SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1985

Klasifikacija podataka korištenjem radnog okvira Apache Spark

Patrik Mihaljević

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2019.

DIPLOMSKI ZADATAK br. 1985

Pristupnik: **Patrik Mihaljević (0036484232)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Klasifikacija podataka korištenjem radnog okvira Apache Spark**

Opis zadatka:

Apache Spark je radni okvir otvorenog koda za raspodijeljenu obradu podataka. Navedite i detaljno opišite implementirane klasifikacijske algoritme u radnom okviru Apache Spark. Na odabranom studijskom slučaju stvarnih podataka predložite i implementirajte klasifikacijski model, eksperimentalno ga evaluirajte te komentirajte dobivene rezultate i predložite moguća poboljšanja.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Izv. prof. dr. sc. Krešimir Pripužić

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Doc. dr. sc. Marko Čupić

SADRŽAJ

1. Uvod

"Treba li banka klijentu odobriti kredit?", *"Hoće li cijene određenih dionica porasti u sljedećih n dana?"*, *"Prikazuje li medicinski nalaz tkiva malignu tvorevinu?"* samo su neke od odluka koje je, bez dubinske analize podataka i dobrog poznavanja područja, jako teško donijeti. Zbog bržeg i pouzdanijeg načina donošenja odluka, računala, odnosno algoritmi strojnog učenja, su sve popularniji u uporabi kao rješenja ovakvih problema. Strojno učenje je programiranje računala tako da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera ili nekog prethodnog iskustva[4].

U ovom radu su izneseni najpoznatiji klasifikacijski algoritmi implementirani u programskom okviru *Apache Spark*. U početnom dijelu rada (drugom i trećem poglavlju) teorijski se iznose najpoznatiji klasifikacijski algoritmi implementirani unutar programskog okvira *Apache Spark*. Zatim se objašnjava sam programski okvir *Apache Spark*, njegovi najbitniji dijelovi i način raspodijeljenog izvršavanja. U središnjem dijelu rada (četvrtom i petom poglavlju) opisuje se i analizira odabrani eksperimentalni slučaj na kojemu će biti implementiran odabir najboljeg klasifikacijskog modela. Uz to, donosi nam i analizu značajki podataka koje će se koristiti u implementaciji, a objašnjena je i sama implementacija evaluacijskog procesa odabira optimalnog klasifikacijskog modela. Na samom kraju rada iskazani su rezultati eksperimentalne evaluacije implementiranog odabira optimalnog klasifikacijskog modela.

2. Klasifikacijski algoritmi

Strojno učenje je grana umjetne inteligencije koja se bavi oblikovanjem algoritama koji svoju učinkovitost poboljšavaju na temelju empirijskih zadataka. Ono je danas jedno od najaktivnijih i najuzbudljivijih područja računarske znanosti, ponajviše zbog brojnih mogućnosti primjene koje se protežu od raspoznavanja uzoraka i dubinske analize podataka do robotike, računalnog vida, bioinformatike i računalne lingvistike. Algoritmi klasifikacije, koji će biti objašnjeni kasnije ovom poglavlju, spadaju u algoritme nadziranog učenja. Osim nadziranog učenja postoji i nenadzirano učenje koje neće biti objašnjeno u okviru ovog rada.

U ovom poglavlju bit će napravljena podjela modela nadziranog učenja, te iznesena teorijska podloga klasifikacijskih metoda koje su podržane u programskom okviru *Apache Spark*. Algoritmi nadziranog učenja (engl. *supervised learning*) se izvode nad skupom podataka koji uz značajke koje opisuju svaki primjer skupa, sadrži i **oznaku** ili **cilj** svakog pojedinog primjera.

Glavni cilj im je naučiti model nad označenim skupom podataka za treniranje. Treniranje nad označenim skupom podataka omogućava donošenje odluka nad podacima koje model nije vidio. Termin *nadzirano* označava da je, za skup uzoraka s kojim treniramo model, očekivani izlaz modela (u ovom slučaju klasa primjera) unaprijed poznat.

Ugrubo rečeno, algoritmi nadziranog učenja su algoritmi koji uče asocirati određeni ulaz s njegovim izlazom. Dan im je skup podataka za treniranje koji sadrži ulazne vrijednosti x i izlazne vrijednosti y. Često vrijednosti y ne mogu biti automatski prikupljene te algoritam treba nadzirati osoba. Odatle potječe naziv za tu skupinu algoritama. Naravno, ovaj termin se koristi i za slučaj kada su izlazne oznake skupa za treniranje automatski prikupljane[12], kao što je slučaj u primjeru koji će biti obrađen u ovom radu.

Algoritme nadziranog učenja dijelimo na regresijske i klasifikacijske algoritme. Klasifikaciju možemo definirati kao postupak dodjeljivanja klase c iz skupa s konačnim brojem elemenata C, prema određenom pravilu, neoznačenom primjeru x koji pripada ulaznom skupu X. U klasifikacijskom algoritmu podatci su u obliku $(ulaz, izlaz) = (x, y)$, i treba naći preslikavanje $\hat{y} = f(x)$, gdje je y diskretna/nebrojčana vrijednost.

Neki od primjera klasifikacije podataka su:

- klasifikacija novinskih dokumenata u rubrike
- detekcija neželjenih poruka e-pošte (engl. *spam detection*)
- predviđanje kretanja dionica
- određivanje smisla višeznačne riječi
- medicinska dijagnostika
- predviđanje ishoda nogometnih utakmica

2.1. Podjela modela nadziranog učenja

2.1.1. Probabilistički i neprobabilistički modeli

Probabilistički pristup bi rezultirao raspodjelom vjerojatnosti na skup klase za svaki ulazni uzorak. Za razliku od probabilističkih, neprobabilistički (iliti deterministički) modeli ne modeliraju raspodjelu klasa, već razdvajaju prostor značajki i vraćaju klasu povezanu s prostorom iz kojeg uzorak potječe. Izlaz probabilističkih modela ima vjerojatnosnu interpretaciju, dok izlaz iz neprobabilističkih modela ne možemo tako interpretirati. Općenito, probabilistički pristupi bili bi prikladniji za uključivanje nesigurnosti klasifikacije i pružaju nam informaciju o tome koliko je sigurno dobiveno predviđanje. Također, probabilističke metode mogu inkorporirati prethodne informacije o distribuciji klasa.

2.1.2. Generativni i diskriminativni modeli

Po načinu određivanja pripadnosti pojedinog primjera klasi postupke klasifikacije dijelimo na generativne i diskriminativne modele. Generativni modeli uče zajedničku razdiobu primjera x i klase C, $P(x, C)$, iz koje se primjenom Bayesove formule dolazi do vjerojatnosti pripadanja primjera x klasi C - $P(C|x)$. Tu vjerojatnost generativni modeli koriste prilikom klasifikacije. Diskriminativni modeli izravno uče uvjetnu razdiobu $P(C|x)$, odnosno pokušavaju modelirati funkciju koja će pojedinom primjeru x dodijeliti odgovarajuću klasu C. Tako generativni modeli pokušavaju naučiti kako primjeri iz neke klase nastaju (kako se generiraju) te to znanje primjenjuju u klasifikaciji dok diskriminativni modeli određuju klasu samo temeljem vjerojatnosti da neki primjer po svojim značajkama pripada određenoj klasi.

Iz prethodnih definicija se može zaključiti da generativni modeli uče raspodjelu klasa, dok diskriminativni modeli uče njihovu granicu, kao što je ilustrirano na slici 2.1. Odluku o tome koji algoritam bi bilo dobro koristiti možemo temeljiti na broju dostupnih primjera za učenje. Zbog toga što generativni modeli pokušavaju iz primjera donositi zaključke znači

(a) Generativni model

(b) Diskriminativni model

Slika 2.1: Vizualizacija rada generativnog i diskriminativnog klasifikacijskog modela

da oni zahtijevaju jako velik broj kako bi pogreška bila prihvatljiva. Za same postupke klasifikacije primjereniji su diskriminativni modeli jer izbjegavaju složene izračune te ne zahtijevaju velik broj primjera za učenje.[5]

2.1.3. Parametarski i neparametarski modeli

Iz odnosa broja parametara modela i primjera za učenje, nadzirane modele strojnog učenja dijelimo na parametarske i neparametarske. U parametarskim modelima složenost modela ne ovisi o broju primjera za učenje, nego se pretpostavlja teorijska razdioba podataka pa je iz tog razloga broj parametara konačan te određen razdiobom koju model koristi. Neparametarski modeli, unatoč tome što temeljem naziva možemo donijeti zaključak da nemaju parametre taj zaključak bi bio pogrešan. Oni imaju parametre, samo broj njihovih parametara direktno ovisi o broju primjera za učenje pa tako i složenost modela raste povećanjem skupa za učenje. Takvi modeli ne pretpostavljaju razdiobu primjera za učenje.[5]

2.1.4. Linearni i nelinearni modeli

Kako odrediti je li model linearan ili nelinearan? Sa slike 2.2 su vidljiva tri modela s različitim diskriminacijskim granicama koje odvajaju dvije klase.

(a) Linearni model

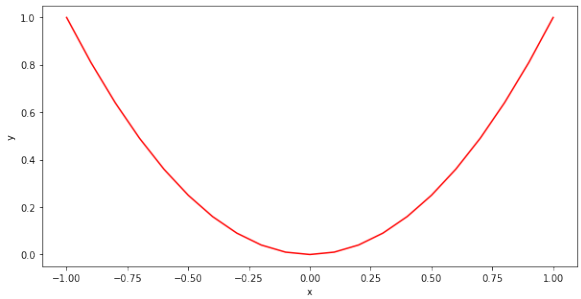
(b) Linearni model

(c) Nelinearni model

Slika 2.2: Vizualizacija razlike linearnog i nelinearnog modela

Iz prethodne slike je vidljivo da su linearni modeli linearni u svojim parametrima koje treba procijeniti, ali ne nužno i u granici (odnosno varijablama funkcije). To objašnjava zašto

slika 2.2b prikazuje linearan model iako granica nije linearna. [17]



Slika 2.3: Parabola $f(x) = ax^2 + bx + c$, za $a = 1, b = 0, c = 0$

Na primjer, funkcija parabole $ax^2+bx+c = 0$ je zakrivljena, ali je u svojim parametrima koje model uči linearna. Nije nezavisna varijabla x ta koja govori o linearnosti modela, već parametri modela (iz slike 2.3 su to a, b i c).

2.2. Logistička regresija

Logistička regresija je diskriminativni model. Unatoč pojmu *"regresija"* u samom imenu, logistička regresija spada u klasifikacijske algoritme. Ona je binarni klasifikator koja mapira linearnu regresiju na logističku funkciju s rasponom $[0, 1]$, koja služi kao odlična početna točka za klasifikacijske probleme.

2.2.1. Model

Prije nego model logističke regresije bude objašnjen, potrebno je prikazati na koji način se aposteriorna vjerojatnost $P(C_j|\boldsymbol{x})$ može modelirati linearnim modelom.

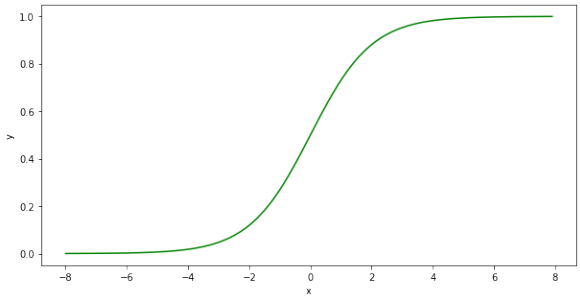
Promotrimo binarnu klasifikaciju u dvije klase, C_1 i C_2. Vrijedi $P(C_1|\boldsymbol{x}) = 1-P(C_2|\boldsymbol{x})$. Iz toga se može napisati da za aposteriornu vjerojatnost klase C_1 vrijedi

$$P(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)P(C_1)}{p(\boldsymbol{x}|C_1)P(C_1) + p(\boldsymbol{x}|C_2)P(C_2)} = \frac{1}{1 + \frac{p(\boldsymbol{x}|C_2)P(C_2)}{p(\boldsymbol{x}|C_1)P(C_1)}} = \frac{1}{1 + \exp(-\alpha)} = \sigma(\alpha) \tag{2.1}$$

gdje je

$$\alpha = ln\frac{p(\boldsymbol{x}|C_1)P(C_1)}{p(\boldsymbol{x}|C_2)P(C_2)}. \tag{2.2}$$

Vidljivo je da je funkcija $\sigma(\alpha)$ upravo **sigmoidalna (logistička) funkcija** prikazana na slici 2.4. Ona je derivabilna te ima vjerojatnosnu, tj. probabilističku interpretaciju $P(C_1|\boldsymbol{x})$.



Slika 2.4: Logistička funkcija $f(x) = \frac{1}{1+\exp(-x)}$

Da bi se funkciju 2.1 moglo tretirati kao poopćeni linearan model, potrebno je α izraziti kao linearnu kombinaciju težina, odnosno

$$\alpha = ln\frac{p(\boldsymbol{x}|C_1)P(C_1)}{p(\boldsymbol{x}|C_2)P(C_2)} = \boldsymbol{w}^T\boldsymbol{x} + w_0 \tag{2.3}$$

To se postiže modeliranjem izglednosti klasa $p(\boldsymbol{x}|C_1)$ i $p(\boldsymbol{x}|C_2)$ Gaussovom razdiobom s dijeljenom kovarijacijskom matricom Σ[1].

$$\begin{aligned}
\alpha = ln\frac{p(\boldsymbol{x}|C_1)P(C_1)}{p(\boldsymbol{x}|C_2)P(C_2)} &= ln(p(\boldsymbol{x}|C_1)P(C_1)) - ln(p(\boldsymbol{x}|C_2)P(C_2)) \\
&= \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + ln(P(C_1)) \\
&\quad - \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 - \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + ln(P(C_2)) \\
&= \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + ln\frac{P(C_1)}{P(C_2)}
\end{aligned} \tag{2.4}$$

Vidljivo je da se dobiveni izraz može reprezentirati kao linearnu kombinaciju težina

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{2.5}$$

$$w_0 = \frac{1}{2}(\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2) + ln\frac{P(C_1)}{P(C_2)} \tag{2.6}$$

što u konačnici dovodi do poopćenog linearnog modela logističke regresije:

$$h(\boldsymbol{x}) = P(C_1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{2.7}$$

[1]Kovarijacijska matrica Σ nekog slučajnog vektora $\boldsymbol{X}$ je matrica kovarijacija njegovih elemenata dimenzija $n \times n$ definirana kao $\Sigma_{ij} = Cov(X_i, X_j)$. Ako je kovarijacijska matrica dijeljena, to znači da je jednaka za svaku klasu.

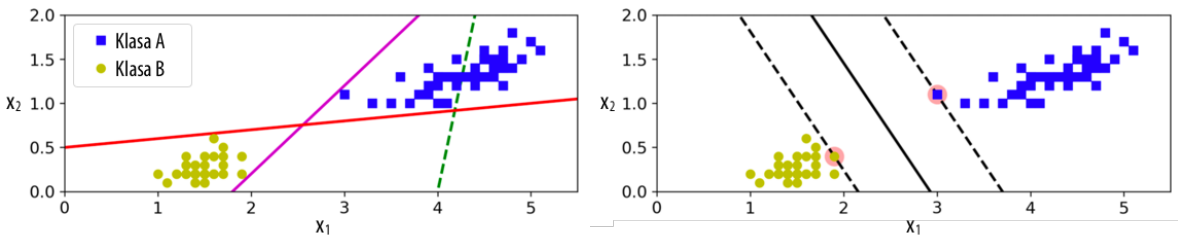
> "Treba međutim imati na umu da se vjerojatnosna interpretacija temelji na pretpostavci da su klase normalno distribuirane i da imaju dijeljenu varijancu. Naravno, ako to nije slučaj, odnosno ako je naša pretpostavka pogrešna i podatci su distribuirani nekako drugačije, vjerojatnosne procjene neće biti dobre, ali nam barem govore koliko su pojedini primjeri udaljeni od granice." [10]

2.3. Stroj potpornih vektora

Stroj potpornih vektora (engl. *Support Vector Machine, SVM*) je jedan od najpopularnijih modela strojnog učenja, sposoban za izvođenje linearne i nelinearne klasifikacije, te čak detektiranje odstupajućih vrijednosti (engl. *outliers*). S obzirom na to da *Apache Spark* ima dostupnu samo implementaciju linearnog SVM-a, nelinearni SVM neće se niti objašnjavati u ovom poglavlju.

2.3.1. Linearni SVM

Temeljna ideja SVM-a je najbolje objašnjena slikom 2.5. Graf 2.5a prikazuje decizijske granice (granice odlučivanja) triju mogućih linearnih klasifikatora. S obzirom na to da je model sa zelenom isprekidanom linijom toliko loš, fokus će biti na pojašnjenju zašto ostala dva modela nisu najbolje rješenje ove klasifikacije. Naime, oni savršeno rade na ovom setu podataka, ali njihove granice dolaze jako blizu pojedinim primjerima da ti modeli vjerojatno neće dobro funkcionirati na novim, neviđenim, instancama. Nasuprot tome, desni grafikon prikazuje SVM klasifikaciju. Ona ne samo da linearno odvaja dvije klase, nego i ostaje što je moguće dalje do najbližih primjera. Takav model se naziva *SVM klasifikator tvrde margine*.[7]



(a) Mogući linearni klasifikatori **(b)** Klasifikacija linearnim SVM-om

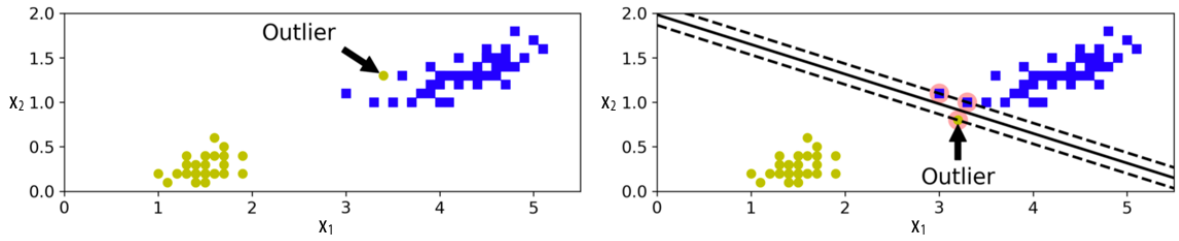
Slika 2.5: Prikaz tvrde margine SVM klasifikatora

Bitno je primijetiti da dodavanjem dodatnih primjera izvan područja margine neće utjecati na pomak granice klasifikacije. Granica je potpuno opisana i samo instancama koje su na rubovima margine - **potpornim vektorima** (zaokruženi primjeri na slici 2.5b).

Također, iz samog ponašanja modela može se primijetiti da je model jako osjetljiv na različito skalirane značajke, stoga je jako bitno prije treniranja skalirati značajke.

2.3.2. Meka margina

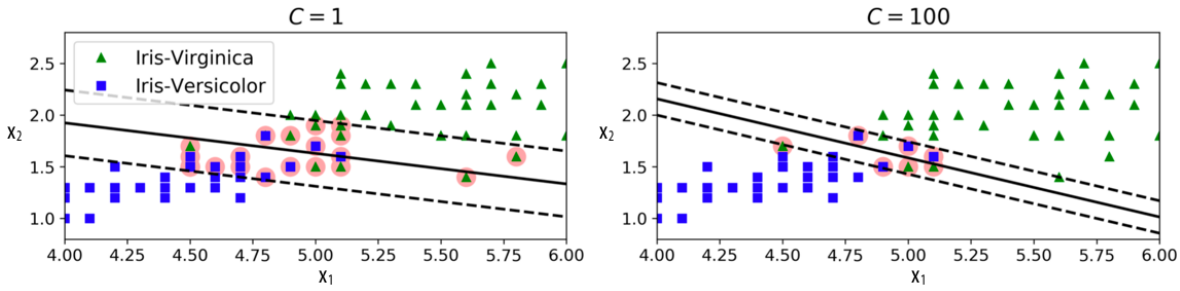
Može se primijetiti da klasifikacija tvrdom marginom donosi dva velika problema. Prvo, učinkovita je samo ako su podaci linearno odvojivi. Drugo, jako je osjetljiva na vrijednosti koje odskaču. Upravo ta dva problema su iznesena slikom 2.6.



(a) Nemoguće klasifikacija **(b)** Jako loša klasifikacija

Slika 2.6: Prikaz osjetljivosti tvrde margine SVM-a na outliere

Da bi se izbjegli ovi problemi, bolje je koristiti malo fleksibilniju varijantu SVM klasifikatora - *SVM klasifikator mekom marginom*. On održava balans između dovoljne širine margine i ograničavanja broja primjera koji se nalaze unutar same margine. Kontrola tog balansa se odvija regularizacijskim faktorom, tj. hiperparametrom C. Manji C vodi do šire margine, ali i dopušta većem broju primjera da se nađu unutar nje. Slika 2.7 prikazuje odnos regularizacijskog hiperparametra C i izgleda margine. To znači da se sprječavanje prenaučenosti modela odvija podešavanjem regularizacijom, tj. smanjivanjem vrijednosti hiperparametra C.[7]



(a) Visoka generalizacija modela **(b)** Niska generalizacija modela

Slika 2.7: Utjecaj regularizacijskog parametra C na izgled margine

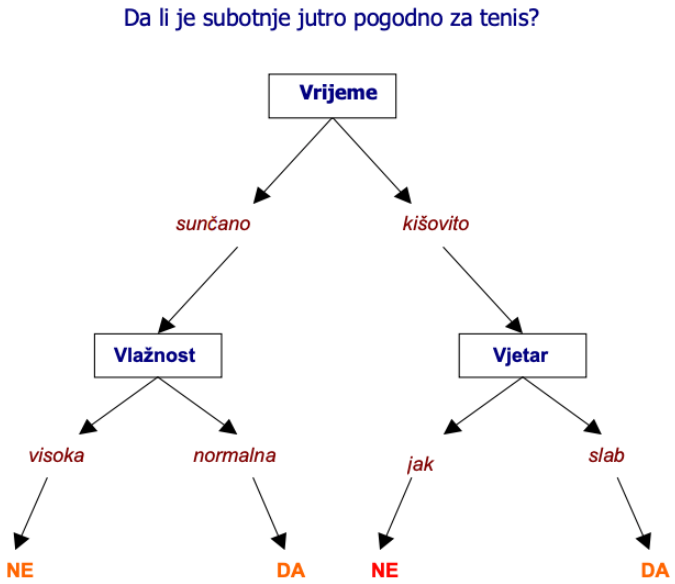
2.4. Slučajne šume

Algoritam slučajnih šuma je naziv za ansambl klasifikacijskih modela stabla odluke, generalno trenirana bagging metodom.

2.4.1. Stabla odluke

Stablo odluke (engl. *decision tree*) je osnovni gradivni blok algoritma slučajnih šuma. Sastoji se od dva osnovna dijela: čvorova i listova. Listovi su krajnji elementi stabla i oni označavaju odluku, dok su čvorovi elementi koji granaju skup podataka po nekoj značajki na dva podskupa.[6]

Na sljedećoj slici se može vidjeti primjer jednostavnog stabla odluke:



Slika 2.8: Primjer stabla odluke

Osim grananja s jednakosti u slučaju kategoričkih značajki, kao što je prikazano na slici 2.8, stablo odluke se ima mogućnost grananja i po opsegu značajki, npr. `padaline > 20mm`. Stabla odluke općenito predstavljaju disjunkciju konjunkcije uvjeta na vrijednosti atributa - *disjunktivnu normalnu formu*.

(**Vrijeme** = *sunčano* $\wedge$ **Vlažnost** = *normalna*) $\vee$ (**Vrijeme** = *oblačno* $\wedge$ **Vjetar** = *slab*)

2.4.2. Treniranje modela

Za treniranje pojedinog stabla, trening set se stvara tako da se iz podataka za treniranje uzme određeni broj primjera, ali na nasumičan način. Ti podskupi izvornog skupa podataka se nazivaju *bootstrap podskupi*. Također, pojedino stablo se ne trenira na cijelom skupu značajki, već na odabranom skupu značajki koji najbolje granaju stablo. Nakon konstrukcije odabranog broja stabala odluke, predikcija klase je većinska predikcija svih modela zajedno. Ovako opisana metoda se naziva *bagging* metodom (engl. bootstrap aggregation). Naziv je dobila s obzirom na to kako se izvodi. Iz skupa za treniranje se prave manji *bootstrap* skupovi

koji treniraju svaki model unutar ansambla zasebno, te se na kraju *agregacijom* predikcija donosi konačna odluka cijelog ansambla.[16]

2.5. Naivni Bayesov klasifikator

Bayesov klasifikator je generativan probabilistički parametarski linearni model. To je vjerojatno najpoznatiji klasifikacijski model te je tipičan predstavnik generativnih modela. Klasifikacija primjera ostvaruje se pomoću Bayesovog pravila, koji nam za svaku klasu daje vjerojatnosti kojom primjer pripada klasi.

Kod Bayesovog klasifikatora, klasifikacija primjera $\boldsymbol{x}$ temelji se na izračunu **aposteriorne vjerojatnost** $P(Y = C_k|X = \boldsymbol{x})$, tj. vjerojatnosti da primjer $\boldsymbol{x}$ pripada klasi C_k. Tu vjerojatnost izračunavamo posredno, na temelju zajedničke gustoće $p(\boldsymbol{x}, C_k)$ iz Bayesovog pravila:

$$P(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, C_k)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_k)P(C_k)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_k)P(C_k)}{\sum_{k=1}^{K} p(\boldsymbol{x}|C_k)P(C_k)} \tag{2.8}$$

Vrijednost $P(C_k)$ nazivamo **apriorna vjerojatnost klase** (engl. *class prior*), a uvjetnu gustoću $p(\boldsymbol{x}|C_k)$ nazivamo **klasom uvjetovana gustoća** (engl. *class conditional density*) ili, općenitije, **izglednost klase** (engl. *class likelihood*).

Najvjerojatnija klasifikacijska odluka jest ona koja maksimizira *aposteriornu* vjerojatnost (engl. *maximum a posteriori*). Primjer $\boldsymbol{x}$ treba klasificirati u onu klasu C_k za koju je $P(C_k|\boldsymbol{x})$ najveći. Takvu hipotezu nazivamo **maksimalna aposteriorna hipoteza** (MAP):

$$h(\boldsymbol{x}) = \underset{C_k}{\operatorname{argmax}}\, p(C_k|\boldsymbol{x}) = \underset{C_k}{\operatorname{argmax}}\, p(\boldsymbol{x}|C_k)P(C_k) \tag{2.9}$$

Budući da ulazni primjer $\boldsymbol{x}$ sadrži n značajki koje ga karakteriziraju možemo ga zapisati kao $x_1, x_2, ..., x_n$. Time 2.9 možemo zapisati na sljedeći način:

$$h(x_1, ..., x_n) = \underset{C_k}{\operatorname{argmax}}\, p(x_1, ..., x_n|C_k)P(C_k) \tag{2.10}$$

Procjena parametra $P(C_k)$ je jednostavna. Ona se izražava kao omjer broja primjera koji pripadaju klasi k i cijelog skupa primjera

$$\hat{P}(C_k) = \frac{N_k}{N} \tag{2.11}$$

tj. relativan udio primjera koji su označeni klasom C_k.

Kako bi model mogao generalizirati moramo uvesti pretpostavku da su varijable $x_1, ...x_n$

međusobno **uvjetno nezavisne** za zadanu klasu, odnosno da vrijedi

$$P(x_i|x_j, C) = P(x_i|C) \tag{2.12}$$

Ovom pretpostavkom, i primjenom pravila lanca na 2.10 dobivamo

$$P(x_1, ...x_n|C_k) = \prod_{i=1}^{n} P(x_i|x_1, ..., x_{i-1}, C_k) = \prod_{i=1}^{n} P(x_i|C_k) \tag{2.13}$$

iz čega dolazimo do modela **naivnog Bayesovog klasifikatora**

$$h(x_1, ...x_n) = \underset{C_k}{\operatorname{argmax}} P(C_k) \prod_{k=1}^{n} P(x_k|C_k) \tag{2.14}$$

Nazivamo ga naivnim upravo zbog pretpostavke o uvjetnoj nezavisnosti koja u praksi uglavnom ne vrijedi. Iako je pretpostavka sasvim sigurno pogrešna, u praksi se pokazuje da ovaj klasifikator jako dobro funkcionira. [10]

3. Programski okvir Apache Spark

3.1. Uvod u Spark

Apache Spark je programski okvir otvorenog koda namijenjen za raspodijeljenu obradu podataka koji pruža sučelje za programiranje računalnih grozdova s implicitnom podatkovnom paralelnošću i tolerancijom na pogreške.

Ovaj brzi, *RAM* orijentiran sustav za obradu podataka s elegantnim izražajnim *API*-jem za razvoj omogućuje računalnim znanstvenicima učinkovitu obradu protočnih, strojno učenih ili *SQL* opterećenja koja zahtijevaju brz interaktivan pristup skupovima podataka. Drugim riječima, omogućava optimizirano particioniranje velikih količina podataka na raspodijeljena računala računalnog grozda na kojima se odvijaju razne operacije nad particioniranim podacima [14].

3.2. Raspodijeljene strukture podataka

Računalnim znanstvenicima jedno od najdražih svojstava Sparka je upravo to što pruža *API*-je koji su laki za korištenje nad velikim skupovima podataka kroz više programskih jezika: *Scala*, *Java*, *Python* i *R*. U ovom potpoglavlju proći ćemo kroz tri različita omotača nad podacima: *RDD*, *DataFrame* i *Dataset*.

3.2.1. Resilient Distributed Dataset (RDD)

RDD (hrv. *elastični distribuirani set podataka*) je primarni *API* koji su programeri koristili u *Sparku* od svog osnutka. U svojoj jezgri, *RDD* je nepromjenjivi distribuirani set podataka, particioniran preko čvorova u računalnom grozdu nad kojima se operacije mogu izvršavati paralelno pružajući *API* niske razine koji nudi transformacije[1] i akcije[2].[1].

[1] transformacije (eng. *transformations*) - funkcije koje transformiraju set podataka (npr. `map`, `filter`, `union`). *U pravilu se izvršavaju na svakom čvoru zasebno.*

[2] akcije (eng. *actions*) - funkcije koje ne mijenjaju set podataka već daju neki izlaz (npr. `reduce`, `collect`, `count`). Zbog samih operacija koje rade, *zahtijevaju dohvat svih podataka na jedan čvor.*

Korištenje *RDD*-ova najviše se preporučuje u sljedećim situacijama:

- želite imati kontrolu primarnih transformacija i akcija nad svojim setom podataka
- vaši podaci su nestrukturirani (npr. medijske datoteke, protok teksta)
- nije vam potrebno nametanje definirane sheme nad podacima (poput stupčastog formata)
- možete se odreći nekih optimizacijskih pogodnosti koje su dostupne u izvedbi s *DataFrame*-ovima i *Dataset*-ovima za strukturirane.

3.2.2. DataFrame

Baš kao i *RDD*, *DataFrame* je također nepromjenjiva distribuirana kolekcija podataka. Prednost *DataFrame*-a nad *RDD*-om se očituje kad su u pitanju strukturirani podaci. *DataFrame* je organiziran u imenovane stupce, kao tablica u relacijskim bazama podataka (svaki stupac sadrži jednu mjeru, svaki redak označava jedan slučaj). Osmišljen je upravo da omogući programerima nametanje strukture nad kolekcijom podataka kojom manipuliraju te upravo tim pristupom omogućava Sparku da bude dostupan i široj količini developera, ne samo inženjerima specijaliziranima za obradu podataka. Uz same podatke koje *DataFrame* pohranjuje, on sadrži i neke meta-podatke kojima omogućava Sparku da izvrši optimizacije nad finaliziranim upitom (prisjetimo se da se Spark upiti izvršavaju s odgodom te upravo zbog toga su i moguće naknadne optimizacije).[1].

3.2.3. Dataset

Uvedeni tek u verziji Sparka 2.0, *Dataset*-ovi koriste dva potpuno odvojena *API*-ja: snažnotipizirani *API* (eng. *strongly-typed API*) i netipizirani *API* (eng. *untyped API*). Konceptualno, možemo zamisliti *DataFrame* kao pandan za kolekciju generičkih (netipiziranih) `Dataset[Row]` objekata, gdje je `Row` generički netipizirani *JVM*[3] objekt. *Dataset* je upravo suprotan koncept od *DataFrame* - kolekcija tipiziranih *JVM* objekata (tip im je definiran klasom).

Preporučuje se korištenje *DataFrame*-ova i *Dataset*-ova gdje god je moguće zbog dostupnosti već ugrađene optimizacije upita neposredno prije izvršavanja (koju *RDD*-ovi nemaju) [1].

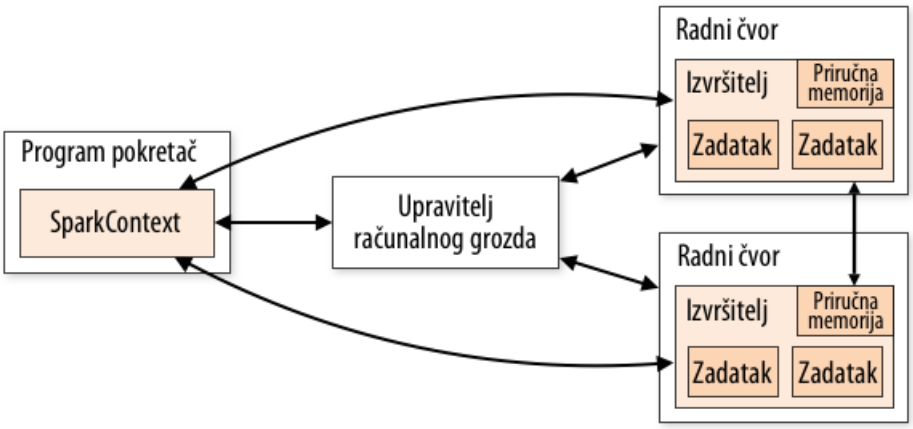
[3] engl. *Java virtual machine*

3.3. Raspodijeljeno izvršavanje Apache Spark programa

Apache Spark koristi **gospodar/radnik arhitekturu** (eng. *master/worker architecture*) kao što prikazuje slika 3.1.

Čvor pokretač (eng. *driver node*) je proces koji upravlja, nadzire, koordinira i raspoređuje zadatke procesima izvršiteljima (eng. *executors*). To su procesi koji se izvode na ostalim računalima unutar računalnog grozda - radnim čvorovima (engl. *worker nodes*). Procesi izvršitelji tijekom odrađivanja zadataka čvoru pokretaču dojavljuju trenutno stanje obrade podataka. Razlog konstantnog obavještavanja je održavanje sigurnosti pravovremenim reagiranjem i primjenom mehanizama oporavka u slučaju ispada čvora ili pak, u većini slučajeva, zbog praćenja opterećenja radnih čvorova te daljnjeg raspoređivanja preostalih zadataka. Uvidom u strukturu podataka, Spark smišlja **fizički plan raspodjele** podataka i izvršavanja naredbi nad znanom arhitekturom računalnog grozda. Naredbe koje se mogu izvršavati neovisno o drugim procesima se grupiraju u zadatke. Zadatak se tako replicira kroz radne čvorove. Takve replike određenog zadatka su objedinjene nazivom faza (engl. *stage*) te se izvršavaju unutar svakog radnog čvora na različitim particijama podataka. [15]

Na slici 3.1 je skicirana arhitektura *Apache Spark* računalnog grozda sastavljenog od jednog čvora s pokretačkim procesom te dva radna čvora s po jednim izvršiteljem[4].



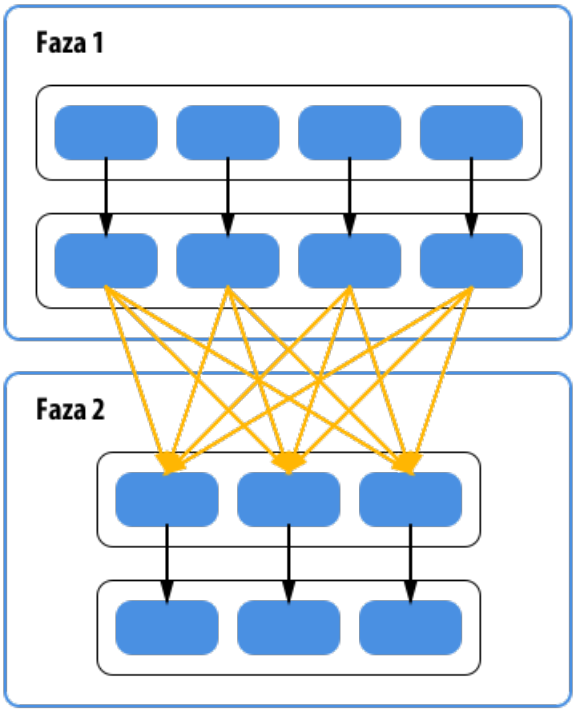
Slika 3.1: Vrste čvorova u Apache Spark računalnom grozdu

3.3.1. Međuprocesna komunikacija Apache Spark računalnog grozda

Međuprocesna komunikacija, poznatija kao **faza miješanja podataka** (engl. *data shuffling phase*), je razmjena podataka između procesa. Ona je vremenski najskuplji dio izvršavanja Spark programa. Njeno odvijanje preko mreže ili spremanjem i učitavanjem s dostupne vanjske memorije uzrokuje znatno usporavanje izvedbe programa. Do faze miješanja podataka,

[4] na jednom radnom čvoru moguće je imati više izvršiteljskih procesa

odnosno razmjene blokova podataka koji sudjeluju u miješanju (engl. *shuffle blocks*) dolazi zbog izvršavanja operacija koje zahtijevaju znanje cijelog seta podataka koji se trenutno obrađuje. Svaka žuta strelica na slici 3.2 predstavlja razmjenu jednog bloka podataka [11]. Ova faza je neizbježna, ali nepažljiva i nepromišljena upotreba takvih operacija dovodi do nekvalitetnog iskorištavanja resursa računalnog grozda, odnosno korištenje takvih operacija mora biti svedeno na minimum.



Slika 3.2: Slikoviti prikaz razmjene blokova podataka između faza

Lokalnim izvršavanjem nekvalitetno napisane i optimizirane *Apache Spark* aplikacije znatno se osjete performanse izvođenja programa. Detalji o ovom svojstvu rada *Apache Spark* programskog okvira te njegov utjecaj, skaliranje te održavanje računalnog grozda mogu biti pronađeni u radu *"Optimizing Shuffle Performance in Spark"* (hrv. *Optimizacija performansi miješanja Sparka*)[8].

3.4. Spark MLlib programska knjižica

Spark MLlib je programska knjižica *Apache Sparka* koja sadrži implementacije algoritama strojnog učenja koji su namijenjeni isključivo za strojno učenje na raspodijeljenim sustavima. Cilj biblioteke je pojednostaviti razvoj i uporabu strojnog učenja nad velikim količinama podataka.

U biblioteci *MLlib* možemo susresti sljedeće vrste algoritama strojnog učenja:

- klasifikacijski algoritmi

– regresijski algoritmi
– algoritmi čestih stavki
– preporučiteljski algoritmi
– algoritmi vađenja i odabira značajki
– algoritmi grupiranja
– statistički algoritmi
– algoritmi linearne algebre

Kao što je već navedeno u uvodu, sadržaj ovog diplomskog rada bit će fokusiran na klasifikacijske algoritme. Teorijska pozadina istih je nešto detaljnije objašnjena u poglavlju 2.

Bitno je naglasiti da unutar *Spark MLlib*-a postoje dvije biblioteke:

– `org.apache.spark.mllib` - za *RDD* bazirane algoritme strojnog učenja
– `org.apache.spark.ml` - za *DataFrame* bazirane algoritme strojnog učenja

`org.apache.spark.ml` je novija biblioteka te se preporučuje njezino korištenje ispred `org.apache.spark.mllib` biblioteke (zbog toga što koristi *DataFrame*-ove umjesto *RDD*-ova).

3.5. Središnji koncepti

Postoji 5 bitnih koncepata za shvatiti prije korištenja `ml` biblioteke:

– **DataFrame** - objašnjen ranije u poglavlju
– **Transformator** - (engl. *transformer*) algoritam koji transformira jedan *DataFrame* u drugi implementirajući metodu `transform()`. Podijeljeni su u dva tipa: *transformatori značajki* (normalizacija, sažimanje, ...) i *naučeni modeli* (vjerojatnosna interpretacija).
– **Procjenitelj** - (engl. *estimator*) algoritam koji je treniran nad *DataFrame*-om implementirajući metodu `fit()` vraća naučeni model
– **Parametri** - skup vrijednosti koje karakteriziraju učenje algoritma
– **Cjevovod** - (engl. *pipeline*) slijed faza Transformatora i Procjenitelja koji se izvršavaju jedan iza drugoga. Cjevovod je sam po sebi procjenitelj te treniran nad *DataFrame*-om vraća naučeni model.

```
// Ucitavanje podataka iz datoteke u DataFrame
val df = spark.read.csv("data.csv")

// Mijenjanje kategoricke varijable numerickom zbog potrebe modela.
// StringIndexer je procjenitelj koji vraca transformator
   StringIndexerModel na poziv metode fit()
val labelIndexer = new StringIndexer()
  .setInputCol("label")
  .setOutputCol("indexedLabel")
  .fit(df)

// Automatsko indeksiranje kategorickih znacajki
// VectorIndexer je procjenitelj koji vraca transformator
   VectorIndexerModel na poziv metode fit()
val featureIndexer = new VectorIndexer()
  .setInputCol("features")
  .setOutputCol("indexedFeatures")
  .fit(df)

// Rastavljanje podataka na train i test skupove
// ne uzima u obzir disbalansiranost klasa
val Array(trainingDF, testDF) = df.randomSplit(Array(0.7, 0.3))

// Procjenitelj. Postavljanje parametara klasifikacijskog modela
   nasumicne sume.
val rf = new RandomForestClassifier()
  .setLabelCol("indexedLabel")
  .setFeaturesCol("indexedFeatures")
  .setNumTrees(10)
  .setMaxDepth(5)

// Vracanje indeksiranih vrijednosti nazad u kategoricke
val labelConverter = new IndexToString()
  .setInputCol("prediction")
  .setOutputCol("predictedLabel")
  .setLabels(labelIndexer.labels)

// Cjevovod. Definicija redosljeda izvrsavanja operacija.
val pipeline = new Pipeline()
```

```
37    .setStages(Array(labelIndexer, featureIndexer, rf,
          labelConverter))
38
39  // Treniranje modela cjevovoda
40  val model = pipeline.fit(trainingDF)
41
42  // Transformator. Dobivanje predikcija iz modela.
43  val predictionsDF = model.transform(testDF)
44
45  // Ispis predikcija na standardni izlaz
46  predictionsDF.select("predictedLabel", "label", "features").show(5)
```

Kôd 3.1: Prikaz središnjih koncepata na primjeru

4. Problem klasifikacije korisnika mobilnih uređaja

Implementacija samog klasifikacijskog algoritma, a kasnije i njegova evaluacija, izvest će se na *TalkingData Mobile User Demographics* setu podataka. Ovaj set podataka je javno dostupan na *Kaggle*-u[1] .

Podatci su prikupljani preko TalkingData SDK modula integriranog u korisničke aplikacije dijela populacije *Narodne Republike Kine*. Dobiveno je potpun pristanak svakog pojedinog korisnika tih aplikacija te je izvršena i odgovarajuća anonimizacija radi zaštite privatnosti. Zbog povjerljivosti *TalkingData* ne daje pojedinosti o tome na koji način su podaci o spolu o dobi prikupljeni, osim te da se to odvijalo kroz njihov instalirani SDK modul. Također daju doznanja da se prilikom korištenja i obrade podataka možemo osloniti na to da predstavljaju temeljnu istinu za predviđanje, odnosno klasifikaciju.[2]

Benefiti mogućnosti klasifikacije korisnika mobilnih uređaja s obzirom na instalirane i korištene aplikacije na njihovim uređajima su brojni. Jedno od mogućnosti je korištenje klasifikatora za specifično ciljanje korisnika s obzirom na spol i/ili dobnu skupinu kojoj pripadaju u svrhu marketinga i objavljivanja oglasa samo za ciljanu skupinu ljudi.

4.1. Opis podataka

Preuzeti podatci sa *Kaggle*-a su raspoređeni u 8 *csv* datoteka. Dodatni opis značajki podataka te njihova poveznica između datoteka iznesena je u tablici 5.3, odnosno na slici 4.1.

- *gender_age_train.csv* - datoteka koja sadrži skup podataka za treniranje modela. Sadrži sljedeće parametre o podacima: `device_id, gender, age, group`.
- *gender_age_test.csv* - datoteka koja sadrži skup podataka za testiranje modela i slanje rezultata na validaciju za natjecanje. Sadrži samo `device_id` te ju zbog toga nećemo koristiti.

[1] *Kaggle* - online platforma za natjecanja u područjima podatkovne znanosti (`https://www.kaggle.com`)

– *app_events.csv* - datoteka koja povezuje aplikaciju i događaj slanja stanja s uređaja kroz *TalkingData SDK*. Sadrži sljedeće parametre o podacima: `event_id`, `app_id`, `is_installed`, `is_active`.

– *events.csv* - datoteka koja sadrži sve poslane događaje. Sadrži sljedeće parametre o podacima: `event_id`, `device_id`, `timestamp`, `longitude`, `latitude`.

– *app_labels.csv* - datoteka koja sadrži poveznicu između aplikacije i kategorije kojima određena aplikacija pripada. Sadrži sljedeće parametre o podacima: `app_id`, `label_id`.

– *label_categories.csv* - datoteka koja sadrži imena kategorija. Sadrži sljedeće parametre o podacima: `label_id`, `category`.

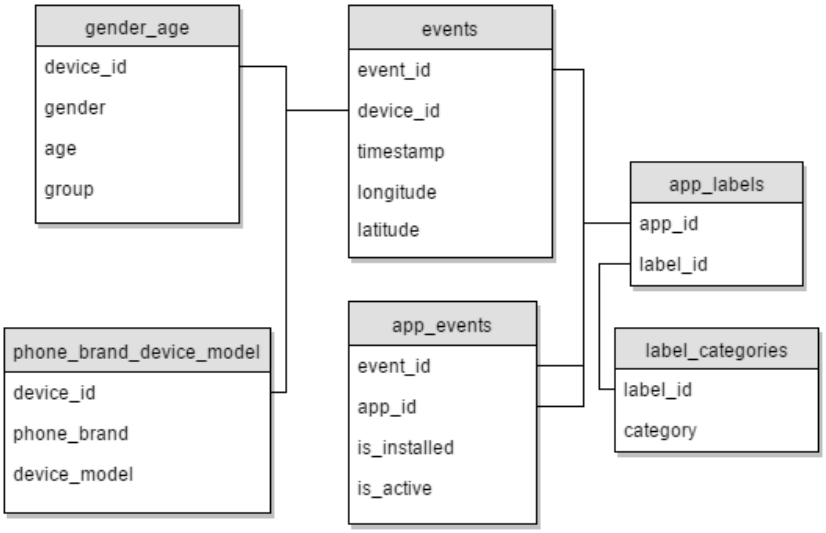
– *phone_brand_device_model.csv* - datoteka koja sadrži opis korištenih uređaja. Sadrži sljedeće parametre o podacima: `device_id`, `phone_brand`, `device_model`.

– *sample_submission.csv* - datoteka koja je primjer predaje rezultata modela. Ovu datoteku neće biti korištena u radu.

Naziv parametra	**Opis parametra**	**Tip podatka**
`device_id`	jedinstveni identifikator uređaja	cjelobrojni
`app_id`	jedinstveni identifikator aplikacije	cjelobrojni
`event_id`	jedinstveni identifikator događaja stanja poslanog kroz *TalkingData SDK*	cjelobrojni
`label_id`	jedinstveni identifikator kategorije	cjelobrojni
`gender`	rod korisnika uređaja	tekstualni
`age`	starost korisnika uređaja	cjelobrojni
`group`	demografska skupina u koju spada korisnik; sastavljena od inicijala roda na engleskom jeziku i starosne skupine (npr. `F29-32`)	tekstualni
`timestamp`	vrijeme slanja stanja uređaja	tekstualni
`longitude`, `latitude`	koordinate lokacije u trenutku promatranja stanja	decimalni
`phone_brand`	marka mobitela koju korisnik koristi	tekstualni
`device_model`	model mobitela koju korisnik koristi	tekstualni
`is_installed`	značajka koja opisuje je li aplikacija instalirana u trenutku promatranja stanja	cjelobrojni $(1/0)$
`is_active`	značajka koja opisuje je li aplikacija aktivna u trenutku promatranja stanja	cjelobrojni $(1/0)$
`category`	naziv kategorije	tekstualni

Tablica 4.1: Opis parametara seta podataka

Zbog toga što datoteka za uređaje iz datoteke *gender_age_test.csv* ne postoje izvorne oznake istine u koje model treba klasificirati korisnike te je nemoguće validirati ispravnost klasifikacije, uređaji iz datoteke *gender_age_train.csv* će biti raspodijeljeni u tri skupa podataka: skup za treniranje modela, validacijski skup i testni skup. Implementacija podijele te uloge pojedinog skupa su opisane u poglavlju 5.



Slika 4.1: Prikaz značajki po datotekama te njihova međusobna povezanost preko primarnih i sekundarnih ključeva.

4.2. Analiza podataka

Originalno je zamišljeno da se za svakog korisnika predviđa jedna od sljedećih klasa:

`F23-`, `F24-26`, `F27-28`, `F29-32`, `F33-42`, `F43+`, `M22-`, `M23-26`, `M27-28`, `M29-31`, `M32-38`, `M39+`

Svaku klasu označava slovo koje predstavlja spol korisnika unutar klase - `M` (engl. *male*) za muškarce i `F` (engl. *female*) za žene. Također, klasa u svojem imenu sadrži i dobni raspon korisnika unutar iste - `33-42` označava korisnike između 33 i 42 (uključivo) godine.

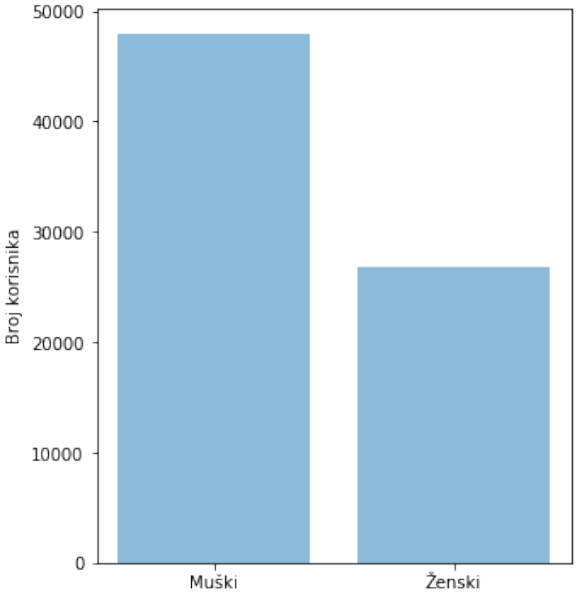
Radi jasnijih rezultata, lakšeg razumijevanja, analize i obrade podataka u sklopu ovog diplomskog bit će razvijena dva zasebna klasifikatora po:

- ***spolu korisnika*** u klase `M` (muško) i `Ž` (žensko)
- ***dobnoj skupiti korisnika*** u klase `0-22`, `23-26`, `27-28`, `29-32`, `33-39` i `40+`.

Upravo tako, na dva dijela, ćemo podijeliti analizu i odabir značajki - s obzirom na spol te s obzirom na dobnu starost korisnika. Analiza podataka je zahtjevniji dio procesa izrade klasifikacijskog modela te ona oduzima otprilike $\frac{3}{4}$ ukupnog vremena. Krivim odabirom i obradom značajki dolazi do znatno lošijih rezultata u fazi evaluacije klasifikatora.

4.2.1. Analiza značajki sprema spolu

Za razumijevanje sveobuhvatne analize bitno je vidjeti raspodjelu labele koju će klasifikator predviđati, klasu spola u setu podataka.

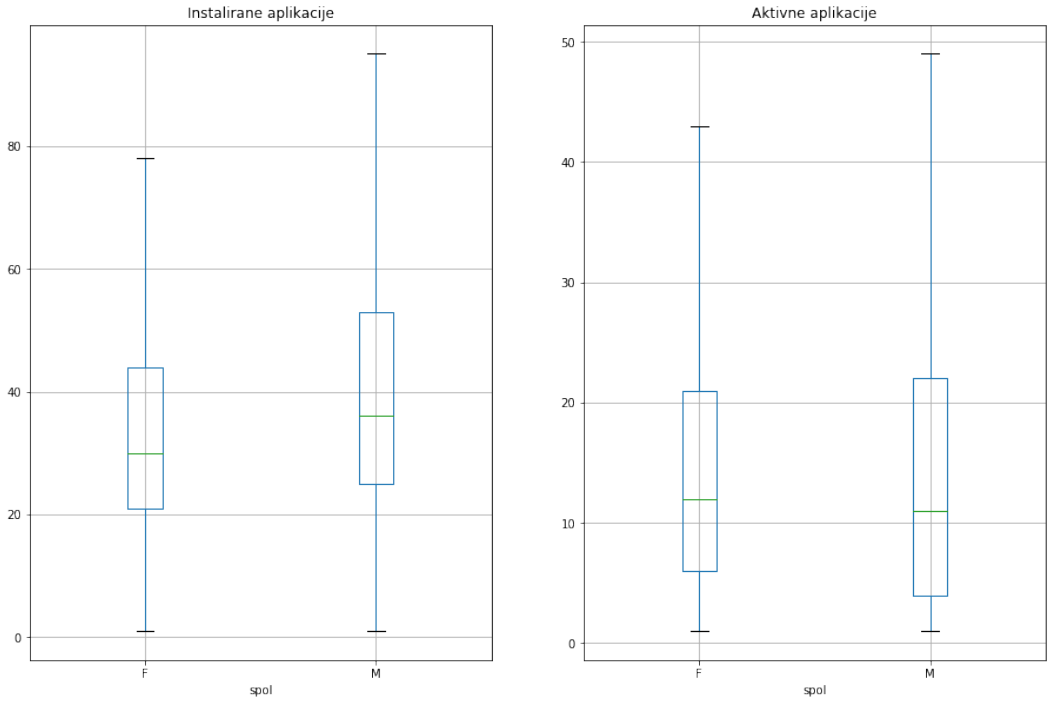


Slika 4.2: Raspodjela populacije korisnika po spolu

Može se primijetiti da u setu podataka muškaraca ima duplo više nego žena. Upravo zbog te disbalansiranosti klasa koje se predviđaju morat će biti posvećena dodatna pažnja podijeli podataka na skup podataka za treniranje i skup podataka za testiranje. Više o toj podijeli je napisano u poglavlju o samoj implementaciji programa.

Broj aplikacija na uređaju

Na slici 4.3 možemo vidjeti raspodjelu prosječnog broja instaliranih i aktivnih aplikacija na uređajima korisnika. Iz grafa, možemo zaključiti da su muškarci ti koji u prosjeku imaju više instaliranih aplikacija, ali da ih jako malo koriste. Također, zbog jednostavnije čitljivosti grafa, priložena je i tablica 4.2 koja konkretnije opisuje granice kutijastog dijagrama.



Slika 4.3: Prikaz raspodijele prosječnog broja instaliranih i aktivnih aplikacija

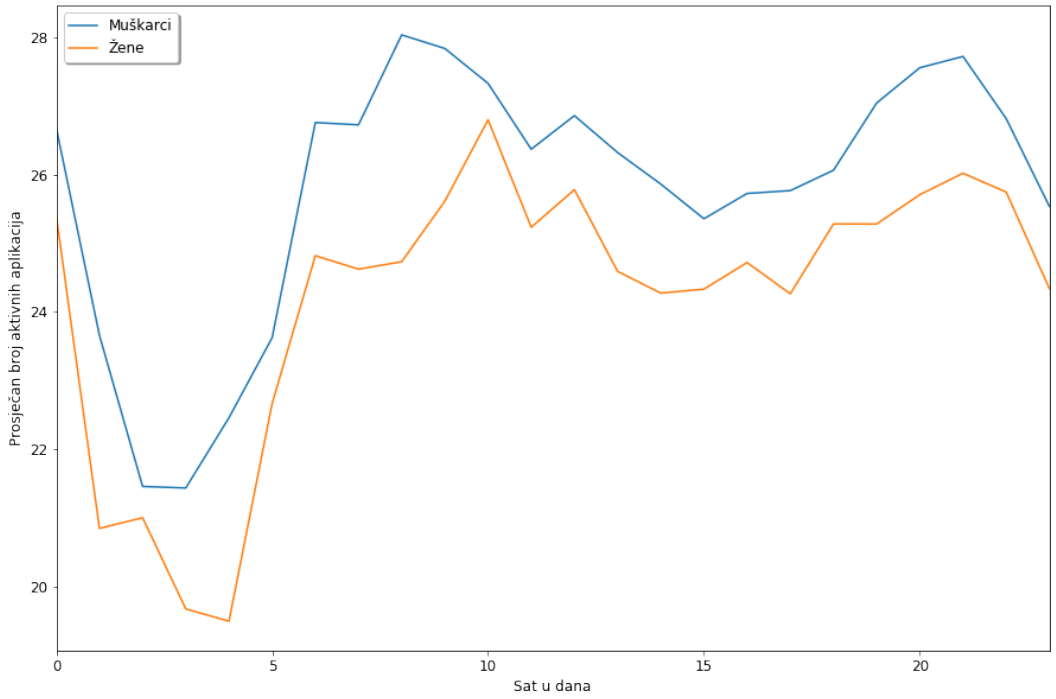
	Instalirane aplikacije		Aktivne aplikacije	
	Muškarci	Žene	Muškarci	Žene
srednja vrijednost	41.697	34.855	15.173	14.772
standardna devijacija	24.378	20.504	13.757	12.139
minimum	1	1	1	1
Q1	25	21	4	6
median	36	30	11	12
Q3	53	44	22	21
maksimum	247	224	127	148

Tablica 4.2: Tablični prikaz raspodijele prosječnog broja instaliranih i aktivnih aplikacija

Korištenje aplikacija kroz dan

Korištenje aplikacija kroz dan bit će analizirano s obzirom na dva parametra iz tablice 5.3 koja opisuje podatke: `timestamp` i `is_active`.

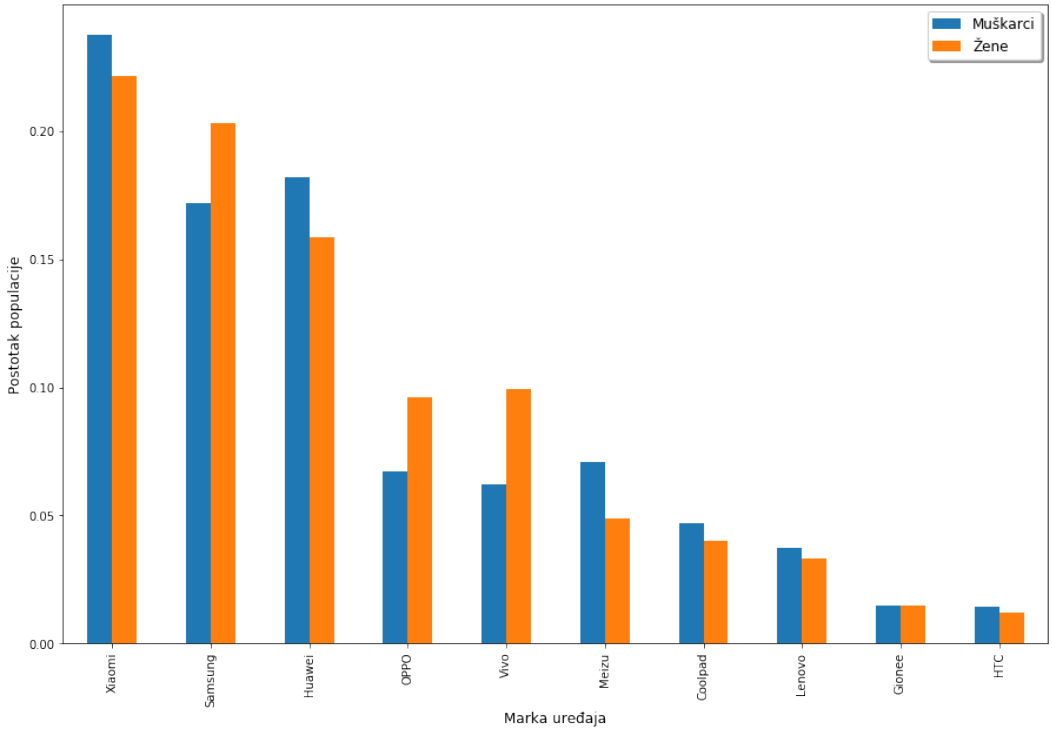
Graf sa slike 4.4 prikazuje upravo taj omjer - prosječan broj aktivnih aplikacija u danu s obzirom na spol korisnika uređaja. Može se primijetiti da prosječno muškarci imaju više aktivnih aplikacija na svojim uređajima te je to razlog zašto ćemo ove dvije značajke - *najaktivniji sat u danu korisnika* i *prosječan broj aktivnih aplikacija u to vrijeme* - koristiti u kasnijem predviđanju spola korisnika.



Slika 4.4: Prosječan broj aktivnih aplikacija na uređaju kroz dan po spolu

Najčešće korištene marke uređaja

Iz grafa sa slike 4.5 možemo primijetiti znatnu razliku u vlasništvu uređaja određene marke između muškaraca i žena. Uzmimo za primjer uređaje marke *Meizu* koji se u populaciji žena nalaze na 4. mjestu najčešće korištenih marki uređaja dok su u muškoj populaciji zauzeli 6. mjesto. Napomenimo da ovaj graf, koji prikazuje deset najčešće korištenih marki uređaja, obuhvaća 90.60% populacije muškaraca, te 92.78% populacije žena.



Slika 4.5: Deset najčešće korištenih marki uređaja

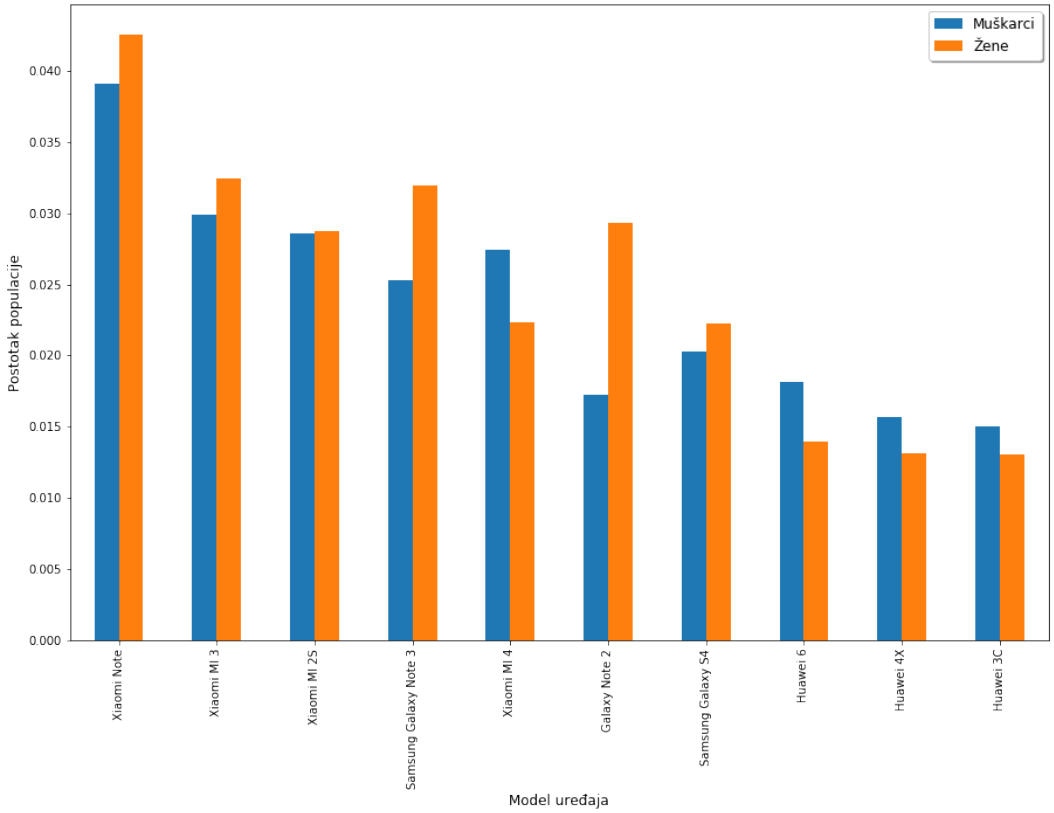
Najčešće korišteni modeli uređaja

Za razliku od najčešće korištenih uređaja, koji obuhvaćaju više od 90% populacije korisnika i unatoč tome što iz grafa sa slike 4.6 možemo primijetiti znatnu razliku u korištenju pojedinih modela između populacije muškaraca i žena, bilo bi dobro uočiti da suma svih deset najčešće korištenih modela uređaja ne prelazi 25% ukupne populacije korisnika[2]. Upravo je to razlog zašto se ova značajka o korištenu pojedinog modela uređaja neće naći u krajnjem odabiru značajki za treniranje modela predikcije spola.

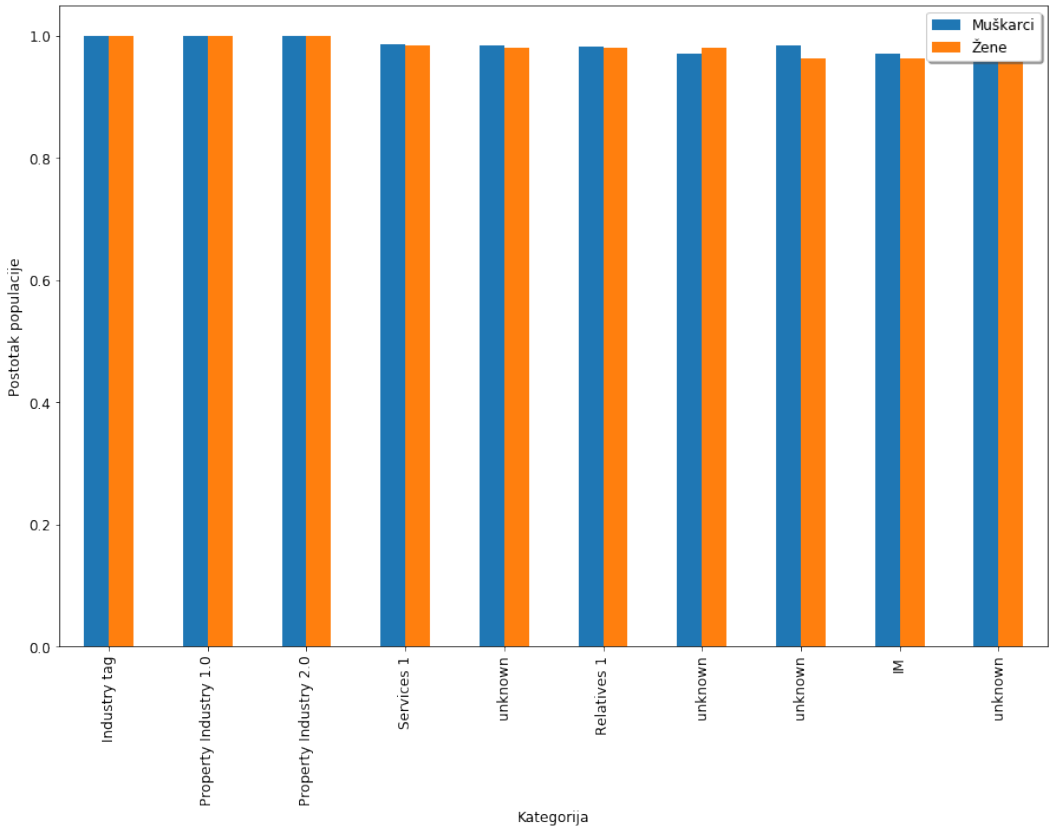
Najčešće kategorije aplikacija na mobilnim uređajima

Jedno od bitnijih značajki za predviđanje spola korisnika su aplikacije koje su instalirane na njihovim uređajima. Umjesto analize samih aplikacija, u ovom potpoglavlju će biti napravljena analiza najčešćih kategorija aplikacija koje se koriste na uređajima. U cijelom datasetu postoji 930 različitih kategorija. Slika 4.7 prikazuje postotak zastupljenosti deset najzastupljenijih kategorija u populaciji muškaraca i žena. Iz samoga grafa se vidi da je zastupljenost jednolika, odnosno jako se malo razlikuje. Upravo zbog toga, umjesto korištenja značajke prisutnosti kategorije kod korisnika, koristit ćemo značajku koja nam opisuje koliko aplikacija te kategorije korisnik ima instalirano na svom mobilnom uređaju.

[2] 23.66% za mušku, odnosno 24.96% za žensku populaciju

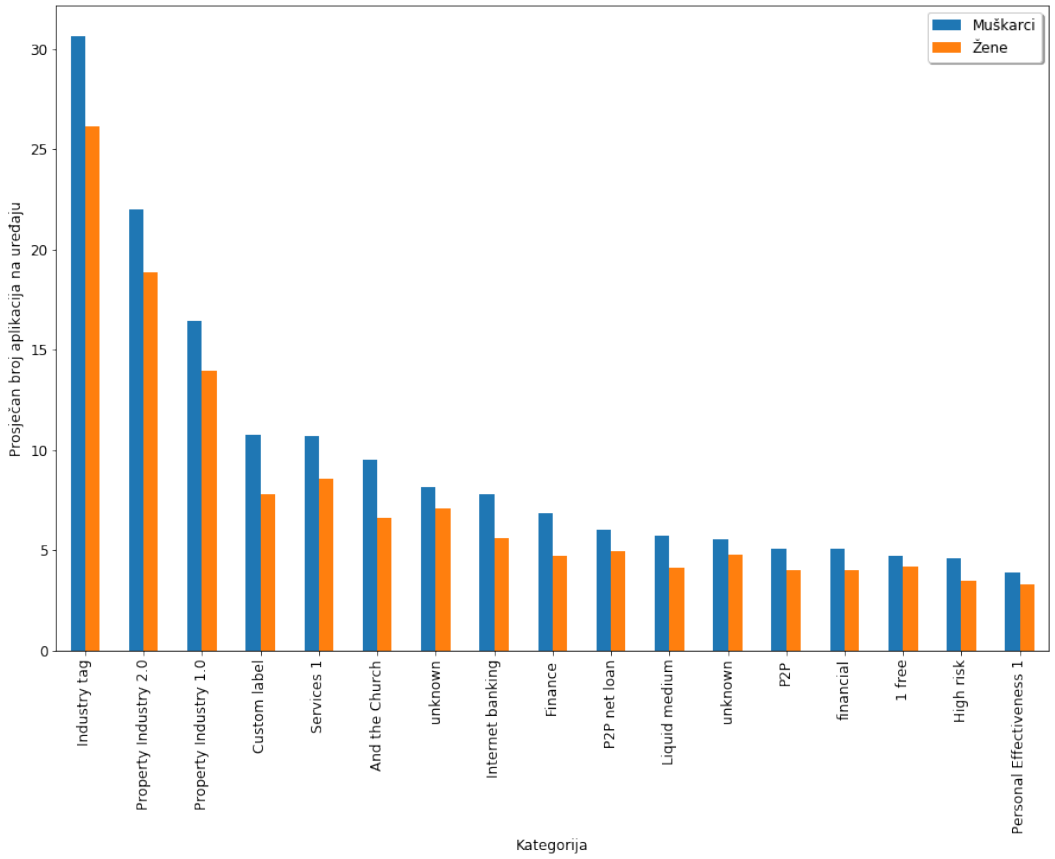


Slika 4.6: Deset najčešće korištenih modela uređaja



Slika 4.7: Deset najzastupljenijih kategorija aplikacija

Kao što se može primijetiti iz grafa sa slike 4.8, prosječan muškarac redovito ima veći broj aplikacija pojedine kategorije instalirano na svom uređaj.

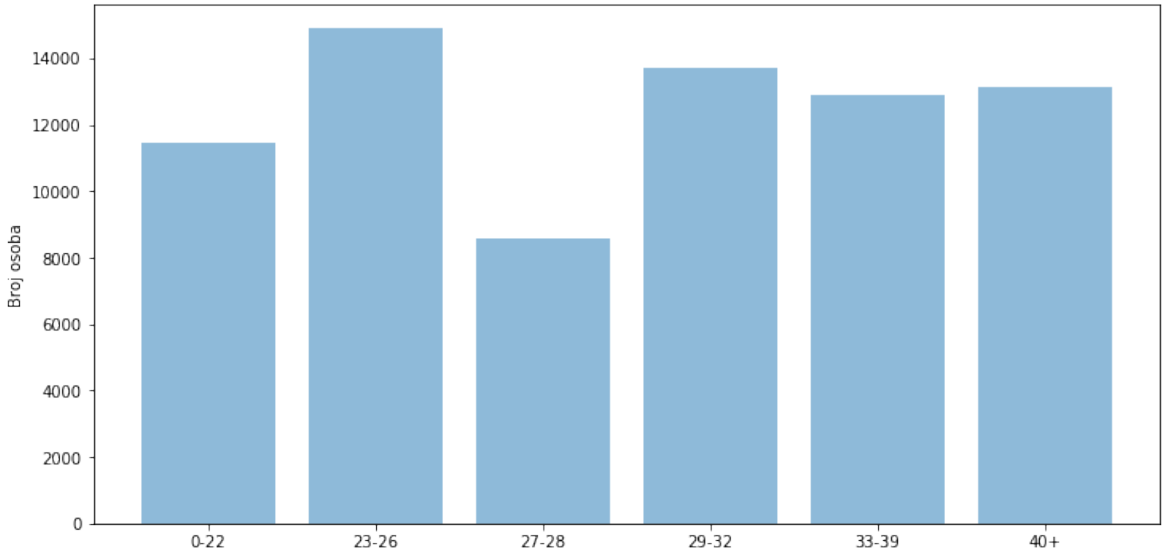


Slika 4.8: Prosječan broj instalacija dvadeset najzastupljenijih kategorija aplikacija

4.2.2. Analiza značajki prema dobnoj skupini

Jednako kako je provedena analiza značajki s obzirom na spol korisnika, u ovom potpoglavlju će biti napravljena analiza istih značajki s obzirom na dobnu skupinu korisnika. Korisnici su podijeljeni u sljedeće dobne skupine: `0-22`, `23-26`, `27-28`, `29-32`, `33-39`, `40+`.

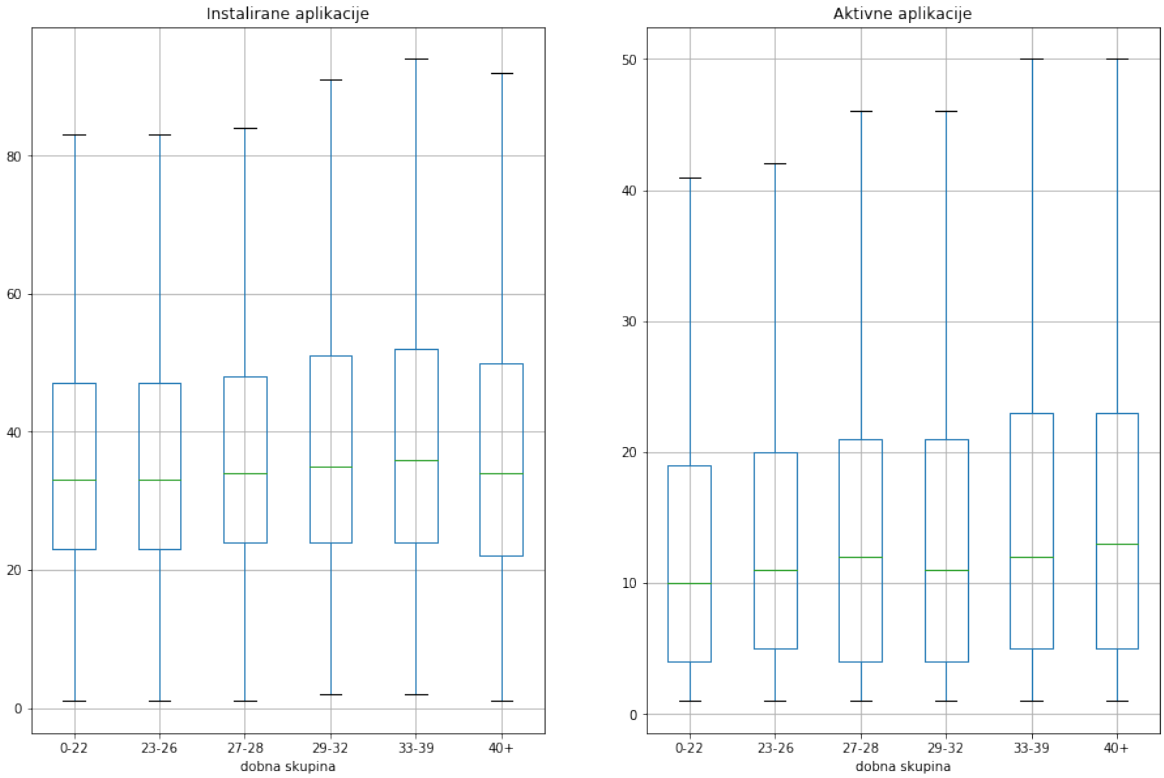
Ova podjela je preuzeta od originalne podijele u grupe iz zadatka samo su maknute oznake spola kao što je već navedeno na samom početku poglavlja 4.2. Pretpostavka je da su korisnici upravo ovako grupirani zbog sličnosti ponašanja i korištenja uređaja unutar navedenih skupina.



Slika 4.9: Raspodjela populacije korisnika po dobnim skupinama

Broj aplikacija na uređaju

Iz slike 4.10 možemo vidjeti raspodjelu prosječnog broja instaliranih i aktivnih aplikacija na uređajima korisnika. Iz grafa, možemo zaključiti da su muškarci ti koji u prosjeku imaju više instaliranih aplikacija, ali da ih jako malo koriste. Također, zbog jednostavnije čitljivosti grafa, priložene su i tablice 4.3 i 4.4 koje konkretnije opisuje granice kutijastih dijagrama.



Slika 4.10: Prikaz raspodijele prosječnog broja instaliranih i aktivnih aplikacija ovisno o populaciji

	Dobne skupine					
	0-22	23-26	27-28	29-32	33-39	40+
srednja vrijednost	37.20	37.55	38.54	40.41	41.53	39.66
standardna devijacija	20.711	21.624	21.511	24.307	24.534	25.026
minimum	1	1	1	2	2	1
Q1	23	23	24	24	24	22
median	33	33	34	35	36	34
Q3	47	47	48	51	52	50
maksimum	190	219	197	247	216	233

Tablica 4.3: Tablični prikaz raspodijele prosječnog broja instaliranih aplikacija po dobnim skupinama

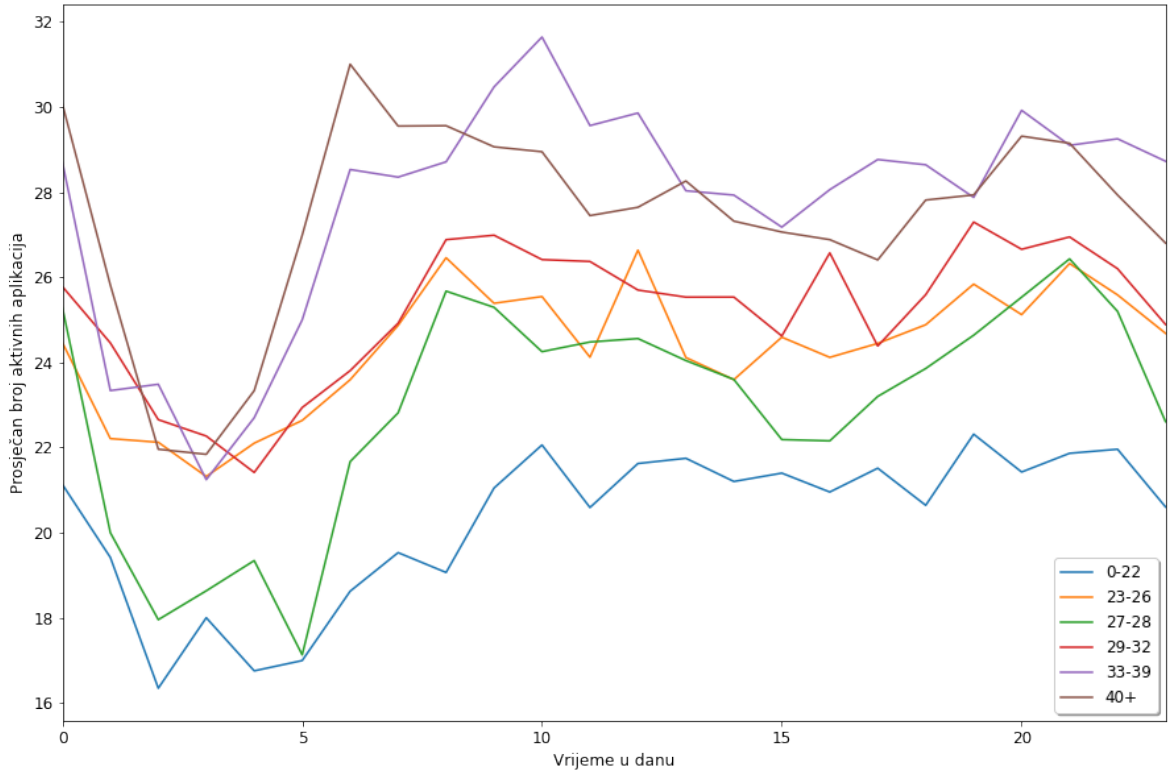
	Dobne skupine					
	0-22	23-26	27-28	29-32	33-39	40+
srednja vrijednost	13.20	14.29	14.59	14.73	16.01	16.50
standardna devijacija	11.296	12.193	12.452	13.005	14.145	14.695
minimum	1	1	1	1	1	1
Q1	4	5	4	4	5	5
median	10	11	12	11	12	13
Q3	19	2	21	21	23	23
maksimum	99	116	98	108	95	148

Tablica 4.4: Tablični prikaz raspodijele prosječnog broja aktivnih aplikacija po dobnim skupinama

Korištenje aplikacija kroz dan

Jednako kao i u analizi odnosa spola pojedinca i aktivnog korištenja aplikacija kroz dan, i u ovoj analizi uzimamo u obzir iste opisne parametre podataka: vrijeme zabilježavanja stanja na uređaju - `timestamp` te opisnu oznaku koja označava aktivnost aplikacije - `is_active`.

Graf sa slike 4.11 prikazuje prosječan broj aktivnih aplikacija u danu s obzirom na dobnu skupinu kojoj korisnik uređaja pripada. Iz grafa se može očitati da mlađe dobne skupine imaju znatno manji broj aktivnih aplikacija na uređaju kroz dan od starijih.



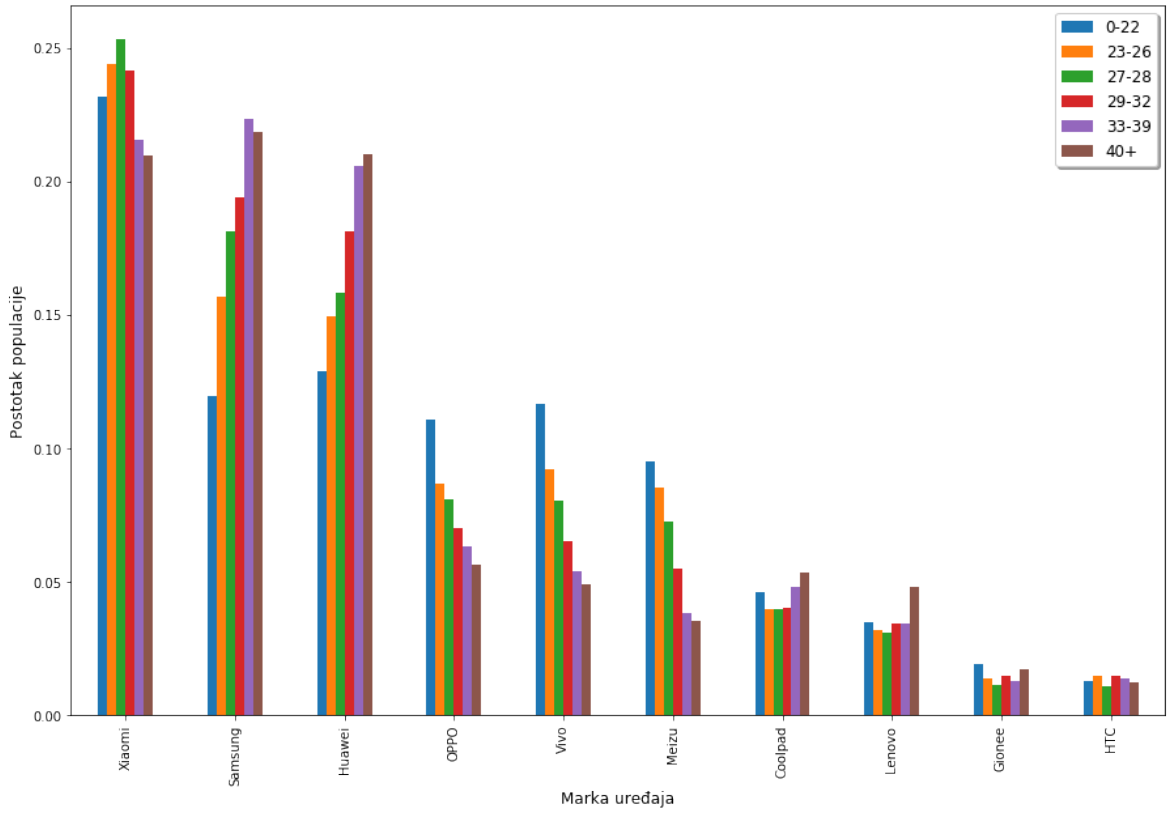
Slika 4.11: Prosječan broj aktivnih aplikacija na uređaju kroz dan po dobnoj skupini

Najčešće korištene marke uređaja

Slika 4.12 prikazuje top 10 najčešće korištenih marki uređaja. Primjećuje se znatna razliku u vlasništvu uređaja određene marke između različitih populacija. Na primjer uređaji marke *Xiaomi* koji se u populaciji `27-28` najčešće koriste te su u samom vrhu najčešće korištenih marki uređaja, upravo ta marka je na trećem mjestu kod starije populacije `40+`.

Ovih deset najčešće korištenih marki uređaja, obuhvaćaju više od 90% ukupne populacije, konkretnije:

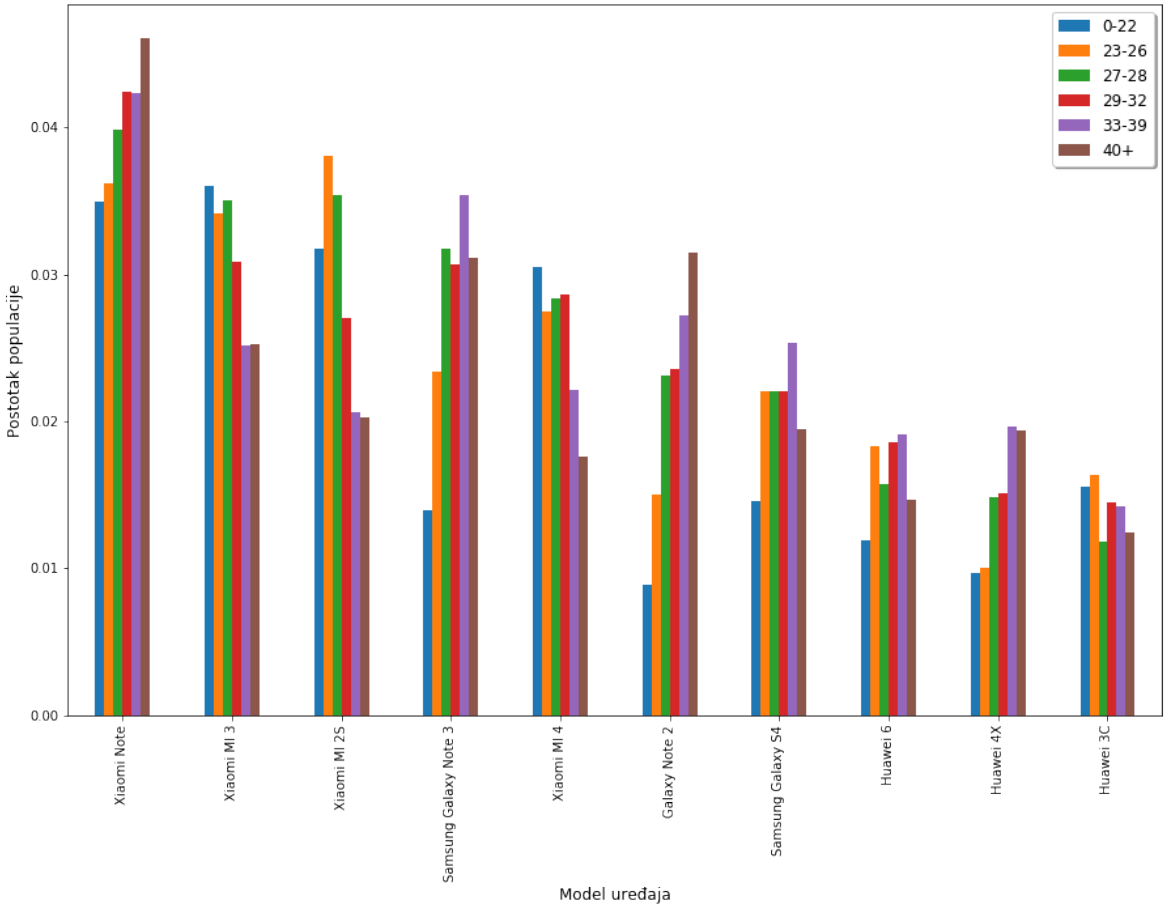
- 91.71% populacije `0-22`
- 91.51% populacije `23-26`
- 92.04% populacije `27-28`
- 91.22% populacije `29-32`
- 90.99% populacije `33-39`
- 91.10% populacije `40+`



Slika 4.12: Deset najčešće korištenih marki uređaja

Najčešće korišteni modeli uređaja

Za razliku od najčešće korištenih uređaja, koji obuhvaćaju više od 90% populacije korisnika i unatoč tome što iz grafa sa slike 4.13 se može primijetiti znatna razlika u korištenju pojedinih modela, zbog toga što suma svih deset najčešće korištenih modela uređaja ne prelazi 25% ukupne populacije utjecaj te značajke u predikciji neće bitan. Jedna od opcija je da koristimo sve modele uređaja te time obuhvatimo cijelu populaciju korisnika, no u ovom slučaju to nije izvedivo jer postoji 1599 različitih modela uređaja (što je prevelika brojka da bi ih sve uvrstili u značajke za predikciju).



Slika 4.13: Deset najčešće korištenih modela uređaja

5. Implementacija odabira optimalnog klasifikacijskog modela

5.1. Odabir značajki

Analizom podataka te uvidom u utjecaj pojedinih značajki na klasu predviđanja, za oba modela će biti korištene značajke opisane u tablici 5.1. Do njih se došlo manipulacijom izvornih podataka, te se izvorni kod njihove konstrukcije nalazi na digitalnom mediju priloženom uz ovaj rad.

Naziv značajke	**Opis značajke**	**Tip podatka**
`aktivne_aplikacije`	broj aktivnih aplikacija na korisnikovom uređaju	cjelobrojni
`instalirane_aplikacije`	broj instaliranih aplikacija na korisnikovom uređaju	cjelobrojni
`najaktivniji_sat`	sat u danu kada je korisnik najaktivniji	$[0-23]$
`top_10_uređaja`	Ovo nije značajka već predstavlja skup od 10 značajki. Govori nam koji od 10 najkorištenijih uređaja u populaciji koristi određeni korisnik.	1 ili 0
`kategorije_aplikacija`	Ovo također nije značajka već predstavlja skup svih značajki kategorija. Za svaku kategoriju dodana je značajka koja govori koliko aplikacija te kategorije korisnik ima instaliranih na svom uređaju.	cjelobrojni

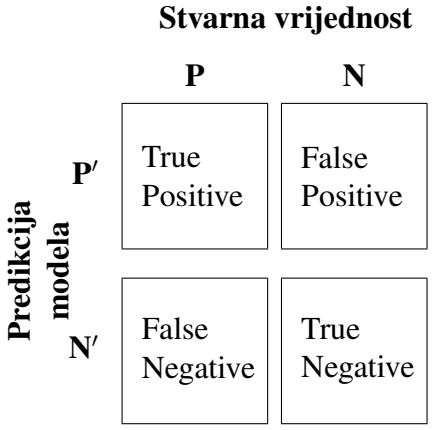
Tablica 5.1: Opis značajki korištenih za treniranje modela

5.2. Vrednovanje klasifikatora

Vrednovanje klasifikatora, odnosno evaluacija klasifikacijskog modela i odabir najboljega je vođen F_1 **evaluacijskom mjerom**. Izračun F_1 mjere počinje s izračunom matrice zabuna, te preko evaluacijskih mjera preciznosti i odziva dolazi se do F_1 evaluacijske mjere.

5.2.1. Matrica zabuna

Matrica zabuna (engl. *confusion matrix*) je matrica koja opisuje performanse klasifikacijskog modela. Jednostavno ju je razumjeti.



Tablica 5.2: Matrica zabuna

Značenje pojedinog elementa tablice 5.2:

- **true positives (TP)** - broj primjera koje je model klasificirao u promatranu klasu i oni stvarno pripadaju toj klasi
- **true negatives (TN)** - broj primjera koje je model klasificirao izvan promatrane klase i oni stvarno ne pripadaju toj klasi
- **false positives (FP)** - broj primjera koje je model klasificirao u promatranu klasu, ali oni stvarno ne pripadaju toj klasi
- **false negatives (FN)** - broj primjera koje je model klasificirao izvan promatrane klase, ali oni stvarno pripadaju toj klasi

5.2.2. Preciznost

Preciznost (engl. *precision*) je udio klasificiranih primjera u skupu pozitivno klasificiranih primjera.

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

5.2.3. Odziv

Odziv (engl. *recall*) je udio točno klasificiranih primjera u skupu svih pozitivnih primjera.

$$R = \frac{TP}{TP + FN}$$

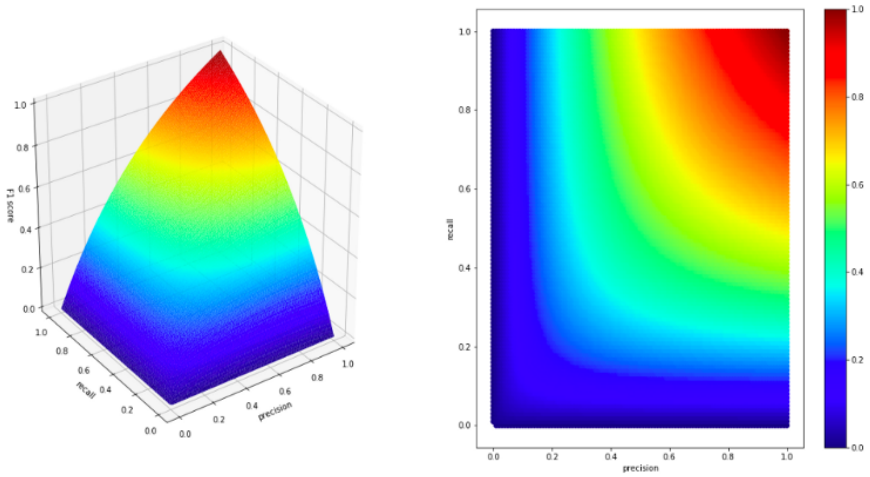
5.2.4. F-mjera

F-mjera je harmonijska sredina preciznosti i odziva. Računa se kao harmonijska sredina upravo zbog toga što je ona najstroža[1][9]. U općem slučaju, važnost preciznosti i odziva kontrolira se parametrom β:

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R}$$

Najčešće se koristi F_1 mjera kod koje su odziv i preciznost jednako bitne metrike[2]. Tako za F_1 mjeru formula izgleda ovako:

$$F_1 = \frac{2PR}{P+R}$$



Slika 5.1: F_1 evaluacijska mjera[13]

Na slici 5.1 je prikazan graf koji opisuje utjecaj preciznosti i odziva na vrijednost F_1 mjere.

5.2.5. Stratificirana unakrsna provjera

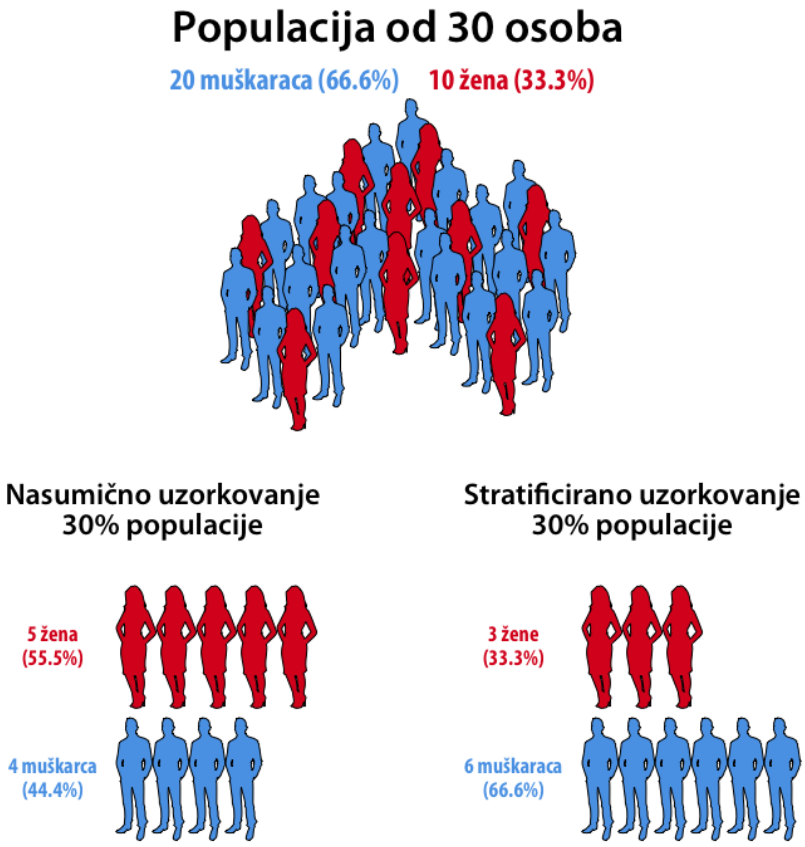
Za razliku od općenite unakrsne provjere koja izvorni skup podataka nasumičnim odabirom primjera dijeli na dva disjunktna skupa - *skup za treniranje modela* i *skup za testiranje modela*, stratificirana unakrsna provjera uzima u obzir udio klasa u izvornom skupu.

Na slici 5.2 je prikaz razlike uzorkovanja prilikom općenite unakrsne provjere i stratificirane unakrsne provjere.

Kod stratificirane unakrsne provjere, omjer izvornih klasa u dobivenom uzorku ostaje jednak. Ovo je jako bitno za skupove podataka kod kojih postoji disbalans u oznaci koju

[1] Stroža je od geometrijske i aritmetičke sredine

[2] $\beta < 1$ - naglašena evaluacijska mjera preciznosti; $\beta > 1$ - naglašena evaluacijska mjera odziva



Slika 5.2: Prikaz razlike općenitog (nasumičnog) uzorkovanja i stratificiranog uzorkovanja

algoritam predviđa. Bitno je da skup za treniranje i skup za testiranje budu reprezentativni primjerak populacije nad kojom se provodi algoritam strojnog učenja. Iz analize seta podataka u poglavlju 4.2 vidljiva je prisutnost disbalansiranosti klasa te je u programskom rješenju implementirana stratificirana unakrsna provjera.

5.2.6. Implementacija evaluacije

Evaluacija je implementirana u 3 koraka koja su objašnjena pojedinačno dalje u poglavlju.

1. korak: Transformacija značajki i podjela skupa podataka na dva dijela

Ovaj korak transformacije značajki zahtjeva *Spark* radi mogućnosti provođenja algoritma. Korišten je razred `VectorAssembler` koji sve značajke zapisuje kao vektor kojeg algoritam zna koristiti za treniranje. Također, za pretvorbu kategoričke klase koju model uči u numeričku vrijednost korišten je razred `StringIndexer`. Nakon transformacije izvorni skup se dijeli na skup za treniranje i skup za testiranje. S obzirom na to da knjižnica *MLlib* nema implementiranu stratificiranu podjelu, ovaj dio je implementiran u sklopu rada.

2. korak: Određivanje hiperparametara

Određivanje optimalnih hiperparametara se radilo pretragom po rešetci. To je metoda u kojoj se skup za treniranje dijeli na dva dijela (najčešće u omjeru 4:1) od kojih se veći dio koristi za treniranje modela s odabranim hiperparametrima, a manji za validaciju istog. Tako se, nakon prolaska kroz cijelu zadanu rešetku hiperparametara, na kraju izabire model s najboljom odabranom evaluacijskom mjerom na skupu za treniranje. Programski okvir *Apache Spark* to ima implementirano kroz klasu `TrainValidationSplit` kako je objašnjeno u sljedećem kodu:

```
1 val trainValidationSplit = new TrainValidationSplit()
2       .setEstimator(pipeline)
3       .setEvaluator(evaluator)
4       .setEstimatorParamMaps(paramGrid)
5       .setTrainRatio(0.8)
```

Kôd 5.1: Korištenje TrainValidationSplit klase

3. korak: Evaluacija F_1 mjerom

Nakon dobivenog modela s najboljom kombinacijom hiperparametara evaluiranima na skupu za validaciju, model se evaluira na do sad neviđenim primjerima - skupu za testiranje. Tim postupkom dobijemo za svaki algoritam najbolji model te njegovu F_1 metriku na skupu za testiranje. Od cijelog tog skupa modela odabire se onaj s najvećom metrikom.

5.3. Pokretanje aplikacije

Prije pokretanja same aplikacije za odabir najboljeg klasifikacijskog modela, potrebno je iz izvornog seta podataka konstruirati značajke opisane u potpoglavlju 5.1. Zbog nepotrebnog izvršavanja takvog procesa na računalnom grozdu, ovaj pomoćni program je ostvaren u programskom jeziku *Python* uz pomoć biblioteke *Pandas*. *Pandas* je biblioteka otvorenog koda koja pruža visoke performanse, jednostavne strukture podataka i alate za analizu podataka [3].

Implementacija konstrukcije opisanih značajki je dostupna na digitalnom mediju priloženom u sklopu ovog rada.

```
1 $ python konstuktorZnacajki.py datoteka/s/izvornim/podacima
```

Kôd 5.2: Pokretanje naredbe za konstrukciju značajki

Opisani program u direktorij s izvornim podacima zapisuje novu datoteku pod nazivom `konstruirane_znacajke.csv`. Nakon što je kreirana datoteka sa značajkama koja će biti korištena za treniranje modela, potrebno je vidjeti kako se pokreće aplikacija te koje parametre koristi za izbor najboljeg modela.

```
$ spark-submit --master yarn class_master_v3.1.jar
    --classifiers svm,rf,logreg,bayes
    --label gender
    --cat-percentage 0.10
    data/konstruirane_znacajke.csv
```

Kôd 5.3: Primjer pokretanja naredbe za pronalazak najboljeg modela

Naziv parametra	**Opis parametra**
`--classifiers`	Odabir klasifikacijskih algoritama, odvojeni zarezom, koji ulaze u izbor za najbolji model. Ovo je opcionalan parametar te ako nije postavljen uzimaju se sva četiri algoritma. Moguće je odabrati između `svm` (stroj potpornih vektora), `rf` (nasumična šuma), `logreg` (logistička regresija) i `bayes` (Naivni Bayes).
`--label`	Odabir značajke koju algoritam uči. Moguće je odabrati `gender` (spol korisnika) ili `group` (dobnu skupinu korisnika).
`--cat-percentage`	Postotak seta kategorija koji ulazi u značajke za učenje. Ovaj parametar je stavljen kao varijabilan jer se pokazalo da određeni modeli bolje rade algoritmi s manjim brojem kategorija.
`filename`	Datoteka koja sadrži ulazni set podataka. Izlazna datoteka pomoćnog programa.

Tablica 5.3: Opis parametara aplikacije za izbor najboljeg klasifikacijskog modela

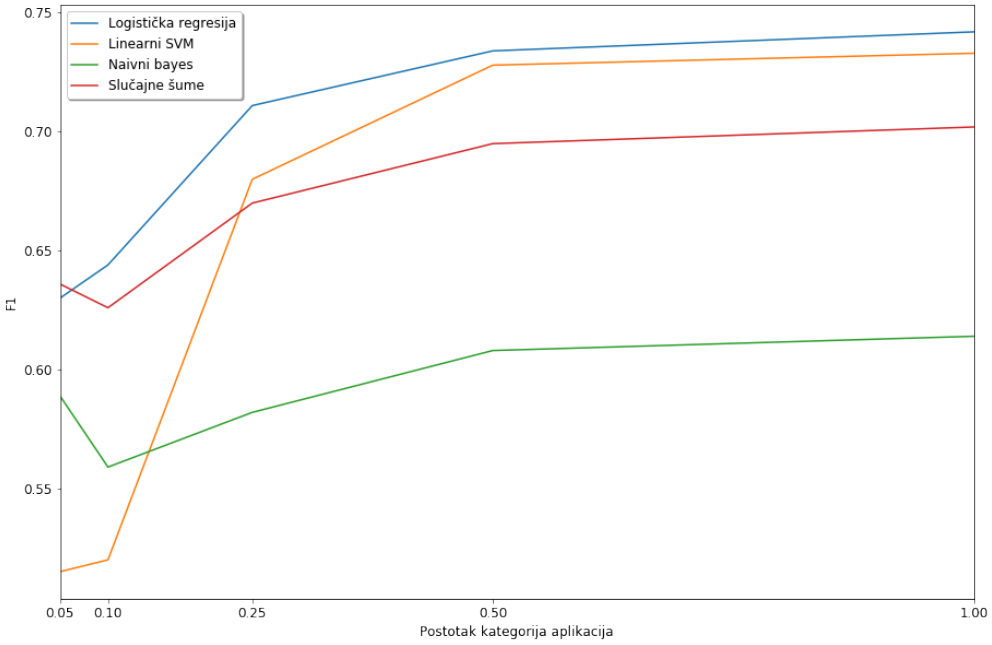
6. Eksperimentalni rezultati

Testiranje i evaluacija performansi je provedena na računalnom grozdu koji se sastoji od 17[1] računala sljedećih specifikacija:

- operacijski sustav *Linux Ubuntu*, verzija 16.04.6 LTS x-64
- procesor *Intel Core* i7-4790, 8 jezgri radnog takta 3.60 GHz
- 16 GB radne memorije

6.1. Rezultati klasifikacije spola korisnika

Iz grafa sa slike 6.1 se može zaključiti da je najbolji model za predikciju spola korisnika model *Logističke regresije*. Također, može se primijetiti da s prelaskom korištenja 50% najfrekventnijih kategorija na korištenje svih kategorija kao značajki modela, nije znatno poboljšalo F_1 mjeru.

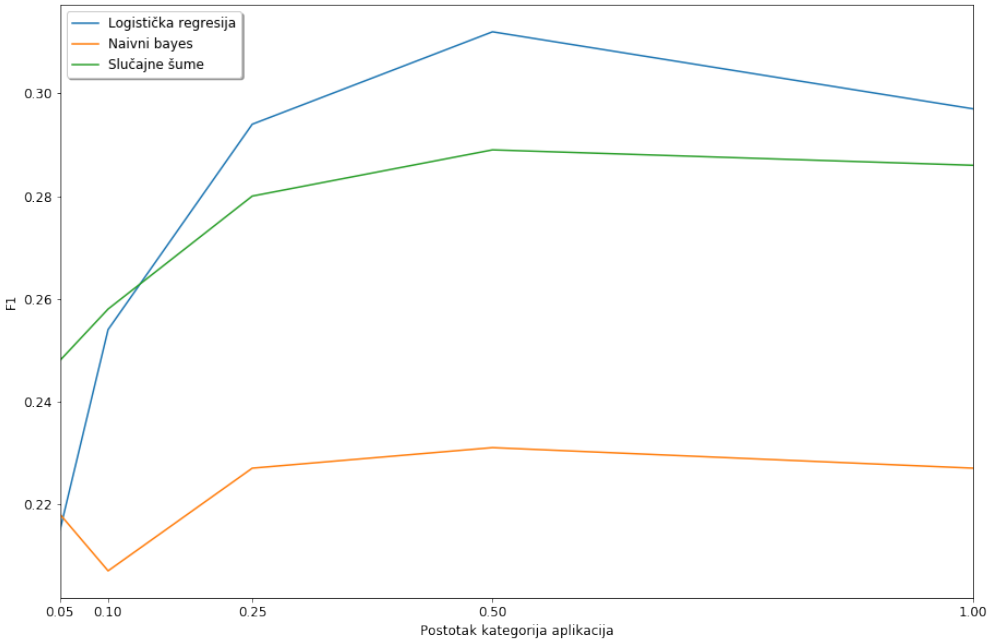


Slika 6.1: Rezultati F_1 mjere najboljih modela predikcije spola korisnika

[1] 16 radnih čvorova + 1 upravljački čvor

6.2. Rezultati klasifikacije dobne skupine korisnika

Graf na slici 6.2 prikazuje da je i u ovom slučaju predikcije dobne skupine korisnika najbolji model *Logističke regresije*. Za razliku od rezultata klasifikacije spola korisnika, u ovom slučaju se može primijetiti da korištenje cijelog seta kategorija instaliranih aplikacija pogoršava F_1 evaluacijsku mjeru svakog modela. Razlog zašto na ovom grafu nema *Linearnog SVM*-a je taj da programski okvir *Apache Spark* ima implementiranu samo binarnu klasifikaciju toga modela koja nije mogla biti korištena u ovom slučaju višeklasne klasifikacije korisnika u dobnu skupinu.



Slika 6.2: Rezultati F_1 mjere najboljih modela predikcije dobne skupine korisnika

Također, iz grafa se može primjetiti da je najbolji model imao F_1 mjeru jednaku 32. To nije najbolja vrijednost mjere te se može zaključiti da se, iz ovako konstruiranih značajki, teško može donijeti dobra predikcija dobne skupine korisnika.

7. Zaključak

Glavni cilj ovog rada bio je objasniti najpoznatije implementirane klasifikacijske algoritme strojnog učenja unutar programskog okvira *Apache Spark*. Također, u sklopu rada bilo je bitno izvesti i eksperimentalno evaluirati implementaciju istih na konkretnom primjeru. Potpun raspodijeljeni klasifikacijski sustav je ostvaren unutar programskog okvira *Apache Spark*, u programskom jeziku *Scala*. Napravljen je kratak i sažet uvod u podjelu algoritama strojnog učenja, a nešto detaljnije su opisani najpoznatiji klasifikacijski algoritmi. Testiranje i evaluacija performansi odabranih modela obavljena je na računalnom grozdu, korištenjem stvarnih podataka proučavanog studijskog slučaja.

U nedostatku implementacije svih potrebnih dijelova učenja klasifikacijskog modela, u radu je izrađena i stratificirana podjela seta podataka s obzirom na klasu koja se predviđa. Evaluacija i odabir najboljeg klasifikacijskog modela su pokazali da je algoritam *Logističke regresije* najpogodniji za korištenje za proučavani slučaj.

Moguća unaprjeđenja odnosila bi se na pronalazak boljeg skupa ulaznih parametara. Tijekom pisanja rada, ukazala se mogućnost klasifikacije vremenskih serija što na kraju nije uzeto u obzir zbog dodatne složenosti koja bi proširila proučavanu temu izvan njenog opsega. Pretpostavka je da bi takav prikaz omogućio precizniju klasifikaciju korisnika u promatrane kategorije. Potrebno bi bilo samo prikazati svakog korisnika kao vremensku seriju događaja koja opisuje njegovo ponašanje.

LITERATURA

[1] *Službena dokumentacija Apache Spark*. Dostupno na https://spark.apache.org/docs/2.3.0/, 11.04.2019.

[2] *Zadatak natjecanja objavnjen na Kaggle stranicama*. Dostupno na https://www.kaggle.com/c/talkingdata-mobile-user-demographics/data, 15.04.2019.

[3] *Pandas - Python Data Analysis Library*, 2019. Dostupno na http://pandas.pydata.org/, 13.06.2019.

[4] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[5] Renato Bošnjak. Usporedba metoda za klasifikaciju tekstualnih dokumenata. Završni rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2017.

[6] Bojana Dalbelo Bašić. *Stabla odluke*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2010. Dostupno na https://www.fer.unizg.hr/_download/repository/UI-10-StablaOdluke.pdf.

[7] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[8] Aaron Davidson i Andrew Or. *Optimizing shuffle performance in spark*, 2013.

[9] Bojana Dalbelo Bašić i Jan Šnajder. *Vrednovanje klasifikatora*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2011. Dostupno na https://www.fer.unizg.hr/_download/repository/SU-12-VrednovanjeKlasifikatora.pdf.

[10] Bojana Dalbelo Bašić i Jan Šnajder. *Strojno učenje*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2014.

[11] Mark Gover i Ted Malaska. *Top 5 mistakes when writing Spark Applications*, 2016. Dostupno na https://youtu.be/WyfHUNnMutg, 18.04.2019.

[12] Ian Goodfellow i Yoshua Bengio i Aaron Courville. *Deep learning*. MIT press, 2016.

[13] Bartosz Mikulski. *F1 score explained*, 2019. Dostupno na `https://www.mikulskibartosz.name/f1-score-explained/`, 10.06.2019.

[14] Nick Pentreath. *Machine learning with spark*. Packt Publishing Ltd, 2015.

[15] Filip Popić. Raspodijeljeni sustav za preporučivanje na platformi apache spark. Diplomski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2017.

[16] Jinde Shubham. *Ensemble Learning — Bagging and Boosting*, 2018. Dostupno na `https://becominghuman.ai/ensemble-learning-bagging-and-boosting-d20f38be9b1e`, 5.06.2019.

[17] Statistics4u. *Linear vs. Nonlinear Models*. Dostupno na `http://www.statistics4u.com/fundstat_eng/cc_linvsnonlin.html`, 19.05.2019.

Klasifikacija podataka korištenjem radnog okvira Apache Spark

Sažetak

U ovom radu dan je pregledan prikaz najpoznatijih implementiranih klasifikacijskih algoritama koje obuhvaća programska knjižica `MLlib` programskog okvira *Apache Spark*. Zbog *Spark*-ovog nedostatka vizualizacije obrade i analize značajki podataka, korištena je programska knjižica *Pandas* u programskom jeziku *Python*. Pri konstrukciji skupa značajki koje će biti korištene za treniranje modela, izrađen je i sam konstrukcijski proces također u programskom jeziku *Python*. Treniranje modela, evaluacija te optimalan odabir klasifikacijskog modela ostvareni su u programskom jeziku *Scala*, uz korištenje programskog okvira *Apache Spark* koji omogućava raspodijeljeno izvođenje. Evaluacija i odabir optimalnog klasifikacijskog modela su provedeni na fakultetskom računalnom grozdu na stvarnim podacima proučavanog slučaja.

Ključne riječi: klasifikacijski algoritmi, raspodijeljeni sustav, Apache Spark, MLlib, strojno učenje, logistička regresija, SVM, slučajne šume, naivni Bayesov klasifikator

Data Classification with Apache Spark Framework

Abstract

This paper gives an overview of the most popular implementations of classification algorithms provided in *Apahce Spark*'s *MLlib* library. Due to *Spark*'s lack of capability to visualize processed and analyzed data, the *Pandas* library was used in *Python* for that purpose. For constructing a set of features which will be used for fitting a model, the construction process itself was also developed and implemented in *Python* programming language. Model fitting, evaluation and optimal selection of the classification model are realized in the *Scala* programming language using *Apache Spark Framework* which allows distributed performance. The evaluation and the selection of the optimal classification model were executed on a faculty cluster on the real life dataset.

Keywords: classification algorithms, distributed system, Apache Spark, MLlib, machine learning, logistic regression, SVM, random forrest, naive Bayes