] = koriguj_tezine(obj, konstanta_ucenja)
            obj.tezine   =   obj.tezine   +   konstanta_ucenja   *
obj.delta_greska    *    (obj.poslednja_predikcija    *    (1    -
obj.poslednja_predikcija)) * obj.poslednji_ulazi;
end
```

IZLISTAK 6.14. FUNKCIJE PREUZMI_BACKPROP_GREŠKE, I FUNKCIJA KORIGUJ_TEŽINE

Objekte klase `Neuron` koristi klasa `NeuronskaMreza`. `NeuronskaMreza` sastoji se od vektora slojeva neurona i od promenljive koja definiše koliko će se brzo odvijati učenje neuronske mreže. Prilikom kreiranja objekta neuronske mreže, prosleđuje se konfiguracija mreže kao ulazni parametar. Ova konfiguracija je u obliku matrice, gde svaki red definiše konfiguraciju za jedan sloj. Prva kolona u matrici govori koliko neurona ima u sloju, a druga kolona govori koliko ulaza svaki neuron treba da ima. Tako, na primer, za konfiguraciju koja je uzeta za implementaciju zadatka: `[8, 3; 16, 8; 8, 16]` govori da neuronska mreža ima 3 sloja, prvi sloj ima 8 neurona sa po 3 ulaza, drugi sloj ima 16 neurona sa po 8 ulaza, a poslednji sloj ima 8 neurona sa 16 ulaza. Pomoću ove konfiguracije se može na vrlo generičan način kreirati onoliko slojeva i onoliko neurona po sloju koliko je potrebno za neuronsku mrežu. Konstanta učenja je u implementaciji fiksirana na vrednost 0.5.

```
function obj = NeuronskaMreza(konfiguracija_slojeva)

slojevi = {};
for sloj_idx = 1 : size(konfiguracija_slojeva, 1)
                sloj = [];
for neuron_idx = 1 : konfiguracija_slojeva(sloj_idx, 1)
                    sloj                =                [sloj,
Neuron(konfiguracija_slojeva(sloj_idx, 2))];
end
                slojevi{end+1} = sloj;
end
            obj.slojevi = slojevi;
            obj.broj_slojeva = size(konfiguracija_slojeva, 1);
            obj.konstanta_ucenja = 0.5;
end
```

IZLISTAK 6.15. KONSTRUKTOR KLASE NEURONSKAMREZA

Izračunavanje rezultata na izlaznom sloju neuronske mreže je prilično jednostavna procedura. Ova procedura obuhvata izračunavanje izlaza na neuronima od prvog sloja do poslednjeg. Počevši od prvog sloja, svaki neuron iskorišćava svoju definisanu internu funkciju `forward_pass` za proračun izlaza i na taj način ulaz u sledeći sloj neuronske mreže postaje izlaz iz trenutnog sloja. Jedino što je `forward_pass` metodi neuronske mreže potrebno je inicijalan vektor ulaza (eksterni ulazi neuronske mreže).

```
function [predikcije] = forward_pass(obj, ulaz)
for sloj = obj.slojevi
                nov_ulaz = [];
for neuron = sloj{:}
```

```
                    nov_ulaz = [nov_ulaz,
neuron.forward_pass(ulaz)];
end
                ulaz = nov_ulaz;
end
            predikcije = nov_ulaz;
end
```

IZLISTAK 6.16. METODA FORWARD_PASS NEURONSKE MREŽE

Treniranje neuronske mreže izvršeno je van klase `NeuronskaMreza`, ali se koristi njena interna metoda `jedan_trening`, koji uzima jedan ulazni podatak za trening koji sadrži rešenje (očekivanu klasifikaciju za podatak). Na početku treninga mreže se izvrši jedno izračunavanje predikcije, koje se iskoristi zajedno sa rešenjem da se nađe greška koju je neuronska mreža napravila. Greška se koristi kako bi se izvršio *backpropagation* sa neuronima na poslednjem sloju. Greške propagirane od svakog neurona se stavljaju u jedan red matrice trenutnih grešaka i iz ove matrice čitava *i*-ta kolona će se koristiti kao greške koje se propagiraju u *i*-ti neuron u prethodnom sloju. Prethodni sloj, uz pomoć ovih grešaka propagira sledeći skup grešaka, sve do prvog sloja neurona. Uz propagiranje greške, vrši se i pamćenje ovih delti, kako bi se kasnije lakše izvelo korigovanje težina na vezama. Dakle, korigovanje težina na vezama je prosto stvar iteracije kroz svaki sloj i svaki neuron na sloju i pozivanju metode `koriguj_tezine` za zadati neuron.

```
function [greska] = jedan_trening(obj, ulaz, resenje)
            predikcija = obj.forward_pass(ulaz);
            greska = resenje - predikcija;
            struktura_gresaka = [{}];
            poslednji_sloj = obj.slojevi{end};
            matrica_gresaka = [];
for neuron_idx = 1 : length(poslednji_sloj)
neuron = poslednji_sloj(neuron_idx);
                neuron.delta_greska = greska(neuron_idx);
                propagacija_greske_neurona =
neuron.preuzmi_backprop_greske();
                matrica_gresaka = [matrica_gresaka;
propagacija_greske_neurona];
end
            struktura_gresaka = {matrica_gresaka};
for sloj_idx = length(obj.slojevi)-1:-1:1
                sloj_neurona = obj.slojevi{sloj_idx};
                prethodne_greske = struktura_gresaka{1};
                matrica_gresaka = [];

for neuron_idx = 1 : length(sloj_neurona)
neuron = sloj_neurona(neuron_idx);
                    sigma_za_neuron = sum(prethodne_greske(:,
neuron_idx));
                    neuron.delta_greska = sigma_za_neuron;
                    propagacija_greske_neurona =
neuron.preuzmi_backprop_greske();
```

```
                    matrica_gresaka = [matrica_gresaka;
propagacija_greske_neurona];
end
                struktura_gresaka = [{matrica_gresaka},
struktura_gresaka];
end
for sloj_idx = 1:length(obj.slojevi);
                sloj = obj.slojevi{sloj_idx};
for neuron_idx = 1:length(sloj)
neuron = sloj(neuron_idx);

neuron.koriguj_tezine(obj.konstanta_ucenja);
end
end
end
```

IZLISTAK 6.17. JEDNA ITERACIJA TRENIRANJA NEURONSKE MREŽE

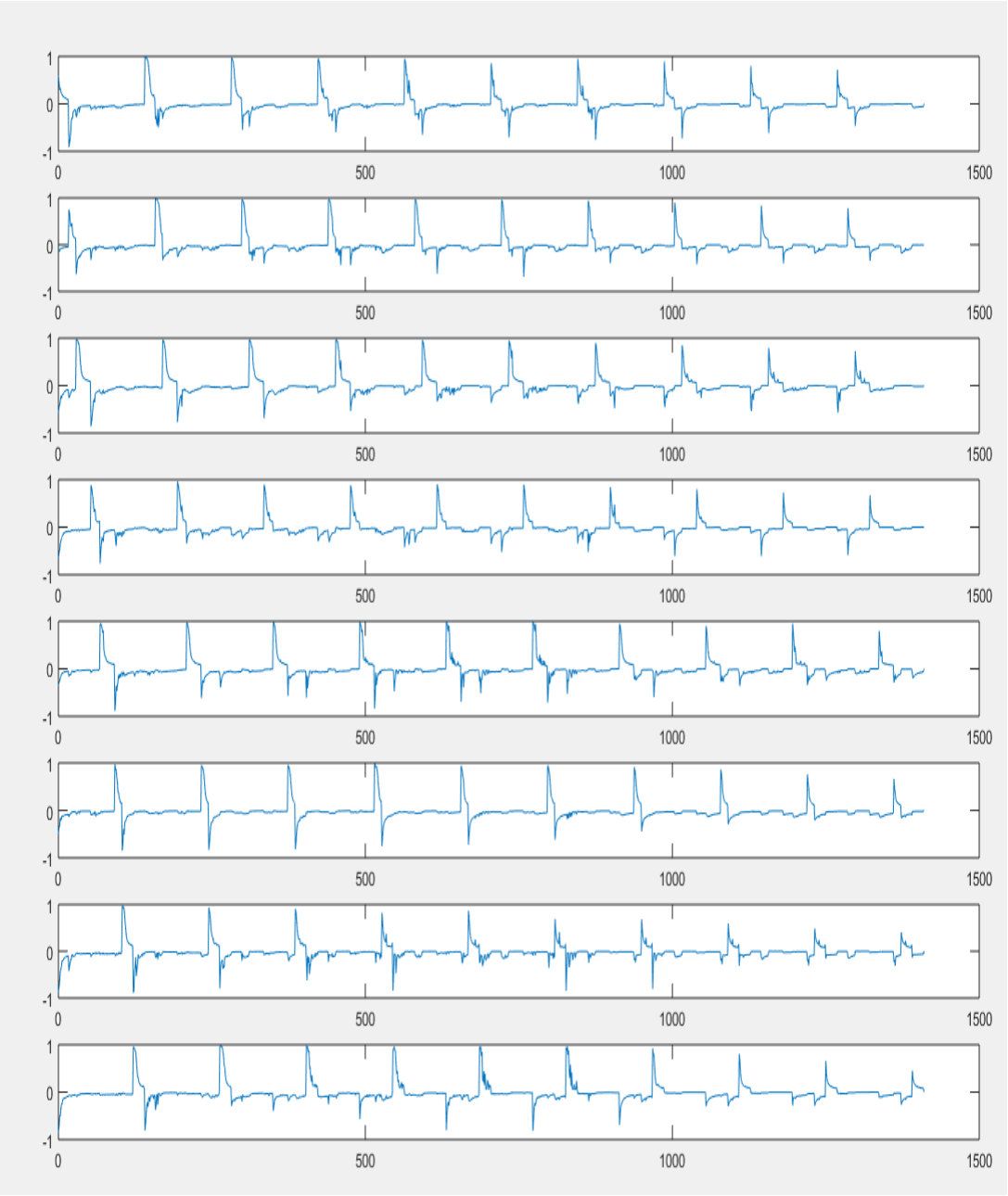
Trening veštačke neuronske mreže vrši se u glavnoj skripti zadatka, pozivajući metodu `treniraj_neuronsku_mrezu`. Ovoj metodi se prosleđuju podaci na osnovu kojih će neuronska mreža biti obučena, kao i broj epoha za korišćenje podataka. Najpre se kreira objekat klase `NeuronskaMreza` sa konfiguracijom od po 3 sloja neurona, sa 8, 16, i 8 neurona od početnog do poslednjeg izlaznog sloja. Nakon konfiguracije mreže formira se skup rešenja za ulazne podatke, tako što se jedinica postavlja na indeks klase koja treba da se nađe na izlaznom sloju. Za ulazni primer koji treba biti klasifikovan kao klasa 4, rešenje bi imalo oblik vektora sa vrednostima $[0, 0, 0, 1, 0, 0, 0, 0]$ - tako da se na četvrtom indeksu nalazi vrenost 1 – *true*.

```
for klasifikacija = 1 : length(uzorci)
        resenje_za_klasu = zeros(1, length(uzorci));
        resenje_za_klasu(klasifikacija) = 1;
for red = 1 : size(uzorci(klasifikacija).xyz, 1)
            ulazne_vrednosti        =        [ulazne_vrednosti;
uzorci(klasifikacija).xyz(red, :)];
            resenja = [resenja; resenje_za_klasu];
end
end
```

IZLISTAK 6.18. KREIRANJE REŠENJA ZA ULAZNE PODATKE

Kada imamo formirana rešenja, sve što je potrebno da se uradi jeste da se iskoriste gotove metode klasa `Neuron`i `NeuronskaMreza`, kao i da se kreira način za praćenje treninga neuronske mreže. Najpre se formira niz vremenskih trenutaka u kojima se trening odvija – ovo je predstavljeno brojevima od 0 do proizvoda epoha sa brojem ulaznih podataka. U promenljivu `metrika_greske` čuvaju se vrednosti koje svaki neuron na izlaznom sloju proizvodi (ukupno 8 vrednosti od -1 do 1). Za svaku ulaznu vrednost neuronska mreža izvršava funkciju `jedan_trening`, kako bi korigovala vrednosti na vezama između slojeva i vraća izračunate vrednosti na izlaznim neuronima. Na kraju treninga neuronske mreže, prikazuju se grafici učenja, na kojima se može videti (slika

6.2) kako svaki izlazni neuron po svakoj epohi sve manju grešku pravi prilikom predikcija.



SLIKA 6.2. GREŠKE IZLAZNIH VREDNOSTI NEURONA NA IZLAZNOM SLOJU

Na kraju obuke neuronske mreže, struktura neuronske mreže (instanca klase) memoriše se u datoteku `trenirana-mreza.mat`, kako bi se izbegao proces treninga sledeći put kada je potrebno izvršiti klasifikaciju.

```
metrika_vreme = 0:broj_epoha * size(resenja, 1);
    metrika_greske        =        zeros(length(metrika_vreme),
size(mreza.slojevi{end}, 2));
    metrika_greska_idx = 1;
for trenutna_epoha = 1 : broj_epoha
for ulaz_idx = 1 : size(ulazne_vrednosti, 1)
            ulaz_red = ulazne_vrednosti(ulaz_idx, :);
            resenje_red = resenja(ulaz_idx, :);
            greska       =       mreza.jedan_trening(ulaz_red,
resenje_red);
            metrika_greske(metrika_greska_idx, :) = greska;
            metrika_greska_idx  = metrika_greska_idx +1;
            clc
            trenutna_epoha
            metrika_greska_idx
end
end
    figure(2)
    subplot(size(metrika_greske, 2), 1, 1)
for idx = 1 : size(metrika_greske, 2)
        subplot(size(metrika_greske, 2), 1, idx);
        plot(metrika_vreme, metrika_greske(:, idx))
end
    pause
    save('trenirana-mreza.mat', 'mreza');
```

IZLISTAK 6.19. OBUKA NEURONSKE MREŽE

Klasifikacija pomoću trenirane neuronske mreže izvršava se u funkciji `neuronske_mreze_obj`, kojoj se prosleđuje neklasifikovani podatak. Na početku se pokušava učitavanje trenirane neuronske mreže, koja se koristi kako bi se pozvala metoda `forward_pass` koja proizvodi finalne predikcije neurona na izlazu mreže, od kojih se uzima predikcija sa najvećom vrednošću kao konačan rezultat predikcije neuronske mreže.

```
function        [predikcije,        klasa,        xyz]        =
neuronske_mreze_obj(neklasifikovani_uzorak)
    load('trenirana-mreza.mat')
    xyz = neklasifikovani_uzorak;
    predikcije                                                  =
round(mreza.forward_pass(neklasifikovani_uzorak), 2);
    [~, klasa] = find(max(predikcije) == predikcije);
end
```

IZLISTAK 6.20. KLASIFIKACIJA NEURONSKOM MREŽOM

Simulacija se pokreće tako što se pokrene izvršavanje `MainScript.m` datoteke. Na početku skripte se brišu sve promenljive iz memorije, briše se prikaz sa ekrana i zatvaraju se svi otvoreni grafici. Generišu se trening podaci, kao i nepoznati podaci koji će predstavljati ulazne podatke u svaki model

klasifikatora. Prave se četiri trodimenzionalna grafika koji predstavljaju inicijalno stanje sistema, stanje nakon klasifikacije sa „naivnim" Bejzom, k-najbližih suseda i neuronskim mrežama, respektivno.

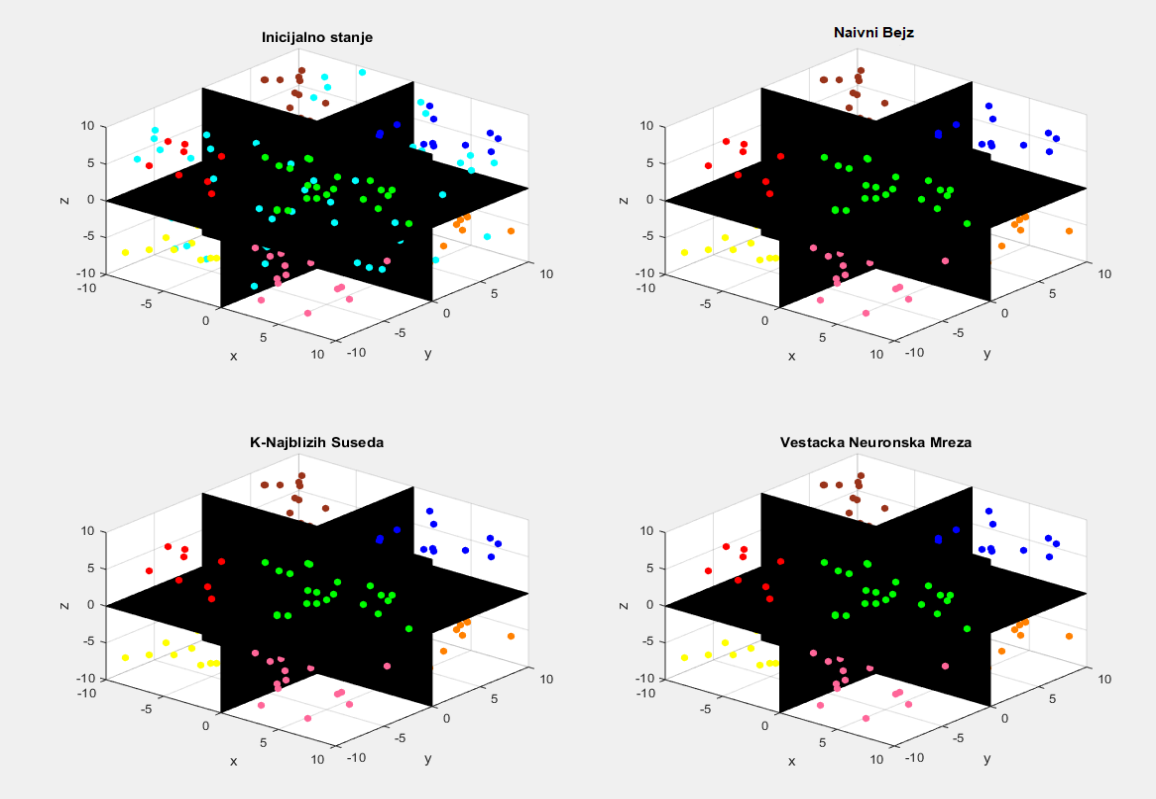
```
clear;
clf;
clc;
[trening_skup, nepoznati_uzorci] = generisi_skup_podataka(10,
25);

for klasifikacija = 1 : 4
    subplot(2, 2, klasifikacija);
if klasifikacija == 1
        naslov = 'Inicijalno stanje';
elseif klasifikacija == 2
        naslov = 'Naivni Bejz';
elseif klasifikacija == 3
        naslov = 'K-Najblizih Suseda';
else
        naslov = 'Vestacka Neuronska Mreza';
end
    nacrtaj_zonu(naslov);
end

for klasifikacija = 1 : length(trening_skup)
    skup_podataka = trening_skup(klasifikacija).xyz;
for sp = 1 : 4
        subplot(2,2,sp);
        hold on;
for red_idx = 1 : size(skup_podataka, 1)
            red_podataka = skup_podataka(red_idx, :);
            nacrtaj_tacku(red_podataka, klasifikacija, false);
end
end
end

subplot(2, 2, 1);
for nepoznati_podatak_idx = 1 : size(nepoznati_uzorci, 1)
    red_podataka = nepoznati_uzorci(nepoznati_podatak_idx, :);
    nacrtaj_tacku(red_podataka, 9, false);
end
```

IZLISTAK 6.21. PRIPREMA PODATAKA I GRAFIKA



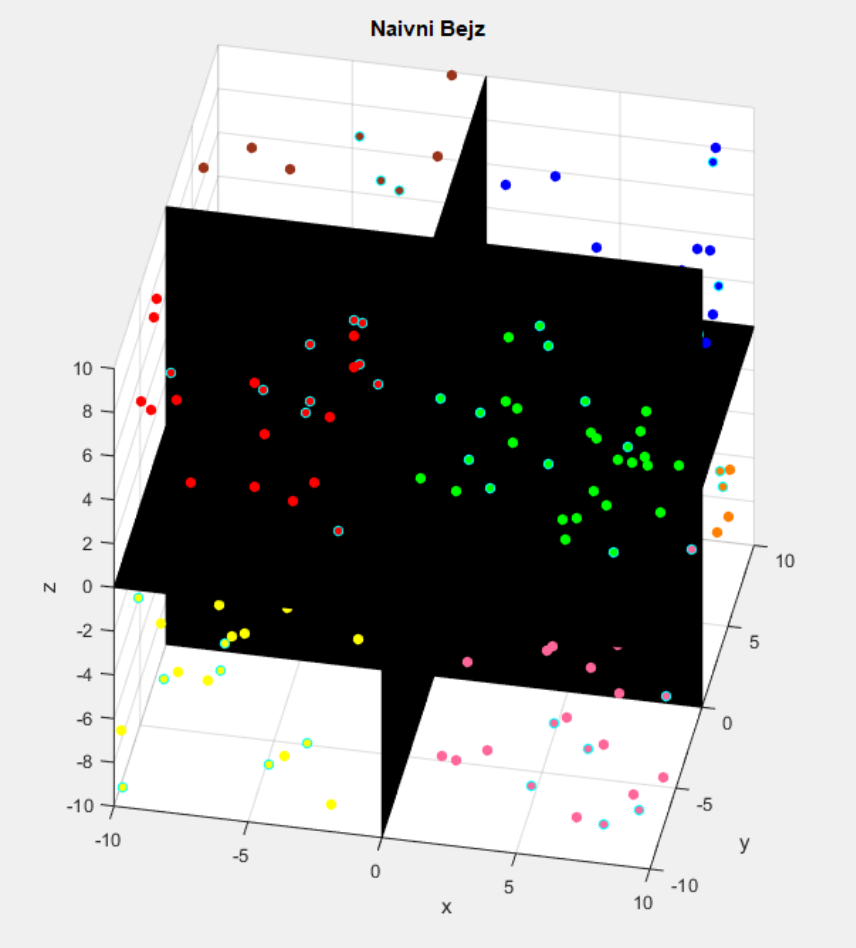
SLIKA 6.3. INICIJALNA STANJA SISTEMA

Na slici 6.3 prikazano je inicijalno stanje simulacije sa stanjima na kojima će biti iscrtani rezultati metodologija. Gornji levi grafik sadrži tačke koje nisu klasifikovane (tirkizne boje) i koje će u ostalim metodologijama biti ispunjene bojama odgovarajuće klase.

Kada su generisane i iscrtane sve tačke, odabira se drugi grafik za crtanje i jedan po jedan obučavajući podaci se prosleđuju funkciji `naivni_bajes` za klasifikaciju. Kako se proizvede rezultat, tako se na odabrani grafik obeleži klasifikovana tačka, tako da je obojena bojom klasifikacije, a oko nje ostavljena svetlo plava boja, da se zna da je bila neklasifikovana (radi lakšeg traženja ranije neklasifikovanih podataka).

```
subplot(2,2,2);
hold on;
for nepoznati_podatak_idx = 1 : size(nepoznati_uzorci, 1)
    red_podataka = nepoznati_uzorci(nepoznati_podatak_idx, :);
    [klasa, verovatnoca, xyz] = naivni_bajes(trening_skup,
red_podataka);
    nacrtaj_tacku(xyz, klasa, true);
end
```

IZLISTAK 6.22. KLASIFIKACIJA "NAIVNIM" BEJZOM



SLIKA 6.4. REZULTAT KLASIFIKACIJE „NAIVNIM" BEJZOM

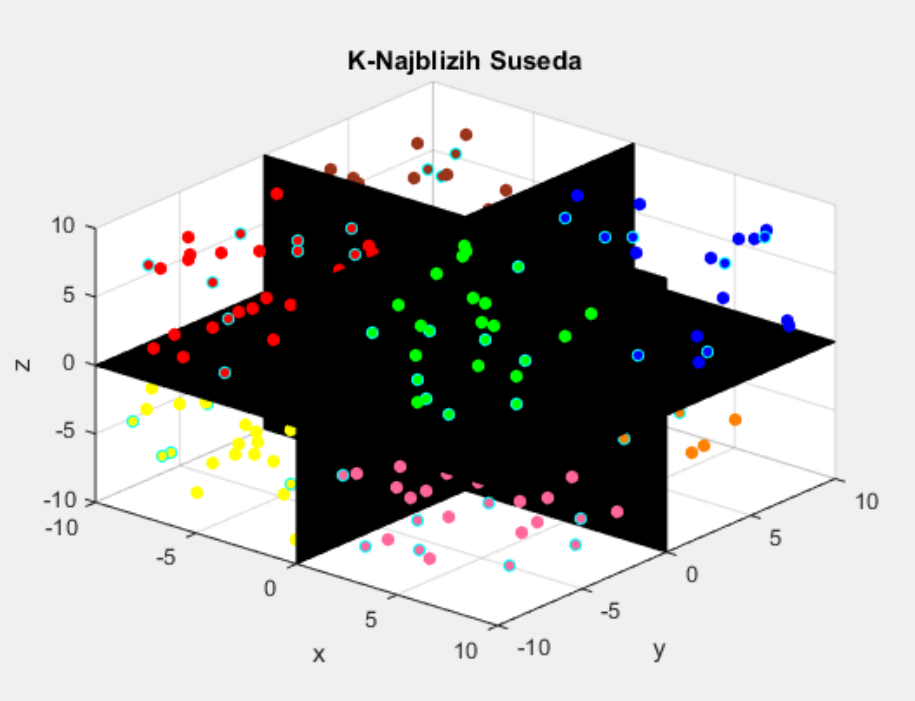
Na slici 6.4 prikazan je jedan pogled na grafik na kojem je iscrtan rezultat klasifikacije „naivnim" Bejzom. Sve tačke koje imaju okvir tirkizne boje su tačke koje su trebale da se klasifikuju, odnosno, predstavljaju tačke koje su bile potpuno obojene tirkiznom bojom u gornjem levom grafiku na slici 6.3. Sve nove tačke koje su klasifikovane su korektno klasifikovane, odnosno ispunjene su bojom odgovarajućih klasa, kao što se vidi na slici 6.4.

Nakon klasifikacije „naivni" Bejz, na isti način se jedan po jedan obučavajući podatak prosleđuje metodi `k_najblizih_suseda` za klasifikaciju.

```
subplot(2,2,3);
hold on;
for nepoznati_podatak_idx = 1 : size(nepoznati_uzorci, 1)
    red_podataka = nepoznati_uzorci(nepoznati_podatak_idx, :);
```

```
    [klasa, distanca, xyz] = k_najblizih_suseda(trening_skup,
red_podataka);
    nacrtaj_tacku(xyz, klasa, true);
end
```

IZLISTAK 6.23. KLASIFIKACIJA POMOĆU KLASIFIKATORA K-NAJBLIŽIH SUSEDA.

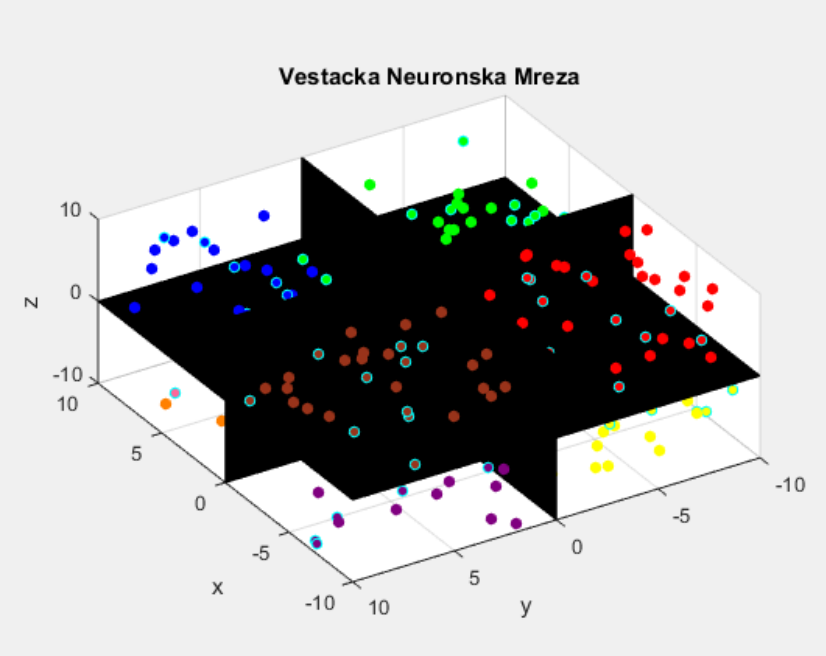


SLIKA 6.5. STANJE SISTEMA NAKON KLASIFIKACIJE POMOĆU METODOLOGIJE K-NAJBLIŽIH SUSEDA

Na slici 6.5 prikazan je rezultat klasifikacije KNN klasifikatora. Vrlo slično klasifikatoru „naivni" Bejz, gotovo svi podaci ispunjeni su odgovarajućim bojama. Jedno odstupanje predstavlja tačka u zoni tačaka klasifikovane rozom bojom, gde je podatak greškom klasifikovan braon bojom (gornji desni ugao zone rozih tačaka). U ovakvim situacijama, greške ovog tipa nisu neuobičajene, jer je zona imala nedovoljno tačaka u gornjem desnom uglu, pa je klasifikator pronašao bliže susede u zoni braon tačaka.

Pre nego što se pređe na klasifikaciju pomoću veštačkih neuronskih mreža, prvo se proverava indikator koji ukazuje na to da li je potrebno učenje neuronske mreže. Kao što je već navedeno, učenje neuronske mreže je dugačak proces i ukoliko već postoji treniran (i dovoljno precizan) model dovoljno je jednostavno učitati model. Ako ne postoji dovoljno dobar model, indikator se može postaviti na vrednost *true* i simulacija će izvršiti treniranje i memorisanje novog modela veštačke neuronske mreže. Nakon ove provere i

potencijalnog treniranja modela, neuronska mreža uzima ulazne podatke i vrši klasifikaciju.



SLIKA 6.6. REZULTAT KLASIFIKACIJE VEŠTAČKIM NEURONSKIM MREŽAMA

Slika 6.6 prikazuje kako izgleda sistem nakon klasifikacije sa veštačkim neuronskim mrežama. Većina tačaka je korektno klasifikovana, ali u zoni plavih tačaka ima dva odstupanja. Neuronska mreža je klasifikovala dve tačke u zelenu boju, umesto u plavu. U slučaju veštačkih neuronskih mreža, ovo obično znači da je bilo manje plavih tačaka u skupu za obučavanje neuronske mreže, ili da je bilo više zelenih tačaka bliže toj konkretnoj zoni.

IZLISTAK 6.24. KLASIFIKACIJA VEŠTAČKIM NEURONSKIM MREŽAMA

```
subplot(2, 2, 4);
hold on;
for nepoznati_podatak_idx = 1 : size(nepoznati_uzorci, 1)
    red_podataka = nepoznati_uzorci(nepoznati_podatak_idx, :);
    [predikcije,            klasa,            xyz]            =
neuronske_mreze_obj(red_podataka);
    nacrtaj_tacku(xyz, klasa, true);
end
```

U ovom poglavlju predstavljena je programska implementacija tri poznata načina klasifikovanja podataka. Implementirani su klasifikatori k-najbližih suseda, „naivni” Bejz klasifikator i klasifikacija pomoću neuronskih mreža.

Klasifikovanje je izvršeno nad podacima koji su veštački generisani, od strane programa.

Prikupljanjem dovoljno velikog skupa obučavajućih podataka iz realnog sveta, kao i njihovom korektnom obeležavanju i promeni obeležja i načina dešifrovanja obeležja u kodu, moguće je skalirati ovaj primer na neki koji se može primeniti u realnom svetu. Preciznost klasifikatora u najvećoj meri zavisi od obučavajučih skupova i u ovom primeru se vidi da klasifikatori za sve nove podatke korektno predviđaju njihove klase.

ZAKLJUČAK

Veštačka inteligencija se sve više primenjuje u praksi. Istraživači konstantno rade na usavršavanju metodologija veštačke inteligencije i na ispravljanju nedostataka u ovom polju.

Postoje razni načini za rešavanje praktičnih problema i obično nije moguće primeniti svaku od ovih metodologija za rešavanje bilo kog problema. Do sada su neuronske mreže pokazale najviše obećanja u praksi, zbog mogućnosti konstrukcija različitih topologija mreža u svrhu rešavanja problema. Ipak, zbog potrebe obučavajućeg skupa, možda je nekada bolje odabrati jednostavnije rešenje, kao što su klasifikatori poput „Naivnog Bejza“, za brzo i dovoljno precizno rešavanje problema.

U ovom radu demonstrirane su metodologije klasifikacija k-najbližih suseda, „naivni“ Bejz klasifikator i klasifikacija pomoću neuronskih mreža. Ove metodologije odabrane su zbog svog značaja u praksi.

„Naivni“ Bejz klasifikator je odabran zbog načina implementacije, koji zavisi od matematičkih definicija i formula. Ovaj klasifikator se može relativno jednostavno obučiti sa prilično malim skupom obučavajućih podataka, kao što je i pokazano u radu, a da i dalje proizvodi dobre rezultate. Implementacija „naivnog“ Bejza u radu se može skalirati i na skupove koji imaju više karakteristika. Za ovako nešto se ne preporučuje korišćenje klasifikatora „naivni“ Bejz, zbog činjenice da predpostavlja da su sve karakteristike jednog podatka nezavisne.

Performanse KNN klasifikatora su vrlo slične klasifikatoru „naivni“ Bejz, u ovom konkretnom slučaju. Problem kod klasifikatora KNN je u performansama koje zavise od metrike rastojanja koja će se koristiti u određivanju najbližih suseda. U ovom radu korišćeno je Euklidovo rastojanje i proizvelo je generalno dobre rezultate. Međutim, kada bi podaci imali više karakteristika, proračun Euklidovog rastojanja za računanje distance od ulaznog podatka do svakog obučavajućeg podatka bi bio izuzetno računarski zahtevan. KNN klasifikator se

često koristi u problemima kada karakteristike zavise jedne od drugih i postoji jasnija razlika podataka koji se proveravaju (u obradi slika, za lociranje objekata po boji itd.). KNN klasifikator se takođe može koristiti kao međukorak nekog složenijeg sistema baziranog na veštačkoj inteligenciji, da se reši jednostavniji problem čiji izlaz predstavlja ulaz sledećeg, kompleksnijeg koraka.

Implementacija neuronskih mreža predstavlja najkompleksniji praktični deo rada. Neuronske mreže, implementirane u bilo kojoj arhitekturi, imaju izuzetno široku primenu u praksi. Razni programski paketi koji se bave veštačkom inteligencijom imaju predefinisane arhitekture neuronskih mreža. U ovom radu, neuronske mreže implementirane su tako da korisnik može svojom voljom odrediti arhitekturu neuronske mreže koju gradi za rešavanje određenog problema. U radu je konstruisana neuronska mreža sa tri sloja, ulazni, izlazni i skriveni sloj, jer se iterativnim postupkom pokazalo da ovakva arhitektura pokazuje najbolje rezultate. Daljim razvojem i testiranjem arhitekture bi se najverovatnije moglo doći i do boljih rezultata. Koliko god dobra arhitektura neuronske mreže bila, opet klasifikacija zavisi od obučavajućih podataka. Neuronska mreža u ovom radu na početku generiše nasumične brojeve za inicijalizaciju težina na vezama, koje se dalje koriguju procesom treniranja. Proces treniranja neuronske mreže je u radu implementiran kao sopstvena funkcija i ne pokreće se kada je potrebno izvršiti predikciju. Na ovaj način, korisnik može sačuvati određeno stanje neuronske mreže, bez da za svaku simulaciju mora da čeka da prođe često dugačak period treniranja neuronske mreže.

Veštačka inteligencija se unapređuje iz dana u dan. Automatizovana analiza dokumenata, prepoznavanje oblika i likova na slikama, kao i vožnja automobila bez vozača su sve oblasti koje zavise od veštačke inteligencije. Ovakve praktične zadatke je nekada samo čovek svojim rezonovanjem mogao da rešava. Vremenom, čovek će sve više svojih poslova moći da prepusti mašinama koje će to raditi umesto njega.

LITERATURA

[1] Abedi Parisa, *Machine Learning Sessions, Linear Classification*, 2016.

[2] Carvalho Carlos , *Classification: Logistic Regression and Naive Bayes, Book Chapter 4*, The University of TexasMcCombs School of Busines, 2013.

[3] Cover T., *Nearest Neighbor Pattern Classification, IEEE Transactions on Information Theory*, IEEE Transactions on Information Theory, 1967.

[4] Cunningham Padraig, Delany Sarah, *K-Nearest neighbour classifiers,* ResearchGate, 2007.

[5] Farhadi Farnoush, *Learning activation functions in deep neural networks,* Univeristy of Montreal, 2017.

[6] Gramacki A., *Nonparametric Kernel Density Estimation and its Computational Aspects,* Springer, 2018.

[7] Larsen Jan, *Introdution to artifical neural networks,* Department of mathematical modelling, Technical University of Denmark,1999.

[8] Lozanov-Crvenković Zagorka, *Statistika*, Prirodno-matematički fakultet, Novi Sad, 2008.

[9] Michie D.,Spiegelhalter D., *Machine Learning, Neural and Statistical Classification,* Overseas Press, 1994.

[10] Mitchell Tom, *Machine Learning,* McGraw-Hill Science, 2015.

[11] Orloff Jeremy , Bloom Jonathan, *Conditional Probability, Independence and Bayes' Theorem,* 2014.

[12] Rajter-Ćirić Danijela, *Verovatnoća*, Univerzitet u Novom Sadu, Prirodno-matematički fakultet, Novi Sad, 2008.

[13] Russell Stuart, Norvig Peter, *Artificial Intelligence, A Modern Approach,* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[14] Sutton Oliver, *Introduction to k Nearest Neighbour Classification and Condensed Nearest Neighbour Data Reduction*, University of Leicester, 2012.

[15] Wagner Christian, *Comparison of Distance Metrics for Hierarchical Data in Medical Databases,* IEEE Xplore Digital Library, 2014.

[16] http://www.stat.yale.edu/Courses/1997-98/101/condprob.htm

[17]https://www.researchgate.net/profile/Emilija_Nikolic-Djoric/publication/277159877_Theoretical_and_practical_aspects_of_the_model_of_nonlinear_regression/links/598b0755a6fdcc7cf9254819/Theoretical-and-practical-aspects-of-the-model-of-nonlinear-regression.pdf

[18] https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

BIOGRAFIJA

Biljana Malinović rođena je 13.06.1986. godine u Glamoču, u Bosni i Hercegovini. Osnovnu školu “Mihajlo Pupin” u Veterniku završila je 2001. godine kao nosilac Vukove diplome. Iste godine upisuje gimnaziju “Svetozar Marković” u Novom Sadu, koju završava 2005. godine takođe kao nosilac Vukove diplome. 2007. godine upisuje studije matematike na Prirodno-matematičkom fakultetu u Novom Sadu, smer matematika finansija. Iste završava 2013. sa prosečnim uspehom 8.55, nakon čega upisuje i master studije na istom fakultetu, smer primenjena matematika. Trenutno je zaposlena u osnovnoj školi kao nastavnik matematike.

UNIVERZITET U NOVOM SADU
PRIRODNO MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj: **RBR**	
Identifikacioni broj: **IBR**	
Tip dokumentacije: **TD**	Monografska dokumentacija
Tip zapisa: **TZ**	Tekstualni štampani materijal
Vrsta rada: **VR**	Master rad
Autor: **AU**	Biljana Malinović
Mentor: **ME**	dr Ivana Štajner-Papuga
Naslov rada: **NR**	Neki metodi za klasifikaciju tačaka u trodimenzionalnom prostoru
Jezik publikacije: **JP**	srpski (latinica)
Jezik izvoda: **JI**	s/en
Zamlja publikovanja: **ZP**	Republika Srbija
Uže geografsko područje: **UGP**	Vojvodina
Godina:	2018.

GO

Izdavač: **IZ**	Autorski reprint
Mesto i adresa: **MA**	Novi Sad, Trd D. Obradovića 4
Fizički opis rada: **FOR**	(9/94/16/1/26/0/24) (brojpoglavlja/strana/lit.citata/tabela/slika/grafika/priloga)
Naučna oblast: **NO**	Matematika
Naučna disciplina: **ND**	primenjena matematika
Predmetne odrednica, ključne reči: (**PO**, **UDK**)	klasifikacija tačaka,veštačka inteligencija, metode klasifikacije
Čuva se: informatiku **ČS**	u biblioteci Departmana za matematiku i
Važna napomena: **VN**	nema
Izvod (**IZ**):	Ovaj rad se bavi pregledom osnovnih i najčešće korišćenih metodologija i algoritama u oblastima veštačke inteligencije i njihovim primenama u stohastičkim neprekidnim modelima.
Datum prihvatanja teme od strane NN veća: **DP**	03.09.2018.
Datum odbrane: **DO**	29. oktobar 2018. .

Članovi komisije:
KO

Predsednik:	Prof. dr Zagorka Lozanov-Crvenković, redovni profesor, PMF
Član:	Prof. dr Marta Takač, redovni profesor, Učiteljski fakultet na mađarskom nastavnom jeziku
Član:	Prof. dr Mirjana Štrboja, vanredni profesor, PMF
Mentor:	Dr Ivana Štajner-Papuga, redovni profesor, PMF

UNIVERSITY OF NOVI SAD
FACULTY OF NATURAL SCIENCES & MATHEMATICS
KEY WORDS DOCUMENTATION

Accession number:
ANO

Identification number:
INO

Document type: **DT**	Monograph type
Type of record: **TR**	Textual printed material
Contents code: **CC**	Master's thesis
Author: **AU**	Biljana Malinović
Mentor: **ME**	Dr Ivana Štajner-Papuga
Title: **TI**	Some mehtods for classification of points in three dimensional space
Language of text: **LT**	Serbian (Latin)
Language of abstract: **LT**	s /en
Country of publication: **CP**	Republic of Serbia
Locality of publication: **LP**	Vojvodina
Publication year: **PY**	2018.
Publisher: **PU**	author's reprint

Publ. place: **PP**	Novi Sad, Trd D. Obradovića 4
Physical description: **PD**	(9/94/16/1/26/0/24)
Scientific field: **SF**	Mathematics
Scientific discipline: **SD**	Applied mathematics
Subject Key words: of classification **SKW**	classification, artificial intelligence, methods
Holding data: **HD**	In the library of Department of Mathematics and Informatics
Note: **N**	
Abstract (**AB**):	This paper deals with an overview of the basic and most commonly used methodologies and algorithms in the areas of artificial intelligence and their applications in stochastic continuous models.
Accepted on Scientific board on: **AS**	03.09.2018.
Defended: **DE**	October 2018.
Thesis Defend board: **DB**	
President:	Zagorka Lozanov-Crvenković PhD, full professor, Faculty of Sience, University of Novi Sad
Member:	Marta Takač PhD, full professor, Učiteljski fakultet na mađarskom nastavnom jeziku
Member:	Mirjana Štrboja PhD, associate professor , Faculty of Sience, University of Novi Sad

Mentor: Ivana Štajner-Papuga PhD, full professor,
Faculty of Sience, University of Novi Sad