LATVIJAS UNIVERSITĀTE
FIZIKAS UN MATEMĀTIKAS FAKULTĀTE
MATEMĀTIKAS NODAĻA

ATBALSTA VEKTORU METODES IZMANTOŠANA TEKSTA KLASIFIKĀCIJAI

MAĢISTRA DARBS

Autore: **Kristīne Dzalbe**
Stud. apl. nr.: kd11016
Darba vadītājs: Dr.sc.comp. Normunds Grūzītis

Rīga
2017

Anotācija

Mūsdienās arvien pieaug uzglabātās informācijas un datu apjoms. Daudz informācijas tiek uzkrāts teksta dokumentos, kas lielākoties tiek uzglabāti nestrukturētā veidā. Maģistra darba mērķis ir iepazīties ar teksta klasifikācijas problemātiku un izpētīt dažādas, biežāk lietotās mašīnmācīšanās metodes, ko izmanto šī uzdevuma atrisināšanai. Tāpat darbā apskatītas metodes teksta datu dimensiju skaita samazināšanai. Darba gaitā veikta angļu valodas datu klasifikācija atbilstoši tēmām, izmantojot "*The New York Times*" ziņu virsrakstu datus. Veikta arī latviešu ziņu portālu komentāru klasifikācija agresīvos un neagresīvos komentāros. Abām datu kopām klasifikācija veikta, izmantojot atbalsta vektoru metodi, klasifikācijas kokus un gadījuma mežus. Labākie rezultāti sasniegti ar atbalsta vektoru metodi.

Atslēgas vārdi: teksta klasifikācija, mašīnmācīšanās, atbalsta vektoru metode

Abstract

Nowadays amount of stored information and data increase exponentially. Besides, a lot of this information is accumulated in textual data. Those text data usually are stored in unstructured way. The aim of this paper is to investigate the problem of text classification and explore most frequently used machine learning methods for this task. Moreover, different dimensionality reduction techniques for textual data are investigated in this paper. To reach the goal of the thesis two different data sources are used: English headlines from "*The New York Times*" and Latvian comments of Latvian news portals. Three classifiers are employed: support vector machines, decision trees and random forests, however, the best classification accuracy are achieved with support vector machines.

Key words: text classification, machine learning, support vector machine

Saturs

Ievads

Pēdējo gadu laikā strauji pieaudzis interneta lietotāju skaits un vēl straujāk ir audzis datu un tekstu apjoms, kas pieejams digitālajā vidē. Publikācijas, ziņas, atsauksmes, komentāri, sūdzības - visu šo datu apjoms ir neizmērojami liels. Šie dati lielākoties ir nestrukturēti vai daļēji strukturēti. Šī iemesla dēļ par aktuālu problēmu mūsdienās ir kļuvusi tekstu un dokumentu klasificēšana. Teksta datu analīze ļauj ātri un ērti iegūt secinājumus par šiem datiem un nav nepieciešams tos manuāli lasīt, kas prasītu daudz resursu un laika.

Teksta dokumentu klasificēšanai arvien vairāk tiek izmantotas dažādas mašīnmācīšanās metodes. Ja dokumentus nepieciešams klasificēt divās grupās, tad labākos rezultātus parasti uzrāda atbalsta vektoru metode. Šīs metodes galvenā priekšrocība ir tāda, ka tā labi strādā gadījumos, kad ir ļoti liels datu dimensiju skaits. Un teksta klasifikācijas uzdevumiem liels dimensiju skaits ir raksturīgi.

Atbalsta vektoru metodi 1963. gadā ieviesa Padomju Savienības matemātiķi Vladimirs Vapniks un Aleksejs Červonenskis. Deviņdesmito gadu sākumā tika izstrādāta kodolu substitūcijas pielietošana atbalsta vektoru metodei, kas pavēra ļoti plašas iespējas šīs metodes pielietošanai praktiskos klasifikācijas uzdevumos.

Darba galvenais mērķis ir izstrādāt modeli, kas klasificētu latviešu ziņu portālu komentārus agresīvos un neagresīvos komentāros. Tā kā šis klasifikācijas uzdevums satur divas grupas, darba gaitā lielāka uzmanība pievērsta tieši divu grupu klasifikācijas problēmām. Tāpat veikta arī angļu valodas datu klasifikācija.

Maģistra darba galvenie uzdevumi

1. iepazīties ar teksta klasifikācijas problēmas būtību,
2. izpētīt mašīnmācīšanās metodes, ko izmanto teksta klasifikācijai,
3. detalizētāk aplūkot atbalsta vektoru metodi,
4. izveidot klasifikācijas modeļus reāliem datiem.

Pirmajā nodaļā apskatīti teksta analīzes un klasifikācijas piemēri. Aprakstīta datu sagatavošana teksta klasifikācijai, to starpā celmošana (*stemming*) un lemmatizācija (*lemmatization*), kā arī dots ieskats par mašīnmācīšanās metodēm, ko izmanto teksta klasifikācijas uzdevumiem.

Otrajā nodaļā sniegts atbalsta vektoru metodes teorētiskais apraksts un pamatojums, kāpēc var tikt veikta transformācija, izmantojot kodolu funkcijas.

Trešajā un ceturtajā nodaļā tiek apskatīti iegūtie rezultāti, veicot teksta datu klasifikāciju. Trešajā nodaļā aplūkota angļu valodas datu klasifikācija. Izmantota "*The New York Times*" ziņu virsrakstu datu kopa. Ceturtajā nodaļā aprakstīta latviešu ziņu portālu komentāru klasifikācija agresīvos un neagresīvos komentāros. Abām datu kopām tiek veikta klasifikācija ar atbalsta vektoru metodi, klasifikācijas kokiem un gadījuma mežiem. Abās nodaļās tiek apskatīti atbalsta vektoru metodes rezultāti atkarībā no kodolu funkcijas.

Visas darbā aprakstītās metodes realizētas programmā R. Darba gaitā izstrādātie programmu kodi ir pievietoti pielikumā.

1. Teksta klasifikācija

Līdz deviņdesmito gadu vidum tekstu klasifikācijai pārsvarā izmantoja diezgan vienkāršus, manuāli definētus kritērijus, kas balstījās uz ekspertu viedokli. Bet pieaugot tekstu un dokumentu apjomam un attīstoties vajadzībai analizēt un apstrādāt šos datus, tekstu klasificēšanai arvien vairāk tiek izmantota datu izrace un mašīnmācīšanās metodes. Atbilstošas un pareizas klasifikācijas metodes izveide un milzīgā teksta datu apjoma pētīšana mūsdienās ir nozīmīga pētījumu nozare.

Teksta klasifikācijai ir ļoti plašas izmantošanas iespējas. Viens no klasiskākajiem piemēriem ir bibliotēkās sakārtot grāmatas atbilstoši tematiem. Mūsdienās digitālajā vidē, digitālajās datubāzēs nepieciešams sagrupēt grāmatas, publikācijas, rakstus atbilstoši tematiem. Uzdevums ir noteikt, vai konkrētā publikācija ir par epidemioloģiju vai tomēr par embrioloģiju. Ir svarīgi dokumentu iekļaut pareizajā grupā, lai pēc tam to būtu ērti atrast un apstrādāt.

Informācijas izvilkšana (*Information Extraction*) ir cits plaši izmantots teksta analīzes uzdevums. Tā būtība ir automātiski izvilkt strukturētu informāciju no nestrukturēta vai daļēji strukturēta teksta. Tālāk iegūto informāciju var izmantot teksta klasificēšanai vai meklēšanas algoritmu izstrādei.

Uzņēmumi, kas darbojas lielas konkurences apstākļos, monitorē to, ko publicē un paziņo konkurenti, lai spētu ātri reaģēt un rīkoties. Taču manuāla konkurentu ziņu monitorēšana ir ļoti lēns un laikietilpīgs process. Efektīvāks veids, kā paveikt šo uzdevumu, ir uzbūvēt "gudro rāpuli" (*smart crawler*), kas interneta lapās ik dienas meklē publicēto informāciju par konkurentiem. Atrodot saistošus un noderīgus rakstus, tie tiek apkopoti un saglabāti vienuviet, ko pēc tam cilvēkam izskatīt ir ļoti ērti.

Teksta analīzes metodes izmanto, lai izstrādātu personalizētu ziņu filtru. Balstoties uz to, ko cilvēks ir iepriekš lasījis, algoritms iemācās atpazīt tās ziņas, kas konkrētajam lasītajam būtu saistošākas un piedāvā tieši šos rakstus.

Tekstu klasifikācijas metodes izmanto, lai no ienākošajiem e-pastiem automātiski atdalītu surogātpastu. Šo uzdevumu veic ne tikai lielākie e-pastu sniedzēji, bet arī dažādas kompānijas, kas darbojas pakalpojumu sniegšanas jomā. Šiem uzņēmumiem klienti sūta e-pastus. Kompānijās ir darbinieki, kas ienākošos e-pastu izlasa un sniedz nepieciešamo atbildi. Bet bieži vien klientu sūtītie e-pasti ir nederīgi, un, lai darbiniekam nebūtu jāvelta laiks to lasīšanai, tiek izmantotas teksta klasifikācijas metodes, lai to paveiktu automātiski. Tāpat ienākošos e-pastus var sadalīt tēmās un automātiski pārsūtīt konkrētam

darbiniekam, kurš spēj atrisināt klienta problēmu. Tādā veidā tiek ietaupīts darbinieku laiks un uzņēmumam samazinās izmaksas.

Interneta vidē bieži vien tiek rakstītas atsauksmes par dažādām grāmatām, filmām, notikumiem, pakalpojumiem, uzņēmumiem utt. Izlasīt visas atsauksmes par interesējošo tematu ir ļoti laikietilpīgi, tāpēc svarīgi ir automātiski noteikt atsauksmju noskaņojumu jeb sentimentu. Sentimentu analīzes (*sentiment analysis*) mērķis ir noteikt to, vai dokuments ir pozitīvs vai negatīvs. Izmantojot sentimentu analīzi, ātri iespējams noteikt klientu vai sabiedrības kopējo noskaņojumu par kādu konkrētu jautājumu, kas, savukārt, palīdz izvērtēt veiktās darbības un saprast, kā labāk rīkoties nākotnē.

Teksta analīžu metodes ir ļoti dažādas, bet turpmāk darbā pievērsīsimies teksta klasifikācijai.

Apakšnodaļas apraksts balstīts uz [18], [1], [16], [15]

1.1. Datu sagatavošana klasifikācijai

1.1.1. Dimensiju skaita samazināšana

Dokumentu klasificēšanas procesā svarīga nozīme ir pareizai datu sagatavošanai, lai samazinātu datu klasificēšanai nepieciešamo laiku un lai klasifikācija būtu precīzāka. Būtiskākā teksta klasifikācijas uzdevuma problēma ir ļoti lielais datu dimensiju skaits. Šī iemesla dēļ daudzas klasifikācijas metodes nav iespējams pielietot, vai arī tās prasa ļoti ilgu laiku, lai veiktu klasifikāciju, tādēļ svarīgi ir samazināt datu dimensiju skaitu. Svarīgi izslēgt nenozīmīgās un liekās pazīmes (*features*), citādi tās var degradēt klasifikācijas algoritmu.

Pirmais solis ir vārdu tokenizēšana (*tokenization*). Tas nozīmē, ka teksts tiek sadalīts sīkās vienībās, parasti vārdos. Tādā veidā tiek nodalītas arī pieturzīmes, skaitļi, e-pasta adreses, telefona numuri utt. Piemēram, "es", "sēdēt", "...", ":)", "2356", "epasts@provider.com", "www.lu.lv" u.c. Veicot teksta apstrādi, nepieciešams visus burtus pārveidot vai nu par lielajiem vai mazajiem burtiem, jo vairākas datu apstrādes programmas ir reģistrjutīgas (*case sensitive*), kas nozīmē, ka vārdi "Jānis" un "jānis" tiks uzskatīti par diviem dažādiem vārdiem.

Nākamais solis ir izņemt pieturzīmes, skaitļus, ja tie nav svarīgi klasificēšanas uzdevumā, kā arī izņemt tā saucamos stopvārdus (*stopwords*), kas tekstos ir bieži sastopami, bet kas ir maznozīmīgi klasifikācijas uzdevumam. Piemēram, stopvārdi ir "un", "gan", "arī", "ka", "bet" u.c.

Tālāk var veikt vārdu celmošanu, kas nozīmē vārdu dalīšanu sastāvdaļās, tas ir, nodalīt priedēkli, vārda sakni, piedēkli un galotni un tālāk modelēšanas procesā par pazīmi uzskatīt tikai vārda sakni. Šādā veidā vārdi "aita", "aitu", "aitiņas" tiks apvienoti vienā pazīmē. Ir izstrādāti vairāki celmošanas algoritmi, kas ir piemēroti angļu valodai: Paisa un Huska celmošanas algoritms (*Paice/Husk stemmer*), Portera celmošanas algoritms (*Porter's stemmer*) un Lovina celmošanas algoritms (*Lovin's stemmer*). Visi šie algoritmi izstrādāti, lai nodalītu vārda galotni. Praksē biežāk lietotais celmošanas algoritms angļu valodai ir Portera algoritms.

Celmošanas algoritmi parasti ir salīdzinoši vienkārši, bet tie nespēj atpazīt sinonīmus, piemēram, "silts" un "karsts" būs divas atšķirīgas pazīmes, lai gan būtu vērts tās apvienot. Tāpat celmošana nespēj atpazīt homoformas. Piemēram, "dod roku", "roku bedri" un "klausos roku" visos šajos gadījumos celmošana vārdu "roku" uztvers kā vienu pazīmi, lai gan īstenībā tās ir trīs atšķirīgas pazīmes. Lai uzlabotu šo procesu, vārdu celmošanu var aizstāt ar vārdu lemmatizāciju (*lemmatization*).

Lemmatizācija vārdus pārveido to pamatformās, izmantojot vārdnīcas un morfoloģisko analīzi. Tas nozīmē, ka lietvārdi tiks pārveidoti vienskaitļa nominatīvā, darbības vārdi - nenoteiksmē utt. Lemmatizācijas procesā var tikt noteiktas vārdšķiras, piemēram, "dod roku" vārds "roku" ir lietots kā lietvārds, bet "roku bedri" tas ir darbības vārds. Sarežģītākie lemmatizācijas algoritmi spēj atpazīt sinonīmus, piemēram, vārdi "mašīna" un "automobilis" var tik uzskatīti kā viena pazīme. Lemmatizācijas algoritmi ir sarežģītāki un to darbības laiks nereti ir ievērojami ilgāks nekā celmošanas algoritmiem. Taču viennozīmīgi celmošana un lemmatizācija ievērojami samazina datu dimensiju skaitu.

Nodaļas aprakstā izmantoti [2], [20], [14], [3]

1.1.2. Pazīmju biežumu veidi

Kad veikta datu dimensiju skaita samazināšana, nepieciešams teksta datus pārveidot, lai būtu iespējams veikt klasifikāciju. Tekstu klasifikācijā bieži tiek pielietota "vārdu maisa" pieeja (*bag-of-words*). Šīs metodes būtība ir tāda, ka vārdu secība tekstā tiek ignorēta un tiek izveidota nesakārtota vārdu kopa.

Pieņemsim, ka $\mathcal{D}$ ir visu dokumentu kopa, ko nepieciešams klasificēt, un $\mathcal{F}$ ir visu sastopamo pazīmju (*features*) kopa. Pieņemsim, ka doti dokumenti $d_i \subseteq \mathcal{D}$, kur $i = 1, ..., n$, un pazīmes $f_j \subseteq \mathcal{F}$, kur $j = 1, ..., m$, tad datus var izkārtot matricā

1.1. tabula. **Dokumentu un pazīmju matrica**

	f_1	f_2	...	f_m
d_1	$a_{1,1}$	$a_{1,2}$	...	$a_{1,m}$
d_2	$a_{2,1}$	$a_{2,2}$	...	$a_{2,m}$
...	...	...	...	...
d_n	$a_{n,1}$	$a_{n,2}$	...	$a_{n,m}$

Vērtības $a_{i,j}$ var tik noteiktas dažādos veidos, bet parasti tas ir pazīmju/izteicienu biežums (*term frequency (TF)*). Pazīmju biežumam $TF_{i,j}$ ir dažādi veidi:

- Vienkāršākais variants ir saskaitīt, cik reizes pazīme f_j ir sastopama dokumentā d_i, tas ir,

$$TF_{i,j} = \sum_{j=1}^{|d_i|} \mathbf{I}_{\{f_j \in d_i\}},$$

  kur $\mathbf{I}_{\{f_j \in d_i\}}$ ir indikatorfunkcija, tas ir,

$$\mathbf{I}_{\{f_j \in d_i\}} = \begin{cases} 1 & \text{, ja } f_j \in d_i \\ 0 & \text{, ja } f_j \notin d_i \end{cases}$$

- Binārais pazīmju biežums:

$$TF_{i,j} = \mathbf{I}_{\{f_j \in d_i\}},$$

  tas parāda to, vai pazīme sastopama konkrētajā dokumentā vai ne neatkarīgi no tā, cik daudz reizes tā ir atkārtojusies.

- Logaritmiskais pazīmju biežums:

$$TF_{i,j} = 1 + ln\left(\sum_{j=1}^{|d_i|} \mathbf{I}_{\{f_j \in d_i\}}\right).$$

Inverso dokumentu biežumu (*inverse document frequency (IDF)*) definē

$$IDF_{j,\mathcal{D}} = ln\left(\frac{n}{\sum_{i=1}^{n} \mathbf{I}_{\{d_i \in \mathcal{D} | f_j \in d_i\}}}\right).$$

Inversais dokumentu biežums parāda, cik daudz informācijas konkrētais vārds apraksta, tas ir, vai konkrētā pazīme ir bieži vai reti sastopama visos dokumentos.

Bieži tiek izmantots pazīmju biežums - inversais dokumentu biežums (*Term frequency – Inverse document frequency (TF-IDF)*), ko definē

$$TFIDF_{i,j,\mathcal{D}} = TF_{i,j} IDF_{j,\mathcal{D}}.$$

TF-IDF novērtējums katrai pazīmei dos svaru, balstoties uz to, cik bieži dotā pazīme sastopama konkrētajā dokumentā un cik unikāla ir konkrētā pazīme. Ja dotā pazīme ir sastopama daudzos dokumentos, tad IDF vērtība būs mazāka, līdz ar to TF-IDF vērtība samazināsies. Piemēram, ja pazīme sastopama visos dokumentos, tad IDF un līdz ar to arī TF-IDF vērtība būs 0. Bet ja pazīme ir sastopama tikai dažos dokumentos, tad tās svars un TF-IDF vērtība būs lielāka.

Apraksts balstīts uz [16], [7]

1.1.3. N-grammas

"Vārdu maisa" pieejas trūkums ir tāds, ka tiek zaudēta vārdu secība teikumā. Piemēram, sentimentu analīzes problēmā "nav labi" tiktu uzskatīts par negatīvu attieksmi, bet, izmantojot vārdu maisa pieeju, šie vārdi tiek uzskatīti par divām atšķirīgām pazīmēm "nav" un "labi". Lai nepazaudētu šo vērtīgo informāciju, teksta analīzes uzdevumos bieži vien izmanto n-grammas (*n-grams*). N-gramma ir pēc kārtas sekojoši n vārdi, kas tiek uztverti kā viena pazīme. Piemēram, konstruējot bigrammas, "nav_labi" tiktu uzskatīta par vienu pazīmi.

Teksta klasifikācijas uzdevumos ne reti izmanto unigrammas kopā ar bigrammām. Dažreiz mēdz lietot arī trigrammas, taču lielāka skaita n-grammas parasti netiek izmantotas. Pirmkārt, izmantojot n-grammas, ļoti strauji pieaug pazīmju skaits, un tas var būt par iemeslu tam, ka mašīnmācīšanās metodes nespēs veikt kvalitatīvu dokumentu klasifikāciju. Otrkārt, lai konstruētu n-grammas un to biežums dažādos dokumentos būtu pietiekami liels, nepieciešama liela apmācības kopa.

1.2. Mašīnmācīšanās algoritmi teksta klasifikācijai

Kad veikta teksta datu tīrīšana, dimensiju skaita samazināšana un pazīmju biežumu novērtēšana, datiem var pielietot kādu no klasifikācijas algoritmiem.

1.2.1. Naivā Beijesa metode

Naivā Beijesa metode klasifikācijas uzdevumos balstās uz Beijesa nosacīto varbūtību formulu

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}.$$

Naivā Beijesa metode pieņem, ka dokuments $d \in \mathcal{D}$ pieder tai klasei $c^* \in \mathcal{C}$, kam izpildās

$$c^* = \max_{c \in \mathcal{C}} P(c|d) = \max_{c \in \mathcal{C}} \frac{P(d|c)P(c)}{P(d)}. \tag{1.1}$$

Izteiksmi (1.1) varam pārrakstīt

$$c^* = \max_{c \in \mathcal{C}} P(f_1, f_2, ..., f_m|c)P(c), \tag{1.2}$$

kur f_i ir dokumentos sastopamās pazīmes. Lai tiešā veidā atrisinātu šo uzdevumu, būtu nepieciešama ļoti liela apmācības kopa, lai korekti varētu noteikt lielo skaitu nepieciešamo varbūtību novērtējumu. Lai cīnītos ar šo problēmu, tiek pieņemts, ka pazīmes savā starpā ir neatkarīgas un ka vārdu secība dokumentā nav svarīga. Tā ir vārdu maisa pieeja. Tādā gadījumā varam izmantot sakarību

$$P(f_1, f_2, ..., f_m|c) = P(f_1|c)P(f_2|c)...P(f_m|c),$$

kas nozīmē to, ka mums nav jānovērtē visa kovariāciju matrica, bet gan tikai mainīgo dispersijas.

Līdz ar to iegūstam Naivo Beijesa klasifikatoru

$$c^* = \max_{c \in \mathcal{C}} P(c) \prod_{f \in \mathcal{F}} P(f|c).$$

Naivā Beijesa metode ir vienkāršs klasifikācijas algoritms, kas dažādos klasifikācijas uzdevumos sniedz ļoti labus rezultātus. Metodes priekšrocība ir tā, ka, lai sasniegtu labus klasificēšanas rezultātus, nav nepieciešama liela apmācības kopa.

1.2.2. Klasifikācijas koki

Klasifikācijas koki sadala sākotnējos novērojumus nešķeļošās kopās, balstoties uz kādu noteiktu kritēriju. Katru jauno apakškopu atkal var sadalīt mazākās kopās, nosakot

jaunu kritēriju. Tādā veidā tiek iegūta koka struktūra, kur gala mezgli norāda dokumenta grupu, bet zari norāda, kuras pazīmes tiek izmantotas, lai nokļūtu līdz konkrētajai grupai. Labi konstruēts klasifikācijas koks ļauj viegli klasificēt konkrētu dokumentu, novietojot to saknes mezglā jeb sākotnējā mezglā un vadot to pa grafu. Viegli uztveramais algoritms ir galvenā klasifikācijas koku priekšrocība.

Eksperimentu rezultāti rāda, ka klasifikācijas koki dokumentu klasifikācijas uzdevumos izmanto lielu skaitu pazīmju, kas ir apmācības kopas dokumentos. Tādēļ bieži vien klasifikācijas koku modelis ir pār - apmācījies (*over - fitting*) un testa kopas dokumentus klasificē nepareizi. Taču ja izmantoto pazīmju skaits nav pārāk liels, tad klasifikācijas koku metode strādā precīzāk.

1.2.3. Neironu tīkli

Teksta klasifikācijai var izmantot neironu tīklus. Neironu tīklu algoritms veidots, cenšoties kaut nedaudz atkārtot cilvēku smadzeņu darbību. Smadzenes uzņem informāciju, tad to apstrādā, pārvadot no viena neirona uz citu līdz pieņemts lēmums saistībā ar sākotnējo informāciju. Pēc līdzīga principa tiek konstruēti neironu tīklu algoritmi.

Šīs metodes priekšrocība ir tas, ka tā labi strādā situācijās, kad ir liels skaits dimensiju. Tāpat neironu tīkli spēj identificēt pretrunīgus, trokšņainus datus un mazināt to svaru klasifikācijas procesā. Taču galvenais neironu tīklu trūkums teksta klasifikācijas uzdevumos ir tāds, ka aprēķinu veikšanai nepieciešams ļoti liels atmiņas daudzums un jaudīgs procesors (*CPU*). Cits metodes trūkums ir sarežģītais algoritms, ko grūti izskaidrot citiem lietotājiem.

1.2.4. Gadījuma meži

Gadījuma mežu metodes būtība ir apvienot daudzus nekorelētus klasifikācijas kokus. Apmācības kopa tiek sadalīta apakškopās, un katrai no šīm apakškopām tiek konstruēts klasifikācijas koks. Iegūtie atsevišķie klasifikācijas koki tiek apvienoti gadījuma mežā.

Tā kā klasifikācijas koki tiek konstruēti mazākām dokumentu kopām, šie modeļi uzrāda labākus rezultātus nekā visai apmācības kopai konstruētais viens klasifikācijas koks. Gadījuma koki daudzās klasifikācijas problēmās uzrāda labus rezultātus. Taču gadījuma mežu metodes konstruēšanai nepieciešami lieli laika un datora jaudas resursi.

1.2.5. Atbalsta vektoru metode

Teksta klasifikācijas uzdevumos ļoti bieži tiek izmantota atbalsta vektoru metode. Daudzos darbos minēts, ka tieši atbalsta vektoru metode dod visprecīzākos rezultātus. Tam par iemeslu ir tas, ka šī metode labi strādā ļoti lielam dimensiju skaitam. Un kā jau minēts, liels dimensiju skaits ir raksturīga teksta klasifikācijas uzdevuma problēma. Atbalsta vektoru metode nosaka to, kuras pazīmes ir nozīmīgas konkrētajā klasifikācijas problēmā un nesvarīgās pazīmes tiek izslēgtas.

Atbalsta vektoru metode strādā ļoti labi gadījumos, kad grupu skaits, kurās nepieciešams saklasificēt dokumentus, nav liels. Ja grupu skaits ir salīdzinoši liels, tad atbalsta vektoru metodes realizēšanai nepieciešams ilgs laiks un liels atmiņas daudzums, lai veiktu aprēķinus. Optimālais grupu skaits ir divi. Šādām teksta klasifikācijas problēmām atbalstu vektoru metode uzrāda labus rezultātus, un aprēķiniem nav nepieciešami lieli laika un jaudas resursi.

Apraksts balstīts uz [2], [16], [4],[19]

1.3. Klasifikācijas modeļu efektivitātes novērtēšana divu klašu gadījumā

Lai konstruētu mašīnmācīšanās algoritmu dokumentu klasificēšanai, nepieciešams zināt to, kurā klasē atbilstošais dokuments ir iekļauts. Ir nepieciešama tā sauktā apmācības kopa (*training set*). Izmantojot šos datus, modelis tiek apmācīts un, balstoties uz apmācīto modeli, var klasificēt datus, kam nav zināma patiesā klase. Taču apmācības kopas izveide ir ļoti laikietilpīgs un dārgs process, jo parasti tas notiek, manuāli lasot datus un novērtējot to, kurā klasē konkrēto dokumentu iekļaut. Taču, lai apmācītu klasifikācijas modeli, precīzas apmācības kopas izveide ir vitāli svarīga. Lai arī teksta datiem bieži vien tiek izmantota klāsteranalīze, taču to parasti neizmanto, lai izveidotu apmācības kopu, jo apmācības kopai svarīgi būt maksimāli precīzai, bet klāsteranalīze kā jau jebkurš mašīnmācīšanās algoritms strādā ar zināmu kļūdu.

Kad izstrādāts klasifikācijas modelis, svarīgi novērtēt to, cik labi iegūtais modelis strādā. Ja konstruēti vairāki modeļi, nepieciešams noteikt to, kurš ir precīzāks. Lai to izdarītu, pirms klasifikācijas modeļa apmācīšanas daļu no apmācības kopas datiem iekļauj testa kopā (*testing set*). Parasti 80% datu tiek iekļauti apmācības kopā un atlikušie 20% dokumentu tiek iekļauti testa kopā.

Kad iegūts klasifikācijas modelis, tiek pieņemts, ka testa kopas datiem nav zināma

patiesā klase, un šie dati tiek klasificēti ar apmācīto modeli. Iegūtās modeļa vērtības tiek salīdzinātas ar patiesajām vērtībām.

Pieņemsim, ka mums ir divas klases. Vienu no tām nosauksim par pozitīvo klasi, bet otru par negatīvo grupu. Līdz ar to ir iespējami četri gadījumi, skatīt tabulu 1.2.

1.2. tabula. **Modeļa rezultāti un patiesie stāvokļi**

		Modeļa rezultāti	
		Pozitīvs	Negatīvs
Patiesais stāvoklis	Pozitīvs	TP	FN
	Negatīvs	FP	TN

Ja klasifikācijas modelis norāda uz pozitīvo stāvokļu grupu un arī patiesā stāvokļa klase ir pozitīva, tad to sauc par *patiesi pozitīvu TP (true positive)*. Ja patiesais stāvoklis ir pozitīvs, bet modelis klasificē negatīvajā klasē, tad to sauc par *aplami negatīvu FN (false negative)*. Analoģiski definē *aplami pozitīvu FP (false positive)* un *patiesi negatīvu TN (true negative)*.

Ir ieviesti dažādi novērtējumi, kas ļauj noteikt to, cik labi konkrētā klasifikācijas metode strādā. Viens no šādiem novērtējumiem ir *jutīgums SE (sensitivity)*. Tā ir varbūtība, ka klasifikācijas modelis ir pareizi identificējis pozitīvās klases, tas ir,

$$SE = \frac{TP}{TP + FN} = TPR. \tag{1.3}$$

Jūtīgumu sauc arī par *patiesi pozitīvo rādītāju TPR (true positive rate)*.

Cits bieži lietots mērs ir

$$PPV = \frac{TP}{TP + FP}, \tag{1.4}$$

ko sauc par *precision*. Tas parāda, cik liela daļa no visiem dokumentiem, ko modelis klasificējis kā pozitīvu, patiešām ir pozitīvās grupas dokumenti.

Cits svarīgs novērtējums ir *specifiskums SP (specificity)* jeb *patiesi negatīvais rādītājs TNR (true negative rate)*. Tā ir varbūtība, ka klasifikācijas modelis pareizi atpazīst negatīvos stāvokļus

$$SP = \frac{TN}{FP + TN} = TNR. \tag{1.5}$$

Testa efektivitāti parāda

$$EFF = \frac{TP + TN}{TP + FP + FN + TN}, \tag{1.6}$$

kas ir pareizi klasificēto dokumentu skaits pret visiem.

Pirmajā acumirklī varētu šķist, ka pietiek novērtēt vienīgi efektivitāti (1.6), taču šādi var palaist garām ļoti svarīgu informāciju par modeli. Pieņemsim, ka dots divu klašu klasifikācijas uzdevums, kur pirmā grupa satur 99% dokumentu, un attiecīgi otra grupa satur tikai 1% no visiem dokumentiem. Ja klasifikācijas modelis visus dokumentus klasificē pirmajā grupā, tad efektivitātes novērtējums ir $EFF = 99\%$. Šajā gadījumā arī patiesi pozitīvais rādītājs TPR ir 99%. Taču skaidrs, ka nav jēgas no modeļa, kas visus dokumentus klasificē vienā grupā. Bet novērtējot patiesi negatīvo rādītāju TNR, iegūstam 1%. Šī iemesla dēļ, pētot modeļu klasificēšanas spējas, ieteicams novērtēt gan patiesi pozitīvo rādītāju, gan patiesi negatīvo rādītāju, gan efektivitāti.

Bieži vien modeļa darbības novērtēšanai tiek konstruēts F-mērs, kas sabalansē SE un PPV, tas ir,

$$F_\beta = \left(1 + \beta^2\right) \frac{PPV * SE}{\beta^2 PPV + SE},$$

kur $\beta > 0$. Parasti izmanto F-mēru pie $\beta = 1$.

Bieži vien sastopami tādi klasifikācijas uzdevumi, kur vitāli svarīgi ir precīzāk atpazīt vienu no klasēm. Piemēram, šāds uzdevums ir e-pastu klasifikācija, lai izdalītu surogātpastu no ienākošajiem e-pastiem. Šādā gadījumā ir svarīgāk, lai klasifikācijas modelis precīzāk atpazīst derīgos e-pastus, jo kļūda, ka derīgs e-pasts tiks iekļauts surogātpastā, ir daudz dārgāka nekā kļūda pretējā gadījumā. Šādos gadījumos nereti tiek upurēta modeļa kopējā precizitāte.

Apraksts balstīts uz [16]

2. Atbalsta vektoru metode

2.1. Atbalsta vektoru metode pilnīgi atdalāmiem klašu sadalījumiem

Atbalsta vektoru metode (*Support Vector Machines*) ir viens no populārākajiem mašīnmācīšanās algoritmiem datu klasifikācijas veikšanai. Šo metodi ieviesa Padomju Savienības matemātiķi Vladimirs Vapniks un Aleksejs Červonenskis 1963. gadā. Deviņdesmito gadu sākumā Vladimirs Vapniks pārcēlās uz ASV, kur viņš, sadarbojoties ar datorzinātnes pētnieci Korinu Kortes, kura šobrīd ir Google Pētniecības nodaļas vadītāja Ņujorkā, ieviesa kodolu substitūcijas pielietošanu atbalsta vektoru metodei.

Atbalsta vektoru metode tiek plaši izmantota dažādu klasificēšanas uzdevumu veikšanai tādās nozarēs kā bioloģija, medicīna, ķīmija, ekonomika u.c. Atbalsta vektoru metodi izmanto ne tikai teksta klasificēšanai, bet arī attēlu atpazīšanai un klasificēšanai, rokraksta atpazīšanai u.c.

Pieņemsim, ka doti N novērojumi x_n, kur $n = 1, ..., N$ un $x_n \in \mathbb{R}^m$. Pieņemsim arī, ka katrs novērojums pieder vienai no divām klasēm $\mathcal{C}_1$ vai $\mathcal{C}_2$ un katram novērojumam tiek piekārtota viena no mērķa vērtībām $t_n \in \{-1, 1\}$, kur

$$t_n = \begin{cases} 1 & \text{, ja } x_n \in \mathcal{C}_1 \\ -1 & \text{, ja } x_n \in \mathcal{C}_2 \end{cases}$$

Apskatīsim lineāru modeli

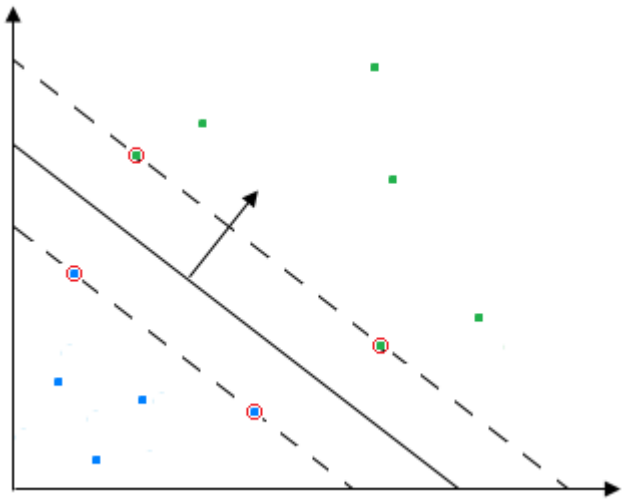
$$y(x) = w^T \phi(x) + b, \tag{2.1}$$

kur $\phi(x)$ apzīmē fiksētu pazīmju telpas transformāciju. Ievērosim, ka dotais modelis (2.1) ir lineārs attiecībā pret nezināmajiem parametriem w un b.

Apmācības kopa satur novērojumus $x_1, ..., x_N$ ar zināmām mērķa vērtībām $t_1, ..., t_N$. Savukārt, jaunie novērojumi x tiek klasificēti atbilstoši $y(x)$ zīmei, tas ir,

$$t = \begin{cases} 1 & \text{, ja } y(x) > 0 \\ -1 & \text{, ja } y(x) < 0 \end{cases}$$

Pieņemsim, ka apmācības kopas novērojumi ir lineāri atdalāmi pazīmju telpā un ka eksistē hiperplakne, kas nodala vienas klases novērojumus no otras klases novērojumiem. Atbalsta vektoru metodes būtība ir atrast tādu hiperplakni, kas maksimizē attālumu starp hiperplakni un katras klases tuvāko punktu, skatīt attēlu 2.1.



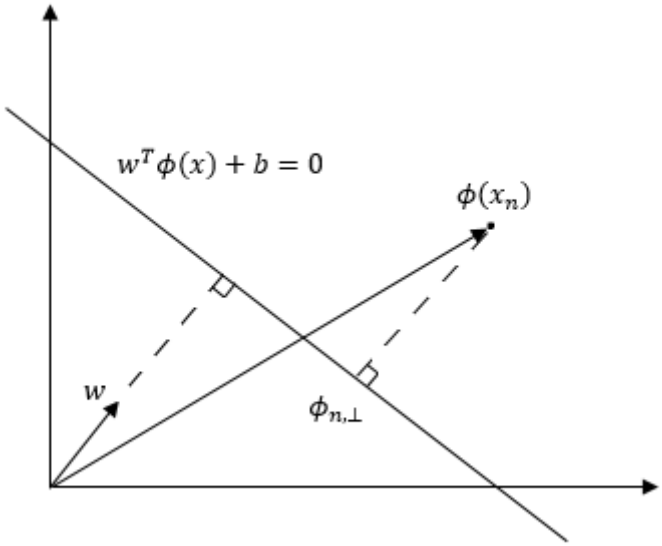
2.1. att. **Novērojumu nodalīšana ar atbalsta vektoru metodi divu dimensiju gadījumā**

Pieņemsim, ka eksistē vismaz viena parametra w un vismaz viena parametra b vērtība tāda, ka funkcija (2.1) apmierina nosacījumus $y(x_n) > 0$, ja $t_n = 1$, un $y(x_n) < 0$, ja $t_n = -1$. Tas nozīmē, ka $t_n y(x_n) > 0$ visiem apmācības kopas novērojumiem.

Ar $\phi_{n,\perp}$ apzīmēsim punkta $\phi(x_n)$ projekciju uz

$$y(x) = w^T \phi(x) + b = 0. \tag{2.2}$$

Punkta $\phi(x_n)$ attālums līdz hiperplaknei (2.2) ir $\|\phi(x_n) - \phi_{n,\perp}\|$, skatīt attēlu 2.2.



2.2. att. **Punkta $\phi(x_n)$ projekcija uz hiperplakni $w^T\phi(x) + b = 0$.**

Tā kā $\phi_{x_n,\perp}$ pieder hiperplaknei (2.2), tad $w^T\phi(x_{n,\perp}) + b = 0$. Tātad

$$y(x_n) = w^T\phi(x_n) + b - \left(w^T\phi(x_{n,\perp}) + b\right) = w^T\phi(x_n) - w^T\phi(x_{n,\perp}) = w^T\left(\phi(x_n) - \phi(x_{n,\perp})\right).$$

No tā seko, ka attālums ir

$$\|\phi(x_n) - \phi(x_{n,\perp})\| = \frac{|y(x_n)|}{\|w\|}.$$

Tā kā mūs interesē tikai tās hiperplaknes, kas visus datu punktus korekti saklasificē, tad katram n jāizpildās $t_n y(x_n) > 0$. Tātad attālums starp punktu x_n un atdalošo hiperplakni ir

$$\frac{|y(x_n)|}{\|w\|} = \frac{t_n y(x_n)}{\|w\|} = \frac{t_n\left(w^T\phi(x_n) + b\right)}{\|w\|}. \tag{2.3}$$

Mūsu mērķis ir maksimizet attālumu (2.3), citiem vārdiem sakot, mērķis ir atrast parametrus w un b, lai maksimizētu tuvākā punkta attālumu līdz atdalošajai virsmai, tas ir,

$$\max_{w,b}\left\{\min_n \frac{t_n\left(w^T\phi(x_n) + b\right)}{\|w\|}\right\},$$

ko var pārrakstīt

$$\max_{w,b}\left\{\frac{1}{\|w\|}\min_n\left\{t_n\left(w^T\phi(x_n) + b\right)\right\}\right\}, \tag{2.4}$$

jo w nav atkarīgs no n.

Optimizācijas uzdevuma (2.4) atrisināšana būtu ļoti komplicēta, tāpēc to pārvērš

ekvivalentā problēmā, ko var vieglāk atrisināt. Ievērosim, ka parametru w un b transformācija $w \to kw$ un $b \to kb$, kur k ir pozitīvs skaitlis, neietekmē attāluma novērtējumu starp punktu x_n un atdalošo hiperplakni. Tātad varam pieņemt, ka

$$t_n \left(w^T \phi(x'_n) + b \right) = 1, \tag{2.5}$$

kur ar x'_n apzīmēsim tādu punktu x_n, kas ir vistuvāk atdalošajai hiperplaknei. Tādā gadījumā visiem punktiem x_n izpildās

$$t_n \left(w^T \phi(x_n) + b \right) \geq 1, \tag{2.6}$$

kur $n = 1, ..., N$.

Ņemot vērā (2.5) un (2.6) optimizācijas uzdevums (2.4) ir pārrakstāms $\max_{w,b} \|w\|^{-1}$, kas ir ekvivalenti $\min_{w,b} \|w\|^2$, ko varam pārrakstīt

$$\min_{w,b} \frac{1}{2} \|w\|^2. \tag{2.7}$$

Tātad optimizācijas uzdevums (2.4) ir reducējies uz (2.7) pie nosacījumiem (2.6). Šo optimizācijas uzdevumu var atrisināt ar Lagranža reizinātāju metodes palīdzību. Tātad nepieciešams minimizēt Lagranža funkciju

$$L\left(w,b,a\right) = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \left(t_n \left(w^T \phi(x_n) + b \right) - 1 \right),$$

kur $a = (a_1, ..., a_n)^T$ ir Lagranža reizinātāji un $a_n \geq 0$ katram n.

Meklējam parciālos atvasinājumus pēc parametriem w un b un pielīdzinām tos 0, tas ir,

$$\frac{\partial L}{\partial w} = w - \sum_{n=1}^{N} a_n t_n \phi(x_n) = 0$$

un parciālais atvasinājums pēc parametra b ir

$$\frac{\partial L}{\partial b} = \sum_{n=1}^{N} a_n t_n = 0.$$

Izmantojot iegūtās parciālo atvasinājumu izteiksmes, Lagranža funkciju varam pārrakstīt

$$\begin{aligned} L\left(w,b,a\right) &= \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n t_n \left(w^T\phi(x_n) + b\right) + \sum_{n=1}^{N} a_n = \\ &= \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(x_n,x_m) = \widehat{L}(a) \end{aligned} \tag{2.8}$$

pie nosacījumiem

$$a_n \geq 0,$$

kur $n = 1,...,N$, un

$$\sum_{n=1}^{N} a_n t_n = 0,$$

un

$$k\left(x_n,x_m\right) = \phi(x_n)^T\phi(x_m). \tag{2.9}$$

Mēs optimizācijas problēmu (2.7) esam pārveidojuši duālajā optimizācijas problēmā (2.8). Lai gan problēma (2.8) satur vairāk mainīgos, līdz ar to tās atrisināšanai nepieciešams ilgāks laiks, esam ieguvuši iespēju bāzes funkciju vietā $\phi(x)$ izmantot kodolu funkciju $k(x,x')$. No (2.9) seko, ka kodolu funkcijai $k(x,x')$ jābūt pozitīvi definītai, jo tas nozīmē, ka Lagranža funkcija $\hat{L}(a)$ ir ierobežota, kas ir viens no optimizācijas problēmas nosacījumiem, lai tai eksistētu ekstremālā vērtība.

Izteiksmi (2.2) varam pārrakstīt, izmantojot parametrus $\{a_n\}$ un kodolu funkciju, iegūstot

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x,x_n) + b. \tag{2.10}$$

Lai klasificētu jaunos datus, izmantojot modeli, kas izstrādāts, balstoties uz apmācības kopu, izmanto klasifikatoru (2.10), novērtējot $y(x)$ zīmi.

Ir pierādīts, ka optimizācijas problēmai (2.8) jāapmierina šādi nosacījumi,

$$a_n \geq 0,$$

$$t_n y(x_n) - 1 \geq 0,$$

$$a_n\left(t_n y(x_n) - 1\right) = 0.$$

Pierādījumu var atrast Kristofera Bišopa grāmatā [5].

Līdz ar to katram datu punktam vai nu $a_n = 0$, vai $t_n y(x_n) = 0$. Tātad tie punkti,

kur $a_n = 0$, neietilps summā (2.10), kas nozīmē to, ka klasifikācijas uzdevums nav atkarīgs no šiem punktiem. Bet atlikušie punkti, kam izpildās $t_n y(x_n) = 1$, tiek saukti par atbalsta vektoriem. Skatīt attēlu 2.1, kur atbalsta vektori ir iezīmēti.

Kad atrastas $\{a_n\}$ vērtības, parametru b novērtē, izmantojot

$$t_n y(x_n) = t_n \left(\sum_{m \in S} a_m t_n k(x_n,x_m) + b \right) = 1,$$

kur S ir atbalsta vektoru indeksu kopa. Izvēloties patvaļīgu atbalsta vektoru x_n, var iegūt, ka

$$b = \frac{1}{N_S} \sum_{n \in S} \left(t_n - \sum_{m \in S} a_m t_m k(x_n,x_m) \right),$$

kur N_S ir atbalsta vektoru skaits.

2.2. Biežāk izmantotās kodolu funkcijas atbalsta vektoru metodē

Iepriekšējā nodaļā esam pamatojuši to, ka atbalsta vektoru metodē var tikt izmantotas kodolu funkcijas. Tas nozīmē, ja dotie datu punkti nav lineāri atdalāmi dotajā telpā, mēs varam tos transformēt citā telpā, kur šos datus jau būs iespējams sadalīt.

Apskatīsim dažas no biežāk izmantotajām kodolu funkcijām.

Polinomiālais kodols

$$k(x_i,x_j) = (x_i x_j + a)^b,$$

kur $a,b \in \mathbb{R}$. Pie tam, ja $a = 0$ un $b = 1$ iegūst lineāro kodolu $k(x_i,x_j) = x_i x_j$.

Gausa kodols, literatūrā sastopams arī nosaukums *Radial basis* kodols

$$k(x_i,x_j) = e^{-\left(\frac{\|x_i - x_j\|^2}{2\sigma^2} \right)}.$$

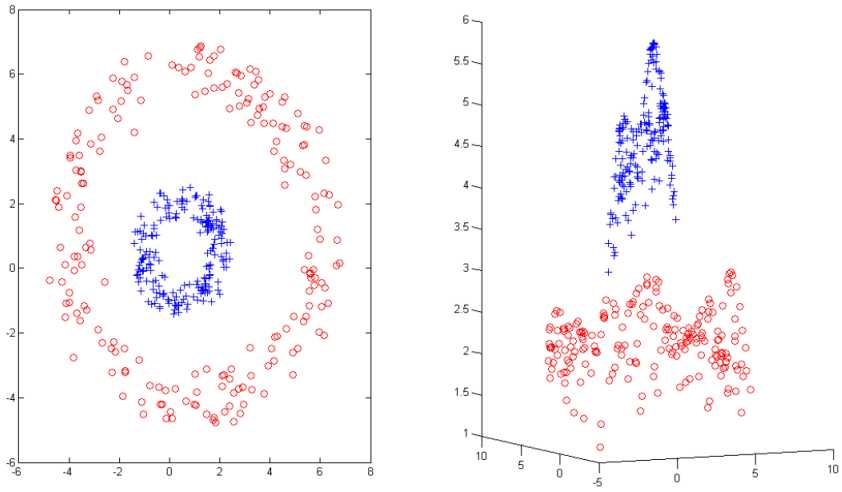
Sigmoidal kodols

$$k(x_i,x_j) = tanh(a x_i x_j - b),$$

kur $a,b \in \mathbb{R}$ un $tanh(x)$ ir hiperboliskais tangenss.

Attēlā 2.3 ilustrēta atbalsta vektoru metode, izmantojot kodolu funkciju, kur skaidri redzams, ka sākotnējā pazīmju telpā klases nav lineāri atdalāmas, bet pielietojot kodolu

funkciju, iegūstam citu telpu, kurā iespējams abu grupu datu lineāra atdalīšana.

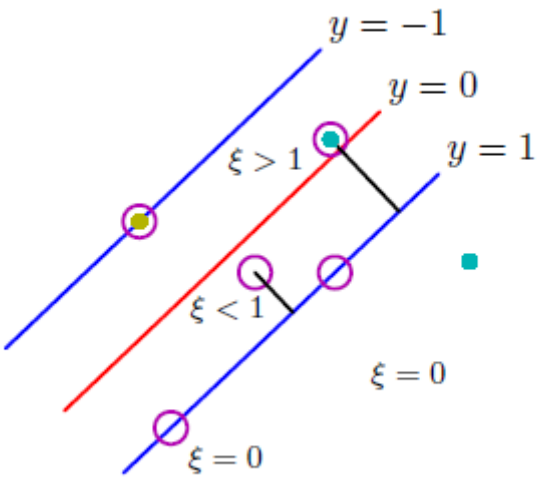


2.3. att. **Atbalsta vektoru metode divu dimensiju gadījumā, izmantojot Gausa kodolu. Ilustrācija no darba [11]**

2.3. Atbalsta vektoru metode gadījumā, kad abu klašu sadalījumi pārklājas

Līdz šim bijām pieņēmuši, ka datus ir iespējams lineāri sadalīt vai nu dotajā pazīmju telpā, vai nu telpā, ko ieguvām ar kodolu funkciju transformāciju. Taču bieži vien datus nav iespējams pilnībā nodalīt, vai arī tas ir iespējams, bet iegūtais modelis būs pār - apmācījies, tāpēc jaunie dati tiks slikti klasificēti.

Lai cīnītos ar šo problēmu, atbalsta vektoru metodi modificē, lai ļautu dažiem apmācības kopas datiem būt iekļautiem nepareizajā klasē. Ieviesīsim papildus mainīgo $\xi_n \geq 0$ katram apmācības kopas punktam. Ja punkts x_n ir pareizi klasificēts un ir salīdzinoši tālu no atdalošās hiperplaknes, tas ir, $t_n y(x_n) \geq 1$, tad $\xi_n = 0$, un $\xi_n = |t_n - y(x_n)|$ pārējiem punktiem. Tas nozīmē, ja punkts x_n atrodas uz atdalošās hiperplaknes, tad $y(x_n) = 0$, līdz ar to $\xi_n = 1$. Punktiem, kas klasificēti pareizi, bet ir tuvu atdalošai hiperplaknei, tas ir, $0 < t_n y(x_n) < 1$, mainīgā vērtība ir $0 < \xi_n < 1$. Bet punktiem, kas ir nepareizi klasificēti, mainīgā vērtība ir $\xi_n > 1$. Skatīt attēlu 2.4.



2.4. att. **Papildus mainīgā ξ vērtības divu dimensiju gadījumā atkarībā no punkta novietojuma pret atdalošo taisni. Ilustrācija no darba [5]**

Šajā gadījumā optimizācijas uzdevuma nosacījums (2.5) tiek aizstāts ar

$$t_n y(x_n) \geq 1 - \xi_n, \tag{2.11}$$

kur $n = 1, ..., N$ un $\xi_n \geq 0$.

Mūsu mērķis vēl joprojām ir maksimizēt attālumu, un papildus tam nepieciešams minimizēt nepareizi klasificētos punktus. Līdz ar to nepieciešams minimizēt sekojošu izteiksmi

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|w\|^2, \tag{2.12}$$

kur parametrs $C > 0$. Tātad uzdevums ir minimizēt izteiksmi (2.12), ņemot vērā ierobežojumus (2.11). Konstruējam Lagranža funkciju

$$L(w,b,a) = \frac{1}{2}\|w\|^2 + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n \left(t_n y(x_n) - 1 + \xi_n\right) - \sum_{n=1}^{N} \mu_n \xi_n, \tag{2.13}$$

kur $\{a_n \geq 0\}$ un $\{\mu_n \geq 0\}$ ir Lagranža reizinātāji, ko līdzīgi kā iepriekš var pārvērst par

$$\widehat{L}(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n,x_m) \tag{2.14}$$

ar ierobežojumiem

$$0 \leq a_n \leq C,$$

$$\sum_{n=1}^{N} a_n t_n = 0.$$

Klasifikācijas funkcija, lai noteiktu klasi jaunajiem punktiem, ir līdzīga kā iepriekš

$$y(x) = \sum_{n=1}^{N} a_n t_n k(x,x_n) + b.$$

Lai novērtētu parametra C vērtību, praksē parasti tiek izmantota krosvalidācija.

2.4. Atbalsta vektoru metode $K > 2$ klašu gadījumā

Atbalsta vektoru metodi izmanto arī tādās problēmās, kur nepieciešama klasifikācija vairāk nekā tikai divās grupās. Apskatītā atbalsta vektoru metode netiek vispārināta vairāku klašu problēmai. Tā vietā $K > 2$ klašu klasifikācijas uzdevums tiek sadalīts vairākās divu klašu klasifikācijas problēmās, ko var atrisināt ar atbalsta vektoru metodi.

Pieņemsim, ka klasifikācijas problēma satur $K > 2$ klases. Viens veids, kā atrisināt šo problēmu, ir vienas klases visus elementus uztvert kā pozitīvos elementus, tas ir, pieņemt, ka tie pieder klasei $\mathcal{C}_1$, bet visus atlikušos $K - 1$ klašu elementus uztvert kā klases $\mathcal{C}_2$ elementus. Šo metodi sauc par viens-pret-visiem pieeju (*one-versus-the-rest*). Jaunie datu punkti tiek klasificēti pēc nosacījuma $y(x) = \max_k y_k(x)$.

Cita pieeja ir apmācīt $\frac{K(K-1)}{2}$ dažādus divu klašu atbalsta vektoru metodes modeļus, kur apskatīti visi iespējamie klašu pāri. Šo pieeju sauc par viens-pret-vienu pieeju (*one-versus-one*). Ja klašu skaits K ir salīdzinoši liels, tad šīs problēmas apmācīšana un testa kopas klasificēšana prasa daudz vairāk laika nekā viens-pret-visiem pieeja. Taču darbā [13] ir parādīts, ka viens-pret-vienu pieeja uzrāda labāku precizitāti par viens-pret-visiem pieeju.

Nodaļas apraksts balstīts uz [5], [11], [8], [13]

3. Angļu valodas teksta klasifikācija

Vispirms apskatīsim angļu valodas teksta klasifikāciju. Izmantosim *NYTimes* datu kopu. Tā satur "*The New York Times*" ziņu virsrakstus, kas ir klasificēti 27 tēmās. Tā kā darba mērķis ir pētīt divu klašu teksta klasifikācijas problēmas, tad atlasām tikai divu tēmu datus. Izvēlamies tās tēmas, kas satur visvairāk datu.

Teksta klasifikācija tiek veikta programmā R [21], izmantojot pakas *RTextTools* [22],[23], *tm* [12], *SnowballC* [6], *wordcloud* [10], *e1071* [17], *randomForest* [24].

3.1. Dimensiju skaita samazināšana

Klasifikācijas veikšanai atlasīti divu tēmu dati, kur pirmā tēma satur 662 dokumentus jeb ziņu virsrakstus, bet otrā tēma satur 185 ziņu virsrakstus. Vienu no tēmām varētu nosaukt par "Ārpolitika", bet otru - "Veselība". Konstruējam sabalansētu kopu, tas ir, iekļaujam vienāda skaita ziņu virsrakstus par abām tēmām.

3.1. tabula. **Dokumentu skaits atkarībā no tēmām**

	Sākotnējais dokumentu skaits	Dokumentu skaits sabalansētā kopā
1. tēma	662	185
2. tēma	185	185

Vienai no grupām piešķirsim mērķa vērtību $t_1 = 1$, šo klasi nosauksim par pozitīvo klasi, bet otrai grupai $t_2 = -1$, attiecīgi šo klasi nosauksim par negatīvo.

Veicam datu dimensiju skaita samazināšanu. Vispirms izņemam skaitļus, pieturzīmes un angļu valodas stopvārdus. Izmantojam stopvārdu sarakstu, kas pieejams pakā *tm*. Tālāk veicam vārdu celmošanu, izmantojot Portera algoritmu, skatīt piemērus tabulā 3.2. Pirmajā kolonnā ir oriģinālais teksts bez jebkādas apstrādes. Otrajā kolonnā ir teksts, kam visi burti ir pārvērsti par mazajiem burtiem, izņemti skaitļi, pieturzīmes un angļu valodas stopvārdi. Trešajā kolonnā ir teksts pēc celmošanas.

3.2. tabula. **Teksta datu piemēri pirms un pēc apstrādes un celmošanas**

Oriģinālais teksts	Teksts pēc mazsvarīgo pazīmju izņemšanas	Teksts pēc mazsvarīgo pazīmju izņemšanas un celmošanas
State Referendums Seeking to Overhaul Health Care System	state referendums seeking overhaul health care system	state referendum seek overhaul health care system
Scientists Find Enzyme Linked To Alzheimer's	scientists find enzyme linked alzheimers	scientist find enzym link alzheim
Heeding a Call to Test Breast Cancer Treatments	heeding call test breast cancer treatments	heed call test breast cancer treatment
Falling Yen Puts Car Makers In Japan in the Driver's Seat	falling yen puts car makers japan drivers seat	fall yen put car maker japan driver seat
Russian Astronauts Insist Errors, While Human, Were on Earth	russian astronauts insist errors human earth	russian astronaut insist error human earth
2 MAJOR HOSPITALS FORM CORPORATION	major hospitals form corporation	major hospit form corpor

Varam ievērot, ka pēc celmošanas lietvārdi daudzskaitlī ir kļuvuši par vārdiem vienskaitlī, piemēram, "*scientists*" pārvērsts par "*scientist*", "*referendums*" par "*referendum*" u.c. Pēc celmošanas vārdiem tiek noņemtas galotnes, bieži vien atstājot tikai vārdu saknes, piemēram, "*puts*" pārveidots par "*put*", "*seeking*" par "*seek*", vārdam "*hospital*" tiek atstāta tikai vārda sakne "*hospit*", kas angļu valodā vispār nav vārds, bet tā kā mūsu mērķis ir pārveidot datus priekš klasifikācijas modeļa, mums nevajadzētu satraukties par to, ka iegūstam vārdu bez nozīmes konkrētajā valodā.

Rezultātā esam ievērojami samazinājuši dimensiju skaitu, skatīt tabulu 3.3.

3.3. tabula. **Pazīmju skaits pirms un pēc datu apstrādes**

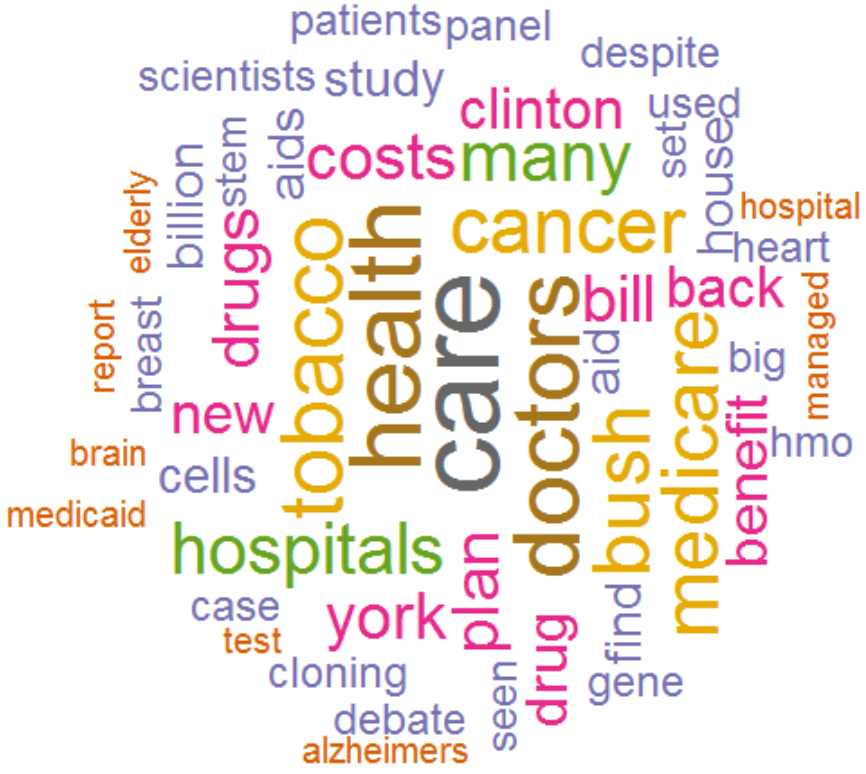
Datu apstrādes veids	Pazīmju skaits
Oriģinālais teksts	1486
Bez mazsvarīgajām pazīmēm	1330
Pēc celmošanas	1108

Neapstrādātie teksta dati satur 1486 unikālas pazīmes. Kad izņemam skaitļus, pieturzīmes un stopvārdus, dimensiju skaits ievērojami samazinās. Angļu valodā diezgan bieži ir sastopami apostrofi. Tā ir komatveida zīme virs vārda. Piemēram, vārdi "*doctors*" un "*doctor's*" pirms pieturzīmju izņemšanas tiks uzskatīti par divām dažādām pazīmēm, bet pēc tam - par vienu.

Pēc celmošanas pazīmju skaits ir samazinājies vēl vairāk. Kopumā ir izdevies pazīmju skaitu samazināt par aptuveni 25% salīdzinājumā ar sākotnējo pazīmju skaitu.

Apskatīsim pirmās un otrās grupas biežāk sastopamās pazīmes jeb vārdus. To darām, konstruējot vārdu mākoņus, kur vārdu izmērs un krāsa ir atkarīga no tā, cik bieži konkrētā pazīme ir sastopama atbilstošās klases dokumentos. Vārdu mākoņus konstruējam tekstam bez maznozīmīgajām pazīmēm un bez celmošanas. Skatīt attēlus 3.1 un 3.2.

3.1. att. **Pirmās klases 50 biežāk sastopamo vārdu mākonis**



3.2. att. **Otrās klases 50 biežāk sastopamo vārdu mākonis**

Apskatot vārdu mākoņus, redzam, ka lielākā daļa pazīmju ir raksturīgas vienai no klasēm. Tikai neliela daļa no biežāk sastopamajiem vārdiem ir raksturīgi abām grupām. Jāatzīmē, ka vārdu mākoņos mēs apskatām tikai 50 biežāk sastopamos vārdus. Starp parējām pazīmēm ir ievērojami lielāks īpatsvars vārdu, kas sastopami abās grupās. Parasti tie ir ikdienā bieži lietotie vārdi.

3.2. Klasifikācijas metožu salīdzinājums

Tālāk veiksim datu klasifikāciju. 80% no dokumentiem jeb 296 ziņu virsrakstus iekļausim apmācības kopā. Balstoties uz šiem datiem, tiks apmācīts klasifikācijas modelis. Atlikušie 20% dokumentu, tas ir, 74 ziņu virsraksti, tiks iekļauti testa kopā. Šiem datiem pielietosim izveidoto klasifikācijas modeli, kurš katru dokumentu iekļaus vienā no divām grupām. Iegūtos modeļa rezultātus salīdzināsim ar patiesajām vērtībām.

Veicam klasifikāciju ar atbalsta vektoru metodi, klasifikācijas koku un gadījuma mežu metodi. Darba gaitā tika izmantots arī neironu tīklu modelis, bet to nebija iespējams realizēt programmā R, jo tika izdots paziņojums, ka programma nespēj veikt nepieciešamos aprēķinus lielā datu dimensiju skaita dēļ.

Tāpat darba gaitā veikta klasifikācija ar naivo Beijesa metodi, taču iegūtie rezultāti nevienā no gadījumiem nesasniedza pat 50% precizitāti. Darbā [3] minēts, ka naivā Beijesa metode uzrāda labākus rezultātus problēmām, kur viena no klasēm ir biežāk sastopama nekā citas grupas. Šī iemesla dēļ naivā Beijesa metode tika veikta ne tikai sabalansētai apmācības kopai, bet arī nesabalansētai. Taču iegūtie rezultāti tik un tā nepārsniedza 50% precizitāti.

Atbalsta vektoru metodi konstruējam pieņemot, ka abu klašu sadalījumi var pārklāties, tas ir, konstruējot atdalošo hiperplakni, pieļausim to, ka daži novērojumi tiks iekļauti nepareizajā klasē. Parametru C (2.12), kurš nosaka svaru tam, cik daudzi novērojumi var tik nodalīti nepareizi, iegūstam ar krosvalidācijas metodi. Izmantojam lineāro kodolu.

Klasifikāciju veicam gan oriģinālajiem datiem, gan datiem, kam izņemtas maznozīmīgās pazīmes, gan datiem bez mazsvarīgajām pazīmēm un pēc celmošanas. Visus modeļus konstruējam, izmantojot gan pazīmju biežumu TF, gan bināro TF, gan normalizēto pazīmju biežumu - inverso dokumentu biežumu TF-IDF.

Vispirms konstruējam patiesi pozitīvo rādītāju (1.3) un patiesi negatīvo rādītāju (1.5), lai novērtētu, cik precīzi modeļi klasificē katru no klasēm. Rezultāti apkopoti tabu-

lās 3.4 un 3.5.

3.4. tabula. **Pareizi klasificētā pozitīvo klašu daļa TPR (1.3) atkarībā no pazīmju biežumu veida, izmantojot atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Oriģinālais teksts	SVM	0.857	0.853	0.816
	TREE	0.842	0.842	0.789
	RF	0.885	0.857	0.840
Bez mazsvarīgajām pazīmēm	SVM	0.909	0.909	0.912
	TREE	1.000	1.000	0.944
	RF	1.000	1.000	0.957
Bez mazsvarīgajām pazīmēm un pēc celmošanas	SVM	0.886	0.886	0.890
	TREE	0.952	0.952	0.905
	RF	0.964	0.963	0.962

Apskatot tabulu 3.4, redzam, ka pozitīvo jeb pirmo grupu visi apskatītie modeļi klasificē diezgan labi. Klasificējot oriģinālo tekstu ar TF biežumu, visi modeļi aptuveni 85% pirmās grupas dokumentus ir pareizi saklasificējuši. Bet ja apskatam tekstu pēc dimensiju skaita samazināšanas, redzam, ka gan ar TF, gan bināro TF, gan TF-IDF biežumu klasifikācijas koki un gadījuma meži pirmās grupas datus ir pareizāk saklasificējuši nekā atbalsta vektoru metode. Gadījuma meži un klasifikācijas koki, izmantojot TF un bināro TF biežumu un tekstu bez mazsvarīgajām pazīmēm, visus pirmās grupas datus ir atpazinuši pareizi.

Bet tagad pievērsīsim uzmanību tam, cik precīzi ir saklasificēti otrās grupas ziņu virsraksti.

3.5. tabula. **Pareizi klasificētā negatīvo klašu daļa TNR (1.5) atkarībā no pazīmju biežuma veida, izmantojot atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Oriģinālais teksts	SVM	0.821	0.800	0.833
	TREE	0.618	0.618	0.600
	RF	0.708	0.717	0.673
Bez mazsvarīgajām pazīmēm	SVM	0.829	0.829	0.850
	TREE	0.607	0.607	0.643
	RF	0.712	0.661	0.706
Bez mazsvarīgajām pazīmēm un pēc celmošanas	SVM	0.805	0.805	0.846
	TREE	0.679	0.679	0.660
	RF	0.783	0.766	0.750

Apskatot tabulu 3.5, varam ievērot, ka otrās grupas dati jeb ziņu virsraksti tiek ievērojami sliktāk klasificēti. Atbalsta vektoru metode visos gadījumos vismaz 80% no otrās grupas dokumentiem ir pareizi atpazinusi, bet gadījuma mežiem un, it sevišķi, klasifikācijas kokiem šis rādītājs ir ievērojami mazāks. Ievērosim, ka klasifikācijas koki tikai 60 līdz 68% no otrās grupas datiem ir pareizi saklasificējuši, kas ir tuvu 50% precizitātei, kas divu stāvokļu klasifikācijas problēmā ir ekvivalenti gadījuma minēšanai (*random guess*). Ievērosim arī to, ka atbalsta vektoru metode diezgan līdzīgi atpazīst gan pirmās, gan otrās grupas dokumentus. Gadījuma meži un klasifikācijas koki biežāk atpazīst tikai vienu grupu - pirmo grupu.

Tāpēc novērtējot modeļa klasificēšanas spējas, svarīgi ir novērtēt kopējo precizitāti. Tā pārada, cik liela daļa no visiem testa kopas datiem tika klasificēti pareizi, tāpēc novērtējam pareizi klasificēto dokumentu daļu EFF (1.6). Tā kā izmantojam sabalansētas izlases, EFF sniegs korektu novērtējumu. Rezultāti apkopoti tabulā 3.6.

3.6. tabula. **Pareizi klasificēto dokumentu daļa EFF (1.6) atkarībā no pazīmju biežuma veida, izmantojot atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Oriģinālais teksts	SVM	0.838	0.824	0.824
	TREE	0.676	0.676	0.649
	RF	0.770	0.770	0.730
Bez mazsvarīgajām pazīmēm	SVM	0.865	0.865	0.878
	TREE	0.676	0.676	0.716
	RF	0.797	0.743	0.784
Bez mazsvarīgajām pazīmēm un pēc celmošanas	SVM	0.851	0.851	0.865
	TREE	0.757	0.757	0.730
	RF	0.843	0.838	0.824

Apskatot modeļu kopējo precizitāti, redzam, ka visos gadījumos atbalsta vektoru metode uzrāda labāku precizitāti par klasifikācijas kokiem un gadījuma mežiem. Apskatīsim detalizētāk atbalsta vektoru metodes rezultātus. Oriģinālais teksts aptuveni 82% līdz 84% gadījumu tiek pareizi atpazīts, bet precizitāte pēc dimensiju skaita samazināšanas ir vēl augstāka. Vislielākā precizitāte 87,8% ir sasniegta, klasificējot tekstu bez maznozīmīgajām pazīmēm un bez celmošanas, izmantojot TF-IDF biežumu.

Klasifikācijas koki un gadījuma meži labākos rezultātus uzrāda gadījumā, kad ir vismazāk pazīmju, proti, tekstam pēc celmošanas un bez mazsvarīgajām pazīmēm. Ievērosim to, ka šajā gadījumā ir mazākais dimensiju skaits. Savukārt, atbalsta vektoru metode uzrāda labākos rezultātus, klasificējot tekstu bez mazsvarīgajām pazīmēm un bez celmošanas. Iespējams, tas ir izskaidrojams ar to, ka veicot celmošanu tiek pazaudēta svarīga informācija par tekstu un pārāk atšķirīgas pazīmes tiek apvienotas, kā arī ar to, ka atbalsta vektoru metode diezgan labi strādā pie liela dimensiju skaita, kas citiem klasifikācijas algoritmiem bieži vien rada problēmas.

3.3. Klasifikācija ar atbalsta vektoru metodi atkarībā no kodolu funkcijas veida

Konstruējam atbalsta vektoru metodes, izmantojot dažāda veida kodolu funkcijas. Atbilstoši katram izvēlētajam kodolam optimālais parametra C novērtējums iegūts ar krosvalidācijas metodi. Balstoties uz iepriekšējās nodaļas secinājumiem, klasificējam datus bez mazsvarīgajām pazīmēm un bez celmošanas. Katrs modelis apskatīts, izmantojot gan TF biežumu, gan bināro TF biežumu, gan normalizēto TF-IDF biežumu.

Konstruēsim modeļa pareizi klasificēto dokumentu daļas EFF (1.6) ticamības intervālu ar krosvalidācijas metodi. Konstruēsim krosvalidācijas procentīļu intervālus. Tas nozīmē, ka no datu kopas izslēdz vienu no ierakstiem, veic klasifikāciju un novērtē tās pareizi klasificēto dokumentu daļu. Šo novērtējumu apzīmēsim ar $\hat{\theta}_i^*$. To atkārto līdz katrs novērojums ir bijis vienreiz izslēgts no datu kopas. Tad $1-\alpha$ krosvalidācijas procentīļu ticamības intervāls ir

$$\left(\hat{\theta}^*_{(M(\alpha/2))}, \hat{\theta}^*_{(M(1-\alpha/2))}\right),$$

kur $M = |\hat{\theta}^*|$ un $\hat{\theta}^*_{(M(\alpha/2))}$ un $\hat{\theta}^*_{(M(1-\alpha/2))}$ ir krosvalidācijas vērtību izlases attiecīgi $\alpha/2$ un $1-\alpha/2$ kvantile.

Ar krosvalidācijas metodi konstruējam 95% ticamības intervālus un 99% ticamības intervālus. Iegūtie rezultāti apkopoti tabulā 3.7 un 3.8.

3.7. tabula. **95% ticamības intervāli pareizi klasificēto dokumentu daļas novērtējumam EFF (1.6), izmantojot atbalsta vektoru metodi, atkarībā no pazīmju biežuma veida un kodola funkcijas veida**

Kodola funkcija	TF	Binārais TF	TF-IDF
Lineārais kodols	(0.849; 0.877)	(0.849; 0.877)	(0.852; 0.890)
Polinomiālais kodols	(0.849; 0.887)	(0.849; 0.887)	(0.853; 0.890)
Gausa kodols	(0.849; 0.877)	(0.849; 0.877)	(0.849; 0.877)
Sigmoidal kodols	(0.863; 0.890)	(0.863; 0.890)	(0.863; 0.887)

3.8. tabula. **99% ticamības intervāli pareizi klasificēto dokumentu daļas novērtējumam EFF (1.6), izmantojot atbalsta vektoru metodi, atkarībā no pazīmju biežuma veida un kodola funkcijas veida**

Kodola funkcija	TF	Binārais TF	TF-IDF
Lineārais kodols	(0.836; 0.879)	(0.836; 0.877)	(0.849; 0.890)
Polinomiālais kodols	(0.846; 0.889)	(0.837; 0.889)	(0.849; 0.890)
Gausa kodols	(0.847; 0.890)	(0.836; 0.890)	(0.849; 0.890)
Sigmoidal kodols	(0.849; 0.890)	(0.849; 0.890)	(0.849; 0.890)

Redzam, ka visi apskatītie atbalsta vektoru modeļi darbojas ar ļoti līdzīgu precizitāti. Ja apskatām 95% ticamības intervālus, redzam, ka pareizi klasificēti tiek aptuveni 85% līdz 88% no ziņu virsrakstiem. Ar TF un bināro TF biežumu konstruētie 95% ticamības intervāli ir ļoti līdzīgi.

Apskatot 99% ticamības intervālus, secinām, ka aptuveni 84% līdz 89% no ziņām tiek pareizi klasificētas. Salīdzinot rezultātus atkarībā no biežumu novērtējumiem un atkarībā no kodolu funkcijas, redzam, ka būtiskas atšķirības nav vērojamas.

4. Latviešu ziņu portālu komentāru klasifikācija

Darba galvenais mērķis ir veikt latviešu ziņu portālu komentāru klasifikāciju agresīvajos un neagresīvajos. Ja tiktu iegūts labs klasifikācijas modelis, tad to varētu izmantot, lai ik dienas monitorētu komentāru kopējo agresivitātes līmeni.

Lai veiktu klasifikāciju, izmantosim 3 ziņu portālu - *apollo.lv, delfi.lv, tvnet.lv* - komentāru datus. No visiem ziņu portāliem kopumā tiek atlasīti 3000 komentāri. Trīs cilvēki, to starp darba autore, šos komentārus ir izlasījuši un saklasificējuši agresīvajos un neagresīvajos komentāros. Tāpat tika izdalīti nederīgie komentāri, piemēram, tādi, kas satur tikai interneta saites adresi vai kas ir angļu vai krievu valodā utt. Kopumā 178 komentāri tika atzīti par nederīgiem. Šie komentāri netika iekļauti ne apmācības, ne testa kopā.

Tātad veiksim klasifikācijas uzdevumu, kura mērķis būs ziņu portālu komentārus sadalīt agresīvajos un neagresīvajos komentāros.

4.1. Dimensiju skaita samazināšana

Tas ir vispārzināms fakts, ka liela daļa interneta komentāru ir gramatiski nekorekti uzrakstīti. Bieži vien garie patskaņi tiek aizvietoti ar diviem secīgiem īsajiem patskaņiem vai vienkārši vienu īso patskani. Mīkstie līdzskaņi un šņāceņi tiek aizstāti ar cipariem vai līdzskaņiem bez mīkstinājuma zīmēm vai šņāceņu apzīmējumiem u.c. Šie kļūdainie dati palielina pazīmju skaitu, jo viens vārds var tikt uzrakstīts dažādos veidos, kas liedz tos apvienot vienā pazīmē.

Lai vismaz nedaudz cīnītos ar šo problēmu, programmā R tika izmantota paka *textcat* [9], kas nosaka teksta valodu. Tas tiek darīts, izmantojot Eiropas valodu vārdu krājumu (*The European Corpus Initiative Multilingual Corpus (ECI)*), kas satur aptuveni 98 miljonus vārdu. Šīs pakas algoritms arī kļūdainam tekstam nosaka valodu, citiem vārdiem sakot, tekstam tiek piešķirta visatbilstošākā valodā. Aptuveni 90% no komentāriem tika atpazīti kā latviešu valodas teksts. Pārējie komentāri lielākoties tika uzskatīti par lietuviešu valodas komentāriem. Komentāri, kam netika atpazīta latviešu valoda, netika iekļauti klasifikācijā.

Pēc nederīgo komentāru izņemšanas un agresīvo un neagresīvo komentāru kopu sabalansēšanas kopumā klasifikācijā izmantoti 1214 komentāri - 607 agresīvi, 607 neagresīvi.

Līdzīgi kā angļu valodas datiem arī šoreiz izņemam ciparus, pieturzīmes un latviešu

valodas stopvārdus. Lai dimensiju skaitu vēl vairāk samazinātu, veiksim teksta lemmatizāciju. Latvijas Universitātes Matemātikas un Informātikas institūta pētnieks Pēteris Paikens ir izstrādājis latviešu valodas lemmatizācijas algoritmu, ko izmantosim komentāru datu lemmatizācijai. Tā ir veikta programmēšanas valodā JAVA.

Pēc lemmatizācijas pazīmju skaits ir ievērojami samazinājies, skatīt tabulu 4.1. Taču, kā jau minēts, dati ir ļoti kļūdaini, lemmatizācijas algoritms ar šo problēmu nespēj cīnīties un tāpēc kļūdainie vārdi tiek atstāti bez izmaiņām.

4.1. tabula. **Pazīmju skaits pirms un pēc lemmatizācijas**

Datu apstrādes veids	Pazīmju skaits
Pirms lemmatizācijas	13 888
Pēc lemmatizācijas	8 875

Pateicoties lemmatizācijai, pazīmju skaits ir samazinājies par 36%.

Pēc lemmatizācijas lietvārdi pārveidoti vienskaitlī un nominatīvā, piemēram, vārds "*stundā*" aizstāts ar "*stunda*", "*gadiem*" pārveidots par "*gads*". Darbības vārdi tiek pārveidoti nenoteiksmē, piemēram, "*beigsies*" aizstāts ar "*beigties*", bet "*jālaiž*" ar "*laist*". Arī vietniekvārdi tiek mainīti, piemēram, "*man*" pārveidots par "*es*". Taču vārdus, kas uzrakstīti kļūdaini, algoritms vai nu atstāj bez izmaiņām, vai arī pārveido citā kļūdainā vārdā, piemēram, nepareizi uzrakstītais vārds "*runat*" aizstāts ar "*runēt*". Piemērus skatīt tabulā 4.2

4.2. tabula. **Komentāru piemēri pirms un pēc apstrādes un lemmatizācijas**

Oriģinālais teksts	Teksts pēc mazsvarīgo pazīmju izņemšanas	Teksts bez mazsvarīgajām pazīmēm un pēc lemmatizācijas
Man navigācija rāda pat par 7 km/h stundā mazāk, nekā spidometrs.	man navigācija rāda kmh stundā mazāk spidometrs	es navigācija rādīt km h stunda mazi spidometrs
tiesa uzpirkta,prokuratura ari,par knab vispar nav ko runat, lielakus naudaz mazgatajus JUMTS.	tiesa uzpirkta prokuratura ari knab vispar nav ko runat lielakus naudaz mazgatajus jumts	tiesa uzpirkt prokuratura art knab vispar runēt lielaks naudaz mazgatajs jumts
Gaidu kad beidzot uz pasaules visi runās vienā valodā un būs vienā savienībā. Beigsies visi kari un naudu izlietos,lai atpūstos, izglītotos un paaugstinātu dzīves kvalitāti.	gaidu beidzot pasaules visi runās vienā valodā vienā savienībā beigsies visi kari naudu izlietoslai atpūstos izglītotos paaugstinātu dzīves kvalitāti	gaidīt beidzot pasaule viss runāt viena valoda viena savienība beigties viss karš nauda izlietot atpūsties izglītot paaugstināt dzīve kvalitāte
Shtokijs lai paprieksh nomaksaa savu firmu nodoklu paradu valstij. Un tad lai padomaa - ka vins iedomajas no Luxemburga registretas firmas uzzinat naudas izcelsmi.	shtokijs paprieksh nomaksaa savu firmu nodoklu paradu valstij padomaa vins iedomajas luxemburga registretas firmas uzzinat naudas izcelsmi	shtokijs paprieksh nomaksaa sava firma nodoklāt parads valsts padomaa vins iedomajēties luxemburgs registreta firma uzzinēt nauda izcelsme
Aizņemties vai nē,ir katra paša darīšana,te visugudras receptes nav. Krāt naudu gadiem arī nav gudri,tā jālaiž apgrozībā. Ja kāds negrib citam atdot ne santīmu,tad jau arī uz veikalu pēc maizes iet ir ķeceriski,jo kāds taču ar to nopelna un savāc tavus grašus!	aizņemties nē ir katra paša darīšana te visugudras receptes nav krāt naudu gadiem nav gudri tā jālaiž apgrozībā kāds negrib citam atdot santīmu tad veikalu maizes iet ķeceriski kāds nopelna savāc tavus grašus	aizņemties katrs pats darīšana visugudra recepte krāt nauda gads gudrs laist apgrozība kāds negrib cits atdot santīms tad jau veikals maize iet ķeceriski kāds nopelns savākt tavs grasis

Konstruēsim agresīvo un neagresīvo komentāru 50 biežāk sastopamo vārdu mākoņus. Skatīt attēlu 4.1 un 4.2

4.1. att. **Agresīvajos komentāros 50 biežāk sastopamo vārdu mākonis**

4.2. att. **Neagresīvajos komentāros 50 biežāk sastopamo vārdu mākonis**

Apskatot 50 biežāk sastopamo vārdu mākoņus, redzam, ka atšķirībā no *NYTimes* datu kopas agresīvo un neagresīvo komentāru vārdu mākoņi ir ļoti līdzīgi. Tas skaidrojams ar to, ka *NYTimes* datu kopas atšķirīgajām grupām bija dažādi temati, un biežāk

tika lietoti katram tematam atbilstošie vārdi. Bet komentāru dati aptver dažādas tēmas. Bieži vien par vienu tēmu izteikti gan neagresīvi, gan agresīvi komentāri, tāpēc šie vārdu mākoņi stipri pārklājas.

4.2. Klasifikācijas metožu salīdzinājums

Veiksim komentāru klasifikāciju. Mērķis ir izveidot modeli, kurš klasificē komentārus agresīvajos un neagresīvajos. Šī problēma savā ziņā ir līdzīga sentimentu analīzes uzdevumam.

Klasifikāciju veiksim ar naivo Beijesa metodi, atbalsta vektoru metodi, klasifikācijas kokiem un gadījuma mežiem. Darba gaitā veikta klasifikācija ar naivo Beijesa metodi arī nesabalansētai datu kopai, taču iegūtie rezultāti rāda, ka visi testa kopas komentāri tiek klasificēti kā neagresīvi, jo šo komentāru ir vairāk nekā agresīvo.

Atbalsta vektoru metodē izmantosim lineāro kodolu un parametru C (2.12) novērtēsim ar krosvalidācijas metodi. 80% no komentāriem iekļausim apmācības kopā, bet atlikušos - testa kopā.

Klasifikāciju veicam tekstam pēc maznozīmīgo pazīmju izņemšanas un tekstam pēc maznozīmīgo pazīmju izņemšanas un lemmatizācijas. Izmantosim gan TF, gan bināro TF, gan TF-IDF biežumu novērtējumus. Darbā [16] minēts, ka sentimentu klasifikācijas uzdevumos binārais TF biežums uzrāda labākus rezultātus nekā parastais TF biežums.

Konstruējam TPR, TNR un EFF novērtējumus. Rezultāti apkopoti 4.3, 4.4 un 4.5

4.3. tabula. **Pareizi klasificētā pozitīvo klašu daļa TPR (1.3) atkarībā no pazīmju biežumu veida, izmantojot naivo Beijesa metodi (NB), atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Bez mazsvarīgajām pazīmēm	NB	0.500	0.514	0.489
	SVM	0.588	0.568	0.566
	TREE	–	–	–
	RF	0.588	0.600	0.627
Bez mazsvarīgajām pazīmēm un pēc lemmatizācijas	NB	0.518	0.511	0.523
	SVM	0.622	0.628	0.601
	TREE	0.612	0.612	0.612
	RF	0.634	0.642	0.619

4.4. tabula. **Pareizi klasificētā negatīvo klašu daļa TNR (1.5) atkarībā no pazīmju biežumu veida, izmantojot naivo Beijesa metodi (NB), atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Bez mazsvarīgajām pazīmēm	NB	0.480	0.643	0.462
	SVM	0.555	0.557	0.579
	TREE	–	–	–
	RF	0.526	0.528	0.536
Bez mazsvarīgajām pazīmēm un pēc lemmatizācijas	NB	0.706	0.587	0.551
	SVM	0.580	0.581	0.638
	TREE	0.520	0.520	0.520
	RF	0.529	0.521	0.568

4.5. tabula. **Pareizi klasificētā dokumentu daļa EFF (1.6) atkarībā no pazīmju biežumu veida, izmantojot naivo Beijesa metodi (NB), atbalsta vektoru metodi (SVM), klasifikācijas kokus (TREE), gadījuma mežus (RF)**

Datu apstrādes veids	Metode	TF	Bin. TF	TF-IDF
Bez mazsvarīgajām pazīmēm	NB	0.498	0.519	0.481
	SVM	0.567	0.562	0.576
	TREE	–	–	–
	RF	0.539	0.543	0.556
Bez mazsvarīgajām pazīmēm un pēc lemmatizācijas	NB	0.531	0.523	0.531
	SVM	0.596	0.592	0.617
	TREE	0.535	0.535	0.535
	RF	0.564	0.567	0.583

Klasifikācijas koku metodi izdevās pielietot tikai lemmatizētajam tekstam. Pirms lemmatizācijas pazīmju skaits ir pārāk liels, lai programmā R spētu konstruēt klasifikācijas koku metodi. Klasifikācijas koki pareizi sagrupē aptuveni 53.5% no lemmatizētajiem komentāriem, kas ir salīdzinoši sliktāk nekā atbalsta vektoru metode un gadījuma mežu metode. Klasifikācijas kokiem TPR ir nozīmīgi lielāks nekā TNR, kas nozīmē to, ka vienas grupas komentāri tiek labāk atpazīti nekā otras grupas komentāri.

Naivā Beijesa metode līdzīgi kā klasifikācijas koki pareizi atpazinuši aptuveni 53%

no lemmatizētajiem komentāriem. Bet tekstam bez lemmatizācijas rezultāti ir sliktāki - pat zem 50%. Vienīgi ar bināro TF biežumu precizitāte ir 52%.

Gadījuma meži uzrāda labākus rezultātus. Lemmatizētos komentārus gadījuma meži ir pareizi saklasificējuši aptuveni 57% gadījumu. Jāpiezīmē, ka gadījuma mežu modeļa aprēķinu veikšanai nepieciešams ievērojami ilgāks laiks nekā klasifikācijas koku un atbalsta vektoru metodei.

Labākos rezultātus uzrāda atbalsta vektoru metode. Komentāri pirms lemmatizācijas ir pareizi saklasificēti aptuveni 56% gadījumu, bet pēc lemmatizācijas - 60% gadījumu. Labākais rezultāts sasniegts ar TF-IDF biežumu - 62%.

Iegūtie atbalsta vektoru metodes modeļi kopumā uzrāda labākus rezultātus nekā citas apskatītās metodes, taču precizitāte nav augsta. Tas varētu būt skaidrojams ar to, ka dati ir ļoti kļūdaini. Pirmkārt, tas palielina dimensiju skaitu, otrkārt, tas neļauj vienādas pazīmes apvienot vienā. Tāpat kļūdainajiem vārdiem lemmatizācijas algoritms nebija pielietojams.

Cits iemesls sliktajai modeļu precizitātei varētu būt tas, ka agresīvajos un neagresīvajos komentāros parādās lielākoties vieni un tie paši vārdi. Atšķirībā no *NYTimes* datu kopas, kur ziņu virsraksti tika klasificēti pēc tēmām un katrai no tēmām bija zināma daļa tikai šai tēmai raksturīgo vārdu. Komentāru datiem šāda situācija nav vērojama. Komentāri ir par ļoti dažādām tēmām, un par katru no tēmām ir gan agresīvi, gan neagresīvi komentāri.

4.3. Klasifikācija ar atbalsta vektoru metodi atkarībā no kodolu funkcijas veida

Konstruēsim atbalsta vektoru metodi, izmantojot dažādus kodolu veidus. Lai novērtētu iegūtos modeļus, konstruēsim krosvalidācijas ticamības intervālus pareizi klasificēto dokumentu novērtējumam. Tā kā apmācības un testa kopa satur 1214 komentārus un aprēķini, secīgi izslēdzot tikai vienu komentāru, būtu ļoti laikietilpīgi, šajā gadījumā secīgi izslēdzam uzreiz pa desmit komentāriem.

Balstoties uz iepriekš apskatītajiem rezultātiem, atbalsta vektoru metodes konstruējam tekstam pēc lemmatizācijas.

4.6. tabula. **95% ticamības intervāli pareizi klasificēto dokumentu daļas novērtējumam EFF (1.6), izmantojot atbalsta vektoru metodi, atkarībā no pazīmju biežumu veida un kodola funkcijas veida**

Kodola funkcija	TF	Bin. TF	TF-IDF
Lineārais kodols	(0.568; 0.602)	(0.560; 0.598)	(0.598; 0.631)
Polinomiālais kodols	(0.576; 0.608)	(0.569; 0.607)	(0.602; 0.631)
Gausa kodols	(0.606; 0.656)	(0.610; 0.639)	(0.444; 0.502)
Sigmoidal kodols	(0.598; 0.631)	(0.593; 0.631)	(0.411; 0.510)

4.7. tabula. **99% ticamības intervāli pareizi klasificēto dokumentu daļas novērtējumam EEF (1.6), izmantojot atbalsta vektoru metodi, atkarībā no pazīmju biežumu veida un kodola funkcijas veida**

Kodola funkcija	TF	Bin. TF	TF-IDF
Lineārais kodols	(0.567; 0.607)	(0.559; 0.601)	(0.594; 0.639)
Polinomiālais kodols	(0.575; 0.611)	(0.566; 0.610)	(0.599; 0.639)
Gausa kodols	(0.604; 0.656)	(0.604; 0.641)	(0.430; 0.504)
Sigmoidal kodols	(0.593; 0.632)	(0.590; 0.632)	(0.393; 0.510)

Apskatot konstruētos ticamības intervālus EFF novērtējumam, redzam, ka, izmantojot lineāro un polinomiālo kodolu, pareizi saklasificētā komentāru daļa ir aptuveni 57-60% ar TF un bināro TF biežumu, savukārt, ar TF-IDF tiek iegūti labāki rezultāti - ap 60 līdz 63%. Atbalsta vektoru metode ar Gausa kodolu labākus rezultātus uzrāda ar TF un bināro TF biežumu, pareizi klasificēti aptuveni 60-64% komentāru. Izmantojot *Sigmoidal* kodolu, iegūstam aptuveni 59 līdz 63% pareizi klasificētu dokumentu. Interesanti, ka atbalsta vektoru metode ar Gausa un *Sigmoidal* kodolu uzrāda ļoti sliktus rezultātus ar TF-IDF biežumu.

Redzam, ka vislabākos rezultātus uzrāda atbalsta vektoru metode ar Gausa kodolu. Konstruētie krosvalidācijas ticamības intervāli ir lielāki par 0.5, tas nozīmē, ka šie modeļi komentāru klasifikācijai dod labākus rezultātus nekā gadījuma minēšana. Bet protams, precizitāte tik un tā nav īpaši augsta.

Darba gaitā tika veikta klasifikācija, kad izņemtas vienreiz sastopamās pazīmes. Taču pareizi klasificēto komentāru daļas novērtējums būtiskus uzlabojumus nesniedza.

Secinājumi

Darbā apskatīta teksta klasifikācijas problēma, izpētītas biežāk lietotās mašīnmācīšanās metodes. Tāpat dots ieskats par celmošanas un lemmatizācijas izmantošanu teksta datu dimensiju skaita samazināšanai. Sniegts detalizēts atbalsta vektoru metodes teorētiskais apraksts.

Veikta angļu valodas teksta klasifikācija, izmantojot "*The New York Times*" ziņu virsrakstu datus. Ziņu klasifikācija veikta atbilstoši tēmām. Iegūtie rezultāti rāda, ka atbalsta vektoru metode uzrāda labākos rezultātus starp visām apskatītajām metodēm. Teksta datu celmošana ir ievērojami samazinājusi pazīmju jeb vārdu skaitu. Pēc celmošanas klasifikācijas koki un gadījuma meži uzrāda labākus rezultātus, bet atbalsta vektoru metodei celmošanas izmantošana nav devusi uzlabojumu. Veikta datu klasifikācija ar atbalsta vektoru metodi, izmantojot dažādas kodolu funkciju transformācijas. Visi apskatītie atbalsta vektoru metodes modeļi uzrāda aptuveni 85% līdz 88% precizitāti.

Darba galvenais mērķis bija veikt latviešu ziņu portālu komentāru klasifikāciju agresīvajos un neagresīvajos. Arī šiem datiem atbalsta vektoru metode uzrāda labākus rezultātus nekā citas apskatītās metodes. Tika veikta komentāru datu lemmatizācija, tās rezultātā tika ievērojami samazināts datu dimensiju skaits un klasifikācijas modeļu precizitāte uzlabojās. Vislabākie rezultāti sasniegti ar atbalsta vektoru metodi, izmantojot Gausa kodolu un TF vai bināro TF pazīmju biežumu. Aptuveni 60% līdz 64% komentāru ir klasificēti pareizi.

Kopējā precizitāte nav augsta un ir ievērojami sliktāka nekā rezultāti "*The New York Times*" datiem. Tam par iemeslu varētu būt tas, ka komentāru dati ir ļoti kļūdaini, kas liedz vienādas pazīmes apvienot un neļauj korekti pielietot lemmatizāciju. Cits iemesls varētu būt saistīts ar klasifikācijas uzdevuma specifiku, proti, ziņu virsraksti tika klasificēti atbilstoši tēmām, kur katrai no abām grupām ir savi nozarei raksturīgi vārdi. Taču komentāru dati ir par dažādām tēmām un par katru no šiem tematiem var būt gan agresīvi, gan neagresīvi komentāri.

Lai uzlabotu komentāru klasifikācijas rezultātus, iespējams, varētu pielietot algoritmu, kas veic teksta pareizrakstības korekciju. Taču šādā gadījumā tiktu izveidots modelis, kas prasa daudz resursu, un tad rodas jautājums, vai šāds modelis tiešām būtu praktiski izmantojams, lai veiktu visu komentāru monitorēšanu ik dienas.

Izmantotā literatūra un avoti

[1] Manisha Pravin Mali, Mohammad Atique. *Applications of Text Classification using Text Mining*, volume 13. International Journal of Engineering Trends and Technology (IJETT), 2014.

[2] Lam Hong Lee, Khairullah Khan, Aurangzeb Khan, Baharum Baharudin. A review of machine learning algorithms for text-documents classification. *Journal of Advances in Information Technology*, 1(1), May 2010.

[3] Adrian Bilski. A review of artificial intelligence algorithms in document classification. *Journal of Electronics and Telecommunications*, 57(3), 2011.

[4] Adrian Bilski. An improved random forest classifier for text categorization. *Journal of Computers*, 7(12), 2012.

[5] Christopher M. Bishop. *Pattern Recognition And Machine Learning*. Springer, 2006.

[6] Milan Bouchet-Valat. *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*, 2014. R package version 0.5.1.

[7] Gerard Salton, Christopher Buckley. *Term - Weighting Approaches in Automatic Text Retrieval*. Department of Computer Science, Cornell University, 1988.

[8] Vladimir Vapnik, Corinna Cortes. *Support Vector Networks*. Kluwer Academic Publishers, 1995.

[9] Kurt Hornik, Patrick Mair, Johannes Rauch, Wilhelm Geiger, Christian Buchta, Ingo Feinerer. The textcat package for n-gram based text categorization in R. *Journal of Statistical Software*, 52(6):1–17, 2013.

[10] Ian Fellows. *wordcloud: Word Clouds*, 2014. R package version 2.5.

[11] Tristan Fletcher. *Support Vector Machines Explained*. UCL, 2008.

[12] Ingo Feinerer, Kurt Hornik. *tm: Text Mining Package*, 2015. R package version 0.6-2.

[13] Chih-Jen Lin, Chih-Wei Hsu. *A Comparison of Methods for Multi-class Support Vector Machines*. National Taiwan University, 2009.

[14] Michal Tomana, Roman Tesara, Karel Jezeka. *Influence of Word Normalization on Text Classification*. Faculty of Applied Sciences University of West Bohemia.

[15] Thorsten Joachims. *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*. Dortmund University.

[16] James H. Martin, Daniel Jurafsky. *Speech And Language Processing*. 2016.

[17] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2012. R package version 1.6-1.

[18] Jie Tang, Mingcai Hong, Duo Zhang, Bangyong Liang, Juanzi Li. *Information Extraction: Methodologies and Applications*. Department of Computer Science, Tsinghua University.

[19] Istvan Pilaszy. *Text Categorization and Support Vector Machines*. Budapest University of Technology and Economics.

[20] Lauma Pretkalniņa. *Morfoloģija*. Valodas tehnoloģiju specseminārs.

[21] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[22] Timothy P. Jurka, Loren Collingwood, Amber E. Boydstun, Emiliano Grossman, Wouter van Atteveldt. *RTextTools: Automatic Text Classification via Supervised Learning*, 2014. R package version 1.4.2.

[23] Timothy P. Jurka, Loren Collingwood, AmberE. Boydstun, Emiliano Grossman, Wouter van Atteveldt. *RTextTools: A Supervised Learning Package for Text Classification*. Number ISSN 2073-4859. The R Journal Vol. 5/1, 2013.

[24] Andy Liaw, Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

Pielikums

Programmas R kods "*The New York Times*" ziņu virsrakstu klasifikācijai

```
## paka xlsx failu ielasīšanai
library(readxl)
## paka teksta klasifikācijai
library(RTextTools)
## paka teksta datu apstrādei (text mining)
library(tm)
## paka SVM klasifikācijai
library(e1071)
library(RColorBrewer)
## paka vārdu mākoņu ilustrācijai
library(wordcloud)
## paka vārdu celmošanai
library(SnowballC)

setwd("C:/Users/Kristīne Dzalbe/Desktop/Magistrantura/Magistra_darbs/Dati")

# read_excel ielasa gan xls, gan xlsx failus
data <- read_excel("NYTimes.xlsx")

dim(data)

new <- rep(1, length(data[,1]))
data$new <- new
head(data)
aggregate(data$new, by = list(data$Topic.Code), FUN = sum)

data1 <- data[data$Topic.Code == 3 ,]
dim(data1)
data2 <- data[data$Topic.Code == 19 ,]
dim(data2)
data2 <- data2[1:nrow(data1),]
dim(data2) == dim(data1)

data_both <- rbind(data1, data2)
dim(data_both)
data_both <- data_both[order(data_both$Title),]
data_model <- data_both[, c(3,5,6)]

data_model$Title_original <- data_model$Title

head(data_model)

## abas klases pārsauc par 1 un -1
data_model[,2][data_model[,2] == 3] <- 1
data_model[,2][data_model[,2] == 19] <- -1

## visus burtus pārvērš par mazajiem
data_model[,1] <- tolower(data_model[,1])
## izņem ciparus
data_model[,1] <- removeNumbers(data_model[,1])
## izņem stopvārdus
sort(stopwords("english"))
data_model[,1] <- removeWords(data_model[,1], stopwords("english"))
## izņem pieturzīmes
data_model[,1] <- removePunctuation(data_model[,1])
```

```
head(data_model)

## celmošana
stem_words <- function(x) {
        split <- strsplit(x, " ")
        return(wordStem(unlist(split), language = "porter"))
    }

stemming <- c()
for (k in 1 : nrow(data_model)) {
stemming[k] <- paste(stem_words(data_model[k,1]), collapse = " ")
}

data_model$stem <- c()
data_model$stem <- stemming
head(data_model)
dim(data_model)

## pārkārto datus, lai iegūtu sabalansētu apmācības un testa izlasi
## apmācības kopa - 80% ierakstu, testa kopa - 20%
n <- length(data_model[,1])
size <- round(0.8*n)
size2 <- size + 1
n; size; size2
## dokumentu skaits testa kopā
n - size

vector <- seq(1, n, by = 1)
vector2 <- sample(vector)
data_model$index <- vector2
data_model <- data_model[order(data_model$index),]
data_test <- data_model[size2:n, ]
agg <- aggregate(data_test$new, by = list(data_test$Topic.Code), FUN = sum)
agg
agg[1,2] == agg[2,2]

while (agg[1,2] != agg[2,2]){
vector2 <- sample(vector)
data_model$index <- vector2
data_model <- data_model[order(data_model$index),]
data_test <- data_model[size2:n, ]
agg <- aggregate(data_test$new, by = list(data_test$Topic.Code), FUN = sum)
}

write.table(data_model, file = "datamodel.txt", sep = "~", col.names = TRUE)

### vārdu biežumss
corpus_data <- Corpus(VectorSource(data_model$Title_original))
tdm <- TermDocumentMatrix(corpus_data)
inspect(tdm)
## Find frequent terms in specified range
findFreqTerms(tdm,  lowfreq = 15, highfreq = 100)
length(findFreqTerms(tdm,  lowfreq = 1, highfreq = 1000))

names(data_model)

## konstruējam vārdu mākoņus pirmās un otrās klases pazīmēm
data_negative <- data_model[data_model$Topic.Code == -1,]
dim(data_negative)
```

```
corpus_data1 <- Corpus(VectorSource(data_negative[,1]))
tdm1 <- TermDocumentMatrix(corpus_data1)
inspect(tdm1)

m1 <- as.matrix(tdm1)
v1 <- sort(rowSums(m1),decreasing = TRUE)
d1 <- data.frame(word = names(v1),freq = v1)
wordcloud(words = d1$word, freq = d1$freq, #min.freq = 20,
          max.words=50, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))


data_positive <- data_model[data_model$Topic.Code == 1,]
dim(data_positive)
corpus_data2 <- Corpus(VectorSource(data_positive[,1]))
tdm2 <- TermDocumentMatrix(corpus_data2)
inspect(tdm2)

m2 <- as.matrix(tdm2)
v2 <- sort(rowSums(m2),decreasing = TRUE)
d2 <- data.frame(word = names(v2),freq = v2)
wordcloud(words = d2$word, freq = d2$freq, min.freq = 3,
          max.words=50, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))


## Acronym -> acronym
trace("create_matrix", edit = T)


####### teksts bez maznozīmīgajām pazīmēm
## veido pazīmju biežumu matricu - TF biežums
dtMatrix.tf <- create_matrix(data_model$Title, language="english",
weight = weightTf)
dtMatrix.tf
attributes(dtMatrix.tf)
dtMatrix.tf$nrow
dtMatrix.tf$ncol

## veido pazīmju biežumu matricu - binārais TF biežums
dtMatrix.bin <- DocumentTermMatrix(Corpus(VectorSource(data_model$Title)),
control = list(weighting = weightBin))
dtMatrix.bin
dtMatrix.bin$ncol

## veido pazīmju biežumu matricu - TF-IDF biežums
dtMatrix.tfidf <- create_matrix(data_model$Title, language="english",
weight = weightTfIdf)
dtMatrix.tfidf
attributes(dtMatrix.tfidf)
dtMatrix.tfidf$ncol


####### teksts bez maznozīmīgajām pazīmēm un pēc celmošanas
## veido pazīmju biežumu matricu - TF biežums
dtMatrixStem.tf <- create_matrix(data_model$stem, language="english",
weight = weightTf)
dtMatrixStem.tf
dtMatrixStem.tf$ncol
```

```
## veido pazīmju biežumu matricu - binārais TF biežums
dtMatrixStem.bin <-
DocumentTermMatrix(Corpus(VectorSource(data_model$stem)),
control = list(weighting = weightBin))
dtMatrix.bin
dtMatrix.bin$ncol

## veido pazīmju biežumu matricu - TF-IDF biežums
dtMatrixStem.tfidf <- create_matrix(data_model$stem, language="english",
weight = weightTfIdf)
dtMatrixStem.tfidf
dtMatrixStem.tfidf$ncol


####### oriģinālais teksts
## veido pazīmju biežumu matricu - TF biežums
dtMatrixOriginal.tf <- create_matrix(data_model$Title_original, language = "english",
removePunctuation = FALSE, removeStopwords = FALSE,
weight = weightTf)
dtMatrixOriginal.tf
dtMatrixOriginal.tf$ncol

## veido pazīmju biežumu matricu - binārais TF biežums
dtMatrixOriginal.bin <-
DocumentTermMatrix(Corpus(VectorSource(data_mode$Title_original)),
control = list(weighting = weightBin))
dtMatrix.bin
dtMatrix.bin$ncol

## veido pazīmju biežumu matricu - TF-IDF biežums
dtMatrixOriginal.tfidf <- create_matrix(data_model$Title_original,
language = "english",
removePunctuation = FALSE,
removeStopwords = FALSE,
weight = weightTfIdf)
dtMatrixOriginal.tfidf
dtMatrixOriginal.tfidf$ncol


###### funkcijas modeļa kvalitātes novērtēšanai

## True positive rate
TPR <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return(TP / (TP + FN))
}

## True negative rate
TNR <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
```

```
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return(TN / (TN + FP))
}

accuracy <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return((TP + TN) / (TP + TN + FP + FN))
}


######################### SVM

## klasificē tekstu bez maznozīmīgajām pazīmēm - TF biežums
## konstruē testa un apmācības kopas
trace("create_container", edit = T)
container.tf <- create_container(dtMatrix.tf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tf@training_matrix, y = container.tf@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tf <- train_model(container.tf, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tf <- classify_model(container.tf, model_svm.tf)
analytics.tf <- create_analytics(container.tf, svm_classify.tf)
summary(analytics.tf)
TPR(analytics.tf@document_summary)
TNR(analytics.tf@document_summary)
accuracy(analytics.tf@document_summary)


## klasificē tekstu bez maznozīmīgajām pazīmēm - binārais TF biežums
## konstruē testa un apmācības kopas
container.bin <- create_container(dtMatrix.bin, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.bin@training_matrix, y = container.bin@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.bin <- train_model(container.bin, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.bin <- classify_model(container.bin, model_svm.bin)
analytics.bin <- create_analytics(container.bin, svm_classify.bin)
summary(analytics.bin)
TPR(analytics.bin@document_summary)
TNR(analytics.bin@document_summary)
accuracy(analytics.bin@document_summary)
```

```
## klasificē tekstu bez maznozīmīgajām pazīmēm - TF-IDF biežums
## konstruē testa un apmācības kopas
container.tfidf <- create_container(dtMatrix.tfidf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tfidf@training_matrix, y = container.tfidf@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tfidf <- train_model(container.tfidf, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tfidf <- classify_model(container.tfidf, model_svm.tfidf)
analytics.tfidf <- create_analytics(container.tfidf, svm_classify.tfidf)
summary(analytics.tfidf)
TPR(analytics.tfidf@document_summary)
TNR(analytics.tfidf@document_summary)
accuracy(analytics.tfidf@document_summary)


## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF biežums
## konstruē testa un apmācības kopas
container.tf2 <- create_container(dtMatrixStem.tf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tf2@training_matrix, y = container.tf2@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tf2 <- train_model(container.tf2, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tf2 <- classify_model(container.tf2, model_svm.tf2)
analytics.tf2 <- create_analytics(container.tf2, svm_classify.tf2)
summary(analytics.tf2)
TPR(analytics.tf2@document_summary)
TNR(analytics.tf2@document_summary)
accuracy(analytics.tf2@document_summary)



## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - binārais TF biežums
## konstruē testa un apmācības kopas
container.bin2 <- create_container(dtMatrixStem.bin, data_model[,2],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.bin2@training_matrix, y = container.bin2@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.bin2 <- train_model(container.bin2, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.bin2 <- classify_model(container.bin2, model_svm.bin2)
analytics.bin2 <- create_analytics(container.bin2, svm_classify.bin2)
summary(analytics.bin2)
TPR(analytics.bin2@document_summary)
TNR(analytics.bin2@document_summary)
accuracy(analytics.bin2@document_summary)
```

```
## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF-IDF biežums
## konstruē testa un apmācības kopas
container.tfidf2 <- create_container(dtMatrixStem.tfidf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tfidf2@training_matrix, y = container.tfidf2@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tfidf2 <- train_model(container.tfidf2, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tfidf2 <- classify_model(container.tfidf2, model_svm.tfidf2)
analytics.tfidf2 <- create_analytics(container.tfidf2, svm_classify.tfidf2)
summary(analytics.tfidf2)
TPR(analytics.tfidf2@document_summary)
TNR(analytics.tfidf2@document_summary)
accuracy(analytics.tfidf2@document_summary)
comp <- analytics.tfidf2@document_summary[,1] == analytics.tfidf2@document_summary[,3]
length(comp[comp == TRUE]) / (n-size2+1)


## klasificē oriģinālos datus - TF biežums
## konstruē testa un apmācības kopas
container.tf3 <- create_container(dtMatrixOriginal.tf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tf3@training_matrix, y = container.tf3@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tf3 <- train_model(container.tf3, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tf3 <- classify_model(container.tf3, model_svm.tf3)
analytics.tf3 <- create_analytics(container.tf3, svm_classify.tf3)
summary(analytics.tf3)
TPR(analytics.tf3@document_summary)
TNR(analytics.tf3@document_summary)
accuracy(analytics.tf3@document_summary)


## klasificē oriģinālos teksta datus - binārais TF biežums
## konstruē testa un apmācības kopas
container.bin3 <- create_container(dtMatrixOriginal.bin, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.bin3@training_matrix, y = container.bin3@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.bin3 <- train_model(container.bin3, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.bin3 <- classify_model(container.bin3, model_svm.bin3)
analytics.bin3 <- create_analytics(container.bin3, svm_classify.bin3)
summary(analytics.bin3)
TPR(analytics.bin3@document_summary)
TNR(analytics.bin3@document_summary)
accuracy(analytics.bin3@document_summary)
```

```
## klasificē oriģinālos datus - TF-IDF biežums
## konstruē testa un apmācības kopas
container.tfidf3 <- create_container(dtMatrixOriginal.tfidf, data_model[,2],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
## ar krosvalidācijas palīdzību novērtē parametru C
obj <- tune.svm(x = container.tfidf3@training_matrix, y = container.tfidf3@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]
model_svm.tfidf3 <- train_model(container.tfidf3, "SVM", kernel="linear",
cost=obj$best.parameters[1,])
svm_classify.tfidf3 <- classify_model(container.tfidf3, model_svm.tfidf3)
analytics.tfidf3 <- create_analytics(container.tfidf3, svm_classify.tfidf3)
summary(analytics.tfidf3)
TPR(analytics.tfidf3@document_summary)
TNR(analytics.tfidf3@document_summary)
accuracy(analytics.tfidf3@document_summary)


## apskatīto SVM modeļu apkopojums
tf <- TPR(analytics.tf@document_summary)
tf2 <- TPR(analytics.tf2@document_summary)
tf3 <- TPR(analytics.tf3@document_summary)
bin <- TPR(analytics.bin@document_summary)
bin2 <- TPR(analytics.bin2@document_summary)
bin3 <- TPR(analytics.bin3@document_summary)
tfidf <- TPR(analytics.tfidf@document_summary)
tfidf2 <- TPR(analytics.tfidf2@document_summary)
tfidf3 <- TPR(analytics.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- TNR(analytics.tf@document_summary)
tf2 <- TNR(analytics.tf2@document_summary)
tf3 <- TNR(analytics.tf3@document_summary)
bin <- TNR(analytics.bin@document_summary)
bin2 <- TNR(analytics.bin2@document_summary)
bin3 <- TNR(analytics.bin3@document_summary)
tfidf <- TNR(analytics.tfidf@document_summary)
tfidf2 <- TNR(analytics.tfidf2@document_summary)
tfidf3 <- TNR(analytics.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- accuracy(analytics.tf@document_summary)
tf2 <- accuracy(analytics.tf2@document_summary)
tf3 <- accuracy(analytics.tf3@document_summary)
bin <- accuracy(analytics.bin@document_summary)
bin2 <- accuracy(analytics.bin2@document_summary)
bin3 <- accuracy(analytics.bin3@document_summary)
tfidf <- accuracy(analytics.tfidf@document_summary)
tfidf2 <- accuracy(analytics.tfidf2@document_summary)
tfidf3 <- accuracy(analytics.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)



### Klasifikācijas koki
## klasificē tekstu bez maznozīmīgajām pazīmēm - TF biežums
```

```
model.tree.tf <- train_model(container.tf, "TREE")
classify.tree.tf <- classify_model(container.tf, model.tree.tf)
analytics.tree.tf <- create_analytics(container.tf, classify.tree.tf)
summary(analytics.tree.tf)
TPR(analytics.tree.tf@document_summary)
TNR(analytics.tree.tf@document_summary)
accuracy(analytics.tree.tf@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm - binārais TF biežums
## konstruē testa un apmācības kopas
model.tree.bin <- train_model(container.bin, "TREE")
classify.tree.bin <- classify_model(container.bin, model.tree.bin)
analytics.tree.bin <- create_analytics(container.bin, classify.tree.bin)
summary(analytics.tree.bin)
TPR(analytics.tree.bin@document_summary)
TNR(analytics.tree.bin@document_summary)
accuracy(analytics.tree.bin@document_summary)
table("Predicted" = analytics.tree.bin@document_summary[,1],
"Actual" = analytics.tree.bin@document_summary[,3])

## klasificē tekstu bez maznozīmīgajām pazīmēm - TF -IDF biežums
model.tree.tfidf <- train_model(container.tfidf, "TREE")
classify.tree.tfidf <- classify_model(container.tfidf, model.tree.tfidf)
analytics.tree.tfidf <- create_analytics(container.tfidf, classify.tree.tfidf)
summary(analytics.tree.tfidf)
TPR(analytics.tree.tfidf@document_summary)
TNR(analytics.tree.tfidf@document_summary)
accuracy(analytics.tree.tfidf@document_summary)


## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF biežums
model.tree.tf2 <- train_model(container.tf2, "TREE")
classify.tree.tf2 <- classify_model(container.tf2, model.tree.tf2)
analytics.tree.tf2 <- create_analytics(container.tf2, classify.tree.tf2)
summary(analytics.tree.tf2)
TPR(analytics.tree.tf2@document_summary)
TNR(analytics.tree.tf2@document_summary)
accuracy(analytics.tree.tf2@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - binārais TF biežums
model.tree.bin2 <- train_model(container.bin2, "TREE")
classify.tree.bin2 <- classify_model(container.bin2, model.tree.bin2)
analytics.tree.bin2 <- create_analytics(container.bin2, classify.tree.bin2)
summary(analytics.tree.bin2)
TPR(analytics.tree.bin2@document_summary)
TNR(analytics.tree.bin2@document_summary)
accuracy(analytics.tree.bin2@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF -IDF biežums
model.tree.tfidf2 <- train_model(container.tfidf2, "TREE")
classify.tree.tfidf2 <- classify_model(container.tfidf2, model.tree.tfidf2)
analytics.tree.tfidf2 <- create_analytics(container.tfidf2, classify.tree.tfidf2)
summary(analytics.tree.tfidf2)
TPR(analytics.tree.tfidf2@document_summary)
TNR(analytics.tree.tfidf2@document_summary)
accuracy(analytics.tree.tfidf2@document_summary)


## oriģinālos datus - TF biežums
model.tree.tf3 <- train_model(container.tf3, "TREE")
```

```
classify.tree.tf3 <- classify_model(container.tf3, model.tree.tf3)
analytics.tree.tf3 <- create_analytics(container.tf3, classify.tree.tf3)
summary(analytics.tree.tf3)
TPR(analytics.tree.tf3@document_summary)
TNR(analytics.tree.tf3@document_summary)
accuracy(analytics.tree.tf3@document_summary)

## oriģinālos datus - binārais TF biežums
model.tree.bin3 <- train_model(container.bin3, "TREE")
classify.tree.bin3 <- classify_model(container.bin3, model.tree.bin3)
analytics.tree.bin3 <- create_analytics(container.bin3, classify.tree.bin3)
summary(analytics.tree.bin3)
TPR(analytics.tree.bin3@document_summary)
TNR(analytics.tree.bin3@document_summary)
accuracy(analytics.tree.bin3@document_summary)

## oriģinālos datus - TF -IDF biežums
model.tree.tfidf3 <- train_model(container.tfidf3, "TREE")
classify.tree.tfidf3 <- classify_model(container.tfidf3, model.tree.tfidf3)
analytics.tree.tfidf3 <- create_analytics(container.tfidf3, classify.tree.tfidf3)
summary(analytics.tree.tfidf3)
TPR(analytics.tree.tfidf3@document_summary)
TNR(analytics.tree.tfidf3@document_summary)
accuracy(analytics.tree.tfidf3@document_summary)


## apskatīto klasifikācijas koku modeļu apkopojums
tf <- TPR(analytics.tree.tf@document_summary)
tf2 <- TPR(analytics.tree.tf2@document_summary)
tf3 <- TPR(analytics.tree.tf3@document_summary)
bin <- TPR(analytics.tree.bin@document_summary)
bin2 <- TPR(analytics.tree.bin2@document_summary)
bin3 <- TPR(analytics.tree.bin3@document_summary)
tfidf <- TPR(analytics.tree.tfidf@document_summary)
tfidf2 <- TPR(analytics.tree.tfidf2@document_summary)
tfidf3 <- TPR(analytics.tree.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- TNR(analytics.tree.tf@document_summary)
tf2 <- TNR(analytics.tree.tf2@document_summary)
tf3 <- TNR(analytics.tree.tf3@document_summary)
bin <- TNR(analytics.tree.bin@document_summary)
bin2 <- TNR(analytics.tree.bin2@document_summary)
bin3 <- TNR(analytics.tree.bin3@document_summary)
tfidf <- TNR(analytics.tree.tfidf@document_summary)
tfidf2 <- TNR(analytics.tree.tfidf2@document_summary)
tfidf3 <- TNR(analytics.tree.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- accuracy(analytics.tree.tf@document_summary)
tf2 <- accuracy(analytics.tree.tf2@document_summary)
tf3 <- accuracy(analytics.tree.tf3@document_summary)
bin <- accuracy(analytics.tree.bin@document_summary)
bin2 <- accuracy(analytics.tree.bin2@document_summary)
bin3 <- accuracy(analytics.tree.bin3@document_summary)
tfidf <- accuracy(analytics.tree.tfidf@document_summary)
tfidf2 <- accuracy(analytics.tree.tfidf2@document_summary)
tfidf3 <- accuracy(analytics.tree.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)
```

```
### Random forests
## klasificē tekstu bez maznozīmīgajām pazīmēm - TF biežums
model.rf.tf <- train_model(container.tf, "RF")
classify.rf.tf <- classify_model(container.tf, model.rf.tf)
analytics.rf.tf <- create_analytics(container.tf, classify.rf.tf)
summary(analytics.rf.tf)
TPR(analytics.rf.tf@document_summary)
TNR(analytics.rf.tf@document_summary)
accuracy(analytics.rf.tf@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm - binārais TF biežums
model.rf.bin <- train_model(container.bin, "RF")
classify.rf.bin <- classify_model(container.bin, model.rf.bin)
analytics.rf.bin <- create_analytics(container.bin, classify.rf.bin)
summary(analytics.rf.bin)
TPR(analytics.rf.bin@document_summary)
TNR(analytics.rf.bin@document_summary)
accuracy(analytics.rf.bin@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm - TF-IDF biežums
model.rf.tfidf <- train_model(container.tfidf, "RF")
classify.rf.tfidf <- classify_model(container.tfidf, model.rf.tfidf)
analytics.rf.tfidf <- create_analytics(container.tfidf, classify.rf.tfidf)
summary(analytics.rf.tfidf)
TPR(analytics.rf.tfidf@document_summary)
TNR(analytics.rf.tfidf@document_summary)
accuracy(analytics.rf.tfidf@document_summary)


## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF biežums
model.rf.tf2 <- train_model(container.tf2, "RF")
classify.rf.tf2 <- classify_model(container.tf2, model.rf.tf2)
analytics.rf.tf2 <- create_analytics(container.tf2, classify.rf.tf2)
summary(analytics.rf.tf2)
TPR(analytics.rf.tf2@document_summary)
TNR(analytics.rf.tf2@document_summary)
accuracy(analytics.rf.tf2@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - binārais TF biežums
model.rf.bin2 <- train_model(container.bin2, "RF")
classify.rf.bin2 <- classify_model(container.bin2, model.rf.bin2)
analytics.rf.bin2 <- create_analytics(container.bin2, classify.rf.bin2)
summary(analytics.rf.bin2)
TPR(analytics.rf.bin2@document_summary)
TNR(analytics.rf.bin2@document_summary)
accuracy(analytics.rf.bin2@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF-IDF biežums
model.rf.tfidf2 <- train_model(container.tfidf2, "RF")
classify.rf.tfidf2 <- classify_model(container.tfidf2, model.rf.tfidf2)
analytics.rf.tfidf2 <- create_analytics(container.tfidf2, classify.rf.tfidf2)
summary(analytics.rf.tfidf2)
TPR(analytics.rf.tfidf2@document_summary)
TNR(analytics.rf.tfidf2@document_summary)
accuracy(analytics.rf.tfidf2@document_summary)


## klasificē oriģinālos datus - TF biežums
```

```
model.rf.tf3 <- train_model(container.tf3, "RF")
classify.rf.tf3 <- classify_model(container.tf3, model.rf.tf3)
analytics.rf.tf3 <- create_analytics(container.tf3, classify.rf.tf3)
summary(analytics.rf.tf3)
TPR(analytics.rf.tf3@document_summary)
TNR(analytics.rf.tf3@document_summary)
accuracy(analytics.rf.tf3@document_summary)

## klasificē oriģinālos datus - binārais TF biežums
model.rf.bin3 <- train_model(container.bin3, "RF")
classify.rf.bin3 <- classify_model(container.bin3, model.rf.bin3)
analytics.rf.bin3 <- create_analytics(container.bin3, classify.rf.bin3)
summary(analytics.rf.bin3)
TPR(analytics.rf.bin3@document_summary)
TNR(analytics.rf.bin3@document_summary)
accuracy(analytics.rf.bin3@document_summary)

## klasificē tekstu bez maznozīmīgajām pazīmēm un pēc celmošanas - TF-IDF biežums
model.rf.tfidf3 <- train_model(container.tfidf3, "RF")
classify.rf.tfidf3 <- classify_model(container.tfidf3, model.rf.tfidf3)
analytics.rf.tfidf3 <- create_analytics(container.tfidf3, classify.rf.tfidf3)
summary(analytics.rf.tfidf3)
TPR(analytics.rf.tfidf3@document_summary)
TNR(analytics.rf.tfidf3@document_summary)
accuracy(analytics.rf.tfidf3@document_summary)


## apskatīto modeļu apkopojums
tf <- TPR(analytics.rf.tf@document_summary)
tf2 <- TPR(analytics.rf.tf2@document_summary)
tf3 <- TPR(analytics.rf.tf3@document_summary)
bin <- TPR(analytics.rf.bin@document_summary)
bin2 <- TPR(analytics.rf.bin2@document_summary)
bin3 <- TPR(analytics.rf.bin3@document_summary)
tfidf <- TPR(analytics.rf.tfidf@document_summary)
tfidf2 <- TPR(analytics.rf.tfidf2@document_summary)
tfidf3 <- TPR(analytics.rf.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- TNR(analytics.rf.tf@document_summary)
tf2 <- TNR(analytics.rf.tf2@document_summary)
tf3 <- TNR(analytics.rf.tf3@document_summary)
bin <- TNR(analytics.rf.bin@document_summary)
bin2 <- TNR(analytics.rf.bin2@document_summary)
bin3 <- TNR(analytics.rf.bin3@document_summary)
tfidf <- TNR(analytics.rf.tfidf@document_summary)
tfidf2 <- TNR(analytics.rf.tfidf2@document_summary)
tfidf3 <- TNR(analytics.rf.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)

tf <- accuracy(analytics.rf.tf@document_summary)
tf2 <- accuracy(analytics.rf.tf2@document_summary)
tf3 <- accuracy(analytics.rf.tf3@document_summary)
bin <- accuracy(analytics.rf.bin@document_summary)
bin2 <- accuracy(analytics.rf.bin2@document_summary)
bin3 <- accuracy(analytics.rf.bin3@document_summary)
tfidf <- accuracy(analytics.rf.tfidf@document_summary)
tfidf2 <- accuracy(analytics.rf.tfidf2@document_summary)
tfidf3 <- accuracy(analytics.rf.tfidf3@document_summary)
round(cbind(tf, bin, tfidf, tf2, bin2, tfidf2, tf3, bin3, tfidf3), 3)
```

```
############## Naive Bayes

head(data_model)
# Create a corpus for training and testing data set
train.corpus <- Corpus(VectorSource(as.vector(data_model$stem[1:size])))
test.corpus <- Corpus(VectorSource(as.vector(data_model$stem[size2:n])))


# Create term document matrix
train.matrix <- DocumentTermMatrix(train.corpus,
control = list(weighting = weightTfIdf))
# inspect(train.matrix[50:60, 152:165])

test.matrix <- DocumentTermMatrix(test.corpus,
control = list(weighting = weightTfIdf))

# Build model with additive smoothing as 1
model <- naiveBayes(as.matrix((train.matrix)), as.factor(data_model[1:size, 2]), laplace = 1)


#Predict
results <- predict(object = model, newdata = as.matrix(test.matrix))

t <- table("Predictions" = results,  "Actual" = data_model[size2:n, 2])
t
tpr <- t[1,1] / (t[1,1] + t[1,2])
tnr <- t[2,2] / (t[2,2] + t[2,1])
eff <- (t[1,1] + t[2,2]) / length(data_model[size2:n, 2])
tpr
tnr
eff




################ SVM atkarībā no kodolu veida

## klasificē tekstu bez maznozīmīgajām pazīmēm - TF biežums
## lineārais kodols = linear
## polinomiālais kodols = polynomial
## Gausa kodols = radial
## sigmoid
container.tf <- create_container(dtMatrix.tf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)

## ar krosvalidācijas palīdzību novērtē parametrus
obj <- tune.svm(x = container.tf@training_matrix, y = container.tf@training_codes,
                cost = 2^(2:8), kernel = "linear")
names(obj)
obj$best.model
obj$best.parameters
model_svm.tf <- train_model(container.tf, "SVM", kernel = "linear",
cost = obj$best.parameters[1,], coef.0=0)
svm_classify.tf <- classify_model(container.tf, model_svm.tf)
analytics.tf <- create_analytics(container.tf, svm_classify.tf)
accuracy(analytics.tf@document_summary)
```

```
## klasificē tekstu bez maznozīmīgajām pazīmēm - TF-IDF biežums
## lineārais kodols = linear
## polinomiālais kodols = polynomial
## Gausa kodols = radial
## sigmoid
container.tfidf <- create_container(dtMatrix.tfidf, data_model[,2], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)

## ar krosvalidācijas palīdzību novērtē parametrus
obj <- tune.svm(x = container.tfidf@training_matrix, y = container.tfidf@training_codes,
                cost = 2^(2:8), kernel="sigmoid")
obj$best.model
obj$best.parameters
model_svm.tfidf <- train_model(container.tfidf, "SVM", kernel="sigmoid",
cost=obj$best.parameters[1,], coef.0=0)
svm_classify.tfidf <- classify_model(container.tfidf, model_svm.tfidf)
analytics.tfidf <- create_analytics(container.tfidf, svm_classify.tfidf)
accuracy(analytics.tfidf@document_summary)



################### SVM krosvalidācija

krosv <- c()

for (i in 1:n) {
data_krosv <- data_model[-i,]
n_krosv <- nrow(data_krosv)
dtMatrix <- create_matrix(data_krosv[,1], language = "english", removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTfIdf)
container <- create_container(dtMatrix, data_krosv[,2], trainSize = 1 : size,
testSize = size2 : n_krosv, virgin = FALSE)
obj <- tune.svm(x = container@training_matrix, y = container@training_codes,
                  cost = 2^(2:8), kernel = "sigmoid")
model_svm <- train_model(container, "SVM", kernel = "sigmoid",
cost = obj$best.parameters[1,], coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
comp <- analytics@document_summary[,1] == analytics@document_summary[,3]
krosv[i] <- length(comp[comp == TRUE]) / (n_krosv - size2 + 1)
}

mean(krosv)

## 95% tic. intervāls
alpha <- 0.05
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)

## 99% tic. intervāls
alpha <- 0.01
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)


### krosvalidācija ar bināro TF
krosv <- c()
```

```
for (i in 1:n) {
data_krosv <- data_model[-i,]
n_krosv <- nrow(data_krosv)
dtMatrix <- DocumentTermMatrix(Corpus(VectorSource(data_krosv[,1])),
control = list(weighting = weightBin))
container <- create_container(dtMatrix, data_krosv[,2], trainSize = 1 : size,
testSize = size2 : n_krosv, virgin = FALSE)
obj <- tune.svm(x = container@training_matrix, y = container@training_codes,
                  cost = 2^(2:8), kernel = "sigmoid")
model_svm <- train_model(container, "SVM", kernel = "sigmoid",
cost = obj$best.parameters[1,], coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
comp <- analytics@document_summary[,1] == analytics@document_summary[,3]
krosv[i] <- length(comp[comp == TRUE]) / (n_krosv - size2 + 1)
}

mean(krosv)

## 95% tic. intervāls
alpha <- 0.05
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)

## 99% tic. intervāls
alpha <- 0.01
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)




## krosvalidācija, pieņemot, ka C=0
krosv <- c()

for (i in 1:n) {
data_krosv <- data_model[-i,]
n_krosv <- nrow(data_krosv)
dtMatrix <- create_matrix(data_krosv[,1], language="english", removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTf)
container <- create_container(dtMatrix, data_krosv[,2], trainSize = 1 : size,
testSize = size2 : n_krosv, virgin = FALSE)
model_svm <- train_model(container, "SVM", kernel = "linear",
cost = 0.00001, coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
comp <- analytics@document_summary[,1] == analytics@document_summary[,3]
krosv[i] <- length(comp[comp == TRUE]) / (n_krosv - size2 + 1)
}

mean(krosv)



################### Random Forests krosvalidācija

krosv <- c()
```

```
for (i in 1:n) {
data_krosv <- data_model[-i,]
n_krosv <- nrow(data_krosv)
dtMatrix <- create_matrix(data_krosv[,1], language="english", removeStopwords=TRUE)
container <- create_container(dtMatrix, data_krosv[,2], trainSize = 1 : size,
testSize = size2 : n_krosv, virgin = FALSE)

model <- train_model(container, "RF")
classify <- classify_model(container, model)
analytics <- create_analytics(container, classify)
comp <- analytics@document_summary[,1] == analytics@document_summary[,3]
krosv[i] <- length(comp[comp == TRUE]) / (n_krosv-size2)
}

mean(krosv)


############### Atsauces
## the basic R reference
citation()
library(randomForest)
## references for a package -- might not have these installed
if(nchar(system.file(package = "nnet"))) citation("nnet")

## extract the bibtex entry from the return value
x <- citation()
```

Programmas R kods latviešu ziņu portālu komentāru klasifikācijai

```
## paka xlsx failu ielasīšanai
library(readxl)
## paka teksta klasifikācijai
library(RTextTools)
## paka teksta datu apstrādei (text mining)
library(tm)
## paka SVM klasifikācijai
library(e1071)
library(RColorBrewer)
## paka vārdu mākoņu ilustrācijai
library(wordcloud)
### atpazīst teksta valodu
library(textcat)

setwd("C:/Users/Kristīne Dzalbe/Desktop/Magistrantura/Magistra_darbs/Dati")

# read_excel reads both xls and xlsx files
data1 <- read_excel("0001-0500_3_apvienojums_4.xlsx")
data2 <- read_excel("0501-1000_3_apvienojums_4.xlsx")
data3 <- read_excel("1001-1500_3_apvienojums_4.xlsx")
data4 <- read_excel("1501-2000_3_apvienojums_4.xlsx")
data5 <- read_excel("2001-2500_3_apvienojums_4.xlsx")
data6 <- read_excel("2501-3000_3_apvienojums_4.xlsx")

data <- rbind(data1, data2, data3, data4, data5, data6)

names(data)
head(data)
dim(data)
data[452,2]

## Translate characters in character vectors, in particular from upper to lower case
data[,2] <- tolower(data[,2])
data[,3] <- tolower(data[,3])
data[,4] <- tolower(data[,4])
data[,5] <- tolower(data[,5])
data[,6] <- tolower(data[,6])
data[,7] <- tolower(data[,7])
data[,8] <- tolower(data[,8])
data[,10] <- tolower(data[,10])
head(data)

data$aggression4[data$aggression1 == data$aggression2] <- "sakrīt"

new <- rep(1, length(data[,1]))
data$new <- new
aggregate(data$new, by = list(data[,3], data[,5], data[,7], data[,8]), FUN = sum)

#izņem nederīgos komentārus
data <- data[!(data$aggression1 == "nederīgs" | data$aggression2 == "nederīgs" |
data$aggression3 == "nederīgs"),]
dim(data)

#### izdzēst tukšās rindiņas
data <- data[!is.na(data$aggression1),]; dim(data)
aggregate(data$new, by = list(data[,3], data[,5], data[,7]), FUN = sum)
```

```
data$aggression1[data$aggression1 == "ir agresīvs"] <- 1
data$aggression1[data$aggression1 == "nav agresīvs"] <- -1
data$aggression2[data$aggression2 == "ir agresīvs"] <- 1
data$aggression2[data$aggression2 == "nav agresīvs"] <- -1
data$aggression3[data$aggression3 == "ir agresīvs"] <- 1
data$aggression3[data$aggression3 == "nav agresīvs"] <- -1

data$aggression1 <- as.numeric(data$aggression1)
is.numeric(data$aggression1)
data$aggression2 <- as.numeric(data$aggression2)
is.numeric(data$aggression2)
data$aggression3 <- as.numeric(data$aggression3)
is.numeric(data$aggression3)

aggregate(data$new, by = list(data[,3], data[,5], data[,7]), FUN = sum)

data$aggression_final <- rep(-1, length(data[,1]))
data$aggression_final <- data$aggression1 + data$aggression2 + data$aggression3
data$aggression_final[data$aggression_final >= 1] <- 1
data$aggression_final[data$aggression_final <= -1] <- -1

aggregate(data$new, by = list(data[,3], data[,5], data[,7], data$aggression_final),
FUN = sum)
aggregate(data$new, by = list(data$aggression_final), FUN = sum)
dim(data)


language <- c()
data$language <- language
for (i in 1:nrow(data)) {
data$language[i] <- textcat(data[i,2])
}
data_latvian <- data[data$language == "latvian",]
dim(data_latvian)

data_latvian <- data_latvian[!is.na(data_latvian$aggression1),]; dim(data_latvian)

aggregate(data_latvian$new, by = list(data_latvian[,3], data_latvian[,5],
data_latvian[,7], data_latvian$aggression_final), FUN = sum)
aggregate(data_latvian$new, by = list(data_latvian$aggression_final), FUN = sum)


### atstāj tikai tos datus, kas nepieciešami modelēšanai
data_model <- data_latvian
data_mod <- data_model[data_model[,3] == -1 & data_model[,5] == -1 & data_model[,7] == -1, ]
dim(data_mod)
data_modd <- data_model[data_model[,12] == 1, ]; dim(data_modd)
data_model <- rbind(data_mod, data_modd)
names(data_model)

### atstāj tikai komentāru un komentāra klasifikācijas grupas numuru
data_model <- data_model[, c(2, 10, 12)]
dim(data_model)
names(data_model)

### "tīra" komentārus - izņem skaitļus, pieturzīmes, stopvārdus
data_model[,1] <- removeNumbers(data_model[,1])
data_model[,1] <- removePunctuation(data_model[,1])
data_model[,2] <- removeNumbers(data_model[,2])
data_model[,2] <- removePunctuation(data_model[,2])
```

```
stopwords <- read.table("C:/Users/Kristīne Dzalbe/Desktop/
Magistrantura/Magistra_darbs/Dati/stopwords.txt")
stopwords_vector <- as.vector(stopwords[,1])
data_model[,1] <- removeWords(data_model[,1], stopwords_vector)
data_model[,2] <- removeWords(data_model[,2], stopwords_vector)

### write.table(data_model, "datapiemeri.csv", sep = "~")

### veido nesabalansētu kopu priekš naivā Beijesa metode
data_model_NB <- data_latvian[, c(2, 10, 12)]

### veido sabalansētu izlasi
datan <- data_model[data_model$aggression_final == -1,]
datap <- data_model[data_model$aggression_final == 1,]
dim(datan); dim(datap)
datann <- datan[c(1:length(datap[,1])),]
dim(datann)
data_model <- rbind(datann, datap)
head(data_model)
dim(data_model)


### vārdu frekvences
corpus_data <- Corpus(VectorSource(data_model[,2]))
tdm <- TermDocumentMatrix(corpus_data)
inspect(tdm)
## Find frequent terms in specified range
findFreqTerms(tdm,  lowfreq = 15, highfreq = 100)
length(findFreqTerms(tdm,  lowfreq = 1, highfreq = 1000))


m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
head(d, 50)

### wordclouds
wordcloud(words = d$word, freq = d$freq, #min.freq = 35,
          max.words = 50, random.order = FALSE, rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))

## neagresīvo komentāru vārdu mākonis
data_non_aggresive <- data_model[data_model$aggression_final==-1,]
dim(data_non_aggresive)
corpus_data1 <- Corpus(VectorSource(data_non_aggresive[,2]))
tdm1 <- TermDocumentMatrix(corpus_data1)
inspect(tdm1)

m1 <- as.matrix(tdm1)
v1 <- sort(rowSums(m1),decreasing=TRUE)
d1 <- data.frame(word = names(v1),freq=v1)
wordcloud(words = d1$word, freq = d1$freq, #min.freq = 20,
          max.words = 50, random.order = FALSE, rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))

## agresīvo komentāru vārdu mākonis
data_aggresive <- data_model[data_model$aggression_final==1,]
dim(data_aggresive)
corpus_data2 <- Corpus(VectorSource(data_aggresive[,2]))
tdm2 <- TermDocumentMatrix(corpus_data2)
```

```
inspect(tdm2)

m2 <- as.matrix(tdm2)
v2 <- sort(rowSums(m2),decreasing=TRUE)
d2 <- data.frame(word = names(v2),freq=v2)
wordcloud(words = d2$word, freq = d2$freq, #min.freq = 20,
          max.words = 50, random.order = FALSE, rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))

head(data_model)


### pārkārto datus, lai training izlase un testa izlases būtu sabalansētas
n <- length(data_model[,1])
size <- round(0.8*n)
size2 <- size + 1
n; size; size2


data_model$new <- rep(1, n)
vector <- seq(1, n, by = 1)
vector2 <- sample(vector)
data_model$index <- vector2
data_model <- data_model[order(data_model$index),]
data_test <- data_model[size2:n, ]
head(data_test)
agg <- aggregate(data_test$new, by = list(data_test[,3]), FUN = sum)
agg
agg[1,2] == agg[2,2]+1

#### pārkārto datus, lai iegūtu sabalansētu testa izlasi
while (agg[1,2] != agg[2,2] + 1){
vector2 <- sample(vector)
data_model$index <- vector2
data_model <- data_model[order(data_model$index),]
data_test <- data_model[size2:n, ]
agg <- aggregate(data_test$new, by = list(data_test[,3]), FUN = sum)
}

aggregate(data_test$new, by = list(data_test[,3]), FUN = sum)
aggregate(data_model$new, by = list(data_model[,3]), FUN = sum)


################## klasifikācija


###### funkcijas modeļa kvalitātes novērtēšanai

## True positive rate
TPR <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return(TP / (TP + FN))
}
```

```
## True negative rate
TNR <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return(TN / (TN + FP))
}

accuracy <- function(analytics) {
tp.vector <- analytics[,1] == analytics[,3] & analytics[,1] == 1
TP <- length(tp.vector[tp.vector == TRUE])
tn.vector <- analytics[,1] == analytics[,3] & analytics[,1] == -1
TN <- length(tn.vector[tn.vector == TRUE])
fp.vector <- analytics[,1] != analytics[,3] & analytics[,1] == -1
FP <- length(fp.vector[fp.vector == TRUE])
fn.vector <- analytics[,1] != analytics[,3] & analytics[,1] == 1
FN <- length(fn.vector[fn.vector == TRUE])
return((TP + TN) / (TP + TN + FP + FN))
}


## Acronym -> acronym
trace("create_matrix",edit=T)

names(data_model)

## konstruē pazīmju biežumu matricu
dtMatrix.norm.tf <- create_matrix(data_model$text_norm, removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTf)
dtMatrix.norm.tf
dtMatrix.norm.tf$ncol

dtMatrix.norm.bin <- DocumentTermMatrix(Corpus(VectorSource(data_model$text_norm)),
control = list(weighting = weightBin))
dtMatrix.norm.bin
dtMatrix.norm.bin$ncol

dtMatrix.norm.tfidf <- create_matrix(data_model$text_norm, removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTfIdf)
dtMatrix.norm.tfidf
dtMatrix.norm.tfidf$ncol


dtMatrix.text.tf <- create_matrix(data_model$text, removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTf)
dtMatrix.text.tf
dtMatrix.text.tf$ncol

dtMatrix.text.bin <- DocumentTermMatrix(Corpus(VectorSource(data_model$text)),
control = list(weighting = weightBin))
dtMatrix.text.bin
dtMatrix.text.bin$ncol

dtMatrix.text.tfidf <- create_matrix(data_model$text, removePunctuation = FALSE,
```

```
removeStopwords = FALSE, weight = weightTfIdf)
dtMatrix.text.tfidf
dtMatrix.text.tfidf$ncol


######################## SVM
## lemmatizētais teksts - TF biežums
container.norm.tf <- create_container(dtMatrix.norm.tf, data_model[,3], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)

## ar krosvalidācijas palīdzību novērtē parametrus
obj <- tune.svm(x = container.norm.tf@training_matrix, y = container.norm.tf@training_codes,
                cost = 2^(2:8))
names(obj)
obj$best.model
obj$best.parameters[1,]

model_svm.norm.tf <- train_model(container.norm.tf, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.norm.tf <- classify_model(container.norm.tf, model_svm.norm.tf)

names(svm_classify.norm.tf)

analytics.norm.tf <- create_analytics(container.norm.tf, svm_classify.norm.tf)
summary(analytics.norm.tf)

plot(analytics.norm.tf@document_summary$SVM_PROB,
analytics.norm.tf@document_summary$MANUAL_CODE,
 col = analytics.norm.tf@document_summary$SVM_LABEL)

accuracy(analytics.norm.tf@document_summary)
TPR(analytics.norm.tf@document_summary)
TNR(analytics.norm.tf@document_summary)


## lemmatizētais teksts - binārais TF biežums
container.norm.bin <- create_container(dtMatrix.norm.bin, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container.norm.bin@training_matrix, y = container.norm.bin@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.norm.bin <- train_model(container.norm.bin, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.norm.bin <- classify_model(container.norm.bin, model_svm.norm.bin)
analytics.norm.bin <- create_analytics(container.norm.bin, svm_classify.norm.bin)
summary(analytics.norm.bin)
accuracy(analytics.norm.bin@document_summary)
TPR(analytics.norm.bin@document_summary)
TNR(analytics.norm.bin@document_summary)


## lemmatizētais teksts - TF-IDF biežums
container.norm.tfidf <- create_container(dtMatrix.norm.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container.norm.tfidf@training_matrix,
y = container.norm.tfidf@training_codes,
                cost = 2^(2:8))
obj$best.model
```

```
model_svm.norm.tfidf <- train_model(container.norm.tfidf, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.norm.tfidf <- classify_model(container.norm.tfidf, model_svm.norm.tfidf)
analytics.norm.tfidf <- create_analytics(container.norm.tfidf, svm_classify.norm.tfidf)
summary(analytics.norm.tfidf)
accuracy(analytics.norm.tfidf@document_summary)
TPR(analytics.norm.tfidf@document_summary)
TNR(analytics.norm.tfidf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF biežums
container.text.tf <- create_container(dtMatrix.text.tf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container.text.tf@training_matrix,
y = container.text.tf@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.text.tf <- train_model(container.text.tf, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.text.tf <- classify_model(container.text.tf, model_svm.text.tf)
analytics.text.tf <- create_analytics(container.text.tf, svm_classify.text.tf)
summary(analytics.text.tf)
accuracy(analytics.text.tf@document_summary)
TPR(analytics.text.tf@document_summary)
TNR(analytics.text.tf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - binārais TF biežums
container.text.bin <- create_container(dtMatrix.text.bin, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container.text.bin@training_matrix,
y = container.text.bin@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.text.bin <- train_model(container.text.bin, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.text.bin <- classify_model(container.text.bin, model_svm.text.bin)
analytics.text.bin <- create_analytics(container.text.bin, svm_classify.text.bin)
summary(analytics.text.bin)
accuracy(analytics.text.bin@document_summary)
TPR(analytics.text.bin@document_summary)
TNR(analytics.text.bin@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF-IDF biežums
container.text.tfidf <- create_container(dtMatrix.text.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container.text.tfidf@training_matrix,
y = container.text.tfidf@training_codes,
                cost = 2^(2:8))
obj$best.model
model_svm.text.tfidf <- train_model(container.text.tfidf, "SVM", kernel = "linear",
cost = obj$best.parameters[1,])
svm_classify.text.tfidf <- classify_model(container.text.tfidf, model_svm.text.tfidf)
analytics.text.tfidf <- create_analytics(container.text.tfidf, svm_classify.text.tfidf)
summary(analytics.text.tfidf)
accuracy(analytics.text.tfidf@document_summary)
```

```
TPR(analytics.text.tfidf@document_summary)
TNR(analytics.text.tfidf@document_summary)



################### random forests
## lemmatizētais teksts - TF biežums
container.norm.tf <- create_container(dtMatrix.norm.tf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.norm.tf <- train_model(container.norm.tf, "RF")
rf_classify.norm.tf <- classify_model(container.norm.tf, model_rf.norm.tf)
analytics_rf.norm.tf <- create_analytics(container.norm.tf, rf_classify.norm.tf)
summary(analytics_rf.norm.tf)
accuracy(analytics_rf.norm.tf@document_summary)
TPR(analytics_rf.norm.tf@document_summary)
TNR(analytics_rf.norm.tf@document_summary)


## lemmatizētais teksts - binārais TF biežums
container.norm.bin <- create_container(dtMatrix.norm.bin, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.norm.bin <- train_model(container.norm.bin, "RF")
rf_classify.norm.bin <- classify_model(container.norm.bin, model_rf.norm.bin)
analytics_rf.norm.bin <- create_analytics(container.norm.bin, rf_classify.norm.bin)
summary(analytics_rf.norm.bin)
accuracy(analytics_rf.norm.bin@document_summary)
TPR(analytics_rf.norm.bin@document_summary)
TNR(analytics_rf.norm.bin@document_summary)


## lemmatizētais teksts - TF-IDF biežums
container.norm.tfidf <- create_container(dtMatrix.norm.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.norm.tfidf <- train_model(container.norm.tfidf, "RF")
rf_classify.norm.tfidf <- classify_model(container.norm.tfidf, model_rf.norm.tfidf)
analytics_rf.norm.tfidf <- create_analytics(container.norm.tfidf, rf_classify.norm.tfidf)
summary(analytics_rf.norm.tfidf)
accuracy(analytics_rf.norm.tfidf@document_summary)
TPR(analytics_rf.norm.tfidf@document_summary)
TNR(analytics_rf.norm.tfidf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF biežums
container.text.tf <- create_container(dtMatrix.text.tf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.text.tf <- train_model(container.text.tf, "RF")
rf_classify.text.tf <- classify_model(container.text.tf, model_rf.text.tf)
analytics_rf.text.tf <- create_analytics(container.text.tf, rf_classify.text.tf)
summary(analytics_rf.text.tf)
accuracy(analytics_rf.text.tf@document_summary)
TPR(analytics_rf.text.tf@document_summary)
TNR(analytics_rf.text.tf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - binārais TF biežums
container.text.bin <- create_container(dtMatrix.text.bin, data_model[,3],
```

```
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.text.bin <- train_model(container.text.bin, "RF")
rf_classify.text.bin <- classify_model(container.text.bin, model_rf.text.bin)
analytics_rf.text.bin <- create_analytics(container.text.bin, rf_classify.text.bin)
summary(analytics_rf.text.bin)
accuracy(analytics_rf.text.bin@document_summary)
TPR(analytics_rf.text.bin@document_summary)
TNR(analytics_rf.text.bin@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF-IDF biežums
container.text.tfidf <- create_container(dtMatrix.text.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_rf.text.tfidf <- train_model(container.text.tfidf, "RF")
rf_classify.text.tfidf <- classify_model(container.text.tfidf, model_rf.text.tfidf)
analytics_rf.text.tfidf <- create_analytics(container.text.tfidf, rf_classify.text.tfidf)
summary(analytics_rf.text.tfidf)
accuracy(analytics_rf.text.tfidf@document_summary)
TPR(analytics_rf.text.tfidf@document_summary)
TNR(analytics_rf.text.tfidf@document_summary)



################### klasifikācijas koki
## lemmatizētais teksts - TF biežums
container.norm.tf <- create_container(dtMatrix.norm.tf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.norm.tf <- train_model(container.norm.tf, "TREE")
tree_classify.norm.tf <- classify_model(container.norm.tf, model_tree.norm.tf)
analytics_tree.norm.tf <- create_analytics(container.norm.tf, tree_classify.norm.tf)
summary(analytics_tree.norm.tf)
accuracy(analytics_tree.norm.tf@document_summary)
TPR(analytics_tree.norm.tf@document_summary)
TNR(analytics_tree.norm.tf@document_summary)


## lemmatizētais teksts - binārais TF biežums
container.norm.bin <- create_container(dtMatrix.norm.bin, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.norm.bin <- train_model(container.norm.bin, "TREE")
tree_classify.norm.bin <- classify_model(container.norm.bin, model_tree.norm.bin)
analytics_tree.norm.bin <- create_analytics(container.norm.bin, tree_classify.norm.bin)
summary(analytics_tree.norm.bin)
accuracy(analytics_tree.norm.bin@document_summary)
TPR(analytics_tree.norm.bin@document_summary)
TNR(analytics_tree.norm.bin@document_summary)


## lemmatizētais teksts - TF-IDF biežums
container.norm.tfidf <- create_container(dtMatrix.norm.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.norm.tfidf <- train_model(container.norm.tfidf, "TREE")
tree_classify.norm.tfidf <- classify_model(container.norm.tfidf, model_tree.norm.tfidf)
analytics_tree.norm.tfidf <- create_analytics(container.norm.tfidf,
tree_classify.norm.tfidf)
```

```
summary(analytics_tree.norm.tfidf)
accuracy(analytics_tree.norm.tfidf@document_summary)
TPR(analytics_tree.norm.tfidf@document_summary)
TNR(analytics_tree.norm.tfidf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF biežums
container.text.tf <- create_container(dtMatrix.text.tf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.text.tf <- train_model(container.text.tf, "TREE")
tree_classify.text.tf <- classify_model(container.text.tf, model_tree.text.tf)
analytics_tree.text.tf <- create_analytics(container.text.tf, tree_classify.text.tf)
summary(analytics_tree.text.tf)
accuracy(analytics_tree.text.tf@document_summary)
TPR(analytics_tree.text.tf@document_summary)
TNR(analytics_tree.text.tf@document_summary)


## teksts bez mazsvarīgajām pazīmēm - binārais TF biežums
container.text.bin <- create_container(dtMatrix.text.bin, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.text.bin <- train_model(container.text.bin, "TREE")
tree_classify.text.bin <- classify_model(container.text.bin, model_tree.text.bin)
analytics_tree.text.bin <- create_analytics(container.text.bin, tree_classify.text.bin)
summary(analytics_tree.text.bin)
accuracy(analytics_tree.text.bin@document_summary)
TPR(analytics_tree.text.bin@document_summary)
TNR(analytics_tree.text.bin@document_summary)


## teksts bez mazsvarīgajām pazīmēm - TF-IDF biežums
container.text.tfidf <- create_container(dtMatrix.text.tfidf, data_model[,3],
trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
model_tree.text.tfidf <- train_model(container.text.tfidf, "TREE")
tree_classify.text.tfidf <- classify_model(container.text.tfidf, model_tree.text.tfidf)
analytics_tree.text.tfidf <- create_analytics(container.text.tfidf,
tree_classify.text.tfidf)
summary(analytics_tree.text.tfidf)
accuracy(analytics_tree.text.tfidf@document_summary)
TPR(analytics_tree.text.tfidf@document_summary)
TNR(analytics_tree.text.tfidf@document_summary)



############## naivais Beijess - sabalansēta kopa
names(data_model)
# Create a corpus for training and testing data set
train.corpus <- Corpus(VectorSource(as.vector(data_model$text[1:size])))
test.corpus <- Corpus(VectorSource(as.vector(data_model$text[size2:n])))

# Create term document matrix
train.matrix <- DocumentTermMatrix(train.corpus,
control = list(weighting = weightBin))
# inspect(train.matrix[50:60, 152:165])

test.matrix <- DocumentTermMatrix(test.corpus,
control = list(weighting = weightBin))
```

```
# Build model with additive smoothing as 1
model <- naiveBayes(as.matrix((train.matrix)), as.factor(data_model[1:size, 3]),
laplace = 1)
#Predict
results <- predict(object = model, newdata = as.matrix(test.matrix))

t <- table("Predictions" = results,  "Actual" = data_model[size2:n, 3])
t
tpr <- t[1,1] / (t[1,1] + t[1,2])
tnr <- t[2,2] / (t[2,2] + t[2,1])
eff <- (t[1,1] + t[2,2]) / length(data_model[size2:n, 2])
tpr
tnr
eff


############## naivais Beijess klaifikators nesabalansētai kopai
names(data_model_NB)
dim(data_model_NB)

new <- rep(1, nrow(data_model_NB))
data_model_NB$new <- new
aggregate(data_model_NB$new, by = list(data_model_NB$aggression_final), FUN = sum)

### veido apmācības un testa kopu
n_NB <- nrow(data_model_NB)
size_NB <- round(0.8 * n_NB, 0)
size2_NB <- size_NB + 1
train_NB <- data_model_NB[1 : size_NB, ]
test_NB <- data_model_NB[size2_NB : n_NB, ]

aggregate(train_NB$new, by = list(train_NB$aggression_final), FUN = sum)
aggregate(test_NB$new, by = list(test_NB$aggression_final), FUN = sum)

train.corpus <- Corpus(VectorSource(as.vector(train_NB$text)))
test.corpus <- Corpus(VectorSource(as.vector(test_NB$text)))

### veido pazīmju biežumu matricu
train.matrix <- DocumentTermMatrix(train.corpus,
control = list(weighting = weightBin))
test.matrix <- DocumentTermMatrix(test.corpus,
control = list(weighting = weightBin))

### veic NB apmācīšanu
model <- naiveBayes(as.matrix((train.matrix)), as.factor(train_NB$aggression_final),
 laplace = 1)

### klasificē testa kopas datus
results <- predict(object = model, newdata = as.matrix(test.matrix))
attributes(results)
length(results[results==1])
length(results)
t <- table("Predictions" = results,  "Actual" = test_NB$aggression_final)
t
tpr <- t[1,1] / (t[1,1] + t[1,2])
tnr <- t[2,2] / (t[2,2] + t[2,1])
eff <- (t[1,1] + t[2,2]) / n_NB
tpr
tnr
eff
```

```
####################### SVM atkarībā no kodola veida
## klasificē tekstu lemmatizēto tekstu
## lineārais kodols = linear
## polinomiālais kodols = polynomial
## Gausa kodols = radial
## sigmoid
container <- create_container(dtMatrix.norm.tfidf, data_model[,3], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)

## ar krosvalidācijas palīdzību novērtē parametrus
obj <- tune.svm(x = container@training_matrix, y = container@training_codes,
                cost = 2^(2:8), kernel = "sigmoid")
obj$best.model
obj$best.parameters
model_svm <- train_model(container, "SVM", kernel = "sigmoid",
cost = obj$best.parameters[1,], coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
accuracy(analytics@document_summary)



################### SVM krosvalidācija

krosv <- c()
m <- round(n/10, 0)

for (i in 1:m) {
j1 <- (i-1)*10
j2 <- (i-1)*10+9
data_krosv <- data_model[-(j1:j2),]
n_krosv <- nrow(data_krosv)
dtMatrix <- create_matrix(data_krosv$text_norm, removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTf)
size <- round(n_krosv*0.8, 0)
size2 <- size + 1
container <- create_container(dtMatrix, data_krosv[,3], trainSize = 1 : size,
testSize = size2 : n_krosv, virgin = FALSE)
obj <- tune.svm(x = container@training_matrix, y = container@training_codes,
                  cost = 2^(2:8), kernel = "linear")
model_svm <- train_model(container, "SVM", kernel = "linear",
cost = obj$best.parameters[1,], coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
comp <- analytics@document_summary[,1] == analytics@document_summary[,3]
krosv[i] <- length(comp[comp == TRUE]) / (n_krosv - size2 + 1)
}

mean(krosv)

## 95% tic. intervāls
alpha <- 0.05
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)

## 99% tic. intervāls
alpha <- 0.01
```

```
c1 <- alpha / 2
c2 <- 1 - alpha / 2
round(quantile(krosv, c(c1, c2)), 3)


############## izņem pazīmes, kas sastopamas tikai vienreiz
names(data_model)
corpus_data <- Corpus(VectorSource(data_model[,2]))
tdm <- TermDocumentMatrix(corpus_data)
remove_words <- findFreqTerms(tdm,  lowfreq = 1, highfreq = 1)
length(remove_words)
remove_words_vector <- as.vector(remove_words)

for (i in 1:length(remove_words)) {
data_model[,2] <- removeWords(data_model[,2], remove_words_vector[i])
}

dtMatrix <- create_matrix(data_model$text_norm, removePunctuation = FALSE,
removeStopwords = FALSE, weight = weightTfIdf)
dtMatrix$ncol

container <- create_container(dtMatrix, data_model[,3], trainSize = 1 : size,
testSize = size2 : n, virgin = FALSE)
obj <- tune.svm(x = container@training_matrix, y = container@training_codes,
                cost = 2^(2:8)), kernel = "polynomial")
obj$best.model
obj$best.parameters
model_svm <- train_model(container, "SVM", kernel = "polynomial",
cost = obj$best.parameters[1,], coef.0 = 0)
svm_classify <- classify_model(container, model_svm)
analytics <- create_analytics(container, svm_classify)
accuracy(analytics@document_summary)
```

Maģistra darbs "Atbalsta vektoru metodes izmantošana teksta klasifikācijai" izstrādāts LU Fizikas un matemātikas fakultātē

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Darba autore: Kristīne Dzalbe

_____ ____.06.2017.
(paraksts)

Rekomendēju darbu aizstāvēšanai
Darba vadītājs: Dr.sc.comp. Normunds Grūzītis

_____ ____.06.2017.
(paraksts)

Recenzents: Dr. math. Mārtiņš Liberts

_____ ____.06.2017.
(paraksts)

Darbs iesniegts Matemātikas nodaļā ____.06.2015.
Dekāna pilnvarotā persona: vecākā metodiķe Dzintra Holsta

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

____.06.2017. prot. nr. _____, vērtējums _____

Komisijas sekretārs/-e: _____ _____
(Vārds, uzvārds) (paraksts)