UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Darko Stošić

IMPLEMENTACIJA I EVALUACIJA METODA KLASIFIKACIJE ZA VELIKE SKUPOVE PODATAKA

master rad

Beograd, 2019.

Mentor:

dr Aleksandar Kartelj, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Vladimir Filipović, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Mladen Nikolić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Naslov master rada: Implementacija i evaluacija metoda klasifikacije za velike skupove podataka

Rezime: Veliki podaci danas predstavljaju važnu temu kako u akademskim tako i u industrijskim okvirima. *Hadoop* i *Apache Spark* predstavljaju platforme koje se najčešće koriste kao rešenje za probleme vezane uz velike podatke. *Hadoop* pruža servis za rad sa velikim podacima koji je pouzdan, skalabilan i otporan na greške. On se zasniva na HDFS-u, distribuiranom sistemu datoteka, i *MapReduce* platformi koji pruža alat za distribuirana izračunavanja. Kako *MapReduce* platforma nije univerzalno rešenje za sve vrste aplikacija, *Apache Spark* je nastao sa ciljem da prevaziđe ograničenja koja ima *Hadoop*. *Apache Spark* pruža čuvanje podataka unutar memorije sa ciljem njihove ponovne upotrebe unutar aplikacije, međutim, ta prednost *Apache Spark* je ujedno i mana jer aplikacije postaju ograničene dostupnom memorijom. U radu su realizovane poznate klasifikacione metode, logistička regresija i veštačka neuronska mreža sa propagacijom unazad, koristeći *Hadoop* i *Apache Spark* platforme. Takođe je u svrhu poređenja razvijena i sekvencijalna implementacija pomenutih klasifikacionih metoda. U radu je razmatrano kvalitativno poređenja kao i poređenje performansi razvijenih rešenja.

Ključne reči: veliki podaci, mašinsko učenje, logistička regresija, veštačke neuronske mreže, Hadoop, Spark

Sadržaj

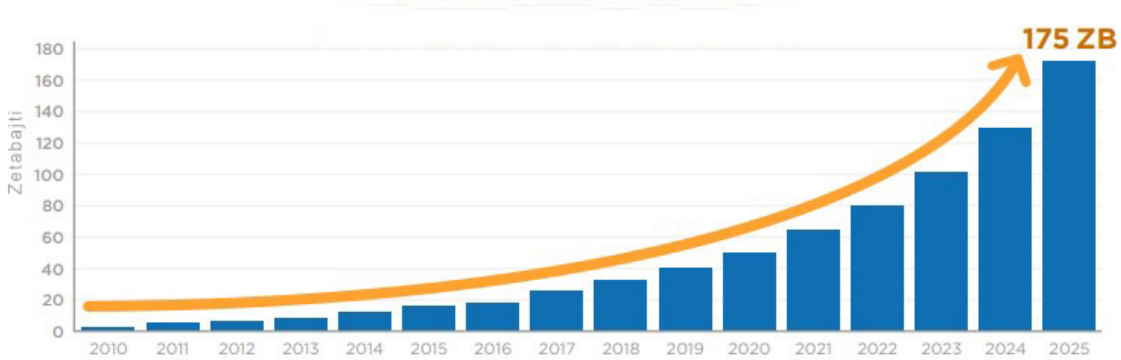
Glava 1

Uvod

1.1 Veliki podaci

Količina novih podataka svakodnevno raste neverovatnom brzinom, zahvaljujući pojavi novih tehnoloških trendova, poput interneta stvari (eng. Internet of things), socijalnih mreža, mrežnog računarstva, itd.

Prema izveštaju IDC (eng. International Data Corporation) [5], jednog od najuticajnijih lidera u oblasti velikih podataka, 2011. godine, količina kreiranih i kopiranih podataka u svetu dostigla je 1.8 ZB (približno $10^{21}B$). Danas, po najnovijim izveštaju IDC, količina podataka dostigla je broj od 33 ZB, a njihova očekivanja su da će do 2025 taj broj porasti do 175 ZB [6].



Slika 1.1: Godišnji prikaz globalnog rasta količine podataka[1]

[1] https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

Pojam veliki podaci (eng. Big data) se uglavnom koristi da opiše enormne skupove podataka. U poređenju sa tradicionalnim skupovima podataka, skupovi velikih podataka sadrže veliku količinu nestruktuiranih podataka. Veliki podaci pružaju priliku za otkrivanje novih vrednosti kao i njihovog skrivenog značenja. Takođe donose i nove izazove, npr. kako efektivno organizovati i upravljati takvim skupovima podataka.

Mnoge industrije su iskazale svoje interesovanje u potencijal velikih podataka, državne agencije uveliko najavljuju planove za ubrzavanje istraživanja skupova velikih podataka i njihovih primena. Veliki javni mediji takođe sve više objavljuju članke vezane za velike podatke.

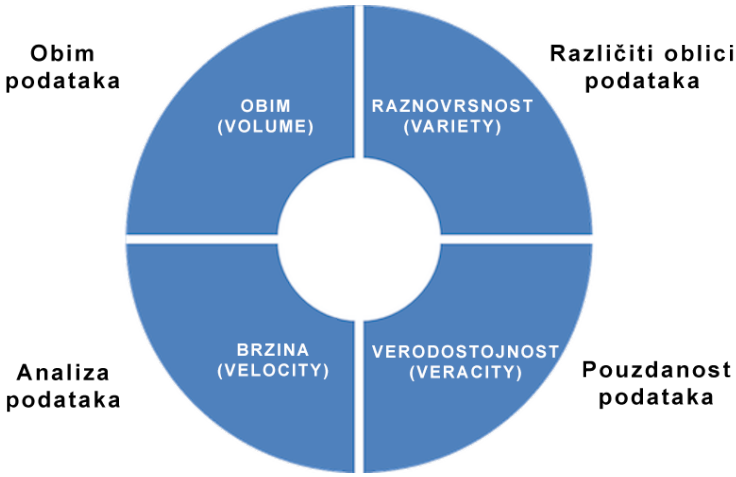
Definicije i karakteristike velikih podataka

Izraz veliki podaci je apstraktan. Osim masovnog broja podataka, poseduje i neke druge karakteristike, koje određuju razliku između samog izraza i izraza masivni podaci ili veoma veliki podaci. Danas, iako je značaj velikih podataka prepoznat, i dalje postoje sukobi mišljenja vezani za pitanje definicije istih. Opšta definicija bi bila da veliki podaci predstavljaju skupove podataka koji se ne mogu uočiti, nabaviti i obrađivati tradicionalnim IT i softverskim/hardverskim alatom u prihvatljivom vremenu.

Zapravo, veliki podaci su definisani još 2001. god. Dag Lanei (eng. Doug Lanei) [8], analitičar META, u istraživačkom izveštaju, definisao je izazove i mogućnosti koje donose skupovi velikih podataka modelom tri slova V, odnosno povećanje obima, brzine, i raznovrsnosti (eng. volume, velocity, variety). U modelu tri slova V, obim(eng. volume) znači, sa stvaranjem i prikupljanjem mase podataka, količina podataka postaje sve veća; brzina(eng. velocity) označava pravovremenost velikih podataka, odnosno, prikupljanje, analiza podataka itd. mora se brzo i blagovremeno sprovoditi kako bi se maksimalno iskoristila komercijalna vrednost velikih podataka; Raznovrsnost(eng. variety) ukazuje na različite vrste podataka, polustruktuirane i nestruktuirane(audio, video, veb stranica i tekst) i tradicionalne struktuirane podatke.

IDC je 2011. god. u izveštaju definisao velike podatke na sledeći način: "Tehnologije velikih podataka opisuju novu generaciju tehnologija i arhitektura, dizajniranih da ekonomski izvlače vrednost iz veoma velikih količina raznovrsnih

[2] https://www.nextview.nl/sap-vora-1-4-the-big-data-picture/



Slika 1.2: Četiri V velikih podataka[2]

podataka, omogućavajući hvatanje, otkrivanje i / ili njihovo analiziranje". Ovom definicijom, karakteristike skupova velikih podataka mogu se sumirati kao četiri V(eng. volume, variety, velocity, veracity) , tj. obim, raznovrsnost, brzina i verodostojnost, kao što je prikazano na slici 1.2.

Vrednost velikih podataka

Tokom pandemije gripa 2009. god., *Google* je dobio blagovremenu informaciju analizom velikih podataka, koji su čak pružili i vredniju informaciju od onih koje su obezbedili centri za prevenciju bolesti. Skoro sve zemlje zahtevaju da bolnice informišu agencije kao što su centri za prevenciju bolesti u slučaju nove vrste gripa. Međutim, pacijenti obično nisu išli kod lekara odmah nakon zaraze. Takođe je trebalo dosta vremena da se informacija pošalje iz bolnica ka centrima za prevenciju bolesti, a dalje da centri za prevenciju bolesti analiziraju i sumiraju takvu informacija. Stoga, kada javnost postane svesna pandemije nove vrste gripa, bolest se dotad možda širila jednu do dve nedelje. *Google* je otkrio da prilikom širenja gripa unosi koji se često traže u pretraživačima razlikuju od onih u uobičajenim vremenima, a frekvencije korišćenja tih unosa bile su u korelaciji sa širenjem gripa u datom vremenu i mestu. *Google* je pronašao 45 ulaznih grupa za pretraživanje koji su bili relevantni za izbijanje gripa i ubacili su ih u specifične matematičke modele kako bi prognozirali širenje gripa i predvideli mesta na kojima se grip širi.

2008. godine, *Microsoft* je kupio *Farecast*, kompaniju iz Sjedinjenih Američkih Država, koja se bavi naučnim istraživanjima. *Farecast* ima sistem za prognozu cena

avio karata, on predviđa trend rasta/opadanja cena karata. Sistem je ugrađen u Bing pretraživač kompanije *Microsoft*. Do 2012. god. sistem je sačuvao skoro 50 USD po karti po putniku, sa predviđenom tačnošću do 75%.

Danas su podaci postali važan faktor u proizvodnji koji može biti uporediv sa materijalnom imovinom i ljudskim kapitalom. Kako se razvijaju multimedija, društveni mediji i IoT(eng. Internet of Things), preduzeća će sakupljati sve više informacija, što će dovesti do eksponencijalnog rasta obima podataka. Veliki podaci će imati sve veći potencijal u kreiranju vrednosti za preduzeća i potrošače.

1.2 Mašinsko učenje

Mašinsko učenje je oblast veštačke inteligencije koja omogućava računarima da uče bez eksplicitnog programiranja. Mašinsko učenje vrši generalizaciju znanja bazirano na prethodnim iskustvima (podataka o pojavama/entitetima koji su predmet učenja). Na osnovu dobijenog znanja daju se odgovori na pitanja za entitete/pojave koji nisu ranije viđeni. Tom Mičel (eng. Tom Mitchell), u svojoj knjizi *Machine Learning* [11], je pružio kratku definiju mašinskog učenja.

Za računarski program se kaže da uči iz iskustva E, vezanog za zadatak T i meru performansi P, ukoliko se njegove performanse na zadatku T, merene metrikama P, unapređuju sa iskustvom E.

Ono u čemu su velike prednosti mašinskog učenja je u tome što je vrlo teško algoritamski opisati neke vrste zadataka koje ljudi lako rešavaju. Uz pomoć mašinskog učenja ne moraju da se implemetiraju sami algoritmi, već treba obezbediti dobar skup podataka na osnovu kojih će algoritam sam da moći da izvuče konkretan zaključak. Kao primer možemo navesti neke jako intuitivne stvari koje čovek svakodnevno rešava kao što su prepoznavanje lica ili prepoznavanje govora. Sa druge strane, neki problemi koje bismo i mogli algoritamski definisati, se mnogo sporije izvršavaju nego primenom mašinskog učenja, ili zahtevaju velike baze znanja.

U [3] je napravljen pregled nekih problema u kojima je mašinsko učenje uspešno primenjeno. To su prepoznavanje lica na slikama, prepoznavanje različitih objekata na slikama i videu, prepoznavanje tumora na medicinskim snimcima, autonomna vožnja automobila, autonomno letenje, igranje igara na tabli, računarskih igara kao što su Super Mario ili Doom, klasifikacija teksta, mašinsko prevodenje, automatsko opisivanje sadržaja slika, analiza osećanja izraženih u tekstu, predvi-

danje razvoja bolesti kod pacijenata i preporučivanje terapije, analiza društvenih mreža, prepoznavanje i sinteza govora, itd.

Tokom godina se ova oblast dosta razvijala, što sa teoretske strane, što sa strane praktične primene, i sam razvoj je išao u više različitih pravaca tako da je vremenom nastala potreba da se izvrši neka dublja podela i specifikacija samih algoritama. Kako se javila bitna razlika između dve grupe algoritama, tako je i na njima zasnovana i osnovna podela algoritama mašinskog učenja.

1. Nadgledano učenje (eng. Supervised Learning) – predstavlja pristup problemu učenja koji se odnosi na situacije u kojima se algoritmu zajedno sa podacima iz kojih uči daju i željeni ulazi.

2. Nenadgledano učenje (eng. Unsupervised Learning) – predstavlja pristup problemu učenja koji se odnosi na situaciju kada se algoritmu koji uči pružaju samo podaci bez izlaza, a od algoritma koji uči očekuje se da sam uoči neke zakonitosti u podacima koji su mu dati.

Kao primer nadgledanog učenja, navodi se pronalaženje rešenja problema određivanja cene nekretnina na osnovu ulaznih podataka koje se sastoje iz konkretnih primera nekretnina sa određenim osobinama i plaćene cene same nekretnine. Još jedan primer je da doktor može korišćenjem podataka o malignim oboljenjima pacijenata da odredi da li neki novi pacijent ima maligni ili benigni tumor. Primer nenadgledanog učenja je takozvano klasterovanje odnosno uočavanje grupa sličnih objekata kada se ne zna unapred koliko grupa postoji i koje su njihove karakteristike.

Klasifikacija

Vodeći se primerom za tumor, cilj je modelovati funkciju koja na osnovu ulaznih podataka o pacijentu i karakteristikama tumora određuje izlaznu vrednost DA u slucaju da je tumor maligni ili NE u slucaju da nije. Povratne vrednosti DA i NE mogu da se zamene sa vrednostima 1 i 0, respektivno. Drugi naziv za klasu obeleženu sa 1 je **pozitivna klasa**, a za klasu obeleženu sa 0 je **negativna klasa**. Ovaj način klasifikacije se drugačije naziva binarna klasifikacija. U slučaju kada bi trebalo da se izvrši klasifikacija na više klasa onda bi oznake klasa uzimale 0, 1, 2... n, gde n predstavlja broj klasa.

Treba utvrditi notaciju koja će biti korišćena u nastavku rada. Funkcija koja vrši predviđanje se naziva **funkcija predviđanja** i u slučaju binarne klasifikacije ona će vraćati vrednost iz intervala [0, 1]. Donošenje konkretne odluke o klasi {0, 1} se kasnije može doneti diskretizacijom na osnovu praga (eng. treshold). Koristićemo $\mathbf{x}^{(i)}$ da obeležimo vektor atributa $x_1, x_2, x_3...x_p$ gde je p broj atributa[3]. Izlaznu promenljivu ćemo obeležiti sa $y^{(i)}$. Par $(\mathbf{x}^{(i)}, y^{(i)})$ se naziva primer dok se skup podataka koji se koristi za učenje funkcije predviđanja sastoji od m trening primera $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, ..., m\}$.

1.3 Razmatrani algoritmi

Logistička regresija

Logistička regresija je jedan od osnovnih klasifikacionih algoritama koriste. Ne treba da buni to što kao deo naziva stoji regresija jer je on nastao iz istorijskih razloga.

U ovom radu, za svrhe logističke regresije, funkciju predviđanja definišemo kao:

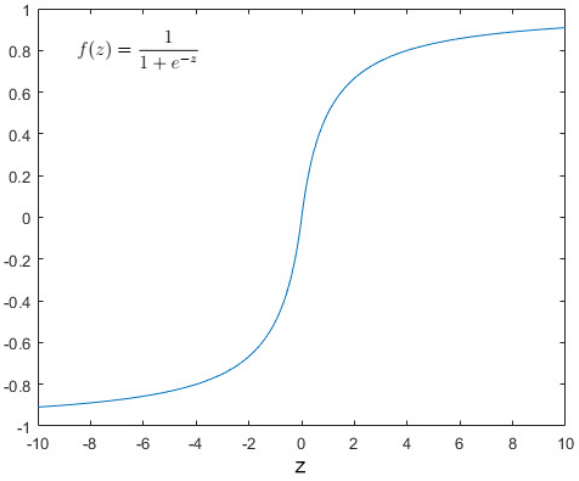
$$f_\theta(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} \tag{1.1}$$

Radi jednostavnosti, možemo uvesti smenu $z = \boldsymbol{\theta}^T \boldsymbol{x}$, odakle sledi:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{1.2}$$

Ova funkcija se zove sigmoidna funkcija. Vektor $\boldsymbol{\theta}$ se sastoji od parametara $\theta_1, \theta_2, \theta_3...\theta_p$ i predstavlja vektor koeficijenata kojim parametrizujemo funkciju predviđanja. Može se primetiti da funkcija 1.1 ima dve asimptote. Ako $\boldsymbol{x} \to +\infty$ onda $e^{-\boldsymbol{x}} \to 0$ pa $f_\theta \to 1$. Ako pak $\boldsymbol{x} \to -\infty$ onda $f_\theta \to 0$. Vidimo da sama funkcija zadovoljava početno ograničenje $0 < f_\theta(\boldsymbol{x}) < 1$.

[3]Superskript (i) u notaciji predstavlja indeks u skup podataka a ne eksponent.



Slika 1.3: Sigmoidna funkcija

Postavlja se pitanje kako interpretirati izlaz funkcije predviđanja. Na primer ako sama funkcija predviđanja za ulazni vektor vrati vrednost 0.7 to onda može da se interpretira kao da postoji verovatnoća od 70% da se korisniku svidi dati sadržaj. Na osnovu vrednosti funkcije predviđanja može se formirati funkcija dodeljivanja klase tako da se vraća 1 ako logistička funkcija vrati vrednost veću ili jednaku 0.5, a u suprotnom vraćamo predviđanje 0.

Sa slike 1.3 može se zaključiti da važi $f_\theta(\boldsymbol{x}) \geq 0.5$ ako je $\boldsymbol{\theta}^T\boldsymbol{x} \geq 0$ tj. funkcija dodeljivanja klase kao povratnu vrednost vraća predviđanje 1. U suprotnom slučaju ako je $\boldsymbol{\theta}^T\boldsymbol{x} < 0$ za funkciju predviđanja važi $f_\theta(\boldsymbol{x}) < 0.5$ tj. funkcija dodeljivanja klase kao povratnu vrednost vraća predviđanje 0.

Funkcija koštanja

Prikazano je da se za jedan isti skup podataka mogu definisati više logističkih funkcija $f_\theta(\boldsymbol{x})$ koje su parametrizovane vektorom $\boldsymbol{\theta}$. Međutim do sada nije diskutovano kako izračunati optimalan vektor $\boldsymbol{\theta}$. Ideja je da se odredi vektor $\boldsymbol{\theta}$, takav da funkcija $f_\theta(\boldsymbol{x})$ dobro aproksimira skup zadatih izlaznih vrednosti (klasa). U tu svrhu definiše se funkcija koštanja (eng. cost function) koja ima oblik:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(f_\theta(\boldsymbol{x}^{(i)})) + (1-y^{(i)})log(1-f_\theta(\boldsymbol{x}^{(i)}))) \tag{1.3}$$

Ova funkcija $J(\boldsymbol{\theta})$ zavisi od vektora $\boldsymbol{\theta}$, što je i logično jer se sama funkcija koštanja određuje na osnovu parametara funkcije predviđanja. Posmatranjem

funkcije može se zaključiti zašto ima ovakav oblik. Vidi se da funkcija sumira parcijalne greške za svaki pojedinačni podatak iz trening podataka i na kraju ih usrednjava. $y^{(i)}$ može imati samo vrednosti 1 ili 0 što znači da samo jedan od dva sabiraka može imati vrednost različitu od 0 za jedan podatak iz skupa trening podataka. Dakle, ako je za dati podatak iz trening skupa tumor malignan ($y^{(i)} = 1$), a logistička funkcija vrati vrednost koja je približno jednaka 1, funkcija koštanja će biti približno jednaka 0. Ako je logistička funkcija pogrešila i vratila vrednost koja je približna nuli, funkcija koštanja će imati jako veliku vrednost. Time se postiže upravo ono što je željeno, a to je da kada logistička funkcija vrati tačnu vrednost, da se ne kažnjava, a ukoliko vrati pogrešnu vrednost da se kažnjava velikom cenom.

Algoritam opadajućeg gradijenta

Pošto je formirana funkcija koštanja $J(\boldsymbol{\theta})$, sledeći je zadatak odrediti parametre $\boldsymbol{\theta}$ za koje funkcija ima minimalnu vrednost. Dakle zadatak se svodi na pronalaženje takvog vektora parametara $\boldsymbol{\theta}^*$ za koji je $J(\boldsymbol{\theta}^*)$ minimalno.

U svrhu minimizacije koristiće se algoritam opadajućeg gradijenta koji se može formulisati pseudo kodom na slici 1.4

Algoritam počinje tako što se vektor $\boldsymbol{\theta}$ inicijalizuje nekom slučajnom vrednošću oko nule i na osnovu nje početna vrednost funkcije koštanja. Dalje se u svakoj iteraciji određuju nove vrednosti vektora $\boldsymbol{\theta}$ po definisanom algoritmu, na osnovu parcijalnih izvoda funkcije koštanja, pomnoženom sa korakom α. Istovremeno se u svakoj iteraciji izračunava funkcija koštanja i onda se na osnovu toga određuje da li je sam algoritam konvergirao.

Postavlja se pitanje kada ovaj algoritam prestaje sa radom, odnosno šta se može smatrati indikatorom da je algoritam konvergirao. Dakle, treba definisati vrednost ε, koja predstavlja razliku vrednosti funcije koštanja u dve susedne iteracije.

Još jedna kritična vrednost za algoritam je vrednost α, koja predstavlja stopu učenja (eng. learning rate). Ova vrednost se može menjati tokom rada algoritma ili ostaviti kao konstanta. Za potrebe ovog algoritma pokazuje se da je dovoljno da bude konstantna. Međutim, ako je vrednost izabrana za ovaj član previše mala algoritmu može trebati dosta vremena da konvergira, a ukoliko je stopa učenja previše velika, algoritam može da divergira.

Kako je algoritam opadajućeg gradijenta definisan, može se odrediti vrednost vektora parametara $\boldsymbol{\theta}$, iz čega se jednoznačno dobija funkcija predviđanja.

Ulaz: Matrica $\boldsymbol{x}$, Vektor $\boldsymbol{y}$, stopa učenja α, uslov konvergencije ε
Izlaz: Optimizovan vektor $\boldsymbol{\theta}$

```
1  m = prebrojTreningPrimere(y);
2  p = prebrojAtribute(x);
3  θ = inicijalizujVektor();
4  J = izračunajFunkcijuKoštanja(θ, x, y);
5  uslov = +∞;
6  while uslov > ε do
7  |  for i = 1 to m do
8  |  |  for j = 1 to p do
9  |  |  |  temp = θ[j] − (α/m) ∗ (sigmoidnaFunkcija(θ, x[i]) − y[i]) ∗ x[i][j]
10 |  |  |  θ[j] = temp;
11 |  |  end
12 |  end
13 |  J' = J;
14 |  J = izračunajFunkcijuKoštanja(θ, x, y);
15 |  uslov = J' − J;
16 end
```

Slika 1.4: Opadajući gradijent

Problemi funkcije odlučivanja

U nekim situacijama logistička regresija može proizvesti liniju razgraničavanja koja nedovoljno dobro da razgraničava podatke (eng. underfitting). Međutim, ako se uvede nelinearni model koji koristi nelinearne funkcije, može se javiti još jedan tip problema koji ovde nismo razmatrali a to je da sama funkcija predviđanja odlično razdvaja podatke linijom razgraničenja nad trening skupom podataka ali da se loše ponaša nad novim podacima tj. podacima koji nisu korišćeni u fazi učenja (engl. overfitting).

Da bi se formirao klasifikacioni model koji proizvodi dobru liniju razgraničavanja, mora se blago izmeniti algoritam za izračunavanje cene koštanja. Da bi se uzela u obzir nelinearnost potrebno je cenu koštanja prilagoditi na sledeći način:

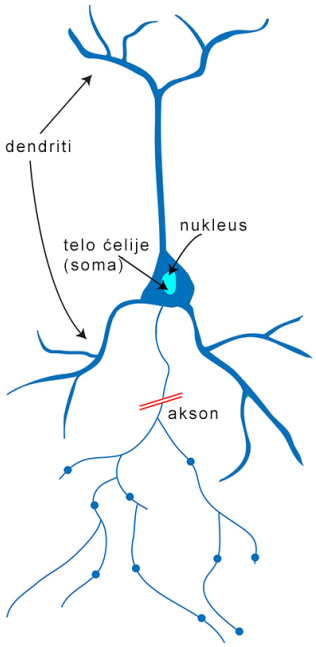
$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(f_\theta(\boldsymbol{x}^{(i)})) + (1-y^{(i)})log(1-f_\theta(\boldsymbol{x}^{(i)}))) + \frac{\lambda}{2m}\sum_{j=1}^{m}\boldsymbol{\theta}_j^2 \quad (1.4)$$

Parametar λ se naziva **regularizacioni parametar**. Izbor regularizacionog parametra je važan u procesu formiranja dobrog klasifikacionog modela.

Veštačke neuronske mreže

Veštačke neuronske mreže predstavljaju računarski model inspirisan biološkim neuronskim mrežama. Njihov cilj je replikacija ljudskog načina učenja. Biološka neuronska mreža se sastoji od skupa neurona međusobno povezanih sinapsama. Analogno biološkoj, veštačka neuronska mreža se sastoji od skupa međusobno povezanih veštačkih neurona. Svaka veza, kao i sinapse u biološkom mozgu, može preneti signal iz jednog veštačkog neurona u drugi. Veštački neuron koji primi signal može ga obraditi i signalizirati veštačke neurone koji su povezani sa njim.

Neuroni su osnovne jedinice mozga i nervnog sistema, ćelije odgovorne za primanje senzornih podataka iz spoljnog sveta, za slanje motornih naredbi našim mišićima, za transformaciju i prenošenje električnih signala za svaki međukorak. Neuron ima tri glavna dela: dendrit, akson i telo ćelije (soma), oni se mogu predstaviti kao grane, koreni i stablo drveta, kao što se može videti na slici 1.5. Dendrit (grana drveta) je mesto gde neuron prima ulaz od drugih ćelija. Akson (koren stabla) je izlazna struktura neurona; kada neuron želi da razgovara sa drugim neuronom, on šalje električnu poruku koja se zove akcioni potencijal kroz čitav akson.



Slika 1.5: Struktura neurona[4]

[4] https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron

Veštački neuron je matematička funkcija zamišljena kao model bioloških neurona i predstavlja osnovnu jedinicu veštačke neuronske mreže. Veštački neuron prima jedan ili više ulaznih podataka preko dendrita, sumira ih unutar svog tela i taj izlaz šalje dalje putem aksona. Svaki ulazni podatak ima svoju težinu i oni se šalju ka nelinearnoj funkciji koju ćemo zvati aktivaciona funkcija. Primenom aktivacione funkcije donosi se odluka da li će neuron proslediti podatak dalje (poput prosleđivanja električnih impulsa unutar biološkog neurona).

Arhitektura veštačke neuronske mreže prestavlja specifično povezivanje neurona u jednu celinu. Struktura veštačke neuronske mreže se razlikuje po broju slojeva. Prvi sloj se naziva ulazni, a poslednji izlazni, dok se slojevi između nazivaju skriveni slojevi. Prvi sloj je ulazni sloj koji prima podatke iz spoljašnje sredine, skriveni slojevi prosleđuju relevantne podatke do trećeg izlaznog sloja. Na izlazu izlaznog sloja dobijaju se konačni rezultati. Složenije neuronske mreže imaju više složenih slojeva. Slojevi su međusobno potpuno povezani.

Slojevi komuniciraju tako što se izlaz svakog neurona iz prethodnog sloja poveže na ulaz svih neurona narednog sloja. Dakle, svaki sloj ima nekoliko ulaza i jedan izlaz. Jačina veza kojom su povezani neuroni naziva se težinski faktor.

Postavlja se pitanje koja se aktivaciona funkcija izračunava u svakom čvoru. Ta funkcija varira od konkretne primene, odnosno od toga koji tip problema se rešava. Za primene u posmatranom domenu problema, obzirom da se rešava problem klasifikacije logično je da se koristi neki od klasifikacionih algoritama. Izabrana je sigmoidna funkcija koja je korišćena i u okviru logističke regresije.

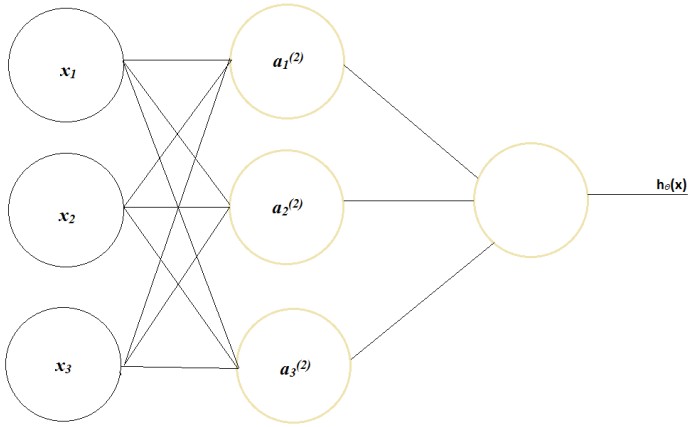
Drugo implementaciono pitanje o samoj strukturi neuronske mreže je broj izlaznih čvorova koje će sama mreža imati. Kada se rešava klasifikacija na dve klase (klasa DA i klasa NE) radi jednostavnosti koristićemo samo jedan izlazni čvor čiju ćemo vrednost interpretirati na sličan način kao i u slučaju logističke funkcije.

Rečeno je da može postojati više skrivenih slojeva. Što više skrivenih slojeva mreža ima, to će ona biti u mogućnosti da nauči složeniju funkciju predviđanja. Međutim samo izračunavanje ovakvog algoritma se dodatno usložnjava dodavanjem svakog dodatnog sloja.

Još jedna stvar jako bitna za implementaciju algoritma jesu slobodni članovi (eng. Bias unit). Dodata je još jedna promenljiva u ulazni vektor $\boldsymbol{x}$ koja uvek ima vrednost 1, te jedinicu imamo i u skrivenim slojevima koja uvek na izlazu daje vrednost 1, bez obzira na ulazni vektor.

Sada kada je definisana veštačka neuronska mreža na logičkom nivou može se

pričati o njenom predstavljanju pomoću matematičkog modela i objasniti model implementacije. Polazi se od jednog jednostavnog modela neuronske mreže date na slici 1.6.



Slika 1.6: Primer veštačke neuronske mreže

Cilj treniranja veštačke neuronske mreže je podešavanje njenih unutrašnjih težina tako da se za date ulazne vrednosti na izlazu dobiju očekivane izlazne vrednosti. Težine su dodeljene vezama između neurona u susednim slojevima, u daljem tekstu će biti označene nizom matrica $\boldsymbol{\theta}^{(j)}$, $1 \leq j \leq l$, gde je l broj slojeva. Ako se obeleži broj čvorova na sloju j sa s_j tada se dimenzija svake matrice $\boldsymbol{\theta}^{(j)}$ može predstaviti kao $s_{j+1} \cdot (s_j + 1)$, gde se zbog slobodnog člana koji pripada sloju j dodaje 1 za kolone. Vrednost neurona se izračunava kao skalarni proizvod svih vrednosti neurona iz prethodnog sloja i težina pridruženih posmatranom neuronu odnosno vrednosti neurona i njihovih težina se sa prethodnog sloja **propagiraju unapred** ka sledećem sloju.

Funkcija koštanja

Slično kao kod logističke regresije tako se i ovde može definisati funkcija koštanja. Ona se kod neuronskih mreža definiše na sledeći način:

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K}(\boldsymbol{y}_k^{(i)} log(f_\theta(\boldsymbol{x}^{(i)}))_k + (1 - \boldsymbol{y}_k^{(i)}) log(1 - f_\theta(\boldsymbol{x}^{(i)}))_k)] + R \quad (1.5)$$

$$R = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\boldsymbol{\theta}_{(ji)}^{(l)})^2 \tag{1.6}$$

Konstanta K u ovoj jednačini predstavlja broj izlaznih čvorova u veštačkoj neuronskoj mreži. Ova unutrašnja suma je tu, jer se u ovom slučaju $\boldsymbol{y}$ predstavlja kao matrica gde svaki red predstavlja vektor koji ima sve vrednosti 0 i samo u jednoj koloni vrednost 1 i to u onoj koloni, čijoj klasi pripada. Nakon toga se sumiraju greške koje se javljaju u svakom od izlaznih čvorova. Drugi sabirak definisan sa 1.6, u ovoj sumi predstavlja regularizacioni parametar, i razlika je samo što nam je u logističkoj regresiji $\boldsymbol{\theta}$ bio vektor, dok sada predstavlja niz matrica. Tako da se sada sumiraju matrice $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \boldsymbol{\theta}^{(3)} ... \boldsymbol{\theta}^{(L-1)}$ gde L predstavlja broj slojeva u veštačkoj neuronskoj mreži. Greške dobijene na izlaznom sloju opisane sa 1.5 se **propagiraju unazad** i na taj način se ažuriraju težine između uzastopnih slojeva.

Glava 2

Platforme za rad sa velikim podacima

2.1 Hadoop i MapReduce

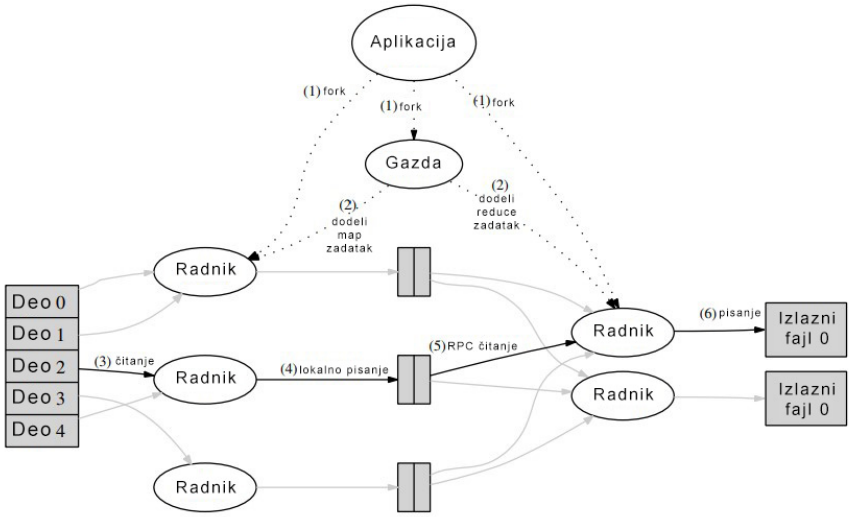
Hadoop je kreirao Dag Kating (eng. Doug Cutting), tvorac *Apache Lucene*, popularne biblioteke za pretraživanje teksta. Hadoop svoje poreklo vodi iz *Apache Nutch*, pretraživača otvorenog kôda, koji je deo *Apache Lucene* projekta.

Apache Nutch je započet 2002. godine, međutim, njegovi tvorci su shvatili da trenutna arhitektura neće dobro skalirati sa rastućim zahtevima interneta. 2004. godine, programeri *Apache Nutch* projekta, počeli su da pišu implementaciju otvorenog kôda za distribuiran sistem datoteka, NDFS (eng. Nutch distributed filesystem). Naredne godine, tvorci *Apache Nutch* projekta su imali radnu implementaciju *MapReduce* platforme, a do sredine te godine su svi glavni algoritmi za *Apache Nutch* prerađeni da koriste *MapReduce* i NDFS. U januaru, 2008. godine, *Hadoop* je postao projekat najvišeg prioriteta za *Apache*, potvrđujući svoj uspeh i njegovu raznoliku, aktivnu zajednicu.

MapReduce je platforma za obradu i generisanje velikih podataka. Tipična aplikacija zasnovana na *MapReduce* platformi obrađuje veliku količinu podataka na više mašina. *MapReduce* deli podatke u nezavisne celine. Funkciju mapiranja (u daljem tekstu *map*) određuje korisnik, ona procesira <ključ, vrednost> (eng. <key-value>) par. Funkciju redukovanja (u daljem tekstu *reduce*) takođe određuje korisnik, ona obrađuje sve međuvrednosti povezane zajedničkim ključem.

Map i *reduce* su dva različita koraka. Svaki korak se vrši paralelno nad skupovima <**ključ, vrednost**> parova. Na taj način programi su podeljeni u *map* i

reduce faze, razdvojene prenosom podataka između čvorova u klasteru. *Map* faza preuzima funkciju i deo podataka kao ulaz, primenjuje funkciju na svaku vrednost iz ulaza i generiše nove vrednosti za izlaz. Izlaz iz *map* faze je skup vrednosti u formi <ključ, vrednost> parova. *MapReduce* platforma onda uređuje izlaz iz *map* funkcija i pruža ih *reduce* fazi kao ulaz. Vrednosti su agregirane na čvoru koji izvršava *reduce* fazu za dat ključ. *Reduce* faza proizvodi još jedan skup <**ključ, vrednost**> parova kao konačan izlaz. *Reduce* faza počinje samo kada se svi podaci iz *map* faze prenesu do odgovarajuće mašine. *MapReduce* platforma zahteva ulaz u obliku <**ključ, vrednost**> koji se može serijalizovati, zato je ograničen na zadatke i algoritme koji koriste <**ključ, vrednost**> parove.



Slika 2.1: Pregled izvršavanja aplikacije na *MapReduce* platformi[1]

Slika 2.1 prikazuje tok izvršavanja *MapReduce* aplikacije. Kada se pokrene aplikacija na *MapReduce* platformi, sledeća sekvenca koraka se odigrava (oznake na slici 2.1 odgovaraju brojevima u listi ispod):

1. *MapReduce* okvir prvo podeli ulazne podatke na M delova nakon čega pokreće više kopija aplikacije na klasteru.

2. Jedna kopija je specijalna, ona predstavlja gazdu. Ostale kopije su radnici, one dobijaju zadatke od gazde. Postoji M *map* zadataka i R *reduce* zadataka

[1] `https://stackoverflow.com/a/26993598`

za dodelu. Gazda bira nezaposlene radnike i dodeljuje im *map* ili *reduce* zadatak.

3. Radnik kojem je dodeljen map zadatak čita sadržaj dodeljenog dela ulaznih podataka. Parsira par <ključ, vrednost> iz ulaznih podataka i prosleđuje taj par map funkciji. Međurezultat, dobijen iz *map* funkcije, u vidu novih <ključ, vrednost> parova, čuva se u memoriji.

4. Periodično, sačuvani parovi se pišu na lokalni disk, gde se particionišu pomoću funkcije particionisanja u R regija. Lokacija ovih sačuvanih parova na lokalnom disku se šalju ka gazdi, on je odgovoran za prosleđivanje lokacije radnicima zaduženim za *reduce* zadatke.

5. Kada radnik zadužen za *reduce* zadatak bude obavešten od gazde o ovim lokacijama, koristi RPC (eng. Remote procedure calls) da bi pročitao sačuvane podatke sa lokalnih diskova radnika zaduženih za map zadatke. Kada je radnik zadužen za *reduce* zadatak pročitao sve međurezultate, sortira ih po ključu, tako da svi parovi sa istim ključem budu grupisani zajedno.

6. Radnici zaduženi za *reduce* zadatak prolaze kroz sortirane međurezultate i za svaki jedinstven ključ na koji naiđu, šalju ključ i skup vrednosti vezan za njega ka *reduce* funkciji.

Nakon uspešnog završetka *MapReduce* aplikacije, njen izlazni sadržaj dostupan je u R izlaznih datoteka (jedan po *reduce* zadatku).

2.2 Apache Spark

Apache Spark (u daljem tekstu *Spark*) je projekat otvorenog kôda koji je izgrađen i održavan od raznolike zajednice programera. *Spark* je započeo 2009. godine kao istraživački projekat u laboratoriji *UC Berkeley RAD Lab*, koja je kasnije postala *AMPLab*. Istraživači u laboratoriji ranije su radili na *Hadoop* platformi i primetili su da je *MapReduce* bio neefikasan za interaktivne upite i iterativne algoritme. Istraživački radovi su objavljeni o *Spark*-u na akademskim konferencijama i ubrzo nakon njegovog stvaranja 2009. godine, već je bio 10 do 20 hiljada puta brži od *MapReduce* okvira za određene zadatke.

Spark je platforma za rad na klasteru dizajnirana da bude brza i sveobuhvatna. Brzina je važna za obradu velikih podataka, jer predstavlja razliku između

istraživanja podataka interaktivno i čekanja minuta ili sati. Jedna od glavnih karakteristika *Spark*-ove brzine je mogućnost pokretanja računanja u memoriji. *Spark* je takođe efikasniji od *MapReduce* platforme za složene aplikacije koje se pokreću na disku. Što se sveobuhvatnost tiče, *Spark* je dizajniran da pokrije širok opseg zadataka koji su prethodno zahtevali odvojene distribuirane sisteme.

Ključna abstrakcija u *Spark*-u su RDD (eng. Resilient distributed database, u daljem tekstu *RDD*) objekti, oni predstavljaju kolekciju objekata otpornih na greške koji su podeljeni na klasteru i kojima se može paralelno manipulisati. Korisnici stvaraju *RDD*-ove primenom operacija pod nazivom „transformacije" (kao što su *map, filter i groupBy*) nad njihovim podacima. *Spark* pruža pristup *RDD*-ovima preko *API*-ja (eng. Application Program Interface) baziranog na funkcionalnom programiranju gde korisnici mogu jednostavno preneti lokalne funkcije za pokretanje u klasteru.

Spark evaluira *RDD*-ove lenjo, omogućavajući mu da pronađe efikasan plan za korisnikova računanja. Konkretno, transformacije vraćaju novi *RDD* objekat koji predstavlja rezultat izračunavanja, ali on se ne stvara odmah. Kada se pozove akcija, operacija nad *RDD*-om koja proizvodi rezultat koji nije novi *RDD*, *Spark* gleda na ceo graf transformacija koji se koriste za stvaranje plana izvršavanja. Na primer, ako je bilo više operacija *filter* ili *map* u nizu, *Spark* ih može uklopiti u jedan prolaz ili, ako zna da su podaci particionisani, može izbeći njihovo pomeranje kroz mrežu za *groupBy*. *Spark* koristi direktne aciklične grafove radi ove optimizacije.

RDD-ovi pružaju eksplicitnu podršku za čuvanje podataka u memoriji radi ponovne upotrebe u novim računanjima. Podrazumevano, *RDD*-ovi su „kratkotrajni" tj. svaki put se izračunavaju kada se koriste za akcije. Međutim, korisnici mogu "sačuvati" odabrane *RDD*-ove u memoriji radi efikasne ponovne upotrebe (ako podaci ne mogu da stanu u memoriju, *Spark* će ih „prosuti" na disk.). Ovo deljenje podataka je glavna razlika između *Spark*a i *MapReduce* platforme. Deljenje podataka pruža veliko ubrzanje, čak i do 100 puta, za interaktivne upite i iterativne algoritme.

Glava 3

Predložene implementacije

3.1 Distribuirana implementacija logističke regresije

Ideja iza paralelizacije logističke regresije zasniva se na paralelizaciji optimizacione funkcije, opadajućeg gradijenta. Posmatrajući algoritam opadajućeg gradijenta, dat jednačinom 3.1 i pseudokodom 1.4, može se uvideti da je on iterativan optimizacioni algoritam:

$$\boldsymbol{\theta_j} := \boldsymbol{\theta_j} - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3.1}$$

Za manje skupove trening podataka iterativni algoritam pruža zadovoljavajuće performanse ali neka je dato milion trening podataka ($m = 1000000$), tada je potrebno proći kroz svih milion da bi se ažurirao vektor $\boldsymbol{\theta}$. Kao pokušaj poboljšanja performansi, algoritam opadajućeg gradijenta se može paralelizovati korišćenjem *MapReduce* i *Spark* platformi.

Algoritam se izvršava na više računara i svaki računar sadrži ista podešavanja početnih vrednosti. Broj trening podataka se deli na više delova koji se raspoređuju po računarima. Na svakom od računara se vrši parcijalno računanje opadajućeg gradijenta u *map* funkciji nad dobijenim delom podataka. Nakon obrade, svaki računar šalje svoj deo obrađenih podataka na centralizovan server, gde se u *reduce* funkciji vrši njihovo sabiranje da bi ažurirali $\boldsymbol{\theta}$ vektor.

Detaljnije ćemo pogledati *map* i *reduce* funkcije, definisane pseudokodovima 3.1 i 3.2.

```
   Ulaz: Ključ K, Iterator kroz trening podatke oblika <x, y>
   Izlaz: Rezultat u obliku para <ključ, vrednost>
 1 parcijalnoTheta = inicijalizujVektor();
 2 foreach vrednost in Iterator do
 3 |   x = vrednost.x;
 4 |   y = vrednost.y;
 5 |   p = prebrojAtribute(x);
 6 |   for j = 1 to p do
 7 |   |   temp = (sigmoidnaFunkcija(θ, x) - y) * x[j];
 8 |   |   parcijalnoTheta[j] = temp;
 9 |   end
10 end
11 Rezultat = <K, parcijalnoTheta>
12 emituj(Rezultat);
```

Slika 3.1: *Map* funkcija opadajućeg gradijenta

Za ulaz, *map* funkcija prima ključ K i iterator kroz trening podatke oblika $<\boldsymbol{x}, y>$. Ključ K predstavlja identifikator dela podataka koji je dodeljen trenutnoj *map* funkciji. Trening podaci se ka *map* funkciji šalju u obliku para $<\boldsymbol{x}, y>$, gde $\boldsymbol{x}$ predstavlja vektor atributa jednog trening podatka a y predstavlja klasu dodeljenu tom trening podatku. Algoritam se sastoji iz dve petlje. Pre ulaska u petlje inicijalizujemo parcijalno $\boldsymbol{\theta}$. Spoljašnja petlja prolazi kroz trening podatke i preuzima vektor $\boldsymbol{x}$, klasu y i broj atributa vektora $\boldsymbol{x}$. Unutrašnja petlja prolazi kroz atribute vektora $\boldsymbol{x}$ i računa parcijalno $\boldsymbol{\theta}$. Sigmoidnoj funkciji se prosleđuje inicijalno $\boldsymbol{\theta}$ koje svaki računar sadrži. Kao izlaz *map* funkcije dalje ka *reduce* funkciji šaljemo rezultat u obliku para <**ključ, vrednost**>, gde **ključ** uzima vrednost K, dok **vrednost** predstavlja vektor parcijalno izračunatog $\boldsymbol{\theta}$ u trenutnoj *map* funkciji.

Reduce funkcija kao ulaz prima iterator parcijalnih $\boldsymbol{\theta}$ vrednosti. Zadatak *reduce* funkcije je da prođe kroz sve parcijalne $\boldsymbol{\theta}$ vrednosti, sumira ih i ažurira staro $\boldsymbol{\theta}$ na osnovu dobijene sume. Kao rezultat se šalje rezultat koji predstavlja ažuriranu $\boldsymbol{\theta}$ vrednost.

Klasifikacionu fazu takođe možemo paralelizovati. Test podaci se kao i u slučaju trening faze, dele na više manjih delova.

U *map* koraku se obavlja klasifikacija delova test podataka. *Map* funkcija 3.3 prolazi kroz test podatke, klasifikuje ih i pravi vektor predviđanja koji sadrži parove vrednosti $<y_{true}, y_{pred}>$. y_{true} predstavlja pravu klasu test podatka dok

Ulaz: Iterator parcijalnih $\boldsymbol{\theta}$ vrednosti
Izlaz: Rezultat kao ažurirana vrednost vektora $\boldsymbol{\theta}$

```
1 temp = inicijalizujVektor();
2 foreach parcijalnoTheta in Iterator do
3 |   temp = temp + parcijalnoTheta;
4 end
5 Rezultat = θ − (α/m) ∗ temp;
6 emituj(Rezultat);
```

Slika 3.2: *Reduce* funkcija opadajućeg gradijenta

Ulaz: Ključ K, Iterator kroz test podatke oblika <$\boldsymbol{x}$, y_{true}>
Izlaz: Rezultat u obliku para <**ključ, vrednost**>

```
1 vektorPredvidjanja = inicijalizujVektor();
2 foreach vrednost in Iterator do
3 |   x = vrednost.x;
4 |   y_true = vrednost.y_true;
5 |   y_pred = logistickaRegresija(x, θ);
6 |   vektorPredvidjanja.dodaj(<y_true, y_pred>);
7 end
8 Rezultat = <K, vektorPredvidjanja>;
9 emituj(Rezultat);
```

Slika 3.3: *Map* funkcija klasifikacione faze

y_{pred} predstavlja predviđenu klasu. Dobijeni vektori predviđanja se prosleđuju ka *reduce* koraku.

Ulaz: Iterator kroz vektore predvidjanja koji sadrže podatke oblika <y_{true}, y_{pred}>
Izlaz: Rezultat statističke analize

```
1 vektorSvihPredvidjanja = inicijalizujVektor()
2 foreach vektor in Iterator do
3 |   vektorSvihPredvidjanja.dodaj(vektor);
4 end
5 Rezultat = statistickaAnaliza(vektorSvihPredvidjanja);
6 emituj(Rezultat);
```

Slika 3.4: *Reduce* funkcija klasifikacione faze

Reduce funkcija prolazi kroz vektore predviđanja i spaja ih u glavni vektor

svih predviđanja dobijenih iz *map* koraka. Nakon toga, nad glavnim vektorom predviđanja se vrše željene statističke analize koje se šalju kao rezultat.

3.2 Distribuirana implementacija veštačke neuronske mreže

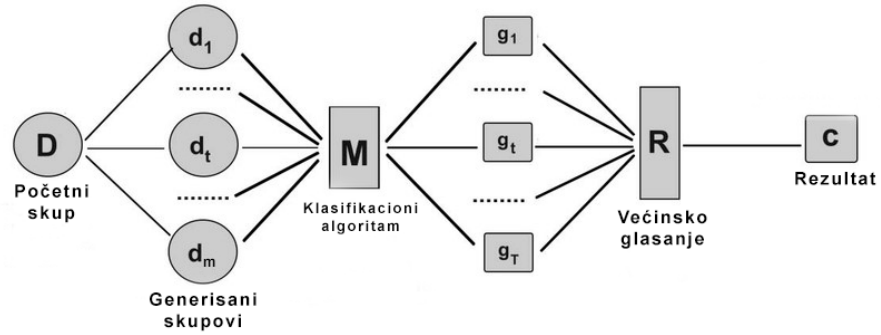
Loša efikasnost neuronskih mreža sa propagacijom u nazad (eng. Backpropagation Neural Networks u daljem tekstu BPNN) pri treningu i klasifikaciji je česta pojava. Ako je količina trening podataka velika, troškovi algoritma pogoršavaju performanse. Slično važi i u slučaju kada je količina podataka za klasifikaciju velika. Verzija neuronske mreže koja je implementirana u ovom radu bavi se poboljšanjem efikasnosti pri treningu i klasifikaciji.

Paralelizacija treninga neuronske mreže

Osnovna ideja za paralelizaciju treninga je razdvajanje podataka. Neka $\boldsymbol{T}$ označava trening podatke, dok je a broj BPNN. Da bi se paralelizovala trening faza $\boldsymbol{T}$ se može razdvojiti u a disjunktnih delova gde je i-ti deo označen sa $\boldsymbol{T}_i$.

Svaki BPNN prihvata jedan deo podataka i započinje trening. Kao rezultat, svaki BPNN postaje istreniran klasifikator koji se može koristiti u budućim klasifikacijama. Na osnovu razdvajanja podataka i BPNN-ova, faza treninga se može ubrzati. Međutim, jednostavno razdvajanje podataka prouzrokuje problem, svaki BPNN je treniran samo delom trening podataka, te manji broj trening podataka može uticati na preciznost klasifikacije. Zbog toga se koristite metode grupisanja (eng. ensemble methods): pakovanje podataka (eng. Bootstrap Aggregating ili Bagging) i većinsko glasanje (eng. Majority Voting). Pomenute metode grupisanja su prikazane na slici 3.5.

Za dati trening skup $\boldsymbol{D}$ veličine n, pakovanjem se generiše m novih trening skupova $\boldsymbol{D}_i$, svaki veličine n', tako što uniformno uzima uzorke iz skupa $\boldsymbol{D}$ sa zamenom. Uzimanjem uzorka sa zamenom, neke obzervacije se mogu ponoviti u svakom skupu $\boldsymbol{D}_i$.



Slika 3.5: Klasifikacija uz pomoć metoda grupisanja, pakovanja podataka i većinskog glasanja [1]

Ulaz: Ključ K, Vektor podataka $\boldsymbol{T}_i$
Izlaz: Istreniran BPNN
1 BPNN = inicijalizujBPNN();
2 BPNN.pokreniTreningFazu($\boldsymbol{T}_i$);
3 emituj(K, BPNN);

Slika 3.6: Map funkcija trening faze veštačke neuronske mreže

Pretpostavimo da postoji a broj *map* funkcija 3.6. Kada počne trening faza, svaka *map* funkcija pravi po jedan BPNN i inicijalizuje težine na nasumičnu vrednost. Nakon toga, i-ti BPNN preuzima deo podataka $\boldsymbol{T}_i$ i započinje prvo propagaciju unapred a zatim i propagaciju unazad. Kada sve map funkcije završe sa radom, dobijamo grupu slabih klasifikatora.

Paralelizacija klasifikacije neuronske mreže

Kao i kod paralelizacije treninga, ideja za paralelizaciju klasifikacije je bazirana na razdvajanju podataka. Neka g označava broj slabih klasifikatora, a $\boldsymbol{C}$ podatke koje treba klasifikovati. Dakle, $\boldsymbol{C}$ može biti podeljen na g disjunktnih delova gde je i-ti deo označen sa $\boldsymbol{C}_i$.

[1] `https://www.researchgate.net/figure/The-Bagging-Bootstrap-Aggregation-scheme_fig2_284156704`

Svaki deo podataka $\boldsymbol{C}_i$ se prosleđuje grupi slabih klasifikatora. Za svaku trening instancu iz $\boldsymbol{C}_i$, vrši se klasifikacija od strane svih klasifikatora iz grupe g. Većinskim glasanjem za svaku instancu određujemo njenu klasu tj. iz vektora dobijenih predviđanja biramo klasu koja se najviše puta ponovila za datu instancu. Podrazumevano, u klasifikacionoj fazi vrši se samo propagacija unapred.

Pretpostavimo da postoji g *map* funkcija 3.7, gde g predstavlja broj klasifikatora dobijenih iz trening faze. Kada počne klasifikaciona faza, u svakoj *map* funkciji vrši se klasifikacija nad delom podataka $\boldsymbol{C}_i$ od strane svih klasifikatora iz grupe g. Za svaku instancu iz $\boldsymbol{C}_i$ dobijamo vektor oblika $\{\boldsymbol{c}_i, r_{g_i}\}$, gde $\boldsymbol{c}_i$ predstavlja konkretan podatak iz dela podataka $\boldsymbol{C}_i$, dok r_{g_i} predstavlja rezultat klasifikatora g_i za dato $\boldsymbol{c}_i$. Kao izlaz *map* funkcije, dalje ka *reduce* funkciji se prosleđuje vektor $\boldsymbol{M}$ koji sadrži vektore oblika $\{\boldsymbol{c}_i, r_{g_i}\}$ označene sa $\boldsymbol{m}_i$, $1 \leq i \leq g$.

```
   Ulaz: Ključ K, Vektor podataka C_i
   Izlaz: Vektor klasifikovanih podataka M
 1 g = istreniraniBPNNovi();
 2 M = inicijalizujVektor();
 3 foreach g_i in g do
 4 |   foreach c_i in C_i do
 5 |   |   r_{g_i} = g_i.klasifikuj(c_i);
 6 |   |   m_i = <c_i, r_{g_i}>;
 7 |   |   M.dodaj(m_i);
 8 |   end
 9 end
10 emituj(K, M);
```

Slika 3.7: Map funkcija klasifikacione faze veštačke neuronske mreže

Reduce funkcija 3.8 preuzima vektor $\boldsymbol{M}$ i od njegovih vektora $\boldsymbol{m}_i$ pravi nove vektore $\boldsymbol{k}_i$ oblika $\{\boldsymbol{c}_i, r_1, r_2, ..., r_g\}$. Vrednosti $r_1, ..., r_g$ predstavljaju predviđanja svih klasifikatora za $\boldsymbol{c}_i$. Nad dobijenim vektorima $\boldsymbol{k}_i$, većinskim glasanjem određujemo klasu svakog $\boldsymbol{c}_i$. Rezultat *reduce* funkcije je vektor $\boldsymbol{V}$ koji sadrži vektore $\boldsymbol{v}_i$ oblika $\{\boldsymbol{c}_i, r_{t_i}\}$, gde je r_{t_i} rezultat većinskog glasanja za $\boldsymbol{c}_i$.

```
   Ulaz: Vektor M
   Izlaz: Rezultat većinskog glasanja vektor V
 1 K = inicijalizujVektor();
 2 for i ← 0 to M.duzina − 1 do
 3 |   k_i = M[i];
 4 |   for j ← 1 to M.duzina do
 5 |   |   if k_i.c_i == M[j].c_i then
 6 |   |   |   k_i.dodajPredvidjanje(M[j]);
 7 |   end
 8 |   K.dodaj(k_i);
 9 end
10 V = inicijalizujVektor();
11 foreach k_i in K do
12 |   v_i = vecinskoGlasanje(k_i);
13 |   V.dodaj(v_i);
14 end
15 emituj(V);
```

Slika 3.8: Reduce funkcija klasifikacione faze veštačke neuronske mreže

Glava 4

Eksperimenti

Kako bi jasno uvideli prednosti i mane platformi za rad sa skupovima velikih podataka opisanih u ovom radu, u daljem tekstu će biti više reči o kvalitativnom poređenju kao i o poređenju performansi.

4.1 Kvalitativno poređenje

Za svrhu kvalitativnog poređenja, korišćeni su skupovi podataka opisani u tabeli 4.1.

Mere korišćenje za kvalitativno poređenje su sledeće:

1. Tačnost (eng. accuracy)

$$Tačnost = \frac{TP}{TP + FP + TN + FN}$$

2. Preciznost (eng. precision)

$$Preciznost = \frac{TP}{TP + FP}$$

3. Odziv (eng. recall)

$$Odziv = \frac{TP}{TP + TN}$$

4. F-mera (eng. F1)

$$F1 = 2 * \frac{Preciznost * Odziv}{Preciznost + Odziv}$$

Skup podataka	Broj instanci	Negativne instance	Pozitivne instance	Broj atributa
Rak dojke u Američkoj državi Viskonsin	569	357	212	32
Oboljenje srca u Američkom gradu Klivlend	303	164	139	76
Autentifikacija novčanica	1372	762	610	4

Tabela 4.1: Korišćeni skupovi podataka u radu

Naivni algoritam je dodat kao dodatna validacija kvaliteta implementiranih klasifikacionih algoritama. Pod naivnim algoritmom se misli na algoritam koji uzima najčešću klasu iz trening podataka i njenu vrednost dodeljujemo svim instancama iz test podataka. Jasno je da predloženi algoritmi ne smeju dati rezultate lošije ili samo neznatno bolje od naivnog algoritma.

Sve implementacije klasifikacionih algoritama su pokretane 10 puta nad svakim skupom podataka za različite početne vrednosti generatora slučajnih brojeva i uzeta je prosečna vrednost za mere. Parametri korišćeni za klasifikacione algoritme su sledeći:

- Logistička regresija
  - $\boldsymbol{\theta}$ slučajna vrednost u intervalu $[0, 1]$
  - α slučajna vrednost u intervalu $[0.05, 0.01]$
  - Broj iteracija je 25000
- Veštačka neuronska mreža
  - Korišćena optimizaciona funkcija je stohastički gradijentni spust
  - Broj skrivenih slojeva je 5
  - Svaki skriveni sloj sadrži broj čvorova jednak broju atributa skupa nad kojim se vrši klasifikacija
  - Broj iteracija je 25000

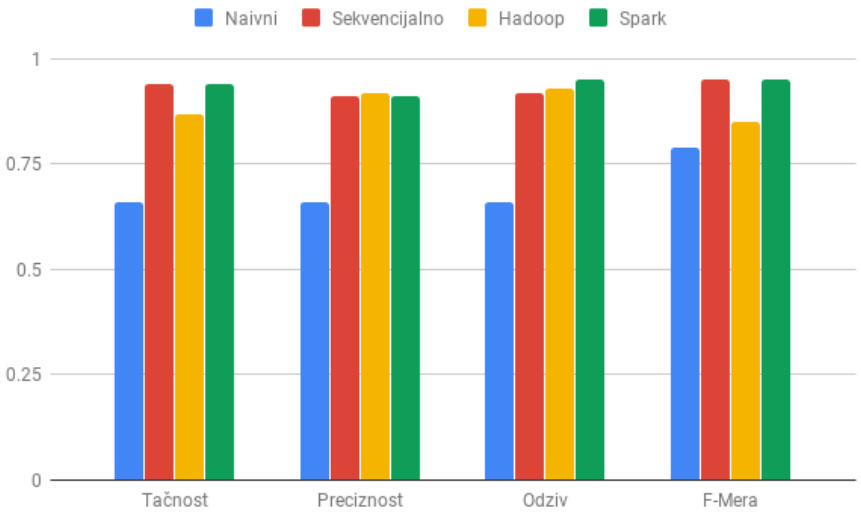
Vrednost naivnog algoritma za preciznost, odziv i f-meru za skup oboljenje srca i skup autentifikacija novca je jednaka nuli. Razlog te vrednosti je to što u oba skupa najčešću klasu predstavlja negativna klasa. Odziv i preciznost su definisani tako da u brojiocu sadrže broj tačno klasifikovanih pozitivnih instanci tj. TP. Kako se nama skup predviđenih vrednosti sastoji samo od vrednosti negativne klase, vrednost TP će biti jednaka nuli. F-mera se izračunava na osnovu preciznosti i odziva te će i ona biti jednaka nuli.

Algoritam	Metrika	Rak dojke	Oboljenje srca	Autentifikacija novca
Naivni	Tačnost	0.66	0.54	0.54
	Preciznost	0.66	0	0
	Odziv	0.66	0	0
	F mera	0.79	0	0
Sekvencijalna LR	Tačnost	0.94	0.74	0.94
	Preciznost	0.91	0.68	0.89
	Odziv	0.92	0.71	1
	F mera	0.95	0.74	0.94
Hadoop LR	Tačnost	0.87	0.75	0.92
	Preciznost	0.92	0.63	0.85
	Odziv	0.93	0.67	0.88
	F mera	0.85	0.73	0.92
Spark LR	Tačnost	0.94	0.74	0.94
	Preciznost	0.91	0.68	0.89
	Odziv	0.95	0.77	0.99
	F mera	0.95	0.74	0.94

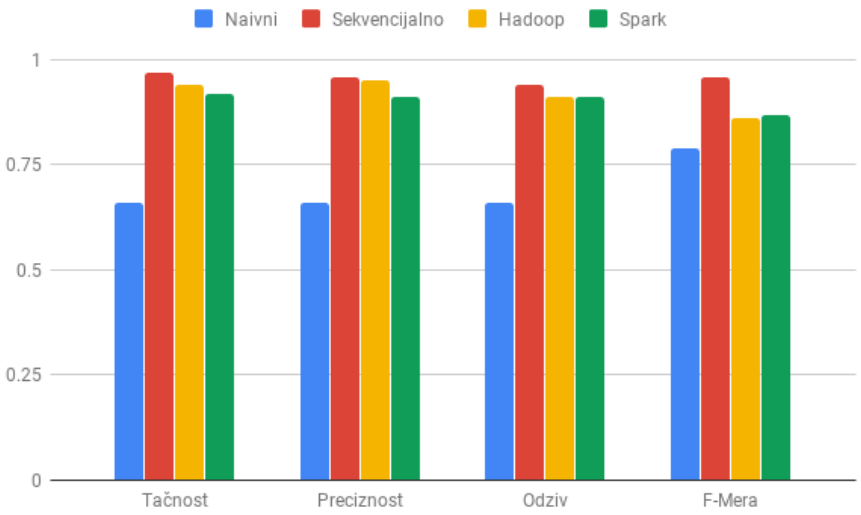
Tabela 4.2: Logistička regresija - Poređenje mera klasifikacionih algoritama nad datim skupovima

Algoritam	Metrika	Rak dojke	Oboljenje srca	Autentifikacija novca
Naivni	Tačnost	0.66	0.54	0.54
	Preciznost	0.66	0	0
	Odziv	0.66	0	0
	F mera	0.79	0	0
Sekvencijalna ANN	Tačnost	0.97	0.75	1
	Preciznost	0.96	0.74	1
	Odziv	0.94	0.78	1
	F mera	0.96	0.78	1
Hadoop ANN	Tačnost	0.94	0.76	0.96
	Preciznost	0.95	0.73	0.98
	Odziv	0.91	0.71	0.95
	F mera	0.86	0.76	0.99
Spark ANN	Tačnost	0.92	0.74	1
	Preciznost	0.91	0.74	1
	Odziv	0.91	0.72	1
	F mera	0.87	0.77	1

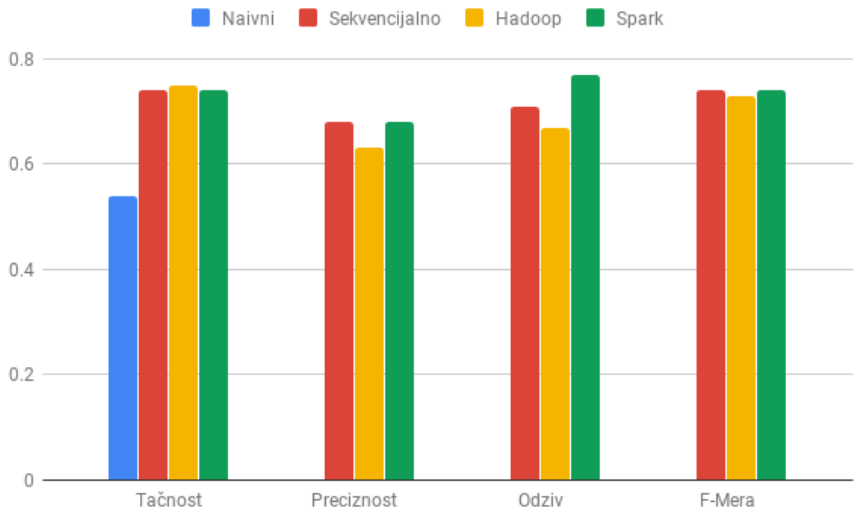
Tabela 4.3: Veštačka neuronska mreža - Poređenje mera klasifikacionih algoritama nad datim skupovima



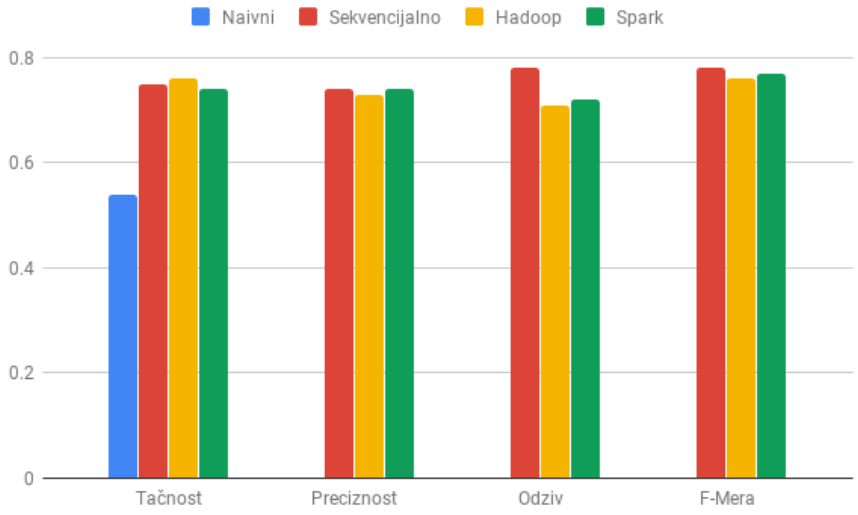
Grafik. 4.1: Rak dojke logistička regresija



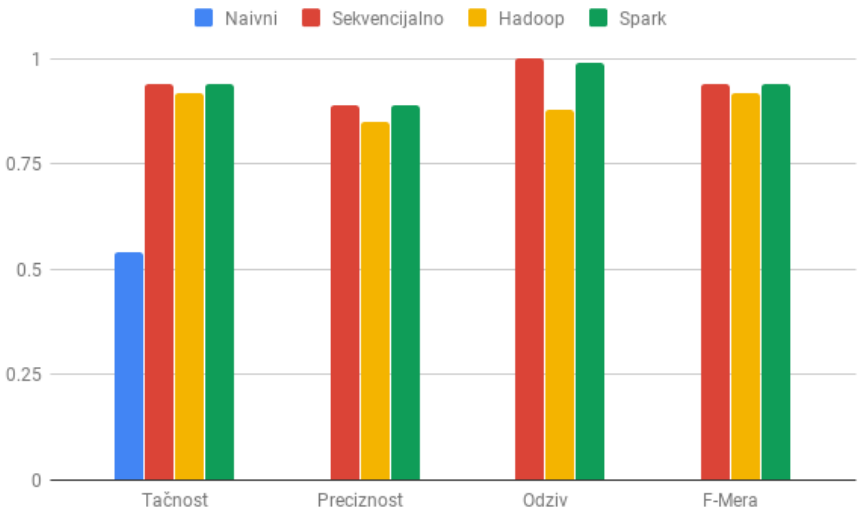
Grafik. 4.2: Rak dojke neuronska mreža



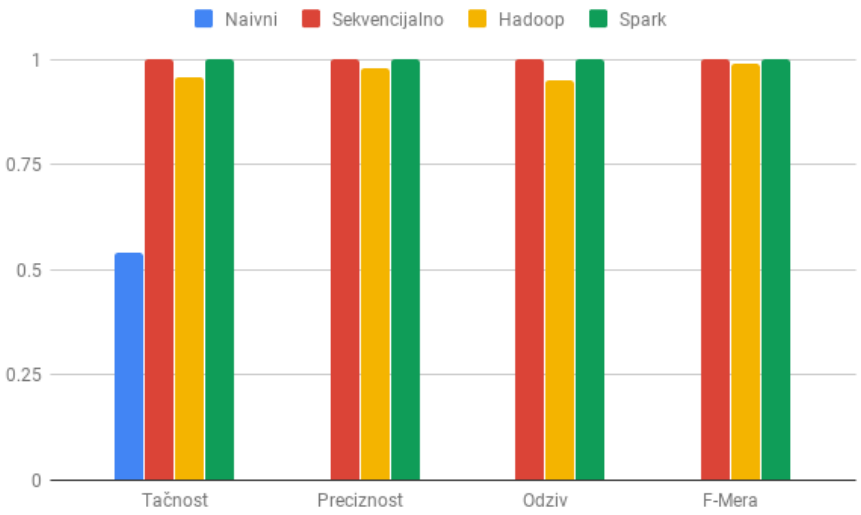
Grafik. 4.3: Oboljenje srca logistička regresija



Grafik. 4.4: Oboljenje srca neuronska mreža



Grafik. 4.5: Autentifikacija novca logistička regresija



Grafik. 4.6: Autentifikacija novca neuronska mreža

Zbog implementacionih detalja postoji razlika u dobijenim vrednostima mera za *Spark* i *Hadoop* pristup. *Hadoop* implementacija koristi regularne izraze da rezultat *map* funkcije pretvori u nisku. *Reduce* funkcija takođe koristi regularne izraze da bi izvršila pretvaranje dobijenog rezultata iz niske nazad u realan broj što dovodi do gubitka preciznosti i manje stabilnog ponašanja *Hadoop* pristupa. *Spark*

pristup nakon inicijalnog učitavanja podataka radi samo sa realnim brojevima bez dodatnih pretvaranja u druge tipove podataka.

4.2 Poređenje performansi

U ovom radu, vršena su dva tipa poređenja: poređenje performansi sekvencijalne, *Spark* i *Hadoop* implementacije klasifikacionih algoritama i poređenje performansi Spark implementacije u odnosu na broj uključenih jezgara i sekvencijalne implementacije.

Postavka okruženja platformi za rad sa skupovima velikih podataka: klaster se sastoji od jednog računara koji je paralelizovan brojem jezgara, menadžer resursa zadužen za *Spark* i *Hadoop* eksperimente je *YARN* (iako Spark sadrži svoj menadžer resursa), ulazne i izlazne operacije sa podacima su obavljane nad *HDFS*-om.

Verzije platformi su sledeće: *Apache Spark 2.3.2, Hadoop 2.8.4*.

Specifikacija računara korišćenog za eksperimente: 16GB DDR3 na 1600MHz, Intel® Core™ i7-4790 CPU @ 3.60GHz sa 8 jezgara pod Ubuntu 14.04 operativnim sistemom.

Skup korišćen za poređenje performansi je skup podataka za rak dojke, umnožen tako da ima veličine: 100 MB, 200MB, 400 MB, 800 MB, 1600 MB, 3200 MB, 6400 MB, 14000 MB.

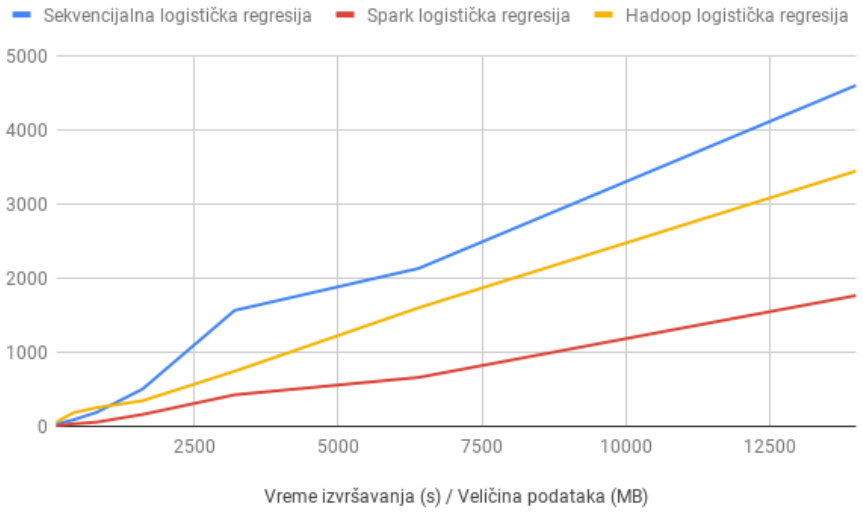
Poređenje performansi sekvencijalne, Spark i Hadoop implementacije

Tabela 4.4 i grafik 4.7 prikazuju poređenje sekvencijalne, Spark i Hadoop implementacije logističke regresije dok tabela 4.5 i grafik 4.8 prikazuju poređenje implementacije veštačke neuronske mreže.

Na graficima 4.7 i 4.8 je primetan linearan rast vremena izvršavanja algoritama. Hadoop implementacija, pri klasifikaciji manjih skupova podataka, kaska za sekvencijalnom zbog cene pokretanja samog algoritma, ali od 1600MB ta cena prestaje da pravi značajnu razliku. Spark implementacija zbog internih optimizacionih tehnika (korišćenje direktnih acikličnih grafova, čuvanje RDD-ova u memoriji, itd.) je i do tri puta brža od sekvencijalne implementacije.

Algoritam / Veličina podataka (MB)	100	200	400	800	1600	3200	6400	14000
Sekvencijalna logistička regresija	30	50	89	189	504	1565	2132	4601
Spark logistička regresija	13	18	34	57	161	426	661	1765
Hadoop logistička regresija	55	101	185	253	346	744	1601	3445

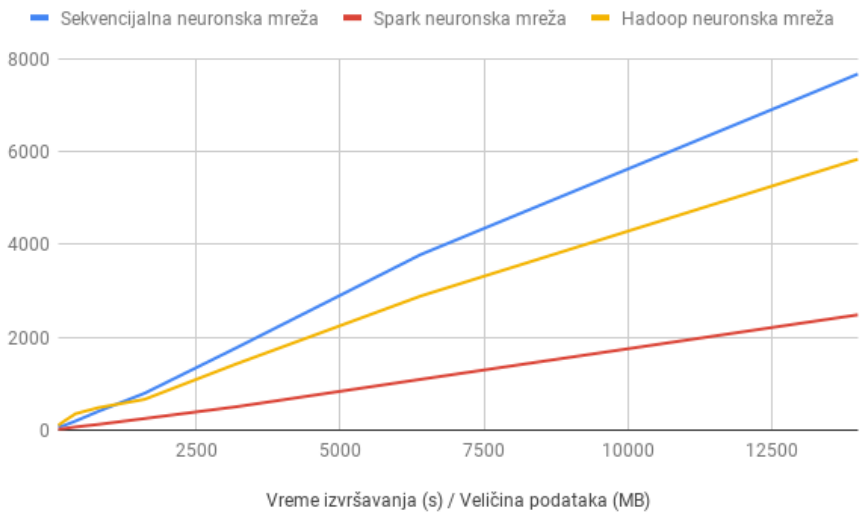
Tabela 4.4: Poređenje performansi logističke regresije izraženo u sekundama



Grafik. 4.7: Poređenje performansi logističke regresije

Algoritam / Veličina podataka (MB)	100	200	400	800	1600	3200	6400	14000
Sekvencijalna neuronska mreža	74	101	197	408	796	1778	3785	7676
Spark neuronska mreža	22	35	70	125	251	506	1096	2486
Hadoop neuronska mreža	106	194	355	486	665	1431	2891	5840

Tabela 4.5: Poređenje performansi neuronske mreže izraženo u sekundama



Grafik. 4.8: Poređenje performansi neuronske mreže

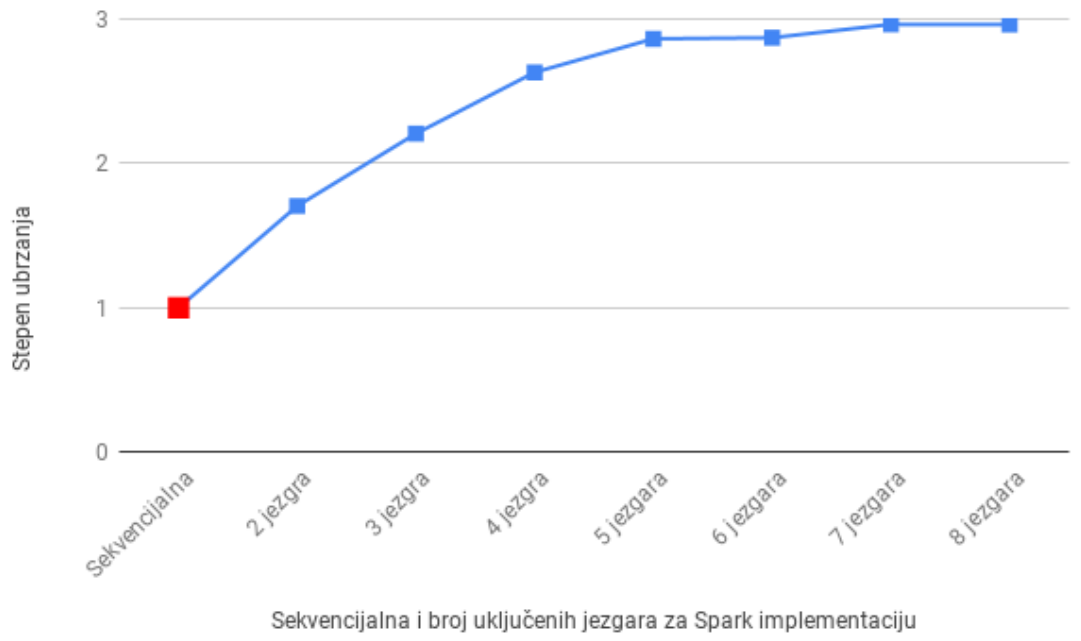
Poređenje performansi na osnovu uključenog broja jezgara na Spark platformi

Radi pravljenja uporednog odnosa platformi za rad sa skupovim velikih podataka, eksperimenti su vršeni na Spark platformi sa različitim brojem jezgara kao i na sekvencijalnoj implementaciji.

Sa tabela 4.6 i 4.7 kao i grafika 4.9 i 4.10 može se zaključiti da Spark implementacija čak i sa dva uključena jezgra ima performanse skoro dva puta bolje od sekvencijalne implementacije. Sa uključenih pet i više jezgara te performanse postaju skoro tri puta bolje. Upoređujući performanse Spark implementacije u odnosu na broj jezgara može se zaključiti da nakon pet uključenih jezgara, rast performansi je neznatan.

Sekvencijalna i broj jezgara \Veličina podataka (MB)	100	200	400	800	1600	3200	6400	14000
Sekvencijalna	30	50	89	189	504	1565	2132	4601
2 jezgra	25	33	48	111	268	765	1220	2739
3 jezgra	14	22	41	84	237	741	921	2057
4 jezgra	12	18	35	79	216	498	726	1926
5 jezgara	11	18	32	66	173	478	683	1910
6 jezgara	14	19	32	57	164	459	667	1937
7 jezgara	12	18	32	56	168	466	649	1777
8 jezgara	13	18	34	57	161	426	661	1765

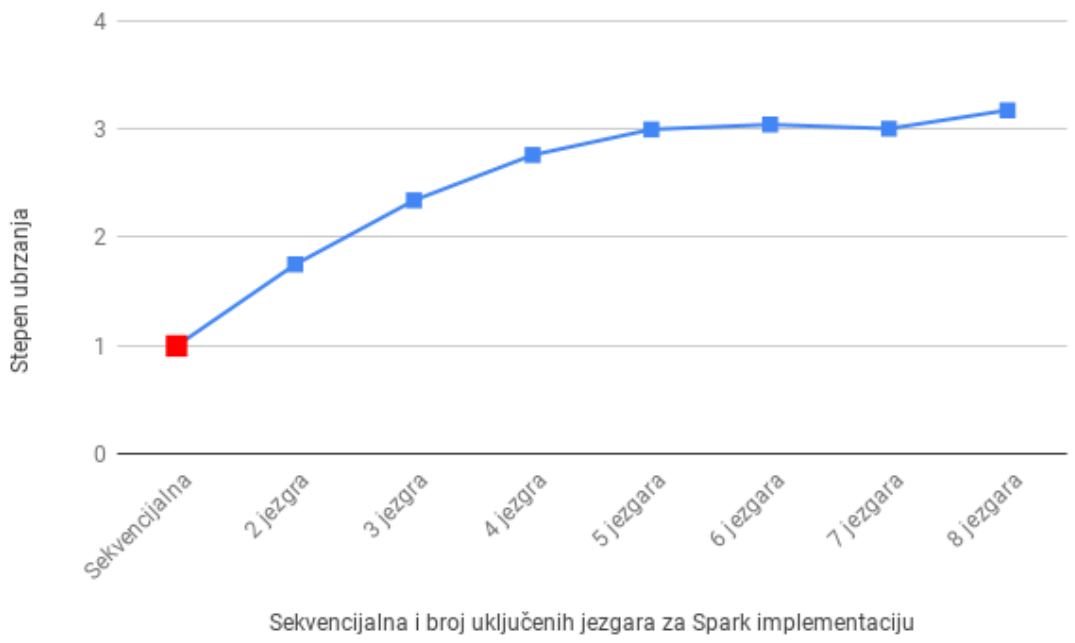
Tabela 4.6: Logistička regresija - Performanse sekvencijalne i Spark implementacije izražene u sekundama



Grafik. 4.9: Rast performansi - Logistička regresija

Sekvencijalna i broj jezgara \Veličina podataka (MB)	100	200	400	800	1600	3200	6400	14000
Sekvencijalna	74	101	197	408	796	1778	3785	7676
2 jezgra	37	61	117	235	510	997	2045	4158
3 jezgra	25	44	87	203	375	765	1520	3171
4 jezgra	22	41	75	168	282	667	1221	2758
5 jezgara	22	37	77	128	274	589	1135	2523
6 jezgara	25	40	64	126	262	565	1135	2431
7 jezgara	23	52	63	122	284	568	1077	2467
8 jezgara	22	35	70	125	251	506	1096	2486

Tabela 4.7: Veštačka neuronska mreža - Performanse sekvencijalne i Spark implementacije izražene u sekundama



Grafik. 4.10: Rast performansi - Veštačka neuronska mreža

Glava 5

Zaključak

U ovom radu implementirani su sekvencijalna i paralelizovana rešenja logističke regresije i veštačke neuronske mreže. Paralelizovano rešenje koristi *Hadoop* i *Apache Spark* platforme. Rezultat ovog rada su kvalitativna poređenja i poređenja performansi implementiranih rešenja koje prikazuju njihove prednosti i mane.

Rezultat kvalitativnog poređenja sekvencijalnih i paralelizovanih rešenja pokazuje mala odstupanja u dobijenim vrednostima za mere poređenja, iz čega možemo zaključiti da su predikcije implemententiranih rešenja podjednako validne. Vreme izvršavanja sekvencijalnog algoritma linearno raste sa porastom veličine podataka. Prednost u performansama *Hadoop* rešenja u odnosu na sekvencijalno nastaje tek od 1600 MB. Lošije performanse *Hadoop* rešenja na manjim skupovima podataka su posledica skupe inicijalizacije same *Hadoop* platforme pri svakom pokretanju rešenja. *Spark* rešenje pokazuje i do tri puta bolje performanse od sekvencijalnih rešenja, dok u poređenju sa *Hadoop* rešenjem, razlika u performansama se smanjuje sa rastom podataka. Poređenje performansi na osnovu uključenog broja jezgara na *Spark* platformi pokazuje linearan rast performansi do pet uključenih jezgara, nakon čega je dobitak u performansama neznatan.

Najveća mana paralelizovanih rešenja je složenost implementacije. Da bi uspešno implementirali paralelizovano rešenje potrebno je znanje implementacije sekvencijalnog rešenja, poznavanje platformi za paralelizaciju i podešavanja okruženja za iste. Za postavljanje klastera sa više računara potrebno je poznavanje mrežnog računarstva. Autor ovog rada iz navedenih razloga nije bio u mogućnosti da implementira klaster koji se sastoji od više računara. *Spark* kao novija platforma od *Hadoop* platforme uprošćava proces korišćenja i implementacije distribuiranih algoritama, ali *Hadoop* i dalje predstavlja veliki korak u odnosu na

sekvencijalnu implementaciju.

Budući koraci u razgraničavanju prednosti i mana sekvencijalnih i paralelizovanih rešenja bi sigurno predstavljali veći fokus na *Spark* platformi i ugrađenim implementacijama klasifikacionih algoritama, kao i uspostavljanje klastera sa više računara.

Literatura

[1] Big data: The next frontier, for innovation, competition, and productivity. https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation.

[2] How much data do we create every day? the mind-blowing stats everyone should read. https://bit.ly/2X5g9pz.

[3] dr Mladen Nikolić Anđelka Zečević. Mašinsko učenje. http://ml.matf.bg.ac.rs/readings/ml.pdf.

[4] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mob. Netw. Appl.*, 19(2):171–209, April 2014.

[5] Reinsel D Gantz J. Extracting value from chaos. *IDC*, pages 1 – 12, 2011.

[6] Rydning J Gantz J, Reinsel D. The digitization of the world from edge to core. *IDC*, pages 1 – 28, 2018.

[7] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Inc., 1st edition, 2015.

[8] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

[9] Yang Liu, Weizhe Jing, and Lixiong Xu. Parallelizing backpropagation neural network using mapreduce and cascading model. *Computational Intelligence and Neuroscience*, 2016:1–11, 01 2016.

[10] Marco Aurélio Lotz. Of mapreduce and men. http://www.marcolotz.com/?p=67.

[11] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[12] Andrew NG. Machine learning coursera course. `https://www.coursera.org/learn/machine-learning`.

[13] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 30(4):431 – 448, 2018.

[14] Senem Sagiroglu and Duygu Sinanc. Big data: A review. pages 42–47, may 2013.

[15] R. Schneider. *Hadoop for Dummies*. John Wiley & Sons Canada, Limited, 2012.

[16] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 4th edition, 2015.

[17] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.