Univerzitet u Beogradu
Matematički fakultet

Aleksandar Nedeljković
1102/2013

Implementacija i evaluacija algoritama mašinskog učenja za filtriranje neželjene elektronske pošte

Master rad

Beograd, 2015.

Mentor: **dr Filip Marić**
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije: **dr Mladen Nikolić**
Univerzitet u Beogradu, Matematički fakultet

dr Jelena Graovac
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Rezime: Slanje neželjene elektronske pošte (eng. *spam e-mail*) značajno je intenzivirano tokom poslednjih 25 godina. Mane u protokolima za slanje i ubrzan razvoj elektronskog biznisa i finansijskih transakcija direktno utiču na uvećanje pretnji putem elektronske pošte. Neželjena elektronska pošta danas predstavlja jedan od glavnih problema na internetu donoseći gubitke finansija i vremena kompanijama i individualnim korisnicima.

Većina postojećih istraživanja na polju suzbijanja neželjene e-pošte fokusira se na dizajn protokola, metode autentifikacije i tehnike filtriranja. Među pristupima koji su razvijeni najvažnija je tehnika filtriranja. U poslednje vreme mašinsko učenje za klasifikaciju neželjene pošte je veoma važno istraživačko pitanje. Da bi se napravio dobar filter neželjene elektronske pošte, analiziraju se reči koje se pojavljuju u tekstu poruke, njihova učestanost, elektronska adresa pošaljioca, kao i mnogi drugi podaci. U procesu učenja učestvuju dobronamerni korisnici koji mogu klasifikovati određene poruke kao spam i ukoliko se te iste poruke pošalju na druge adrese u okviru istog mejl servera filter neželjene elektronske pošte će ih sam klasifikovati kao spam. Efikasan sistem filtriranja e-pošte zahteva preciznost (*nizak stepen lažno pozitivnih i lažno negativnih instanci*), mogućnost samorazvijanja (*sistemi koji imaju mogućnost da se adaptiraju na nove neželjene poruke*) i visoke performanse (*detekcija novih neželjenih poruka mora se detektovati brzo*).

Cilj rada je implementacija i evaluacija nekih algoritama mašinskog učenja (Bajesovo filtriranje, metoda neuronskih mreža, metoda podržavajućih vektora i slično) za klasifikaciju i filtriranje neželjene pošte i njihova evaluacija na nekoliko korpusa elektronske pošte.

Ključne reči: klasifikacija tekstualnih dokumenata, filtriranje neželjene elektronske pošte, algoritmi mašinskog učenja, Naivni Bajesov algoritam, veštačke neuronske mreže (ANN), metod podržavajućih vektora (SVM), k najbližih suseda (kNN)

Sadržaj

Spiskovi

Algoritmi

Tabele

Slike

1 Uvod

Elektronska pošta (*e-mail*) je mrežni servis koji omogućava slanje i primanje poruka raznovrsnog sadržaja. Ime predstavlja analogiju tradicionalnoj pošti, pri čemu poštansko sanduče zamenjuju serveri na kojima se pošta čuva dok je korisnik ne preuzme. Programi za rad sa elektronskom poštom se zasnivaju na uređivaču teksta za sastavljanje poruka. U oktobru 1971. (mada ima izvora koji tvrde da se to dogodilo jula 1970.) stručnjak za računarstvo Rej Tomlinson je napisao prvi program za razmenu poruka između dva računara. Pored programa za razmenu e-pošte, Rej Tomlinson je odgovoran za masovnu upotrebu znaka `@` (et, poznato i kao *ludo a* ili *majmunsko a*). Smišljajući kako da razvrsta primaoce poruka odlučio je da njihova imena i imena računara na kojima se nalaze njihovi nalozi razdvoji nekim znakom interpunkcije. Kako je na svojoj tastaturi imao samo 12 takvih znakova na raspolaganju, odlučio se za onaj koji se nikada ne koristi u pisanju poruka [30]. Taj princip adresiranja koristi se i danas. Svaka poruka elektronske pošte se sastoji iz tela i zaglavlja. Zaglavlje nosi sledeće podatke:

- adresu i ime pošaljioca
- adresu primaoca
- adresu servera preko kojeg je poruka poslata kao i servera koji su prosleđivali poruku na njenom putu do odredišta
- datum slanja
- ime programa koji je korišćen za slanje poruke
- prioritet

Svaka stavka zaglavlja se sastoji od imena stavke i vrednosti stavke, razdvojenih dvotačkom. Sledi primer jednog dela zaglavlja poruke:

```
Date: Fri, 24 Oct 2015 11:51:07 -0400
To: marko.markovic@example.com
From: Nemanja Petrovic <n.petrovic@example.com>
Reply-to: n.petrovic2@example.com
Subject: Naslov poruke
X-Priority: 3
```

Slika 1: Primer jednog dela zaglavlja poruke

Telo poruke se može sastojati iz više delova u zavisnosti od toga da li se sa tekstom poruke šalju i datoteke. Ukoliko se šalju onda se u samom zaglavlju poruke to može označiti pomoću stavke sledećeg oblika:

```
Content-Type: multipart/alternative;
    boundary="b1_8d5c3f4ac4f174c9d0bbc13814d16891"
```

Slika 2: Primer jednog dela tela poruke

Korišćenje elektronske pošte je ugroženo od strane četiri pojave: bombardovanja porukama, spamom, pokušajima preuzimanja ličnih podataka i prenošenjem virusa. Na internetu termin *spam* označava nepoželjnu, besciljnu (*untargeted*) elektronsku poštu. U osnovi, to je slanje komercijalnih poruka, najčešće reklama i mrežnih marketing šema[1], na stotine hiljada pa i miliona adresa korisnika širom mreže, bez njihovog odobrenja. Blaži oblik spama predstavlja TDEM (*Targeted Direct Email Marketing*) ili direktni marketing putem ciljnih grupa. Bez obzira u koju vas grupu budu stavili nemilosrdni spameri, jedno je sigurno − vaše e-mail sanduče svakoga dana biće bombardovano desetinama beskorisnih poruka.

1.1 E-mail komunikacija

Elektronska pošta je jedan od najpopularnijih vidova komunikacije danas. Iznenađujuće brzo prihvatanje ovog vida komunikacije se najbolje ilustruje brojem trenutnih korisnika. Broj naloga elektronske pošte u svetu je 2011. godine iznosio 3.1 milijardu i očekuje se rast do 4.1 milijarde do kraja 2015. godine što predstavlja prosečnu brzinu rasta od približno 7% svake godine. Geografski gledano u 2011. godini najveći broj korisnika elektronske pošte nalazio se u Aziji, ukupno 49% korisnika elektronske pošte u svetu. Evropski nalozi elektronske pošte 2011. činili su 22% ukupnog broja naloga na internetu dok je Severna Amerika činila oko 14%. Ostatak sveta čini preostalih 15% naloga elektronske pošte na interentu [24].

U 2011. godini 25% naloga elektronske pošte na interetu činili su korporativni e-mail nalozi. U naredne četiri godine očekuje se brži tempo rasta korporativnih e-mail naloga zbog sve pristupačnijih e-mail servisa koji su bazirani na računarstvu u oblaku. Mnoge korporacije koriste e-mail servise računarstva u oblaku kao način za proširivanje usluga elektronske pošte radnicima koji možda nisu imali pristup elektronskoj pošti u prošlosti.

	2011.	**2012.**	**2013.**	**2014.**	**2015.**
Broj e-mail naloga na internetu u milionima	3146	3375	3606	3843	4087
Broj korpotarivnih e-mail naloga u milionima	788	850	918	991	1070
Broj korpotarivnih e-mail naloga u %	25%	25%	25%	26%	26%
Broj korisničkih e-mail naloga u milionima	2358	2525	2688	2852	3017
Broj korisničkih e-mail naloga u %	75%	75%	75%	74%	74%

Tabela 1: Korporativni naspram korisničkih e-mail naloga u periodu od 2011. do 2015.

[1]Mrežni marketing (engl. *Multi-Level-Marketing*) označava sistem koji koristi preporuke osobe od poverenja za svrhu poslovanja pri prodaji roba ili usluga. Mrežu organizuju sami zadovoljni potrošači, usmenom preporukom, tj. reklamom od usta do usta, što čini suštinu mrežnog marketinga.

Prosečni korporativni e-mail korisnik primi oko 105 e-mail poruka dnevno. Uprkos spam filterima, otprilike 19% elektronske pošte koja je dostavljena u prijemno sanduče je neželjena pošta. Brzina rasta poslatih i primljenih e-mail poruka postepeno usporava zbog ubrzanog rasta drugih vidova komunikacije poput instant poruka i socijalnih mreža. Prosečni godišnji rast instant poruka iznosi oko 11% i očekuje se da broj naloga u 2015. godini premaši cifru od 3.8 milijardi korisnika. Socijalne mreže takođe beleže ubrzan rast kako kod potrošačkih tako i kod korporativnih korisnika. U 2011. ukupan broj naloga na socijalnim mrežama iznosio je oko 2.4 milijarde dok se u 2015. očekuje da taj broj naraste do 3.9 milijardi [24].

	2011.	**2012.**	**2013.**	**2014.**	**2015.**
Prosečan broj primljenih i poslatih e-mailova jednog korisnika po danu	105	110	115	120	125
Prosečan broj primljenih e-mailova	72	75	78	81	84
Prosečan broj legitimnih e-mailova	58	62	65	68	71
Prosečan broj neželjene pošte	*14*	*13*	*13*	*13*	*13*
Prosečan broj poslatih e-mailova	33	35	37	39	41

Tabela 2: Korporativna primljena i poslata e-pošta jednog korisnika po danu u periodu od 2011. do 2015.

1.2 Problem neželjene elektronske pošte

U poslednje vreme neželjena pošta postala je veliki problem na internetu i predstavlja gubljenje vremena, prostora za skladištenje i protoka informacija. Problem spama se godinama uvećava. Po skorašnjim statistikama oko 15 milijardi mejlova dnevno su neženjena pošta i koštaju internet korisnike oko 355 miliona dolara godišnje. Spameri su počeli da koriste nekoliko trikova kako bi zaobišli metode filtriranja korišćenjem različitih adresa slanja i različitih karaktera kojim započinju i završavaju naslov poruke [14]. Tradicionalni sistemi za otkrivanje neželjene pošte koji koriste sisteme bazirane na rečima su lako poraženi jer su spameri pronašli nove načine za reprezentovanje reči. Na primer reč hipoteka se može zapisati kao *h-i-p-o-t-e-k-a* ili kao *h i p o t e k a* i biće neregistrovana od strane ovakvih sistema. Zbog toga bi se svaka detektovana modifikovana reč morala dodavati u bazu. Fudamentalna potreba bilo kog filtera neželjene pošte je da nikada ne označi dobru poruku kao neželjenu čak i ako to znači neotkrivanje nekolio neželjenih poruka.

Pristup mašinskog učenja je mnogo efikasniji od prethodnog pristupa jer nema potrebu za navođenjem nikakvih pravila [18]. Umesto toga koristi se skup trening podataka koji su prethodno već klasifikovani. Specifičan algoritam se koristi za učenje pravila klasifikacije iz trening skupa podataka. Pristup mašinskog učenja je veoma proučen i veliki broj algoritama se može koristiti za filtriranje elektronske pošte. Oni uključuju Naivni Bajesov algoritam, K najbližih suseda, SVM (*metod podržavajućih vektora*) itd.

Azija je odgovorna za 55,5 % spamova u svetu u 2013. godini, prati je Severna Amerika sa 19 %, dok je na trećem mestu Istočna Evropa sa 13,3 %. Srbija se nalazi na 27. mestu globalne liste zemalja koje su izvori spama sa 0,47%, dok su vodeće zemlje Kina sa 23% i Sjedinjene Američke Države sa 18%, pokazuje istraživanje kompanije Kaspersky lab. Udeo Istočne Evrope se skoro udvostručio u odnosu na prethodnu godinu, stavljajući ovu oblast na treće mesto sa 13,3%, Zapadna Evropa ostaje na četvrtom mestu uprkos smanjenju od 2,4%, dok je udeo Latinske Amerike na petom mestu sa trostrukim padom u poređenju sa 2012. godinom.

Internet je prepun ljudi koji žele da nas prevare, a jedan od najpopularnijih načina je takozvana *Nigerijska prevara*[2]. U pitanju je e-mail u kojem piše da ste navodno dobili ogroman novac, iako ni sami ne znate osobu koja vam je to saopštila. Prvi talas ovih prevara potekao je iz Nigerije, pa je po tome i dobila ime. Prevara počinje tako što vas nepoznate osobe kontaktiraju putem e-maila nudeći ogromnu sumu novca koji ste navodno dobili ili nasledili. Kako biste dobili nasledstvo ili novac koji žele da vam pošalju, od vas će tražiti da platite razne takse ili da im date informacije o vašim računima. U početku je to mala količina novca, kako bi sve delovalo uverljivo, ali ukoliko nasednete na ovo, sigurno će izmisliti još nešto što mora da se plati kako bi dobili svoju nagradu.

Proizvodi	*25 %*
Finansijski	*20 %*
Za odrasle	*19 %*
Prevara	*9 %*
Zdravlje	*7 %*
Internet	*7%*
Opuštanje	*6%*
Duhovni	*4%*
Ostali	*3 %*

Tabela 3: Raspodela spam poruka

Jedan od novijih vidova spama jeste spam putem internet pretraživača. Sajtovi koji pretražuju internet, kao što su Google, Yahoo, Microsoft-ov Bing, se danas susreću sa novim poteškoćama. Kako bi neki pretraživač mogao da zna koji je sajt popularniji, kvalitetniji, koristi se poznati *PageRank* algoritam za rangiranje i njegove modifikacije. Sam algoritam se oslanja na broj linkova koji pokazuje na određenu internet stranicu, smatrajući stranicu na koju pokazuje veći broj linkova popularnijom, što je u prvim godinama interneta zaista i važilo. Poznavajući tu metodologiju, spameri su se dosetili i

[2]Više o *Nigerijskoj prevari* možete pročitati na sajtuMUP-a.

rešili da naprave gomile stranica koje ukazuju na neku stranicu, ne bi li je podigli na što bolje plasirano mesto kada korisnici vrše pretragu po internetu. Navedena vrsta spama se naziva *spamdexing*, što je reč nastala spajanjem dve reči (eng. *spam* i *indexing*). Sama činjenica da se to masovno radi nam dokazuje da je spam jako ozbiljna stvar i da zaista možemo tvrditi da se spam pretvara u posao. Prema nekim okvirnim procenama polovina internet sadržaja su pornografske stranice, dok je polovina preostalih stranica spam. Zaista, spam sve više ugrožava normalan razvoj interneta i pravi probleme.

Saveti za zaštitu od neželjenih poruka:

1. Ne otvarajte neželjene poruke, izbrišite ih bez prethodnog otvaranja. Otvaranjem poruke obaveštavate pošiljaoca da vaša e-mail adresa postoji.
2. Nikada ne odgovarajte na spam poruke bez obzira šta se od vas traži jer u suprotnom na ovaj način obaveštavate pošiljaoca o validnosti vaše e-mail adrese.
3. Ne kupujte proizvode koji se reklamiraju putem spam poruka bez obzira koliko povoljni bili. Ukoliko kupite proizvod očekujte još više reklamnih poruka u vašem poštanskom sandučetu.
4. Ne prosleđujte spam poruke, dovoljno je što ste dobili spam, ne morate i sami da postanete spamer. Ukoliko nastavite sa prosleđivanjem vaša adresa će stići i kod ostalih spamera, zaključite sami, dobijaćete još više neželjenih poruka. Nemojte prosleđivati spam poruke, jer postoji opasnost da je poruka sadržala virus, samim tim i vi postajete pošiljalac virusa.

Prema podacima FBI, u 2013. godini je ukupni gubitak od svih internet prevara bio skoro 782 miliona dolara [9]. Broj prijava koje je ova agencija prikupila u 2013. godini iznosio je 262.813, a najviše su bili pogođeni ljudi srednjih godina. Ovo i nije tako neuobičajeno, jer generalno gledano, ta generacija je odrastala bez interneta, pa većina ne zna kakve prevare sve postoje.

2 Klasifikacija

Zadatak klasifikacije je da objektima dodeli jednu od nekoliko predefinisanih kategorija. Ulazni podatak u proces klasifikacije je skup slogova koji se nazivaju instance. Slogovi su oblika (x, y) gde je x vektor vrednosti nekih atributa a, y atribut koji određuje oznaku klase. Ulazni podaci se dele na podatke za *učenje* i *testiranje*. Na osnovu podataka za učenje pronalazi se klasifikator koji svaki vektor vrednosti nekih atributa x preslikava u neku predefinisanu kategoriju y. Cilj je da klasifikacija bude što preciznija, a određivanje tačnosti modela se postiže na osnovu podataka za testiranje korišćenjem neke od mera evaluacije.

Klasifikacija nekog objekta se zasniva na pronalaženju sličnosti sa unapred određenim objektima koji su pripadnici različitih klasa, pri čemu se sličnost dva objekta određuje analizom njihovih karakteristika. Pri klasifikaciji se svaki objekat svrstava u neku od klasa sa određenom verovatnoćom. Zadatak je da se na osnovu karakteristika objekata čija klasifikacija je unapred poznata, napravi model na osnovu koga će se vršiti klasifikacija novih objekata. U problemu klasifikacija, broj klasa je unapred poznat i ograničen.

Definicija: Klasifikacija je proces učenja **ciljne funkcije** f koja predefinisane klase y pridružuje vektoru vrednosti atributa x [27].

Ciljna funkcija je takođe poznata i kao **klasifikacioni model**. Klasifikacioni model je koristan za sledeće namene.

Deskriptino modelovanje. Predstavlja glavne karakteristike podataka. U suštini rezimira podatke i omogućava nam da proučavamo najvažnije aspekte podataka bez uzimanja u obzir veličine skupa podataka.

Prediktivno modelovanje. Ima specifičan cilj da nam omogući da predvidimo vrednosti nekih ciljanih karakterisitka objekta na bazi posmatranja vrednosti drugih karakteristika objekta.

2.1 Klasifikacija tekstualnih dokumenata

Automatska klasifikacija tekstualnih dokumenata je važna istraživačka tema još od prihvatanja digitalizacije dokumenata. Intuitivno, klasifikacija teksta je zadatak klasifikovanja dokumenta u jednu ili više predefinisanih kategorija. Formalnije, ako je d_i dokument iz skupa dokumenata D i $\{c_1, c_2, ..., c_n\}$ je skup svih kategorija, tada je klasifikacija teksta dodeljivanje jedne ili više kategorija c_j dokumentu d_i.

Definicija: Kategorizacija teksta je zadatak dodeljivanja *Boolean* vrednosti svakom paru $\langle d_i, c_j \rangle \in \mathcal{D} \times \mathcal{C}$, gde je $\mathcal{D}$ domen dokumenata i $\mathcal{C} = \{c_1, ..., c_{|\mathcal{C}|}\}$ je skup predefinisanih kategorija [25].

Vrednost T (*True*) dodeljena paru $\langle d_i, c_j \rangle$ ukazuje na odluku da je dokument d_i pod kategorijom c_j, dok vrednost F (*False*) ukazije na odluku da dokument d_i nije pod kategorijom c_j. Zadatak je aproksimirati ciljnu funkciju $f : \mathcal{D} \times \mathcal{C} \rightarrow \{T, F\}$.

Različite zavisnosti se mogu javiti pri klasifikaciji teksta u zavisnosti od primene. Na primer za dato celobojno k, tačno k elemenata iz $\mathcal{C}$ treba dodeliti svakom $d_i \in \mathcal{D}$. Slučaj u kome tačno jedna kategorija mora biti dodeljena svakom dokumentu $d_i \in \mathcal{D}$ se često naziva jednoznačna (eng. *single-label*) klasifikacija, dok se slučaj u kome se svakom dokumentu $d_i \in \mathcal{D}$ dodeljuje više od jedne kategorije naziva višeznačna (eng. *multi-label*) klasifikacija. Specijalan slučaj jednoznačne klasifikacije je *binarna* klasifikacija u kojoj se svakom dokumentu $d_i \in \mathcal{D}$ dodeljuje i kategorija c_j ili njen komplement $\overline{c_j}$.

Iz teoretske tačke gledišta binarni slučaj (jednoznačna klasifikacija takođe) je opštiji od višeznačne klasifikacije jer se algoritam binarne klasifikacije može upotrebiti i za višeznačnu klasifikaciju. Nepohodno je samo transformisati problem višeznačne klasifikacije tako da se kategorije $\{c_1, c_2, ..., c_n\}$ podele u $|\mathcal{C}|$ nezavisnih problema binarne klasifikacije kao $\{c_j, \overline{c_j}\}$, za $i = 1, ..., |\mathcal{C}|$ [25]. Međutim potrebno je da su kategorije stohastički nezavisne jedna od druge.

2.2 | Mere kvaliteta i tehnike evaluacije klasifikatora

Ocena kvaliteta klasifikatora teksta se sprovodi eksperimentalno, a ne analitički, jer analitička procena zahteva poznavanje formalne specifikacije problema koji sistem pokušava da reši. Da bi što preciznije mogli da uporedimo različite algoritme klasifikacije, najbitnije je koristiti iste skupove podataka sa istom podelom dokumenata za učenje i testiranje. Ukoliko se klasifikator dobro ponaša na trening skupu podataka, a loše na test skupu podataka u pitanju je *problem preprilagođenog modela* (eng. overfitting problem). Taj problem nastaje jer se formira model koji savršeno opisuje trening podatke, ukljucujuci i specifičnosti nebitne za cilj učenja.

Postupak evaluacije klasifikatora predstavlja poređenje unapred poznate klase sa onom koju je predložio klasifikator. Na taj način se dobijaju ispravno i neispravno klasifikovani podaci. Na osnovu tih informacija formira se **matrica konfuzije**.

Tabela 4 prikazuje matricu konfuzije za problem binarne klasifikacije. Svaki unos f_{ij} u ovoj tabeli označava broj zapisa iz klase i koji su predviđeni u klasi j. Na primer, f_{01}

je broj zapisa iz klase 0 pogrešno predviđenih kao klasa 1. Na osnovu unosa iz matrice konfuzije ukupan broj predviđanja koje je napravio model je $(f_{11} + f_{00})$ i ukupan broj neispravnih predviđanja je $(f_{10} + f_{01})$.

		Predviđena klasa		
		Klasa = 1	*Klasa = 0*	*Ukupno*
Stvarna klasa	*Klasa = 1*	f_{11}	f_{10}	$f_{11} + f_{10}$
	Klasa = 0	f_{01}	f_{00}	$f_{01} + f_{00}$
	Ukupno	$f_{11} + f_{01}$	$f_{10} + f_{00}$	N

Tabela 4: Matrica konfuzije za problem 2 klase *(binarne klasifikacije)*

Vrednosti matrice konfuzije obrazložene na primeru klasifikacije elektronske pošte:

1. f_{11} − vrednost ,,stvarno pozitivni" (eng. true positive) predstavlja broj dokumenata koji su zaista spamovi a koje je klasifikator prepoznao kao spamove
2. f_{01} − vrednost ,,lažno pozitivni" (eng. false positive) predstavlja broj dokumenata koji nisu spamovi a koje je klasifikator klasifikovao u grupu spamova
3. f_{00} − vrednost ,,stvarno negativni" (eng. true negative) predstavlja broj dokumenata koji nisu spamovi a klasifikator ih je klasifikovao kao legitimne poruke
4. f_{10} − vrednost ,,lažno negativni" (eng. false negative) predstavlja broj dokumenata koji su spamovi a koji su svrstani u grupu legitimnih poruka

Iako matrica konfuzije pruža informacije neophodne za određivanje koliko dobro klasifikacioni model funkcioniše, svođenje ovih informacija na jedan konkretan broj omogućava mnogo jednostavnije poređenje performansi različitih modela.

Tačnost (eng. accuracy) klasifikatora je mera koja nam daje procenat uspešno klasifikovanih dokumenata. Tačnost definišemo na sledeći način:

$$\textit{Tačnost} = \frac{\textit{Broj ispravnih predviđanja}}{\textit{Ukupan broj predviđanja}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

Za mnoge primere klasifikacije tačnost je korisna mera, međutim postoje slučajevi kada nam tačnost ne može dati informacije koje tražimo. To su uglavnom scenariji u kojima je jedna klasa značajno manja od druge. Tada je moguće dobiti visoku tačnost svrstavnjem svih instanci u veću grupu.

Ekvivalentno performanse modela se mogu izraziti u obliku **stope greške**, koja je data na sledeći način:

$$\textit{Stopa greške} = \frac{\textit{Broj pogrešnih predviđanja}}{\textit{Ukupan broj predviđanja}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}} = \frac{FN + FP}{TP + FN + FP + TN}$$

Standardne mere kvaliteta klasifikatora su *preciznost* i *odziv*. To su numeričke vrednosti koje predstavljaju mere očekivanja korisnika za klasifikaciju da će klasifikator ispravno klasifikovati slučajno odabran dokument.

Preciznost (eng. precission) je mera koja nam daje informaciju o udelu stvarno pozitivnih instanci. Od svih poruka koje su označene kao spam, koji procenat čine poruke koje su stvarno spam?

$$\textit{Preciznost} = \frac{f_{11}}{f_{11} + f_{01}} = \frac{TP}{TP + FP}$$

Odziv (eng. recall) je mera suprotna preciznosti. Od svih poruka koje su stvarno spam, koji procenat poruka je klasifikovan kao spam?

$$\textit{Odziv} = \frac{f_{11}}{f_{11} + f_{10}} = \frac{TP}{TP + FN}$$

Kombinovanjem preciznosti i odziva dobijena je f_1 mera koja se definiše na sledeći način:

$$f_1 = \frac{2 \cdot \textit{Preciznost} \cdot \textit{Odziv}}{\textit{Preciznost} + \textit{Odziv}}$$

2.2.1 Unakrsna validacija

Tehnika za evaluaciju klasifikatora koja se često koristi je *unakrsna validacija* (eng. cross validation). Neophodno je da se skup podataka podeli na k delova približno iste veličine, a zatim se svaki od $k-1$ delova koristi kao skup za učenje a sam taj preostali deo kao skup za testiranje. Postupak se ponavlja k puta tako da je svaki od delova po jednom učestvovao u ulozi testnog skupa podataka. Greška klasifikacionog modela je prosečna greška svih k iteracija u postupku.

Čest je slučaj da se postupak slučajnog izbora modifikuje tako da se osigura približno jednaka zastupljenost klasa u svakom od k delova. Taj postupak se naziva *stratifikacija* i njime se obezbeđuje da je pri svakoj iteraciji zastupljenost klasa u skupu za učenje i testiranje približno jednaka zastupljenosti u inicijalnom skupu primera.

Složenost evaluacije klasifikacionog modela ovom metodom zavisi od k. U praksi se najčešće uzima $k = 5$ ili $k = 10$ jer se takva unakrsna validacija pokazala kao dovoljno tačna a nije previše zahtevna.

3 Algoritmi filtriranja elektronske pošte

U ovom poglavlju predstaviću pregled osnovne teorije i ideje algoritama mašinskog učenja koji se koriste pri filtriranju elektronske pošte.

3.1 Bajesovo filtriranje

Najpopularnija tehnika filtriranja elektronske pošte je Bajesovo filtriranje, koje je jednostavan metod koji se zasniva na verovatnoći. U kontekstu elektronske pošte to jednostavno znači da se verovatnoća da je pošta neželjena može izračunati na osnovu određenih reči ukoliko znamo verovatnoću neželjenih mejlova i koliko često se te reči pojavljuju u njima, podeljene sa verovatnoćom koliko često se reči pojavljuju u telima sve elektronske pošte [23]. Određene reči imaju određene verovatnoće pojavljivanja u neželjenoj i legitimnoj elektronskoj pošti. Većina korisnika kada vidi reč *Viagra* zna da se radi o neželjenoj pošti i retko će je videti u pošti koja nije neželjena. Filter ne zna ove verovatnoće unapred i neophodan mu je trening da bi mogao da odredi verovatnoće pojavljivanja određenih reči u neželjenoj pošti. Za treniranje filtera neophodno je da korisnici elektronske pošte ručno obeležavaju da li je pošta neželjena ili ne. Za sve reči svakog mejla pri testiranju filter će prilagoditi verovatnoće u svojoj bazi podataka. Na primer Bajesov filter neželjene pošte će sa veoma velikom verovatnoćom naučiti da se radi o neželjenoj pošti ukoliko se pojavljuje reč *Viagra* ali veoma malu verovatnoću da je reč o neželjenoj pošti za reči koje su se pojavile u legitimnoj pošti. Nakon treniranja poznate su verovatnoće reči koje se nalaze u bazi i moguće je izračunati verovatnoću da li je pošta neželjena ili ne. Ukoliko je izračunata verovatnoća svih reči u mejlu veća od nekog praga (npr. 95%) filter će označiti ovaj mejl kao neželjen.

3.1.1 Matematička osnova

Bajesov filter elektronske pošte koristi Bajesovu teoremu[3]

Bajesova teorema:

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

gde je:
$P(A)$ – *početna verovatnoća događaja* A
$P(B)$ – *početna verovatnoća pojavljivanja instance* B
$P(B|A)$ – *uslovna verovatnoća pojavljivanja instance* B *uz uslov ispravnosti događaja* A
$P(A|B)$ – *uslovna verovatnoća ispravnosti događaja* A *nakon pojavljivanja instance* B *koja je zanimljiva sa stanovišta indukcije znanja jer omogućava procenu ispravnosti događaja nakon posmatranja pojave novih instanci* B.

[3]Ime je dobila po engleskom matematičaru Tomasu Bajesu(1702 – 1761). Teoremu nikad nije objavio već su njegove beleške obelodanjene nakon njegove smrti od strane Ričarda Prajsa u Kraljevskom društvu Londona 1763.

Teorema sledi direktno iz definicije uslovne verovatnoće:

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

Kod konačno mnogo disjunktnih slučajeva $A_i, i = 1, ..., n$ Bajesova teorema glasi:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)}$$

Primer Bajesove teoreme[12]:

- Doktor zna da meningitis u 50% slučajeva prouzrokuje kočenje vrata.
- Prethodna (poznata) verovatnoća da bilo koji pacijent ima meningitis je $\frac{1}{50000}$.
- Prethodna verovatnoća da bilo koji pacijent ima ukočen vrat je $\frac{1}{20}$.

Ako pacijent ima ukočen vrat, koja je verovatnoća da ima i meningitis?

$$P(M|S) = \frac{P(S|M)P(M)}{P(S)} = \frac{0.5 \cdot \frac{1}{50000}}{\frac{1}{20}} = 0.0002$$

□

Pretpostavimo da poruka sadrži reč *replika*. Filter neželjene pošte će koristiti sledeću formulu koja se zasniva na Bajesovoj teoremi:

$$P_r(S|W) = \frac{P_r(W|S) \cdot P_r(S)}{P_r(W|S) \cdot P_r(S) + P_r(W|H) \cdot P_r(H)}$$

gde je:
$P_r(S|W)$ verovatnoća da je poruka neželjena ako znamo da se reč *replika* nalazi u njoj.
$P_r(S)$ ukupna verovatnoća da je bilo koja data poruka neželjena.
$P_r(W|S)$ verovatnoća da se reč *replika* nalazi u neželjenoj poruci.
$P_r(H)$ ukupna verovatnoća da je bilo koja data poruka legitimna.
$P_r(W|H)$ verovatnoća da se reč *replika* nalazi u legitimnoj poruci.

Nedavne statistike [26] pokazuju da je verovatnoća da je neka poruka neželjena 80%, pa je: $P_r(S) = 0.8$ i $P_r(H) = 0.2$. Međutim većina filtera neželjene pošte pravi pretpostavku da ne postoji razlog da dolazeća pošta ima veću verovatnoću da bude neželjena i pretpostavlja da je verovatnoća u oba slučaja jednaka 50%: $P_r(S) = 0.5$ i $P_r(H) = 0.5$. Za filtere koji koriste ovu hipotezu kaže sa da su nepristrasni jer nemaju predrasude o

dolazećoj pošti. Ova pretpostavka dozvoljava nam pojednostavljivanje polazne formule na sledeći način[4]:

$$P_r(S|W) = \frac{P_r(W|S)}{P_r(W|S) + P_r(W|H)}$$

Broj $P_r(W|S)$ koji se koristi u ovoj formuli je aproksimiran frekvenciji poruka koje sadrže reč *replika* u porukama koje su identifikovane kao neželjene tokom faze učenja. Slično je $P_r(W|H)$ je aproksimirano frekvenciji poruka koje sadrže reč *replika* u porukama koje su identifikovane kao legitimne tokom faze učenja. Da bi ove aproksimacije imale smisla neophodno je da skup poruka u fazi učenja bude dovoljno velik i reprezentativan. Takođe je preporučljivo da skup poruka u fazi učenja bude usaglašen sa 50% hipoteze o ponovnoj podeli poruka, odnosno da su skupovi podataka za neželjenu i legitimnu poštu iste veličine[1]. Naravno, odlučivanje da li je poruka neželjena ili legitimna na osnovu samo jedne reči *replika* je podložno greškama, pa zbog toga Bajesov filter pokušava da napravi zaključak na osnovu više reči kombinovanjem više verovatnoća.

3.1.2 Kombinovanje individualnih verovatnoća

Većina Bajesovih algoritama za filtriranje elektronske pošte su zasnovani na formulama koje su striktno validne samo ako su reči prezentovane u poruci uslovno nezavisni događaji. Ovaj uslov nije generalno zadovoljen jer je na primer u prirodnim jezicima verovatnoća nalaženja prideva zavisi od verovatnoće javljanja imenica u tekstu, ali je korisno kao idealizacija jer su statističke korelacije pojedinih reči obično nepoznate. Na osnovu toga može se izvesti sledeća formula iz Bajesove teoreme [3]:

$$p = \frac{p_1 p_2 \cdots p_N}{p_1 p_2 \cdots p_N + (1-p_1)(1-p_2)\cdots(1-p_N)}$$

gde je:

p verovatnoća da je posmatrana poruka neželjena
p_1 je verovatnoća $p(W_1|S)$ da je poruka neželjena znajući da sadrži prvu reč (npr. *replika*)
p_2 je verovatnoća $p(W_2|S)$ da je poruka neželjena znajući da sadrži drugu reč (npr. *satovi*)
...
p_N je verovatnoća $p(W_N|S)$ da je poruka neželjena znajući da sadrži N-tu reč (npr. *kuća*)

Spam filteri bazirani na ovoj formuli ponekad se nazivaju naivni Bajesovi klasifikatori. Rezultat p se tipično poredi sa pragom koji odlučuje da li je poruka neželjena ili legitimna. Ako je p niže od praga poruka se smatra legitimnom, inače se smatra neželjenom.

[4]Vrednost $P_r(S|W)$ se u literaturi popularno naziva *spamicity* ili *spaminess* i predstavlja procenat pojavljivanja date reči u neželjenoj pošti. Ona se računa za svaku reč koja se pojavljuje u poruci i kreće se u opsegu od 0 do 1.

3.1.3 | Rad sa retkim rečima

U slučajevima kada reč nikada nije korišćena u fazi učenja i brojilac i imenilac su jednaki nuli. Filter može da odluči da odbaci takve reči za koje ne postoji informacija. Reči sa kojima smo se susreli samo nekoliko puta u fazi učenja prouzrokovaće problem jer je pogrešno slepo verovati informacijama koje pružaju. Jednostavno rešenje je da se izbegava uzimanje u obzir takvih nepouzdanih reči.

Primena Bajesove teoreme pod pretpostavkom klasifikacije između neželjene i legitimne pošte koja sadrži reč *replika* je slučajna promenljiva sa beta raspodelom. Neki programi koriste sledeću korigovanu verovatnoću:

$$P_r^{'}(S|W) = \frac{s \cdot P_r(S) + n \cdot P_r(S|W)}{s + n}$$

gde je:
$P_r^{'}(S|W)-$ korigovana verovatnoća da je posmatrana poruka neželjena, znajući da sadrži datu reč
$s -$ vrednost koju prosleđujemo kao dodatnu informaciju o dolaznoj neželjenoj poruci
$P_r(S)-$ verovatnoća da je dolazna poruka neželjena
$n -$ broj pojavljivanja date reči tokom faze učenja
$P_r(S|W)-$ verovatnoća da je poruka neželjena ukoliko znamo da se data reč nalazi u njoj

$P_r(S)$ se ponovo može uzeti da je jednako 0.5 da bi izbegli suvišnu sumljičavost oko dolazne elektronske pošte. Za promenljivu s dobra vrednost je 3 što znači da je u fazi učenja neophodno da se data reč mora sadržati u više od tri poruke kako bi imali više poverenja u $P_r(S|W)$ vrednost nego u podrazumevanu vrednost.
Ova formula se može proširiti na slučaj kada je n jednako nuli i u ovom slučaju se ocenjuje kao $P_r(S)$.

3.1.4 | Druge heuristike

Neutralne reči kao *,,the"*, *,,a"*, *,,some"* ili *,,is"* (na engleskom) ili njihovi ekvivalenti na drugim jezicima mogu da se ignorišu. Neki Bajesovi filteri ignorišu reči čiji je *spamicity*[4] blizu 0.5 jer veoma slabo dobrinose dobrom odlučivanju. Reči koje se uzimaju u razmatranje su one čiji je *spamicity* blizu 0.0 (*karakteristični znaci da je poruka legitimna*) ili 1.0 (*karakteristični znaci da je poruka neželjena*).

Neki programi uzimaju u obzir činjenicu da li se data reč pojavljuje više puta u ispitivanoj poruci [8], a neki softverski proizvodi koriste šablone (*nizove reči*) umesto izolovanih reči prirodnih jezika [31]. Na primer ukoliko koristimo šablone za kontekst od četiri reči *spamicity* će se računati nad nizom od te četiri reči *,,vijagra je dobra za"* umesto izračunavanja *spamicity* za *,,vijagra"*, *,,je"*, *,,dobra"*, *,,za"*. Samim tim ovaj metod daje više osetljivosti na kontekst i bolje eliminiše šum[5].

[5] Ukoloko podaci imaju šum data pripadnost u skupu obuke ne mora biti tačna.

3.1.5 | Primena

Kada prihvatimo poruku m moramo definisati funkciju odlučivanja f koja dodeljuje poruku m svojoj klasi. Spam poruke obeležavaćemo sa S, a legitimne sa L. Ako je G_M skup poruka tada funkciju f definišemo na sledeći način:

$$f : G_M \longrightarrow \{S, L\}$$

Kod ovakvih tehnika prvo moramo proveriti neke karakteristike koje mogu uticati na klasifikaciju poruke. Pozivaćemo se na te karakteristike korišćenjem vektora $\overrightarrow{x}$. Neka je $P(\overrightarrow{x}/c)$ verovatnoća da klasa c generiše poruku čiji je vektor karakteristika $\overrightarrow{x}$. Ako pretpostavimo da legitimna poruka nikad nije sadržala tekst t =*"Kupi sada"* i da je $x = (m = utv)$ gde su u i v dva stringa, tada je verovatnoća $P(\overrightarrow{x}/L) = 0$. Sada je problem izračunati verovatnoću da poruka koja sadrži karakteristični vektor $\overrightarrow{x}$ pripada klasi c što zapisujemo kao $P(c/\overrightarrow{x})$[22]. To možemo dobiti posmatranjem Bajesovog pravila:

$$P(c/\overrightarrow{x}) = \frac{P(\overrightarrow{x}/c)P(c)}{P(\overrightarrow{x})} = \frac{P(\overrightarrow{x}/c)P(c)}{P(\overrightarrow{x}/S)P(S) + P(\overrightarrow{x}/L)P(L)}$$

$P(\overrightarrow{x})$ predstavlja apriornu verovatnoću pojavljivanja poruke čiji je karakteristični vektor $\overrightarrow{x}$, a $P(c)$ predstavlja verovatnoću da bilo koja poruka pripada klasi c. Znajući verovatnoće $P(c)$ i $P(\overrightarrow{x}/c)$ dovoljno je zaključiti $P(c/\overrightarrow{x})$. Tada imamo sledeće pravilo klasifikacije: Ukoliko je $P(S/\overrightarrow{x}) > P(L/\overrightarrow{x})$ (*aposteriorna verovatnoća poruka koje imaju karakterističan vektor* $\overrightarrow{x}$ *su veće od istih poruka koje pripadaju klasi* L) tada se poruka m klasifikuje kao neželjena. Da bi smo mogli da klasifikujemo poruku neophodno je da odredimo verovatnoće $P(c)$ i $P(\overrightarrow{x}/c)$ za bilo koju poruku m ali to očigledno ne može da se odredi precizno. Međutim možemo da aproksimiramo ove verovatnoće trening podacima. Na primer verovatnoća $P(S)$ se može grubo odrediti izračunavanjem odnosa broja neželjenih poruka i broja svih poruka u trening podacima.

Zbog jednostavnosti možemo smatrati da je karakterističan vektor binaran gde se prisustvo reči w u poruci m reprezentuje jednom jedinicom. Tada možemo reći da je:

$$P(x_\omega = 1/S) \approx \frac{\textit{Broj neželjenih poruka u kojima je w obeleženo}}{\textit{Broj svih neželjenih poruka}}$$

U suštini mi predstavljamo prisutnost reči ω_i u poruci m vrednošću 1 u karakterističnom vektoru $\overrightarrow{x} = (x_1, x_2, ..., x_n)$. Međutim algoritam 1 mora da računa 2^n vrednosti x što je nepraktično. Kako bismo to izbegli uvodimo pretpostavku da su dve prisutne reči uslovno nezavisne jedna od druge što znači da je:

$$P(\overrightarrow{x}/c) = \prod_{i=1}^{n} P(\overrightarrow{x}_i/c) \qquad \Lambda = \prod_{i=1}^{n} \Lambda_i(\overrightarrow{x}_i)$$

Algoritam 1 Trening algoritam za Bajesov klasifikator

for all $c \in \{S, L\}$ **do**
    Izračunaj $p(c)$
    for all $x_\omega \in \{0, 1\}$ **do**
        Izračunaj $p(\overrightarrow{x}_\omega / c)$
    end for
end for
for all $\overrightarrow{x}_\omega \in \{0, 1\}$ **do**
    Izračunaj $p(\overrightarrow{x}_\omega / c)$ ▷ koristeći Bajesovo pravilo
end for
for all $x_\omega \in \{0, 1\}$ **do**
    Izračunaj $\Lambda(\overrightarrow{x}_\omega)$
end for
Izračunaj $\lambda \frac{P(L)}{P(S)}$

Algoritam 2 Klasifikacija zasnovana na Bajesovom klasifikatoru

Izračunaj vektor $\overrightarrow{x}_\omega$ *ulazne poruke m*
if $\Lambda(\overrightarrow{x}_\omega) > \lambda \frac{P(L)}{P(S)}$ **then**
    poruka m je neželjena
else
    poruka m je legitimna
end if

Pri čemu $\Lambda(\overrightarrow{x})$ predstavlja količnik $\frac{P(\overrightarrow{x}/S)}{P(\overrightarrow{x}/L)}$, dok je λ parametar koji ukazuje na rizik kada klasifikujemo legitimnu poruku kao neželjenu.

$$\lambda = \frac{\mathcal{L}(L, S)}{\mathcal{L}(S, L)}$$

Funkcija $\mathcal{L}(c_1, c_2)$ određuje cenu loše klasifikacije pojavom klase c_1 umesto klase c_2. Logično je reći da je $\mathcal{L}(L, L) = \mathcal{L}(S, S) = 0$. Možemo definisati funkciju rizika kao:

$$R(c/\overrightarrow{x}) = \mathcal{L}(S, c)P(S/\overrightarrow{x}) + \mathcal{L}(L, c)P(L, \overrightarrow{x})$$

Klasifikovanje legitimne poruke kao neželjene je generalno mnogo veća greška od dopuštanja da neželjena poruka prođe kroz filter. Ukoliko imamo dva tipa grešaka $L \longrightarrow S$ i $S \longrightarrow L$ pretpostavljamo da je u slučaju greške $L \longrightarrow S$ parametar λ nekoliko puta veći nego u slučaju greške $S \longrightarrow L$. Poruka se klasifikuje kao neželjena ukoliko je sledeći kriterijum ispunjen:

$$\frac{P(\overrightarrow{x}|c = S)}{P(\overrightarrow{x}|c = L)} > \lambda$$

Ukoliko važi pretpostavka nezavisnosti procenjene verovatnoće su precizne i klasifikator na osnovu ovih kriterijuma postiže optimalne rezultate [5].

3.2 K najbližih suseda

Ovaj metod se naziva još i metod zasnovan na memoriji ili instancama. Pripada porodici algoritama učenja koji umesto izvođenja eksplicitne generalizacije upoređuje nove instance sa instancama koje se nakon trening faze nalaze u memoriji. Učenje zasnovano na instancama je vrsta *lenjog učenja* [6]. Zajednička karakteristika ovih metoda je da svi oni smeštaju trening instance u memorijske strukture i koriste ih direkrno za klasifikaciju. Najednostavniji oblik memorijske strukture je višedimenzionalni prostor definisan atributima u inicijalni vektor. Svaka trening instanca je reprezentovana kao tačka u prostoru. Procedura klasifikacije je najčešće jednostavna varijanta algoritma *k najbližih suseda*.

Klasifikacija novih instanci se obavlja prema principu najbližeg suseda, gde se nova instanca upoređuje sa instancama iz skupa za učenje korišćenjem definisane metrike. Metrika definiše rastojanje instanci na osnovu vrednosti njihovih atributa, a odgovara intuitivnom shvatanju sličnosti instanci tako da ako su instance sličnije rastojanje je manje. Nova instanca se klasifikuje na osnovu pretraživanja skupa za učenje sa ciljem pronalaženja instance koja mu je u smislu rastojanja najbliža. Nova instanca koja se klasifikuje dobija klasu te instance.

Ideja algoritma *k najbližih suseda*:

1. **Trening** - smeštanje trening poruka
2. **Klasifikacija** - Konkretna poruka x sadrži k najbližih suseda među porukama u trening skupu. Ukoliko ima više spam poruka među ovim susedima, poruka će biti klasifikovana kao spam, u suptrotnom klasifikator će je klasifikovati kao legitimnu poruku.

Kao što se može primetiti, trening faza ne postoji u običajenom smislu. Cena ovoga je sporija procedura odlučivanja jer u cilju klasifikovanja jedne poruke moramo računati rastojanja do svih trening poruka i pronaći k najbližih suseda. Ovo pri najtrivijalnijoj implementaciji može trajati $O(nm)$ vremena za trening skup od n poruka koje sadrže vektore obeležija sa m elemenata. Izvršavaje pametnog indeksiranja u trening fazi će omogućiti smanjenje kompleksnosti klasifikacije poruka na $O(n)$ [7]. Drugi problem koji se javlja kod ovog algoritma je nepostojanje parametara čije će podešavanje smanjiti broj lažno pozitivnih instanci. Taj problem se jednostavno rešava modifikacijom klasifikacionog pravila na sledeće l/k pravilo:

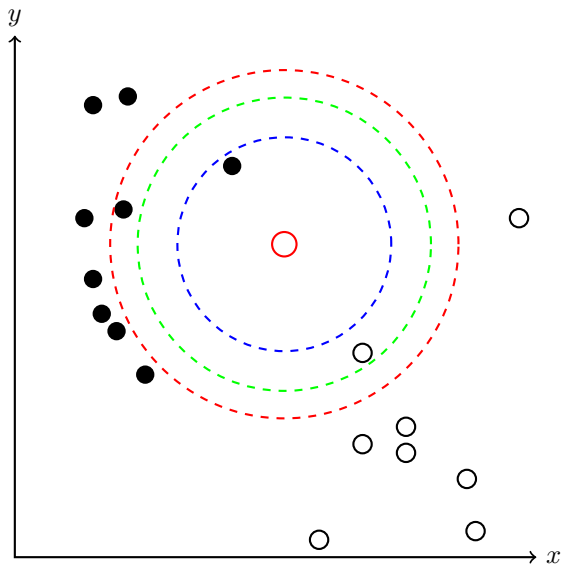
Ukoliko je l ili više poruka među k najbližih suseda poruke x spam klasifikuj x kao spam, u suprotnom klasifikuj x kao legitimnu poruku.

[6] Lenjo učenje u veštačkoj inteligenciji je poželjna osobina učenja ako postoji zahtev za stalnim menjanjem baze znanja, gde svaka takva promena ne povlači ponavljanje celog postupka učenja već samo efikasno inkrementalno dodavanje znanja.

Pseudo kod algoritma *k najbližih suseda*:

Algoritam 3 K najbližih suseda

Ulaz:
k - broj suseda
z - nova instanca opisana sa m obeležija
TS - trening skup od n instanci opisan sa m obeležja
Kat - klase instanci iz trening skupa
Izlaz:
c - predviđena kategorija nove instance
procedure kNajbližihSuseda(k, z, TS, Kat)
    for all $x \in \{TS\}$ **do** ▷ računanje rastojanja od z do svih ostalih instanci x iz trening skupa
        $d(x) = rastojanje(x, z)$;
    end for
    $[sortirane\_instance, index] = sort(d)$; ▷ sortiranje distanci u rastućem poretku
    $kategorija\_suseda = Kat(index(1 : k))$;
    $c = vecinaGlasova(kategorija\_suseda)$;
return c
end procedure



Slika 3: Primer koji ilustruje algoritam *k najbližih suseda* za $k = 1$, $k = 2$, $k = 3$

Mana klasifikatora zasnovanih na memoriji je cena izračunavanja u fazi klasifikacije zbog njihove *,,lenje"* prirode jer svi trening primeri morajui da sprovedu klasifikaciju što je sa velikim skupovima podataka posebno teško.

3.2.1 | Euklidska metrika

Element tehnike klasifikacije zasnovane na instancama koji utiče na oblik modela je metrika, pri čemu je u upotrebi više različitih metrika, a najčešće se koristi *euklidska*. Euklidsko rastojanje ispunjava sledeće dobro poznate osobine metrike:

1. Pozitivna određenost
   $d(x, y) \geq 0$ za svako x i y
   $d(x, y) = 0$ samo ako je $x = y$
2. Simetrija
   $d(x, y) = d(y, x)$ za svako x i y
3. Nejednakost trougla
   $d(x, z) \leq d(x, y) + d(y, z)$ za svako x, y i z

Euklidsko rastojanje instanci x i y se može predstaviti na sledeći način:

$$d(x, y) = ||x - y|| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

U datim izrazima se implicitno pretpostavlja da su svi atributi numeričkog tipa, tj. da su vrednosti kordinata brojevi, a kako bi se definicija euklidskog rastojanja mogla primeniti na nominalne atribute potrebno je definisati operaciju razlike nad nominalnim vrednostima.

Ako su sa $a_i, a_j \in Dom(A_i)$ označene dve proizvoljne vrednosti nominalnog atributa A_i, onda je razlika vrednosti a_i i a_j definisana $0 - 1$ funkcijom razlikovanja na sledeći način:

$$a_i - a_j = \begin{cases} 0 & \text{za } a_i = a_j \\ 1 & \text{inače} \end{cases}$$

Zbog različitih skala merenja za različite atribute postoji problem vezan za korišćenje različitih metrika. Tako na primer, atribut čiji se raspon vrednosti kreće unutar desetih delova merne jedinice imaće zanemariv uticaj na konačni rezultat u odnosu na atribut sa rasponom vrednosti od nekoliko desetina mernih jedinica. Zato je kod metoda klasifikacije zasnovane na instancama uobičajan postupak normalizacije svih numeričkih atributa na intervalu $[0, 1]$, korišćenjem funkcije:

$$f(x) = \frac{x - x_{min}}{x_{max} - x_{min}}$$

gde x_{min} i x_{max} označavaju najmanju, odnosno najveću vrednost posmatranog atributa. Kod metoda klasifikacije zasnovanim na instancama, neodređene vrednosti atributa se

tretiraju slično nominalnim atributima, tj. proširuje se definicija operacije razlike, tako da se razlika definiše na način da je neodređena vrednost maksimalno udaljena od bilo koje posmatrane vrednosti atributa.

3.2.2 | Pretraga prostora rešenja

Neke varijante klasifikacije zasnovane na instancama nastoje da redukuju broj instanci u skupu za učenje, prvenstveno radi smanjenja opsega pretraživanja pri klasifikaciji novih instanci, jer skup instanci za učenje po pravilu sadrži veliki broj redundantnih instanci. Kod problema klasifikacije najvažnije su instance koje se nalaze u blizini granica među klasama, a instance iz unutrašnjosti određenog područija klasa mogu se izostaviti bez posledica na tačnost klasifikacije. Postupak formiranja podskupa instanci je iterativan, a sastoji se od uvrštavanja ili eliminisanja instanci prema unapred definisanom kriterijumu. Zbog toga je u takvom modelu neophodno zadržati reprezentativne instance koje dobro ograničavaju područije u kojem se nalaze, a iz skupa izbaciti instance koje bitno ne doprinose oblikovanju područija odgovarajuće klase. Kriterijumi prihvatanja i eleminisanja instanci uglavnom pretstavljaju nepovratne strategije pohlepnog karaktera. Najjednostavniji kriterijum je da se ispituje instanca prema rezultatu klasifikacije korišćenjem do tada izdvojenih reprezentativnih instanci. Ako je u pitanju netačna klasifikacija instance ona se pridodaje u skup reprezentativnih instanci jer je evidentno da menja granice klasa. Ukoliko je klasifikacija instance tačna tada se ona proglašava suvišnom jer informacija koju nosi je već sadržana u skupu pomoću kojeg je klasifikovana[11].

Nedostaci datog kriterijuma za izbor instanci[11]:

- U početnoj fazi procesa pretraživanja postoji nezanemariva verovatnoća odbacivanja instanci koje se mogu pokazati važnim za tačnost klasifikacije rezultujućeg modela.
- Pored ovoga, izabrani podskup reprezentativnih instanci ne zavisi samo od polaznog skupa, već i o redosledu evaluacije instanci.
- Bitan nedostatak se odnosi na loše ponašanje u uslovima šuma u podacima, jer s obzirom da se u skup uvrštavaju i netačno klasifikovane instance, ovaj kriterijum ima tendenciju akumuliranja instanci sa šumom u rezultirajućem skupu instanci, što dovodi do smanjenja njegove reprezentativnosti.

3.2.3 | Udaljenost instanci

Pored izbora instanci za pamćenje, na oblik klasifikacionog modela se može uticati i modifikacijom funkcije rastojanja. Jednak uticaj svih atributa u instanci na konačan rezultat je jedno od svojstava euklidskog rastojanja, ali su u praksi retki problemi kod kojih svi atributi imaju jednaku vrednost za proces klasifikacije, čime se stvara mogućnost za poboljšanje tehnike klasifikacije zasnovane na instancama. Modifikacija euklidskog rastojanja podrazumeva uvođenje težinskih vrednosti atributa. Ako sa w_i označimo težinsku vrednost pridruženu atributu A_i, onda modifikovano Euklidsko rastojanje instanci x i y

možemo predstaviti na sledeći način:

$$d_w(x, y) = \sqrt{\sum_{i=1}^{n} w_i^2 (x_i - y_i)^2}$$

Veliki uticaj na proračun rastojanja instanci pruža mu velika težinska vrednost pridružena atributu. Menjanje težinskih vrednosti atributa je jedan od načina korekcije klasifikacijskog modela u tehnici klasifikacije zasnovane na instancama. I pored toga što postoje nezavisni postupci ocene vrednosti za proces klasifikacije atributa, u kontekstu ove tehnike modeliranja težine atributa se najčešće određuju interno, prilikom formiranja relevantnog podskupa instanci za učenje[11].

Svim atributima je inicijalno pridružena težinska vrednost 1, koja se iterativno modifikuje pri razmatranju svake od instanci iz skupa za učenje. U podskupu relevantnih instanci se pronalazi instanca y najbliža posmatranoj instanci x, kao i pri klasifikaciji instanci. Ako instance x i y pripadaju istoj klasi, smanjuje se težinska vrednost atributa čije se vrednosti u instancama x i y najviše razlikuju, jer se razlika u vrednostima tih atributa pripisuje slabijoj korelaciji sa klasom, kao i što se u slučaju da instance x i y pripadaju različitim klasama, težinska vrednost atributa sa najvećom razlikom vrednosti povećava. Povećanje, odnosno smanjenje težinske vrednosti proporcionalno je razlici vrednosti atributa u instancama x i y.

3.2.4 Šum u podacima za učenje

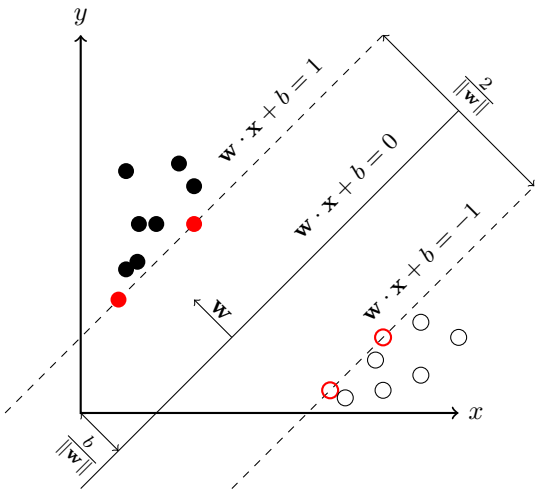
Osnovni oblik tehnike *k najbližih suseda* je prilično podložan problemu šuma u podacima za učenje, a razlog tome je da se klasifikacija nove instance oslanja na samo jednu (najbližu) instancu iz skupa za učenje. Značajno smanjenje uticaja šuma može se sprovesti proširenjem postupka klasifikacije prema principu k najbližih suseda, gde se umesto izdvajanja samo jedne najbliže instance iz skupa za učenje, izdvaja k najbližih instanci, za neki unapred određeni broj k. U klasifikaciji nove instance učestvuje k pronađenih instanci, po principu većinskog glasanja, pri čemu se instanci pridružuje najfrekventnija klasa unutar izdvojenih k instanci. Specijalni slučaj ovog uopštenja za $k = 1$ predstavlja osnovni algoritam klasifikacije. Neophodno je da k ne bude suviše malo jer kNN klasifikator će biti podložan preprilagođavanju (*overfitting*[7]) zbog šuma u trening podacima.

Vrednost konstante k zavisi od količine šuma u podacima za učenje, pri čemu ako, je više šuma, povoljnije je izabrati veće vrednosti konstante k. Na ovaj način se postiže poboljšanje tačnosti klasifikacije u uslovima šuma, zbog čega je varijanta k najbližih suseda gotovo u potpunosti istisnula osnovni oblik algoritma. Za posmatrani skup instanci, može se dokazati da za $|S| \to \infty$ i $k \to \infty$ na način da $\frac{k}{|S|} \to 0$ verovatnoća pogrešne klasifikacije teži teoretskom minimumu[11].

[7]Do preprilagođavanja dolazi kada je model takav da je greška pri treniranju mala, a greška pri testiranju značajno veća. Postoje razni razlozi zašto dolazi do preprilagođavanja modela kao što su prisustvo šuma ili nepostojanje reprezentativnih instanci.

3.3 | Metod podržavajućih vektora (SVM)

Metod podržavajućih vektora predstavlja binarni klasifikator koji konstruiše hiper-ravan u i na taj način stvara model koji predviđa kojoj od dve klase pripada nova instanca. Ovaj metod je razvijen 1995. godine i veoma je popularan zbog dobrih rezultata koji se dobijaju. Ideja metode je da se u vektorskom prostoru u kome su podaci predstavljeni nađe razdvajajuća hiper-ravan tako da su svi podaci iz iste klase sa iste strane ravni, kao što je prikazano na slici 4.



Slika 4: Primer koji ilustruje optimalnu hiper-ravan sa maksimalnom marginom koja razdvaja podatke za trening u 2 klase

Ako znamo da su podaci linearno razdvojivi, u fazi treniranja je neophodno pronaći optimalnu razdvajajuću hiper-ravan, a to je ravan sa maksimalnom marginom. Margina predstavlja širinu razdvajanja između klasa koju treba maksimizovati i iznosi $\rho = \frac{2}{||\mathbf{w}||}$. Kada pronađemo najbolju hiper-ravan i njenu jednačinu, imamo model na osnovu kog određujemo klasu nove instance[13]. Na slici 4 je prikazan linearni klasifikator kod koga prava $\mathbf{w} \cdot \mathbf{x} + b = 0$ predstavlja granicu odlučivanja. SVM pronalazi optimalno rešenje koje maksimizuje razdaljinu između hiper-ravni i tačaka koje su blizu razdvajajuće hiper-ravni i predstavlja intuitivno rešenje: ako nema tačaka blizu linije razdvajanja, onda će klasifikacija biti relativno laka.

3.3.1 | Linearno razdvojive klase

Definicija: Neka $X = (x_i, y_i)$, $x_i \in \mathbb{R}^D$, $y_i \in \{-1, +1\}$ označava skup primera za testiranje u $D-$dimenzionalnom prostoru, gde svaki uzorak ima D atributa.

Ukoliko su podaci linearno razdvojivi možemo da nađemo pravu za dvodimenzionalni slučaj, odnosno hiper-ravan za višedimenzionalni slučaj. Tada izrazom $\mathbf{w} \cdot \mathbf{x} + b = 0$ možemo opisati hiper-ravan pri čemu je $\mathbf{w}$ normala hiper-ravni i $\frac{b}{||\mathbf{w}||}$ vertikalna udaljenost hiper-ravni od kordinatnog početka. Instance najbliže razdvajajućoj hiper-ravni su podržavajući vektori. Cilj je izabrati hiper-ravan maksimalno udaljenu od najbližih uzoraka obe klase, kao što je prikazano na slici 4. Neophodan uslov za pronalaženje razdvajajuće hiper-ravni svodi se na izbor parametara $\mathbf{w}$ i b, takvih da ulazne podatke možemo opisati sledećim izrazima:

$$x_i \cdot \mathbf{w} + b \geq +1 \ za \ y_i = +1$$
$$x_i \cdot \mathbf{w} + b \leq -1 \ za \ y_i = -1$$

Kombinovanjem dva prethodna izraza dobijamo:

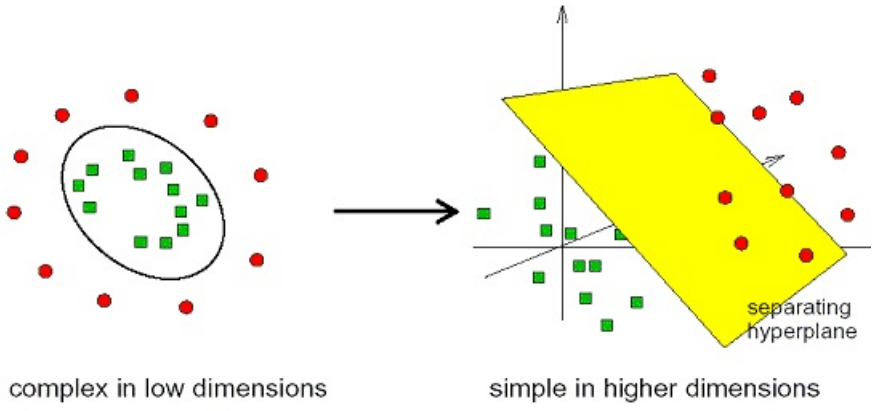
$$y_i(x_i \cdot \mathbf{w} + b) - 1 \geq 0, \forall i$$

Da bi izabrali hiper-ravan maksimalno udaljenu od potpornih vektora, potrebno je maksimizovati marginu, što je ekvivalentno pronalaženju:

$$\min ||\mathbf{w}|| \ takav \ da \ y_i(x_i \cdot \mathbf{w} + b) - 1 \geq 0, \forall i$$

3.3.2 | Linearno nerazdvojive klase

U slučaju linearno nerazdvajajućih problema, koristimo nelinearni SVM, pri čemu je osnovna ideja da se osnovni (ulazni) vektorski prostor preslika u neki višedimenzioni prostor u kome je skup podataka za trening linearno razdvojiv[13]. Na slici 5 prikazano je preslikavanje u višedimenzioni prostor u kome je skup podataka za trening linearno razdvojiv.



Slika 5: Preslikavanje u višedimenzionalni prostor u kome je skup podataka linearno razdvojiv

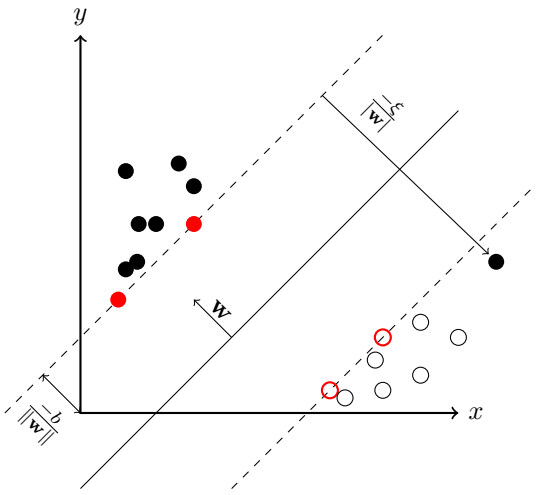
Uvođenjem nenegativne vrednosti ξ_i ublažavamo uslove linearnog SVM-a i dobijamo sledeće izraze:

$$x_i \cdot \mathbf{w} + b \geq +1 - \xi_i \ za \ y_i = +1$$
$$x_i \cdot \mathbf{w} + b \leq -1 + \xi_i \ za \ y_i = -1$$

Kombinovanjem dva prethodna izraza dobijamo:

$$y_i(x_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0, \xi_i \geq 0, \forall i$$

Primenjena metoda naziva se *metoda meke margine* (eng. soft margin method) [16], a izvorno je nastala sa idejom dozvoljavanja pogrešnog označavanja klasa pre samog postupka učenja. Slika 6 prikazuje hiper-ravan kroz dve linearno nerazdvojive klase, gde je vidljiv i uzorak sa pogrešne strane hiperravni zbog kojeg prostor nije linearno razdvojiv. Mera rastojanja tog uzorka od pripadajueg potpornog vektora je ξ.



Slika 6: Primer koji ilustruje dve linearno nerazdvojive klase

Izbor razdvajajuće hiper-ravni svodi se na pronalaženje:

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{L}\xi_i \text{ takav da } y_i(x_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0, \xi_i \geq 0, \forall i$$

gde vrednost C predstavlja faktor greške, kojim dozvoljavamo određene greške pri treniranju, bez čega pronalazak hiper-ravni ne bi bio moguć.

Složenost treniranja SVM algoritma zavisi od kernela kao i od konačnog broja podržavajućih vektora koji se kriju u podacima za rešavanje dualnog problema. Kako broj podržavajućih vektora asimptotski raste linearno sa brojem primera, sledi da je u najboljem slučaju složenost $O(n^2)$ kada je C malo i $O(n^3)$ kada je C veliko. Kada je C preveliko optimizacioni algoritam će pokušati da smanji $||\mathbf{w}||$ koliko god je moguće, tako

da hiper-ravan pokušava pravilno da klasifikuje svaku instancu, ali to dovodi do gubitka svojstva generalizacije klasifikatora. S druge strane ukoliko je C premalo, optimizacioni algoritam će pokušati da poveća $||\mathbf{w}||$ previše, što dovodi do velike trening greške.

3.3.3 | Upotreba kernela

Kernel je funkcija koja odgovara skalarnom proizvodu u nekom proširenom prostoru (prostoru veće dimenzije). Računanje skalarnog proizvoda između vektora:

$$K(x_i, x_j) = x_i^T \cdot x_j$$

Preslikavanjem svake tačke u prostor veće dimezije koristeći transformaciju $\Phi : x \to \phi(x)$ skalarni proizvod postaje

$$K(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$$

pomoću koje se izračunavanje vrši na mnogo jednostavniji način.

Primer preslikavanja Φ:
Ulazni (originalni) dvodimezionalni prostor: $x = (x_1, x_2)$. Neka je $K(x_i, x_j) = (1+x_i^T x_j)^2$. Pokažimo da vredi $K(x_i, x_j) = \phi(x_i)^T \cdot \phi(x_j)$, ako je $\phi(x) = [1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1\sqrt{2}x_2]$

$$K(x_i, x_j) = (1 + x_i^T x_j)^2 = 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{j1}x_{i2}x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} =$$
$$[1, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2, \sqrt{2}x_{i1}\sqrt{2}x_{i2}]^T[1, x_{j1}^2, \sqrt{2}x_{j1}x_{j2}, x_{j2}^2, \sqrt{2}x_{j1}\sqrt{2}x_{j2}] = \phi(x_i)^T \cdot \phi(x_j)$$

Dakle kerneli omogućavaju pretvaranje nerazdvojivih problema u razdvojive i vrše bolje preslikavanje u prostore viših dimenzija. Matematička teorija definiše uslove koje data funkcija treba da zadovolji da bi predstavljala skalarni proizvod u nekom vektorskom prostoru: simetričnost, pozitivna definisanost, zatvorenost za linearne kombinacije, množenje skalarom, itd. Funkcija koja zadovolji te uslove može da se koristi kao kernel.

Primeri uobičajenih kernela:

1. **Linearni** – $K(x_i, x_j) = x_i^T \cdot x_j$
2. **Polinomijalni** – $K(x_i, x_j) = (1 + x_i^T x_j)^d$
3. **RBF** – $K(x_i, x_j) = e^{\frac{-|x_i - x_j|}{2\sigma^2}}$
4. **Sigmoid** – $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

Najjednostavniji je linearni kernel i odlično se ponaša za linearno odvojive podatke. Za potrebe evaluacije upotrebljen je upravo linearni kernel, a u okviru SVM^{light} biblioteke koja je korišćena pri implementaciji SVM algoritma, moguće je koristiti više različitih tipova kernela.

```
kernelParm.kernel_type = 0;
/* 0=linear, 1=poly, 2=rbf, 3=sigmoid, 4=custom, 5=matrix */
```

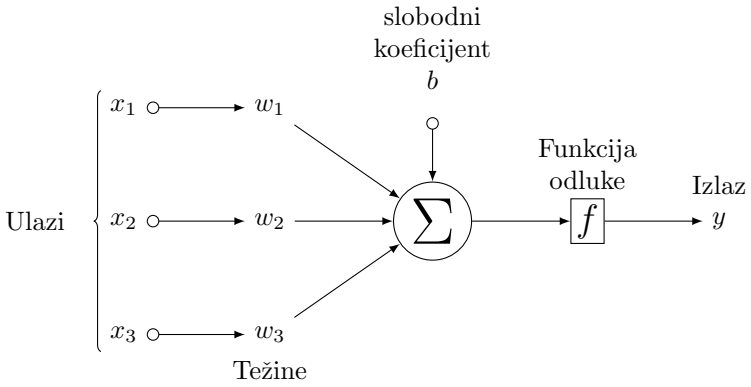
3.4 | Veštačke neuronske mreže

Neuronske mreže nastale su iz težnje za razvijanjem matematičkih struktura koje bi bile u mogućnosti da simuliraju rad ljudskog mozga, kao i da koriste te strukture u rešavanju praktičnih problema. Kako bi razumeli osnovnu strukturu veštačkih neuronskih mreža potrebno je razmotriti osnovnu strukturu ljudskog mozga. Ljudski mozak, čija je struktura složena, a mreža neurona gusta, sastoji se od oko 10^{11} neurona koji su međusobno povezani u slojeve, koji čine složenu mrežu. Biološka ćelija koja obrađuje informacije je neuron. Zbog složene strukture neurona još uvek nije došlo do detaljnijih saznanja o funkcionisanju ljudskog mozga.

Sastoje se iz velike klase algoritama koji imaju svoju primenu u klasifikaciji, regresiji i proceni gustine. U suštini, neuronske mreže su određene funkcijama složene reprezentacije koje mogu biti dekomponovane na manje delove (*neuroni, procesne jedinice*) i reprezentovane grafički kao mreža neurona. Veoma mnogo funkcija se može reprezentovati na ovaj način i zbog toga nije uvek jasno koji algoritam pripada kom polju neuronskih mreža. Međutim možemo reći da je perceptron osnovna jedinica neuronske mreže, dok je višeslojni perceptron jedan od najkorišćenijih algoritama nadgledanog učenja u modelima neuronskih mreža.

3.4.1 | Perceptron

Ideja perceptrona je pronaći linearnu funkciju vektora obeležja $f(x) = \mathbf{w}^T\mathbf{x} + b$ tako da je $f(x) > 0$ za vektore jedne klase i $f(x) < 0$ za vektore druge klase. Neka je $\mathbf{w} = (w_1, w_2, ..., w_m)$ vektor koeficijenata (*težina*) funkcije i b takozvani *slobodni koeficijent (bias)* [8]. Ako označimo klase brojevima $+1$ i -1 možemo reći da tražimo funkciju odluke $d(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$. Funkcija odluke se može reprezentovati grafički kao neuron i zbog toga se perceptron smatra neuronskom mrežom. To je naravno najtrivijalnija mreža sa jednom procesnom jedinicom.



Slika 7: Perceptron kao neuron

[8] Slobodni koeficijent ili bias (*sklonost*) je vrednost koja nam omogućuje transliranje funkcije odluke levo ili desno, što može biti kritično za uspešno učenje.

Ako vektori koji trebaju da budu klasifikovani imaju samo dve koordinate (važi da $x \in \mathbb{R}^2$) mogu se reprezentovati kao tačke na ravni. Funkcija odluke se može reprezentovati kao linija koja razdvaja ravan na dva dela. Vektori koji pripadaju jednoj polu-ravni biće klasifikovani kao pripadnici jedne klase dok će vektori koji pripadaju drugoj poluravni pripadati drugoj klasi. Ukoliko vektor ima tri koordinate, granica odlučivanja će biti u trodimenzionalnom prostoru, uopšteno ukoliko je vektor obeležja n-dimenzionalan granica odlučivanja je n-dimenzionalna hiper-ravan. Ove činjenice ukazuju na to da je perceptron linearni klasifikator.

Učenje perceptrona je urađeno iterativnim algoritmom. On počinje proizvoljno izabranim parametrima $(\mathbf{w}_0, b_0)$ za funkciju odluke i ažurira ih iterativno. Na n-toj iteraciji algoritma, trening primer $(\mathbf{x}, c)$ je odabran tako da ga trenutna funkcija odluke ne klasifikuje tačno (važi $\text{sign}(\mathbf{w}^T\mathbf{x} + b) \neq c$). Parametri $(\mathbf{w}_n, b_n)$ se ažuriraju korišćenjem pravila:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + c\mathbf{x} \qquad\qquad b_{n+1} = b_n + c$$

Algoritam se zaustavlja kada funkcija odluke tačno klasifikuje sve trening primere. Ukoliko takva funkcija ne postoji (klase nisu linearno razdvojive) algoritam učenja nikada ne konvergira i perceptron se ne može primeniti u ovom slučaju. Ukoliko podaci nisu linearno razdvojivi najbolje je zaustaviti trening algoritam kada broj pogrešno klasifikovanih postane dovoljno mali. Za očekivati je da pri klasifikovanju elektronske pošte podaci uvek budu linearno razdvojivi [9].

Algoritam 4 Treniranje perceptrona

1: Inicijalizuj $\mathbf{w}$ i b na nasumično odabrane vrednosti vrednosti ili na nulu
2: Pronađi trening primer $(\mathbf{w}, c)$ za koji važi $\text{sign}(\mathbf{w}^T\mathbf{x}+b) \neq c$. Ukoliko ne postoji takav primer trening je završen. Sačuvaj poslednje $\mathbf{w}$ i b i stani. U suprotnom idi na sledeći korak.
3: Ažuriraj $(\mathbf{w}, c)$: $\mathbf{w} := \mathbf{w} + c\mathbf{x}$, $b := b + c$. Idi na prethodni korak.

Algoritam 5 Klasifikacija perceptron pravilom

1: Za datu poruku $\mathbf{x}$, odredi njenu klasu kao $\text{sign}(\mathbf{w}^T\mathbf{x} + b)$

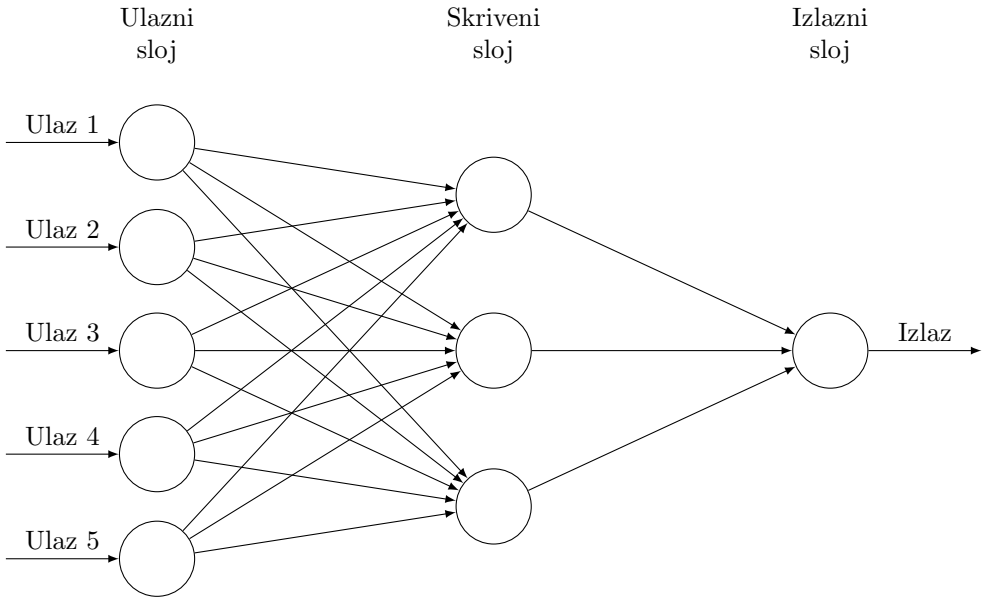
[9] Ovo važi zbog toga što je vektor obeležja koji koristimo mnogo veći od broja trening primera. Znamo da su u n-dimenzionalnom prostoru $n+1$ tačka linearno razdvojive u svakom slučaju. Činjenica da je vektor obeležja veći od trening primera može značiti da imamo "previše obeležja" i to nije uvek poželjno (pogledati [15]).

3.4.2 | Višeslojni perceptron

Višeslojni perceptron je funkcija koja se može vizualizovati kao mreža sa nekoliko slojeva neurona povezanih unapred. Neuroni u prvom sloju se nazivaju *ulazni neuroni* i reprezentuju ulazne promenljive. Neuroni u poslednjem sloju se nazivaju *izlazni neuroni* i pružaju funkciji rezultujuću vrednost. Slojevi između prvog i poslednjeg se nazivaju *skriveni slojevi*. Svaki neuron u mreži je sličan perceptronu jer na osnovu ulaznih vrednosti $x_1, x_2, ..., x_k$ izračunava izlaznu vrednost o korišćenjem formule:

$$o = \phi(\sum_{i=1}^{k} w_i x_k + b)$$

gde w_i predstavljaju težine, b slobodnog koeficijenta neurona i ϕ je određena nelinearna funkcija. Najčešće se za $\phi(x)$ uzima $\frac{1}{1+e^{ax}}$ ili $\tanh(x)$.



Slika 8: Struktura višeslojnog perceptrona

Treniranje višeslojnog perceptrona znači pretragu za takvim težinama i biasom svih neurona za koje će mreža imati najmanju moguću grešku na trening skupu. To jest, ukoliko označimo funkciju koju implementira mreža kao $f(\mathbf{x})$, tada u cilju treniranja mreže moramo da pronađemo parametre koji minimalizuju ukupnu grešku pri treniranju:

$$E(f) = \sum_{i=1}^{n} (f(\mathbf{x}_i) - c_i)^2$$

gde su $(\mathbf{x}_i, c_i)$ trening primeri. Ova minimizacija se može obaviti bilo kojim iterativnim algoritmom optimizacije. Najpopularniji je jednostavan gradijentni spust koji se u konkretnijem slučaju naziva *propagiranje greške unazad*. Detaljna specifikacija ovog algoritma je opisana u mnogim knjigama i radovima (pogledati [21] [19] [28]).

Višeslojni perceptroni su nelinearni klasifikatori jer su njihovi modeli nelinearne granice odluke između klasa. Kao što je malo pre pomenuto, trening podaci koje koristimo su linearno razdvojivi i korišćenje nelinearne granice odlučivanja teško da može poboljšati performanse. Zbog toga su najbolji rezultati koje možemo da očekujemo rezultati jednostavnog perceptrona.

Od svih algoritama mašinskog učenja, višeslojni perceptron verovatno sadrži najveći broj parametara koji se moraju štelovati *ad hoc*. Nije najjasnije koliko skrivenih neurona treba da sadrži i koje parametre izabrati za algoritam propagacije unazad u cilju postizanja dobre generalizacije. Mnogi radovi i knjige su napisani pokrivajući ovu temu ali treniranje višeslojnog perceptrona i dalje nije najjasnije. Na sreću to nas ne sprečava da ovaj metod učenja koristimo jer se uspešno primenjuje u zadacima filtriranja neželjenih poruka (pogledati [29]).

3.4.3 Karakteristike veštačkih neuronskih mreža

Sledi generalni rezime karakteristika veštačkih neuronskih mreža [27]:

1. Višeslojne neuronske mreže sa najmanje jednim skrivenim slojem se mogu koristiti za aproksimaciju bilo koje ciljne funkcije. Kako je metoda veoma izražajna važno je izabrati odgovarajuću mrežnu topologiju za dati problem da bi izbegli preprilagođavanje modela.

2. Veštačke neuronske mreže mogu da se izbore sa redudantnim karakteristikama podataka jer se težine automatski uče tokom procesa treniranja. U slučaju redudantnih karakteristika algoritam nastoji da težine učini malim.

3. Neuronske mreže su veoma osetljive na prisustvo šuma u trening podacima. Jedan pristup za rukovanje šumom je korišćenje validacionog skupa za određivanje greške generalizacije modela. Drugi pristup je smanjivanje težina za proizvoljan faktor pri svakoj iteraciji.

4. Metod gradijentnog spusta koji se koristi za učenje težina u veštačkim neuronskim mrežama često konvergira nekom lokalnom minimumu. Jedan način da se izbegne lokalni minimum je da se dodaje izraz na formulu za ažuriranje težina.

5. Treniranje veštačkih neuronskih mreža je proces koji troši mnogo vremena, posebno kada je broj skrivenih čvorova veliki. Ipak test primeri se mogu klasifikovati brzo.

3.5 Tehnika maksimalne entropije

Maksimalna entropija je klasičan model koji se često koristi u procesiranju prirodnih jezika [6]. Princip je da se pronađe odgovarajuća raspodela verovatnoće $p(a, b)$ koja maksimizuje entropiju:

$$H(p) = -\sum_{x \in A \times B} p(x) \log p(x)$$

gde A označava skup mogućih klasa, a B skup svih mogućih vrednosti vektora obeležja. Ova maksimizacija bi trebalo da održi p u skladu sa informacijama u trening skupu. p tada postaje:

$$p(a, b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha^{f_j(a,b)}$$

gde je k veličina vektora obeležja i

$$Z(b) = \sum_{a} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

je normalizacioni faktor koji osigurava

$$\sum_{a} p(a, b) = 1.$$

Eksponencijalni oblik funkcije raspodele je sledećeg oblika

$$p(a, b) = \frac{1}{Z(\alpha_1, ..., \alpha_m)} e^{[\alpha_1 f_1(a,b) + ... + \alpha_m f_m(a,b)]}$$

α_j se može izračunati korišćenjem generalizovanog iterativnog skaliranja [17]. Funkcija f se definiše na sledeći način:

$$f_{cp,a'}(a, b) = \begin{cases} 1, & \text{ako } a = a' \text{ i } cp(b) = true. \\ 0, & \text{inače.} \end{cases}$$

gde cp preslikava parove (a,b) u $\{true, false\}$. Rezultati prikazani u [32] pokazuju da je stopa greške bolja nego kod Bajesovog klasifikatora kada trening primeri rastu.

4 Podaci

U ovom poglavlju su opisani dostupni korpusi koji su korišćeni za testiranje algoritama koji su implementirani.

Korpus	Poruke	Spam	Legitimne
PU1	*1099*	*481*	*618*
PU2	*721*	*142*	*579*
PU3	*4139*	*1826*	*2313*
PUA	*1142*	*571*	*571*
Enron-Spam	*33716*	*17171*	*16545*
Ling-Spam	*2893*	*481*	*2412*

Tabela 5: Raspodela poruka u korpusima

4.1 Ling spam korpus

Ling spam korpus [10] sadrži nekoliko izvučenih verzija od 481 spam poruka kombinovanih sa 2412 legitimnih poruka pri čemu su informacije zaglavlja, izuzev linije naslova, uklonjene. U nekim varijantama korpusa, stop reči su uklonjene ili su reči zamenjene njihovim korenima. Duplirane poruke su umanjene. Standardna podela podataka je napravljena kako bi se olakšala desetostruka unakrsna validacija (*k-unakrsnih validacija za* $k = 10$).

Iako su spam i legitimne poruke izvučene iz različitih izvora, a informacije korisne za neke filtere su uklonjene, Ling spam korpus je predstavljao veoma realnu procenu u vreme njegovog publikovanja i veliki broj studija ga je koristio za evaluaciju.

4.2 PU1, PU2, PU3 i PUA spam korpusi

Skup podataka koji je korišćen pri istraživanju je *PU1 korpus* [4]. Korpus sadrži 1099 poruka, od kojih su 481 spam. Sastoji se od 4 direktorijuma koji odgovaraju kodiranim verzijama korpusa:

```
bare: Lemmatiser disabled, stop-list disabled.
lemm: Lemmatiser enabled, stop-list disabled.
lemm_stop: Lemmatiser enabled, stop-list enabled.
stop: Lemmatiser disabled, stop-list enabled.
```

Svaki od ova četiri direktorijuma sadrži po 10 poddirektorijuma, koji odgovaraju za 10 delova korpusa koji su korišćeni pri evaluaciji. U svakom ponavljanju jedan deo je rezervisan za testiranje dok su drugih 9 korišćeni za učenje. Svaki od ovih 10 poddirektorijuma sadrži i spam i legitimne poruke. Fajlovi čija su imena oblika **spmsg*.txt*

predstavljaju spam poruke, dok fajlovi čiji su nazivi oblika **legit*.txt* predstavljaju legitimne poruke.

Sve poruke unutar korpusa su predprocesirane: svi prilozi, HTML tagovi, zaglavlja izuzimajući *Subject* su uklonjeni dok su reči kodirane brojevima.

Razlikuje se od *Ling spam korpusa* u dva navedena aspekta, ali su inače slični:

- izvedeni su od realnih poruka elektronske pošte koje su slate pojedincima
- sadržaj tih privatnih poruka je kodiran tako da je svaka posebna reč zamenjena brojem

```
Subject: 10834 82 22908 18063 13824

 10834 82 22908 18063 13824 6306 118 86 86 6513 84 6598 84 6353 84 6383 86 10834 24731
 5068 5890 23139 20606 20526 131 10834 144 13521 24417 20259 47 2721 1351 1350 9956
 84 84 84 84 82 14766 797 80 6176 4806 22272 10727 22168 80 13566 80 9598 10932 12404
 80 19453 80 4898 80 8285 14313 12940 80 3144 80 2717 80 1083 3877 14519 9655 18124
 84 82 21537 22180 5890 16594 23139 20606 8503 15532 80 11406 80 21341 525 16422
 84 82 15973 24317 18104 12404 9887 19468 84 82 23478 24417 8704 725 80 14097 709
 51 7335 51 21343 13067 13508 20662 17880 22168 10160 9886 17820 17261 7659 23139
 187 86 228 10996 84 19468 80 14911 21537 15983 22168 20606 8503 9568 80 23139 23899
 16248 13242 80 11406 8669 9338 80 7059 7335 80 3239 289 6567 9655 19468 10834 22180
 4797 13886 47 19444 13343 12492 80 9855 18472 23139 18472 80 10834 1847 18882 8122
 10126 11817 80 21946 23139 20606 1350 84 13886 2170 23590 47 82 82 82 82 82 82 82 82
 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82
```

Slika 9: Primer kodirane poruke iz PU1 korpusa

PU2 korpus je najmanji, dok je PU3 korpus značajno veći od PU1 i PU2 korpusa (videti tabelu 5). PUA korpus sadrži 1142 poruka od čega je polovina legitimna dok polovina predstavlja spam. Sva tri korpusa sadrže poruke kodirane brojevima poput PU1 korpusa zbog privatnosti.

4.3 | Enron spam korpus

Enron spam korpus [2] je naslednik Ling Spam i PU korpusa koji sadrži hronološki podeljene e-mail poruke koje su primali šest Enronovih zaposlenih kombinovanih sa spamom iz različitih izvora. Korpus se sastoji iz paketa od šest parova skupa podataka za trening i testiranje u kojima su poruke u svakom trening skupu prethodile odgovarajućim u test skupu. Postoje dve verzije korpusa. Jedna verzija je predprocesirana na isti način kao Ling Spam korpus, dok druga verzija sadrži sirove poruke. Svaka poruka je u odvojenom tekstualnom fajlu. Broj na početku svakog naziva fajla predstavlja poredak pristizanja poruka.

5 Implementacija

U ovom poglavlju će biti prikazana implementacija algoritama mašinskog učenja za klasifikaciju elektronske pošte. Algoritmi su implementirani u programskom jeziku C++ sa ciljem testiranja na korpusima elektronske pošte koji su opisan u poglavlju 4. Svrha implemetacije je gruba procena performansi, brzine i vremenske zahtevnosti algoritama. Kod programa se može pogledati u dodatnom poglavlju A.

Implementirani su sledeći algoritmi:

◇ Naivni Bajesov klasifikator

◇ K najbližih suseda

◇ Metod podržavajućih vektora

◇ Neuronske mreže (perceptron)

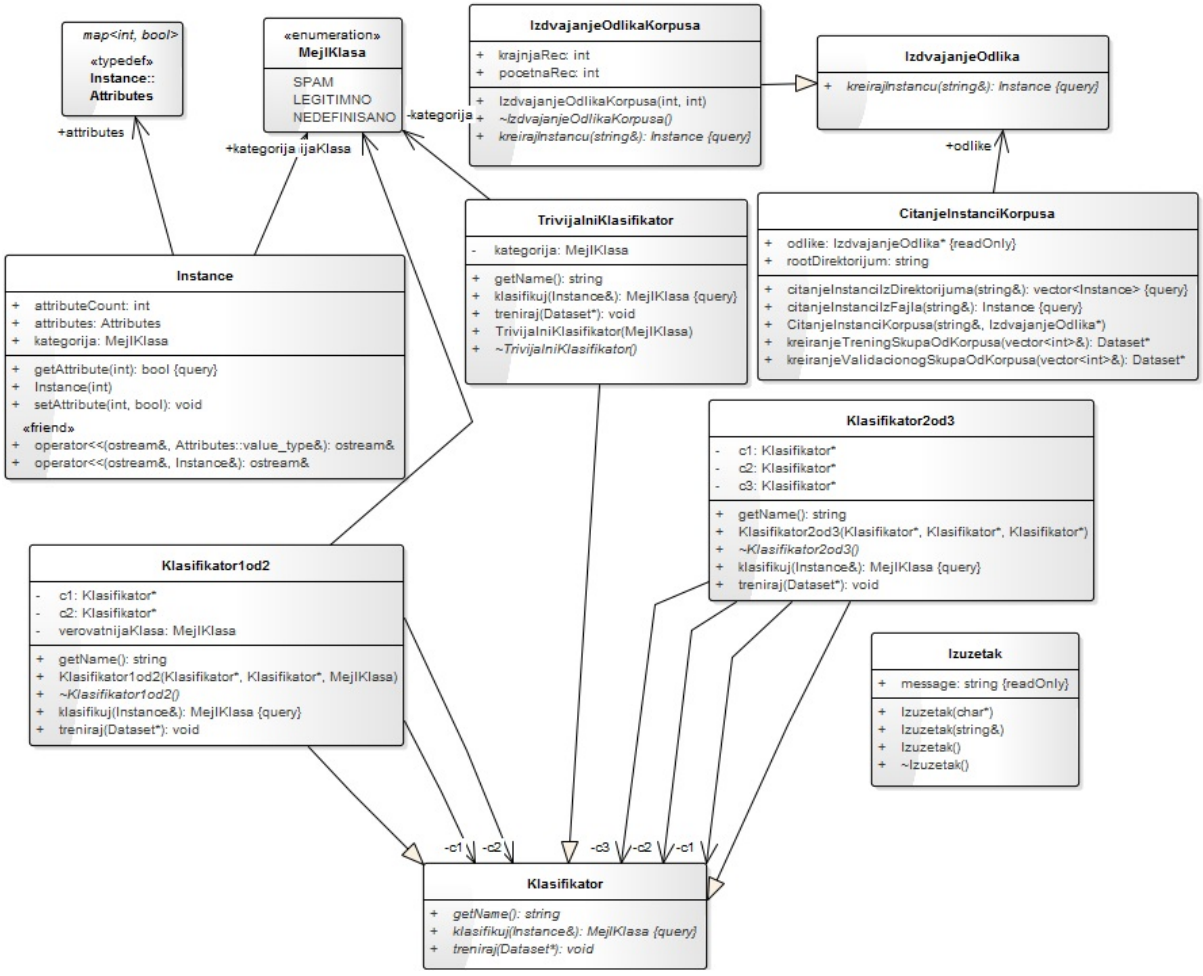
Glavni izvršni fajl će sve implementirane algoritme primeniti na poruke u korpusu i ispisaće statistiku. Podešavanje parametara programa se vrši unutar fajla `parametri.txt`. Podešavanja su zapisana u parovima `PARAMETAR VREDNOST`, pojedinačno po liniji. Linije koje počinju `#` predstavljaju komentare. Podešavanja data konfiguracionim fajlom se mogu premostiti na komandnoj liniji pri pozivanju izvršnog fajla na sledeći način `./master --PARAMETAR=VREDNOST`.

U tabeli 6 su prikazani parametri programa sa podrazumevanim vrednostima i opisom.

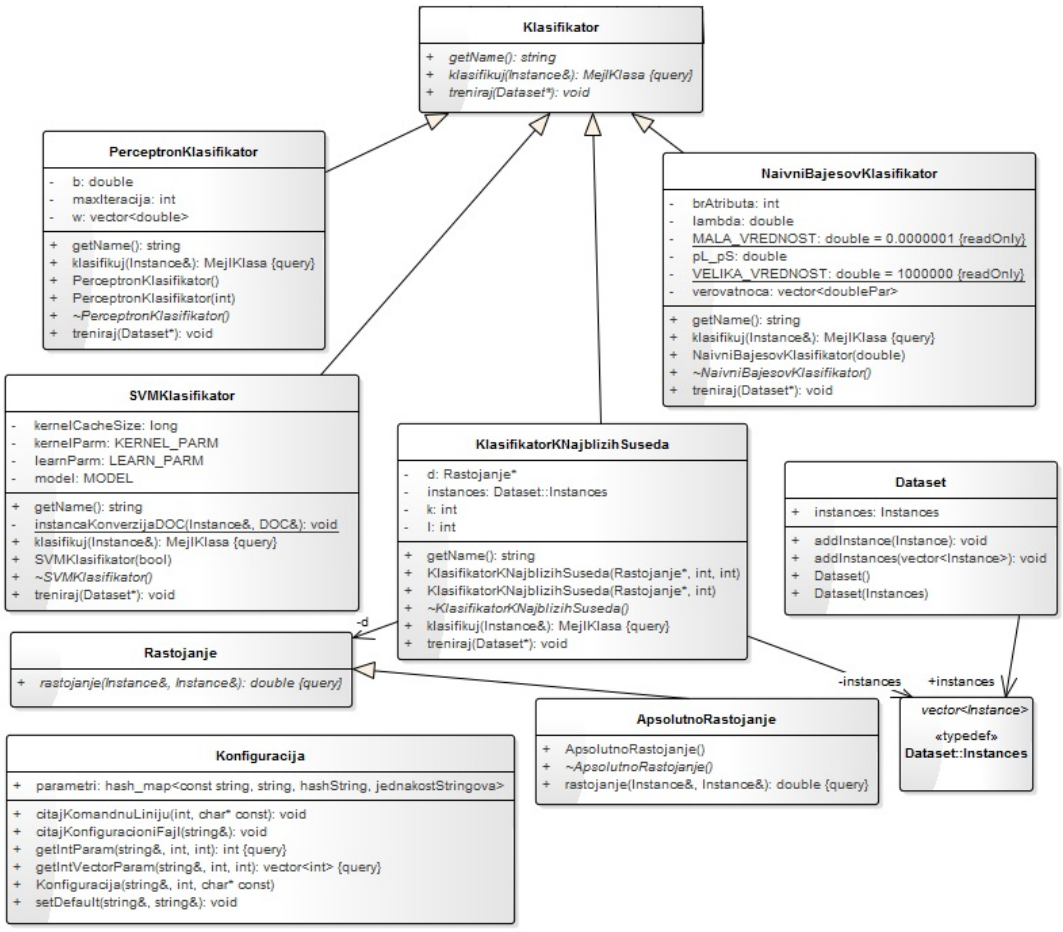
Naziv parametra	Podrazumevana vrednost	Opis
`direktorijum-korpusa`	`../korpusi/PU123ACorpora/pu_corpora_public/pu1`	*direktorijum koji sadrži željeni korpus*
`deo-korpusa`	1	*deo korpusa koji ostavljamo za validaciju*
`odlike-od`	1	*odlike koje će biti izdvojene od ove reči*
`odlike-do`	30000	*odlike koje će biti izdvojene do ove reči*
`produzeni-validacioni-skup`	0	*da li ce trening instance biti dodate skupu za validaciju zajedno sa validacionim instancama (0 - ne, 1 - da)*
`nb-lambda`	8	*lambda Naivnog Bajesovog klasifikatora*
`knn-k`	50	*k najbližih suseda*
`knn-l`	30	*ako je l ili više poruka među k najbližih suseda poruke x spam klasifikuj x kao spam*
`svm-mm`	1	*korišćenje meke margine za SVM (0 - ne, 1 - da)*

`trivijalni`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`perceptron`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`naivnibajes`	1	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`svm`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`knn`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`svm-naivnibajes`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*
`svm-naivnibajes-perceptron`	0	*klasifikator se testira (0 - ne testirati, 1 - testirati)*

Tabela 6: Parametri programa



Slika 10: Dijagram klasa 1. deo



Slika 11: Dijagram klasa 2. deo

5.1 Biblioteka SVM^{light}

Biblioteka SVM^{light} [20] [10] sadrži podršku za klasifikaciju instanci metodom podržavajućih vektora, uz niz dodatnih alata koji olakšavaju pripremu ulaznih podataka i izbor ispravnih parametara. Biblioteka je implementirana u čistom C-u, a ne u C++ tako da je bilo neophodno uraditi sledeće

```
extern "C" {
    #include "svm_common.h"
    #include "svm_learn.h"
}
```

da bi zaglavlja koja su napisana u C-u C++ kompajler mogao da linkuje što se može

[10] Preuzimanje `http://svmlight.joachims.org`.

videti i u samoj implementaciji adapter klase u dodatnom poglavlju A.11.

`svm_learn` − modul za učenje
`svm_classify` − modul za klasifikaciju

Glavne odlike programa su sledeće:

✓ brz optimizacioni algoritam

✓ rešava probleme klasifikacije i regresije

✓ izračunava XiAlpha-procenu stope greške, preciznost i odziv

✓ može trenirati SVM sa modelima cena

✓ efikasno obrađuje više hiljada podržavajućih vektora

✓ obrađuje i nekoliko desetina hiljada trening primera

5.2 Kombinovanje klasifikatora

Kombinovanjem klasifikatora dobijamo `Klasifikator1od2`, sa većom preciznošću ukoliko koristimo sledeće klasifikaciono pravilo:

Klasifikuj poruku x kao neželjenu ukoliko je jedan od dva klasifikatora klasifikuje kao neželjenu, u suprotnom klasifikuj poruku kao legitimnu.

Ovo može delovati opasno jer legitimna poruka može da se klasifikuje kao neželjena ukoliko je samo jedan klasifikator klasifikuje kao neželjenu, ali pretpostavka je da će izabrani klasifikatori imati nizak stepen lažno pozitivnih instanci. Pri evaluaciji su odabrani metod *podržavajućih vektora* i *Naivni Bajes* jer se pokazalo u prvih nekoliko korpusa da oni imaju najniži stepen lažno pozitivnih instanci.

Ekvivalentno tome je realizovano pravilo `Klasifikator2od3`, koje određuje klasu instance prema odlukama tri klasifikatora sledećim pravilom:

Instanca se klasifikuje u klasu C ukoliko se najmanje dva klasifikatora odluče za klasu C.

Pri evaluaciji je pored *metoda podržavajućih vektora* i *Naivnog Bajesa* odabran *perceptron* kao treći klasifikator zbog svoje visoke preciznosti.

6 Eksperimentalni rezultati

U ovom poglavlju su predstavljeni eksperimentalni rezultati testiranja implementiranih algoritama nad različitim korpusima elektronske pošte.

6.1 PU1 korpus

6.1.1 *Naivni Bajesov klasifikator*

Pozivanje programa za klasifikaciju poruka PU1 korpusa *Naivnim Bajesovim klasifikatorom* za $\lambda = 5$.

```
./master --direktorijum-korpusa=../korpusi/PU123ACorpora/pu_corpora_public/pu1
--naivnibajes=1 --nb-lambda=5
```

		Predviđena klasa		
		Klasa = SPAM	*Klasa = LEGITIMNO*	*Ukupno*
Stvarna klasa	*Klasa = SPAM*	61	13	$TP + FN = 74$
	Klasa = LEGITIMNO	0	35	$FP + TN = 35$
	Ukupno	$TP + FP = 61$	$FN + TN = 48$	$N = 109$

Tabela 7: Matrica konfuzije PU1 korpusa prilikom klasifikacije *Naivnim Bajesovim klasifikatorom* za $\lambda = 5$

Algoritam	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	0.553436 s	0.014049 s	0.8807	0.1192	1	0.824324	0.903704
Naivni Bajesov klasifikator ($\lambda = 50$)	0.552885 s	0.013871 s	0.8624	0.137615	1	0.802632	0.890511
Naivni Bajesov klasifikator ($\lambda = 500$)	0.550187 s	0.013875 s	0.8532	0.146789	1	0.792208	0.884058
Naivni Bajesov klasifikator ($\lambda = 999$)	0.551718 s	0.013794 s	0.8532	0.146789	1	0.792208	0.884058

Tabela 8: Testiranje performansi klasifikatora PU1 korpusa *Naivnim Bajesovim klasifikatorom*

6.1.2 | K najbližih suseda

Pozivanje programa za klasifikaciju poruka PU1 korpusa algoritmom *K najbližih suseda* za $k = 50$ i $l = 30$.

```
./master --direktorijum-korpusa=../korpusi/PU123ACorpora/pu_corpora_public/pu1
--knn=1 --knn-k=50 --knn-l=30
```

		Predviđena klasa		
		Klasa = SPAM	*Klasa = LEGITIMNO*	*Ukupno*
Stvarna klasa	*Klasa = SPAM*	58	10	$TP + FN = 68$
	Klasa = LEGITIMNO	3	38	$FP + TN = 41$
	Ukupno	$TP + FP = 61$	$FN + TN = 48$	$N = 109$

Tabela 9: Matrica konfuzije PU1 korpusa prilikom klasifikacije algoritmom *K najbližih suseda* za $k = 50$ i $l = 30$

Algoritam	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
K najbližih suseda ($k = 50$ i $l = 30$)	0.006408 s	0.461033 s	0.8807	0.1192	0.95	0.852941	0.899225
K najbližih suseda ($k = 50$ i $l = 33$)	0.005354 s	0.442313 s	0.8532	0.146789	1	0.792208	0.884058
K najbližih suseda ($k = 50$ i $l = 35$)	0.005546 s	0.443605 s	0.7982	0.2018	1	0.73494	0.847222
K najbližih suseda ($k = 30$ i $l = 20$)	0.005621 s	0.441475 s	0.8624	0.137615	0.97	0.819444	0.887218
K najbližih suseda ($k = 60$ i $l = 40$)	0.005629 s	0.443173 s	0.8257	0.174312	1	0.7625	0.865248

Tabela 10: Testiranje performansi klasifikatora PU1 korpusa algoritmom *K najbližih suseda*

6.1.3 | *Metod podržavajućih vektora*

Pozivanje programa za klasifikaciju poruka PU1 korpusa *Metodom podržavajućih vektora* bez meke margine.

```
./master --direktorijum-korpusa=../korpusi/PU123ACorpora/pu_corpora_public/pu1
--svm=1 --svm-mm=0
```

		Predviđena klasa		
		Klasa = SPAM	*Klasa = LEGITIMNO*	*Ukupno*
Stvarna klasa	*Klasa = SPAM*	59	1	$TP + FN = 60$
	Klasa = LEGITIMNO	2	47	$FP + TN = 49$
	Ukupno	$TP + FP = 61$	$FN + TN = 48$	$N = 109$

Tabela 11: Matrica konfuzije PU1 korpusa prilikom klasifikacije *Metodom podržavajućih vektora* bez meke margine

		Predviđena klasa		
		Klasa = SPAM	*Klasa = LEGITIMNO*	*Ukupno*
Stvarna klasa	*Klasa = SPAM*	61	7	$TP + FN = 68$
	Klasa = LEGITIMNO	0	41	$FP + TN = 41$
	Ukupno	$TP + FP = 61$	$FN + TN = 48$	$N = 109$

Tabela 12: Matrica konfuzije PU1 korpusa prilikom klasifikacije *Metodom podržavajućih vektora* sa mekom marginom

Algoritam	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*	*Broj podržavajućih vektora*	*Broj evaluacija kernela*
Metod podržavajućih vektora bez meke margine	0.066 s	0.023 s	0.97	0.0275	0.97	0.983	0.975	254	26797
Metod podržavajućih vektora sa mekom marginom	0.071 s	0.043 s	0.94	0.0642	1	0.897	0.945	450	21555

Tabela 13: Testiranje performansi klasifikatora PU1 korpusa *Metodom podržavajućih vektora* sa i bez meke margine

6.1.4 | *Perceptron*

Pozivanje programa za klasifikaciju poruka PU1 korpusa korišćenjem *Perceptrona*.

```
./master --direktorijum-korpusa=../korpusi/PU123ACorpora/pu_corpora_public/pu1
--perceptron=1
```

		Predviđena klasa		
		Klasa = SPAM	*Klasa = LEGITIMNO*	*Ukupno*
Stvarna klasa	*Klasa = SPAM*	59	1	$TP + FN = 60$
	Klasa = LEGITIMNO	2	47	$FP + TN = 49$
	Ukupno	$TP + FP = 61$	$FN + TN = 48$	$N = 109$

Tabela 14: Matrica konfuzije PU1 korpusa prilikom klasifikacije korišćenjem *perceptrona*

Algoritam	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Perceptron	0.0124 s	0.0002 s	0.9725	0.0275	0.97	0.9833	0.975

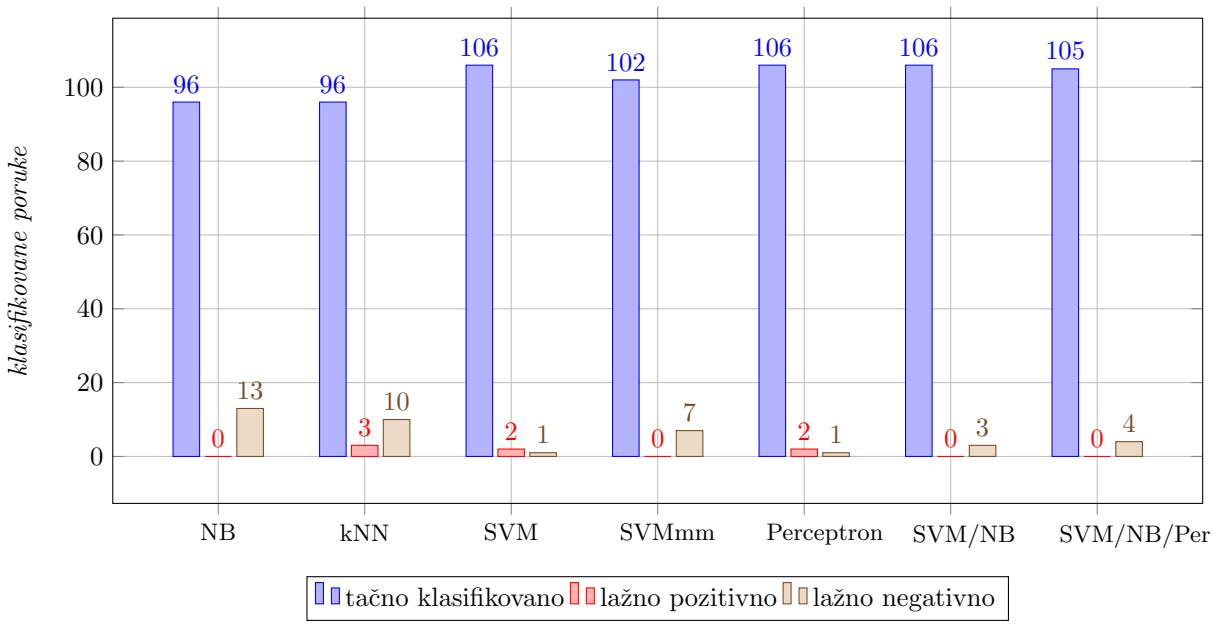
Tabela 15: Testiranje performansi klasifikatora PU1 korpusa korišćenjem *perceptrona*

6.1.5 | *Kombinovanje klasifikatora*

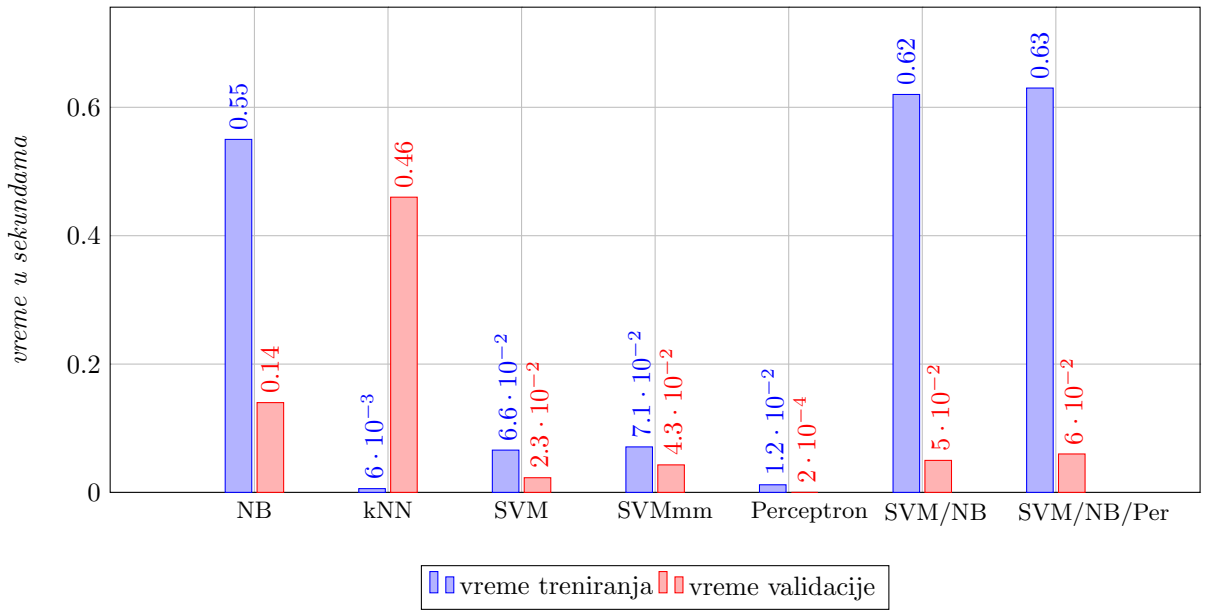
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
SVM/Naivni Bajesov klasifikator	0	3	0.62 s	0.05 s	0.97	0.0275	1	0.953125	0.976
SVM/Naivni Bajesov klasifikator/Perceptron	0	4	0.63 s	0.06 s	0.96	0.0367	1	0.9384	0.978

Tabela 16: Testiranje performansi kombinovanih klasifikatora nad PU1 korpusom

Na osnovu svih dobijenih rezultata lako se uočava da klasifikator zasnovan na *perceptronu* ima najveću tačnost i najkraće vreme treniranja i validacije, dok ga u stopu prati klasifikator zasnovan na *metodu podržavajućih vektora*.



Slika 12: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad PU1 korpusom



Slika 13: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad PU1 korpusom

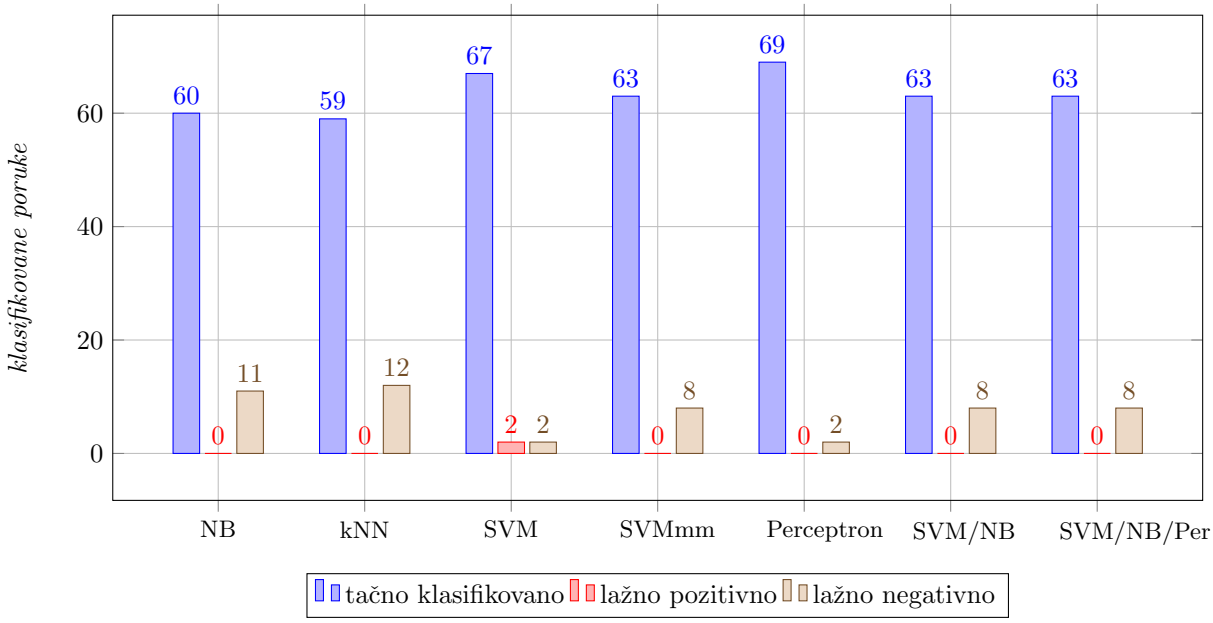
6.2 PU2 korpus

PU2 korpus je najmanji od korpusa korišćenih za testiranja implementiranih klasifikatora. Rezultati testiranja klasifikatora nad PU2 korpusom dati su u tabeli 17.

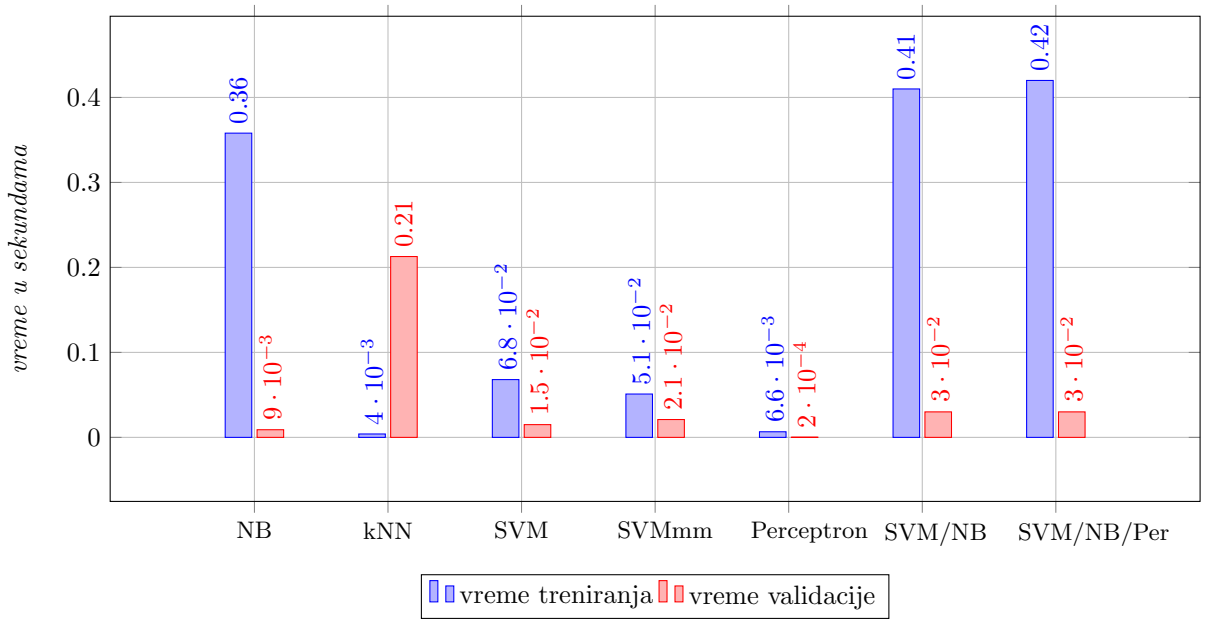
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	0	11	0.359 s	0.009 s	0.845	0.155	1	0.8382	0.912
Naivni Bajesov klasifikator ($\lambda = 50$)	0	11	0.359 s	0.009 s	0.845	0.155	1	0.8382	0.912
Naivni Bajesov klasifikator ($\lambda = 500$)	0	11	0.359 s	0.009 s	0.845	0.155	1	0.8382	0.912
Naivni Bajesov klasifikator ($\lambda = 999$)	0	11	0.359 s	0.009 s	0.845	0.155	1	0.8382	0.912
kNN ($k = 30, l = 20$)	0	12	0.004 s	0.2128 s	0.8309	0.169	1	0.8261	0.9047
kNN ($k = 50, l = 30$)	0	13	0.004 s	0.2125 s	0.8169	0.1831	1	0.8142	0.8976
kNN ($k = 20, l = 15$)	1	10	0.004 s	0.2131 s	0.8451	0.1549	0.9824	0.8485	0.9106
kNN ($k = 15, l = 10$)	2	6	0.004 s	0.2129 s	0.8873	0.1127	0.965	0.9016	0.9322
Metod podržavajućih vektora bez meke margine	2	2	0.0688 s	0.0158 s	0.9436	0.0563	0.9649	0.9649	0.9649
Metod podržavajućih vektora sa mekom marginom	0	8	0.0507 s	0.0215 s	0.8873	0.1127	1	0.8769	0.9344
Perceptron	0	2	0.0066 s	0.0002 s	0.9718	0.0281	1	0.9661	0.9827
SVM/Naivni Bajesov klasifikator	0	8	0.41 s	0.03 s	0.89	0.11	1	0.8769	0.9344
SVM/Naivni Bajesov klasifikator/Perceptron	0	8	0.42 s	0.03 s	0.89	0.11	1	0.8769	0.9344

Tabela 17: Rezultati testiranja implementiranih klasifikatora nad PU2 korpusom

Iz priloženih rezultata se jasno može videti da klasifikator zasnovan na *perceptronu* daje najbolje rezultate, kako u pogledu performansi tako i u pogledu tačnosti. Od 71 instance za validaciju samo 2 su lažno negativne (*spam klasifikovan kao legitimna poruka*), a tačnost iznosi 97.18%.



Slika 14: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad PU2 korpusom



Slika 15: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad PU2 korpusom

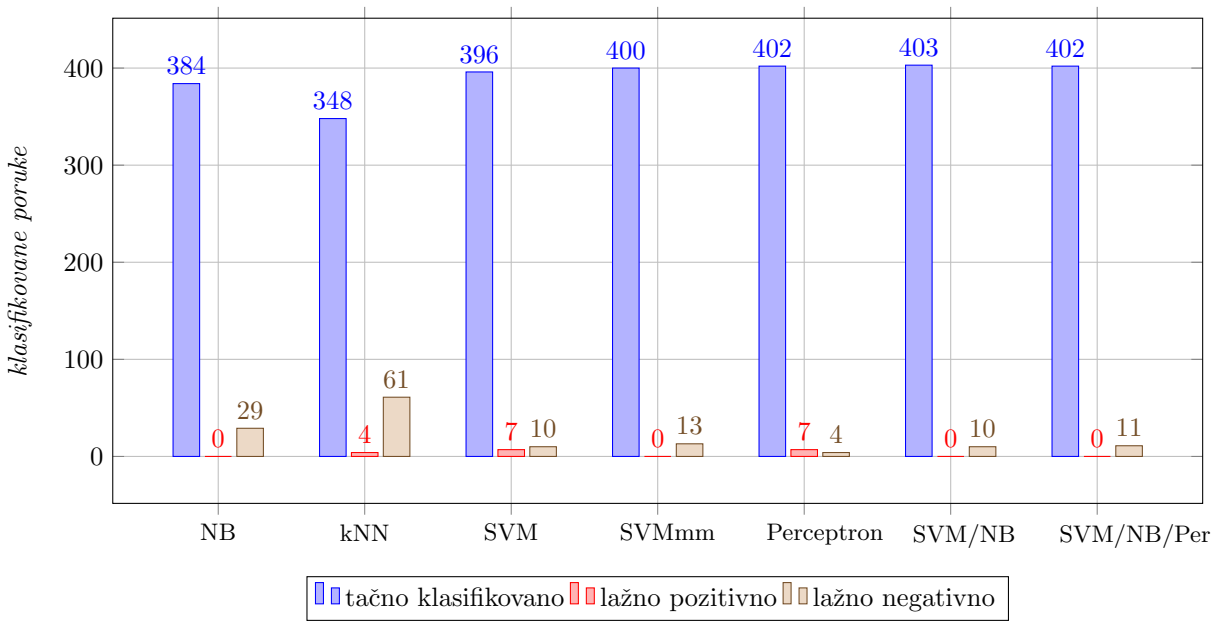
6.3 PU3 korpus

PU3 korpus je značajno veći od PU1 i PU2 korpusa i sadrži 4139 poruka od čega 1826 predstavlja spam dok su preostalih 2313 poruka legitimne. Rezultati testiranja klasifikatora nad PU3 korpusom dati su u tabeli 18.

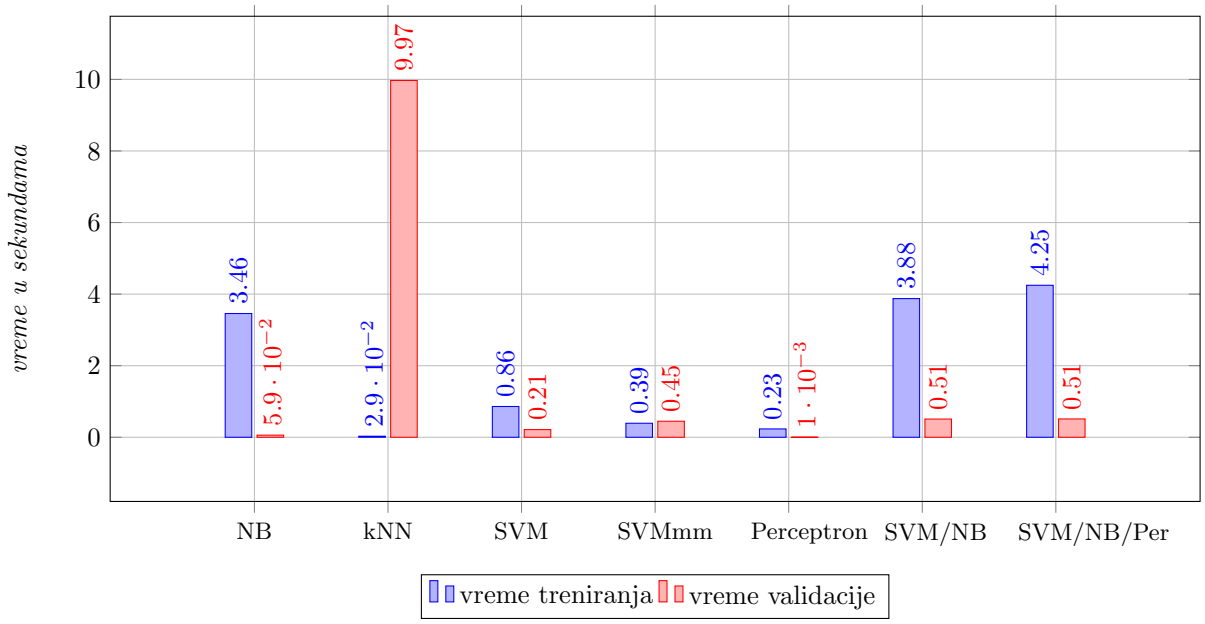
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	0	29	3.4588 s	0.05933 s	0.9297	0.0702	1	0.8884	0.941
Naivni Bajesov klasifikator ($\lambda = 50$)	0	33	3.4588 s	0.05933 s	0.9201	0.078	1	0.875	0.933
Naivni Bajesov klasifikator ($\lambda = 500$)	0	36	3.497 s	0.0596 s	0.9128	0.0875	1	0.8652	0.9277
Naivni Bajesov klasifikator ($\lambda = 999$)	0	36	3.497 s	0.0596 s	0.9128	0.0875	1	0.8652	0.9277
kNN ($k = 100, l = 65$)	4	61	0.0298 s	9.9754 s	0.8426	0.1574	0.9827	0.78819	0.8747
kNN ($k = 81, l = 51$)	6	48	0.0276 s	9.9779 s	0.8629	0.1307	0.974	0.8242	0.8928
kNN ($k = 131, l = 81$)	7	59	0.0291 s	9.971 s	0.8402	0.1598	0.9696	0.7915	0.8716
kNN ($k = 61, l = 35$)	23	23	0.0296 s	9.9091 s	0.8886	0.1114	0.9004	0.9004	0.9004
Metod podržavajućih vektora bez meke margine	7	10	0.8602 s	0.2145 s	0.9588	0.0411	0.9696	0.9572	0.9634
Metod podržavajućih vektora sa mekom marginom	0	13	0.3919 s	0.4487 s	0.9685	0.0314	1	0.9467	0.9726
Perceptron	7	4	0.2327 s	0.0014 s	0.9734	0.0266	0.9697	0.9824	0.976
SVM/Naivni Bajesov klasifikator	0	10	3.875 s	0.509 s	0.9758	0.0242	1	0.9585	0.9788
SVM/Naivni Bajesov klasifikator/Perceptron	0	11	4.2471 s	0.5125 s	0.9734	0.0266	1	0.9545	0.9767

Tabela 18: Rezultati testiranja implementiranih klasifikatora nad PU3 korpusom

Dobijeni rezultati su nešto drugačiji u odnosu na PU2 korpus ali to je i razumljivo imajući u vidu da validacioni skup sadrži 413 instanci. Klasifikator zasnovan na *perceptronu* i dalje ima veoma visoku tačnost 97.34% kao i kod PU2 korpusa ali je broj lažno pozitivnih instanci 7. Kombinovani klasifikatori postižu bolje rezultate jer ne sadrže lažno pozitivne instance uz isti stepen tačnosti ali je vreme treniranja i validacije znatno veće u odnosu na klasifikator zasnovan na *perceptronu*.



Slika 16: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad PU3 korpusom



Slika 17: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad PU3 korpusom

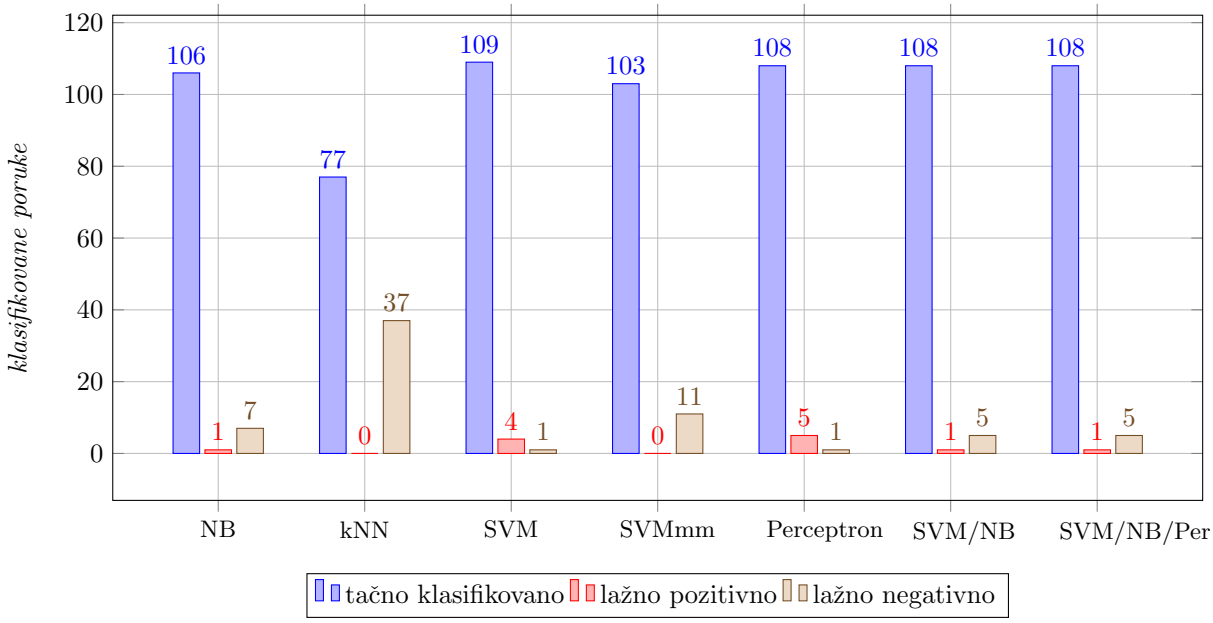
6.4 PUA korpus

PUA korpus sadrži 1142 poruka od čega je polovina legitimna dok polovina predstavlja spam. Rezultati testiranja klasifikatora nad PUA korpusom dati su u tabeli 19.

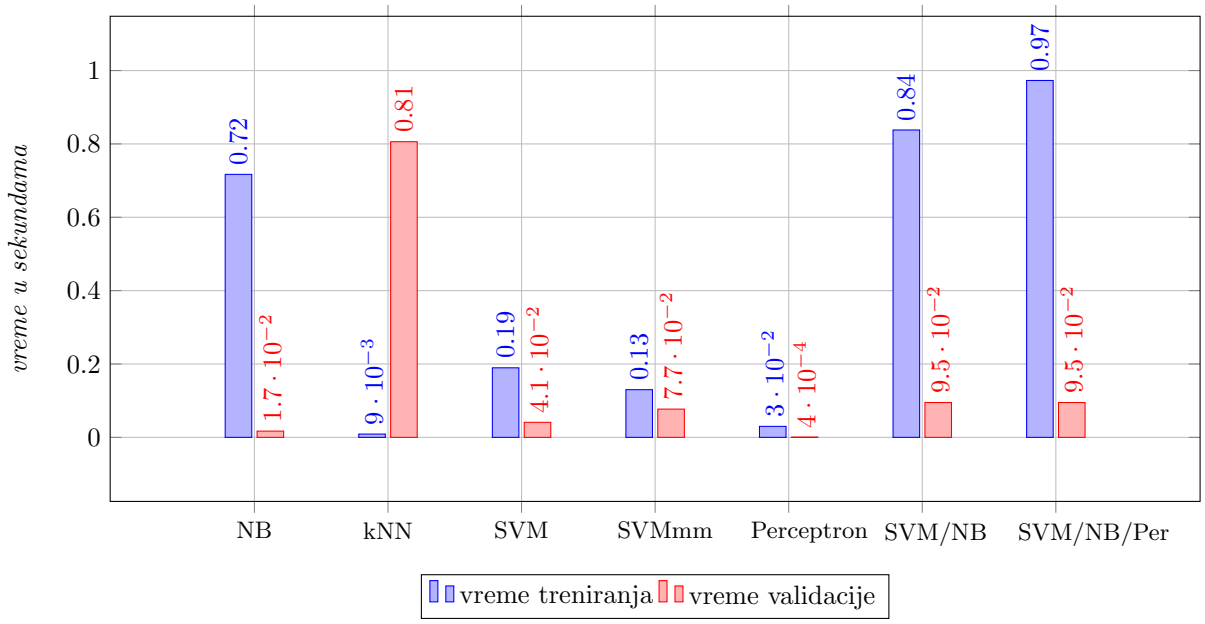
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	1	7	0.7176 s	0.0176 s	0.9298	0.0702	0.9825	0.8888	0.9333
Naivni Bajesov klasifikator ($\lambda = 50$)	1	7	0.7176 s	0.0176 s	0.9298	0.0702	0.9825	0.8888	0.9333
Naivni Bajesov klasifikator ($\lambda = 500$)	1	7	0.7176 s	0.0176 s	0.9298	0.0702	0.9825	0.8888	0.9333
Naivni Bajesov klasifikator ($\lambda = 999$)	1	7	0.7176 s	0.0176 s	0.9298	0.0702	0.9825	0.8888	0.9333
kNN ($k = 61, l = 56$)	0	37	0.0093 s	0.8046 s	0.6754	0.3245	1	0.6064	0.755
kNN ($k = 51, l = 30$)	46	0	0.0097 s	0.8059 s	0.5965	0.4035	0.1930	1	0.3235
kNN ($k = 41, l = 25$)	43	0	0.0098 s	0.8046 s	0.6228	0.3772	0.2456	1	0.3944
kNN ($k = 31, l = 20$)	40	0	0.0094 s	0.8058 s	0.6491	0.3509	0.2982	1	0.4594
Metod podržavajućih vektora bez meke margine	4	1	0.1896 s	0.0405 s	0.9561	0.0438	0.9298	0.9814	0.955
Metod podržavajućih vektora sa mekom marginom	0	11	0.1301 s	0.0772 s	0.9035	0.0965	1	0.8382	0.912
Perceptron	5	1	0.0308 s	0.0004 s	0.9473	0.0526	0.9123	0.9813	0.9454
SVM/Naivni Bajesov klasifikator	1	5	0.8388 s	0.0946 s	0.9474	0.0526	0.9824	0.918	0.949
SVM/Naivni Bajesov klasifikator/Perceptron	1	5	0.8732 s	0.0951 s	0.9474	0.0526	0.9824	0.918	0.949

Tabela 19: Rezultati testiranja implementiranih klasifikatora nad PUA korpusom

Klasifikator *k najbližih suseda* daje iznenađujuće loše rezultate. Lažno pozitivne instance su eliminisane tek kada je parametar l vrlo blizak parametru k što dovodi do zaključka da su instance korpusa nepovoljno rasprostranjene u prostoru za upotrebu ovog algoritma dok su se ostali algoritmi dosta dobro pokazali sa tačnošću većom od 90%. Najbolje se pokazao *metod podržavajućih vektora sa mekom marginom* koji nema lažno pozitivnih instanci pri čemu je vreme treniranja i validacije nisko.



Slika 18: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad PUA korpusom



Slika 19: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad PUA korpusom

6.5 Ling spam korpus

Ling spam korpus sastoji se od 481 spam poruke kombinovanih sa 2412 legitimnih poruka. Za razliku od prethodnih korpusa poruke nisu kodirane brojevima i zato je neophodno modifikovati funkciju `IzdvajanjeOdlikaKorpusa` u fajlu `IzdvajanjeOdlika.h` kao i funkciju `citanjeInstanciIzFajla` zbog određivanja kategorija instanci jer su nazivi fajlova drugačiju u odnosu na PU korpuse. Rezultati testiranja klasifikatora nad *Ling spam korpusom* dati su u tabeli 20.

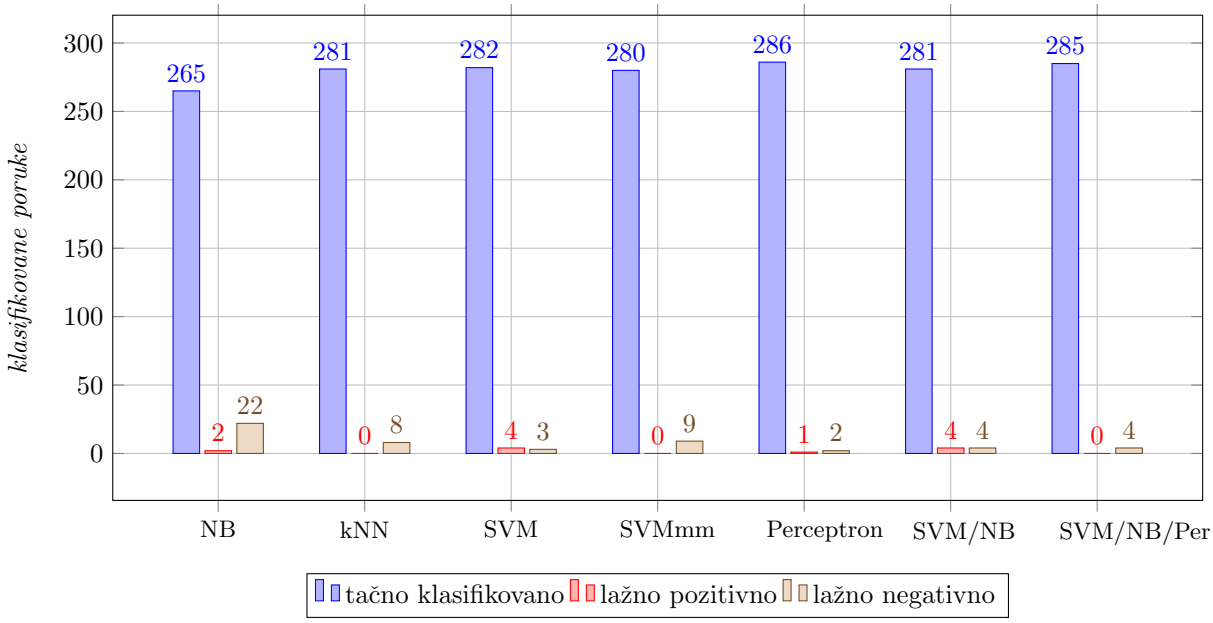
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	4	21	4.9359 s	0.0947 s	0.9135	0.0865	0.983	0.9186	0.9499
Naivni Bajesov klasifikator ($\lambda = 50$)	2	22	4.8562 s	0.0955 s	0.9169	0.0830	0.9917	0.9157	0.9522
Naivni Bajesov klasifikator ($\lambda = 500$)	2	22	4.8562 s	0.0955 s	0.9169	0.0830	0.9917	0.9157	0.9522
Naivni Bajesov klasifikator ($\lambda = 999$)	2	22	4.8562 s	0.0955 s	0.9169	0.0830	0.9917	0.9157	0.9522
kNN ($k = 151, l = 57$)	0	8	0.0164 s	2.231 s	0.9723	0.0277	1	0.9678	0.9836
kNN ($k = 101, l = 43$)	0	14	0.0155 s	2.4242 s	0.9515	0.0484	1	0.9451	0.9717
kNN ($k = 71, l = 31$)	0	13	0.0148 s	2.2015 s	0.955	0.045	1	0.9488	0.9737
kNN ($k = 51, l = 23$)	0	14	0.0155 s	2.4242 s	0.9515	0.0484	1	0.9451	0.9717
Metod podržavajućih vektora bez meke margine	4	3	0.1047 s	0.0466 s	0.9757	0.024	0.9834	0.9875	0.9854
Metod podržavajućih vektora sa mekom marginom	0	9	0.1067 s	0.083 s	0.9688	0.0311	1	0.964	0.982
Perceptron	1	2	0.037 s	0.0004 s	0.9896	0.01	0.9958	0.9917	0.9937
SVM/Naivni Bajesov klasifikator	4	4	4.91 s	0.17 s	0.9723	0.027	0.9834	0.9834	0.9834
SVM/Naivni Bajesov klasifikator/Perceptron	0	4	4.875 s	0.174 s	0.986	0.0138	1	0.984	0.992

Tabela 20: Rezultati testiranja implementiranih klasifikatora nad *Ling spam korpusom*

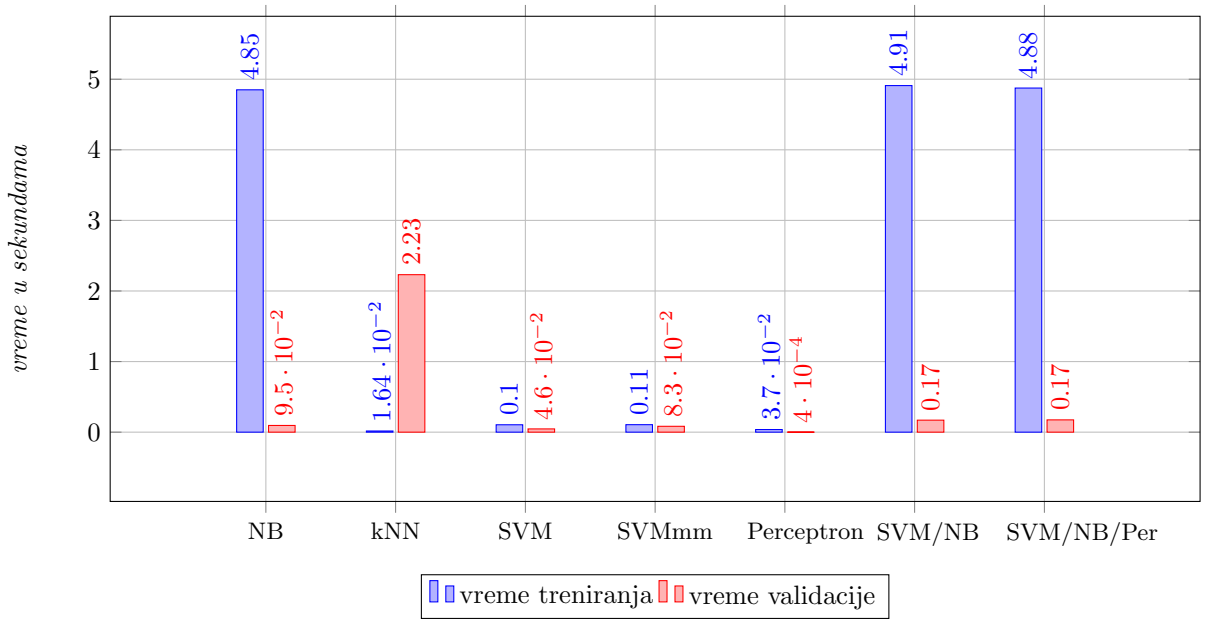
Veličina validacionog skupa je 289 instanci. Prilikom testiranja klasifikatora korišćen je folder *lemm_stop* u kojem su nad porukama izvršene lematizacija[11] i stop lista reči[12].

[11] Predstavlja lingvistički proces u kome se više različitih reči mogu grupisati u jednu zbog njihove lakše analize. Npr. engleska reč 'walk' se može javiti u oblicima 'walk', 'walked', 'walks', 'walking'. Osnovna forma 'walk' koja se javlja u rečniku je lema te reči.

[12] Reči koje su filtrirane pre ili posle procesiranja prirodnih jezika. Stop reči najčešće predstavljaju najkorišćenije reči u jeziku kao sto su engleske reči *the, is, at, which, on* itd.



Slika 20: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad Ling spam korpusom



Slika 21: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad Ling spam korpusom

6.6 Enron spam korpus

Enron spam korpus je naslednik Ling Spam i PU korpusa koji sadrži hronološki podeljene e-mail poruke koje su primali šest Enronovih zaposlenih kombinovanih sa spamom iz različitih izvora. Korpus se sastoji iz paketa od šest skupova podataka za trening i testiranje u kojima su poruke hronološki sortirane.

6.6.1 *enron1*

```
Legitimna posta
----------
- Vlasnik: farmer-d
- Ukupan broj: 3672 mejlova
- Datum prvog mejla: 1999-12-10
- Datum poslednjeg mejla: 2002-01-11
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

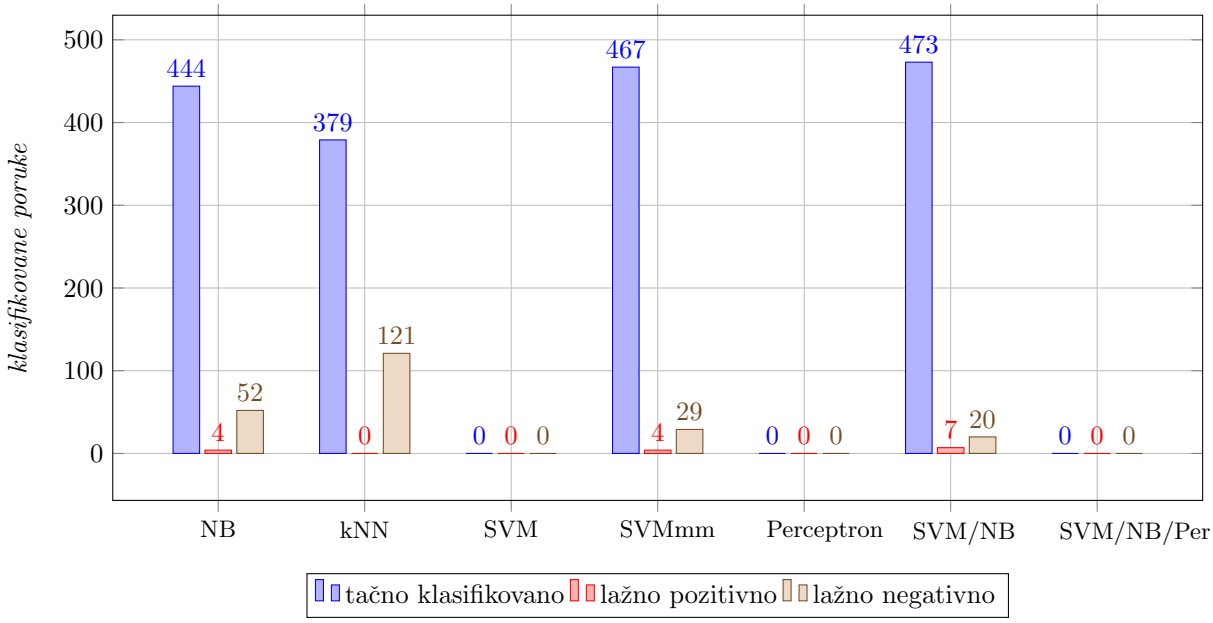
```
Spam
----------
- Vlasnik: GP
- Ukupan broj: 1500 mejlova
- Datum prvog mejla: 2003-12-18
- Datum poslednjeg mejla: 2005-09-06
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

`Spam:Legitimne poruke odnos` $\approx 1 : 2.5$

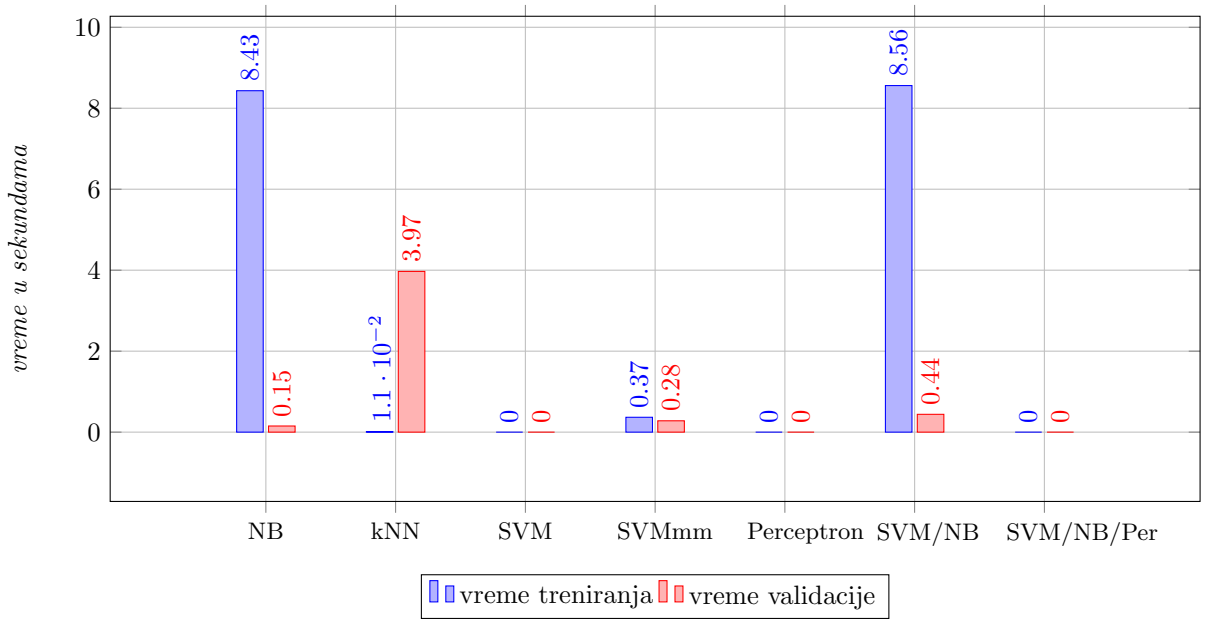
`Ukupan broj mejlova (legitimni + spam): 5172`

Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	5	50	8.2027 s	0.1517 s	0.89	0.11	0.9864	0.8792	0.9297
Naivni Bajesov klasifikator ($\lambda = 50$)	4	52	8.4335 s	0.1514 s	0.888	0.112	0.9891	0.8753	0.9287
kNN ($k = 301, l = 150$)	3	111	0.011 s	3.97 s	0.772	0.228	0.9918	0.7673	0.8652
kNN ($k = 301, l = 156$)	0	121	0.0118 s	3.9652 s	0.758	0.242	1	0.7531	0.8591
Metod podržavajućih vektora bez meke margine	-	-	> 10 min	*Najverovatnije podaci nisu linearno razdvojivi!*					
Metod podržavajućih vektora sa mekom marginom	4	29	0.367 s	0.2805 s	0.934	0.066	0.9892	0.9264	0.9567
Perceptron	*Perceptron ne konvergira. Najverovatnije podaci nisu linearno razdvojivi!*								
SVM/Naivni Bajesov klasifikator	7	20	8.5609 s	0.4428 s	0.946	0.054	0.981	0.9476	0.964
SVM/Naivni Bajesov klasifikator/Perceptron	*Perceptron ne konvergira. Najverovatnije podaci nisu linearno razdvojivi!*								

Tabela 21: Rezultati testiranja implementiranih klasifikatora nad *enron1 korpusom*



Slika 22: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron1 korpusom



Slika 23: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron1 korpusom

6.6.2 | *enron2*

```
Legitimna posta
----------
- Vlasnik: kaminski-v
- Ukupan broj: 4361  mejlova
- Datum prvog mejla: 1999-12-10
- Datum poslednjeg mejla: 2001-05-22
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

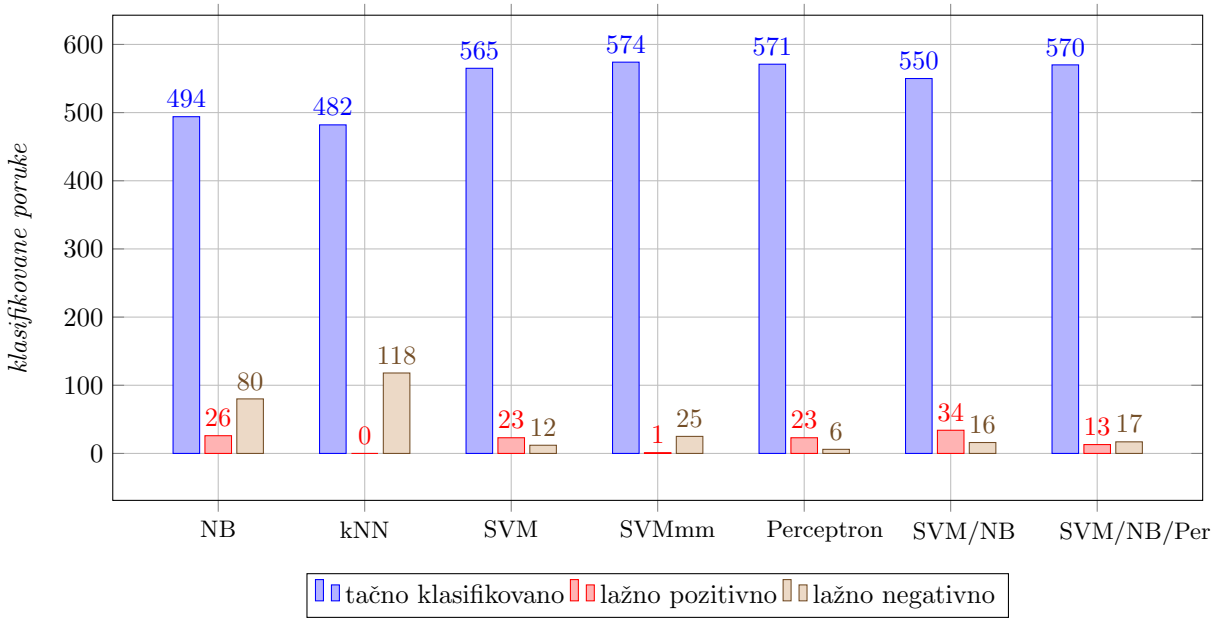
```
Spam
----------
- Vlasnik: SpamAssassin + HoneyPot
- Ukupan broj: 1496  mejlova
- Datum prvog mejla: 2001-05-25
- Datum poslednjeg mejla: 2005-07-22
- Brisanje slicnih: Da
- Enkodiranje: Ne
```

Spam:Legitimne poruke odnos $\approx 1:3$

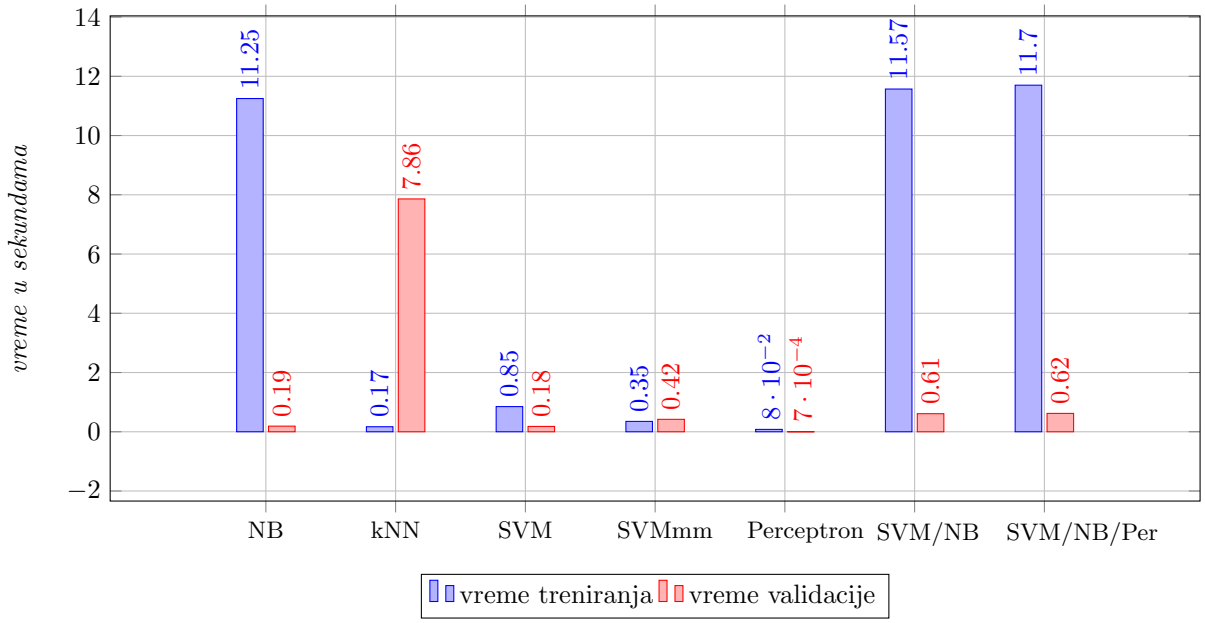
Ukupan broj mejlova (legitimni + spam): 5857

Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	33	70	11.2648 s	0.1953 s	0.8283	0.1716	0.9271	0.8571	0.8907
Naivni Bajesov klasifikator ($\lambda = 50$)	28	75	11.2931 s	0.1959 s	0.8233	0.1716	0.9382	0.85	0.8919
Naivni Bajesov klasifikator ($\lambda = 500$)	27	70	11.2295 s	0.1948 s	0.8216	0.1783	0.9404	0.8419	0.8884
Naivni Bajesov klasifikator ($\lambda = 999$)	26	80	11.2553 s	0.1954 s	0.8233	0.1766	0.9426	0.8422	0.8896
kNN ($k = 201, l = 90$)	0	124	0.0175 s	7.8313 s	0.7933	0.2033	1	0.7851	0.8796
kNN ($k = 201, l = 100$)	0	135	0.0165 s	7.8189 s	0.775	0.225	1	0.7704	0.8703
kNN ($k = 201, l = 80$)	4	93	0.0173 s	7.8501 s	0.8383	0.1616	0.9912	0.8284	0.9025
kNN ($k = 301, l = 130$)	0	118	0.0169 s	7.8623 s	0.8033	0.1967	1	0.7933	0.8847
Metod podržavajućih vektora bez meke margine	23	12	0.8559 s	0.1817 s	0.9492	0.0583	0.9492	0.9728	0.961
Metod podržavajućih vektora sa mekom marginom	1	25	0.3545 s	0.4186 s	0.9567	0.0433	0.9978	0.9476	0.972
Perceptron	23	6	0.0808 s	0.0007 s	0.9516	0.0483	0.9492	0.9862	0.9674
SVM/Naivni Bajesov klasifikator	34	16	11.5764 s	0.6133 s	0.9167	0.0833	0.9249	0.9632	0.9437
SVM/Naivni Bajesov klasifikator/Perceptron	13	17	11.7054 s	0.6169 s	0.95	0.05	0.9713	0.9628	0.967

Tabela 22: Rezultati testiranja implementiranih klasifikatora nad *enron2 korpusom*



Slika 24: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron2 korpusom



Slika 25: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron2 korpusom

6.6.3 | *enron3*

```
Legitimna posta
----------
- Vlasnik: kitchen-l
- Ukupan broj: 4012 mejlova
- Datum prvog mejla: 2001-02-07
- Datum poslednjeg mejla: 2002-02-06
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

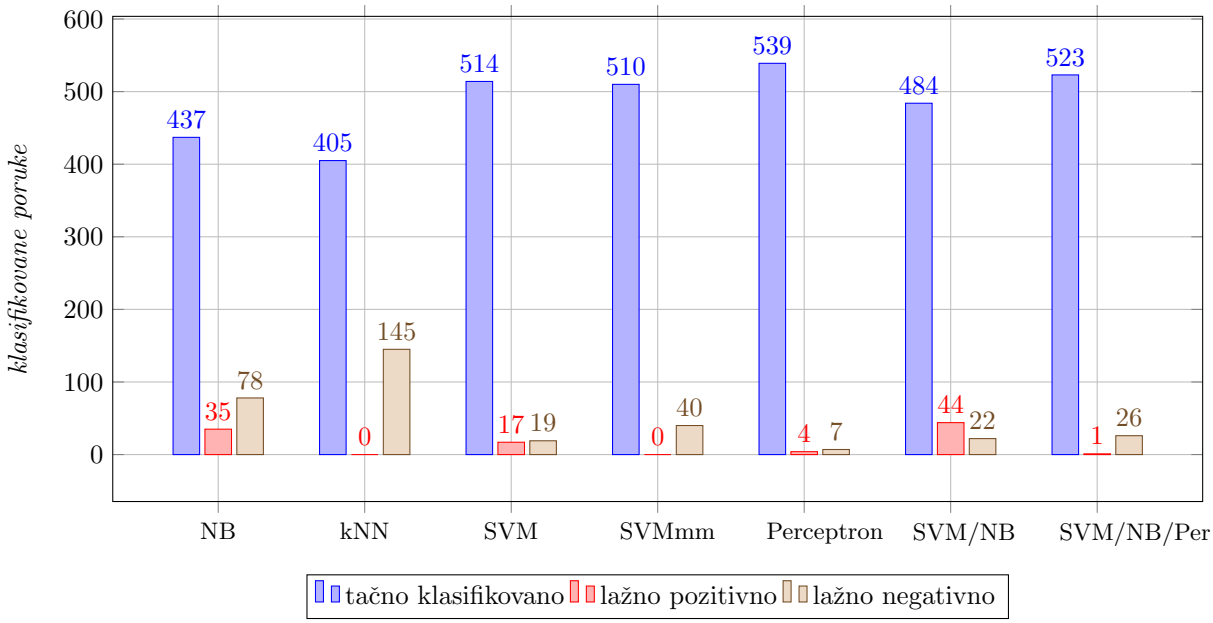
```
Spam
----------
- Vlasnik: BG
- Ukupan broj: 1500 mejlova
- Datum prvog mejla: 2004-08-01
- Datum poslednjeg mejla: 2005-07-31
- Brisanje slicnih: Da
- Enkodiranje: Ne
```

`Spam:Legitimne poruke odnos` $\approx 1:3$

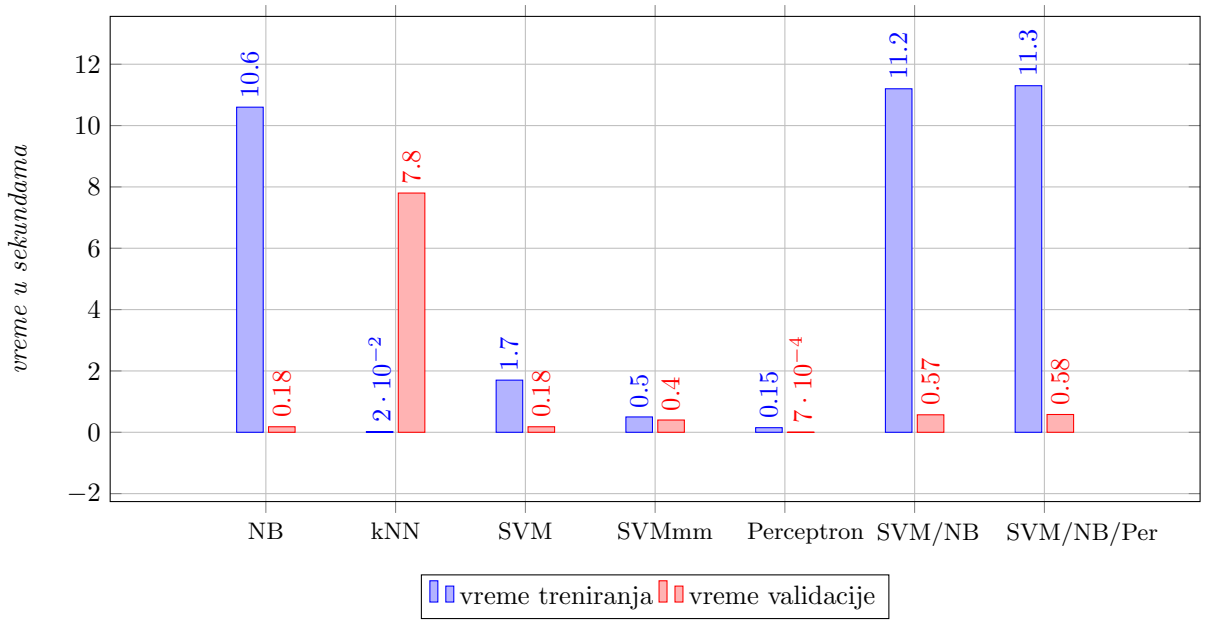
`Ukupan broj mejlova (legitimni + spam): 5512`

Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	45	75	11.1431 s	0.1793 s	0.7818	0.2182	0.8866	0.8243	0.8544
Naivni Bajesov klasifikator ($\lambda = 50$)	40	75	10.7045 s	0.1806 s	0.7909	0.2091	0.8992	0.8263	0.8613
Naivni Bajesov klasifikator ($\lambda = 500$)	36	76	10.6999 s	0.1805 s	0.7963	0.2036	0.9093	0.8261	0.8657
Naivni Bajesov klasifikator ($\lambda = 999$)	35	78	10.599 s	0.1805 s	0.7945	0.2054	0.9118	0.8227	0.865
kNN ($k = 101, l = 75$)	0	153	0.0193 s	7.8384 s	0.7364	0.2636	1	0.7324	0.8456
kNN ($k = 101, l = 50$)	0	153	0.0201 s	7.8384 s	0.7364	0.2636	1	0.7324	0.8456
kNN ($k = 301, l = 200$)	0	153	0.0203 s	7.8384 s	0.7364	0.2636	1	0.7324	0.8456
kNN ($k = 201, l = 60$)	0	145	0.01949 s	7.8384 s	0.7364	0.2636	1	0.7324	0.8456
Metod podržavajućih vektora bez meke margine	17	19	1.7605 s	0.1861 s	0.9345	0.0654	0.9572	0.9524	0.9548
Metod podržavajućih vektora sa mekom marginom	0	40	0.5046 s	0.3903 s	0.9272	0.0727	1	0.9085	0.952
Perceptron	4	7	0.1505 s	0.0007 s	0.98	0.02	0.9899	0.9825	0.9862
SVM/Naivni Bajesov klasifikator	44	22	11.2177 s	0.5693 s	0.88	0.12	0.8892	0.9413	0.9145
SVM/Naivni Bajesov klasifikator/Perceptron	1	26	11.2792 s	0.5737 s	0.951	0.049	0.9975	0.9384	0.967

Tabela 23: Rezultati testiranja implementiranih klasifikatora nad *enron3 korpusom*



Slika 26: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron3 korpusom



Slika 27: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron3 korpusom

6.6.4 | *enron4*

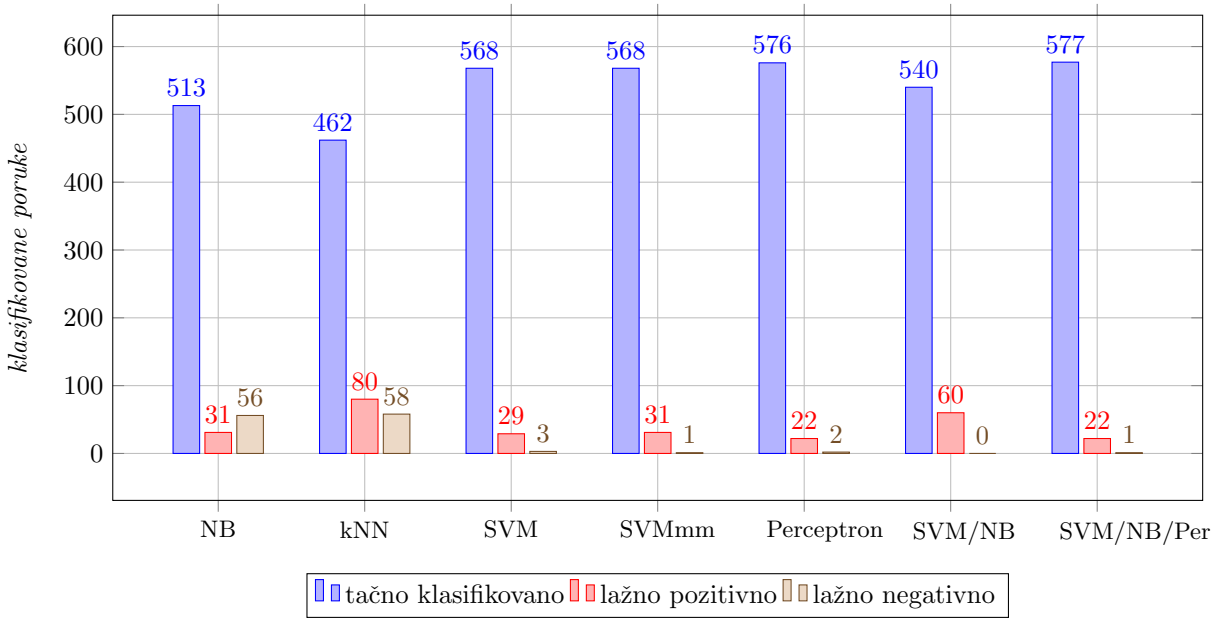
```
Legitimna posta
----------
- Vlasnik: williams-w3
- Ukupan broj: 1500 mejlova
- Datum prvog mejla: 2001-04-02
- Datum poslednjeg mejla: 2002-02-07
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

```
Spam
----------
- Vlasnik: GP
- Ukupan broj: 4500 mejlova
- Datum prvog mejla: 2003-12-18
- Datum poslednjeg mejla: 2005-09-06
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

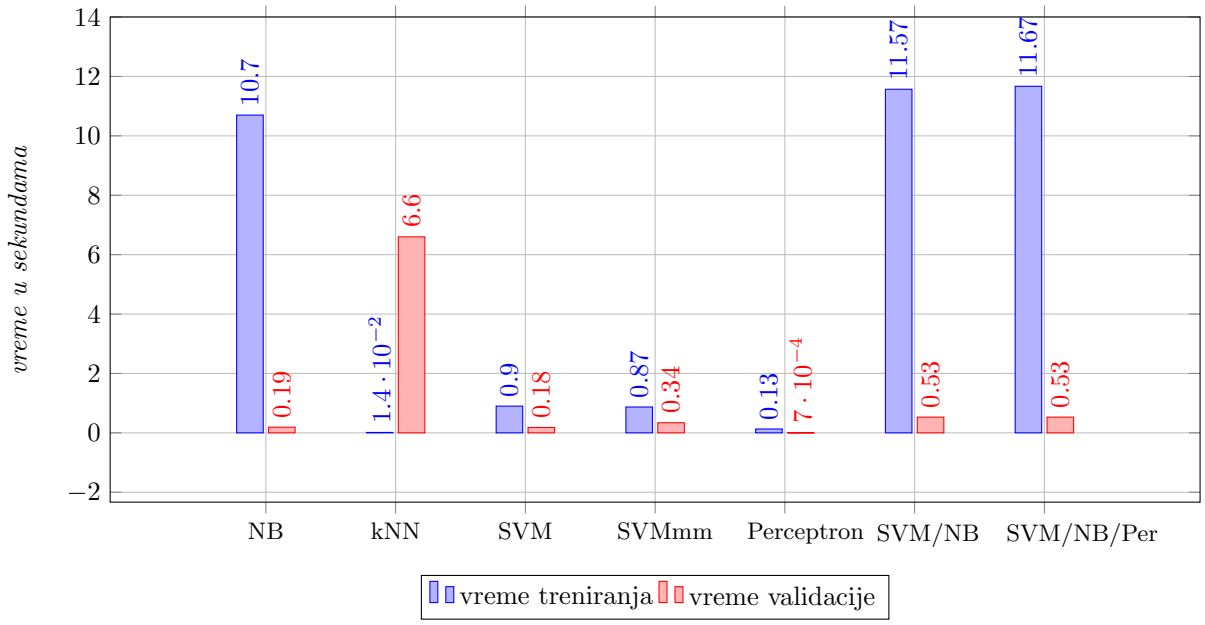
```
Spam:Legitimne poruke odnos = 3:1
Ukupan broj mejlova (legitimni + spam): 6000
```

Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	34	50	10.9389 s	0.1904 s	0.86	0.14	0.7638	0.6875	0.7237
Naivni Bajesov klasifikator ($\lambda = 50$)	33	52	10.7158 s	0.1901 s	0.8583	0.1417	0.7708	0.681	0.7231
Naivni Bajesov klasifikator ($\lambda = 500$)	31	56	10.8306 s	0.2174 s	0.855	0.145	0.7847	0.6686	0.722
Naivni Bajesov klasifikator ($\lambda = 999$)	31	56	10.6929 s	0.1911 s	0.855	0.145	0.7847	0.6686	0.722
kNN ($k = 201, l = 100$)	130	4	0.0174 s	6.6853 s	0.7766	0.2233	0.0972	0.7777	0.1728
kNN ($k = 301, l = 200$)	115	15	0.0142 s	6.6884 s	0.7833	0.2167	0.2014	0.6591	0.3085
kNN ($k = 301, l = 240$)	80	58	0.0148 s	6.6733 s	0.77	0.23	0.993	0.2531	0.4034
kNN ($k = 301, l = 295$)	1	422	0.0142 s	6.6506 s	0.295	0.705	0.444	0.5246	0.4812
Metod podržavajućih vektora bez meke margine	29	3	0.9095 s	0.1769 s	0.9467	0.0533	0.7986	0.9745	0.8778
Metod podržavajućih vektora sa mekom marginom	31	1	0.8711 s	0.3394 s	0.9467	0.0533	0.7847	0.9912	0.876
Perceptron	22	2	0.1259 s	0.0007 s	0.96	0.04	0.8472	0.9839	0.9104
SVM/Naivni Bajesov klasifikator	60	0	11.5695 s	0.5355 s	0.9	0.1	0.5833	1	0.7368
SVM/Naivni Bajesov klasifikator/Perceptron	22	1	11.6689 s	0.5311 s	0.9617	0.0383	0.8472	0.9918	0.9138

Tabela 24: Rezultati testiranja implementiranih klasifikatora nad *enron4 korpusom*



Slika 28: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron4 korpusom



Slika 29: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron4 korpusom

6.6.5 | *enron5*

```
Legitimna posta
----------
- Vlasnik: beck-s
- Ukupan broj: 1500 mejlova
- Datum prvog mejla: 2000-01-17
- Datum poslednjeg mejla: 2001-05-24
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

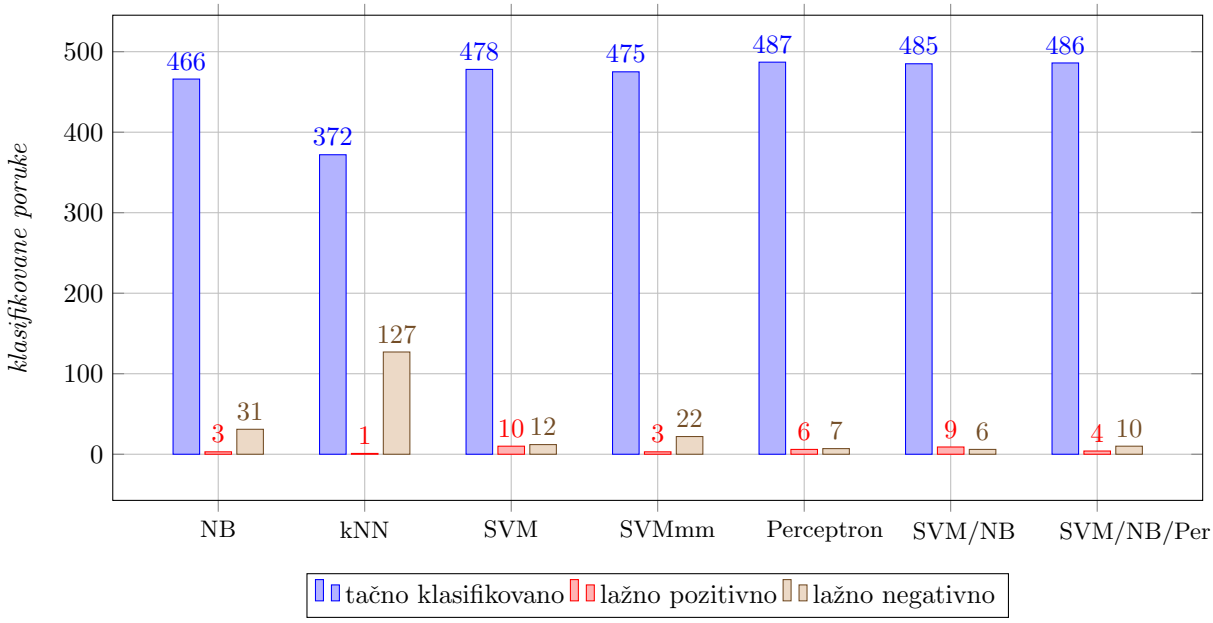
```
Spam
----------
- Vlasnik: SpamAssassin + HoneyPot
- Ukupan broj: 3675 mejlova
- Datum prvog mejla: 2001-05-25
- Datum poslednjeg mejla: 2005-07-22
- Brisanje slicnih: Da
- Enkodiranje: Ne
```

`Spam:Legitimne poruke odnos` $\approx 3:1$

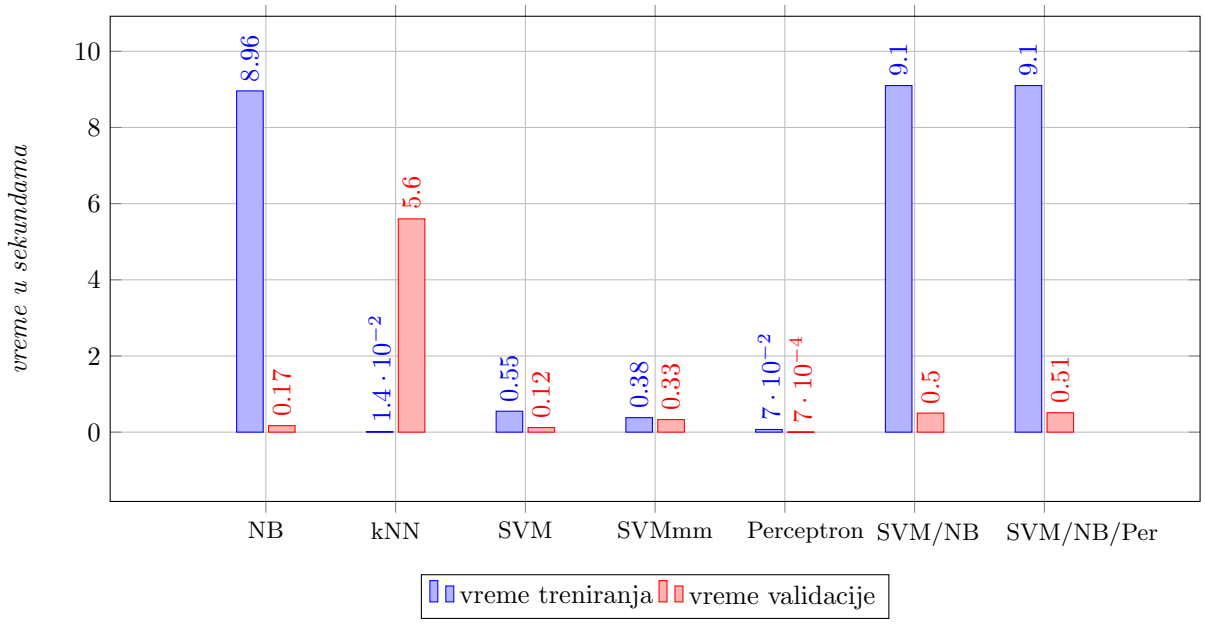
`Ukupan broj mejlova (legitimni + spam): 5175`

Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 0$)	6	23	8.975 s	0.1716 s	0.942	0.058	0.9589	0.8589	0.9061
Naivni Bajesov klasifikator ($\lambda = 50$)	3	31	9.091 s	0.1768 s	0.932	0.068	0.9794	0.8218	0.8937
Naivni Bajesov klasifikator ($\lambda = 500$)	3	40	8.9127 s	0.1758 s	0.914	0.086	0.9794	0.7814	0.8693
Naivni Bajesov klasifikator ($\lambda = 999$)	3	40	8.9127 s	0.1758 s	0.914	0.086	0.9794	0.7814	0.8693
kNN ($k = 201, l = 100$)	18	57	0.0142 s	5.679 s	0.85	0.15	0.8767	0.6919	0.7734
kNN ($k = 201, l = 120$)	1	145	0.0144 s	5.6944 s	0.708	0.292	0.9931	0.5	0.6651
kNN ($k = 301, l = 165$)	11	59	0.0143 s	5.6711 s	0.86	0.14	0.9246	0.6959	0.7941
kNN ($k = 301, l = 180$)	1	127	0.0142 s	5.6558 s	0.774	0.256	0.9931	0.5331	0.6938
Metod podržavajućih vektora bez meke margine	10	12	0.5515 s	0.1221 s	0.956	0.044	0.9315	0.9189	0.9252
Metod podržavajućih vektora sa mekom marginom	3	22	0.3831 s	0.3383 s	0.95	0.05	0.9794	0.8667	0.9196
Perceptron	6	7	0.0712 s	0.0007 s	0.974	0.026	0.9589	0.9524	0.9556
SVM/Naivni Bajesov klasifikator	9	6	9.059 s	0.5068 s	0.97	0.03	0.9383	0.958	0.9481
SVM/Naivni Bajesov klasifikator/Perceptron	4	10	9.0774 s	0.5088 s	0.972	0.028	0.9726	0.9342	0.953

Tabela 25: Rezultati testiranja implementiranih klasifikatora nad *enron5 korpusom*



Slika 30: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron5 korpusom



Slika 31: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron5 korpusom

6.6.6 | *enron6*

```
Legitimna posta
----------
- Vlasnik: lokay-m
- Ukupan broj: 1500 mejlova
- Datum prvog mejla: 2000-06-06
- Datum poslednjeg mejla: 2002-03-25
- Brisanje slicnih: Ne
- Enkodiranje: Ne
```

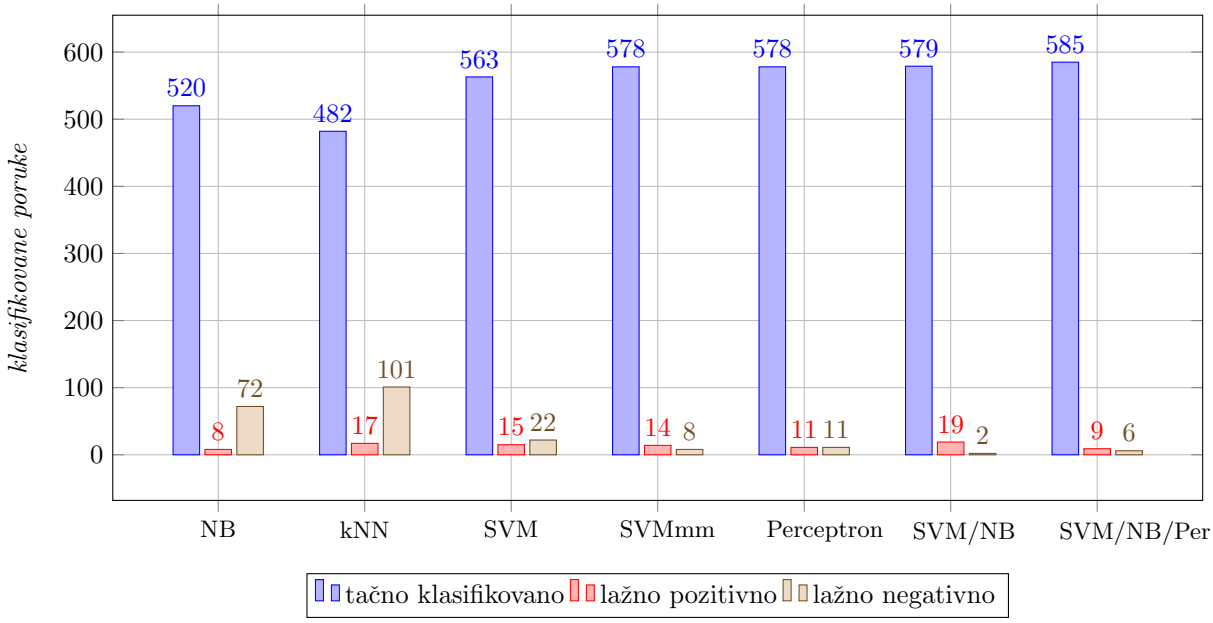
```
Spam
----------
- Vlasnik: BG
- Ukupan broj: 4500 mejlova
- Datum prvog mejla: 2004-08-01
- Datum poslednjeg mejla: 2005-07-31
- Brisanje slicnih: Da
- Enkodiranje: Ne
```

```
Spam:Legitimne poruke odnos = 3:1
Ukupan broj mejlova (legitimni + spam): 6000
```

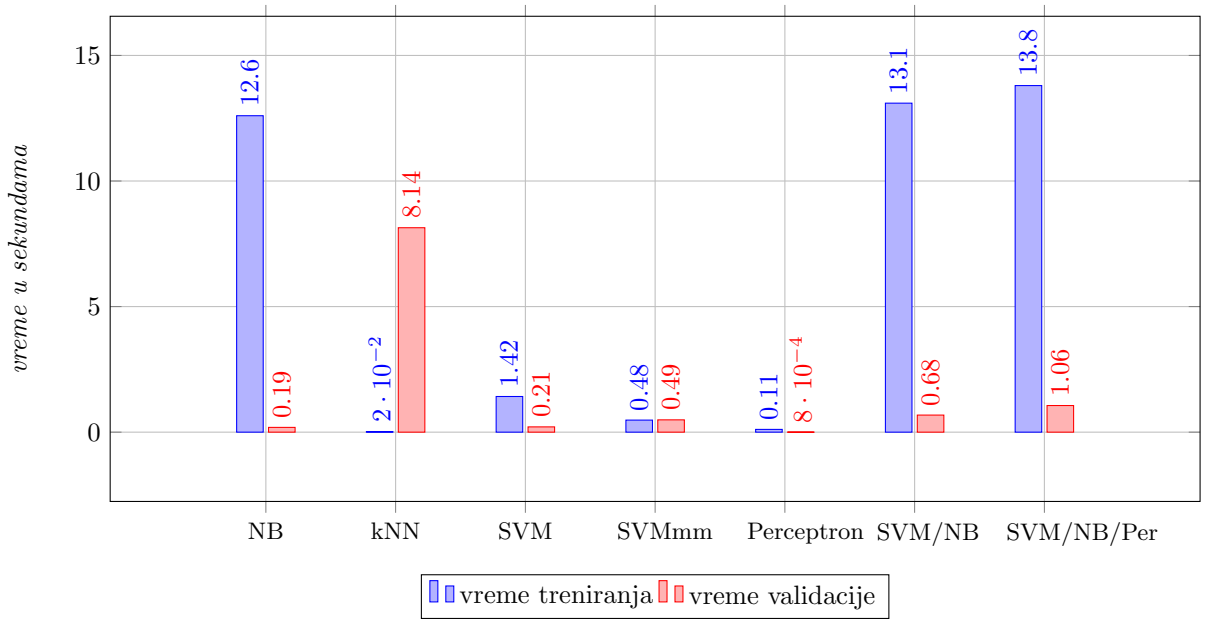
Algoritam	*Lažno pozitivni*	*Lažno negativni*	*Vreme treniranja*	*Vreme validacije*	*Tačnost*	*Stopa greške*	*Preciznost*	*Odziv*	*f1 mera*
Naivni Bajesov klasifikator ($\lambda = 5$)	11	51	12.5867 s	0.1949 s	0.8967	0.1033	0.9281	0.9357	0.821
Naivni Bajesov klasifikator ($\lambda = 50$)	9	58	12.713 s	0.1961 s	0.8883	0.1116	0.9412	0.7128	0.8113
Naivni Bajesov klasifikator ($\lambda = 500$)	8	72	12.6436 s	0.1951 s	0.8667	0.1333	0.9477	0.6682	0.7838
Naivni Bajesov klasifikator ($\lambda = 999$)	8	73	12.6462 s	0.1952 s	0.865	0.135	0.9477	0.6651	0.7816
kNN ($k = 201, l = 120$)	13	146	0.0205 s	8.1379 s	0.735	0.265	0.915	0.4895	0.6378
kNN ($k = 301, l = 150$)	96	0	0.0175 s	8.1801 s	0.84	0.16	0.3725	1	0.5428
kNN ($k = 301, l = 175$)	17	101	0.0184 s	8.1445 s	0.8033	0.1967	0.8889	0.5738	0.6974
kNN ($k = 301, l = 200$)	1	353	0.0181 s	8.2025 s	0.41	0.59	0.9934	0.3009	0.462
Metod podržavajućih vektora bez meke margine	15	22	1.4252 s	0.2117 s	0.9383	0.0616	0.902	0.8625	0.8818
Metod podržavajućih vektora sa mekom marginom	14	8	0.4801 s	0.4916 s	0.9633	0.0367	0.9085	0.9456	0.9267
Perceptron	11	11	0.1157 s	0.0008 s	0.9633	0.0367	0.9281	0.9281	0.9281
SVM/Naivni Bajesov klasifikator	19	2	13.078 s	0.686 s	0.965	0.035	0.8758	0.9853	0.9273
SVM/Naivni Bajesov klasifikator/Perceptron	9	6	13.8014 s	1.0644 s	0.975	0.025	0.9608	0.9423	0.9515

Tabela 26: Rezultati testiranja implementiranih klasifikatora nad *enron6 korpusom*

Napomena: Perceptron nije konvergirao pa je neophodno povećati interval povratnih vrednosti funkcije `string_hash` sa 31973 na 83339 i vrednost parametara `odlike-do` sa 30000 na 80000.



Slika 32: Dijagram ispravno i neispravno klasifikovanih instanci svih implementiranih algoritama nad enron6 korpusom



Slika 33: Dijagram vremena treniranja i validacije svih implementiranih algoritama nad enron6 korpusom

7 Zaključak

Filtriranje neželjene elektronske pošte postaje važan aspekt modernih *e-mail* sistema zbog velikog povećanja obima spama u poslednjih nekoliko godina. Mašinsko učenje privlači pažnju mnogih istraživača na ovom polju kao moćna računarska metodologija koja može da pomogne u ublažavanju takvih problema. U ovom radu su primenjeni i evaluirani neki algoritami mašinskog učenja nad različitim korpusima elektronske pošte u cilju pronalaženja optimalnog algoritma za rešavanje problema spama. Neophodni parametri su određivani eksperimentalno sa velikim brojem kombinacija, tako da su njihove vrednosti verovatno dosta blizu idealnih.

U ovom poglavlju će biti navedeni zaključci o eksperimentalnim rezultatima prethodnog poglavlja. Iz dobijenih rezultata izvedeni su sledeći zaključci:

⋆ Najmanje vremena za treniranje troši algoritam *k najbližih suseda*, dok najmanje vremena za validaciju troši klasifikator zasnovan na *perceptronu*.

⋆ Najviše vremena za treniranje troši *Naivni Bajesov klasifikator*, dok najviše vremena za validaciju troši algoritam *k najbližih suseda*.

⋆ Kako se određivanje vrednosti parametara k i l za algoritam *k najbližih suseda* vrši empirijski, a sama tačnost klasifikacije se pokazala kao najlošijom od svih primenjenih algoritama dolazi se do zaključka da algoritam *k najbližih suseda* ne treba koristiti kod problema filtriranja neželjene elektronske pošte.

⋆ Kod *Naivnog Bajesovog klasifikatora* se pokazalo da uvećavanjem parametra λ koji ukazuje na rizik kada klasifikujemo legitimnu poruku kao neželjenu smanjujemo broj lažno pozitivnih instanci ali i povećavamo broj lažno negativnih sa gotovo istim stepenom tačnosti. Kako je fudamentalna potreba bilo kog filtera neželjene elektronske pošte da nikada ne označi dobru poruku kao neželjenu, odnosno da broj lažno pozitivnih bude što manji u interesu nam je da parametar λ bude što veći.

⋆ Najveću tačnost klasifikacije sa najnižim brojem lažno pozitivnih instanci pokazali su *metod podržavajućih vektora* i klasifikator zasnovan na *perceptronu*. Oba algoritma troše jako malo vremena na treniranje i validaciju u poređenju sa ostalim implementiranim algoritmima. Iz svega navedenog se zaključuje da su *metod podržavajućih vektora* i klasifikator zasnovan na *perceptronu* kao i njihovo kombinovanje najbolji izbor za rešavanje problema spama.

Na kraju se izvodi zaključak da je oblast suzbijanja spama danas zrela i dosta dobro razvijena. Ali zašto je onda naš *inbox* i dalje često pun spama? To je verovatno zato što algoritmi koji se koriste nisu savršeni a sami spameri koriste razne trikove kojima zaobilaze metode filtriranja.

Bibliografija

[1] Introduction to bayesian filtering. *Process Software*. URL http://www.process.com/psc/fileadmin/user_upload/whitepapers/pmas/intro_bayesian_filtering.pdf.

[2] Ling-spam, pu and enron corpora. URL http://csmining.org/index.php/enron-spam-datasets.html.

[3] Combining probabilities. *MathPages*. URL http://www.mathpages.com/home/kmath267.htm.

[4] Pu1 corpus. URL http://www.csmining.org/index.php/pu1-and-pu123a-datasets.html?file=tl_files/Project_Datasets/PU1andPU3/pu1_encoded.tar.tar.

[5] Bayes decision theory - chapter 2. *Pattern Classification and Scene Analysis, John Wiley.*, pages 10–43, 1973.

[6] A simple introduction to maximum entropy models for natural language processing. *Technical report, University of Pennsylvania*, 1997.

[7] Text categorization: A survey. 1999. URL http://citeseer.ist.psu.edu/aas99text.html.

[8] Spamprobe - bayesian spam filtering tweaks. 2003. URL http://spamprobe.sourceforge.net/paper.html.

[9] Fbi internet crime report. 2013. URL http://www.ic3.gov/media/annualreport/2013_IC3Report.pdf.

[10] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, George Paliouras, and Constantine D Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013*, 2000.

[11] Dr Jasmina Novaković. Rešavanje klasifikacionih problema mašinskog učenja. *Monografija Br.4*, 2013.

[12] Mitić Nenad. Klasifikacija - dodatne metode, slajdovi sa predavanja. *Matematički fakultet, Univerzitet u Beogradu*, 2015. URL http://poincare.matf.bg.ac.rs/~nenad/ip.2015/6.klasifikacija.dodatne.metode.pdf.

[13] Mitić Nenad. Svm klasifikacija, slajdovi sa predavanja. *Matematički fakultet, Univerzitet u Beogradu*, 2015. URL http://poincare.matf.bg.ac.rs/~nenad/ip.2015/7.SVM_klasifikacija.pdf.

[14] Gordon V Cormack, Mark D Smucker, and Charles LA Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[15] Harris Drucker, Donghui Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on*, 10(5):1048–1054, 1999.

[16] Tristan Fletcher. Support vector machines explained. *Online]. http://sutikno. blog. undip. ac. id/files/2011/11/SVM-Explained. pdf.[Accessed 06 06 2013]*, 2009.

[17] Joshua Goodman. Sequential conditional generalized iterative scaling. In *In ACL 02*, pages 9–16, 2002.

[18] Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009.

[19] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[20] Thorsten Joachims. Svmlight support vector machine, 2008. URL `http://svmlight.joachims.org/`.

[21] Vojislav Kecman. *Learning and soft computing: support vector machines, neural networks, and fuzzy logic models*. MIT press, 2001.

[22] Ahmed Khorsi. An overview of content-based spam filtering techniques. *Informatica*, 31(3), 2007.

[23] Ahmed Obied. Bayesian spam filtering. *Department of Computer Science University of Calgary amaobied@ucalgary.ca*, 2007.

[24] PhD Sara Radicati. *Email Statistics Report, 2011-2015*. THE RADICATI GROUP, INC., May 2011.

[25] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[26] Symantec. State of spam. *A monthly Report, Report 33*, 2009.

[27] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Always learning. Pearson Education, Limited, 2014. ISBN 9781292026152. URL `https://books.google.rs/books?id=BExxngEACAAJ`.

[28] Konstantin Tretyakov. Machine learning techniques in spam filtering. In *Data Mining Problem-oriented Seminar, MTAT*, volume 3, pages 60–79, 2004.

[29] Michael Vinther. Intelligent junk mail detection using neural networks. *URL: http://www. logicnet. dk/reports/JunkDetection/Jun kDetection. pdf*, 2002.

[30] Tom Van Vleck. The history of electronic mail. `http://www.multicians.org/thvv/mail-history.html`. [Online; accessed 16-November-2015].

[31] J. A. Zdziarski. Bayesian noise reduction: Contextual symmetry logic utilizing pattern consistency analysis. 2004.

[32] Le Zhang and Tian-shun Yao. Filtering junk mail with a maximum entropy model. In *Proceeding of 20th international conference on computer processing of oriental languages (ICCPOL03)*, pages 446–453, 2003.

A Dodatak

U nastavku se nalazi kod programa koji je korišćen u implementaciji i evaluaciji algoritama mašinskog učenja za filtriranje neželjene elektronske pošte.

A.1 FileUtil.h

```
/*
 * FileUtil.h  -  Fajl sistem operacije
 *
 */

#ifndef __FileUtil_h__
#define __FileUtil_h__
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>
#include <iterator>
#include <algorithm>
#include <sys/types.h>
#include <dirent.h>

#include "Izuzetak.h"
using namespace std;

#ifndef DT_REG
  #define DT_REG 8
  #define NOT_LINUX
  #include <sys/stat.h>
#endif

#ifndef DT_DIR
  #define DT_DIR 4
#endif

// Vraca vektor naziva fajlova u datom direktorijumu. Tip parametra precizira koje fajlove da lista.
// Koristiti DT_REG za regularne fajlove, DT_DIR za direktorijume, 255 za sve. Baca izuzetak na neuspeh.

vector<string> citajDirektorijume(const string& dir, unsigned short int type = DT_REG)
{
    vector<string> rezultat;
    DIR* pDir = opendir(dir.c_str());
    if (pDir)
    {
        dirent* pEntry;
        while ( (pEntry = readdir(pDir)) )
        {
            #ifdef NOT_LINUX
            struct stat info;
            if (stat((dir + '/' + string(pEntry->d_name)).c_str(), &info) != 0)
                throw Izuzetak("Nedostupni atributi fajla " + string(pEntry->d_name));
            if ((type & DT_REG  &&  S_ISREG(info.st_mode)) || (type & DT_DIR  &&  S_ISDIR(info.st_mode)))
            {
                rezultat.push_back(string(pEntry->d_name));
            }
            #else
            if ((pEntry->d_type & type))
                rezultat.push_back(string(pEntry->d_name));
            #endif
        }
        closedir(pDir);
    }
    else throw Izuzetak("Nije moguce procitati direktorijum " + dir);
    return rezultat;
}
```

```
62
63  // Citanje fajla kao string. Baca izuzetak na neuspeh.
64
65  string citajFajl(const string& fileName)
66  {
67      ifstream ulazniFajl(fileName.c_str());
68      if (!ulazniFajl) throw Izuzetak("Nije moguce procitati fajl " + fileName);
69
70      // citanje fajla karakter po karakter
71      ostringstream os;
72      ulazniFajl >> noskipws;
73      copy(istream_iterator<char>(ulazniFajl), istream_iterator<char>(), ostream_iterator<char>(os));
74      ulazniFajl.close();
75      return os.str();
76  }
77  #endif
```

A.2 | Instance.h

```
1   /*
2    * Instance.h  -  bazna klasa za kategorizaciju spama
3    */
4
5   #ifndef __Instance_h__
6   #define __Instance_h__
7
8   #include <vector>
9   #include <iostream>
10  #include <algorithm>
11  #include <iterator>
12  #include <map>
13
14  #include "Izuzetak.h"
15  using namespace std;
16
17  /**
18   * Reprezentuje kategorije mejl poruka: SPAM / LEGITIMNO / NEDEFINISANO
19   */
20  enum MejlKlasa
21  {
22      SPAM, LEGITIMNO, NEDEFINISANO
23  };
24
25  ostream& operator<<(ostream& out, const MejlKlasa& cat)
26  {
27      if (cat == SPAM)
28          return out << "S";
29      else
30          if (cat == LEGITIMNO)
31              return out << "L";
32      else
33          return out << "?";
34  }
35
36  // Reprezentuje instancu koja se klasifikuje. Cuva vektor obelezja za instancu.
37  class Instance
38  {
39  public:
40      typedef map<int, bool> Attributes;  // atribut je par indeks->vrednost.
41                                          // prvi atribut ima vrednost 1 !
42      Attributes attributes;
43      MejlKlasa kategorija;
44      int attributeCount;
45      /* Broj instanci atributa (indeks poslednjeg moguceg atributa)
46         NIJE isto kao attributes.size() jer se cuvaju samo ne nula atributi */
47
48      // kreiranje instanci sa odredjenim brojem atributa
49      Instance(int attributeCount):
50          attributes(),
51          kategorija(NEDEFINISANO),
52          attributeCount(attributeCount)
53      {}
54
```

```
55      // Vraca vrednost atribita sa datim indeksom
56      // Nije dobro koristiti attributes[index] jer ce to ubaciti vrednosti u mapu
57      inline bool getAttribute(int index) const
58      {
59          if (attributes.find(index) == attributes.end())
60              return false;
61          else
62              return (attributes.find(index)->second);
63      }
64
65      inline void setAttribute(int index, bool value)
66      {
67          if (index > attributeCount || index < 1)
68              throw Izuzetak("Neispravan indeks atributa");
69          if (value)
70              attributes[index] = true;
71          else
72              if (attributes.find(index) != attributes.end())
73                  attributes.erase(attributes.find(index));
74      }
75
76      // Ispis atributa na izlaz
77      friend ostream& operator<<(ostream& out, const Attributes::value_type& attr)
78      {
79          return out << attr.first << ":" << attr.second;
80      }
81
82      // Ispis instanci na izlaz
83      friend ostream& operator<<(ostream& out, const Instance& inst)
84      {
85          out << "Instanca(";
86          int gotovo = 0;
87          for (Attributes::const_iterator i = inst.attributes.begin(); i != inst.attributes.end(); i++)
88          {
89              if (gotovo++ != 0) out << ", ";
90              out << &i;
91          }
92          out << "; " << inst.kategorija << ")";
93          return out;
94      }
95  };
96
97  /**
98   * Predstavlja skup instanci.
99   * U sustini je apstrakcija vectora instanci.
100  * Skup podataka takodje mora proveriti da sve instance imaju
101  * isti broj atributa sto nisam implementirao.
102  * Druga vazna razlika je da vektor instanci obicno rukuje
103  * "po vrednosti" i kopira napred nazad,
104  * a koristi se kao "normalna klasa" pozivanjem.
105  */
106 class Dataset
107 {
108 public:
109     typedef vector<Instance> Instances;
110     Instances instances;
111
112     Dataset(): instances() {};
113
114     Dataset(Instances instances): instances(instances) {};
115
116     void addInstance(Instance instance)
117     {
118         instances.push_back(instance);
119     }
120
121     void addInstances(vector<Instance> newInstances)
122     {
123         instances.insert(instances.end(), newInstances.begin(), newInstances.end());
124     }
125 };
126
127 #endif
```

A.3 | IzdvajanjeOdlika.h

```
1  /*
2   * IzdvajanjeOdlika.h  -  Klasa za izdvajanje odlika (karakteristika) iz stringova
3   */
4
5  #ifndef __IzdvajanjeOdlika_h__
6  #define __IzdvajanjeOdlika_h__
7
8  #include <fstream>
9  #include <sstream>
10 #include <string>
11 #include <vector>
12 #include <algorithm>
13 #include <cctype>
14
15 #include "Instance.h"
16 #include "FileUtil.h"
17 #include "Izuzetak.h"
18
19 using namespace std;
20
21 /* f-ja za hesiranje reci u brojeve */
22 int string_hash(char *key, int len)
23 {
24     int hash, i;
25     for(hash = i = 0; i < len; ++i)
26     {
27         hash += key[i];
28         hash += (hash << 10);
29         hash ^= (hash >> 6);
30     }
31     hash += (hash << 3);
32     hash ^= (hash >> 11);
33     hash += (hash << 15);
34     return hash % 31973; // 83339
35 }
36
37 /**
38  * Izdvajac odlika (karakteristika) je klasa koja sadrzi jednu f-ju koja konvertuje string u instancu
39  */
40 class IzdvajanjeOdlika
41 {
42 public:
43     // Dati string s vraca odgovarajuci vektor odlika
44     virtual Instance kreirajInstancu(const string& s) const = 0;
45 };
46
47 /**
48  * Izdvajac odlika koji odgovara za test korpuse
49  */
50 class IzdvajanjeOdlikaKorpusa: public IzdvajanjeOdlika
51 {
52 public:
53     int pocetnaRec;
54     int krajnjaRec;
55
56     // Konstruktor
57     // pocetnaRec - rec koja odgovara prvom atributu
58     // krajnjaRec - rec koja odgovara poslednjem atributu
59     IzdvajanjeOdlikaKorpusa(int pocetnaRec, int krajnjaRec):
60         pocetnaRec(pocetnaRec), krajnjaRec(krajnjaRec)
61     {}
62
63     virtual ~IzdvajanjeOdlikaKorpusa(){}
64
65     virtual Instance kreirajInstancu(const string& s) const
66     {
67         Instance rezultat(krajnjaRec - pocetnaRec + 1);
68         istringstream in(s);
69
70         // Preskoci "Subject:" deo poruke
71         char c = ' ';
72         while (c != ':') in >> c;
```

```
73
74          // citaj brojeve iz stringa,
75          // za one koji su manji od brAtributa podesi odgovarajuce atribute u instanci
76          int n;
77          string rec; // koristiti za ne PU korpuse
78          while (in)
79          {
80              //in >> rec;  // citamo rec po rec
81              //n = string_hash((char*) rec.c_str(),rec.length());  //procitane reci kodiram u brojeve
82              in >> n; // koristiti za PU korpuse
83
84              if (n >= pocetnaRec && n <= krajnjaRec)
85                  rezultat.setAttribute(n - pocetnaRec + 1, true);
86          }
87          return rezultat;
88      }
89  };
90
91  /**
92   * Klasa koja konvertuje poruke korpusa u instance
93   */
94  class CitanjeInstanciKorpusa
95  {
96  public:
97      // root direktorijum koji sadrzi korpus
98      string rootDirektorijum;
99      const IzdvajanjeOdlika* odlike;
100
101     // kreiranje novog objekta
102     CitanjeInstanciKorpusa(const string& rootDirektorijum, const IzdvajanjeOdlika* odlike)
103     {
104         if (rootDirektorijum[rootDirektorijum.length() - 1] != '/')
105             this->rootDirektorijum = rootDirektorijum + '/';
106         else
107             this->rootDirektorijum = rootDirektorijum;
108         this->odlike = odlike;
109     }
110
111     // citanje instanci iz fajla
112     // putanja fajla je data relativno u odnosu na root direktorijum
113     // baca izuzetak u slucaju neuspeha
114     Instance citanjeInstanciIzFajla(const string& filename) const
115     {
116         string file = rootDirektorijum + filename;
117         string s = citajFajl(file);
118
119         if (!odlike)
120             throw Izuzetak("Izdvajac odlika nije naveden!");
121         Instance inst = odlike->kreirajInstancu(s);
122
123         // Odredjivanje kategorija instanci. LEGITIMNO je ako naziv fajla
124         // sadrzi "legit", SPAM ukoliko naziv fajla sadrzi "spm" za PU korpuse, a "ham" i "spam" za enron korpus
125         if (filename.find("legit", 0) != string::npos)
126             inst.kategorija = LEGITIMNO;
127         else if (filename.find("spm", 0) != string::npos)
128             inst.kategorija = SPAM;
129         return inst;
130     }
131
132     //cita sve fajlove u direktorijumu koji su dati relativno u odnosu na root direktorijum
133     vector<Instance> citanjeInstanciIzDirektorijuma(const string& dir) const
134     {
135         string punaPutanjaDoDirektorijuma = rootDirektorijum + dir + "/";
136         vector<Instance> rezultat;
137         vector<string> fajlovi = citajDirektorijume(punaPutanjaDoDirektorijuma);
138         for (vector<string>::iterator i = fajlovi.begin(); i != fajlovi.end(); i++)
139         {
140             rezultat.push_back(citanjeInstanciIzFajla(dir + "/" + *i));
141         }
142         return rezultat;
143     }
144
145
146
147
```

```
148     // Kreira trening skup iz korpusa
149     // n je vektor od 1 do 10 - deo koji se izostavlja iz treniranja
150     Dataset* kreiranjeTreningSkupaOdKorpusa(const vector<int>& n)
151     {
152         // dodavacemo direktorijume koji se ne nalaze u datom vektoru
153         vector<int> dodavanjeDirektorijuma;
154         for (int i = 1; i <= 10; i++)
155         {
156             if (find(n.begin(), n.end(), i) == n.end())
157                 dodavanjeDirektorijuma.push_back(i);
158         }
159         return kreiranjeValidacionogSkupaOdKorpusa(dodavanjeDirektorijuma);
160     }
161
162     // Kreiranje validacionog skupa iz korpusa
163     // n je vektor od 1 do 10 - delovi koji se uzimaju za validaciju
164     Dataset* kreiranjeValidacionogSkupaOdKorpusa(const vector<int>& n)
165     {
166         Dataset* rezultat = new Dataset();
167         for (vector<int>::const_iterator i = n.begin(); i != n.end(); i++)
168         {
169             if (*i > 10 || *i < 1)
170                 throw Izuzetak("Ocekuje se n od 1 do 10!");
171             ostringstream os;
172             os << "part" << *i;
173             rezultat->addInstances(citanjeInstanciIzDirektorijuma(os.str()));
174             cout << " Dodane instance iz foldera " << os.str() << ", velicina skupa je sada: "
175                  << rezultat->instances.size() << " instanci"<< endl;
176         }
177         return rezultat;
178     }
179 };
180
181 #endif
```

A.4 Izuzetak.h

```
1  /* Izuzetak.h  -  jednostavna klasa za izuzetke */
2
3  #ifndef __Izuzetak_h__
4  #define __Izuzetak_h__
5
6  #include <string>
7  #include <sstream>
8  using std::string;
9  using std::ostringstream;
10
11 // Konkatenacija stringova sa celobrojnim brojevima.
12 string operator+(const string& s, int i)
13 {
14     ostringstream o;
15     o << s << i;
16     return o.str();
17 }
18
19 class Izuzetak
20 {
21 public:
22     const string message;
23
24     Izuzetak(const char* message): message(message) {}
25
26     Izuzetak(const string& message): message(message) {}
27
28     Izuzetak(): message() {}
29
30     ~Izuzetak(){}
31 };
32
33 #endif
```

A.5 Klasifikator.h

```
/*
 * Klasifikator.h  -  osnovna klasa za klasifikator
 */

#ifndef __Klasifikator_h__
#define __Klasifikator_h__

#include "Instance.h"
#include <vector>
#include <ctime>
using namespace std;

/*
 * Osnovna klasa za klasifikator. Navodi samo interfejs.
 */
class Klasifikator
{
public:
    /**
     * Ime klasifikatora
     */
    virtual string getName() = 0;

    /**
     * Treniranje klasifikatora na podacima
     */
    virtual void treniraj(const Dataset* data) = 0;

    /**
     * Klasifikovanje date instance
     */
    virtual MejlKlasa klasifikuj(const Instance& inst) const = 0;
};

/**
 * Trivijalni klasifikator koji klasifikuje sve kao jednu istu klasu
 */
class TrivijalniKlasifikator: public Klasifikator
{
private:
    MejlKlasa kategorija;
public:
    // kategorija odredjuje klasu u koju ce ovaj klasifikator sve klasifikovati
    TrivijalniKlasifikator(MejlKlasa kategorija): kategorija(kategorija) {}
    virtual ~TrivijalniKlasifikator(){};

    string getName()
    {
        return "Trivijalni klasifikator";
    }

    void treniraj(const Dataset* data)
    {
    }

    MejlKlasa klasifikuj(const Instance& inst) const
    {
        return kategorija;
    }
};

/**
 * Klasa za procenu performansi klasifikatora
 */
class ProcenaPerformansiKlasifikatora
{
public:
    static void testirajKlasifikator(Klasifikator* c, const Dataset* treningSkup, const Dataset* validacioniSkup)
    {
        cout << "Trening klasifikator: " << c->getName() << endl;
        clock_t trenirajStart = clock();
        c->treniraj(treningSkup);
```

```
73          clock_t trenirajKraj = clock();
74          cout << "Testiranje performansi klasifikatora" << endl;
75
76          int fp = 0;
77          int fn = 0;
78          int tp = 0;
79          int tn = 0;
80          int velicinaValidacionogSkupa = validacioniSkup->instances.size();
81
82          clock_t ispravanStart = clock();
83          for (vector<Instance>::const_iterator i = validacioniSkup->instances.begin();
84               i != validacioniSkup->instances.end(); i++)
85          {
86              MejlKlasa stvarna_klasa = i->kategorija;
87              MejlKlasa predvidjena_klasa = c->klasifikuj(*i);
88              if (stvarna_klasa != predvidjena_klasa)
89              {
90                  if (stvarna_klasa == LEGITIMNO)
91                  {
92                      fp++;
93                  }
94                  else
95                  {
96                      fn++;
97                  }
98              }
99              else if (stvarna_klasa == predvidjena_klasa)
100             {
101                 if (stvarna_klasa == LEGITIMNO)
102                 {
103                     tp++;
104                 }
105                 else
106                 {
107                     tn++;
108                 }
109             }
110         }
111         clock_t ispravanKraj = clock();
112
113         double preciznost = ((double)(tp))/(tp+fp);
114         double odziv = ((double)(tp))/(tp+fn);
115
116         cout << "Matrica konfuzije:" << endl;
117         cout << "-----------" << endl;
118         cout << "| "<< tp << " | "<< fn <<" |" << endl;
119         cout << "-----------" << endl;
120         cout << "| "<< fp <<"  | "<< tn <<" |" << endl;
121         cout << "-----------" << endl;
122
123         cout << endl;
124         cout << "-------------------------------------------------------------" << endl;
125         cout << "  Rezultati (" << c->getName() << "): " << endl;
126         cout << "  Velicina validacionog skupa: " << velicinaValidacionogSkupa << endl;
127         cout << "  Lazno pozitivni (LEGITIMNO klasifikovano kao SPAM): " << fp << endl;
128         cout << "  Lazno negativni (SPAM klasifikovano kao LEGITIMNO): " << fn << endl;
129         cout << "  Vreme treniranja: " << ((double)(trenirajKraj - trenirajStart))/CLOCKS_PER_SEC << endl;
130         cout << "  Vreme validacije: " << ((double)(ispravanKraj - ispravanStart))/CLOCKS_PER_SEC << endl;
131         cout << "  Tacnost: " << ((double)(velicinaValidacionogSkupa - (fp + fn)))/velicinaValidacionogSkupa << endl;
132         cout << "  Stopa greske: " << ((double)(fp + fn))/velicinaValidacionogSkupa << endl;
133         cout << "  Preciznost: " << preciznost << endl;
134         cout << "  Odziv: " << odziv << endl;
135         cout << "  f1 mera: " << (double) (2 * preciznost * odziv) / (preciznost + odziv) << endl;
136         cout << "-------------------------------------------------------------" << endl;
137     }
138 };
139
140 #endif
```

A.6 Klasifikator1od2.h

```
/*
 * Klasifikator1od2.h  -  klasifikator koji jednostavno kombinuje odluke druga dva klasifikatora
 *
 */

#ifndef __Klasifikator1od2_h__
#define __Klasifikator1od2_h__

#include "Instance.h"
#include "Klasifikator.h"
#include "Izuzetak.h"
#include <vector>
using namespace std;

/**
 * 1od2 klasifikator uzima rezultate dva klasifikatora i "jake klase".
 * Instanca je klasifikovana u klasu C ako je C jaka klasa i najmanje jedan klasifikator
 * klasifikuje instancu u klasu C ili C nije jaka klasa i oba klasifikatora
 * klasifikuju instancu u klasu C.
 */
class Klasifikator1od2: public Klasifikator
{
private:
    Klasifikator* c1;
    Klasifikator* c2;
    MejlKlasa jakaKlasa;
public:

    Klasifikator1od2(Klasifikator* c1, Klasifikator* c2, MejlKlasa jakaKlasa):
        c1(c1), c2(c2), jakaKlasa(jakaKlasa) {}

    virtual ~Klasifikator1od2(){};

    string getName()
    {
        return "Klasifikator 1 od 2";
    }

    void treniraj(const Dataset* data)
    {
        if (c1 == NULL || c2 == NULL)
            throw Izuzetak("Neinicijalizovan klasifikator!");
        c1->treniraj(data);
        c2->treniraj(data);
    }

    MejlKlasa klasifikuj(const Instance& inst) const
    {
        MejlKlasa d1 = c1->klasifikuj(inst);
        MejlKlasa d2 = c2->klasifikuj(inst);

        if (d1 == jakaKlasa || d2 == jakaKlasa)
            return jakaKlasa;
        else
            return (d1 == d2 ? d1 : NEDEFINISANO);
    }
};

#endif
```

A.7 Klasifikator2od3.h

```
/*
 * Klasifikator2od3.h  -  odredjuje klasu instance prema odlukama 3 klasifikatora
 */

#ifndef __Klasifikator2od3_h__
#define __Klasifikator2od3_h__

#include "Instance.h"
#include "Klasifikator.h"
#include "Izuzetak.h"
#include <vector>
using namespace std;

/**
 * Instanca se klasifikuje u klasu C ako se najmanje
 * dva klasifikatora odluce za klasu C.
 */
class Klasifikator2od3: public Klasifikator
{
private:
    Klasifikator* c1;
    Klasifikator* c2;
    Klasifikator* c3;
public:

    Klasifikator2od3(Klasifikator* c1, Klasifikator* c2, Klasifikator* c3):
        c1(c1), c2(c2), c3(c3) {}

    virtual ~Klasifikator2od3(){};

    string getName()
    {
        return "Klasifikator 2 od 3";
    }

    void treniraj(const Dataset* data)
    {
        if (c1 == NULL || c2 == NULL || c3 == NULL)
            throw Izuzetak("Neinicijalizovani klasifikator");
        c1->treniraj(data);
        c2->treniraj(data);
        c3->treniraj(data);
    }

    MejlKlasa klasifikuj(const Instance& inst) const
    {
        MejlKlasa d1 = c1->klasifikuj(inst);
        MejlKlasa d2 = c2->klasifikuj(inst);
        MejlKlasa d3 = c3->klasifikuj(inst);

        if (d1 == NEDEFINISANO || d2 == NEDEFINISANO || d3 == NEDEFINISANO)
            return NEDEFINISANO;
        int c = 0;
        if (d1 == SPAM)
            c++;
        if (d2 == SPAM)
            c++;
        if (d3 == SPAM)
            c++;
        return (c >= 2 ? SPAM : LEGITIMNO);
    }
};

#endif
```

A.8 KlasifikatorKNajblizihSuseda.h

```
/*
 * KlasifikatorKNajblizihSuseda.h  -  definicija kNN klasifikatora
 */

#ifndef __KlasifikatorKNajblizihSuseda_h__
#define __KlasifikatorKNajblizihSuseda_h__

#include "Instance.h"
#include "Klasifikator.h"
#include "Izuzetak.h"
#include <vector>
using namespace std;

/**
 * Rastojanje izmedju instanci
 */
class Rastojanje
{
public:
    virtual double rastojanje(const Instance& a, const Instance& b) const = 0;
};

class ApsolutnoRastojanje: public Rastojanje
{
public:
    ApsolutnoRastojanje(){}
    virtual ~ApsolutnoRastojanje(){}

    double rastojanje(const Instance& a, const Instance& b) const
    {
        double rezultat = 0;
        if (a.attributeCount != b.attributeCount)
            throw Izuzetak("Nemoguce je racunanje rastojanja izmedju vektora razlicitih dimenzija!");

        Instance::Attributes::const_iterator aAtribut = a.attributes.begin();
        Instance::Attributes::const_iterator bAtribut = b.attributes.begin();

        while (aAtribut != a.attributes.end() || bAtribut != b.attributes.end())
        {
            if (aAtribut == a.attributes.end())
            {
                rezultat++;
                bAtribut++;
            }
            else if (bAtribut == b.attributes.end())
            {
                rezultat++;
                aAtribut++;
            }
            else if (aAtribut->first < bAtribut->first)
            {
                rezultat++;
                aAtribut++;
            }
            else if (aAtribut->first > bAtribut->first)
            {
                rezultat++;
                bAtribut++;
            }
            else
            {
                if (aAtribut->second != bAtribut->second) rezultat++;
                aAtribut++;
                bAtribut++;
            }
        }
        return rezultat;
    }
};
```

```
73  class KlasifikatorKNajblizihSuseda: public Klasifikator
74  {
75  private:
76      int k;  // pozitivan i neparan
77      int l;
78      Dataset::Instances instances;
79      Rastojanje* d;
80
81      /* Binarna f-ja za poredjenje dva para za primenu partial_sort algoritma */
82      class poredjenjeParova
83      {
84      public:
85          bool operator ()(const pair<int, double> a, const pair<int, double> b)
86          {
87              return (a.second < b.second);
88          }
89      };
90
91  public:
92      /* d - rastojanje
93         k - k najbizih suseda
94         l - broj suseda koji moraju biti spam da bi klasifikovali poruku kao spam */
95      KlasifikatorKNajblizihSuseda(Rastojanje* d, int k, int l)
96      {
97          if (k < 1 || l > k)
98              throw Izuzetak("k mora biti pozitivno i vece od l!");
99          this->k = k;
100         this->d = d;
101         this->l = l;
102     }
103     KlasifikatorKNajblizihSuseda(Rastojanje* d, int k)
104     {
105         if (k < 1 || k % 2 == 0)
106             throw Izuzetak("k mora biti pozitivno i neparno!");
107         this->k = k;
108         this->d = d;
109         this->l = (k - 1)/2 + 1;
110     }
111
112     virtual ~KlasifikatorKNajblizihSuseda(){};
113
114     string getName()
115     {
116         return "Klasifikator K Najblizih Suseda";
117     }
118
119     void treniraj(const Dataset* data)
120     {
121         instances = data->instances;
122
123         // provera da li ima nedefinisanih instanci
124         for (Dataset::Instances::iterator i = instances.begin(); i != instances.end(); i++)
125         {
126             if (i->kategorija == NEDEFINISANO)
127                 throw Izuzetak("Nedefinisane instance nisu dozvoljene u trening skupu!");
128         }
129     }
130
131     MejlKlasa klasifikuj(const Instance& inst) const
132     {
133         if (d == NULL) throw Izuzetak("Rastojanje nije navedeno!");
134         if ((int)instances.size() < k)
135             throw Izuzetak("K Najblizih Suseda zahteva da velicina trening skupa bude najmanje k !");
136
137         // za svaku instancu racunamo rastojanje do instance koja se klasifikuke
138         vector<pair<int, double> > v(instances.size());
139
140         for (int i = 0; i < (int)instances.size(); i++)
141         {
142             v[i] = pair<int, double>(i, d->rastojanje(inst, instances[i]));
143         }
144
145         // Uzimamo k najblizih instanci
146         // parcijalno sortiranje - http://www.cplusplus.com/reference/algorithm/partial_sort/
147         partial_sort(v.begin(), v.begin() + k, v.end(), poredjenjeParova());
```

```
148
149         // gledamo koja klasa dominira
150         int brSpam = 0;
151         for (vector<pair<int, double> >::iterator i = v.begin(); i != v.begin() + k; i++)
152         {
153             if (instances[i->first].kategorija == SPAM)
154                 brSpam++;
155         }
156
157         if (brSpam >= 1)
158             return SPAM;
159         else
160             return LEGITIMNO;
161     }
162 };
163
164 #endif
```

A.9 NaivniBajesovKlasifikator.h

```
1  /*
2   * NaivniBajesovKlasifikator.h  -  definicija naivnog Bajesovog klasifikatora
3   *
4   */
5
6
7  #ifndef __NaivniBajesovKlasifikator_h__
8  #define __NaivniBajesovKlasifikator_h__
9
10 #include "Instance.h"
11 #include "Klasifikator.h"
12 #include "Izuzetak.h"
13 #include <vector>
14 using namespace std;
15
16 /**
17  * Naivni Bajesov klasifikator
18  */
19 class NaivniBajesovKlasifikator: public Klasifikator
20 {
21 private:
22     int brAtributa; // broj atributa u instanci
23
24     struct doublePar
25     {
26         double value[2];
27         double& operator [](int index)
28         {
29             return value[index];
30         }
31         double operator[](int index) const
32         {
33             return value[index];
34         }
35
36     };
37
38     vector<doublePar> verovatnoca; // cuva verovatnoce za sve vrednosti svih atributa
39     double lambda; // sklonost klasifikacije
40     double pL_pS;  // P(L) / P(S)
41     static const double VELIKA_VREDNOST = 1000000; // koristim da ne bih kod racunanja
42     static const double MALA_VREDNOST = 0.0000001; // dosao do beskonacnosti ili NaN vrednosti
43                                                    // kao kod na primer deljenja nulom
44 public:
45     NaivniBajesovKlasifikator(double lambda): lambda(lambda){};
46     virtual ~NaivniBajesovKlasifikator(){};
47
48     string getName()
49     {
50         return "Naivni Bajesov klasifikator";
51     }
52
53
```

```
54      void treniraj(const Dataset* data)
55      {
56          // odbacivanje praznog skupa podataka
57          int brInstanci = data->instances.size();
58          if (brInstanci == 0)
59              throw Izuzetak("Prazan skup podataka!");
60
61          // racunanje P(L) i P(S)
62          int brLegitimnih = 0;
63          int brSpam = 0;
64
65          Dataset::Instances::const_iterator i;
66          for (i = data->instances.begin(); i != data->instances.end(); i++)
67          {
68              if (i->kategorija == SPAM)
69                  brSpam++;
70              else if (i->kategorija == LEGITIMNO)
71                  brLegitimnih++;
72              else
73                  throw Izuzetak("Neklasifikovana poruka u trening skupu!");
74          }
75
76          if (brLegitimnih == 0 || brSpam == 0)
77              throw Izuzetak("Ocekuju se bar po jedna legitimna i spam poruka u trening skupu!");
78
79          pL_pS = (double)brLegitimnih/(double)brSpam;
80
81
82          // racunanje verovatnoce za sve atribute i sve vrednosti
83          brAtributa = data->instances[0].attributeCount;
84          if (brAtributa == 0)
85              throw Izuzetak("Instance nemaju atribute!");
86
87          verovatnoca = vector<doublePar>(brAtributa);
88
89          for (int attrib = 0; attrib < brAtributa; attrib++)
90          {
91              int brTacnoKlasifikovanihSpamPoruka = 0;
92              int brTacnoKlasifikovanihLegitimnihPoruka = 0;
93              for (i = data->instances.begin(); i != data->instances.end(); i++)
94              {
95                  if (i->getAttribute(attrib + 1) == true)
96                  {
97                      if (i->kategorija == SPAM)
98                          brTacnoKlasifikovanihSpamPoruka++;
99                      else
100                         brTacnoKlasifikovanihLegitimnihPoruka++;
101                 }
102             }
103
104             // verovatnoca P(x) = P(x | S)/P(x | L)
105             if (brTacnoKlasifikovanihSpamPoruka == 0)
106                 verovatnoca[attrib][1] = MALA_VREDNOST;
107             else
108                 if (brTacnoKlasifikovanihLegitimnihPoruka == 0)
109                     verovatnoca[attrib][1] = VELIKA_VREDNOST;
110             else
111                 verovatnoca[attrib][1] = ((double)brTacnoKlasifikovanihSpamPoruka/
112                                           (double)brTacnoKlasifikovanihLegitimnihPoruka)*pL_pS;
113
114             int brPogresnoKlasifikovanihLegitimnihPoruka = brLegitimnih - brTacnoKlasifikovanihLegitimnihPoruka;
115             int brPogresnoKlasifikovanihSpamPoruka = brSpam - brTacnoKlasifikovanihSpamPoruka;
116
117             if (brPogresnoKlasifikovanihSpamPoruka == 0)
118                 verovatnoca[attrib][0] = MALA_VREDNOST;
119             else if (brPogresnoKlasifikovanihLegitimnihPoruka == 0)
120                 verovatnoca[attrib][0] = VELIKA_VREDNOST;
121             else verovatnoca[attrib][0] = ( (double)brPogresnoKlasifikovanihSpamPoruka/
122                                             (double)brPogresnoKlasifikovanihLegitimnihPoruka)*pL_pS;
123         }
124     }
125
126
127
128
```

```
    MejlKlasa klasifikuj(const Instance& inst) const
    {
        // racunanje verovatnoce P(x|S)/P(x|L) za instancu.
        // naivno pretpostavljamo da je to proizvod verovatnoca odnosa atributa
        double ukupnaVerovatnoca = 1;
        for (int i = 0; i < brAtributa; i++)
        {
            ukupnaVerovatnoca = ukupnaVerovatnoca * verovatnoca[i][inst.getAttribute(i + 1)];
        }

        // poruka se klasifikuje kao spam ako je verovatnoca > lambda * P(L)/P(S)
        if (ukupnaVerovatnoca > lambda * pL_pS)
            return SPAM;
        else
            return LEGITIMNO;
    }
};

#endif
```

A.10 PerceptronKlasifikator.h

```
/* PerceptronKlasifikator.h  -  implementacija klasifikatora baziranog na perceptronu */


#ifndef __PerceptronKlasifikator_h__
#define __PerceptronKlasifikator_h__

#include "Instance.h"
#include "Klasifikator.h"
#include "Izuzetak.h"
#include <vector>
using namespace std;

class PerceptronKlasifikator: public Klasifikator
{
private:
    // Perceptron implementira linearnu f-ju odluke sign(wx + b)
    vector<double> w;
    double b;

    int maxIteracija; // u slucaju da podaci nisu linearno razdvojivi algoritam
                      // nece konvergirati i zato ogranicavamo broj iteracija
public:

    PerceptronKlasifikator(): maxIteracija(10000){};

    PerceptronKlasifikator(int maxIteracija):
      maxIteracija(maxIteracija){};

    virtual ~PerceptronKlasifikator(){};

    string getName()
    {
        return "Perceptron klasifikator";
    }

    void treniraj(const Dataset* data)
    {
        if (data->instances.size() == 0)
            throw Izuzetak("Prazan skup!");

        bool konvergira = false;
        int preostaleIteracije = maxIteracija;

        // Inicijalizacija tezinskog vektora
        w = vector<double>(data->instances[0].attributeCount);
        b = 0;

        while (preostaleIteracije-- > 0 && !konvergira)
        {
            konvergira = true;

```

```
            // trazimo instance koje nisu klasifikovane korektno
            // shodno tome azuriramo tezinski vektor
            for (Dataset::Instances::const_iterator i = data->instances.begin();
                 i != data->instances.end(); i++)
            {
                if (klasifikuj(*i) != i->kategorija)
                {
                    konvergira = false;
                    double c = i->kategorija == SPAM ? 1 : -1;
                    b += c;
                    for (Instance::Attributes::const_iterator attr = i->attributes.begin(); attr != i->attributes.end(); attr++)
                    {
                        int atributIndeks = attr->first;
                        w[atributIndeks - 1] += c;
                    }
                }
            }
        }
        if (preostaleIteracije <= 0)
            throw Izuzetak("Perceptron ne konvergira. Najverovatnije podaci nisu linearno razdvojivi !");
    }

    MejlKlasa klasifikuj(const Instance& inst) const
    {
        double rezultat = 0;
        for (Instance::Attributes::const_iterator i = inst.attributes.begin();
             i != inst.attributes.end(); i++)
        {
            int atributIndeks = i->first;
            rezultat += w[atributIndeks - 1];
        }

        if (rezultat + b > 0)
            return SPAM;
        else
            return LEGITIMNO;
    }
};

#endif
```

A.11 SVMKlasifikator.h

```
/*
 * SVMKlasifikator.h  -  SVM klasifikator (adapter klasa za SVMLight biblioteku).
 */

#ifndef __SVMKlasifikator_h__
#define __SVMKlasifikator_h__

extern "C" {
    #include "svm_light/svm_common.h"
    #include "svm_light/svm_learn.h"
}

#include <iostream>
#include <vector>

#include "Instance.h"
#include "Izuzetak.h"

using namespace std;

/**
 * SVMKlasifikator obavlja SVM klasifikaciju.
 * Ova klasa je samo adapter funkcijama iz SVMLight biblioteke koja obavlja sav posao.
 */
class SVMKlasifikator: public Klasifikator
{
private:
    // Promenljive sadrze podatke potrebne za SVMLight procedure

    // Parametri ucenja
```

```
31      LEARN_PARM learnParm;
32      KERNEL_PARM kernelParm;
33      long kernelCacheSize;
34
35      // Model
36      MODEL model;
37
38      // Konvertuje instance u DOC strukturu koju koristi SVMLight
39      static void instancaKonverzijaDOC(const Instance& inst, DOC& doc)
40      {
41          // alocira memoriju za odgovarajuci broj zapisa
42          doc.words = (WORD*) malloc(sizeof(WORD) * (inst.attributes.size() + 1));
43          if (doc.words == NULL)
44              throw Izuzetak("Nije moguce alocirati memoriju!");
45
46          int curWordIndex = 0;
47          for (Instance::Attributes::const_iterator i = inst.attributes.begin(); i != inst.attributes.end(); i++)
48          {
49              doc.words[curWordIndex].wnum = i->first;
50              doc.words[curWordIndex].weight = (FVAL)i->second;
51              curWordIndex++;
52          }
53          doc.words[curWordIndex].wnum = 0;
54          doc.words[curWordIndex].weight = 0.0;
55
56          doc.queryid = 0;     // neophodno za rangiranje
57          doc.costfactor = 1; // troskovi reklasifikacije za ovu instancu
58          doc.docnum=-1;       // pozicija dokumenta u nizu trening skupa
59          doc.twonorm_sq=sprod_ss(doc.words, doc.words); // neophodno za unutrasnju upotrebu za SVMLight-a
60      }
61
62  public:
63      // Kreiranje klasifikatora
64      SVMKlasifikator(bool softmargin = false)
65      {
66          // Podesavanje podrazumevanih parametara ucenja
67
68          // Konfigurisanje parametara ucenja
69          // podrazumevane vrednosti su u svm_learn_main.c::read_input_parametareters
70          learnParm.type = CLASSIFICATION;      /* tip ucenja je klasifikacija, a mogu biti jos i regresija, ranking ili optimizacija*/
71          learnParm.biased_hyperplane = 1;      /* koriscenje wx + b za razdvajajucu hiper-ravan */
72          learnParm.remove_inconsistent = 0;    /* ako je ukljuceno izostavlja primere sa alfa uz C
73                                                   i model se ponovo trenira */
74          learnParm.skip_final_opt_check = 0;   /* ne preskakanje Karush-Kuhn-Tucker uslova na kraju optimizacije
75                                                   za primere uklonjene smanjivanjem */
76          learnParm.svm_maxqpsize = 10;         /* velicina radnog skupa q */
77          learnParm.svm_newvarsinqp = 0;        /* nove promenljive se unose u radni skup u novoj iteraciji */
78          learnParm.svm_iter_to_shrink = -9999; /* iteracije h posle kojih primer moze da se ukloni smanjivanjem */
79          learnParm.svm_c = 0.0;                /* gornja granica parametra C na alfa*/
80          learnParm.eps  = 1.0;                 /* regresija epsilon (eps=1.0 za klasifikaciju) */
81          learnParm.transduction_posratio = -1.0; /* deo neobelezenih primera koji se klasifikuju kao pozitivni */
82          learnParm.svm_costratio = 1.0;          /* faktor kojim se mnozi C za pozitivne primere */
83          learnParm.svm_costratio_unlab = 1.0;
84          learnParm.svm_unlabbound = 1E-5;
85          learnParm.epsilon_crit = 0.001; /* tolerisana greska za rastojanja koja se koristi
86                                             za kriterijum zaustavljanja*/
87          learnParm.compute_loo = 0;      /* ako je ukljuceno izracunava procenu jednog izostavljanja */
88          learnParm.rho = 1.0;            /* parametar u xi/alpha proceni za leave-one-out odsecanje
89                                             ciji je opseg [1..2] */
90          learnParm.xa_depth = 0;         /* parametar u xi/alpha proceni gornje
91                                             granice broja porzavajucih vektora */
92
93          learnParm.predfile[0] = 0;      /* fajl za predikciju neobelezenih primera
94                                             u transdukciji */
95          learnParm.alphafile[0] = 0;     /* fajl u kome se smesta optimalno alfa, koriste se prazni
96                                             stringovi ukoliko alfe ne treba da budu na izlazu*/
97
98          // konstante koje ne treba dirati
99          learnParm.epsilon_const = 1E-20;
100         learnParm.epsilon_shrink = 1E-6;
101         learnParm.opt_precision = 1E-21;
102         earnParm.epsilon_a = 1E-15;
103
104         // postavka kernela
105         kernelParm.custom[0] = 0;
```

```
106         kernelParm.kernel_type = 0;  /* 0=linear, 1=poly, 2=rbf, 3=sigmoid, 4=custom, 5=matrix */
107         kernelParm.poly_degree = 3;
108         kernelParm.rbf_gamma = 1.0;
109         kernelParm.coef_lin = 1;
110         kernelParm.coef_const = 1;
111
112         kernelCacheSize = 40;
113
114         // provera doslednosti parametara
115         if(learnParm.svm_iter_to_shrink == -9999) {
116             if(kernelParm.kernel_type == LINEAR)
117                 learnParm.svm_iter_to_shrink=2;
118             else
119                 learnParm.svm_iter_to_shrink=100;
120         }
121         if((learnParm.skip_final_opt_check)
122             && (kernelParm.kernel_type == LINEAR)) {
123             cout << "\nPreskakanje poslednje provere optimalnosti linearnih kernela!\n\n";
124             learnParm.skip_final_opt_check=0;
125         }
126
127         if((learnParm.skip_final_opt_check)
128             && (learnParm.remove_inconsistent))
129             throw Izuzetak("Neophodno je odraditi poslednju proveru optimalnosti prilikom uklanjanja nekozistentnih primera!");
130         if((learnParm.svm_maxqpsize<2))
131             throw Izuzetak("Maksimalna velicina podproblema kvadratnog programiranja nije u validnom opsegu od [2,...]!");
132         if((learnParm.svm_maxqpsize<learnParm.svm_newvarsinqp))
133             throw Izuzetak("Maksimalna velicina podproblema kvadratnog programiranja mora biti veca od broja novih promenljivih koje
                    ulaze u radni skup pri svakoj iteraciji!");
134         if(learnParm.svm_iter_to_shrink<1)
135             throw Izuzetak("Maksimalan broj iteracija za smanjivanje nije u validnom opsegu [1,...]!");
136         if(learnParm.svm_c<0)
137             throw Izuzetak("Parametar C mora biti veci od nule!");
138         if(learnParm.transduction_posratio>1)
139             throw Izuzetak("Deo neobelezenih primera koji su klasifikovani kao pozitivni mora biti manji od 1.0!");
140         if(learnParm.svm_costratio<=0)
141             throw Izuzetak("Parametar svm_costratio mora biti veci od nule!");
142         if(learnParm.epsilon_crit<=0)
143             throw Izuzetak("Parametar epsilon_crit mora biti veci od nule!");
144         if(learnParm.rho<0)
145             throw Izuzetak("Parametar rho za xi/alpha procenu i leave-one-out odsecanje mora biti veci od nule (obicno 1.0 ili 2.0)!");
146         if((learnParm.xa_depth<0) || (learnParm.xa_depth>100))
147             throw Izuzetak("Parametar xa_depth za xi/alpha procenu mora biti u granicama [0..100]!");
148
149         // level of SVM-light debugging infos
150         verbosity = 1;
151
152         if (!softmargin)
153         {
154             learnParm.svm_c = 1E15; // parametar C kod tvrde margine
155         }
156         else
157         {
158             // stelovanje cene kostanja kojom se mnozi parametar C za pozitivne primere kod meke margine
159             learnParm.svm_costratio = 0.3;
160         }
161     }
162
163     virtual ~SVMKlasifikator(){};
164
165     string getName()
166     {
167         return "SVM Klasifikator";
168     }
169
170     void treniraj(const Dataset* data)
171     {
172         // provera da li ima neklasifikovanih instanci
173         if (data->instances.size() == 0) throw Izuzetak("Za treniranje se ocekuje neprazan skup podataka!");
174
175         // prevodjenje vektora instanci u skup DOC format za SVMLight
176         DOC *docs;      // trening primeri
177         double *target; // oznake
178         long totwords = data->instances[0].attributeCount; // broj atributa
179         long totdoc = data->instances.size(); // broj primera
```

```
180
181         docs = (DOC*) malloc(sizeof(DOC)*totdoc);
182         target = (double *) malloc(sizeof(double)*totdoc);
183         if (!docs || !target)
184             throw Izuzetak("Neuspesna alokacija memorije!");
185
186         int curDoc = 0;
187         for (Dataset::Instances::const_iterator i = data->instances.begin(); i != data->instances.end(); i++)
188         {
189             instancaKonverzijaDOC(*i, docs[curDoc]);
190             if (i->kategorija == NEDEFINISANO)
191                 throw Izuzetak("Nedefinisane instance nisu dozvoljene u trening skupu!");
192             else if (i->kategorija == SPAM)
193                 target[curDoc] = 1.0;
194             else
195                 target[curDoc] = -1.0;
196             curDoc++;
197         }
198
199         // model za ucenje
200         if(kernelParm.kernel_type == LINEAR)
201         {
202             cout << "treniranje klasifikatora" << endl;
203             svm_learn_classification(docs,target,totdoc,totwords,&learnParm, &kernelParm,NULL,&model);
204         }
205         else
206         {
207            // Uvek uzima novi kernel kes. Nije moguce koristiti isti kes za dva razlicita treniranja.
208            KERNEL_CACHE kernelCache;
209            kernel_cache_init(&kernelCache,totdoc,kernelCacheSize);
210            svm_learn_classification(docs, target, totdoc, totwords, &learnParm, &kernelParm, &kernelCache, &model);
211            // oslobodi memoriju koriscenu za kesiranje.
212            kernel_cache_cleanup(&kernelCache);
213         }
214     }
215
216     MejlKlasa klasifikuj(const Instance& inst) const
217     {
218         if (model.sv_num == 0)
219             throw Izuzetak("Neobuceni klasifikator!");
220
221         DOC doc;
222         instancaKonverzijaDOC(inst, doc);
223
224         if (classify_example(const_cast<MODEL*>(&model), &doc) > 0)
225             return SPAM;
226         else
227             return LEGITIMNO;
228     }
229 };
230
231 #endif
```

A.12 main.cpp

```
1  /*
2   * main.cpp  -  glavni fajl projekta za klasifikaciju poruka el. poste
3   *                  algoritmima masinskog ucenja
4   */
5
6  #include <iostream>
7
8  #include "Instance.h"
9  #include "IzdvajanjeOdlika.h"
10 #include "Klasifikator.h"
11 #include "NaivniBajesovKlasifikator.h"
12 #include "KlasifikatorKNajblizihSuseda.h"
13 #include "SVMKlasifikator.h"
14 #include "PerceptronKlasifikator.h"
15 #include "Klasifikator1od2.h"
16 #include "Klasifikator2od3.h"
17 #include "Izuzetak.h"
18 #include "Konfiguracija.h"
```

```
19
20  using namespace std;
21
22  int main(int argc, char* argv[])
23  {
24    try{
25      cout << "Klasifikacija poruka el. poste algoritmima masinskog ucenja." << endl;
26
27      // Citanje parametara
28      Konfiguracija podesavanja("parametri.txt", argc, argv);
29
30      podesavanja.setDefault("direktorijum-korpusa", "../korpusi/PU123ACorpora/pu_corpora_public/pu1");
31      podesavanja.setDefault("deo-korpusa", "1");   // deo korpusa koji ostavljamo za validaciju
32      podesavanja.setDefault("odlike-od", "1");     // odlike koje ce biti izdvojene od ove reci
33      podesavanja.setDefault("odlike-do", "30000"); // odlike koje ce biti izdvojene do ove reci
34      podesavanja.setDefault("produzeni-validacioni-skup", "1"); // da li ce trening instance
35                                                                 // biti dodate skupu za validaciju
36      podesavanja.setDefault("nb-lambda", "1"); // lambda Naivnog Bajesovog klasifikatora
37      podesavanja.setDefault("knn-k", "50");    // k najblizih suseda
38      podesavanja.setDefault("knn-l", "30");    // ako je l ili vise poruka medju k najblizih suseda
39                                                // poruke x spam klasikuj x kao spam
40      podesavanja.setDefault("svm-mm", "0");    // koriscenje meke margine za SVM
41
42      /* klasifikator se testira (0 - ne testirati, 1 - testirati) */
43      podesavanja.setDefault("naivnibajes", "1");
44      podesavanja.setDefault("knn", "1");
45      podesavanja.setDefault("perceptron", "1");
46      podesavanja.setDefault("trivijalni", "1");
47      podesavanja.setDefault("svm", "1");
48      podesavanja.setDefault("svm-naivnibajes", "1");
49      podesavanja.setDefault("svm-naivnibajes-perceptron", "1");
50
51      vector<int> deoKorpusa = podesavanja.getIntVectorParam("deo-korpusa", 1, 10);
52      int odlikeOdReci = podesavanja.getIntParam("odlike-od", 1, 90000);
53      int odlikeDoReci = podesavanja.getIntParam("odlike-do", 1, 90000);
54      if (odlikeDoReci < odlikeOdReci)
55          throw Izuzetak("parametar odlike-do treba biti veci od parametra odlike-od");
56      int produzenaValidacija = podesavanja.getIntParam("produzeni-validacioni-skup", 0, 1);
57      int nbLambda = podesavanja.getIntParam("nb-lambda", 1, 1000);
58      int knnK = podesavanja.getIntParam("knn-k", 1, 1000);
59      int knnL = podesavanja.getIntParam("knn-l", 1, 1000);
60      if (knnL > knnK)
61          throw Izuzetak("knn-k treba da bude vece od knn-l");
62
63      int svmMekaMargina = podesavanja.getIntParam("svm-mm", 0, 1);
64      int radiNaivniBajes = podesavanja.getIntParam("naivnibajes", 0, 1);
65      int radiKNN = podesavanja.getIntParam("knn", 0, 1);
66      int radiPerceptron = podesavanja.getIntParam("perceptron", 0, 1);
67      int radiTrivijalniKlasifikator = podesavanja.getIntParam("trivijalni", 0, 1);
68      int radiSVM = podesavanja.getIntParam("svm", 0, 1);
69      int radiSvmNaivniBajes = podesavanja.getIntParam("svm-naivnibajes", 0, 1);
70      int radiSvmPerceptron = podesavanja.getIntParam("svm-naivnibajes-perceptron", 0, 1);
71
72
73      //Priprema citanja instanci korpusa
74      IzdvajanjeOdlika* io = new IzdvajanjeOdlikaKorpusa(odlikeOdReci, odlikeDoReci);
75      CitanjeInstanciKorpusa korpus(podesavanja.parametri["direktorijum-korpusa"], io);
76
77      cout << "Pripremanje trenirajuceg skupa ( deo-korpusa=";
78      copy(deoKorpusa.begin(), deoKorpusa.end(), ostream_iterator<int>(cout, " "));
79      cout << ")" << endl;
80      Dataset* treningSkup = korpus.kreiranjeTreningSkupaOdKorpusa(deoKorpusa);
81      cout << "Pripremanje validacionog skupa" << endl;
82      Dataset* validacioniSkup = korpus.kreiranjeValidacionogSkupaOdKorpusa(deoKorpusa);
83      if (produzenaValidacija)
84          validacioniSkup->addInstances(treningSkup->instances);
85
86
87      // kNN klasifikator
88      Rastojanje* d = new ApsolutnoRastojanje();
89      Klasifikator* knn = new KlasifikatorKNajblizihSuseda(d, knnK, knnL);
90      // Naivni Bajesov klasifikator
91      Klasifikator* nb = new NaivniBajesovKlasifikator(nbLambda);
92      // Trivijalni klasifikator
93      Klasifikator* tc = new TrivijalniKlasifikator(LEGITIMNO);
```

```
94      // SVM klasifikator
95      Klasifikator* svm = new SVMKlasifikator(svmMekaMargina == 1);
96      // Perceptron klasifikator
97      Klasifikator* perceptron = new PerceptronKlasifikator();
98      // SVM sa mekom marginom
99      Klasifikator* svmSoft = new SVMKlasifikator(true);
100     // 1 od 2 (SVM meka margina/Naivni Bajes)
101     Klasifikator* svmnb = new Klasifikator1od2(svmSoft, nb, SPAM);
102     // 2 od 3 (SVM meka margina/Naivni Bajes/Perceptron);
103     Klasifikator* svmnbperceptron = new Klasifikator2od3(svmSoft, nb, perceptron);
104
105     if (radiTrivijalniKlasifikator)
106         ProcenaPerformansiKlasifikatora::testirajKlasifikator(tc, treningSkup, validacioniSkup);
107     if (radiPerceptron)
108         ProcenaPerformansiKlasifikatora::testirajKlasifikator(perceptron, treningSkup, validacioniSkup);
109     if (radiNaivniBajes)
110         ProcenaPerformansiKlasifikatora::testirajKlasifikator(nb, treningSkup, validacioniSkup);
111     if (radiSVM)
112         ProcenaPerformansiKlasifikatora::testirajKlasifikator(svm, treningSkup, validacioniSkup);
113     if (radiKNN)
114         ProcenaPerformansiKlasifikatora::testirajKlasifikator(knn, treningSkup, validacioniSkup);
115     if (radiSvmNaivniBajes)
116         ProcenaPerformansiKlasifikatora::testirajKlasifikator(svmnb, treningSkup, validacioniSkup);
117     if (radiSvmPerceptron)
118         ProcenaPerformansiKlasifikatora::testirajKlasifikator(svmnbperceptron, treningSkup, validacioniSkup);
119
120
121     cout << "Uspesno zavrseno." << endl;
122   }
123   catch(Izuzetak e){
124     cerr << "Greska: " << e.message << endl;
125     return 1;
126   }
127   return 0;
128 }
```