UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET

Borislav Bosnić

Gradient boosting algoritmi

diplomski rad

Banja Luka, jul 2020.

Tema: ***GRADIENT BOOSTING*** **ALGORITMI**

Ključne riječi:
Mašinsko učenje
Boosting
Gradijentni spust
Stablo odlučivanja
Regresija
Klasifikacija
Loss **funkcija**
Tačnost
Preciznost
Odziv
Pristrasnost
Varijansa
AdaBoost
XGBoost
CatBoost
LightGBM

Komisija: **prof. dr Slavko Marić, predsjednik**
prof. dr Zoran Đurić, mentor
Aleksandar Keleč, ma, član

Uz rad je priložen CD.

Kandidat:
Borislav Bosnić

UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET
KATEDRA ZA RAČUNARSTVO I INFORMATIKU

Tema: *GRADIENT BOOSTING* ALGORITMI

Zadatak: Mašinsko učenje. Tipovi algoritama za mašinsko učenje. *Gradient Boosting* algoritmi. Detaljno analizirati LightGBM, XGBOOST, CatBoost i AdaBoost algoritme. Praktičan rad — treniranje različitih modela pomoću analiziranih algoritama. Uporedna analiza dobijenih rezultata.

Mentor: prof. dr Zoran Đurić

Kandidat: Borislav Bosnić (1165/14)

Banja Luka, jun 2020.

Sadržaj

1. UVOD

U vrijeme brzog tehnološkog napretka i svojevrsne informatičke revolucije računarski i informacioni sistemi čine okosnicu na kojoj se u dobroj mjeri zasniva cjelokupna ljudska egzistencija. Teško je zamisliti bilo koju sferu ljudskog djelovanja u koju, u manjoj ili većoj mjeri, nisu uključeni pomenuti sistemi: počevši od industrije, ekonomije, cjelokupne privrede, pa sve do društvenih mreža. U takvom okruženju, svakodnevno se generišu ogromne količine podataka. Prema istraživanju Univerziteta Berkley, godišnja svjetska produkcija informacija za 2002. godinu je iznosila 5 EB[1] [1]. Ovaj broj je od tada eksponencijalno rastao, da bi 2012. bilo objavljeno da je 90% od ukupne količine podataka generisane u ljudskoj istoriji nastalo u prethodne dvije godine. [2]. Danas je ukupnu količinu podataka na internetu nemoguće odrediti, ali je jasno da u njima leži najveće blago moderne civilizacije – informacije, odnosno znanje. Informacija je glavna sirovina u informatičkoj ekonomiji i pokretač modernog svijeta. Velike kompanije, kao što su Google[2], Microsoft[3], Amazon[4], Facebook[5], svoju moć zasnivaju na ogromnoj količini podataka koje posjeduju. Međutim, nije dovoljno samo posjedovati podatke, potrebno ih je pretvoriti u informacije[6]. S tim ciljem razvijeni su mnogi sistemi koji podatke organizuju i obrađuju. Otišlo se i korak dalje pa je kreirana nova generacija algoritama sposobnih da nauče veze i zavisnosti koje postoje između podataka, te da stečeno znanje iskoriste u donošenju zaključaka i predviđanju određenih aspekata vezanih za neke buduće, do tada neviđene podatke. Ova sposobnost je širom otvorila vrata vještačkoj inteligenciji i mašinskom učenju.

U oblasti mašinskog učenja sve veću važnost imaju *gradient boosting* algoritmi. Njihova robusnost, efikasnost, sposobnost ispravljanja sopstvenih grešaka kontinualno kroz iteracije i veoma široka oblast primjene su samo neki od razloga takvog naglog rasta popularnosti. U ovom radu je prikazan način funkcionisanja ovih algoritama, deskriptivno i formalno matematički. Takođe je dat pregled 4 osnovne implementacije ovog tipa algoritama, kao i njihovo poređenje, kako u domenu noviteta koje svaka od ovih implementacija uvodi, tako i u pogledu performansi (uzimajući u obzir vrijeme potrebno za izvršenje zadatka, kao i rezultate tog izvršavanja). Uporedna analiza, iznesena u ovom radu, ima za cilj da olakša izbor adekvatnog algoritma za rješavanje nekog konkretnog problema.

U drugoj glavi daje se generalno objašnjenje vještačke inteligencije i mašinskog učenja, te uspostavlja veza između njih. Neveden je i širok spektar oblasti u kojima algoritmi mašinskog učenja imaju svoju primjenu, kao i njihova podjela.

U trećoj glavi detaljno su opisani *gradient boosting* algoritmi. Izvršena je njihova dekompozicija na tri osnovne cjeline od kojih su izgrađeni: stabla odlučivanja, *boosting* i metodu gradijentnog spusta. Svaka od navedenih cjelina je zasebno objašnjena, nakon čega je prikazan način funkcionisanja *gradient boosting* algoritma korak po korak, uz propratnu matematičku formulaciju svakog koraka.

Četvrta glava sadrži dodatne koncepte mašinskog učenja, kao što su *overfitting* i *underfitting*, pristrasnost i varijansu, razloge podjele podataka na trening, validacione i testne

[1] eng. *exabyte* – $5*10^{18}$ bajta
[2] https://www.google.com/
[3] https://www.microsoft.com/
[4] https://www.amazon.com/
[5] https://www.facebook.com/
[6] Podatak predstavlja neku vrijednost, niz simbola bez konteksta i značenja. Informacija je obrađen podatak koji nosi saznajnu poruku i ima značenje za primaoca.

Commented [ZD1]: Generalne primjedbe:
1. Svaka slika i tabela mora biti referencirana u tekstu. Primjeri:
- Na slici X.Y prikazano je...
- Ml algoritmi se dijele na (slika X.Y): ...
Svaku sliku i tabelu potrebno je pojasniti kroz kratak prateći tekst.
2. Slika mora biti što je moguće bliže tekstu u kojem se referencira
3. Prazne linije ispred naslova poglavlja ili sekcije - pogledati sljedeći komentar
4. Ukloniti višestruke bjeline između riječi (obično dvije)
5. Ukloniti višak praznih redova (npr. između slika 3.6 i 3.7)
6. Natpisi slika i tabela treba da imaju font za 1 manji od teksta
7. Za slike preuzete iz literature, navesti u fusnoti odakle su preuzete, a ne navoditi redni broj iz literature.

Commented [ZD2]: Ako je cilj bio da nazivi poglavlja budu „niže“ u odnosu na početak stranice, onda ste možda mogli koristiti spacing opciju na stilovima, bez ovog praznog reda koji prethodi naslovu. Ovo je samo zbog toga da bi rad bio bolje formatiran i zbog eventualnoh problema kod izvoza u drugi format, za potrebe štampe i sl.

Commented [ZD3]: možda dodati „, u dobroj mjeri,“ ili nešto slično.
Vjerovatno bi čovjek preživio i bez njih, a prema ovoj rečenici to se ne bi desilo... ☺

Commented [ZD4]: U fusnotu staviti URL adrese web sajtova ovih kompanija....

Commented [ZD5]: Šablone čijeg ponašanja? Preformulisati...

podatke, načine pripreme podataka prije njihovog prosljeđivanja nekom algoritmu mašinskog učenja, te mjere za evaluaciju rezultata algoritma. Ovi koncepti nisu vezani samo za *gradient boosting* algoritme, već generalno za mašinsko učenje, te su stoga izdvojeni u zasebnoj glavi.

U petoj glavi se govori o četiri *gradient boosting* algoritma: AdaBoost, XGBoost, LightGBM i CatBoost. Za svakog od njih navedena su osnovna svojstva i unapređenja u odnosu na ostale algoritme, te izdvojene razlike u implementaciji i performansama.

Commented [ZD6]: referenca(e)

Šesta glava sadrži pojedinosti o praktičnom radu. U njoj su detaljno opisani podaci nad kojima su primijenjeni pomenuti algoritmi, opisan je proces njihove pripreme kao i pojedinosti oko podešavanja samih algoritama. Finalno, prikazani su rezultati svakog od njih i izvršena uporedna analiza. Sav kôd prikazan u ovoj glavi napisan je u programskom jeziku Python.

Na kraju rada dat je zaključak.

2. VJEŠTAČKA INTELIGENCIJA I MAŠINSKO UČENJE

2.1. Vještačka inteligencija

U modernom svijetu se sve češće i glasnije pominje termin vještačke inteligencije (eng. *Artificial Inteligence – AI*). Bilo da se o ovom terminu priča u futurističkom kontekstu kao velikom iskoraku ljudske civilizacije, ili sa dozom bojazni od neželjenih posljedica koje bi široka primjena vještačke inteligencije mogla da prouzrokuje[7], teme koje obuhvataju ovu oblast uvijek privlače veliku pažnju svjetske javnosti.

Vještačka inteligencija se definiše kao studija dizajna inteligentnih agenata – sistema koji djeluju inteligentno u nekom okruženju. Sve što ti agenti čine je u skladu sa okolnostima i definisanim ciljem. Agent je fleksibilan na promjenu ciljeva i okruženja, uči iz iskustva i na osnovu toga sam donosi odluke [6]. Termin vještačka inteligencija se često koristi da opiše mašine (ili računare) koji oponašaju kognitivne funkcije ljudskog mozga kao što su učenje ili rješavanje problema [7]. Vještačka inteligencija se zasniva na brojnim naukama kao što su matematika, statistika, teorija vjerovatnoće, logika, filozofija, lingvistika, te raznim neuronaukama.

Osnovne AI oblasti su robotika, računarski vid (eng. *computer vision*), mašinsko učenje (eng. *Machine Learning - ML*) i obrada prirodnog jezika (eng. *Natural Language Processing - NLP*).

2.2. Mašinsko učenje

2.2.1. Definicija i osnovne karakteristike

Kao što je već rečeno, ML je jedna od osnovnih oblasti AI. ML se bavi računarskim algoritmima koji imaju sposobnost da se automatski poboljšavaju kroz iskustvo. Jedno od prvih pojavljivanja termina ML seže još iz 1959. godine u radu Arthura Samuela posvećenom pobjedi računara nad čovjekom u igri dame (eng. *checkers*). Ovo je prvi put u istoriji da je računar pobijedio čovjeka u nekoj igri što se smatra veoma važnim događajem u istoriji ML-a. U pomenutom radu se navodi kako će programiranje računara da uče iz iskustva skoro u potpunosti eliminisati potrebe za detaljnim programerskim naporom [8].

ML algoritmi kao ulaz primaju ogromne skupove podataka koje potom procesiraju i u kojima uočavaju određene obrasce i veze između njih. Nakon toga, ovakvi algoritmi kreiraju model kojim objašnjavaju posmatrani fenomen opisan u ulaznom skupu podataka (eng. *input dataset*, u daljem tekstu se svaki skup podataka referencira kao *dataset*). Zahvaljujući kreiranom modelu, ovi algoritmi su na kraju sposobni da daju svoje predviđanje ponašanja srodnog, do tada neviđenog fenomena zahvaljujući znanju stečenom u fazi učenja. Bitno je napomenuti da podaci koji opisuju taj novi fenomen ne moraju postojati u ulaznom *dataset*-u (otuda ovaj epitet "neviđeni"). Iako su, tehnički gledano, ti podaci za njega potpuno novi, ML algoritam je ipak u

[7] Fizičar Stephen Hawking, osnivač Microsofta, Bill Gates [3], kao i osnivač SpaceX-a, Elon Musk, [4] izrazili su svoju zabrinutost da bi vještačka inteligencija mogla evoluirati do te tačke da je ljudska rasa više ne bi mogla kontrolisati. Neki su otišli toliko daleko da razvoj vještačke inteligencije vide kao egzistencijalni rizik. Hawking je u svom obraćanju za BBC izjavio da bi razvoj vještačke inteligencije mogao ukazati na kraj ljudske rase. Jednom razvijena, vještačka inteligencija bi nastavila samu sebe nadograđivati sve većom brzinom, sa kojom se ljudi, ograničeni sporom biološkom evolucijom, ne bi mogli takmičiti [5].

stanju da izvuče traženi zaključak samo na osnovu veza i šablona prepoznatih i naučenih u pomenutom ulaznom *dataset*-u.

Ono što takođe karakteriše ovakve algoritme je činjenica da priroda problema koji se rješava uopšte nije bitna, jer će se algoritam uspješno prilagoditi bilo kojem novom problemu bez potrebe za ikakvim modifikacijama koda ili zadavanjem dodatnih instrukcija. Dovoljno je algoritmu samo zadati "novi materijal za učenje", odnosno promijeniti ulazni *dataset*, a algoritam će ponovo iz promijenjenog *dataset*-a učiti nove zavisnosti i izvlačiti nove zaključke koje će ugraditi u model, te kasnije koristiti za rješavanje tog novog problema.

2.2.2. Primjena

Ova osobina ML algoritama omogućila je njihovu primjenu u različitim oblastima, kao što su: medicina, ekonomija, bankarstvo, marketing, bioinformatika, poljoprivreda itd. Zanimljiva je primjena ovakvih algoritama u detekciji prevara na internetu, kao i prevara vezanih za upotrebu kreditnih kartica, zatim detekciji *spam*[8] poruka, izgradnji *recommendation* sistema[9], analizi ponašanja korisnika, analizi sentimenta[10], *online* reklamiranju gdje se reklame prikazuju ciljanim korisnicima na osnovu njihovih interesovanja i potreba izvučenih iz stranica koje najčešće posjećuju i sl. Takođe, široku primjenu su našli u mašinskom prevođenju, razumijevanju i procesiranju prirodnog jezika, prepoznavanju rukopisa[11] i govora.

U posljednje vrijeme se ozbiljno radi na primjeni ML-a u davanju medicinskih dijagnoza, Već postoje algoritmi koji zahvaljujući sakupljenom znanju o milionima različitih bolesti i načinima njihove manifestacije otkrivaju obrasce oboljenja u elektronskim zdravstvenim zapisima pacijenata i obavještavaju medicinsko osoblje o bilo kakvim uočenim anomalijama. Oni se najčešće koriste u detekciji i dijagnozi kancerogenih stanja pacijenta. Algoritam posmatra oblik i veličinu ćelija, neregularnosti na površini ćelije, znakove invazije na tkiva i krvne sudove, tragove nekroze i sl, te ih upoređuje sa podacima iz ulaznog *dataset*-a koji sadrži milione zapisa o zdravim i nezdravim ćelijama i iz toga sa određenim stepenom vjerovatnoće predviđa konačnu dijagnozu.

Uzimajući u obzir ove, kao i mnoge druge primjene ML algoritama, jasno je da će oni u značajnoj mjeri unaprijediti donošenje važnih odluka i olakšati brojne izazove sa kojima se čovječanstvo susreće.

Commented [ZD7]: pojašnjenje u fusnoti

Commented [ZD8]: tj. predviđa dijagnozu sa određenim stepenom vjerovatnoće

[8] Neželjna ili netražena poruka koja se obično šalje velikom broju primalaca najčešće u svrhu komercijalnog oglašavanja.

[9] Sistemi za predlaganje sadržaja na osnovu interesovanja korisnika, kao što su *playlist* generatori na Youtube-u, Netflix-u, Spotify-u, predlagači proizvoda i servisa na amazonu, ili predlagači sadržaja na društvenim mrežama kao što su Facebook i Twitter.

[10] Analiziranje teksta u cilju automatske ekstrakcije pozitivnih ili negativnih mišljenja koje se prožimaju tekstom, odnosno određivanja tona u kojem je tekst napisan. Sirovina za algoritme primijenjene u ovoj oblasti su *review*-ovi i odgovori na ankete, kao i sadržaj sa društvenih mreža. Analiza sentimenta nad ovakvim sadržajima daje uvid u stavove i mišljenja potrošača, glasača na izborima ili bilo koje druge ciljne grupe.

[11] Prepoznavanje rukopisa (eng. *handwriting recognition)* je samo podtip mnogo šire oblasti iz ML-a – prepoznavanja slika (eng. *image recognition)*. Prepoznavanje slika ima veoma široku primjenu, a možda je najpoznatija ona razvijena od strane Facebooka, koji je u stanju da prepozna osobu na slici sa preciznošću od 98%, gotovo jednako dobro kao i čovjek [9].

2.2.3. Tipovi ML algoritama

ML algoritmi su organizovani u taksonomije, zasnovane na njihovoj svrsi i željenom ishodu algoritma [10]. Osnovni tipovi ML algoritama su:

- Nadgledano učenje (eng. *Supervised learning*)
- Nenadgledano učenje (eng. *Unsupervised learning*)
- Polunadgledano učenje (eng. *Semi-supervised learning*)
- Učenje potkrepljivanjem (eng. *Reinforcement learning*)

Kod **nadgledanog učenja** algoritam generiše funkciju koja mapira ulaze u željene izlaze. Ovdje su ti željeni izlazi već poznati i prisutni u pomenutom ulaznom *dataset*-u. Takvi izlazi nazivaju se ciljne vrijednosti (eng. *target values*), a nerijetko se koristi i termin labele – pa se *dataset* sa poznatim izlaznim vrijednostima često naziva i labelisani *dataset*. Primjer jednog takvog *dataset*-a je skup podataka o pacijentima, njihovim fizičkim karakteristikama, trenutnim simptomima i genetskim predispozicijama za određenu bolest, a ciljne vrijednosti su informacije o svakom pojedinačnom pacijentu, da li ima ili nema pomenutu bolest. ML algoritam za nadgledano učenje će pokušati da modeluje veze i zavisnosti između ciljnih vrijednosti i ulaznih podataka, kako bi na osnovu tih veza i generisanog modela mogao predvidjeti ciljnu vrijednost za nove podatke (odnosno u ovom slučaju potencijalno oboljenje na osnovu simptoma i pomenutih karakteristika nekog novog pacijenta). Najčešći problemi koji se rješavaju ovim tipom ML algoritama su klasifikacioni i regresioni problemi[12]. Najpoznatiji algoritmi ovog tipa su: linearni klasifikatori (logistička regresija, Bajesov klasifikator, metod potpornih vektora (eng, *Support Vector Machine – SVM*), perceptron), linearna regresija, neuronske mreže, stabla odlučivanja, *random forest*, te *boosting* algoritmi.

Kod **nenadgledanog učenja** ciljne vrijednosti nisu prisutne, odnosno ulazni *dataset* nije labelisan. Ova familija algoritama se najčešće koristi kod detekcije šablona i deskriptivnog modelovanja. S obzirom na to da ovdje ne postoje ciljne vrijednosti, algoritam nije u stanju da modeluje veze između njih i podataka u *dataset*-u, nego prepoznaje veze između samih podataka, traži zakonitosti, detektuje šablone, sumira i grupiše podatke što za cilj ima detaljnije opisivanje podataka i generisanje smislenih uvida u iste. Najčešći problemi koji se rješavaju ovim tipom ML algoritama su problemi grupisanja odnosno klasterizacije podataka (eng. *clustering)*[13], učenja pravilom asocijacije (eng. *association rule learning*)[14], pa čak i problem redukcije dimenzionalnosti (eng. *dimensionality reduction*). Najpoznatiji algoritmi ovog tipa su: *K-means clustering* i DBSCAN (klasterizacija), FP-rast i apriori algoritam (učenja pravilom asocijacije), analiza glavnih komponenata (eng. *Principal Component Analysis – PCA)* i singularna dekompozicija (eng. *Singular Value Decomposition – SVD*) (redukcija dimenzionalnosti).

[12] Klasifikacioni problem zahtijeva klasifikaciju ulaznih primjera u dvije ili više klasa, a regresioni problem na osnovu ulaznog primjera predviđa neku kontinualnu vrijednost. Formalnije, klasifikacija je funkcija koja mapira ulazne vrijednosti X u diskretne izlazne vrijednosti Y koje se često nazivaju labele ili kategorije. Regresija je funkcija koja mapira ulazne vrijednosti X u kontinualne izlazne vrijednosti Y. Primjer klasifikacionog problema: da li pacijent boluje od neke bolesti, npr. da li ima rak? Primjer regresionog problema: određivanje cijene stana na osnovu lokacije, površine, broja soba, godine izgradnje itd.

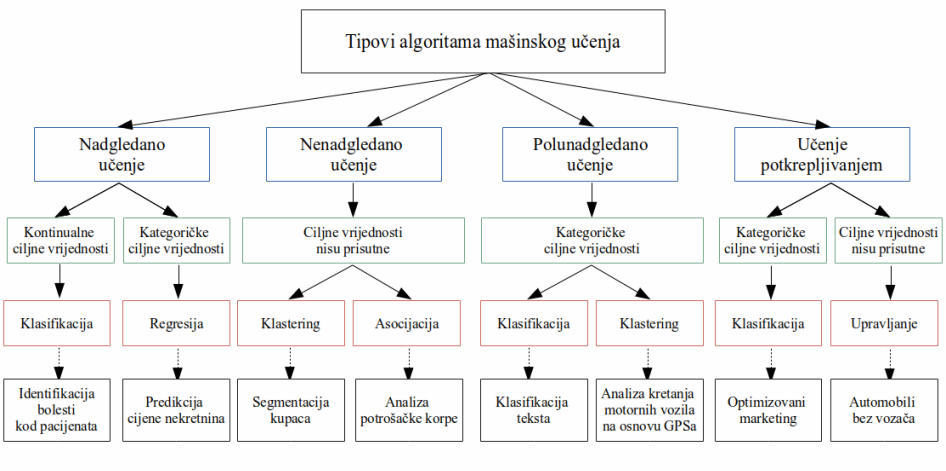
[13] Klasterizacija je proces dijeljenja instanci podataka u određen broj grupa, prema njihovoj sličnosti. Primjer problema koji se rješavaju klasterizacijom: identifikacija zajednica na društvenim mrežama ili detekcija raznorodnih tkiva na medicinskim snimcima.

[14] Učenje pravilom asocijacije je metod za otkrivanje interesantnih odnosa između promjenljivih u velikim bazama podataka.

Za razliku od prethodna dva tipa, gdje je čitav *dataset* labelisan ili labele ne postoje ni za jednu instancu podataka, kod **polunadgledanog učenja** je prisutna kombinacija ovog dvoga. Algoritmi polunadgledanog učenja predstavljaju hibrid prethodna dva tipa algoritama i oni koriste labelisane i nelabelisane podatke da bi generisali odgovarajuću funkciju ili klasifikator. Često je u praksi cijena labelisanja visoka, jer zahtijeva obučene stručnjake koji bi to radili manuelno. U situacijama kada ciljne vrijednosti postoje samo za određeni dio instanci podataka, dok je većina *dataset*-a nelabelisana, ovi algoritmi su najbolji kandidati za izgradnju modela.

Učenje potkrepljivanjem je tip učenja gdje ML algoritam (koji se često naziva agent) kontinualno uči iz svoje okoline. Ovo učenje se odvija na iterativan način, a u tom procesu agent uči iz svojih iskustava u okruženju dok ne istraži puni spektar mogućih stanja. Pojačano učenje omogućava mašinama i softverskim agentima da automatski odrede idealno ponašanje u određenom kontekstu, kako bi se maksimizirale njihove performanse. Algoritam uči kako postupiti uzimajući u obzir opservacije iz spoljašnjeg svijeta. Svaka radnja ima određeni uticaj na okolinu, a okolina daje povratne informacije koje usmjeravaju proces učenja i utiču na dalje akcije agenta. Neke od aplikacija ovakvog tipa ML algoritama su igranje šaha, robotska ruka, ili automobili bez vozača. Najpoznatiji algoritmi ovog tipa su: Q-učenje (eng. *Q-learning*), učenje vremenskih razlika (eng. *Temporal Difference learning — TD*) i duboke suparničke mreže (eng. *deep adversarial networks*).

Navedeni tipovi ML algoritama su prikazani na slici 2.1.



Slika 2.1 – Tipovi ML algoritama

3. *GRADIENT BOOSTING* ALGORITMI

Gradient boosting je ML tehnika za rješavanje klasifikacionih i regresionih problema koja koristi niz slabih predikcionih modela, najčešće stabla odlučivanja (eng. *Decision Tree* – DT)[15], kako bi kombinacijom predikcija svakog od pomenutih modela izračunala finalnu predikciju. Ovi slabi predikcioni modeli – tzv. slabi prediktori (eng. *weak predictors)* ili slabi "učenici" (eng. *weak learners*) su klasifikatori koji nisu snažno korelisani sa pravom klasifikacijom, odnosno daju predikciju koja je tek nešto bolja od nasumične[16] [11][19]. Ideja iza *gradient boosting*-a je da se kombinacijom mnogo slabih prediktora kreira jedan snažni prediktor i to kroz niz iteracija u kojima bi se ispravljale greške koje su napravili prethodni slabi prediktori [12].

Prvi dio imena ovih algoritama – *gradient* odnosi se na metodu gradijentnog spusta (eng. *gradient descent*) koja se koristi prilikom minimizacije *loss* (u literaturi se još navodi i *cost*) funkcije (funkcija greške koju prave ovi algoritmi u pokušaju da otkriju relaciju između ulaza i izlaza). Drugi dio – *boosting* odnosi se na način kojim se kroz niz iteracija ispravlja pomenuta greška. DT, *boosting* i metoda gradijentnog spusta kao osnovni elementi pomenutog algoritma su detaljno obrađeni u narednim sekcijama.

3.1. Stablo odlučivanja

DT je jedan od najbitnijih elemenata *gradient boosting* algoritama. Kao što je već rečeno, jedan DT nije dobar prediktor, ali kombinacijom više njih dobija se jedan od moćnijih alata u okviru ML-a. Ono što daje prednost DT-u je upravo njegova jednostavnost, kao i mogućnost vizualizacije donošenja odluka.

DT se sastoji od čvorova (eng. *nodes*) povezanih strelicama koje reprezentuju grane stabla (eng. *branches*). Čvor koji se dijeli na više podčvorova naziva se roditeljski čvor (eng. *parent node*), dok se ti podčvorovi nazivaju djeca ili čvorovi potomci (eng. *child nodes*). Prvi čvor u stablu se naziva korijen (eng. *root node*) i to je jedini čvor u stablu koji nema svoj roditeljski čvor. Čvorovi koji nemaju djece nazivaju se listovi stabla (eng. *leaf/terminal nodes*). Čvorovi koji se nalaze između korijena i listova nazivaju se interni ili unutrašnji čvorovi (eng. *internal nodes* ili samo *nodes*) – slika 3.1.

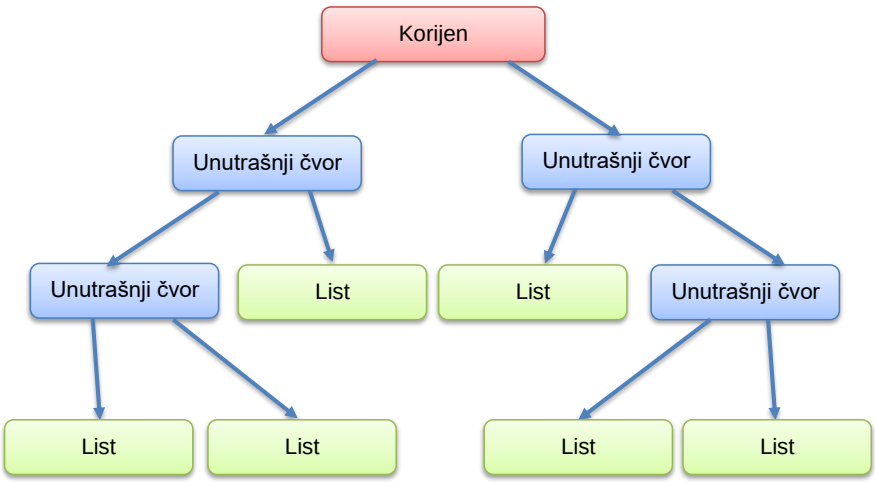
U tabeli 3.1 prikazan je pojednostavljen primjer *dataset*-a kakav se inače prosljeđuje kao ulaz u neke od ML algoritama. Svaki red u tabeli predstavlja jednog pacijenta za koga je prethodno utvrđeno da li ima bolest srca. Ovakvi redovi čine osnovnu gradivnu jedinicu bilo kojeg *dataset*-a. Jedan red predstavlja instancu podataka (eng. *data instance*), a još se nazivaju i stavke (eng. *data items*) ili uzorci (eng. s*amples*)[17]. Kolone "bol u prsima", "dobra cirkulacija" i "blokirane arterije" nazivaju se atributi (eng. *features*) i na osnovu njih se predviđa ciljna

[15] Za *gradient boosting* algoritme koji koriste DT kao slabe prediktore se često koristi skraćenica GBDT – *Gradient Boosting Decision Tree*.

[16] Određivanje pola na osnovu visine je primjer slabog prediktora ili slabog klasifikatora. Ukoliko se svi ljudi iznad 175cm klasifikuju kao muškarci, a svi ispod kao žene, jasno je da će u takvoj klasifikaciji biti mnogo grešaka, ali će ipak cjelokupna preciznost predikcije biti iznad 50%.

[17] Ukoliko se na redove u *dataset*-u misli kao na niz atributa bez ciljne vrijednosti, onda se često koristi termin primjeri (eng. e*xamples*). Jedan primjer X se definiše kao vektor atributa $X=(X_1, X_2, ..., X_n)$ gdje je n dimenzija vektora. Primjeri se mogu interpretirati kao tačke u n-dimenzionalnom vektorskom prostoru koji se naziva prostor primjera ili prostor instanci (eng. *instance space*).

vrijednost – bolest srca. Bitno je razlikovati dva tipa atributa: kategoričke ili kvalitativne i kontinualne ili kvantitativne [13]. Kategorički atributi imaju konačan broj kategorija između kojih može, a ne mora postojati zakon poretka. Tako, na primjer, pol osobe ili opština u kojoj osoba živi spadaju u kategoričke atribute koji nemaju definisan poredak i koji se nazivaju nominalni kategorički atributi, dok kod ocjena nekog filma od strane gledalaca postoji definisan poredak (jer je na primjer "dobar film" sigurno bolje od "grozan film", a lošije od "odličan film"), te se takvi atributi u skladu s tim nazivaju ordinalni. Sa druge strane, kontinualni atributi sadrže numeričke vrijednosti koje ukazuju na neku količinu. Ukoliko bi se *dataset*-u iz tabele 3.1 dodala visina pacijenta, to bi bio primjer kontinualnog atributa, dok su sva tri postojeća, primjeri posebnog tipa kategoričkog atributa koji se naziva dihotomni ili binarni, jer može da sadrži samo dvije vrijednosti (da ili ne). Jasno je da se ovdje radi o klasifikacionom problemu, a svaki primjer iz ulaznog *dataset*-a može se svrstati u jednu od dvije klase – klasa pacijenata koji imaju bolest srca i klasa onih koji nemaju, te se radi o binarnoj (*two-class*) klasifikaciji. Postoje klasifikacioni problemi gdje se primjeri klasifikuju u više od dvije klase i tada se radi o *multi-class* klasifikaciji.



Slika 3.1 – DT, nazivi čvorova

DT se formira tako što se u svakom čvoru koji nije list postavlja pitanje koje se odnosi na neki od pomenutih atributa (kaže se da svaki čvor predstavlja test nad atributom, a svaka grana ishod tog testa). U slučaju kategoričkih atributa, test obično predstavlja provjeru jednakosti sa nekom konkretnom vrijednošću atributa ili provjeru pripadnosti nekom skupu vrijednosti, dok kod kontiunalnih atributa tipičan test predstavlja provjeru da li je vrijednost atributa manja ili veća od neke prethodno definisane vrijednosti[18] [14]. Primjer takvog testa je provjera da li pacijent ima bol u prsima, pa ukoliko je odgovor potvrdan, pacijent se klasifikuje u onaj čvor do kojeg se stiže *true* granom, dok se u suprotnom pacijent klasifikuje u čvor na koji pokazuje *false* grana. Potom se u

[18] Ova prethodno definisana vrijednost je izračunata kao ona vrijednost za koju bi suma kvadratnih ostataka (eng. *Sum of Squared Residuals – SSR*) bila minimalna. Ostataci se dobijaju oduzimanjem stvarnih ciljnih vrijednosti i vrijednosti koje bi DT predvidio ukoliko bi se tu završila izgradnja stabla.

novim čvorovima postavlja iduće pitanje, koje se tiče nekog od idućih atributa i klasifikacija se nastavlja.

Tabela 3.1 – Pojednostavljen ulazni dataset koji sadrži podatke o pacijentima sa kardiovaskularnim problemima i njihovim simptomima

Bol u prsima	Dobra cirkulacija	Blokirane arterije	Bolest srca
Ne	Ne	Ne	Ne
Da	Da	Da	Da
Da	Da	Ne	Ne
Da	Ne	Da	Da
...	...	...	...
Da	Da	Ne	Ne

3.1.1. Mjere za određivanje nehomogenosti čvora

Prilikom kreiranja DT-a postavlja se pitanje šta uzeti kao korijen, odnosno od kojeg atributa krenuti u klasifikaciju. U inicijalnoj fazi, sve instance podataka se nalaze u korijenskom čvoru. Kako će oni dalje biti klasifikovani umnogome zavisi od redoslijeda izbora atributa po kojima će se vršiti klasifikacija. Zbog toga se za prvi atribut bira onaj koji donosi najveću saznajnu vrijednost, odnosno koji na najbolji način razvrstava instance. Idealno bi bilo da se pronađe takav atribut koji će kreirati potpuno "čiste" čvorove potomke u kojima se nalaze samo instance iz jedne klase. Kod problema binarne klasifikacije kao što je ovaj, to bi značilo da u jednom čvoru potomku budu samo pacijenti koji nemaju bolest srca, dok bi se u drugom nalazili samo oni koji pomenutu bolest imaju. Kako su u praksi ovakvi primjeri rijetkost (da nisu ne bi ni postojala potreba za DT-om), prvi atribut se bira tako da se u čvorovima potomcima ostvari što veća homogenost. U tu svrhu koriste se mjere za određivanje "nečistoće" čvora (eng. *impurity measures*). Dvije najčešće korištene mjere su entropija i *gini index* [14][15].

Commented [ZD9]: umnogome - spojeno

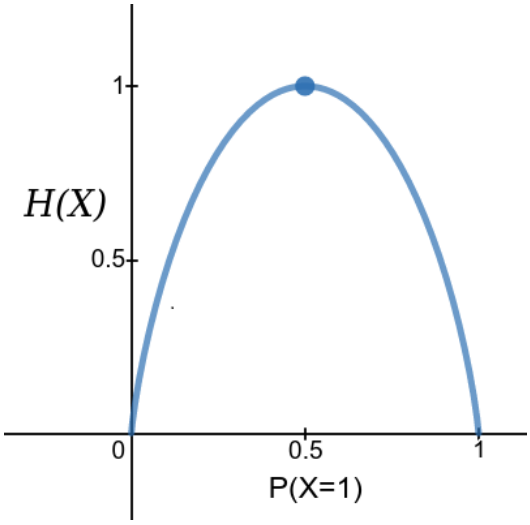
Entropija se u opštem slučaju, za C klasa ciljne vrijednosti, formalno definiše kao:

$$H(p_1, \dots, p_C) = -\sum_{i=1}^{C} p_i log_2 p_i \qquad (3.1)$$

gdje je p_i vjerovatnoća klase i, a C broj klasa (u posmatranom primjeru C = 2) [14].

Kao što se vidi iz formule (3.1), entropija je 0 ukoliko sve instance iz jednog čvora pripadaju istoj klasi, dok je najveća ukoliko su sve klase uniformo raspoređene nad tim instancama.

U posmatranom binarnom problemu, ukoliko se u jednom čvoru nađe 50% instanci iz jedne klase i 50% iz druge, tada je entropija: $-0.5log_2 0.5 - 0.5log_2 0.5 = 1$, dok u slučaju kada sve instance pripadaju istoj klasi $-1log_2 1 = 0$. To se može vidjeti i sa slike 3.2.



Slika 3.2 – Entropija – H(X)

Gini index (relacija 3.2) je, slično kao i entropija, maksimalan ukoliko su uzorci jednako raspoređeni po klasama, dok je jednak nuli, ukoliko je čvor potpuno homogen [14].

$$G(p_1, \dots, p_C) = 1 - \sum_{i=1}^{C} {p_i}^2 \tag{3.2}$$

Na osnovu ovih mjera određuje se ukupni informacioni dobitak (eng. *information gain* – *IG*) (termin koji se koristi kada se primjenjuje entropija), ili redukcija vrijednosti *gini index*-a koju odabrani atribut ostvaruje (termin koji se koristi kada se primjenjuje *gini index*), a koji u opštem slučaju ima oblik:

$$M(D) - \sum_{i=1}^{C} \frac{|D_i|}{|D|} * M(D_i) \tag{3.3}$$

gdje je *M* entropija ili *gini index*, *D* skup svih instanci u roditeljskom čvoru, a D_i skup svih instanci za koje je vrijednost atributa *A* jednaka *i* (odnosno skupovi instanci u čvorovima potomcima) [14].

IG sa entropijom je pristrasan prema testovima sa većim brojem ishoda, tj. radije selektuje atribute sa većim brojem vrijednosti. Stoga se uvodi i treća mjera – *gain ratio* (GR) što je zapravo

samo varijacija IG-a koja otklanja ovaj nedostatak i obezbjeđuje uniformnu raspodjelu vrijednosti atributa [15][16].

IG, GR ili redukcija *gini index*-a se izračunavaju u svakom koraku DT algoritma, kada god se vrši sljedeći test i bira idući atribut, te je zbog toga moguće da se na istom nivou u lijevoj i desnoj grani stabla nađu potpuno različiti atributi, jer je struktura čvorova na tim nivoima različita, te će i IG za iste atribut biti različit.

Gini index metrika je jednostavnija za implementirati i koristi se kod CART tipa DT-a, dok se IG sa entropijom koristi kod ID3, a GR kod C4.5 tipa stabla[19] [17].Veoma je teško odrediti koja od mjera daje bolje rezultate. Uz napomenu da je IG sa entropijom nešto sporiji zbog prisustva logaritmovanja, mnogi do sada objavljeni empirijski rezultati ukazuju na to da je nemoguće odrediti bolju mjeru, te da neslaganja između njih postoje u svega 2% slučajeva [18].

3.1.2. Uslov završetka izgradnje stabla

Prilikom generisanja DT-a polazi se od cjelokupnog ulaznog *dataset*-a, koji se u svakom koraku dijeli na manje podskupove i to tako što se u svakom čvoru bira jedan atribut[20] iz vektora svih atributa, koji ima najveći IG ili najbolju redukciju *gini index*-a, pa će kao takav najbolje klasifikovati podatke. Pomenuti podaci se dijele po tako odabranom atributu, kreirajući dva nova podskupa podataka (ili u opštem slučaju *n* novih podskupova) koja se smještaju u čvorove potomke. Ukoliko su ti čvorovi potpuno homogeni, proglašavaju se terminalnim. U protivnom se nad njima opet pronalazi najbolji atribut, te se klasifikacija nastavlja. Ovaj proces se ponavlja sve dok se ne potroše svi atributi, ne dostigne neki unaprijed definisani kriterijum kao što su broj listova, dubina stabla ili minimalni pomak u redukciji nehomogenosti, ili ukoliko se podjelom nekog čvora po najboljem od preostalih atributa povećava korištena mjera nečistoće skupa podataka (uzorci su bolje klasifikovani u trenutnom stanju nego što bi bili ako bi se odradio dodatni test). U idealnom slučaju proces izgradnje stabla se završava kada su svi listovi 100% čisti, odnosno u svakom listu se nalaze uzorci samo iz jedne klase [14][16].

3.1.3. Finalna predikcija

Nakon što se primjenom svih navedenih pravila kreira DT, isti se koristi za predikciju ciljne vrijednosti nekih novih instanci podataka. Tako na primjer, kada dođe novi pacijent, za kojeg još nije određeno da li ima bolest srca, podaci iz atributa za tog pacijenta se propuštaju kroz kreirano stablo, te on završava u nekom od listova stabla. Kod klasifikacionih problema, za određivanje finalne predikcije se koristi prosto pravilo većinske klase prisutnih instanci. Pacijent će biti klasifikovan u onu klasu koja preovladava u datom listu. S druge strane, u slučaju regresije ta vrijednost se izračunava kao aritmetička sredina ciljnih vrijednosti koje su pridružene datom listu [14].

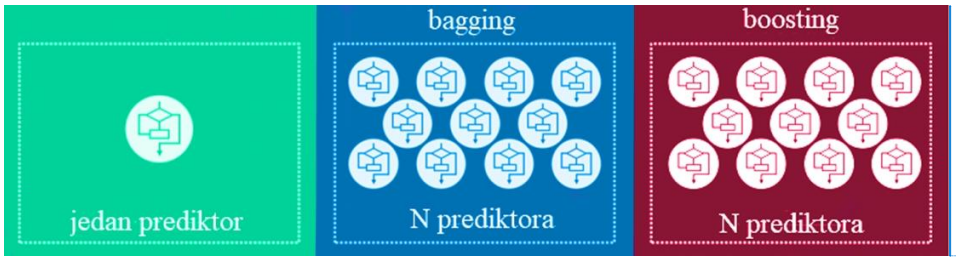
[19] Postoji veliki broj varijanti DT-ova, a navedena tri (ID3, C4.5 i CART) su najpoznatije implementacije ovog algoritma. CART (*Classification And Regression Trees*) je nastao 1984, ID3 (*Iterative Dichotomiser* 3) 1984, a C4.5 kao proširenje ID3 algoritma 1986. godine. Pored različite mjere homogenosti, međusobno se razlikuju i u tipovima podataka koje mogu obrađivati, načinu na koji rukuju vrijednostima koje nedostaju, broju testova koji se mogu vršiti u jednom čvoru stabla, ugrađenoj podršci za orezivanje stabla (eng. *pruning*) i brzini izvršavanja [17].

[20] Ovo je najčešći slučaj, ali postoje i neke varijante DT-ova koje omogućavaju testove nad više atributa odjednom, na primer ispitivanje vrijednosti neke njihove linearne kombinacije [14][17].

3.2. *Bagging* i *boosting*

Prethodno je navedeno da je DT slabi prediktor, te sam nije sposoban da dovoljno dobro klasifikuje podatke. Stoga se vrši kombinovanje više DT-ova, kreirajući tako ansambl – niz više algoritama za učenje, odnosno modela koje oni kreiraju, koji zajedno postižu bolje prediktivne performanse od onih koje bi postigao bilo koji od modela samostalno.

Bagging[21] i *boosting*[22] su ML tehnike za kreiranje ansambla. Obe tehnike pozivaju osnovni algoritam za učenje više puta, nad različitim skupom ulaznih podataka (nad različitim trening skupovima). Preciznije, ove tehnike kombinuju N slabih prediktora, u ovom slučaju DT-ova (slika 3.3), ali se kod svakog od njih uzima nasumičan podskup ulaznog *dataset*-a koji se uči (nad kojim se model trenira), te se potencijalno dobija N različitih modela. Zatim svaki od tih modela daje svoju predikciju, a finalna predikcija predstavlja kombinaciju predikcija tih N modela [19].



Slika 3.3 – Ansambl metode – bagging i boosting[23]

Kod *bagging*-a se svaki trening skup formira tako što se iz originalnog ulaznog *dataset*-a uniformno biraju instance podataka (u ovom kontekstu češće navođeni kao uzorci), gdje će svaki novokreirani skup biti iste veličine kao i originalni, ali je moguće da se isti uzorak u jednom skupu ponovi više puta, dok se neki uopšte neće pojaviti. Zbog načina odabira trening skupa, svi prediktori su potpuno nezavisni, te se kao takvi mogu trenirati paralelno, a finalna predikcija se dobija glasanjem (eng. *voting*) svih prediktora. Instanca podataka će biti one klase koju izglasa većina prediktora [20].

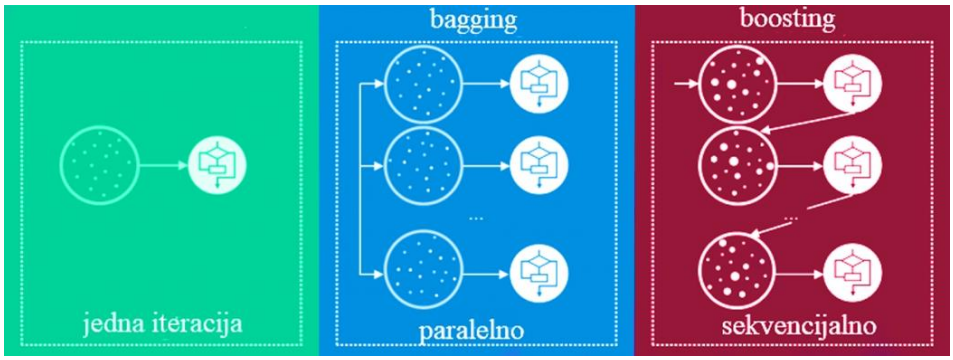
Kod *boosting*-a za razliku od *bagging*-a prediktori nisu nezavisni, već svaki sljedeći uči iz grešaka prethodnog. To se postiže tako što se daje veća težina onim instancama koje je prethodni prediktor loše klasifikovao, pa kad se bude birao podskup ulaznog *dataset*-a, instance sa većom težinom imaju veću šansu da ponovo budu odabrane. Jasno je da se u ovom slučaju prediktori ne mogu trenirati paralelno već sekvencijalno (slika 3.4) [20]. S obzirom na to da se prediktori fokusiraju na ispravljanje grešaka svojih prethodnika, do stvarne predikcije se dolazi u manje iteracija (potrebno je manje prediktora) nego kod *bagging*-a. Pored dodjeljivanja težine svakoj od pogrešno klasifikovanih instanci, težine se dodjeljuju i samim prediktorima odnosno treniranim modelima. Modelima koji daju bolje klasifikacione rezultate biće dodijeljena veća težina nego onima sa gorim rezultatima [21].

[21] Eng. *bootstrap aggregation* – "*bagging*"; Breimann, 1996.
[22] Adaboost familija algoritama – "*boosting*"; Freund & Schapire, 1996.
[23] Slika preuzeta sa https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Commented [AK10]: Tekstovi u fusnoti na nekim mjestima u Arial a na nekim TNR fontu. Izjednačite ih…

Commented [AK11]: Slike sa ovakvim bojama neće tako dobro izgledati na papiru ako štampanje ne bude u boji, čisto da imate na umu…

Istraživanja su pokazala da klasifikatori na kojima je primijenjena *bagging* ili *boosting* tehnika daju primijetno bolje rezultate od njihovih standardnih verzija [22]. Također, uporedna analiza ove dvije tehnike na velikom skupu različitih *dataset*-ova ukazuje da *boosting* smanjuje grešku klasifikacije u većem procentu nego što to čini *bagging*, što ga čini boljom tehnikom [21]. Ipak, iako u prosjeku bolji, *boosting* je na nekim *dataset*-ovima dovodio do značajne degradacije performansi, gdje bi u nekim slučajevima uvećavao klasifikacionu grešku i do 36%. Razlog tome je prisustvo šûma (eng. *noise*) u samim podacima[24], jer im *boosting* daje veću težinu u pokušaju da ih ispravno klasifikuje. U takvim situacijama *bagging* daje mnogo bolje rezultate [21].



Slika 3.4 –Paralelno procesiranje kod bagginga i sekvencijalno kod boostinga[25]

Iz imena *gradient boosting* algoritama se lako može zaključiti da koriste *boosting* tehniku tj. da se fokusiraju na ispravljanje grešaka kroz iteracije, što ih čini jednim od najuspješnijih ML algoritama. S druge strane, najpoznatiji predstavnik ML algoritma sa *bagging* tehnikom je *random forest* koji, kao slabe prediktore, koristi upravo DT-ove (otuda i naziv *random forest* – skup mnogo DT-ova koji koriste nasumično odabrane podskupove ulaznog *dataset*-a).

3.3. Gradijentni spust

Gradijentni spust predstavlja matematički temelj na kojem su izgrađeni *gradient boosting* algoritmi. Motivacija za njegovo uvođenje u svijet ML-a, kao i prednosti koje sa sobom donosi, biće razumljivija ukoliko se prvo razmotri jednostavan linearni prediktivni model.

[24] Neželjene instance podataka, atributi ili zapisi koji ne pomažu u objašnjavanju samih atributa ili veze između atributa i ciljne vrijednosti. Razlozi za pojavljivanje šuma su najčešće anomalije u podacima, nebitni atributi koji nisu u korelaciji sa ciljnom vrijednošću ili instance koji ne oslikavaju oblike i zakonitosti koje postoje u ostalim instancama iz posmatranog *dataset*-a.

[25] Slika preuzeta sa https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

3.3.1. *Regresija*

Neka je *D dataset* koji se sastoji od parova (*x*,*y*), gdje je *x* vektor vrijednosti atributa (u literaturi se navodi još i vektor dimenzija) $x = [x_1, x_2, x_3, \dots, x_n]^T$, a *y* skalarna ciljna vrijednost koja se pokušava predvidjeti. Instance podataka se mogu interpretirati kao tačke u *n*-dimenzionalnom vektorskom prostoru koji se naziva ulazni prostor (eng. *input space*) ili prostor instanci (eng. *instance space*). Neka je χ skup svih mogućih instanci. Na temelju *dataset*-a $D = \{x^{(i)}, y^{(i)}\}$ potrebno je naučiti nepoznatu funkciju $f : \chi => \mathbb{R}$ tako da, idealno, $y^{(i)} = f(x^{(i)})$. Takva funkcija ima oblik:

$$f(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_0 = \sum_{i=1}^{n} w_ix_i + w_0 = w^Tx + w_0 \qquad (3.4)$$

gdje su w_i, $i=0,..,n$ parametri koje treba naučiti na osnovu skupa primjera *D*, i koji mogu biti prikazani kao w^T — vektor težina. Budući da vrijednost *f(x)* linearno zavisi od ulazne vrijednosti *x*, ova prediktivna metoda se naziva linearna regresija [23].

Potrebno je odrediti parametre w_i tako da funkcija (3.4) na najbolji način opisuje stvarnu zavisnost između *x* i *y*, odnosno tako da je greška predikcije minimalna. To se postiže tako što se definiše *loss* funkcija, te se uzmu oni parametri za koje je vrijednost te funkcije minimalna.

Kod linearne regresije se najčešće koristi metoda najmanjih kvadrata (eng. *least squares*) [23] gdje se uzima SSR kao *loss* funkcija.[26]

$$loss = \sum_{i=1}^{n} [y^{(i)} - f(x^{(i)})]^2 = \sum_{i=1}^{n} [y^{(i)} - (w^Tx^{(i)} + w_0)]^2 \qquad (3.5)$$

Kako prvi izvod neke funkcije u tački *x* daje koeficijent pravca tangente te funkcije u toj tački, funkcija će imati minimum u onoj tački u kojoj je tangenta paralelna sa x osom, odnosno u kojoj je koeficijent *k* jednak nuli. Iz toga slijedi da se rješavanjem jednačine $loss'(w) = 0$ dobijaju oni parametri w_i za koje će greška modela biti minimalna.

3.3.2. *Metoda gradijentnog spusta*

U nekim slučajevima nije moguće analitički odrediti parametre w_i, ili je izračunavanje ovih parametara previše složeno. U tim slučajevima koristi se metoda gradijentnog spusta.

Za razliku od linearne regresije, kod gradijentnog spusta nije potrebno određivati kada je koeficijent pravca *loss* funkcije jednak nuli. Optimalne vrijednosti parametara se određuju tako što se inicijalne vrijednosti odaberu nasumično, a zatim se kroz niz koraka vrijednosti modifikuju, krećući se u smjeru nagiba *loss* funkcije, sve dok se ne dođe do minimalne vrijednosti. Koraci, odnosno vrijednosti za koje će parametri biti modifikovani (eng. *learning step*), su u početku veći, da bi se približavanjem minimalnoj vrijednosti smanjivali, kako bi se što preciznije odredile tražene vrijednosti prametara za koje će pomenuta funkcija imati svoj minimum (slika 3.5).

Najčešći kriterijumi za zaustavljanje algoritma su dosezanje unaprijed određenog broja iteracija ili stagnacija u promjeni vrijednosti greške.

Commented [ZD12]: Pogledati prvi komentar

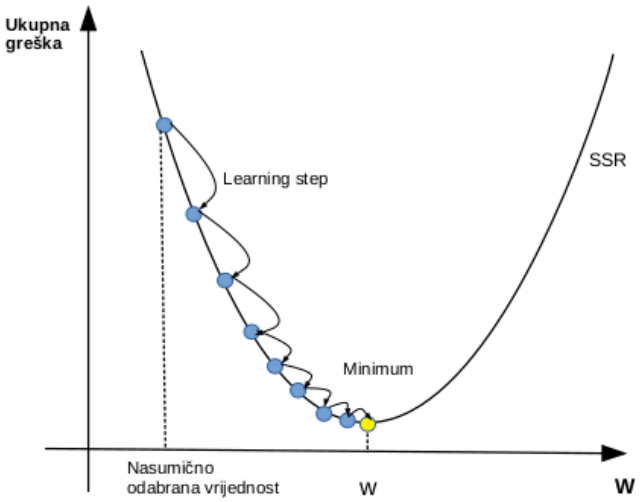
[26] Često se naziva *Mean Squared Error – MSE*, Važno je napomenuti da postoje i mnoge druge *loss* funkcije kao što su *Mean Absolute Error – MAE*, *logistic loss* ili *log loss*, *hinge loss* itd. [24].

3.3.3. Matematička formulacija

Neka je *Z* ulazni *dataset* koji se sastoji od instanci z_i, i=1,2,…,n. Svaka instanca *z* je uređena dvojka (*x*, *y*) koja se sastoji od proizvoljnog ulaza *x* i skalarnog izlaza (ciljne vrijednosti) *y*. Neka je definisana *loss* funkcija $\int(\hat{y}, y)$, gdje je $\hat{y}$ predviđena, a *y* stvarna vrijednost i familija $\mathbb{F}$ funkcija $f_w(x)$ parametrizovana težinskim faktorom *w*. Potrebno je pronaći funkciju $f_w \in \mathbb{F}$ koja minimizuje grešku predikcije $Q(z, w) = \int(f_w(x), y)$ na instancama *z*.

$$w_{t*1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{3.6}$$

gdje je *t* broj iteracije, γ adekvatno izabran *learning rate*[27], a $\nabla_w Q(z_i, w_t)$ gradijent funkcije Q odnosno *loss* funkcije $\int$ [25]. Gradijent funkcije više promjenljivih je vektorsko polje čije su komponente parcijalni izvodi funkcije po svakoj promjenljivoj, u ovom slučaju $\nabla_w Q(z_i, w_t) = (\frac{\partial Q}{w_0}, \frac{\partial Q}{w_1}, \ldots, \frac{\partial Q}{w_n})$ [26]. S obzirom na to da se u svakoj iteraciji *t* preduzimaju koraci proporcionalni negativnom gradijentu ∇Q ažuriranjem vrijednosti parametara w_i, čime se spušta ka minimalnoj vrijednosti *loss* funkcije, ova metoda je i dobila naziv metoda gradijentnog spusta, a po njoj i porodica *gradient boosting* algoritama.



Slika 3.5 –Metoda gradijentnog spusta

Commented [AK13]: Tekstovi ispod slika, tabela itd treba da imaju manji font, npr. 10

[27] Broj kojim se skalira dobijena vrijednost gradijenta kako bi se uticalo na veličinu koraka kojim se težinski faktor *w* mijenja, idući ka vrijednosti za koju *loss* funkcija ima svoj minimum. *Learning rate* je najčešće dat u vidu parametra koji se može proizvoljno mijenjati kako bi *loss* funkcija brže ili sporije konvergirala ka svom minimumu.

3.3.4. Stohastički gradijentni spust

U prethodno opisanom algoritmu gradijentni vektor se izračunava ukupno za sve primjere iz skupa za učenje. Zbog toga se taj algoritam često naziva i grupni gradijentni spust (eng. *batch gradient descent*). Alternativa je stohastički gradijentni spust (eng. *stochastic gradient decent*), koji predstavlja drastično pojednostavljenje gdje se gradijent izračunava na bazi jednog nasumično izabranog primjera z_t u t-oj iteraciji. Na taj način se zapravo dobija aproksimaciju stvarnog vektora gradijenta.

$$w_{t*1} = w_t - \gamma_t \nabla_w \mathrm{Q}(z_t, w_t) \tag{3.7}$$

Pretpostavka je da će se (3.7) ponašati kao (3.6) uprkos šumu uvedenom zbog pojednostavljenja procedure [27].

Za razliku od grupnog gradijentnog spusta, stohastički gradijentni spust u praksi je manje računski zahtjevan, što ga čini bržim i (kod funkcija koje nisu konveksne) manje podložan zaglavljivanju u lokalnom minimumu [27].

3.4. *Gradient boosting* algoritam kao cjelina

Gradient boosting algoritam osmislio je i razvio Jerome H. Friedman početkom 1999. godine, na temelju ideje Lea Breimana, prema kojoj se *boosting* može posmatrati kao optimizacioni algoritam odgovarajuće *loss* funkcije [28]. U uvodnom dijelu ove glave data je uopštena definicija ovog algoritma, a u idućem paragrafu razmotreni su detalji implementacije u vidu niza koraka kroz koje algoritam iterativno prolazi. S obzirom na to da se pomenuti koraci malo razlikuju u zavisnosti od toga da li je u pitanju regresioni ili klasifikacioni problem, ova dva slučaja posmatraće se odvojeno.

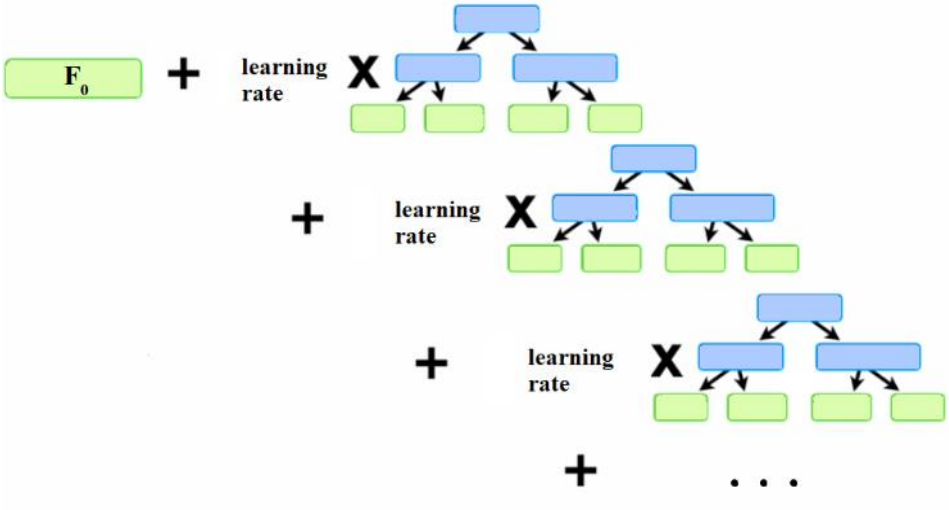
3.4.1. Regresioni problemi

Kao što je već rečeno, regresioni problemi su oni kod kojih je ciljna vrijednost koja se pokušava predvidjeti kontinualna. *Gradient boosting* te probleme rješava tako što izgradi svojevrsnu linearnu kombinaciju slabih prediktora (najčešće DT-ova), kod kojih svaki idući DT pokušava ispraviti grešku koju su napravila prethodna stabla. Na kraju, sistem takvih stabala predviđa traženu ciljnu vrijednost.

Prvi korak u rješavanju je određivanje početne vrijednosti F_0 za koju će SSR biti najmanji za ulazni *dataset*. To je zapravo aritmetička sredina svih ciljnih vrijednosti iz *dataset*-a. Vrijednost F_0 je inicijalna predikcija i ako bi se algoritam ovdje zaustavio, predvidio bi F_0 kao ciljnu vrijednost za svaku instancu iz ulaznog *dataset*-a. Naravno, algoritam se ne zaustavlja ovdje, već izračunava ostatak između predviđene i stvarne ciljne vrijednosti za svaku od instanci. Nakon toga se kreira stablo, koje će koristiti atribute iz ulaznog *dataset*-a da predvidi ostatak, a ne stvarnu ciljnu vrijednost[28]. Stablo se kreira na način na koji je to objašnjeno u sekciji 3.1. Novokreirano stablo će svaku od instanci iz ulaznog *dataset*-a klasifikovati u neki od listova, a predikcija koju stablo daje za te instance će biti aritmetička sredina svih ostataka koje su završile u tom listu.

[28] Na ovaj način se zapravo predviđaju greške u predikciji ciljne vrijednosti koje se zatim iterativno ispravljaju.

Nakon toga se dobijena predikcija skalira sa *learning rate*-om, te se potom sabira sa inicijalnom vrijednošću iz prvog koraka – F_0, dajući novu predikciju za ciljnu vrijednost[29]. Na ovaj način se ispravlja greška i u duhu metode gradijentnog spusta čini korak u pravom smjeru ka stvarnoj ciljnoj vrijednosti. Zatim se ponovo računaju ostaci u odnosu na novu predikciju, te ponovo kreira stablo sa novim ostacima kao ciljnim vrijednostima. Ovaj proces se iterativno ponavlja, ispravljajući pri tom greške prethodnih stabala, sve dok se ne dosegne unaprijed određeni broj iteracija ili se dogodi stagnacija u ispravljanju ukupne greške (slika 3.6). Bitno je napomenuti da raspored atributa u svakom idućem stablu može biti drugačiji u odnosu na prethodna.



Slika 3.6 – Gradient boosting algoritam — grafički prikaz

[29] Razlog zašto se predikcija novog stabla skalira sa *learning rate*-om leži u tome što prema Friedmanu emprijski dokazi ukazuju na to da preduzimanje mnogo malih koraka daje bolje rezultate sa novim podacima koje algoritam još uvijek nije vidio, odnosno koji se ne nalaze u ulaznom *dataset*-u.

Na slici 3.7 su izloženi koraci implementacije u vidu matematičke formulacije algoritma [12].

Commented [ZD14]: Prvi komentar...

Ulaz: Dataset $\{(x_i, y_i)\}_{i=1}^{n}$ i diferencijabilna loss funkcija $L(y_i, F(x))$

Korak 1: Inicijalizovanje modela sa konstantnom vrijednošću:

$$F_0 = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

Korak 2: za m=1,2,...,M:

(A) Izračunati $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ i=1,...,n

(B) Izgraditi regresiono stablo sa r_{im} kao ciljnim vrijednostima i kreirati terminalne regije (listove) R_{jm}, gdje je j=1,2,...,J_m

(C) Za j=1,2,...,J_m izračunati $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Ažurirati $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x_i \in R_{jm})$

Korak 3: Izlaz $F_M(x)$

Slika 3.7 – Gradient boosting algoritam – pseudo kod

Diferencijabilna *loss* funkcija koja se najčešće koristi kod regresionih problema data je relacijom (3.8)[30] [29].

$$L(y, F) = \frac{(y - F)^2}{2} \tag{3.8}$$

Pokazalo se da je za ovu *loss* funkciju aritmetička sredina zaista ona vrijednost koja minimizuje sumu navedenu na slici (3.7) u koraku 1, te se ona dodjeljuje F_0 kao što je ranije navedeno. Takođe, rješavanjem jednačine iz koraka 2.C, dobija se da je vrijednost γ_{jm} koja minimizuje datu sumu u tom koraku zapravo aritmetička sredina svih vrijednosti ostataka koje su završile u listu (terminalnoj regiji) R_{jm}. Finalno, suma iz koraka 2.D znači da je potrebno sabrati sve vrijednosti γ_{jm} za sve listove R_{jm} u kojima se može naći primjer x. Kako u ovom slučaju x može završiti samo u jednom listu, ovaj korak nije ništa drugo do ažuriranje predikcije sa novom vrijednošću γ_{jm} koja predstavlja korak u pravom smjeru ka finalnoj ciljnoj vrijednosti y. Grčko slovo ν (ni) je već pomenuti *learning rate*.

[30] Kvadrat greške se dijeli sa dva zbog pojednostavljenog računa prilikom vađenja izvoda.

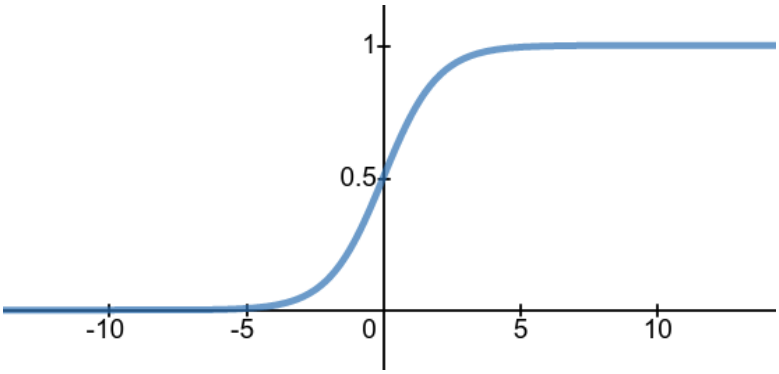
3.4.2. Klasifikacioni problemi

Kod klasifikacionih problema se primjeri iz ulaznog *dataset*-a klasifikuju u dvije ili više klasa. Navedeni generalizovani koraci [12] *gradient boosting* algoritma sa slike 3.7 važe za oba tipa problema, ali će se njihovi ishodi kao i *loss* funkcija koja se koristi, razlikovati. S obzirom na to da je sam algoritam za klasifikacione probleme veoma sličan kao i za regresione, proces nije objašnjen od nule, već je fokus samo na razlikama.

Kako kod klasifikacionih problema ciljna vrijednost y obično uzima binarne vrijednosti $y=\{0,1\}$[31] (binarna klasifikacija, y pripada nekoj klasi – 1, ili ne pripada – 0), pretpostavlja se da dolazi iz Bernulijeve distribucije, te se stoga *loss* funkcija koja se koristi u ovom slučaju često naziva Bernulijeva *loss* funkcija [30] ili negativni *log-likelihood* i ima oblik:

$$L(y,F) = \log(1 + e^{-2yF})$$ [32] [29] (3.9)

gdje je $y=\{-1,1\}$.



Slika 3.8 – Sigmoidna funkcija

Iako su svi koraci navedeni u prethodnom paragrafu identični za klasifikacione i regresione probleme, moguće je da se krajnji rezultat nekog od koraka razlikuje zbog činjenice da se koristi drugačija *loss* funkcija. Tako na primjer, u prvom koraku, vrijednost F_0 u ovom slučaju neće biti aritmetička sredina svih ciljnih vrijednosti iz ulaznog *dataset*-a, već logaritam omjera vjerovatnoća da će nasumično odabrani uzorak pripadati jednoj, odnosno drugoj klasi (eng. *log of the odds*). Zaista se pokazuje da je za *loss* funkciju iz relacije (3.9), vrijednost γ koja najbolje minimizuje datu *loss* funkciju upravo pomenuti log(*odds*).

[31] češće zapisuje kao y={-1,1} radi pojednostavljene notacije

[32] Izračunavanje vjerovatnoće pripadnosti primjera nekoj klasi najbolje se može optimizovati minimizacijom negativnog *log-likelihood*-a. U statistici postoji metoda poznata kao procjena maksimalne vjerovatnoće (eng. *maximum likelihood estimation* - *MLE*) kojom se procjenjuju parametri raspodjele vjerovatnoće maksimizovanjem funkcije vjerovatnoće, tako da su po pretpostavljenom statističkom modelu uočeni podaci najvjerovatniji. Kako je ipak u pitanju *loss* funkcija, gdje su manje vrijednosti bolje, MLE prelazi u minimizaciju negativnog *log-likelihood*-a. Otuda minus ispred 2*yF*.

S obzirom na to da ciljna vrijednost mora biti kategorička, a DT daje kontinualne vrijednosti, potrebno je rezultat provući kroz neku funkciju koja će vratiti ciljnu vrijednost u binarnom obliku. To se postiže tako što se rezultat iz bilo koje iteracije propusti kroz sigmoidnu funkciju (poznatu i kao logistička funkcija – koristi se kod logističke regresije) (relacija 3.10). Graf funkcije je prikazan na slici 3.8. Sada se zavisno promjenljiva *y* kreće u intervalu [0,1], pa se može postaviti prag (najčešće 0.5)[33] za koji se kaže da sve vrijednosti iznad njega pripadaju datoj klasi — 1, a u skladu s tim, sve ispod ne pripadaju — 0.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.10}$$

Commented [ZD15]: Bilo bi dobro dodati još jedan pasus nakon ovog i prokomentarisati ovaj prag, odnosno njegovu vrijednost... Svakako, ovo treba da bude u skladu s ovim što se pominje u sekciji 4.5. Takođe, moguće je ovdje, umjesto ovog dodatnog pasusa, navesti da će ovaj prag biti detaljnije objašnjen kasnije (u sekciji 4.5)...

[33] Pomenuti prag, kao i uticaj njegove promjene na konačne rezultate algoritma, detaljnije je objašnjen u sekciji 6.3.3.

4. ML ALGORITMI – DODATNI KONCEPTI

4.1. *Overfitting* i *underfitting*

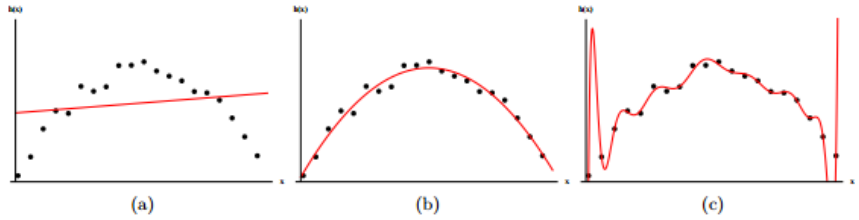
Cilj bilo kojeg ML algoritma je da uoči određene zakonitosti koje postoje u podacima, da prepozna šablone ponašanja, te nauči veze i zavisnosti između podataka, kako bi to znanje iskoristio u predikciji specifikovane ciljne vrijednosti za neke nove podatke koje ranije nije vidio[34]. Pretpostavlja se da u tim novim podacima važe iste zakonitosti kao i u onima iz kojih je to znanje stečeno. Na osnovu stečenog znanja ML algoritam gradi model koji se potom koristi u predikcijama ciljne vrijednosti (npr. pripadnost klasi) novih podataka.

Međutim, u nekim slučajevima izgrađeni model nije dovoljno složen da bi opisao sve zavisnosti koje postoje između podataka, dok je u drugim model posvetio previše pažnje učenju detalja te je naučio neke zavisnosti koje su specifične samo za taj ograničeni *dataset* iz kojeg se uči, ali ne i za podatke generalno.

Prvi slučaj se naziva *underfitting*. Algoritam nije posvetio dovoljno vremena učenju podataka, odnosno izgradnji modela, ili je model naprosto previše jednostavan. Ukoliko se podaci između kojih ne postoji veza linerane zavisnosti pokušaju predstaviti pravom linijom, jasno je da ona neće moći obuhvatiti sve veze koje postoje između podataka, odnosno neće dobro modelovati podatke.

Sa druge strane, *overfitting* je pojava prenaučenosti, kada generisani model skoro u potpunosti odgovara određenom skupu podataka, te zbog toga ne uspijeva pouzdano predvidjeti buduće opservacije. Drugim riječima, model ne generalizuje[35] dobro.

Na slici 4.1 prikazani su *underfit*-ovani, *overfit*-ovani i optimalni model za podatke generisane regresijom funkcije $y = -x^2 + \varepsilon$. Jasno je da obična linearna regresija 4.1(a) ne može dovoljno dobro opisati postojeću zakonitost, dok je polinom visokog stepena 4.1(c) tačno opisao podatke, ali će kod predikcije za neke nove primjere x prouzrokovati znatno veću grešku nego što bi to učinio optimalni model 4.1(b).



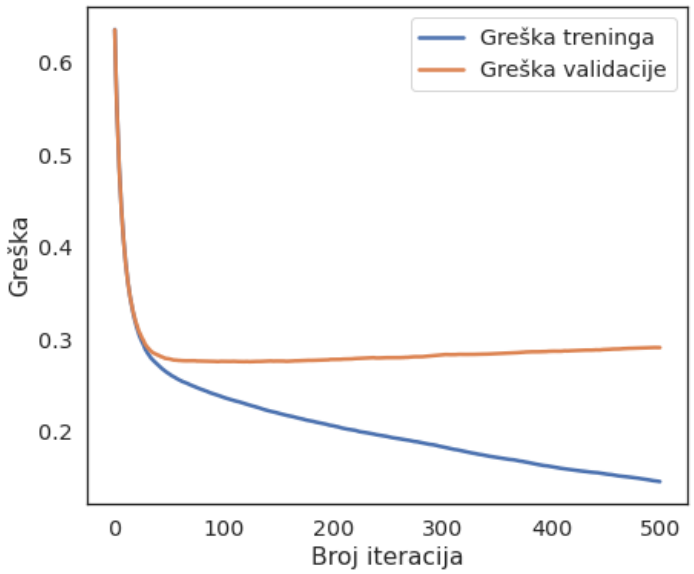
Slika 4.1 – Regresija funkcije $y = -x^2 + \varepsilon$: (a) underfitting (linearna regresija), (b) optimalan model (regresija polinomom drugog stepena), (c) overfitting (regresija polinomom 15. stepena[36]

[34] Ne postoje u ulaznom *dataset*-u.
[35] Generalizacija je sposobnost modela da daje razumne predikcije za skupove ulaza koje ranije nikad nije vidio.
[36] Slika preuzeta iz knjige "Strojno učenje" autora Jana Šnajdera i Jovane Dalbelo Bašić.

4.2. Trening, validacioni i testni skup podataka

S obzirom na pomenute probleme sa *underfitting*-om i *overfiting*-om i da prosječna greška predikcije ne govori puno o tome kako će se model ponašati u dodiru sa novim neviđenim podacima, potreban je neki mehanizam za određivanje ispravnosti modela. Zbog toga se cjelokupni *dataset* kojim se raspolaže dijeli na tri dijela, i to na jedan veći – skup za treniranje i dva manja – validacioni i testni skup.

Na trening skupu se kao što mu ime kaže, trenira model, odnosno odatle algoritam uči veze i pravilnosti koje postoje u podacima. Testni skup služi za provjeru koliko dobro model generalizuje. U njemu se nalaze novi podaci različiti od onih nad kojima je algoritam učio. Validacioni skup se koristi za evaluaciju modela tokom njegove izgradnje. U procesu treniranja modela vrši se kontinualna minimizacija *loss* funkcije, čime greška nad trening skupom postaje sve manja. U takvim uslovima postoji bojazan od pojave *overfitting*-a, pa je potrebno paralelno pratiti grešku i na validacionom *dataset*-u i u momentu kada se ona počne povećavati prekinuti treniranje. Kod mnogih ML algoritama moguće je precizirati tačan broj iteracija nakon kojih se, ukoliko greška na validacionom skupu kontinualno raste, prekida treniranje. Više riječi o ovome dato je u praktičnom dijelu rada (glava 6). Na slici 4.2 vidljivo je kako se i trening i validaciona greška smanjuju do određene tačke, nakon koje greška validacije počinje da raste. To je trenutak kada se potencijalno javlja *overfitting* i kada je potrebno prekinuti treniranje.



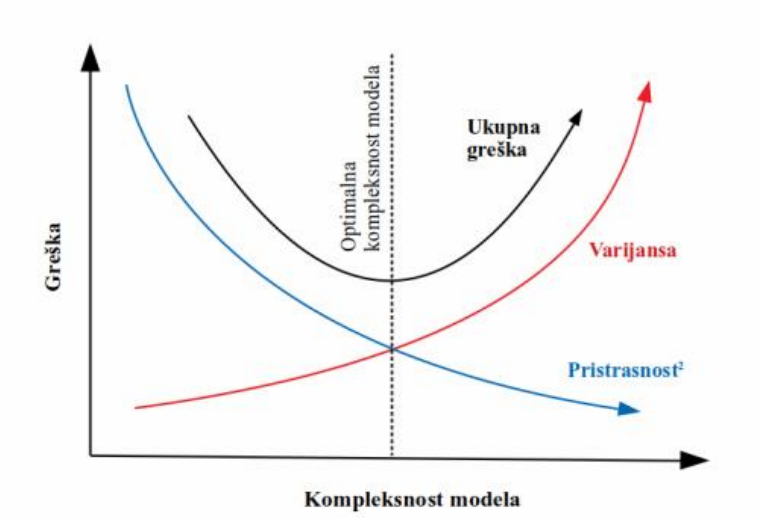
Slika 4.2 – Validaciona i trening greška u odnosu na broj iteracija kod gradient boosting algoritma

4.3. Pristrasnost i varijansa

Pristrasnost (eng. *bias*) i varijansa (eng. *variance*) su termini koji su usko vezani za *overfitting* i *underfitting*. Obe pojave prouzrokuju grešku u predikciji, ali do te greške dolaze iz različitih razloga. Pristrasnost je razlika između prosječne predikcije modela i prave vrijednosti koju pokušava predvidjeti. To je zapravo nemogućnost nekog jednostavnog modela da modeluje pravu vezu između ulaza (primjera) i ciljne vrijednosti. Modeli koji imaju visoku pristrasnost nisu sposobni da nauče kompleksne zakonitosti koje vladaju među podacima, bez obzira na to koliko se trudili (kao na prethodnom primjeru sa slike 4.1 gdje linearna regresija ne može modelovati postojeću zavisnost ulaza *x* od izlaza *h(x)*, bez obzira na to koji se koeficijent pravca i odsječak na *y* osi odabere tokom treniranja).

Sa druge strane, varijansa je greška koja nastaje zbog varijabilnosti predikcija modela na različitim *dataset*-ovima. Drugim riječima, za neke skupove podataka model će biti gotovo idealan, dok će za druge biti veoma loš. Razlog tome je što se model previše prilagodio podacima nad kojim je treniran, odnosno došlo je do *overfitting*-a.

ML algoritmi koji imaju visoku pristrasnost, kao što su linearna i logistička regresija, daju osrednje, ali konstantne predikcije na različitim *dataset*-ovima (stabilni klasifikatori). Algoritmi sa visokom varijansom, kao što su DT-ovi, neuronske mreže ili SVM daju vrlo nekonzistentne predikcije, te, ako bi se izgradnja modela ponovila više puta, vjerovatno bi se dobili rezultati raštrkani oko prave ciljne vrijednosti, ali su "u prosjeku tačni" (nestabilni klasifikatori) [31]. Da bi se izgradio dobar model neophodno je naći pravi omjer između pristrasnosti i varijanse koji će minimizovati ukupnu grešku, kao na slici 4.3. Zbog toga u praksi uvijek postoji tzv. *bias-variance trade-off*, gdje se smanjivanjem jednog povećava drugi i obratno [32].



Slika 4.3 – Bias-variance trade-off

4.4. Priprema podataka

Priprema podataka (eng. *data preparation*) ili pretprocesiranje podataka (eng. *data preprocessing*) je proces transformacije sirovih podataka (eng. *raw data*) kako bi se oni mogli iskoristiti kao ulaz za neki od ML algoritama. Ovo je proces na koji u prosjeku odlazi više od pola vremena utrošenog za rad na nekom ML projektu. Većina ML algoritama zahtijeva strukturirane podatke, pa je, stoga, podatke neophodno prikupiti i organizovati prema atributima, kao što je to urađeno u tabeli 3.1. Međutim, i tako strukturirani *dataset*-ovi često zahtijevaju dodatnu obradu. Neki od razloga za to su:

- Nepotpuni zapisi i nedostajuće vrijednosti atributa,
- Anomalije,
- Nekonzistentne vrijednosti i nedostatak standardizacije,
- Loše performanse modela i potreba za inženjeringom atributa (eng. *feature engineering*),
- Transformacija kategoričkih atributa,
- *Oversampling* i *undersampling*.

4.4.1. Nepotpuni zapisi i nedostajuće vrijednosti atributa

Nedostajuće vrijednosti (eng. *missing values)* su jedan od najčešćih problema koji se javljaju u procesu pripreme podataka. Više je potencijalnih razloga koji dovode do ovog problema, a neki od njih su korumpirani podaci, greška pri učitavanju podataka, nepotpuna ekstrakcija i sl. Uzimajući ovo u obzir, teško je očekivati da će svaki zapis u *dataset*-u sadržati vrijednosti svakog od atributa. Kada takvih vrijednosti nema na njihovom mjestu se najčešće nalazi prazna ćelija, fiksne vrijednosti kao što su NULL i N/A, ili poseban karakter, kao na primjer upitnik „?“. Odabir načina na koji će se rukovati ovim podacima predstavlja veliki izazov s kojim se analitičari susreću, a važnost donošenja dobre odluke po ovom pitanju se ogleda u robusnosti izgenerisanog modela. Jedan od najjednostavnijih načina je brisanje redova sa takvim vrijednostima, ali ovo je moguće uraditi samo ukoliko je procenat takvih redova veoma nizak, jer će u protivnom doći do velikog gubitka informacija. Drugi način je zamjena vrijednosti koja nedostaje sa aritmetičkom sredinom, medijanom[37] ili modusom[38] svih vrijednosti tog atributa koje postoje u posmatranom *dataset*-u. Dobra strana ovog načina je što ne dovodi do gubitka informacija, ali treba biti oprezan, jer se ovime uvodi pristrasnost, a kod velikih *dataset*-ova bi računanje ovih vrijednosti nad čitavim *dataset*-om moglo biti vremenski zahtjevno. Ukoliko se radi o kategoričkim atributima, elegantno rješenje je zamjena praznih polja novom kategorijom, u slučaju čega će algoritam tu kategoriju posmatrati na isti način kao i bilo koju drugu. Još neki od načina su predviđanje nedostajućih vrijednosti na osnovu nekog drugog atributa koji takve vrijednosti nema ili korištenje ML algoritama koji imaju ugrađenu podršku za rukovanje pomenutim vrijednostima, kao što je to slučaj sa većinom *gradient boosting* algoritama (XGBoost, LightGBM, CatBoost).

[37] Medijana se u teoriji vjerovatnoće i statistici opisuje kao broj koji razdvaja gornju polovinu uzoraka od donje. Drugim riječima, to je vrijednost koja zauzima centralnu poziciju kada se sve opservacije sortiraju po rastućem ili opadajućem redoslijedu [13].

[38] Modus je vrijednost koja se pojavljuje najčešće u podacima [13].

4.4.2. Anomalije

Anomalije ili izuzeci (često se nazivaju *outliers*) su neočekivane vrijednosti koje se mogu pojaviti u *dataset*-u i koje leže na abnormalnoj udaljenosti u odnosu na vrijednosti preostalog dijela podataka. Uzrok nastanka anomalija unutar *dataset*-a može biti različit, a često se javljaju ukoliko se radi o podacima iz nepoznatih izvora sa slabom kontrolom valjanosti podataka. Kada se ovakve anomalije detektuju, neophodno je izvršiti njihovu analizu, te na osnovu nje donijeti odluku o potencijalnom uklanjanju detektovanih izuzetaka iz *dataset*-a. Ukoliko su anomalije nastale zbog greške u mjerenju ili unosu podataka, potrebno je iste ispraviti, ili, ukoliko to nije moguće, poptuno ih ukloniti iz *dataset*-a. U situacijama kada su anomalije prirodni dio fenomena koja se posmatra, njih tada nije moguće ukloniti jer bi to dovelo do gubitka informacija i iskrivljenosti rezultata.

4.4.3. Nekonzistentne vrijednosti i nedostatak standardizacije

Nedostatak standardizacije je još jedan od čestih problema zbog kojeg je neophodno pripremiti podatke prije nego se oni proslijede nekom ML algoritmu. Često se dešava da se semantički iste vrijednosti zapisuju na različite načine, pa se tako nazivi država nekad mogu pisati njihovim dvoslovnim ili troslovnim oznakama, a nekad punim imenom. U takvim slučajevima je potrebno proći kroz čitav *dataset* i sve takve vrijednosti zamijeniti jedinstvenim nazivom kako ih algoritam ne bi posmatrao kao potpuno odvojene kategorije.

4.4.4. Potreba za inženjeringom atributa

Inženjering atributa obuhvata konstrukciju novih varijabli i dodavanje novih atributa u postojeći *dataset* kako bi se poboljšale performanse ML modela. Ove performanse često veoma zavise od reprezentacije vektora atributa (eng. *feature vector)*, što je još jedan od razloga zašto se ulaže toliko vremena u pripremu i transformaciju podataka. Stoga se ovaj vektor proširuje dodatnim varijablama nastalim najčešće kao produkt kombinacija i matematičkih transformacija postojećih atributa, čime se ostvaruje dublje razumijevanje podataka [33]. Kako se odabir načina na koji će se sprovoditi ovaj proces mahom zasniva na intuiciji, a manje na egzaktno definisanim pravilima, s pravom se kaže da je inženjering atributa više umjetnost nego nauka.

4.4.5. Transformacija kategoričkih atributa

Iako postoje ML algoritmi koji mogu obrađivati kategoričke vrijednosti kao takve, bez dodatnih transformacija (kao npr. CatBoost), za većinu ostalih algoritama neophodno je naći način da se takve vrijednosti konvertuju u numeričke. Ponekad je dovoljno svakoj kategoričkoj vrijednosti samo dodijeliti jedinstvenu *integer* vrijednost (kao što je slučaj sa LightGBM-om). Međutim, većina ML algoritama nema ugrađen mehanizam za rukovanje kategoričkim vrijednostima konvertovanim u numeričke vrijednosti, već bi to posmatrali kao običnu kontinualnu vrijednost za koju postoji pravilo poretka. Time bi ovakvi algoritmi zaključili da su one kategorije sa većim brojevima bolje u odnosu na one sa manjim, što je vjerovatno pogrešno. Stoga se nekad pribjegava i drugim tehnikama.

Kada je kardinalnost skupa kategorija mala (kao npr. za dane u sedmici ili ograničenu paletu boja), moguće je kreirati novi atribut za svaku kategoriju i indeksirati svaki od atributa.

Ovakav tip mapiranja se naziva rječnik [34]. Svaka kategorija se tada predstavlja vektorom, koji ima jedinicu na mjestu odgovarajućeg atributa, a nule na svim ostalim, pa se dalje ovi vektori koriste u treniranju modela. Ova tehnika je poznatija kao *one-hot-encoding.* Problem sa ovom tehnikom je veličina vektora i zauzeće memorije. Jedan od načina da se ovaj problem ublaži je *Out of Vocab* tehnika (OOV) gdje se sve one kategorije koje se veoma rijetko pojavljuju svrstavaju pod jedan atribut, što potencijalno smanjuje veličinu samog vektora, ne gubeći pri tom previše informacija [34]. Još neke od mogućih tehnika su heširanje (eng. *hashing*), hibridni pristup između *hash*-iranja i rječnika i *embeddings*[34]. Posljednje predstavlja distribuiranu reprezentaciju kategoričkih vrijednosti, gdje se semantički slične kategorije preslikavaju u vektore koji se nalaze na relativno maloj udaljenosti u njihovom vektorskom prostoru. Ova tehnika potiče iz NLP-a i originalno je razvijena za distribuiranu reprezentaciju riječi, gdje vektor u sebi sadrži semantiku same riječi.

Commented [AK16]: heširanje (eng. hashing)

4.4.6. Oversampling i undersampling

U praksi se često dešava da je jedna klasa ciljnih vrijednosti dominanta u odnosu na ostale, odnosno da se pojavljuje u većini zapisa u nekom *dataset*-u. Ovakvi *dataset*-ovi se nazivaju nebalansirani i često mogu dovesti do ignorisanja manjinskih klasa od strane ML algoritma, što je poseban problem jer su tipično upravo manjinske klase one čije su predikcije najvažnije. *Oversampling* i *undersampling* su dva pristupa za jednostavno rješavanje ovog problema. *Oversampling* uključuje umnožavanje zapisa koji sadrže manjinsku klasu i dodavanje kopija nastalih umnožavanjem u trening skup. Sa druge strane, *undersampling* podrazumijeva nasumično biranje zapisa sa većinskom klasom i njihovo izbacivanje iz trening skupa. U oba slučaja se postiže uravnoteženost prethodno nebalansiranog *dataset*-a, ali to može da prouzrokuje i negativne posljedice. Tako na primjer, *oversampling* povećava vjerovatnoću pojave *overfitting*-a s obzirom na to da klasifikator donosi pravila na osnovu višestrukih kopija istog zapisa, pa iako se čini da doneseno pravilo važi za veliki broj slučajeva, zapravo važi za nekoliko izolovanih primjera i kao takvo ima malu moć generalizacije. Kod *undersampling*-a, sa druge strane, postoji problem gubitka velike količine informacija, jer je u praksi omjer zapisa sa manjinskom i većinskom klasom reda 1:100, 1:1000 ili čak 1:10000 [35]. Ipak, i pored tog gubitka, istraživanja (provedena nad C4.5 algoritmom) su pokazala da, iznenađujuće, *undersampling* daje značajno bolje rezultate od *oversampling*-a koji često uopšte ne utiče značajno na performanse ovog ML algoritma [36].

4.5. Mjere za evaluaciju performansi modela

Nakon što ML algoritam kreira model, potrebno je ispitati valjanost tog modela, što se postiže njegovom provjerom na testnom *dataset*-u kao što je to opisano u paragrafu 5.2. Predviđanja koja model kreira za svaku instancu potrebno je uporediti sa stvarnim ciljnim vrijednostima iz testnog skupa i izvući sumarnu informaciju o tome koliko dobro je model predvidio te ciljne vrijednosti. Za to se koriste mjere za evaluaciju performansi modela, kao što su tačnost (eng. *accuracy*), preciznost (eng. *precision*), odziv (eng. *recall*) i F1 skor (eng. *F1-score* ili *F1-mean*).

Ukoliko se radi o problemu binarne klasifikacije, kao što je to slučaj sa *dataset*-om u tabeli 3.1, instance mogu spadati u dvije klase (odgovor na pitanje da li pacijent ima bolest srca može biti pozitivan ili negativan). Svi slučajevi gdje je model uspješno predvidio pozitivan odgovor

vode se kao stvarno pozitivni (eng. *true positive – TP*), a slučajevi gdje je uspješno predvidio negativan odgovor vode se kao stvarno negativni (eng, *true negative – TN*). Sličnom logikom, svi oni za koje model predviđa pozitivan odgovor, a zapravo su negativni, nazivaju se lažno pozitivni (eng. *false positive – FP*), dok su slučajevi gdje model kaže da nemaju bolest srca, a oni zapravo imaju, lažno negativni (eng. *false negative – FN*).

Tačnost (t) je najintiutivnija mjera od četiri navedene i predstavlja odnos broja slučajeva kada je model dobro predvidio ciljnu vrijednost i ukupnog broja predviđanja. Formalno:

$$t = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \tag{4.1}$$

Iako veoma intuitivna, tačnost nije uvijek adekvatna mjera za evaluaciju modela, iz prostog razloga što daje podjednaku važnost i pozitivnim i negativnim opservacijama. Ako bi *dataset* iz tabele 3.1 sadržao podatke o 1000 pacijenata od kojih samo 3 imaju bolest srca, a model predvidio da nema niko (*TP* = 0, *TN* = 997), tačnost takvog modela je 99.7% što je veoma visok procenat. Ipak, od 3 bolesna pacijenata model nije uspio da prepozna nijednog, što znači da je uspješnost prepoznavanja bolesnih pacijenata ovog modela 0%. Iz ovog se jasno vidi zašto se pored tačnosti moraju koristiti i druge mjere evaluacije, pogotovo kod veoma nebalansiranih *dataset*-ova.

Preciznost (p) je odnos dobro predviđenih pozitivnih opservacija i ukupnog broja pozitivnih predviđanja. Drugim riječima – od svih slučajeva kada je model vratio pozitivan odgovor, koliko od njih je stvarno pozitivnih?

$$p = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4.2}$$

Odziv (r) (negdje se navodi i senzitivnost) je odnos ispravno predviđenih pozitivnih opservacija i ukupnog broja stvarno pozitivnih instanci (čija je stvarna ciljna vrijednost pozitivna). U ovom slučaju – od svih stvarno bolesnih pacijenata, koliko njih je model prepoznao kao bolesne?

Commented [ZD17]: Možda ke bolje korititi oznaku R, jer su i ostale oznake potekle od eng. termina.

$$r = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.3}$$

U nekim slučajevima preciznost ima veću saznajnu vrijednost, dok je u drugim odziv logičniji izbor kao adekvatna mjera za evaluaciju performansi. Na primjer, kod sistema za detekciju i filtriranje *spam mail*-ova se ne bi smjelo desiti da se neki validan *mail* označi kao *spam*. Cijena takve greške je mnogo veća nego da se za neki *spam mail* pogrešno predvidi da je negativan, pa su ovakvi sistemi veoma osjetljivi na *FP* i u takvim slučajevima preciznost ima mnogo veću važnost nego odziv koji u svojoj formuli ne sadrži *FP*. Sa druge strane, kod predviđanja prevara u transakcijama ili različitih bolesti cijena pogrešnog klasifikovanja transakcije ili pacijenta kao negativnih (*FN*) je ekstremno visoka, pogotovo ako može dovesti do širenja zaraze ili smrtnog ishoda pacijenta, pa je u takvim slučajevima odziv pravi izbor.

F1 skor (f1) je harmonijska sredina preciznosti i odziva, odnosno:

$$f1 = 2 * \frac{\mathrm{p} * \mathrm{r}}{\mathrm{p} + \mathrm{r}} \tag{4.4}$$

F1 skor se koristi kada je potrebno naći balans između preciznosti i odziva i, za razliku od tačnosti, mnogo manje je osjetljiva na nejednaku raspodjelu ciljnih vrijednosti, odnosno klasa koje se predviđaju.

5. ANALIZA NAJPOZNATIJIH IMPLEMENTACIJA *GRADIENT BOOSTING* ALGORITAMA

5.1. AdaBoost

AdaBoost (skraćeno od *Adaptive Boosting*) je prvi praktični *boosting* algoritam. Algoritam su razvili Yoav Freund i Robert Schapire 1996. godine, za šta su nagrađeni Gödelovom nagradom[39]. Iako jedan od najstarijih *boosting* algoritama, AdaBoost se i dalje masovno izučava, te ima široku primjenu u mnogim poljima, a na IEEE konferenciji o *data mining*-u iz 2006. godine, proglašen je jednim od 10 najboljih algoritama svih vremena u ovoj oblasti [37].

AdaBoost je meta-algoritam koji se danas smatra specijalnim slučajem *gradient boostinga* zbog određenih razlika u samom algoritmu. Primjera radi, *gradient boosting* nedostatke slabih prediktora rješava korištenjem gradijenta *loss* funkcije, čime preduzima neophodne korake ka minimizaciji iste, dok se ti nedostaci kod AdaBoost-a rješavaju dodjeljivanjem različitih težina svakoj od instanci podataka, gdje će pogrešno klasifikovane instance imati daleko veću težinu i samim tim će se na njih obratiti veća pažnja u idućoj iteraciji u cilju njihove ispravne klasifikacije.

5.1.1. Koraci implementacije AdaBoost algoritma

Kao slabe prediktore AdaBoost koristi stabla koja se sastoje samo od korijena i dva lista. Ovakva stabla se nazivaju ”panjevi” (eng. *stumps*). Jasno je da će tačnost predikcija ovakvih stabala biti veoma mala, nekad čak i manja od 50%, što AdaBoost elegantno rješava dodjeljivanjem težina svakom od prediktora i to na način da se *stump*-ovima, čija je preciznost preko 50%, dodjeljuje pozitivna težina, a onima ispod te vrijednosti – negativna. Finalnu predikciju donose svi *stump*-ovi, ali bitniju ulogu imaju oni sa većom preciznošću, odnosno većom dodjeljenom težinom.

Proces počinje tako što se za svaku instancu iz ulaznog *dataset*-a $\{(x_i, y_i)\}_{i=1}^{m}$, gdje $x_i \epsilon X$ i $y \epsilon Y = \{-1,1\}$ inicijalizuje težina $D_1(i) = \frac{1}{m}$, za i=1,...,m [38][39]. Težine su inicijalno iste za svaku instancu i njihova suma je jednaka jedinici. Nakon toga se trenira slabi prediktor, odnosno kreira se *stump*, tako što se odabere onaj atribut koji će najbolje klasifikovati instance iz *dataset*-a, što se može odrediti putem *gini index*-a na način opisan u trećem poglavlju. Za istrenirani slabi prediktor se potom računa greška ε_t, kao suma svih težina uzoraka koji su pogrešno klasifikovani, odnosno:

$$\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_{t(i)} \tag{5.1}$$

[39] Gödelova nagrada se dodjeljuje na godišnjem nivou za izvanredne radove u oblasti teorijske računarske nauke.

gdje je h_t izlazna vrijednost slabog prediktora, a *t* njegov redni broj. Potom se, na osnovu greške, određuje težina pomenutog slabog prediktora, što je prikazano relacijom (5.2)[40] [38][39].

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{5.2}$$

Iz grafika ove funkcije, prikazanog na slici 5.1, jasno je da će precizniji *stump* imati veću težinu pri donošenju odluke, dok će *stump* sa preciznošću od 50% imati težinu nula. Za *stump* koji većinu uzoraka klasifikuje pogrešno, odnosno konstantno daje predikciju suprotnu od istinite, težina će biti velika negativna vrijednost. Ovo pratkično znači da će ovaj *stump* i dalje imati veliku ulogu u donošenju finalne odluke, ali će zbog minusa glasati suprotno odluci koju je samostalno donio (na primjeru *dataset*-a iz tabele 3.1, ukoliko *stump* koji konstantno griješi kaže da pacijent ima bolest srca, negativna težina će okrenuti njegovu odluku u "pacijent nema bolest srca").

Pošto je sada poznato koje su instance pogrešno klasifikovane, moguće je ažurirati njihove težine, na način prikazan relacijom (5.3).

$$D_{t+1}(i) = \frac{D_t e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \tag{5.3}$$

gdje je Z_t normalizacioni faktor, odabran tako da bi D_{t+1} bila distribucija, odnosno da bi zbir svih težina bio jednak jedan [38][39]. Jasno je da će, ukoliko je instanca podataka ispravno klasifikovana ($h_t(x_i) = y_i$) i preciznost slabog prediktora veća od 50% ($\alpha_t > 0$), nova težina biti manja, dok će u slučaju pogrešne klasifikacije, ili ispravne klasifikacije prediktora koji konstantno griješi, nova težina biti znatno veća. Nakon ovog koraka kreira se novi *dataset* (trening skup idućeg *stump*-a), biranjem instanci iz postojećeg, posmatrajući pri tom težine kao distribuciju. Instance sa većom težinom imaju veću šansu da budu odabrane. Moguće je da se isti uzorak više puta nađe u novokreiranom *dataset*-u. Ukoliko se takav uzorak ponovo pogrešno klasifikuje, greška koju on unosi će ovoga puta biti znatno veća, kreirajući tako veću "kaznu" za slabog prediktora koji ga nije uspio dobro klasifikovati. Na ovaj način se insistira na ispravnoj klasifikaciji uzoraka i ispravljanju greške prethodnih slabih prediktora, što je i cilj *boosting*-a.

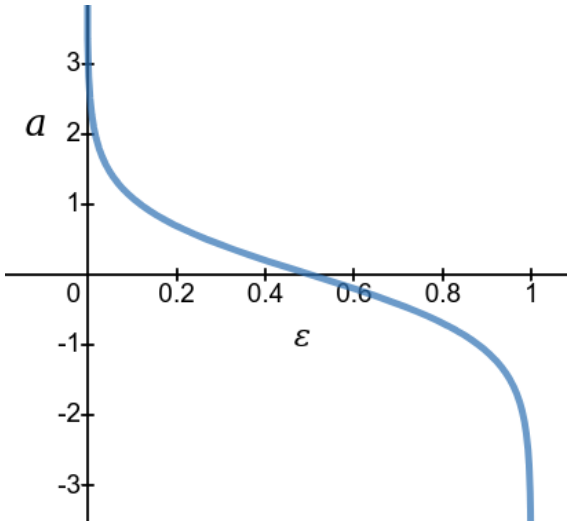
Finalna predikcija je data u vidu sume svih prediktora, uzimajući u obzir njihove težine, što je prikazano relacijom (5.4) [38][39].

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{5.4}$$

Relacija (5.3) se još naziva i finalna ili kombinovana hipoteza [39], koja izračunava znak težinske kombinacije slabih hipoteza $h_t(\mathrm{x})$, odnosno slabih prediktora (5.5).

[40] Ukoliko su sve instance iz *dataset*a dobro klasifikovane, greška je jednaka nuli, $\varepsilon_t = 0$. Takođe, ukoliko su sve instance pogrešno klasifikovane, greška je jednaka jedinici, $\varepsilon_t = 1$. Kako relacija 5.2 nije definisana za te vrijednosti greške, ε_t mora biti u intervalu (0,1), odnosno $0 < \varepsilon_t < 1$. Da bi se osiguralo da će ε_t zaista biti u tom intervalu, u praksi se u ova dva slučaja vještački dodaje veoma mala vrijednost greške za ε_t.

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{5.5}$$



Slika 5.1 – Odnos težine α i greške klasifikacije ε

5.1.2. Minimizacija loss funkcije

Mnoge ML metode koje su danas u upotrebi mogu se posmatrati kao procedure minimzacije *loss* funkcije. Iako originalno nije dizajniran za tu svrhu, AdaBoost algoritam se takođe alternativno može posmatrati na taj način. Ovakvo viđenje algoritma pomoglo je u pojašnjavanju cilja algoritma, dokazivanju njegovih konvergencijskih osobina, kao i razvoju novih algoritama nastalih kao produkt generalizacije AdaBoost-a i njegovog osposobljavanja za nove izazove. Upravo to je otvorilo vrata novim *gradient boosting* algoritmima kao jednim od najvažnijih algoritama u oblasti ML-a.

AdaBoost se može razumijeti kao postupak minimizacije eksponencijalne *loss* funkcije prikazane relacijom (5.6)

$$\frac{1}{m}\sum_{i=1}^{m} e^{-y_i F(x_i)} \tag{5.6}$$

gdje je F(x) dato relacijom (5.5) [39].

5.2. XGBoost

XGBoost je DT bazirani ansambl ML algoritam koji koristi *gradient boosting framework*. Razvijen je kao istraživački projekat Univerziteta u Washingtonu, te objavljen 2014. godine. Od svog objavljivanja postao je veoma popularan u ML svijetu, te našao široke primjene u mnogim oblastima koji se zasnivaju na obradi strukturiranih podataka. Širok spektar aplikacija ovog algoritma obuhvata regresione, klasifikacione, probleme rangiranja te korisnički definisane probleme. Odlikuju ga visoke performanse u pogledu preciznosti i tačnosti predikcija, te relativno kratko vrijeme izvršavanja. Razlog široke primjene XGBoost-a upotpunjuje i njegova portabilnost, jer radi na Windows, Linux i OS X operativnim sistemima, te činjenica da podržava najpopularnije programske jezike, kao što su Java, Python, R, C++, Scala i Julia.

XGBoost za slabe prediktore koristi DT-ove CART tipa, primjenjujući na njima *boosting* principe, gdje se kroz niz iteracija greška svakog od slabih prediktora pokušava svesti na minimum. U tom procesu XGBoost uvodi niz noviteta, poboljšanja i sistemskih optimizacija čime direktno utiče na brzinu i performanse samog algoritma.

5.2.1. Prevencija overfitting-a

U cilju prevencije *overfitting*-a urađena su poboljšanja u regularizaciji *loss* funkcije. Na postojeću *loss* funkciju se dodaje regularizacioni faktor koji kažnjava kompleksnost modela.

$$Loss = \sum_i \int(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \tag{5.7}$$

$$\Omega(\mathrm{f}) = \gamma T + \frac{1}{2}\lambda||w||^2 \tag{5.8}$$

gdje je *k* redni broj iteracije (odnosno stabla), *T* broj listova a *w* skor ili težina svakog od listova [40]. Dodatni regularizacioni faktor pomaže da se izglade finalne naučene težine i izbjegne *overfitting*. Ovakva regularizovana *loss* funkcija teži da izabere jednostavniji model koji će i dalje davati predikcije sa visokom tačnošću, ali će lakše generalizovati na novim podacima.

XGBoost takođe uvodi niz hiperparametara (eng. *hyperparameters*)[41] koje korisnik može definisati u cilju kontrole složenosti modela i izbjegavanja *overfiting*-a. Jedan od najbitnijih je definisanje maksimalne dubine stabla, preko koje algoritam neće prelaziti prilikom izgradnje modela [40].

XGBoost posjeduje ugrađenu unakrsnu validaciju (eng. *cross-validation*)[42] za svaku iteraciju, čime je uklonjena potreba za njenim eksplicitnim programiranjem i specifikovanjem tačnog broja iteracija u cilju dobijanja optimalnih rezultata. Algoritam će se sam zaustaviti kad primijeti da se greška validacije ne popravlja nakon određenog broja iteracija [40].

[41] Potrebno je praviti razliku između parametara i hiperparametara modela. Parametar je interni za model i njegova vrijednost se može izračunati iz podataka. Hiperparametar je eksterna konfiguracija koju korisnik prosljeđuje algoritmu i njegova vrijednost se ne može izračunati iz podataka.

[42] Unakrsna validacija je tehnika evaluacije modela kojom se procjenjuje njegova moć generalizacije. Ova validacija se često vrši u više rundi, tako što se u svakoj od njih uzima različit podskup podataka iz validacionog skupa, nakon čega se izračunava prosječna greška. Ovime se dobija preciznija procjena prediktivnih performasni modela.

5.2.2. Algoritmi za pronalaženje najboljeg split-a

Weighted quantile sketch i *sparsity-aware split finding* su dva algoritma koje XGBoost koristi za pronalaženje najboljeg načina za podjelu kontinualnih atributa, odnosno pronalaska one vrijednosti (u daljem tekstu – *split*-a) u odnosu na koju će se vršiti test u čvoru stabla kao što je to opisano u paragrafu 3.1. Ovi algoritmi spadaju u najveća poboljšanja uvedena XGBoost-om.

Prilikom izgradnje DT-a, u svakom njegovom čvoru se određeni atribut testira, a na osnovu tog testa se instance dijele u potomke tog čvora. Međutim, ukoliko se radi o kontinualnoj vrijednosti, postavlja se pitanje kako odrediti granicu, pa da sve instance sa vrijednošću atributa manjom od *split*-a idu u jedan čvor, a one sa većom u drugi. Standardni algoritam (poznat i kao *pre-sorted* algoritam) bi prvo sortirao sve instance po vrijednostima posmatranog atributa, a potom bi za svaku računao koliki je tom slučaju SSR. Ovaj proces je vremenski veoma zahtjevan, te zbog toga XGBoost uvodi *weighted quantile sketch* kao svojevrsnu aproksimaciju tog osnovnog algoritma za pronalaženje *split*-a. *Weighted quantile sketch* pronalazi kandidate za *split* na osnovu raspodjele vrijednosti atributa, a zatim, po pronađenim vrijednostima, dijeli instance u tzv. *bin*-ove (algoritam zasnovan na metodi histograma (eng. *histogram-based algorithm*)). Nakon toga se računa SSR samo za kandidatske *split*-ove i bira onaj *split* sa minimalnom vrijednošću SSR-a koji će shodno tome prouzrokovati najmanju grešku.

Sparsity-awarness je još jedna moćna odlika XGBoost-a. U problemima iz realnog svijeta podaci koji se prikupljaju ponekad nisu potpuni i vrijednosti nekih atributa često nedostaju. Veoma je važno da algoritam bude toga svjestan. Ukoliko vrijednost atributa za neki uzorak nedostaje, na koju stranu klasifikovati instance u čvoru u kojem se postavlja pitanje za taj atribut? Kako bi se ovi problemi riješili, XGBoost predlaže dodavanje podrazmijevanog pravca klasifikacije za svaki od čvorova. Optimalni pravac klasifikacije se uči iz preostalih podataka u *dataset*-u. Bez obzira na razlog izostanka vrijednosti, XGBoost sve šablone razuđenosti podataka uči na uniforman način. *Sparsity-awarness* je značajno ubrzao vrijeme izgradnje modela. Eksperimenti su pokazali da je za neke *dataset*-ove brzina *sparsity-aware* XGBoost algoritma preko 50 puta veća u odnosu na njegovu osnovnu verziju [40].

5.2.3. Optimizacija dizajna sistema

Pored već navedenih poboljšanja i noviteta u implementaciji samog algoritma, XGBoost uvodi niz optimizacija dizajna sistema s ciljem povećanja brzine izračunavanja i smanjenja vremenske i računske složenosti algoritma.

Jedna takva optimizacija je uvođenje paralelizacije. Iako *boosting* princip podrazumijeva sekvencijalnu izgradnju DT-ova, oduzimajući time mogućnost paralelizacije tog procesa, moguće je paralelizovati određene faze izgradnje samog stabla. Jedan od najzahtjevnijih dijelova ovog algoritma je sortiranje podataka kako bi se nad njima tražio najbolji *split*. Kako bi se smanjio trošak sortiranja, XGBoost čuva podatke u *in-memory* strukturama koje se nazivaju blokovi. Podaci su razbijeni po kolonama, gdje je svaka kolona sortirana po vrijednostima odgovarajućih atributa. Blokovi se kreiraju samo jednom, prije ulaska u proces treniranja, te se kao takvi iznova koriste u svakoj iteraciji. Jedno skeniranje ovakvog bloka je dovoljno da se prikupe sve statistike o kandidatima za *split*. XGBoost paralelizuje ovaj proces prikupljanja statistika, čime se automatski dobija paralelizovan algoritam za pronalaženje najboljeg *split*-a, što značajno utiče na cjelokupnu složenost.

Cache-aware access je još jedna u nizu optimizacija koja omogućava algoritmu da bude svjestan količine CPU keša i shodno tome odabere veličinu bloka tako da iskorištenost resursa

bude optimalna. Tako na primjer, ukoliko su blokovi previše mali, iskorišćenost procesora će biti niska, što dovodi do neefikasne paralelizacije. Sa druge strane, previše veliki blokovi neće moći stati u keš procesora, rezultujući velikim brojem keš promašaja. Ovo je važno iz razloga što uprkos sortiranim podacima u blokovima, pristup memoriji za dohvatanje statistika za određivanje najboljeg *split*-a nije kontinualan, što takođe utiče na vrijeme izvršavanja algoritma. Pokazano je da implementacija *cache-aware* pristupa uzrokuje da algoritam radi duplo brže nego verzija kod koje to nije implementirano [40].

Pored procesora i memorije, potrebno je optimizovati i druge dijelove sistema, kao što je disk na kojem se skladište blokovi koji su previše veliki da stanu u memoriju. Ulazno-izlazne operacije često traju duže od samih izračunavanja pa se u cilju ubrzanja algoritma na njih treba obratiti posebna pažnja. XGBoost za poboljšanje *out-of-core* izračunavanja korist dvije tehnike: *block compression* i *block shrading*. Prvom se smanjuje prostor koji blok zauzima na disku čime se dio procesnog vremena utrošenog na dekompresiju daje u zamjenu za kraće vrijeme upisa/čitanja sa diska, dok se drugom podaci dijele na više diskova (ukoliko su takvi diskovi dostupni) povećavajući tako protočnost podataka prilikom čitanja sa diska.

Slika 5.2 – XGBoost optimizacije osnovnog gradient boosting algoritma[43]

5.2.4. Priprema kategoričkih atributa

Bitno je napomenuti da XGBoost nema ugrađen mehanizam za rukovanje kategoričkim atributima, pa ne postoji opcija da se putem posebnog hiperparametra proslijede redni brojevi kolona sa kategoričkim vrijednostima. Stoga je takve atribute u cilju njihove pravilne obrade prije prosljeđivanja algoritmu potrebno na neki način enkodovati. To se najčešće radi *one-hot encoding*-

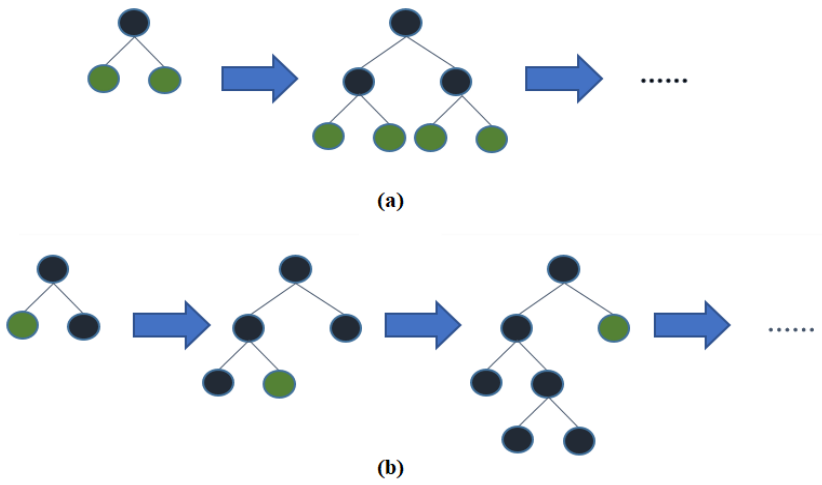
[43] Slika preuzeta sa https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.

om kako je to objašnjeno u sekciji 4.4.5, ili nekim drugim načinom transformacije u numeričke vrijednosti, uz dozu opreza, jer XGBoost može pretpostaviti da između njih postoji ordinalna zavisnost, što za nominalne kategoričke vrijednosti svakako nije slučaj. Na ovaj problem je također skrenuta pažnja u pomenutoj tački.

5.3. LightGBM

LightGBM je još jedna implementacija *gradient boosting* algoritama, razvijena od strane Microsoft-a i objavljena početkom 2017. godine. Značajan porast količine podataka koja se svakodnevno generiše uticao je na to da i *dataset*-ovi postaju sve veći i složeniji, kako u pogledu količine instanci tako i u pogledu broja atributa. Stoga se javila potreba za još bržim i skalabilnijim algoritmom, što je dovelo do pojave LightGBM-a. Otuda i ovaj prefiks *light* s obzirom na veliku brzinu izvršavanja i njegovu manju memorijsku zahtjevnost.

LightGBM je nastao u vrijeme kada se XGBoost već afirmisao kao najbolji iz ove porodice algoritama i ubjedljivo najpopularniji algoritam na mnogim takmičenjima [41]. Stoga je XGBoost korišten kao polazna tačka na koju je LightGBM nadograđen unoseći određena poboljšanja. Ova poboljšanja nisu išla u smjeru povećanja preciznosti algoritma, već smanjenja njegove računske, memorijske i vremenske složenosti. Cilj je bio da se drastično smanji vrijeme izvršavanja uz očuvanje iste ili veoma slične preciznosti, što je i urađeno [42].



Slika 5.3 – (a) level-wise i (b) leaf-wise rast stabla[44]

LigtGBM sadrži sve inovacije uvedene XGBoost-om, pa tako i on koristi metodu histograma da grupiše vrijednosti kontinualnih atributa u diskretne *bin*-ove, te koristi specijalni

[44] Slika preuzeta iz LightGBM dokumentacije, https://lightgbm.readthedocs.io/en/latest/Features.html.

algoritam za pronalaženje najboljeg *split*-a između tih *bin*-ova, čime se složenost smanjuje sa O(broj_instanci) na O(broj_*bin*-ova). Takođe, LightGBM izvlači maksimum iz paralelizacije pronalaženja najboljeg *split*-a i to na nivou kolona (atributa), ali i redova, jer omogućava dijeljenje *dataset*-a na više cjelina, gdje se svaka cjelina obrađuje paralelno, kreirajući histogram za svaku do njih. Zatim se ti pojedinačni histogrami spajaju u globalni histogram koji se kasnije koristi prilikom procesa izgradnje stabla.

LightGBM stablo gradi prema *leaf-wise* pristupu, za razliku od mnogih postojećih *gradient boosting* algoritama koji to rade *level-wise*[45]. Kod *leaf-wise* pristupa, LightGBM pronađe onaj list koji će najviše smanjiti *loss* i vrši podjelu samo po tom listu, dok ostale listove iz tog nivoa ostavlja netaknute. Sa druge strane, *level-wise* pristup dijeli svaki list iz trenutnog nivoa, čime se grade simetrična stabla, dok je rezultat *leaf-wise* strategije stablo koje je asimetrično (slika 5.3). Zbog ovoga se *loss* brže smanjuje, ali se lakše događa *overfitting*, pa se ne preporučuje korištenje LightGBM-a kod malih *dataset*-ova [43].

Iako je XGBoost-ov algoritam za pronalaženje najboljeg *split*-a značajno smanjio vrijeme treniranja modela, efikasnost i skalabilnost su još uvijek bili nezadovoljavajući. Na njih je najviše uticao uvećan broj instanci i atributa u *dataset*-ovima. Javila se potreba za smanjivanjem njihovog broja, ali na optimalan način koji neće dovesti do smanjene preciznosti modela, što nije trivijalan zadatak. Kako bi riješio ovaj problem, LightGBM uvodi dvije nove tehnike: *Gradient-based One-Side Sampling* (GOSS) i *Exclusive Feature Bundling* (EFB) [42].

5.3.1. Gradient-based One-Side Sampling (GOSS)

GOSS je metoda kojom se vrši *undersampling* nad podacima iz *dataset*-a, odnosno kojom se izbacuje određen broj instanci, a preostali podaci se koriste u izgradnji stabla. GOSS to radi na takav način da ostvaruje dobar balans između smanjenja broja instanci podataka i održavanja preciznosti za naučene DT-ove. Najveći izazov ove metode je upravo u tome da se za izgradnju stabla odaberu one instance koje imaju najviše uticaja na IG i koje će najbolje održati pomenuti balans. Kod AdaBoost-a, težine svake instance služe kao veoma dobar indikator važnosti te instance. Kako kod *gradient boosting* pristupa ne postoje takve izvorne težine, ovaj metod ne može biti primijenjen, ali se u tu svrhu može koristiti gradijent za svaku instancu. Kod instanci sa manjim gradijentom je greška treninga mala, pa su takve instance već dobro istrenirane, dok su one sa velikim gradijentom nedovoljno dobro istrenirane te se na njih treba obratiti veća pažnja. Stoga se vrši izbacivanje određenog procenta instanci sa malim gradijentom, pri čemu se ne smije dozvoliti da se distribucija podataka promijeni, jer bi to negativno uticalo na tačnost naučenog modela. Kako bi izbjegao ovaj problem, GOSS uvodi konstantni multiplikator za instance sa malim gradijentima. Preciznije, GOSS prvo sortira sve instance prema apsolutnim vrijednostima njihovih gradijenata, pa zatim selektuje prvih a*100% instanci (instance sa velikim gradijentom). Nakon toga nasumično bira b*100% instanci od preostalih podatka (instance sa malim gradijentom). Finalno, GOSS umnožava odabrane instance konstantom (1-a)/b. Ovim je stavljen fokus na slabo istrenirane instance bez značajnijeg mijenjanja originalne distribucije podataka [42].

[45] *Leaf-wise* rast je bio ekskluzivna karakteristika LightGBM-a ali je kasnije i XGBoost dodao ovu mogućnost podesivu kroz *grow_policy* hiperparametar algoritma. U LightGBM dokumentaciji i njihovim naučnim radovima se navodi da XGBoost nema podršku za *leaf-wise* rast, ali su to ispravili na njihovom blogu [44].

5.3.2. Exclusive Feature Bundling (EFB)

EFB je pristup kojim se smanjuje broj atributa na način da su gubici saznajne vrijednosti prilikom treniranja modela gotovo zanemarljivi. U praksi se često dešava da su podaci kojima se raspolaže poprilično razuđeni. Mnogi atributi su međusobno isključivi, odnosno nikad ne uzimaju nenulte vrijednosti istovremeno. EFB omogućava grupisanje ovakvih atributa u pakete ili tzv. *bundle*-ove pri čemu se oni posmatraju kao jedan zaseban atribut. Ovim se složenost izgradnje histograma smanjuje sa O(broj_instanci*broj_atributa) na O(broj_instanci*broj_*bundle*-ova), gdje je broj_*bundle*-ova << broj_atributa. Ovim se značajno ubrzava proces treniranja GBDT-a bez značajnijeg uticaja na tačnost modela. Ipak, pokazalo se da je proces pronalaženja optimalnih *bunlde*-ova *NP-Hard* problem[46]. Stoga je EFB pojednostavljen na način da se kreira težinski graf čiji su čvorovi atributi, a težine na ivicama grana odgovaraju ukupnom broju konflikata između dva atributa. Pomenute težine se sortiraju u opadajućem redoslijedu, a atributi sa najmanjim brojem konflikata kreiraju zajednički *bundle* [42].

5.3.3. Priprema kategoričkih atributa

LightGBM za razliku od XGBoost-a ima ugrađen mehanizam obrade kategoričkih atributa. On u pozadini koristi Fisherov algoritam [46], pa nije potrebno vršiti *one-hot encoding*[47]. Dovoljno je kategoričke vrijednosti enkodovati jednostavnim *Label Encoder*-om koji će svakoj novoj kategoričkoj vrijednosti dodijeliti novi *integer* inkrementovan za jedan počevši od nule. Redni brojevi kolona sa ovako enkodovanim vrijednostima se prosljeđuju algoritmu kroz poseban hiperparametar, kako bi algoritam znao da se ne radi o kontinualnim vrijednostima i da između njih ne postoji poredak. Eksperimenti su pokazali da izostavljanje *one-hot encoding*-a ubrzava algoritam oko 8 puta [48].

5.3.4. Poređenje performansi sa prethodnicima

Eksperimentalno je dokazano da je uvođenjem GOSS-a koji koristi 10-20% podataka iz *dataset*-a vrijeme izvršavanja LightGBM algoritma duplo kraće od osnovne verzije bez ove metode[48], dok EFB za *dataset*-ove sa većim brojem atributa može skratiti vrijeme i do deset puta [42]. U tabeli 5.1 su prikazana prosječna vremena izvršavanja jedne iteracije ovog algoritma (izražena u sekundama) nad 5 različitih *dataset*-ova u osnovnoj verziji bez GOSS-a i EFB-a (osnovni_lgb), samo sa EFB-om (lgb_sa_EFB) i sa obe tehnike (LightGBM), upoređena sa XGBoost algoritmom sa osnovnim *presorted* algoritmom (xgb_presorted) i sa histogramom (xgb_hist). Tabela 5.2 pokazuje da, uprkos značajnom smanjenju vremena izvršavanja, tačnost ostaje približno ista, što je i bio cilj.

[46] *NP-hard* je klasa kompleksnosti koja obuhvata probleme odlučivanja suštinski teže od onih problema koji se mogu riješiti nedeterminističkom Turingovom mašinom u polinomijalnom vremenu [45].

[47] Iako pri korištenju LightGBM-a nije potrebno vršiti *one-hot encoding*, moguće je eksplicitno odrediti slučajeve u kojima algoritam ipak koristi takav *encoding* u pozadini. To se postiže navođenjem parametra *max_cat_to_one_hot*. Ukoliko je kardinalnost skupa vrijednosti nekog kategoričkog atributa manja ili jednaka broju proslijeđenom ovim parametrom, na takav atribut će se primijeniti logika za *one-hot encoding* [47].

[48] Iako GOSS gradi stabla učeći samo iz pomenutih 10-20% podataka i dalje se gradijenti i predikcije računaju za čitav *dataset*. Stoga ubrzanje algoritma nije linearno povezano sa postotkom odabranih podataka, ali je i dalje veoma značajno.

Tabela 5.1 – Poređenje vremena treninga XGBoost i LightGBM algoritma

	xgb_presorted	xgb_hist	osnovni_lgb	lgb_sa_EFB	LightGBM
Allstate	10.85	2.63	6.07	0.71	0.28
Flight Delay	5.94	1.05	1.39	0.27	0.22
LETOR	5.55	0.63	0.49	0.46	0.31
KDD10	108.27	OOM	39.85	6.33	2.85
KDD12	191.99	OOM	168.26	20.23	12.67

Commented [AK18]: Ubaciti negdje podatak o kojoj jedinici vremena se radi. Može u naslovu tabele, u tabeli, ili u tekstu u kojem se referencira tabela..

Tabela 5.2 – Poređenje tačnosti XGBoost i LightGBM algoritma nad testnim dataset-ovima

	xgb_presorted	xgb_hist	osnovni_lgb	LightGBM
Allstate	0.607	0.6089	0.6093	0.6093 ± 9e-5
Flight Delay	0.7601	0.7840	0.7847	0.7846 ± 4e-5
LETOR	0.4977	0.4982	0.5277	0.5275 ± 5e-4
KDD10	0.7796	OOM	0.78735	0.78732 ± 1e-4
KDD12	0.7029	OOM	0.7049	0.7051 ± 5e-5

5.4. CatBoost

U aprilu 2017. godine jedna od vodećih ruskih IT kompanija – Yandex, objavila je *open source* bibilioteku pod nazivom CatBoost. Kao najmlađi od pomenuta četiri *gradient boosting* algoritma, CatBoost preuzima sve dobre karakteristike njegovih prethodnika, uz neka dodatna poboljšanja. Ta poboljšanja se prije svega odnose na način na koji se obrađuju kategorički atributi. Za razliku od njegovih prethodnika koji kategoričke atribute najčešće enkoduju uz pomoć *one-hot encoding*-a, što može biti veoma zahtjevno kod atributa sa velikom kardinalnošću, CatBoost koristi pristup poznat kao *target statistics* - TS [49].

5.4.1. Target statistics i curenje targeta

TS pristup podrazumijeva da se svaka kategorička vrijednost zamijeni sa aritmetičkom sredinom svih ciljnih vrijednosti onih instanci iz skupa za treniranje koje imaju tu kategoričku vrijednost. Ovo se sve radi u pozadini pa nije potrebna bilo kakva manuelnu transformacija ovih vrijednosti. Algoritam je sposoban da ih obradi u sirovom formatu kao stringove. *TS* je daleko jednostavniji i računski manje zahtjevan od prethodnih načina reprezentacije kategoričkih vrijednosti. Međutim, ispostavilo se da ovakav princip dovodi do fenomena poznatog kao curenje targeta (eng. *target leakage).* Do toga dolazi jer se ciljne vrijednosti, tj. targeti, koriste za izračunavanje reprezentacije kategoričkih vrijednosti, da bi se nakon toga te iste kategoričke vrijednosti koristile za predviđanje targeta. Kreatori CatBoost-a su dokazali da se specijalni oblik curenja targeta dešava u svim postojećim implementacijama *gradient boosting* algoritma, iako oni ne koriste TS. Do njega dolazi jer su gradijentne statistike koje se koriste u svakom koraku *boosting*-a, izračunate na osnovu ciljnih vrijednosti istih instanci podataka nad kojima je izgrađeno trenutno stablo. Ovaj poseban oblik curenja targeta dovodi do pomjeranja predikcija (eng. *prediction shift*) – promjene distribucije izračunatih gradijenata u bilo kojem domenu prostora atributa u odnosu na pravu distribuciju gradijenata na tom domenu. Kako bi se ovo izbjeglo i riješio

problem curenja targeta, CatBoost uvodi *ordered boosting*, kao alternativu klasičnom *boosting*-u koji postoji u prethodnim implementacijama *gradient boosting* algoritma [49].

5.4.2. Ordered boosting

Ordered boosting je inspirisan *online learning* algoritmima koji podatke za trening dobijaju sekvencijalno tokom vremena. Kako bi se ova ideja prilagodila *offline* načinu rada, CatBoost uvodi vještačko vrijeme koje dodjeljuje svakoj instanci podataka, te za računanje modela koristi samo one podatke koji su se do tog momenta "pojavili". Ovo znači da bi se za svaku novu instancu podataka trebao trenirati novi model, nad trenutnim *dataset*-om koji predstavlja jednu permutaciju. Time bi se u potpunosti uklonio *prediction shift*, ali je ovakav pristup potpuno neisplativ i značajno povećava složenost i memorijske zahtjeve algoritma. Zbog toga se, umjesto treniranja različitih modela za svaku instancu podataka, trenira log(broj_instanci) modela. Na ovaj način, ukoliko je model treniran na *n* instanci, biće iskorišten za računanje ostataka (gradijentnih statistika) idućih *n* instanci podataka [49].

Tabela 5.3 – plain boosting način rada – log loss i zero-one loss[49] mjere na različitim dataset-ovima i njihova promjena prikazana realtivno u odnosu na ordered boosting [49]

	Log loss		*Zero-one loss*	
	Ordered-boosting	*Plain boosting*	*Ordered boosting*	*Plain boosting*
Adult	0.272	+1.1%	0.127	-0.1%
Amazon	0.139	-0.6%	0.044	-1.5%
Click	0.392	-0.5%	0.156	+0.19%
Epsilon	0.266	+0.6%	0.11	+0.9%
Appetency	0.072	+0.5%	0.018	+1.5%
Churn	0.232	-0.06%	0.072	-0.17%
Internet	0.217	+3.9%	0.099	+5.4%
Upselling	0.166	+0.1%	0.049	+0.4%
Kick	0.285	-0.2%	0.095	-0.1%

CatBoost, pored *ordered boosting*-a, podržava i klasični *boosting*. Ovo ponašanje se lako mijenja postavljanjem *boosting_type* hiperparametra na *plain* za klasični ili *ordered* za *ordered boosting*. Bitno je napomenuti da se u slučaju klasičnog *boosting*-a permutacije i dalje kreiraju za računanje TS-a, ali se za sve ostalo koriste principi klasičnog *boosting*-a, pa se tako neće trenirati novi model za svaku permutaciju, a prilikom izgradnje stabla koriste se svi podaci prisutni u *dataset*-u. Eksperimenti su pokazali da *ordered boosting* daje nešto bolje rezultate u poređenju sa *plain* tipom. Poboljšanje je najviše izraženo kod realtivno malih *dataset*-ova, gdje smanjuje grešku predikcije za 1 do 4% (tabela 5.3). Kod velikih *dataset*-ova poboljšanje je gotovo zanemarljivo. S druge strane, pokazalo se da je *plain boosting* oko 1.7 puta brži od *ordered boosting* implementacije ovog algoritma [49].

[49] *Log loss* i *zero-one loss* su takođe *loss* funkcije na osnovu kojih se izračunava ukupna greška predikcija modela nad *dataset*om. *Log loss* za izračunvanje greške koristi logaritam, a *zero-one loss* svakoj ispravno klasifikovanoj vrijendosti dodjeljuje 0, a svakoj neispravnoj 1.

5.4.3. Druga unapređenja

Za razliku od ostalih implementacija *gradient boosting* algoritama, CatBoost koristi simetrična stabla (eng. *symmetric trees*, u literaturi se još navodi i *oblivious trees*). Karakteristika ovih stabala je da se na jednom nivou stabla koristi isti kriterijum za test, odnosno svi čvorovi na jednom nivou stabla će biti podijeljeni na osnovu vrijednosti istog atributa. Ovakva stabla su balansirana, manje sklona *overfitting*-u i značajno ubrzavaju vrijeme izvršavanja algoritma, jer je jedan od vremenski najzahtjevnijih zadataka prilikom izgradnje stabla pronalaženje najboljeg *split*-a za svaki pojedinačni čvor.

CatBoost takođe ima specifičan način rukovanja vrijednostima atributa koje nedostaju. U zavisnosti od opcije koju korisnik odabere, sve takve vrijednosti biće zamijenjene minimalnom ili maksimalnom vrijednošću tog atributa prisutnom u *dataset*-u.

5.4.4. Priprema kategoričkih atributa

CatBoost je u odnosu na svoje prethodnike otišao korak dalje u olakšavanju načina pripreme kategoričkih atributa. Tako, prilikom korištenja ovog algoritma, nije potrebno njihovo pretprocesiranje. Dovoljno je samo brojeve kolona u kojima se nalaze takvi tipovi atributa proslijediti algoritmu, a same vrijednosti mogu ostati u izvornom formatu i biće konvertovane implicitno korištenjem TS-a.

5.4.5. Poređenje performansi sa prethodnicima

Kreatori CatBoost-a su uporedili performanse njihovog algoritma sa dvjema najpopularnijim *open-source* bibliotekama, LightGBM i XGBoost. Pokazalo se da na svim *dataset*-ovima korištenim u ovom eksperimentu CatBoost daje bolje rezultate (tabela 5.4) [49]. Što se tiče vremena izvršavanja, na osnovu rezultata dobijenih na Epsilon *dataset*-u, može se zaključiti da je *plain* implementacija CatBoost-a u rangu sa LightGBM-om, što ga svrstava u red veoma brzih *gradient boosting* algoritama, dok je implementacija sa *ordered boosting*-om nešto sporija. Prosječno vrijeme treniranja isto je za *plain* CatBoost i LightGBM i iznosi 1.1 sekund po stablu, za *ordered* CatBoost 1.9s, a za XGBoost 3.9s po stablu [49].

Tabela 5.4 – Poređenje sa najpopularnijim GBDT implementacijama: log loss/zero-one loss vrijednosti kod CatBoost-a i njihova relativna promjena kod LightGBM-a i XGBoost-a

	CatBoost	LightGBM	XGBoost
Adult	0.270/0.127	+2.4% / +1.9%	+2.2% / +1%
Amazon	0.139/0.044	+17% / +21%	+17% / +21%
Click	0.392/0.156	+1.2% / +1.2%	+1.2% / +1.2%
Epsilon	0.265/0.109	+1.5% / +4.1%	+11% / +12%
Appetency	0.072/0.018	+0.4% / +0.2%	+0.4% / +0.7%
Churn	0.232/0.072	+0.1% / +0.6%	+0.5% / +1.6%
Internet	0.209/0.094	+6.8% / +8.6%	+7.9% / +8.0%
Upselling	0.166/0.049	+0.3% / +0.1%	+0.04% / +0.3%
Kick	0.286/0.095	+3.5% / +4.4%	+3.2% / +4.1%

6. PRAKTIČNI RAD

Za potrebe praktičnog rada posmatran je *census dataset* (takođe poznat i kao *adult income dataset*) [50]. Ovaj *dataset* sadrži podatke o stanovnicima Sjedinjenjih Američkih Država, prikupljene tokom 1994. godine. Pomenuti podaci se koriste za predviđanje ukupnog godišnjeg prihoda svakog stanovnika navedenog u *dataset*-u. Stanovnici se dijele u dvije grupe, oni čiji su godišnji prihodi manji ili jednaki 50 hiljada dolara, i oni čiji su godišnji prihodi veći od pomenute svote, čime je problem sa regresionog sveden na problem binarne klasifikacije.

Model istreniran nad ovim *dataset*-om može biti koristan prilikom provođenja raznih analiza i prikupljanja statistike, određivanja kreditnog rejtinga, potrošačke korpe ili životnog standarda pojedinca. Interesantna je primjena u *online* marketinškim kampanjama, za određivanje platežne moći pojedinca i u skladu s tim tipa proizvoda koji se pojedincu nudi.

Census dataset sadrži sljedeće kolone [51]:

- *age* – godine pojedinca
- *workclas* –generalni pojam koji reprezentuje radni status pojedinca
- *fnlwgt* – *final weight*, procjena broja ljudi koje ovaj zapis predstavlja (koliko ljudi u SAD je opisano takvim vrijednostima atributa kakve su sadržane u zapisu)
- *education* - najviši nivo edukacije ostvarene od strane pojedinca
- *education-num* - najviši nivo edukacije zapisan u numeričkoj formi
- *martial-status* - bračno stanje pojedinca
- *occupation* – generalni tip zanimanja pojedinca
- *relationship* – predstavlja vezu pojedinca u odnosu na ostale (muž, žena, nije u porodici itd. Dijelom redundantno u odnosu na *martial-status*)
- *race* – rasa pojedinca
- *gender* – pol pojedinca
- *capital-gain* – prihodi pojedinca od prodaje imovine
- *capital-loss* – rashodi pojedinca usljed ulaganja i sticanja nove imovine
- *hours-per-week* – broj radnih sati prijavljenih od strane pojedinca
- *native-country* – zemlja porijekla pojedinca
- *label* – ciljna vrijednost koja govori da li pojedinac zarađuje vise od 50 hiljada dolara na godišnjem nivou

6.1. Istraživačka analiza podataka

Istraživačka analiza podataka (eng. *Exploratory Data Analysis – EDA*) je proces koji obuhvata analiziranje *dataset*-a u cilju sumarizacije njegovih karakteristika, najčešće uz vizuelnu reprezentaciju podataka kroz tabele i grafike. EDA pomaže u boljem shvatanju samih podataka te omogućava otkrivanje šablona ili anomalija koje u njima postoje.

S obzirom na to da se izvorni podaci nalaze u csv formatu, *dataset* se vrlo jednostavno može učitati korištenjem Pythonove biblioteke za analizu podataka – *Pandas*, pozivom metode *read_csv*. Ovako učitani podaci se smještaju u *Pandas*-ovu dvodimenzionalnu tabelarnu strukturu podataka — *dataframe* nad kojim je onda moguće vršiti čitav niz operacija. Prvih 5 redova *dataset*-a prikazano je na slici 6.1.

Za analizu podataka i primjenu ML algoritama, veoma je važno znati tipove podataka za svaku od kolona, sto se vrlo jednostavno može dobiti dohvatanjem *dtypes* atributa *dataframe*-a (Slika 6.2 lijevo).

Kod analize i pripreme podataka dobra praksa je provjeriti postoje li duplirani redovi, te ako postoje takve je često najbolje ukloniti. Pozivom metode *duplicated* nad *dataframe*-om dobija se da postoji ukupno 24 duplikata. S obzirom na to da čitav *dataset* ima 32561 red i imajući u vidu prirodu atributa prisutnih u ovom *dataset*-u, može se reći da postoji realna šansa da ova 24 duplikata nisu rezultat nekih anomalija u prikupljanju podataka, već "stvarni" podaci koju slučajno imaju identične vrijednosti. Takve redove ne treba uklanjati, kao što je i navedeno u sekciji 4.4.2.

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	label
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Slika 6.1 – Primjer podataka u dataset-u

```
In [6]: # Provjera tipova podataka
        df_census.dtypes

Out[6]: age                int64
        workclass         object
        fnlwgt             int64
        education         object
        education-num      int64
        marital-status    object
        occupation        object
        relationship      object
        race              object
        gender            object
        capital-gain       int64
        capital-loss       int64
        hours-per-week     int64
        native-country    object
        label             object
        dtype: object
```

```
In [14]: import numpy as np

         # Provjera null vrijednosti
         df_replaced = df_census.replace([' ?',0], np.nan)
         null_value_stats = df_replaced.isnull().sum(axis=0)
         null_value_stats

Out[14]: age                 0
         workclass        1836
         fnlwgt              0
         education           0
         education-num       0
         marital-status      0
         occupation       1843
         relationship        0
         race                0
         gender              0
         capital-gain    29849
         capital-loss    31042
         hours-per-week      0
         native-country    583
         label               0
         dtype: int64
```

Slika 6.2 –Tipovi podataka u dataset-u (lijevo) i pregled null vrijednosti po kolonama (desno)

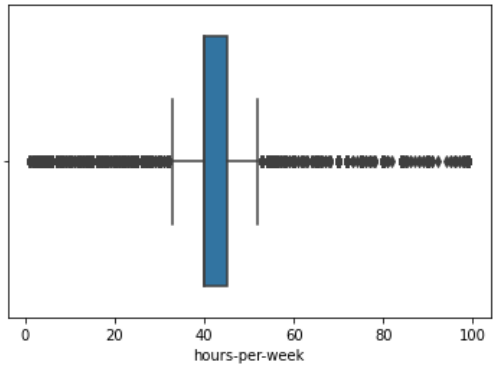
Provjera prisustva nedostajućih vrijednosti je također veoma važna stavka u EDA. One su u ovom *dataset*-u za kategoričke vrijednosti prikazane upitnikom, a za kontinualne nulom. Stoga su oba prikaza zamjenjena *NumPy*-jevom[50] reprezentacijom *null* vrijednosti i izvršeno je sumiranje u cilju određivanja koliko takvih vrijednosti postoji (slika 6.2 desno). Za kolone *workclass* i occupation je taj broj nešto preko 5% ukupnih vrijednosti, za native-country ispod 2%, a za capital-

[50] NumPy je biblioteka za programski jezik Python sa podrškom za velike multidimenzionalne nizove kao i velikom kolekcijom matematičkih funkcija.

gain i capital-loss je to ogromna većina. Uobičajeni pristupi rješavanju ovog problema su uklanjanje redova kod kolona sa malim brojem vrijednosti koje nedostaju (kao što je to slučaj sa kolonom native-country) i potpuno uklanjanje kolona, ukoliko taj broj predstavlja većinu (kolone capital-gain i capital-loss) ili neki od drugih pristupa navedenih u tački 4.4.1. S obzirom na to da *gradient boosting* algoritmi imaju ugrađenu podršku za obradu ovakvih vrijednosti, one se neće uklanjati, ali će se njihov format u određenim slučajevima mijenjati kako bi se prilagodile odgovarajućem algoritmu.

Prilikom detekcije anomalija i onih vrijednosti koje drastično odskaču od ostalih, često se koristi *boxplot*. To je standardizovan način prikazivanja distribucije podataka na osnovu 5 bitnih brojeva: minimuma, prvog kvartila, medijane, trećeg kvartila i maksimuma. Minimum predstavlja najniži broj ispod kojeg će se vrijednosti smatrati vanserijskim (*outliers*-ima), maksimum najviši, dok kvartili dijele distribuciju podataka na 4 jednaka dijela, pa se između prvog i trećeg nalazi 50% podataka. Na slici 6.3 se vidi da pomenutih 50% pojedinaca radi 40 ili malo više od 40 sati sedmično, ali, ipak, postoje oni pojedinci koji rade 0 sati ili čak do 100 sati tokom sedmice. Iako ove vrijednosti odstupaju od većine jasno je da su obe situacije moguće i realne.

```
In [45]: import seaborn as sns

         # Analiza outlier-a
         sns.boxplot(x=df_census['hours-per-week'])
```



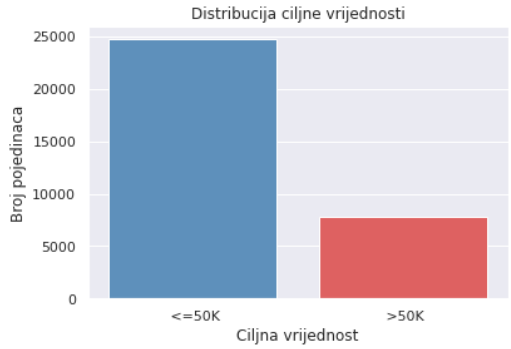
Slika 6.3 – Analiza anomalija uz pomoć boxplot-a

Kao što je već rečeno, u *dataset*-u se nalazi 32561 instanca, koje uključuju 24720 ili 76% pojedinaca koji zarađuju manje od 50 hiljada dolara i 7841 ili 24% onih koji zarađuju više od pomenute svote na godišnjem nivou (slika 6.4).

Ukoliko se posmatra po godinama, najveći procenat ljudi koji zarađuju preko zacrtane sume ima između 40 i 50 godina (slika 6.5 – gore). Isti problem posmatran po nivou obrazovanja jasno ukazuje da što veće obrazovanje pojedinac ima, veća su mu i godišnja primanja, što se i moglo pretpostaviti. Na slici 6.5 (dolje) se jasno vidi da kod osoba sa najvišim stepenom obrazovanja skoro da i ne postoje one čija su primanja ispod graničnih 50 hiljada dolara.

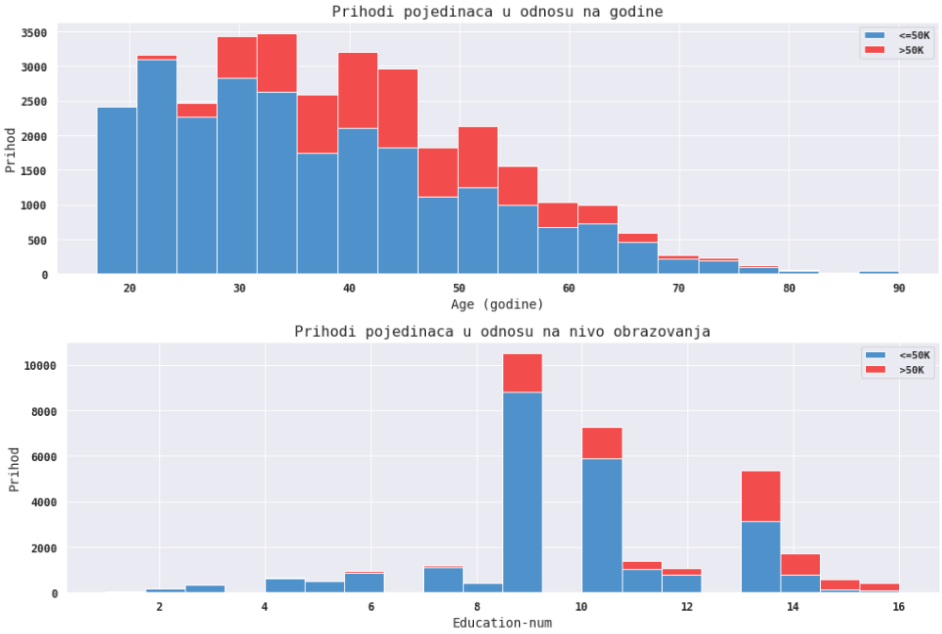
Dobar uvid u važnost samih atributa se može dobiti matricom korelacije. Matrica korelacije ukazuje na to koliko su svaki od atributa međusobno korelisani, te kolika je njihova korelacija sa ciljnom vrijednošću. Atributi i ciljne vrijednosti čine kolone i redove ove matrice, a vrijednosti ćelija predstavljaju stepen korelacije odnosnih atributa. Korelacija može biti pozitivna i negativna. Kada mala vrijednost jednog atributa odgovara maloj vrijednosti drugog atributa/ciljne vrijednosti ili velika vrijednost jednog odgovara velikoj vrijednosti drugog radi se o pozitivnoj korelaciji. Ukoliko mala vrijednost jednog atributa odgovara velikoj vrijednosti drugog radi se o negativnoj korelaciji. Negativna korelacija može biti podjednako značajna kao i pozitivna, jer iz nje algoritam uči da mala vrijednost negativno korelisanog atributa znači veću ciljnu vrijednost i obratno. Ukoliko se na osnovu jednog atributa ne može zaključiti ništa o drugom atributu/ciljnoj vrijednosti, tada korelacija ne postoji, a na matrici korelacije su vrijednost u ćeliji koja odgovara ovakvim atributima bliske nuli.

```
In [284]: sns.set(color_codes=True)
          sns.set(style="darkgrid")
          sns.set_palette(my_pallete)

          # Prikaz distribucije ciljne vrijednosti

          b = sns.countplot(x='label',data = df_census)
          b.axes.set_title('Distribucija ciljne vrijednosti')
          b.set_xlabel("Ciljna vrijednost")
          b.set_ylabel("Broj pojedinaca")

Out[284]: Text(0, 0.5, 'Broj pojedinaca')
```



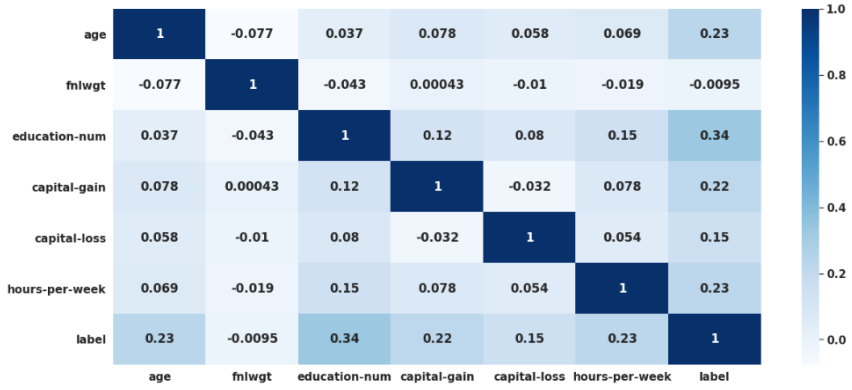
Slika 6.4 –Odnos broja pojedinaca koji zarađuju ispod i preko 50 hiljada dolara godišnje

Na slici 6.6 prikazana je matrica korelacije za *census dataset*. Očigledno je da su prikazani samo numerički atributi, jer za kategoričke nije moguće odrediti korelaciju. Najveći stepen korelacije sa ciljnom vrijednošću ima nivo edukacije, a najniži procjena koliko još ljudi van ovog *dataset*-a ima iste karakteristike kao trenutno posmatrani pojedinac (*fnlwgt*) i to je jedini atribut čija je korelacija negativna. Za oba atributa je ovakav ishod očekivan. S obzirom na to da *fnlwgt* nije u korelaciji sa ciljnom vrijednošću, on se prilikom pripreme podataka odbacuje. Iduće dvije najniže korelacije su *capital-gain* i *capital-loss*, ali ovo je najvjerovatnije posljedica velikog broja *null* vrijednosti. Može se naslutiti da onaj dio postojećih vrijednosti snažno utiče na ciljnu vrijednost, ali je taj uticaj zbog pomentih *null* vrijednosti smanjen.



Slika 6.5 – Raspodjela ciljne vrijednosti posmatrano po godinama(gore) i nivou obrazovanja (dolje)

```
In [337]: sns.set(font_scale=1.4)
          df_census.replace([' <=50K', ' >50K'], [0, 1], inplace=True)
          plt.figure(figsize=(20,9))
          ad= df_census.corr()
          sns.heatmap(ad,cmap="Blues",annot=True)
Out[337]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5b229514d0>
```



Slika 6.6 –Matrica korelacije

6.2. *Hyperopt*

Za treniranje dobrog i robusnog modela, od ključne je važnosti odabrati onaj skup hiperparametara za koji će model davati najbolje rezultate. Čak i male promjene tih parametara mogu dovesti do značajne razlike u performansama. Pronalazak najbolje kombinacije hiperparametara i njihovih vrijednosti je veoma obiman zadatak, pogotovo ako se uzme u obzir da je njihov broj kao i opseg njihovih vrijednosti kod većine modernih algoritama jako veliki. Ovo iziskuje potrebu za automatizacijom pretrage hiperparametara i poređenja modela istreniranih njihovom primjenom – proces poznatiji kao *grid search*. *Grid search* podrazumijeva treniranje modela sa svim mogućim kombinacijama navedenih vrijednosti hiperparametara, što može da bude veoma procesno i vremenski zahtjevno. U svrhu pronalaženja, odnosno optimizacije hiperparametara ovdje se koristi *hyperopt* kao naprednija verzija *grid search*-a. *Hyperopt* ne prelazi slijepo preko prostora mogućih vrijednosti hiperparametara, već koristi TPE (*Tree-structured Parzen Estimator*) ili ATPE (*Adaptive TPE*) algoritam da u svakoj narednoj evaluaciji (u svakom narednom treniranju i validaciji modela) pronađe hiperparametre, slične onima koji su dali do sada najbolje modele (sa najmanjim *loss*-om) [52]. Opcija za potpunu nasumičnu pretragu je takođe podržana [53].

Najbitniji dijelovi *hyperopt*-a su prostor pretraživanja hiperparametara i ciljna funkcija (eng. *objective function*) koja se minimizuje. Prostor pretraživanja obuhvata skup hiperparametara nad kojima se vrši optimizacija i opseg njihovih mogućih vrijednosti. U ciljnoj funkciji se vrši inicijalizacija algoritma i treniranje modela sa kombinacijom parametara iz prostora pretraživanja i statičkih parametara identičnih za svaku *hyperopt* evaluaciju. Povratna vrijednost ove funkcije može biti *floating-point* podatak − *loss* ili Python rječnik (eng. *dictionary*). U drugom slučaju moguće je vratiti mnogo više statističkih i dijagnostičkih podataka o evaluaciji kroz rječnik, ali dvije vrijednosti su obavezne: *status* (jedan od ključeva iz hyperopt.STATUS_STRINGS, kao npr. *ok* za uspješno izvršavanje ili *fail* za neuspješno) i *loss* (*floating-point* vrijednost koja se pokušava minimizovati) [53]. Treći bitan dio *hyperopt*-a je *trials* objekat, koji predstavlja svojevrsnu bazu podataka u kojoj se čuva istorija svih povratnih vrijednosti ciljne funkcije, za svaku od evaluacija u procesu optimizacije hiperparametara. Upotreba *Trials* objekta je opciona.

Hyperopt se pokreće prosljeđivanjem ciljne funkcije, prostora pretraživanja, *trials* objekta, ukupnog broja evaluacija i tipa algoritma za pretraživanje (TPE, ATPE ili *random*) funkciji *fmin*, čija je povratna vrijednost rječnik sa onim hiperparametrima kojima je izgenerisan model sa najmanjim *loss*-om. Ti hiperparametri se onda koriste za ponovnu izgradnju najboljeg modela.

Izgled ciljne funkcije i prostora pretraživanja hiperparametara prikazani su na slici 6.8.

Commented [ZD19]: Čini mi se da nigdje nije naveden eng. termin. Uradite to kod prvog pojavljivanja ovog termina, ako to već nije urađeno.

6.3. AdaBoost

6.3.1. Priprema podataka

AdaBoost spada u grupu algoritama koji ne podržavaju kategoričke atribute pa se priprema podataka za ovaj algoritam svodi na njihovo enkodovanje. U ovu svrhu se koristi *one-hot encoding* (tačka 4.4.5). S obzirom na to da je ciljna vrijednost binarnog tipa tj. da može uzeti samo dvije vrijednosti, za nju nije korišten nikakav specijalizovani enkoder, već jednostavna *replace* metoda koja mijenja sve vrijednosti "<=50K" sa nulama a ">50K" sa jedinicama. Isti je slučaj sa polom pojedinca (kolona *gender*). Nad ostalim kolonama je primijenjen *LabelBinarizer* iz *sklean.preprocessing* paketa (slika 6.7). *LabelBinarizer* svaku kolonu dijeli na više kolona, a svaka od njih odgovara jednoj vrijednosti koju atribut iz podijeljene kolone može da ima. Zbog toga se i od *null* vrijednosti (predstavljenih upitnikom) kreira jedna kolona, koju je potrebno odbaciti, a preostale kolone konkatenirati sa prethodnim *dataframe*-om.

```
# zamjena binarnih atributa nulama i jedinicama
df_census_one_hot['gender'].replace([' Female',' Male'],[0,1], inplace=True)
df_census_one_hot.replace([' <=50K', ' >50K'], [0, 1], inplace=True);

# one-hot encoding preostalih atributa
for category in categorical_features:
    print(category)
    lb = LabelBinarizer()
    lb_results = lb.fit_transform(df_census_one_hot[category])
    lb_results_df = pd.DataFrame(lb_results, columns=lb.classes_)
    df_census_one_hot.drop([category], axis=1, inplace=True)
    print(lb.classes_)
    df_census_one_hot = pd.concat([df_census_one_hot, lb_results_df], axis=1)

df_census_one_hot.drop([' ?'], axis=1, inplace=True)
```

Slika 6.7 – Priprema podataka za AdaBoost algoritam

Što se tiče ostalih standardnih koraka u pripremi podataka obrazloženih u paragrafu 4.4, većina ih je obrađena tokom EDA u paragrafu 6.1. Zbor nepostojanja korelacije između atributa *fnlwgt* i ciljne vrijednosti, kolona sa ovim atributom se odbacuje. U *dataset*-u nisu uočene nekonzistentne vrijednosti, a ne postoji potreba ni za inženjeringom atributa. *Dataset* jeste nebalansiran, sa 3.17 puta više pozitivnih nego negativnih ciljnih vrijednosti, ali nema potrebe za *oversampling*-om ili *undersampling*-om, jer svi algoritmi osim AdaBoost-a omogućavaju rukovanje ovakvim situacijama kroz parametre. Takođe, pokazalo se da kod ovog *dataset*-a čak ni balansiranje kroz parametre algoritma ne donosi značajno bolje rezultate, češće ih kvari, te je stoga i ovaj korak izostavljen.

Finalno, transformisani *dataset* se dijeli na trening, validacioni i testni skup u omjeru 70:15:15.[51] Ovaj odnos je održan pri pripremi podataka za svaki od preostala 3 algoritma.

[51] Ne postoji generalno pravilo koliki omjer treba da bude. Važno je da u trening skupu ima što više podataka, a da istovremeno u validacionom i testnom ostane dovoljno instanci kako bi one činile reprezentativan uzorak. S obzirom na to da ovakva podjela ostavlja po skoro pet hiljada instanci u svakom od preostala dva *dataset*-a, pretpostavlja se da je to dovoljno da obuhvati veći dio varijabilnosti u podacima.

6.3.2. *Izgradnja modela*

Izgradnja modela je faza u kojoj se vrši treniranje algoritma nad trening skupom podataka, uz mogućnost kontinualne validacije tokom procesa treniranja (poređenjem sa nekim drugim, validacionim skupom, kako bi se izbjegao *overfitting*). Krajnji proizvod ove faze je gotov istreniran model, kojeg je na kraju potrebno testirati trećim, testnim skupom podataka, čime bi se potvrdilo da je model dobar i da daje razumne rezultate i na do sada neviđenim podacima.

```
def objective(params):
    # Objective funkcija za optimizaciju gradient boosting hiperparametara

    global ITERATION
    ITERATION += 1

    # Dodavanje parametara iz space-a inicijalnom skupu parametara
    param.update(params)

    start = timer()
    # Inicijalizacija AdaBoost klasifikatora sa trenutnim skupom parametara
    abc = AdaBoostClassifier(n_estimators =int(params['n_estimators']),
                             learning_rate =params['learning_rate'],
                             algorithm = 'SAMME.R', #default
                             random_state = 0)
    # Treniranje modela
    model = abc.fit(X_train, Y_train)

    # Pronalaženje predikcija za instance iz validacionog seta
    y_pred = model.predict(X_valid)
    end = timer()

    # Dohvatanje najboljeg skora (najnižeg loss-a)
    score = metrics.log_loss(Y_valid, y_pred)
    print(score)
    loss = score

    # Upisivanje statistika o svakoj iteraciji u fajl
    of_connection = open(out_file, 'a')
    writer = csv.writer(of_connection)
    writer.writerow([score, params, ITERATION, params['n_estimators'], end-start])
    of_connection.close()

    # Rječnik sa informacijama o evaluaciji
    return {'loss': loss, 'params': params, 'iteration': ITERATION, 'status': STATUS_OK}

# Definisanje prostora pretraživanja hiperparametara
space = {
    'n_estimators': hp.quniform('n_estimators', 250, 1000, 2 ),
    'learning_rate': hp.quniform('learning_rate', 1, 3, 0.1 ),
}
```

Slika 6.8 – Primjer ciljne funkcije i prostora pretraživanja hiperparametara

Na slici 6.8 prikazani su izgled ciljne funkcije i prostor pretraživanja za AdaBoost algoritam. U svakoj evaluaciji *hyperopt*-a se funkciji prosljeđuju vrijednosti hiperparametara iz prostora pretraživanja. Za AdaBoost su od interesa svega dva parametra, *n_estimators* koji se odnosi na broj stabala i *learning_rate*. Njihove vrijednosti se biraju uniformno iz navedenih

intervala[52] (za broj stabala interval od 250 do 1000 sa korakom 2, a za *learning rate* interval od 1 do 3 sa korakom 0.1). Pored ova dva hiperparametra, AdaBoost klasifikatoru proslijeđen je i SAMME.R algoritam (osnovni tip *boosting* algoritma koji postiže nižu grešku za manje iteracija) kao i *random_state* čija vrijednost može biti bilo koji *integer*, a koji kontroliše nasumični *seed* i time omogućava reprodukciju rezultata [54]. S obzirom na to da je AdaBoost meta-algoritam, njemu se može eksplicitno reći koji algoritam da koristi kao osnovu ansambla, te se nad njim vrši *boosting*. To se postiže navođenjem *base_estimator* parametra. Kada ovaj parametar nije naveden, podrazumijevano se koristi DT sa maksimalnom dubinom − 1 (*stump*). AdaBoost klasifikator korišten u ovom primjeru je *AdaBoostClassifier* iz *sklearn.ensemble* paketa [54].

X_train i *Y_train* sa slike 6.8 sadrže podatke iz prethodno pripremljenog trening *dataset*-a učitanog sa fajl sistema. *X_train* sadrži sve atribute i njihove vrijednosti, dok *Y_train* sadrži samo ciljne vrijednosti. Nad ove dvije promjenljive se vrši treniranje modela, dok se nad *X_valid* i *Y_valid* koji sadrže odgovarajuće podatke iz validacionog skupa vrši kontinualna validacija tokom treniranja.

Loss koji se koristi kao povratna vrijednost funkcije i koji se pokušava minimizovati dolazi iz validacionog skupa, a ne skupa za treniranje. Razlog tome je prevencija *overfitting*-a, jer, ukoliko bi se minimizovao trening *loss*, model s njegovom minimalnom vrijednošću bi se previše prilagodio podacima u trening skupu i sigurno ne bi dobro generalizovao, što bi dovelo do velikih grešaka u predikcijama (trening i validaciona greška, paragraf 4.2).

Na kraju su ciljna funkcija i prostor pretraživanja sa slike 6.8 proslijeđeni funkciji *fmin*. Istoj funkciji je proslijeđen i *trials* objekat za slučaj ukazivanja potrebe za naknadnom analizom evaluacija *hyperopt*-a. Broj evaluacija je postavljen na 1000, a za algoritam za pretragu prostora odabran je ATPE.

Finalni rezultat opisanog procesa su optimalni parametri uz pomoć kojih se izgrađuje model sa najmanjim *loss*-om. Nad izgrađenim modelom se pozivom metode *predict_proba* koja kao parametar prima testni skup (bez ciljnih vrijednosti), dobijaju vjerovatnoće pripadnosti odnosnim klasama (vjerovatnoće pozitivne i negativne predikcije), za svaku instancu podataka. Postoji i metoda *predict* koja pomenute vjerovatnoće pretvara u konkretnu predikciju (0 ili 1, sa pragom 0.5).

6.3.3. Analiza rezultata

Za 1000 evaluacija *hyperot*-u je bilo potrebno ukupno 2 sata i 10 minuta, u prosjeku 7.81 sekundi po evaluaciji[53]. Najbolji *log loss* postignut je u 687. evaluaciji i iznosi 4.12206. Vrijednosti hiperparametara koji su rezultirali modelom sa najboljim *loss*-om su: 1.6 za *learning rate* i 298 za broj stabala. Naizgled veoma visoka vrijednost *log loss*-a ne treba da čudi, jer za razliku od ostala 3 algoritma koja su obrađena u ovom radu i koja daju veoma visoke ili veoma niske predikcije (bliske nuli ili jedinici), sve predikcije AdaBoost-a se kreću oko vrijednosti 0.5 (preciznije, minimalna vrijednost predikcije za svih 4884 redova iz testnog skupa je 0.4347, a maksimalna 0.5216). Jasno je da veliko rastojanje između stvarnih ciljnih vrijednosti iz testa (a koje mogu biti samo 0 ili 1) i ovakvih predikcija utiče na povećanje vrijednosti *log loss*-a.

[52] Funkcija *uniform* vraća vrijednost iz navedenog intervala sa uniformnom raspodjelom, a *quniform* vraća vrijednost po ključu: *round*(*uniform*(donja_granica, gornja_granica)/q)*q, gdje je q korak naveden kao treći parametar *quniform* funkcije [53].

[53] Evaluacije pokretane na mašini sa Intelovim i7 procesorom osme generacije (Intel© Core™ i7-8700 CPU 3.20GHz × 6) i 16GB RAM memorije, na operativnom sistemu Linux Mint 19.3.

U tabeli 6.1 prikazani su rezultati performansi modela kroz mjere za evaluaciju opisane u paragrafu 4.5. U svrhu poređenja rezultata različitih algoritama, koje može biti veoma nezgodno iz prostog razloga što različite mjere mogu favorizovati različite algoritme, umjesto fiksnog praga, za određivanje konačne ciljne vrijednosti iz predikcije, uvedeni su pokretni pragovi. Obično vrijednost praga iznosi 0.5, pa sve predikcije preko toga se smatraju jedinicama – što u ovom slučaju znači da pojedinac sa takvom predikcijom ima godišnja primanja preko 50 hiljada dolara, a sve ispod nulama − što znači da su primanja ispod pomenute sume. Za različite pragove model ima različite vrijednosti mjera za evaluaciju, jer se svakim pomijeranjem praga, distribucija pozitivnih i negativnih predikcija mijenja. Za svaki od četiri posmatrana algoritam, postavljeni su oni pragovi, za koje su vrijednosti odziva 50%, 55%, 60%, 65%, 70% i tako do 100%. Izjednačavanjem rezultata algoritama po odzivu omogućeno je njihovo jednostavno poređenje.

Tabela 6.1 – Performanse AdaBoost modela na različitim odzivima

Prag	Tačnost	Preciznost	Odziv	F1 skor
0.500449	86.02%	87.95%	50%	0.6396
0.500296	86.43%	84.67%	55%	0.6667
0.500137	86.51%	80.36%	60%	0.6872
0.500027	86.96%	78.48%	65%	0.7111
0.499901	87.02%	75.63%	70%	0.727
0.499776	86.77%	72.44%	75%	0.7368
0.499612	85.54%	67.48%	80%	0.7322
0.4994	83.21%	61.60%	85%	0.7143
0.499142	80.30%	56.33%	90%	0.6928
0.498762	74.57%	49.23%	95%	0.6486
0.467243	28.32%	25.62%	100%	0.4079

AdaBoost je najveću tačnost od 87.02% ispravno predviđenih instanci iz testnog skupa podataka postigao na odzivu od 70%. Preciznost na istom odzivu iznosi 75.63% dok je najveća očekivano na najnižem odzivu i iznosi 87.95%. F1 skor koji objedinjava preciznost i odziv najviši je na odzivu od 75% i iznosi 0.7368.

Commented [ZD20]:

6.3.4. AdaBoost sa DTC

Iako osnovna implementacija AdaBoost-a kao slabi prediktor koristi stablo odlučivanja dubine 1 (*stump*), radi poređenja i sveobuhvatnosti primjera, pored osnovnog, istreniran je model sa AdaBoost algoritmom koji koristi stablo odlučivanja veće dubine. Inicijalno je opseg dubina postavljen između 2 i 10, ali je u procesu treniranja revidiran. Slično se dogodilo i sa druga dva parametra koja se odnose na stablo: *min_samples_leaf* (minimalan broj instanci u listu) i *min_samples_split* (minimalan broj instanci u čvorovima potomcima). Konačni opseg pretraživanja za ove parametre iznosi 2 do 6 za dubinu, 2 do 20 po logaritamskoj raspodjeli za broj instanci u listu i 10 do 400 za broj instanci u potomcima. Za stablo odlučivanja korišten je *DecisionTreeClassifier* (DTC) iz Pythonovog paketa *sklearn.tree* (slika 6.9).

Izvršeno je 4000 evaluacija *hyperopt*-a, za šta je bilo potrebno 9 sati i 39 minuta, tj. 8.7 sekundi po evaluaciji, što ga čini ubjedljivo najsporijim od svih posmatranih algoritama. Najniži *loss* je ostvaren u 2136. rundi i iznosi 4.100847, a finalne vrijednosti hiperparametara su: *learning_rate* – 0.4, *max_depth* – 2, *min_samples_leaf* – 16, *min_samples_split* – 280.0, *broj stabala* – 280.0. Rezultati su prikazani u tabeli 6.2.

Tabela 6.2 – Performanse AdaBoost algoritma sa DTC na različitim odzivima

Prag	Tačnost	Preciznost	Odziv	F1 skor
0.5019	86.10%	88.81%	50%	0.6398
0.5013	86.67%	85.99%	55%	0.6707
0.5007	86.94%	82.35%	60%	0.6939
0.5002	87.41%	80.25%	65%	0.7183
0.4997	87.41%	76.94%	70%	0.733
0.4991	87.33%	74.00%	75%	0.7452
0.4985	86.06%	68.68%	80%	0.7392
0.4978	84.01%	63.08%	85%	0.7241
0.4967	81.16%	57.59%	90%	0.7023
0.4945	74.06%	48.70%	95%	0.644
0.3942	30.47%	26.21%	100%	0.4153

```
DTC = DecisionTreeClassifier(max_depth = int(param['max_depth']),
                             min_samples_leaf = int(param['min_samples_leaf']),
                             min_samples_split = int(param['min_samples_split']))

abc = AdaBoostClassifier(base_estimator = DTC,
                         n_estimators = int(param['n_estimators']),
                         learning_rate = param['learning_rate'],
                         algorithm = 'SAMME.R', #default
                         random_state = 0)
```

Slika 6.9 – Primjer ciljne funkcije i prostora pretraživanja hiperparametara

Iz tabele 6.2 se vidi da je najveća tačnost 87.41% ostvarena na odzivima od 65 i 70%. Na odzivu od 75% ostvaren je najviši f1 skor koji iznosi 0.7452, a najviša preciznost je na najnižem prikazanom odzivu i iznosi 88.81%.

6.4. XGBoost

6.4.1. Priprema podataka

XGBoost trenutno podržava dva tekstualna formata za čitanje ulaznih podataka, a to su CSV (*Comma-Separated Values*) i LibSVM (*Library for Support Vector Machines*) [55].

Prije prebacivanja ulaznog *datset*-a u neki od ova dva formata, potrebno je prvo izvršiti transformacije samih vrijednosti atributa. XGBoost baš kao i AdaBoost nema podršku za kategoričke atribute, pa je i za njega potrebno izvršiti *one-hot encoding*. Kako je taj dio pripreme podataka identičan kao i za AdaBoost može se iskoristiti taj već prethodno pripremljen *dataset* i njega direktno konvertovati u neki od ova dva formata. Radi sveobuhvatnosti ovog primjera odabrano je da to bude LibSVM format. Takva transformacija zahtijeva za svaku instancu podataka, odnosno svaki red iz *dataset*-a postavljanje ciljne vrijednosti na početak reda, a zatim navođenje svakog atributa i njegove vrijednosti u formatu indeks_kolone:vrijednost_atributa. Svaki atribut sa njegovom vrijednošću je razmakom (*blank* karakterom) razdvojen od ostalih. Atributi čije su vrijednosti nula se ne trebaju navoditi. Isječak pripremljenog ulaznog *dataset*-a u LibSVM formatu je prikazan na slici 6.10.

```
0 1:39 2:13 3:1 4:2174 6:40 13:1 24:1 35:1 38:1 53:1 62:1 101:1
0 1:50 2:13 3:1 6:13 12:1 24:1 33:1 41:1 52:1 62:1 101:1
0 1:38 2:9 3:1 6:40 10:1 26:1 31:1 43:1 53:1 62:1 101:1
0 1:53 2:7 3:1 6:40 10:1 16:1 33:1 43:1 52:1 60:1 101:1
0 1:28 2:13 6:40 10:1 24:1 33:1 47:1 57:1 60:1 67:1
0 1:37 2:14 6:40 10:1 27:1 33:1 41:1 57:1 62:1 101:1
0 1:49 2:5 6:16 10:1 21:1 34:1 45:1 53:1 60:1 85:1
1 1:52 2:9 3:1 6:45 12:1 26:1 33:1 41:1 52:1 62:1 101:1
1 1:31 2:14 4:14084 6:50 10:1 27:1 35:1 47:1 53:1 62:1 101:1
1 1:42 2:13 3:1 4:5178 6:40 10:1 24:1 33:1 41:1 52:1 62:1 101:1
```

Slika 6.10 – Izgled dijela datoteke koja sadrži podatke u LibSVM formatu

Commented [ZD21]: izgled dijela datoteke koja sadrži podatke u LibSVM formatu

6.4.2. Izgradnja modela

Prethodno pripremljeni podaci su učitani uz pomoć *Dmatrix*-a, što je XGBoost-ova interna struktura koja je optimizovana kako u pogledu efikasnosti memorije tako i u pogledu brzine treniranja [56].

Tako učitani podaci se zajedno sa hiperparametrima prosljeđju *train* metodi koja se poziva unutar ciljne funkcije i kojom se vrši treniranje modela (slika 6.11). U istu *train* metodu moguće je kroz parametar *evals* poslati i listu skupova podataka u odnosu na koje će se pratiti *loss* tokom treniranja (u ovom slučaju proslijeđena su sva tri skupa podataka, trening, testni i validacioni). Tokom treniranja modela, za svaku iteraciju tj. za svako novoizgrađeno stablo, na standardni izlaz se ispisuje *loss* za svaki od tri navedena *dataset*-a. Količinu ispisa koje XGBoost generiše tokom treniranja modela moguće je podešavati kroz parametar *verbose_eval*. Informacije o svakoj iteraciji se čuvaju u objektu *results*, proslijeđenom *train* metodi kroz parametar *evals_results*. Iz ovog objekta se dohvata validacioni *loss* po kojem se cilljna funkcija minimizuje. Dva preostala parametra proslijeđena posmatranoj funkciji su *num_iterations* kojim je definisan maksimalan broj iteracija, tj. maksimalan broj stabala koji se generišu prilikom treniranja modela i *early_stopping_rounds*. Navođenjem ovog posljednjeg parametra aktivirano je rano zaustavljanje

(eng. *early stopping*), koje podrazumijeva da se validacioni *loss* mora popraviti makar jednom u svakih n rundi (n je broj proslijeđen kroz ovaj parametar, ovdje 100), u protivnom se treniranje zaustavlja i uzima onaj model koji je generisao najbolji validacioni *loss*. Ukoliko se u *evals* nalazi više skupova, za validacioni se uzima onaj koji je posljednji naveden [56]. Ovime je odbačena mogućnost *overfitting*-a i kreiran model koji ima veliku moć generalizacije. Povratna vrijednost *train* metode je model iz posljednje, a ne iz najbolje iteracije. Stoga se prilikom predikcija, metodi *predict* pored testnog skupa za predikciju, mora proslijediti i limit broja stabala koji se nalaze u modelu. Ova vrijednost je sadržana u atributu *best_ntree_limit* vraćenog modela.

```
num_iterations=10000
watchlist = [(dtrain,'train'),(dtest,'test'),(dvalid,'eval')]
results = {}
bst = xgb.train(param, dtrain, num_iterations, evals=watchlist, early_stopping_rounds=100,
                evals_result=results, verbose_eval=0)

n_estimators=bst.best_iteration
loss = results.get('eval').get('logloss')[bst.best_iteration]
```

Slika 6.11 – Poziv train metode za treniranje XGBoost modela

Što se tiče hiperparametara, tokom izgradnje modela korišteni su *booster* čija vrijednost je postavljena na *gbtree* (osnovni tip DT-a), *objective* sa vrijednošću *binary:logistic* koji definiše osnovni tip problema koji se rješava (u ovom slučaju binarna klasifikacija), *eval_metric* sa vrijednošću *logloss* koja definiše metriku za evaluaciju modela i na kraju *nthread* sa vrijednošću postavljenom na 6, koja definiše stepen paralelizacije odnosno broj *thread*-ova koji će biti korišteni pri treniranju (preporuka je da broj *thread*-ova bude jednak broju jezgara procesora). Uz njih su još postavljeni i *num_iterations* i *early_stopping_rounds*, ali se oni navode odvojeno u *train* metodi. Svi ovi parametri spadaju u statičke i njihova vrijednost je identična u svakoj evaluaciji.

```
space = {
    'eta': hp.quniform('eta', 0.01, 0.25, 0.01),
    'gamma': hp.quniform('gamma', 0.1, 1.5, 0.1),
    'lambda': hp.quniform('lambda', 0.1, 1, 0.1),
    'max_depth': hp.choice('max_depth', [5, 6, 7, 8, 9, 10]),
    'min_child_weight':hp.quniform('min_child_weight', 0.02, 2, 0.02)
}
```

Slika 6.12 – Prostor hiperparametara za XGBoost

Pored statičkih, navedeno je još 5 dinamičkih hiperparametara čije vrijednosti se dobijaju iz prostora pretraživanja. Dinamički hiperparametri i njihove vrijednosti prikazani su na slici 6.12. *Eta* (alias *learning*_rate) je XGBoost-ov naziv za prethodno objašnjeni *learning rate*. *Gamma* (alias *min_split_loss*) predstavlja minimalnu redukciju *loss*-a potrebnu da bi se nastavilo dalje dijeljenje listova stabla (ukoliko navedeno smanjenje nije ostvareno dijeljenjem nekog čvora, taj čvor se ni neće dijeliti već se proglašava terminalnim, odnosno listom). *Max_depth* je parametar kojim se kontroliše maksimalna dubina stabla, a *min_child_weight* definiše minimalnu sumu težina instanci koje moraju postojati u čvorovima potomcima da bi se uopšte vršilo dijeljenje čvora. *Lambda* je hiperparametar kojim se definiše L2 regularizacija (postoji i parametar *alpha*

kojim se definiše L1 regularizacija[54], ali on u ovom primjeru nije korišten) [57]. Bitno je napomenuti da se XGBoost konstantno razvija i da postoje mnogi hiperparametri ili dodatne vrijednosti već postojećih hiperparametara nastale kasnije, kojima se može uticati da XGBoost više liči na neke novije algoritme kao što su LightGBM i CatBoost. Za potrebe poređenja ovih algoritama namjerno su odabrani hiperparametri koji predstavljaju XGBoost u svom izvornom obliku.

Opsezi vrijednosti parametara sa slike 6.12 su inicijalno bili još veći, ali su nakon pokretanja *hyperopta* sa nešto manjim brojem evaluacija i utvrđivanja oko kojih vrijednosti se otprilike kreće minimalan *loss* smanjeni. Isti proces je ponovljen i za preostala dva algoritma.

6.4.3. Analiza rezultata

S obzirom na nešto veći broj parametara i opsege njihovih vrijednosti izvršeno je 8000 evaluacija *hyperopt*-a, ukupnog trajanja 4 sati i 10 minuta, tj. u prosjeku 1.88 sekundi po evaluaciji, što XGBoost čini preko 4 puta bržim od AdaBoost-a. Najbolji *loss* je ostvaren u 5287. evaluaciji i iznosi 0.268159. Hiperparametri koji su uticali na izgradnju ovakvog modela su: *eta* – 0.05, *gamma* – 0.2, *lambda* – 0.3, *max_depth* – 6, *min_child_weight* – 0.88. Broj stabala potrebnih za izgradnju ovog modela je 407.

XGBoost je najveću tačnost od 87.49% ispravno predviđenih instanci iz testnog skupa podataka postigao na odzivu od 70%. Preciznost na istom odzivu iznosi 77.22% dok je najveća očekivano na najnižem odzivu i iznosi 89.73%. F1 skor najviši je na odzivu od 75% i iznosi 0.7467. Ostvareni rezultati su prikazani u tabeli 6.3.

Tabela 6.3 – Performanse XGBoost modela na različitim odzivima

Prag	Tačnost	Preciznost	Odziv	F1 skor
0.689	86.24%	89.73%	50%	0.6422
0.6263	86.40%	84.57%	55%	0.6663
0.5686	86.92%	82.18%	60%	0.6938
0.5177	87.33%	79.92%	65%	0.717
0.4504	87.49%	77.22%	70%	0.7342
0.3987	87.43%	74.30%	75%	0.7467
0.3378	85.67%	67.77%	80%	0.7338
0.272	84.21%	63.47%	85%	0.7267
0.1873	81.14%	57.56%	90%	0.702
0.1094	76.52%	51.32%	95%	0.6665
0.0002	26.21%	25.07%	100%	0.4009

XGBoost takođe nudi dobar API za grafički prikaz informacija o istreniranom modelu. Tako je moguće prikazati bilo koje stablo iz modela jednostavnim navođenjem njegovog rednog broja u metodi *plot_tree*, ili dobiti informaciju o važnosti atributa putem metode *plot_importance* [56]. Ovim se ostvaruje bolji uvid u rezultate i dublje razumijevanje istreniranog modela.

[54] Regularizacija je tehnika kojom se sprečava *overfitting*, dodavanjem regularizacionog izraza na postojeću *loss* funkciju i to sume kvadrata koeficijenata algoritma (ovdje težina listova) za L2, te sume apsolutnih vrijednosti koecifijenata za L1 regularizaciju [14].

6.5. LightGBM

6.5.1. Priprema podataka

Priprema podataka za LightGBM znatno je jednostavnija nego za prethodna dva algoritma. Pored zamjene ciljnih vrijednosti sa 0 i 1, i izbacivanja kolone sa *fnlwgt* atributom, potrebno je još samo enkodovati kategoričke atribute. S obzirom na to da LightGBM za njih ima ugrađenu podršku, nema potrebe za *one-hot encoding*-om, već se koristi obični *LabelEncoder*, koji za razliku od *one-hot encoding*-a ne kreira vektor sa nulama i jedinicom na mjestu odgovarajuće vrijednosti atributa, već jednostavno svaku novu kategoričku vrijednost zamijeni *integer*-om, počevši od nula. Poređenja radi, za kategorički atribut kardinalnosti 16, rezultat *one-hot encoding*-a je 16 kolona sa vrijednostima 0 ili 1, a rezultat *label encoding*-a jedna kolona sa vrijednostima od 0 do 15. Prilikom transformacije vrijednosti kategoričkih atributa u *integer*-e, namjerno su izostavljene vrijednosti sa upitnikom. One su naknadno zamijenjene sa -1, zbog načina na koji LightGBM tretira *null* vrijednosti (za kolone navedene kao kategoričke, sve negativne vrijednosti se tretiraju kao da nedostaju [47]). Finalno se ciljna vrijednost postavlja na mjesto prve kolone, što je njena obavezna pozicija kod LightGBM-a. Svi ostali koraci vezani za anomalije, nekonzistentne vrijednosti, *oversampling* i *undersampling*, dijeljenje na trening, validacioni i testni skup su identični kao za prethodno opisane algoritme.

6.5.2. Izgradnja modela

Poziv metode za treniranje LightGBM modela potpuno je identičan kao za XGBoost. Međutim, za razliku od XGBoost-a, *train* metoda kod LightGBM-a vraća model iz najbolje iteracije (a ne posljednje), pa prilikom predikcije nije neophodno (ali je moguće) navesti broj najbolje iteracije. Kod LightGBM-a su pri učitavanju *dataset*-ova odmah navedeni nazivi kategoričkih atributa i to kroz parametar *categorical_feature* konstruktora *Dataset* objekta iz *lightgbm* paketa. Drugi način za specifikovanje ovog parametra je da se on navede kao jedan u nizu statičkih parametara proslijeđenih *train* metodi. Preostali dio statičkih konfiguracionih parametara čine: *boosting_type* podrazumijevano postavljen na *gbdt*, *objective* postavljen na *binary*, kao i *metric* postavljen na *binary_logloss* (jer se radi o problemu binarne klasifikacije), *use_two_round_loading* postavljen na *false*, što je instrukcija LightGBMu da ne dijeli učitavanje fajla nego da sav smjesti u memoriju, čime se dobija na brzini, *boost_from_average* postavljen na *false*, čime je spriječeno podešavanje incijalnog skora na srednju vrijednost labela radi brže konvergencije, *zero_as_missing* postavljen na *true* čime je rečeno da se za kontinualne atribute nula smatra nedostajućom vrijednošću i na kraju *num_threads* kao broj *thread*-ova postavljen na 6. Pored navedenih, specifikuje se još 5 hiperparametara čije se vrijednosti pokušavaju optimizovati u svrhu dobijanja najboljeg modela. Njihov prostor pretraživanja prikazan je na slici 6.13.

```
space = {
    'num_leaves': hp.choice('num_leaves', [15, 31, 63, 127, 255]),
    'learning_rate': hp.quniform('learning_rate', 0.01, 0.1, 0.005),
    'min_data_in_leaf': scope.int(hp.qloguniform('min_data_in_leaf', np.log(3), np.log(60), 3)),
    'cat_smooth': scope.int(hp.qloguniform('cat_smooth', np.log(30), np.log(150), 5)),
    'min_gain_to_split': hp.quniform('min_gain_to_split', 0.3, 0.8, 0.02)
}
```

Slika 6.13 – Prostor hiperparametara za LightGBM

Parametrom *num_leaves* se podešava maksimalan broj listova u jednom stablu, *learning_rate* je već objašnjen, a sa *min_data_in_leaf* se određuje minimalan broj instanci podataka u jednom listu (ukoliko bi dijeljenjem čvora nastala dva lista od kojih bar jedan ima manje instanci od definisanog broja, dijeljenje se ne bi izvršilo i taj čvor se proglašava listom). Hiperparametar *cat_smooth* se koristi kada *dataset* sadrži kategoričke atribute. Njegova uloga je da umanji efekte šuma koji se mogu pojaviti, pogotovo kod kategoričkih atributa sa malom kardinalnošću [47]. Kod oba prethodna hiperparametra se za dohvatanje vrijednosti koristi funkcija *qloguniform*, koja je veoma slična prethodno opisanoj *quniform* s tom razlikom da je ovdje logaritam povratne vrijednosti uniformno raspodjeljen [53]. Posljednji hiperparametar je *min_gain_to_split* kojim se definiše minimalna količina saznajne vrijednosti (detaljnije o IG-u i redukciji *gini index*-a u tački 3.1.1) dobijena dijeljenjem čvora. Ukoliko navedeni minimum ne može biti ostvaren, do dijeljenja čvora uopšte neće doći i on se proglašava listom [47].

6.5.3. Analiza rezultata

Zbog prostora pretraživanja slične obimnosti, za LightGBM je isto kao i za XGBoost izvršeno 8000 evaluacija *hyperopt*-a. Ipak, ukupno trajanje svih evaluacija iznosi 2 sata i 50 minuta, što je u prosjeku 1.28 sekundi po evaluaciji. LightGBM-u je za isti posao trebalo sat i po vremena manje nego XGBoost-u. Najbolji *loss* je ostvaren u 3853. evaluaciji i iznosi 0.2684. Hiperparametri koji su uticali na izgradnju ovakvog modela su: *learning_rate* – 0.095, *num_leaves* – 15, *cat_smooth* – 110, *min_data_in_leaf* – 3 i *min_gain_to_split* – 0.46. Broj stabala potrebnih za izgradnju ovog modela je 186.

Tabela 6.4 – Performanse LightGBM modela na različitim odzivima

Prag	Tačnost	Preciznost	Odziv	F1 skor
0.6896	86.14%	89.07%	50%	0.6405
0.6368	86.61%	85.66%	55%	0.6697
0.5789	86.92%	82.18%	60%	0.6938
0.5269	87.18%	79.35%	65%	0.7147
0.4671	87.67%	77.81%	70%	0.7373
0.4078	87.74%	75.23%	75%	0.7513
0.3372	86.38%	69.47%	80%	0.7437
0.2671	84.52%	64.06%	85%	0.7306
0.1698	80.26%	56.28%	90%	0.6924
0.0956	75.16%	49.85%	95%	0.6539
0.0007	33.33%	27.03%	100%	0.4255

LightGBM je najveću tačnost od 87.74% ispravno predviđenih instanci iz testnog skupa podataka postigao na odzivu od 70%. Preciznost na istom odzivu iznosi 75.23% dok je najveća očekivano na najnižem odzivu i iznosi 89.07%. F1 skor je najviši na odzivu od 75% i iznosi 0.7513. Ostvareni rezultati su prikazani u tabeli 6.4.

6.6. CatBoost

6.6.1. Priprema podataka

Tvorci CatBoost-a su se potrudili da posao koji treba uložiti u pripremu podataka svedu na minimum. Kao rezultat toga stvoren je algoritam sposoban da samostalno konvertuje kategoričke podatke u numeričku formu. Ukoliko je kardinalnost posmatranog kategoričkog atributa veoma mala, primjenjuje se *one-hot encoding*, u suportnom se koristi TS kako je to opisano u tački 5.4.1 [58]. Ova konverzija se odvija interno, bez potrebe za učešćem korisnika, koji ipak može uticati na ovaj poces definisanjem granične kardinalnosti iznad koje će se primjenjivati TS. To je moguće uraditi prosljeđivanjem parametra *one_hot_max_size.* Njegova podrazumijevana vrijednost zavisi od nekoliko činilaca, ali najčešće je postavljena na 2 [58]. Kako nema potrebe za manuelnom transformacijom vrijednosti atributa, dovoljno je sirov *dataset*, podijeljen na osnovu odnosa navedenog u AdaBoost sekciji pripreme podataka i iz kojeg je iz ranije navedenih razloga izbačena kolona *fnlwgt*, proslijediti direktno algoritmu i on će obaviti sve ostalo. Takođe, nema potrebe za specijalnom pripremom *null* vrijednosti, jer kod kategoričkih atributa CatBoost ne obrađuje ovakve vrijednosti na bilo koji specifičan način [59].

6.6.2. Izgradnja modela

Prije same izgradnje modela potrebno je prvo sa fajl sistema učitati trening i validacioni skup, što se jednostavno postiže korištenjem *read_csv* metode. Nakon učitavanja iz svakog skupa se razdvajaju ciljne vrijednosti u jednu promjenljivu, a ostali atributi u drugu (kao što je urađeno sa *X_train* i *Y_train* kod AdaBoost-a. Tako razdvojeni, *dataset*-ovi se šalju u konstruktor klase Pool iz catboost paketa, zajedno sa nazivima kategoričkih atributa (slično kao kod *Dataset* klase iz *lightgbm* paketa). Nakon ovog koraka podaci su spremni za treniranje.

Kao priprema za treniranje modela, u ciljnoj funkciji se instancira objekat klase *CatBoostClassifier* (iz *catboost* paketa), kome je kroz konstruktor proslijeđen rječnik hiperparametara i njihovih vrijednosti. Nad takvim objektom poziva se *fit* metoda kojom se inicira treniranje modela. Pomenutoj metodi dovoljno je proslijediti prethodno kreirane objekte klase *Pool* sa trening i validacionim podacima čime je izgradnja modela pokrenuta. Za razliku od LightGBM-a i XGBoost-a, ukupan broj iteracija i broj rundi ranog zaustavljanja se navode zajedno sa ostalim parametrima pri instanciranju *CatBoostClassifier*-a, a ne u metodi koja pokreće treniranje. Nad izgrađenim modelom se baš kao i kod AdaBoost-a, pozivom metode *predict_proba* dobijaju vjerovatnoće pripadnosti instance datim klasama, a moguće je koristiti i metodu *predict* koja će pomenute vjerovatnoće pretvoriti u konkretnu predikciju (0 ili 1, sa pragom 0.5).

Statički parametri proslijeđeni CatBoost-u su *iterations* sa vrijednošću 10000 kao i do sada, *loss_function* i *eval_metric* obe sa vrijednošću *Logloss*, *early_stopping_rounds* postavljen na 100, *random_seed* koji se koristi iz istog razloga kao kod AdaBoost-a sa vrijednšću 0 (što je zapravo podrazumijevana vrijednost pa nije bilo neophodno navoditi ga), *thread_count* postavljen na 6 i na kraju *grow_policy* sa vrijednošću *SymmetricTree*. *SymmetricTree* je podrazumijevana vrijednost za *grow_policy* i podrazmijevani tip stabla u CatBoost-u (tačka 5.4.3). Pored ove, moguće su još dvije vrijednosti *grow_policy*-ja: *Depthwise* (stablo se gradi nivo po nivo dok se ne dosegne specifikovana dubina, ali čvorovi istog nivoa se dijele po kriterijumu koji najviše smanjuje *loss*, za razliku od *SymmetricTree* pristupa gdje se svi čvorovi na jednom nivou dijele po istom kriterijumu) i *Lossguide* (stablo se gradi list po list, dok se ne dosegne specifikovan broj listova) [60].

Od hiperparametara koji se optimizuju, navedeni su samo *l2_leaf_reg* (za l2 regularizaciju), *max_depth* i *learning_rate*. Postoje i hiperparametri kao što su *max_leaves* ili *min_data_in_leaf* ali oni nisu navedeni jer se prvi može koristiti samo uz *Lossguide*, a drugi uz *Lossguide* ili *Depthwise grow_policy* [60].

6.6.3. Analiza rezultata

Kako je prostor pretraživanja hiperparametara kod CatBoost-a manji u odnosu na prethodna dva algoritma, i broj evaluacija *hyperopt*-a je smanjen na 3000. CatBoost je svih 3000 evaluacija izvršio za 4 sata i 26 minuta (5.33 sekundi po evaluaciji), pa je u ovom primjeru sporiji i od LightGBM-a i od XGBoost-a. Najbolji validacioni *loss* ostvaren u ovim evaluacijama iznosi 0.27028, a vrijednosti hiperparametara kojima je to ostvareno su redom: *learning_rate – 0.11*, *l2_leaf_reg* – 3.6 i *max_depth* – 5.

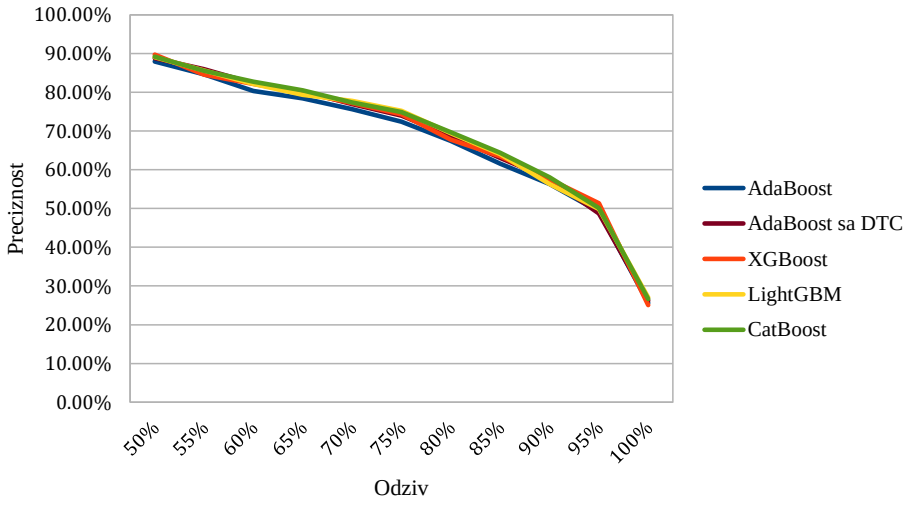
Tabela 6.5 – Performanse CatBoost modela na različitim odzivima

Prag	**Tačnost**	**Preciznost**	**Odziv**	**F1 skor**
0.6839	86.16%	89.20%	50%	0.6408
0.6261	86.59%	85.55%	55%	0.6694
0.5692	87.04%	82.74%	60%	0.6958
0.5161	87.47%	80.49%	65%	0.7193
0.469	87.53%	77.36%	70%	0.7349
0.4125	87.63%	74.92%	75%	0.7498
0.3511	86.47%	69.68%	80%	0.7449
0.2789	84.71%	64.42%	85%	0.7329
0.1966	81.49%	58.07%	90%	0.7061
0.1043	75.41%	50.11%	95%	0.6562
0.0005	32.00%	26.64%	100%	0.4207

Najveća tačnost od 87.63% ispravno predviđenih instanci iz testnog skupa podataka postignuta je na odzivu od 75%. Preciznost na istom odzivu iznosi 74.92% dok je najveća preciznost ostvarena na najnižem odzivu i iznosi 89.2%. Najveći F1 skor je na odzivu od 75% i iznosi 0.7498. Ostvareni rezultati su prikazani u tabeli 6.5.

6.7. Uporedna analiza rezultata

AdaBoost, XGBoost, LightGBM i CatBoost spadaju u istu grupu algoritama koji kombinuju veliki broj slabih prediktora u cilju kreiranja jednog jakog, čineći to kroz niz iteracija u kojima se dodavanjem novog slabog prediktora ispravljaju greške prethodnika. Sa malim izuzetkom AdaBoost-a, čiji se koncept *boosting*-a blago razlikuje od ostalih (sekcija 5.1), osnova ovih algoritama je identična, a razlike koje među njima postoje su plod raznih poboljšanja, optimizacija dizajna sistema, uvođenja paralelizacije, uvođenja regularizacije radi prevencije *overfitting*-a, pojednostavljenja i aproksimacija radi bržeg izvršavanja, proširenja opcija i uvođenja različitih načina rada čime se korisniku daje veliki uticaj na sam ishod izvršavanja algoritma i sl. Treba napomenuti da se posljednja 3 navedena algoritma još uvijek razvijaju, ali utisak je da su svakom novom verzijom sve sličniji jer time ispravljaju potencijalne nedostake u odnosu na druge. Dobar primjer toga je mogućnost konfiguracije načina rasta stabla, gdje bilo koji od algoritama može koristiti način rasta koji je u ranijim verzijama bio ekskluzivna karakteristika samo jednog algoritma. U takvim slučajevima, za svaki algoritam su pri treniranju korištena podrazumijevana podešavanja odnosnih parametara.



Slika 6.14 – Preciznost algoritama na različitim odzivima – census dataset

Uzimajući u obzir sve rečeno i ne čudi sličnost postignutih rezultata (slika 6.14). Maksimalno odstupanje između preciznosti algoritama je na odzivu od 85% i iznosi 2.82%, a minimalno na odzivu od 55% i iznosi 1.32%. Primijetno je da je AdaBoost u osnovnom obliku sa *stump*-ovima kao slabim prediktorom nešto gori od ostalih, te da ni na jednom odzivu ne daje najvišu preciznost, dok je svaki do preostala 4 algoritma (računajući i AdaBoost sa dubljim stablom) bar na jednom odzivu imao najveću preciznost. Algoritam koji je takvih slučajeva imao najviše je CatBoost, dok je maksimalan F1 skor i tačnost (mjere koje objedinjuju preciznost i odziv) ostvario LightGBM na odzivu od 75% i iznose 0.7513 i 87.74% respektivno (tabela 6.6).

Iako je sličnost rezultata očekivana i objašnjena sličnošću samih algoritama, razloge ovako malih odstupanja u performansama treba tražiti i u samom *dataset*-u. Postoji mogućnost da je raspodjela instanci u njemu takva, da se za neke od njih vrlo lako određuje kojoj klasi pripadaju, dok za druge na osnovu vrijednosti njihovih atributa nije moguće sa velikom sigurnošću predvidjeti ciljnu klasu, bez obzira koji algoritam se koristi.

Tabela 6.6 – Preciznost algoritama na različitim odzivima

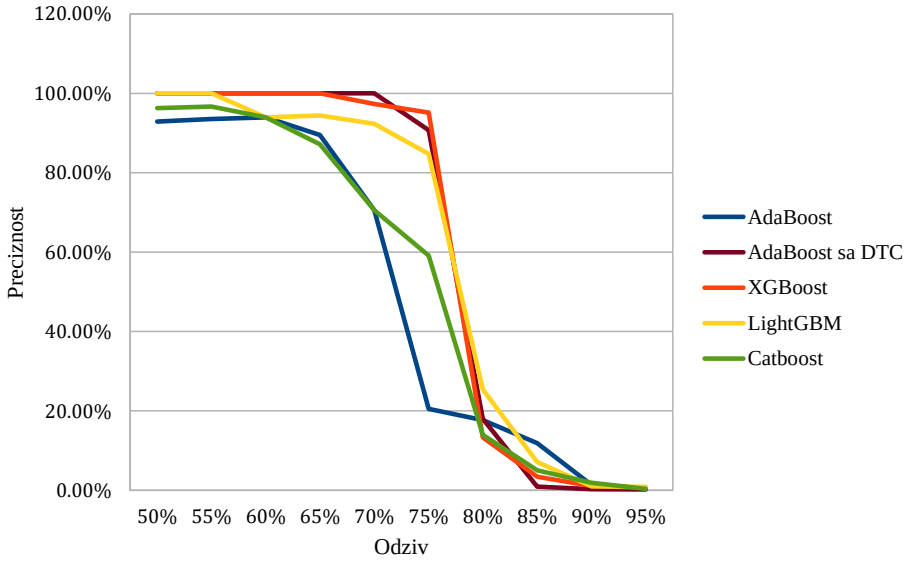
Odziv	AdaBoost	AdaBoost sa DTC	XGBoost	LightGBM	CatBoost
50%	87.95%	88.81%	89.73%	89.07%	89.20%
55%	84.67%	85.99%	84.57%	85.66%	85.55%
60%	80.36%	82.35%	82.18%	82.18%	82.74%
65%	78.48%	80.25%	79.92%	79.35%	80.49%
70%	75.63%	76.94%	77.22%	77.81%	77.36%
75%	72.44%	74.00%	74.30%	75.23%	74.92%
80%	67.48%	68.68%	67.77%	69.47%	69.68%
85%	61.60%	63.08%	63.47%	64.06%	64.42%
90%	56.33%	57.59%	57.56%	56.28%	58.07%
95%	49.23%	48.70%	51.32%	49.85%	50.11%
100%	25.62%	26.21%	25.07%	27.03%	26.64%

Kao primjer *dataset*-a gdje su razlike između algoritama uočljivije naveden je *dataset* za detekciju prevara pri transakciji korištenjem kreditnih kartica. Veoma je važno prepoznati lažne transakcije kako se korisnicima ne bi naplatila stavka koji nisu kupili. U tu svrhu se koristi pomenuti *dataset* koji sadrži 284 807 transakcija, od kojih su 492 uključivale neki vid prevare. Podaci su prikupljeni u septembru 2013. godine i obuhvataju transakcije obavljene od strane vlasnika kartica sa prebivalištem u Evropi [61]. Jasno je da je *dataset* jako nebalansiran, ali to je riješeno podešavanjima hiperparametara algoritama. *Dataset* sadrži 30 atributa, od kojih je 28 dobijeno kao rezultat PCA transformacije[55], a preostale dvije su relativno vrijeme transakcije (broj proteklih sekundi od prve transakcije) i ukupna svota novca. Svih 30 atributa su numerički. Ciljnih vrijednosti ima dvije: 0 – transakcija nije lažna i 1 – jeste. Priprema podataka za algoritme je identična kao i za *census dataset*, s tom razlikom što ovdje nema kategoričkih atributa pa samim tim nema ni potrebe za njihovom transformacijom. Kolona sa relativnim vremenom je izbačena jer nije u korelaciji sa ciljnom vrijednošću. *Dataset* je podijeljen u identičnom omjeru kao i *census* 70:15:15. Konačni rezultati performansi algoritama u istom formatu kao za *census dataset* prikazani su na slici 6.15.

Sa slike 6.15 se vidi da AdaBoost sa *stump*-ovima ima najgore performanse, što je u skladu s rezultatima za prethodni *dataset*-a (slika 6.14). LightGBM, XGBoost i AdaBoost sa DTC također imaju slične performanse, s tim da je LightGBM daje nešto lošije rezultate na nižim odzivima, ali je zato na višim bolji od druga dva algoritma. Najznačajnija razlika u odnosu na *census dataset* primijetna je kod CatBoost-a. Razlog treba tražiti u činjenici da posmatrani *dataset* nema

[55] PCA transformacija je proces redukcije dimenzionalnosti *dataset*-a kroz transformaciju atributa između kojih postoji određen nivo korelacije, u novi manji skup atributa, a da se pri tom ne izgubi mnogo saznajne vrijednosti koju transformisani atributi nose u sebi [62].

kategoričkih atributa, a upravo to je nešto u čemu CatBoost stiče svoju prednost (CatBoost je svoj naziv dobio upravo po glavnom unapređenju – specifičnom načinu obrade kategoričkih atributa − tačka 5.4.1). Uzimajući u obzir sve navedeno, može se reći da su ovakvi rezultati bili očekivani, te da su u skladu sa rezultatima dobijenim na *census datesetu*, uz uvažavanje činjenice da su ovdje razlike između algoritama ipak više ispoljene.



Slika 6.15 – Preciznost algoritama na različitim odzivima – dataset za detekciju prevara pri transakcijama korištenjem kreditnih kartica

Poređenjem dobijenih rezultata dolazi se do zaključka da među posmatranim algoritmima nema jasnog pobjednika. Razlike u postignutim rezultatima češće zavise od *dataset*-a i podešavanja hiperparametara nego od nekog fundamentalnog unapređenja u samom algoritmu. Ipak, postoje indicije da je AdaBoost sa *stump*-ovima ispod nivoa ostalih algoritama. Ukoliko je cilj dobar model, ali je i vrijeme njegove izgradnje takođe od važnosti, LightGBM se nameće kao dobar izbor. Ukoliko je potrebno na brz i efikasan način krenuti u treniranje modela, bez previše razmišljanja oko pripreme podataka i transformacije kategoričkih atributa, CatBoost je prava opcija. Kada je potrebno napraviti robusan i tačan model, gdje vrijeme treniranja modela nije važno, a nije problem uložiti malo truda u pripremu podataka, što je najčešće slučaj, XGBoost je svakako algoritam kojeg treba isprobati.

Kako su svi algoritmi veoma slični, a njihove performanse često zavise od posmatranog *dataset*-a, najbolja praksa je isprobati sve i odabrati onaj koji najviše odgovara tom *dataset*-u i trenutnim potrebama.

7. ZAKLJUČAK

Sve što je ljudski rod stvorio nastalo je iz potrebe, kao vid rješenja nekog postojećeg problema. ML je rastuća oblast sa ogromnim potencijalom rješavanja širokog spektra problema, od onih banalnih do najkompleksnijih pitanja današnjice. Čak i ako za neka od njih odgovori već postoje, ML nudi efikasniji i autonomniji put ka rješenju. Biranje pravog alata ključni je korak na tom putu.

Poznavanje prirode problema kao i osnovnih tipova ML algoritama preduslovi su za biranje najpogodnijeg tipa, odnosno porodice ML algoritama specijalizovane za datu vrstu problema. Iako je odabir pravog tipa algoritma jako dobra polazna tačka, izdvajanje konkretne implementacije algoritma iz date porodice koji najviše odgovara trenutnim potrebama, može donijeti odlučujuću prednost. To je ipak nešto zahtjevniji i neodređeniji zadatak. Kada se pogledaju rezultati praktičnog dijela ovog rada jasno je zašto je to tako. Ne postoji egzaktno pravilo na osnovu kojeg se može reći da je jedan algoritam bolji od drugog. Ono što svakako spada u domen dobre prakse je izdvajanje vremena za testiranje različitih algoritama, ali i mnogih različitih podešavanja istog algoritma. Svaka konkretna situacija i svaki *dataset* nose u sebi određen nivo raznolikosti, a raznolikost je ono što čini da ovi algoritmi nadmaše jedni druge. Najbolji način za odabir onog pravog je upravo njegova primjena na stvarnim podacima, te poređenje rezultata ostvarenih od strane svakog od njih.

Ono što je takođe korisno napomenuti je da ne treba težiti kompleksnosti. Kompleksniji algoritam ne mora uvijek značiti i bolji. Mnogo je primjera gdje je isti nivo performansi u pogledu konačnih rezultata algoritma i valjanosti istreniranog modela postignut sa veoma jednostavnim pristupom. Iz nekih podataka se zaključci izvlače sa ogromnom lakoćom, dok je iz drugih nemoguće izvući smislen zaključak, bez obzira na to koliko je primijenjeni postupak složen. Insistiranje na jednostavnosti, te postepenom povećavanju kompleksnosti modela do postizanja željenih performansi, može dovesti do uštede resursa i povećanja efikasnosti.

S obzirom na nivo generalnosti i lakoću prilagođavanju bilo kojem *dataset*-u, kao i na izvanredne rezultate koje postižu, rijetki su problemi gdje *gradient boosting* algoritmi nisu barem opcija koju treba ozbiljno uzeti u razmatranje, a najčešće i iskoristiti kao konačno rješenje. Posmatrajući njihove rezultate ne može se reći da bi odabir bilo koje konkretne implementacije bio promašaj, jer bi svaka od njih ostvarila zavidan nivo performansi, ali navedene finese u pogledu konkretnih potreba, prihvatljivog vremena izvršavanja i prilagodljivosti konkretnim podacima samo bi dodatno uvećale prednost ostvarenu upotrebom *gradient boosting* algoritama.

LITERATURA

[1] Lyman Peter, Varian Hal. “How much information is produced in the world each year?” University of California at Berkley 2003.

[2] Wu, Xindong, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. "Data mining with big data." *IEEE transactions on knowledge and data engineering* 26, br. 1, str. 97-107, 2013.

[3] Rawlinson, Kevin. "Microsoft's Bill Gates insists AI is a threat". *BBC News*, 29.01.2015, https://www.bbc.com/news/31047780, posjećeno: 12.04.2020. godine.

[4] Gibbs, Samuel. "Elon Musk: artificial intelligence is our biggest existential threat". *The Guardian*, 27.10.2014, https://www.theguardian.com/technology/2014/oct/27/elon-musk-artificial-intelligence-ai-biggest-existential-threat, posjećeno: 12.04.2020. godine.

[5] Cellan-Jones, Rory. "Stephen Hawking warns artificial intelligence could end mankind." *BBC news*, 02.12.2014, https://www.bbc.com/news/technology-30290540, posjećeno: 12.04.2020.

[6] Poole, David, Alan Mackworth, and Randy Goebel. "Computational intelligence: a logical approach", 1998.

[7] Russell, Stuart J., Peter Norvig. “Artificial intelligence: a modern approach”, Pearson Education Limited, Malaysia, 2016.

[8] Samuel, Arthur. “Some Studies in Machine Learning Using the Game of Checkers”. *IBM Journal of Research and Development* br. 3, str. 210-229, Jul 1959.

[9] Stacey Higginbotham, “How Facebook is teaching computers to see”, https://fortune.com/2015/06/15/facebook-ai-moments/, posjećeno: 24.11.2019. godine

[10] Ayodele, Taiwo Oladipupo. "Types of machine learning algorithms." *New advances in machine learning*, str. 19-48, 2010.

[11] Joshi, Mahesh V., Ramesh C. Agarwal, and Vipin Kumar. "Predicting rare classes: Can boosting make any weak learner strong?." In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, str. 297-306. 2002.

[12] Friedman, Jerome H. "Stochastic gradient boosting." *Computational statistics & data analysis* 38, br. 4, str. 367-378, 2002.

[13] Urdan, Timothy C. ”Statistics in plain English”. Taylor & Francis, 2016.

[14] M. Nikolić, A. Zečević, “Mašinsko učenje”, Matematički fakultet, Univerzitet u Beogradu, 2019.

[15] Shouman, Mai, Tim Turner, and Rob Stocker. "Using decision tree for diagnosing heart disease patients." P*roceedings of the Ninth Australasian Data Mining Conference*, br. 121, 2011.

[16] Rokach, L., & Maimon, O. “Decision trees”. In *Data mining and knowledge discovery handbook*, str. 165-192. Springer, Boston, MA, 2005.

[17] Sharma, H. & Kumar, S, "A survey on decision tree algorithms of classification in data mining". *International Journal of Science and Research (IJSR)* 5, br. 4, str. 2094-2097, 2016.

[18] Raileanu, Laura Elena, and Kilian Stoffel. "Theoretical comparison between the gini index and information gain criteria." *Annals of Mathematics and Artificial Intelligence* 41, br. 1, str. 77-93, 2004.

[19] Sutton, C. D. "Classification and regression trees, bagging, and boosting", *Handbook of statistics*, br. 24, str. 303-329, 2005.

[20] Xristica, "What is the difference between Bagging and Boosting", 20.04.2016. https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/ posjećeno: 19.04.2020.

[21] Dietterich, Thomas G. "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization." *Machine learning* 40, br. 2, str. 139-157, 2000.

[22] Quinlan, J. Ross. "Bagging, boosting, and C4. 5." *AAAI/IAAI*, br. 1, 1996.

[23] Downey, A. "Think stats: exploratory data analysis." O'Reilly Media, Inc. 2014.

[24] Rosasco, L., Vito, E. D., Caponnetto, A., Piana, M., & Verri, A. "Are loss functions all the same?", *Neural Computation* 16, br. 5, str. 1063-1076. 2004.

[25] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent." In *Proceedings of COMPSTAT'2010*, str. 177-186. Physica-Verlag HD, 2010.

[26] Jovanović Petar, Filip Marić. "Sistem za paralelizaciju neuronskih mreža na OpenCL akceleratorima", 2018.

[27] Bottou, Léon. "Stochastic gradient learning in neural networks." *Proceedings of Neuro-Nımes* 91, br. 8, str. 12, 1991.

[28] Breiman, Leo. "Arcing the edge". Technical Report 486, Statistics Department, University of California at Berkeley, 1997.

[29] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." *Annals of statistics*, str. 1189-1232, 2001.

[30] Natekin, Alexey and Alois Knoll. "Gradient boosting machines, a tutorial." *Frontiers in neurorobotics*, br. 7, str. 21, 2013.

[31] Breiman, Leo. "Bias, variance, and arcing classifiers." Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA, 1996.

[32] Fortmann-Roe, Scott. "Understanding the bias-variance tradeoff", http://scott.fortmann-roe.com/docs/BiasVariance.html, jun 2012, posjećeno: 18.06.2020.

[33] Heaton, J. An empirical analysis of feature engineering for predictive modeling. In *SoutheastCon 2016*, str. 1-6, mart 2016.

[34] Transforming categorical data, https://developers.google.com/machine-learning/data-prep/transform/transform-categorical, posjećeno: 25.04.2020.

[35] Chawla, N. V., Japkowicz, N., & Kotcz, A. “Special issue on learning from imbalanced data sets.” *ACM SIGKDD explorations newsletter* 6, br. 1, str. 1-6, 2004.

[36] Drummond, C., & Holte, R. C. “C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling.” In *Workshop on learning from imbalanced datasets II*, br. 11, str. 1-8, Washington DC: Citeseer, Avgust 2003.

[37] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., ... & Zhou, Z. H. “Top 10 algorithms in data mining.”, *Knowledge and information systems* 14, br. 1, str. 1-37, 2008.

[38] Freund, Yoav, Robert Schapire, and Naoki Abe. "A short introduction to boosting." *Journal-Japanese Society For Artificial Intelligence*, br.14, str. 771-780, 1999.

[39] Schapire, Robert E. "Explaining adaboost." In *Empirical inference*, str. 37-52. Springer, Berlin, Heidelberg, 2013.

[40] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, str. 785-794, 2016.

[41] Nielsen, Didrik. "Tree boosting with xgboost-why does xgboost win" every" machine learning competition?" Master's thesis, NTNU, 2016.

[42] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. “Lightgbm: A highly efficient gradient boosting decision tree.” In *Advances in neural information processing systems*, str. 3146-3154, 2017

[43] LightGBM documentation, Features https://lightgbm.readthedocs.io/en/latest/Features.html, posjećeno: 16.05.2020.

[44] Miguel Fierro, Mathew Salvaris, Guolin Ke, and Tao Wu, “Lessons Learned From Benchmarking Fast Machine Learning Algorithms”, Machine Learning Blog, Microsoft Documentation, 25.07.2017, https://docs.microsoft.com/en-us/archive/blogs/machinelearning/lessons-learned-benchmarking-fast-machine-learning-algorithms, posjećeno: 28.04.2020.

[45] NP-hard, National Institute of Standards and Technology – NIST https://xlinux.nist.gov/dads/HTML/nphard.html, posjećeno: 28.04.2020.

[46] Walter D. Fisher. "On Grouping for Maximum Homogeneity." *Journal of the American Statistical Association* 53, br. 284, str. 789-798, decembar 1958

[47] LightGBM documentation, Parameters, https://lightgbm.readthedocs.io/en/latest/Parameters.html, posjećeno: 16.05.2020.

[48] LightGBM documentation, Quick Start, https://lightgbm.readthedocs.io/en/latest/Quick-Start.html, posjećeno: 16.05.2020.

[49] Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. "CatBoost: unbiased boosting with categorical features." In *Advances in neural information processing systems*, str. 6638-6648. 2018.

[50] Adult *Dataset*, Machine learning repository, University of California, Irvine, https://archive.ics.uci.edu/ml/*datasets*/Adult, posjećeno: 30.05.2020.

[51] Chet Lemon, Chris Zelazo and Kesav Mulakaluri, "Predicting if in-come exceeds $50,000 per year based on 1994 US Census Data with Simple Classification Techniques", http://cseweb.ucsd.edu/classes/sp15/cse190-c/reports/sp15/048.pdf, posjećeno: 30.05.2020.

[52] Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. "Algorithms for hyper-parameter optimization". In *Advances in neural information processing systems*, str. 2546-2554, 2011.

[53] Hyperopt documentation, Fmin, https://github.com/hyperopt/hyperopt/wiki/FMin, posjećeno: 01.06.2020.

[54] Scikit-learn documentation, AdaBoostClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html, posjećeno: 01.06.2020.

[55] XGBoost documentation, Text Input Format of Dmatrix, https://xgboost.readthedocs.io/en/latest/tutorials/input_format.html, posjećeno: 02.06.2020.

[56] XGBoost documentation, Python API Reference, https://xgboost.readthedocs.io/en/latest/python/python_api.html, posjećeno: 02.06.2020.

[57] XGBoost documentation, XGBoost parameters, https://xgboost.readthedocs.io/en/latest/parameter.html, posjećeno: 03.06.2020.

[58] CatBoost documentation, Categorical features, https://catboost.ai/docs/features/categorical-features.html, posjećeno: 04.06.2020.

[59] CatBoost documentation, Missing values processing, https://catboost.ai/docs/concepts/algorithm-missing-values-processing.html posjećeno: 08.06.2020.

[60] CatBoost documentation, Python package training parameters, https://catboost.ai/docs/concepts/python-reference_parameters-list.html posjećeno: 04.06.2020.

[61] Credit Card Fraud Detection, Kaggle, https://www.kaggle.com/mlg-ulb/creditcardfraud, posjećeno 11.06.2020.

[62] Wold, S., Esbensen, K., & Geladi, P. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, br. 1-3, str. 37-52. 1987.