VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS
BAKALAURO STUDIJŲ PROGRAMA
FINANSŲ IR DRAUDIMO MATEMATIKA

Baigiamasis bakalauro darbas

Mašininio mokymosi algoritmų taikymas neapykantos kalbos nustatymui iš lietuviško teksto

(Hate Speech Detection Using Machine Learning on Lithuanian Text)

Atliko: Raminta Rasimavičiūtė
VU el. paštas: raminta.rasimaviciute@mif.stud.vu.lt
Darbo vadovas: doc., dr. Raivydas Šimėnas

Vilnius
2022

Santrauka

Darbe naudojant mašininio mokymo algoritmus sprendžiamas teksto klasifikavimo uždavinys neapykantos kalbos nustatymui lietuviškam tekstui. Tyrime naudojamas atraminių vektorių klasifikatorius, ilgalaikės-trumpalaikės atminties neuroninis tinklas, bei daugiasluoksnio perceptrono klasifikatorius. Minėti metodai pritaikyti duomenų aibei sudarytai iš 1000 komentarų surinktų iš socialinių tinklų bei naujienų portalų. Darbe yra siekiama pasirinktiems metodams rasti aukščiausią tikslumą ir išskirti efektyviausią. Šiam tikslui pasiekti taikomas duomenų apdorojimas, lyginami keli vektorizacijos metodai bei optimizuojamas modelis ieškant hiperparametrų, kurie duotų didžiausią tikslumą. Įvertinus minėtų metodų tikslumą, nustatytas efektyviausias - LSTM.

Darbo raktiniai žodžiai: teksto klasifikavimas, natūraliosios kalbos apdorojimas, hiperparametrai, optimizacija, mašininis mokymasis, neapykantos kalbos aptikimas

Abstract

In this bachelor's thesis, machine learning algorithms are used such as the support vector (SVM) classifier, the multi-layer perceptron (MLP) algorithm and the long short-term memory (LSTM) artificial neural network are used to solve hate speech detection problem for Lithuanian text. For training and testing, 1000 comments were selected from social media and news portals. First of all, data preprocessing was done, various vectorization techniques were tested for SVM, MLP classifier, also hyperparameter tuning methods were applied in order to reach the highest accuracy for machine learning algorithms. After evaluating the accuracy of each of the mentioned methods, the most effective one was determined – LSTM.

Keywords: text classification, natural language processing, hyperparameter tuning, machine learning, hate speech detection

Turinys

Įvadas

Darbe sprendžiamas neapykantos kalbos nustatymo uždavinys naudojant mašininio mokymo algoritmus. Augant socialinių tinkų bei naujienų portalų populiarumui, auga ir neapykantos kalbos naudojimas internete. Išaugus neapykantos kalbos naudojimui, auga poreikis kuo efektyviau nustatyti neapykantos kalbą automatiškai - pasitelkus mašininį mokymą. Jog tyrimai neapykantos kalbos tematikai yra svarbūs, patvirtina [Bac11] darbo autorė, kuri savo darbe analizuoja trijų grupių neapykantos kalbą ir tiria skatinamąsias priežastis, bei teigia, kad neapykantos kalbos tyrimai yra būtini norint suvaldyti "neigiamą poveikį visuomenei, organizacijoms, bei asmenims". Problemai spręsti yra įvairių būdų, tačiau šiame darbe problema yra sprendžiama naudojant mašininio mokymo algoritmus. Taip pat siekiama palyginti modelių efektyvumą su kitais algoritmais bei nustatyti geriausią metodą neapykantos kalbos aptikimui. Darbo uždaviniai norint pasiekti užsibrėžtą tikslą:

1. Paruošti duomenų imtį, sudarytą iš komentarų, paimtų iš socialinių tinklų bei naujienų portalų.

2. Apžvelgti kuo daugiau analogiškos literatūros ir išsirinkti potencialius metodus sudarytai duomenų aibei pritaikyti.

3. Analizuojant analogišką literatūrą ir turimus komentarus rasti vietas duomenyse, kuriose būtini modifikavimai, norint pasiekti geresnius modelių rezultatus, bei aptikti skirtumus duomenų apdorojime lietuvių ir anglų kalboms, kadangi didžioji dalis analogiškos literatūros neapykantos kalbos uždaviniui spręsti yra anglų kalba ir taikyti metodai taip pat anglų kalbai.

4. Realizuoti algoritmus neapykantos kalbos aptikimo uždaviniui spręsti, rasti parametrus, duodančius geriausius rezultatus.

5. Analizuoti klasifikatorius, jų trūkumus, privalumus, bei naudojant įvertinimo parametrus išskirti vieną geriausiai pasirodžiusį klasifikatorių.

6. Atlikus realizuotų modelių efektyvumo analizę, pateikti rekomendacijas, kaip spręsti neapykantos kalbos nustatymo uždavinį lietuviškam tekstui.

Sprendžiant neapykantos kalbos nustatymo uždavinį lietuvių kalbai, sunku rasti rekomendacijų geriausių metodų parinkimui, kadangi literatūros, kurioje neapykantos kalbos aptikimo metodai būtų taikomi lietuviškam tekstui nėra daug. Dėl šios priežasties yra svarbu atlikti tyrimus neapykantos kalbai nustatyti pasitelkiant mašininio mokymosi metodus.

1. Literatūros apžvalga

1.1. Mašininis mokymas

Terminą mašininis mokymasis 1959 m. sukūrė kompiuterinių žaidimų ir dirbtinio intelekto pradininkas Arthur Samuel. Jis 1952 parašė pirmąją mokymosi kompiuteriu programą. Programa buvo šaškių žaidimas, o IBM kompiuteris, kuriuo daugiau žaisdavo, bei tyrinėdavo, kurių ėjimų metu buvo sudarytos laimėjimo strategijos, vis labiau tobulėjo žaidime, įtraukdamas tuos judesius į savo programą.

Mašininis mokymasis (angl. Machine Learning) - tai dirbtinio intelekto metodų klasė, kuriai būdingas ne tiesioginis problemos sprendimas, o mokymasis. Mašininis mokymasis yra būtinas šiuolaikinio verslo ir tyrimų aspektas daugeliui organizacijų. Jis naudoja algoritmus ir neuroninių tinklų modelius, kad padėtų kompiuterių sistemoms gerinti savo našumą. Mašininio mokymosi algoritmai automatiškai sukuria matematinį modelį naudodami pavyzdinius duomenis, taip pat žinomus kaip mokymosi duomenis. Kad priimtų sprendimus, nereikia jų tam specialiai užprogramuoti.

[JM15] 2015 metais publikuotame darbe, kuriame nagrinėjamos mašininio mokymosi tendencijos ir perspektyvos, apie mašininį mokymąsi rašoma, kad jis sprendžia klausimą, kaip sukurti tobulėjančius kompiuterius, automatiškai tobulėjančius per patirtį ir, kad tai yra viena iš sparčiausiai augančių technikos sričių. Mašininio mokymosi pažangą lėmė naujų mokymosi algoritmų ir teorijos kūrimas bei nuolatinis internetinių duomenų ir pigių skaičiavimų prieinamumas.

1.1.1. Neprižiūrimas mašininis mokymas

Mašininis mokymas vadinamas neprižiūrimu, kai turimi įvesties duomenys (x) neturi jiems priskirtų išvesties reikšmių. Čia nėra teisingų ar neteisingų išvesčių. Pagrindinis šio mokymosi tikslas yra modeliuoti duomenų struktūrą arba pasiskirstymą norint gauti daugiau informacijos apie turimus duomenis. Šio tipo uždaviniai grupuojami į klasterizavimą (angl. clustering), kai norima sugrupuoti turimus duomenis į jiems būdingas duomenų grupes, bei asociaciją (angl. association) kai uždavinys yra rasti taisykles, apibūdinančias didelę duomenų dalį.

1.1.2. Prižiūrimas mašininis mokymas

Mašininis mokymas vadinamas prižiūrimu kai visi modeliui paduodami duomenys yra sužymėti. Šio darbo atveju tai būtų būtent šis metodas, kai visi programai paduodami komentarai turi sužymėtus 1 ir 0. Jie analogiškai reiškia neapykantos ir neutralią kalbą. Kiekviena duomenų įvestis (x) turi jau priskirtą duomenų išvestį (Y). Šio mašininio mokymosi tipo tikslas yra turimą funkciją $Y = f(x)$ apibrėžti taip tiksliai, kad gavus naują įvesties reikšmę (x), būtų galima jai priskirti rezultatą (Y). Toliau šio tipo uždaviniai grupuojami į regresiją, kai išvesties reikšmes turime skaičius, bei klasifikavimą, kai išvestį galime apibrėžti kaip turimų duomenų grupavimą. Šį būdą naudojame ir šiame darbe. Duomenys yra skirstomi į dvi grupes: neapykantos ir neutralios

kalbos.

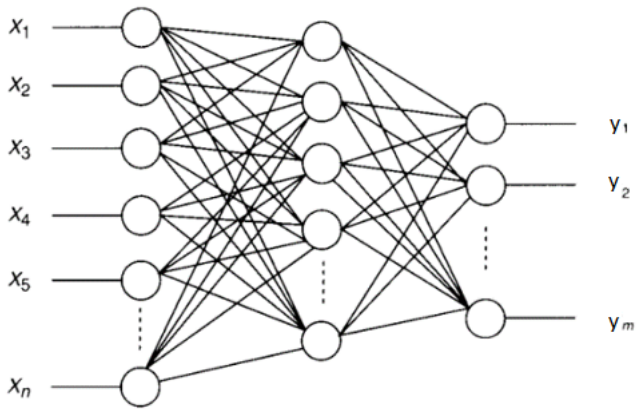
1.2. Neuroniniai tinklai

Neuroniniai tinklai sudaro giliojo mokymosi pagrindą, kuris yra mašininio mokymosi dalis. [Bis94] darbo autoriaus žodžiais, neuroniniai tinklai suteikia daugybę naujų galingų metodų, skirtų problemoms spręsti pagal šablono atpažinimą, duomenų analizę ir valdymą. Jie turi keletą svarbių savybių, įskaitant aukštą apdorojimo greitį ir galimybę išmokti problemos sprendimą iš pavyzdžių rinkinio. Darbe nagrinėjami neuroninių tinklų taikymai, tarp jų yra ir taikymai, apimantys klasifikavimo uždavinius. Iš daugelio neuroninių tinklų modelių, kurie buvo sukurti, daugiausia dėmesio skiriama ir plačiai aprašomi daugiasluoksnis perceptronas (angl. multilayer perceptron, MLP) ir radialinio pagrindo funkcijų tinklas (angl. radial basis function network). Autoriaus tyrimo publikavimo metais (1994), šie tinklai sudarė pagrindą daugumos praktinių uždavinių pritaikymui. Atsižvelgus į tai buvo pasirinkti šie neuroninių tinklų modeliai.

Dirbtinis neuroninis tinklas – tai dirbtinių neuronų grupė, sujungta tarpusavyje, kurių veikimas apibūdinamas panašiai kaip žmogaus smegenų neuronų veikla. Jis ir buvo kuriamas siekiant algoritmą išmokyt mąstyti kaip žmogų. Pasitelkę dirbtinius neuronus, kurių struktūra yra matematinės formulės, turime, kad padavę n įėjimų $(x_0, x_1, x_2, ..., x_n)$ neuroniniui tinklui, gausime vieną išėjimą (y). Išėjimo reikšmė gaunama pagal formulę:

$$y = \varphi\left(\sum_{j=0}^{n} w_j x_j\right)$$

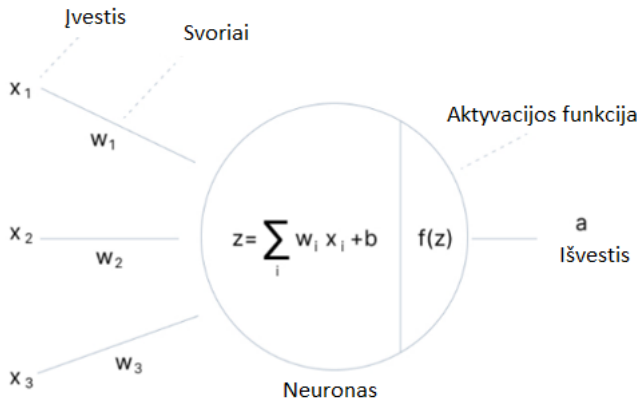
Koeficientai ω vadinami įėjimų svoriais, o funkciją φ - aktyvacijos funkcija. Neuroninis tinklas turi pagrindinius tris sluoksnius. Sluoksnis, kuriame duomenys yra paduodami algoritmui, vadinamas įvesties sluoksniu (angl. Input Layer), o sluoksnis, kuriame gauname modelio rezultatų duomenis - išvesties sluoksniu (angl. Output Layer). Svarbiausias sluoksnis yra paslėptasis (angl. Hidden Layer), kuriame atliekami visų rūšių per įvesties sluoksnį įvestų savybių skaičiavimai ir perduodami rezultatai į išvesties sluoksnį.



1 pav. Neuroninis tinklas

1.2.1. Aktyvacijos funkcijos

Neuroniniuose tinkluose svarbų vaidmenį atlieka aktyvacijos funkcijos. Jos transformuoja svertinį paslėptojo sluoksnio vidurkį, kur vėliau transformuoti duomenys yra perduodami kaip kitas paslėptasis sluoksnis arba duomenys paduodami kaip rezultatai išvesties sluoksnyje.



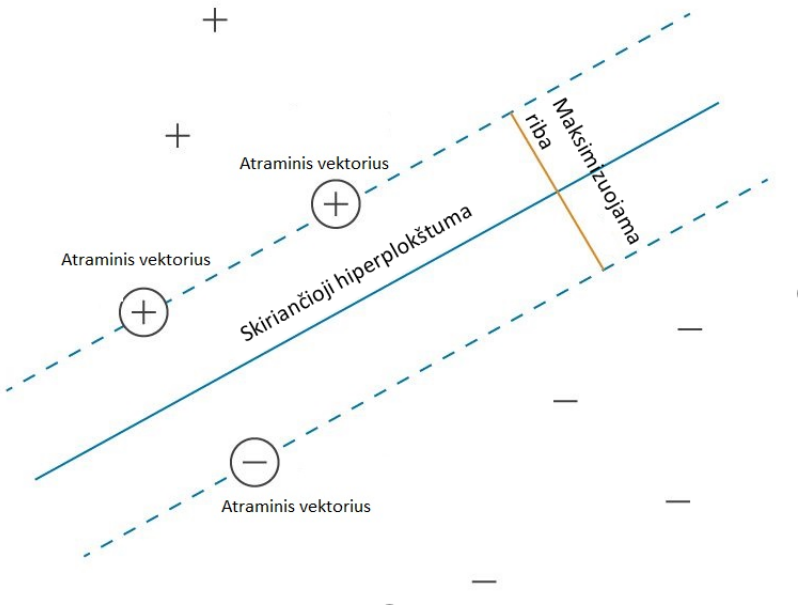
2 pav. Neuroninio tinklo veikimas

Nenaudojant aktyvacijos funkcijų neuroniniuose tinkluose, kiekvienas neuronas atliks tik tiesinę įvesties transformaciją naudodamas svorius ir paklaidas. Nors neuroninis tinklas taptų paprastesnis, tačiau sudėtingesnės užduoties tokiam modeliui išmokti ir efektyviai vykdyti būtų neįmanoma, nes neuroninio tinklo modelis atitiktų tiesiog tiesinės regresijos modelį. Taigi siekdami efektyvesnių rezultatų, naudojame aktyvacijos funkcijas. Viena iš dažniausiai pasitaikančių aktyvacijos funkcijų yra sigmoidinė (angl. sigmoid) aktyvacijos funkcija. Tai yra matematinė funkcija, turinti būdingą "S" formos kreivę (kitaip sigmoidinę kreivę). Naudodami ją, išvestyje gauname 1 arba 0. Šios reikšmės apskaičiuojamos naudojant aktyvacijos funkcijos formulę:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} = 1 - S(-x)$$

1.2.2. Atraminių vektorių klasifikatorius

Atraminių vektorių klasifikatorius laikomas paprastesniu nei neuroninių tinklų algoritmai ypač jei programuotojai nėra su jais susidūrę. Atraminių vektorių klasifikatorius yra mašininio mokymosi algoritmas skirtas prižiūrimo mokymosi metodams, kai duomenys yra iš anksto sužymėti grupėmis. Turint vienmačius, juos įmanoma realizuoti ranka, tačiau turint daugiau dimensijų patogu naudoti automatizuotus algoritmus. Taigi šis klasifikatorius nustato sprendimo ribą (angl. „decision boundary“) arba hiperplokštumą, kad atstumas nuo jos iki dviejų taškų priklausančių skirtingoms duomenų grupėms, būtų didžiausias. Pavyzdžiui komentarai priklausantys dviem grupėm, negatyvūs ir neutralūs. Dėl to atraminių vektorių klasifikatorių galima pavadinti ir didžiausios maržos klasifikatoriumi (angl. Maximum margin classifier). Per šiuos taškus einančios tiesės vadinamos atraminiais vektoriais.



3 pav. Atraminiai vektoriai (angl Support vector)

Paprasčiausiu atveju, tokiu koks pavaizduotas paveikslėlio pavyzdyje atraminius vektorius skirianti hiperplokštuma yra sudaroma pagal $w^T X + b = 0$ formulę, kur w yra įvesties vektorių svoris, X įvesties vektoriaus reikšmė, o b yra skaliaras (angl. bias). Turime neigiamų ir teigiamų klasių reikšmių pasiskirstymą, kur teigiamai klasei priskiriamas $y = 1$, o neigiamai: $y = -1$. Klasifikuojant siekiama, kad:

$$\begin{cases} w^T * X_i + b > 0, & \text{kai } Y_i = 1, \\ w^T * Xi + b < 0, & \text{kai } Y_i = -1. \end{cases}$$

Klasifikavimo taisyklė f testavimo duomenims X apibrėžiama

$$f(X) = sgn(w^T X + b).$$

čia $sgn()$ yra simbolio funkcija (angl. sign function)

$$sgn(x) := \begin{cases} -1, & \text{kai } x < 0, \\ 0, & \text{kai } x = 0, \\ 1, & \text{kai } x > 0. \end{cases}$$

Jei taške X_i turima išraiška $w^T * X_i + b = 1$. Tuomet skaičius $\frac{w^T X_i + b}{\|w\|} = \frac{1}{\|w\|}$ žymi hiperplokštumos (sprendimo ribos) $w^T(X - X_i) = 0$ atstumą iki koordinačių pradžios taško. Skaičius

$$\frac{2}{\|w\|} = \frac{2}{\sqrt{w^T w}}$$

yra lygus atstumui tarp atraminių vektorų $w^T X + b = \pm 1$. Formulės išvedimas iš [Bas] šaltinio.

Gautas maksimalus atstumas nuo sprendimo paviršiaus iki artimiausio duomenų taško (angl. margin) yra $\frac{2}{\|w\|}$. Tačiau dažniau pasitaikantis atvejis yra kai tarp dviejų duomenų grupių neįmanomą brėžti tiesios hiperplokštumos, tokiu atveju turime netiesinį klasifikavimą, kur sprendimo riba apskaičiuojama su Lagranžo išraiška.

1.2.3. Daugiasluoksnis perceptronas

Daugiasluoksnis perceptronas yra vienas iš lengviausiai klasifikavimui pritaikomų neuroninių tinklų. Skirtingai nei atraminių vektorių klasifikatorius, MLP klasifikavimo užduočių atlikimui naudoja neuroninius tinklus. MLP sudaro mažiausiai trys mazgų (angl. nodes) sluoksniai: įvesties sluoksnis, paslėptas sluoksnis ir išvesties sluoksnis. Išskyrus įvesties mazgus, kiekvienas mazgas yra neuronas, kuris naudoja netiesinę aktyvavimo funkciją. MLP naudoja prižiūrimą mokymosi techniką, vadinamą atgaliniu mokymusi (angl. backpropagation). Atgalinis mokymasis kitaip „klaidų sklidimas atgal“, yra algoritmas skirtas prižiūrimam dirbtinių neuroninių tinklų mokymuisi naudojant gradiento nusileidimą (angl. gradient descent). Taip pat, tai yra vienas iš būdų, kaip efektyviai nustatyti nedidelius svorių pakeitimus, kad būtų sumažintos neuroninio tinklo klaidos.

Atgalinio mokymosi algoritmas, kuris aprašomas [Joh] šaltinyje, priklauso nuo penkių lygčių. Pirma, skaičiuojamos dalinės išvestinės:

$$\frac{\partial E_d}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}.$$

Galutinio sluoksnio klaidos formulė:

$$\delta_1^m = g_o'(a_1^m)(\hat{y_d} - y_d).$$

Paslėptojo sluoksnio klaidos apskaičiuojamos formule:

$$\delta_j^k = g'(a_j^k) \sum_{l=1}^{r^{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

Formulė, naudojama dalinių išvestinių jungimui kiekvienai įvesties - išvesties porai:

$$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^k} \left(\frac{1}{2} (\hat{y_d} - y_d)^2 \right) = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial E_d}{\partial w_{ij}^k}$$

čia $E(X,\theta) = \frac{1}{2N} \sum_{i=i}^{N} (\hat{y_i} - y_i)^2$ klaidos funkcijos gradientas (angl. gradient of the error function), kur y_i yra siekiama reikšmė, įvesties - išvesties porai $(\vec{x_i}, y_i)$, o $\hat{y_i}$ yra apskaičiuota neuroninio tinklo išvestis, naudojant $\vec{x_i}$ įvestį. Naudojama E_d klaidos funkcija yra lygi $E_d = \frac{1}{2}(\hat{y} - y)^2$. Formulė, naudojama svorių atnaujinimui:

$$\Delta w_{ij}^k = -\alpha \frac{\partial E(X,\theta)}{\partial w_{ij}^k}$$

Naudoti kintamieji:

w_{ij}^k : mazgo j svoris, l_k sluoksnyje, skaičiuojamas ateinančiam mazgui i,
b_i^k : mazgo i paklaida (angl. bias) sluoksnyje l_k,
a_i^k : produkto suma plius paklaida (angl. bias) i mazgui esančiame l_k sluoksnyje,
o_i^k : išvesties mazgas i, l_k sluoksnyje,
r_k : mazgų (angl. nodes) skaičius sluoksnyje l_k,
g: paslėptojo sluoksnio aktyvacijos funkcija,
g_o: išvesties sluoksnio aktyvacijos funkcija,
α: mokymosi greitis,
N: įvesties - išvesties porų skaičius,
θ: bendras žymėjimas kiekvienai gradiento nusileidimo iteracijai, atnaujinančiai svorius ir paklaidas (angl. biases).

1.2.4. Ilgalaikės-trumpalaikės atminties modelis

Dvi pagrindinės teksto klasifikavimo gilaus mokymosi architektūros yra konvoliuciniai neuroniniai tinklai (angl. Convolutional Neural Networks, CNN) ir pasikartojantys neuroniniai tinklai (angl. Recurrent Neural Networks, RNN). O būtent LSTM yra viena iš architektūrų egzistuojančių pasikartojančiuose neuroniniuose tinkluose. LSTM taip pat buvo sukurta siekiant išspręsti nykstančio gradiento problemą RNN. LSTM naudoja tris vartus, vadinamus įvesties, išvesties ir pamiršimo vartais (angl. forget gate). Šie vartai nustato, kurią informaciją išlaikyti. LSTM modelyje apibrėžiamos dvi būsenos, paslėptoji būsena (angl. hidden state), kuri yra žinoma kaip trumpalaikė atmintis, o ląstelės būsena (angl. cell state), ši yra žinoma kaip ilgalaikė atmintis. Taigi ląstelės būsenoje LSTM tinklas pirmiausia sprendžia, ar reikia saugoti informaciją iš ankstesnės laiko žymos, ar ją vertėtų pamiršti. Čia yra pamiršimo vartų lygtis:

$$f_t = \sigma(x_t * U_f + H_{t-1} * W_f)$$

Čia x_t - dabartinės laiko žymos įvestis, U_f - įvesties svoriai, H_{t-1} - ankstesnės laiko žymos paslėptoji būsena, W_t - paslėptosios būsenos svorių matrica. Tuomet σ aktyvacijos funkcija pritaikoma (laikoma, kad σ aktyvacijos funkcija yra sigmoidinė (angl. sigmoid function)), paverčianti dydi f_t į reikšmę esančią tarp 0 ir 1, bei padauginama iš ankstesnės laiko žymos ląstelės būsenos C_{t-1}. Jei $C_{t-1} * f_t = 0$, tuomet ląstelės būsenoje yra pamirštama viskas, o jei $C_{t-1} * f_t = C_{t-1}$, tai tuomet ląstelės būsenoje nepamirštama niekas.

Toliau turimi įvesties vartai, kurie naudojami kiekybiškai įvertinti naujos informacijos, kurią perduoda įvestis, svarbą. Čia yra įvesties vartų lygtis:

$$i_t = \sigma(x_t * U_i + H_{t-1} * W_i)$$

Čia x_t - įvestis esamos laiko žymos t, U_i - įvesties svorių matrica, H_{t-1} - ankstesnės laiko žymos paslėptoji būsena, W_i - paslėptosios būsenos svorių matrica, analogiškai pritaikoma aktyvacijos

funkcija σ ir gaunama reikšmė taro 0 ir 1.

Atrinkus svarbią informaciją ji yra pridedama į ląstelės būseną, kur kaupiama informacija ilgalaikėje atmintyje. Informacija pridedama su formule:

$$N_t = tanh(x_t * U_c + H_{t-1} * W_c)$$

Kadangi naudojama aktyvacijos funkcija yra tanh, tai N_t reikšmės bus tarp -1 ir 1. Jei N_t reikšmė yra neigiama, informacija pašalinama iš ląstelės būsenos, o jei vertė yra teigiama, tai informacija pridedama prie ląstelės būsenos su esama laiko žyme. Tačiau prieš pridedant naują informaciją jai taikoma atnaujinimo lygtis:

$$C_t = f_t * C_{t-1} + i_t * N_t$$

Čia C_{t-1} yra ląstelės būsena dabartinėje laiko žymėje.

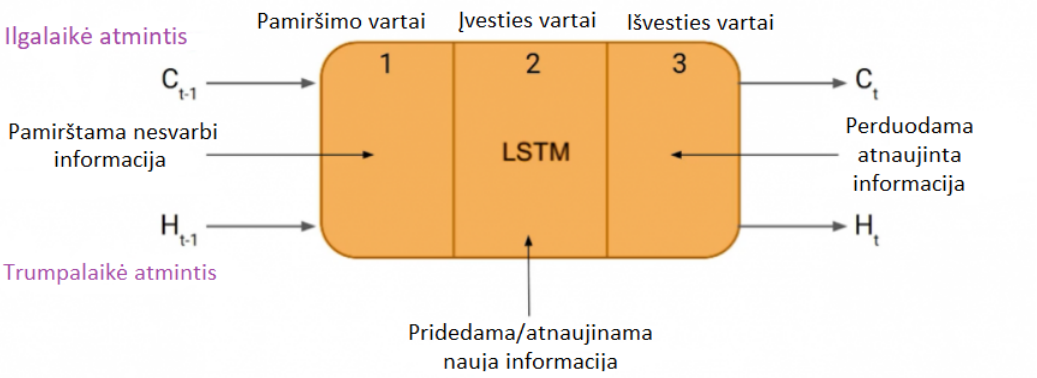
Toliau skaičiuojama išvesties vartų lygtis:

$$o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$$

Bei apskaičiuojama dabartinės laiko žymės paslėptųjų vartu lygtis naudojant aukščiau apskaičiuotą o_t, bei tangentą atnaujintos ląstelės būsenos C_t:

$$H_t = o_t * tanh(C_t)$$

Taigi, paslėptoji būsena yra ilgalaikės atminties C_t ir dabartinės laiko žymės išvesties o_t funkcija. Ryškesniam vaizdui žemiau esančiame paveikslėlyje 4 iš [Sax21] šaltinio, vizualizuojama LSTM ląstelėje, bei joje esantys vartai, bei jų funkcijos:



4 pav. LSTM ląstelė (angl. LSTM cell)

1.3. Natūraliosios kalbos apdorojimas

Norint tekstą panaudoti mašininio mokymo algoritme, pirmiausia reikia jį apdoroti taip, kad algoritmas jį suprastų. Šį procesą vadiname natūraliosios kalbos apdorojimu (angl. Natural lan-

guage processing, NLP). Natūralios kalbos apdorojimas [Dzi19] darbe apibrėžiamas kaip dirbtinio intelekto ir kalbotyros sritis informatikos moksluose, sukurta norint padėti kompiuteriams suprasti žmonių kalba parašytus teiginius ar žodžius bei palengvinti žmonių darbą su kompiuteriais, suteikiant galimybę žmonėms bendrauti su sistemomis naudojant jiems žinomą kalbą. Natūralios kalbos apdorojimas nagrinėja žmonių kalbą naudodamas įvairius skaičiavimo algoritmus, kurie išverčia tekstą žmonių kalba į kompiuteriui suprantamus duomenis. Šis metodas yra plačiai taikomas įvairiose srityse problemoms spręsti, [KKK+17] analizuojami keli pagrindiniai pritaikymo būdai, tokie kaip:

1. Mašininis vertimas (angl. Machine Translation), kuris vienos kalbos sakinius išverčia kita, jis naudojamas norint įveikti kalbos barjerą, tačiau sprendžiant šį uždavinį susiduriama su nemažai problemų norint išverstiems sakiniams išlaikyti jų reikšmę, bei nepažeisti gramatikos taisyklių.

2. El. pašto šiukšlių aptikimas (angl. Spam Filtering) .

3. Atsakymai į klausimus (angl. Dialogue System). Šiuo metu žinomi pritaikymai būtų Google assistant, Windows Cortana, Apple Siri ir Amazon Alexa.

4. Informacijos apibendrinimas (angl. Summarization).

5. Informacijos gavimas (angl. Information Extraction).

6. NLP pritaikymas egzistuoja ir medicinoje.

1.3.1. Teksto apdorojimas

Natūraliosios kalbos apdorojimo pritaikyme dažnai kyla problemų dėl kalbos laikų, deminutyvų, skirtingų žodžių galūnių. Norit šią problemą išspręsti reikia, kad algoritmas suprastų žodžius su vienodomis reikšmėmis, bet ne su identiška rašyba. Tam yra naudojamas lematizavimas (angl. Lemmatization) ir kamienavimas (angl. Stemming). Tie metodai populiarūs natūraliosios kalbos apdorojime. Jie ypač išplėtoti anglų kalbai, tačiau pritaikyti juos lietuvių kalbai yra sunku. Lentelėje vaizduojamas skirtumas tarp kamienavimo ir lematizavimo.

Žodis	Kamienavimas	Lematizavimo
Rašydamas	Raš	Rašyti
Kėdės	Kėd	Kėdė
Changer	Chang	Change

1 lentelė. Skirtumai tarp kamienavimo ir lematizavimo

Apie šiuos automatinius šaknies/kamieno nustatymo metodus plačiau rašo [KSN+20] mokslinis darbas, kuriame nagrinėjami bendras natūralios kalbos apdorojimas, lematizavimo ir kamienų sudarymas. Taip pat jame paaiškinama natūralios kalbos apdorojimo koncepcija, jos raida bėgant

metams, taikymas, privalumai ir trūkumai. Čia kamienavimas apibrėžiamas kaip procesas, kurio metu gaunami šakninio / pagrindinio žodžio variantai. NLTK (angl. Natural Language Toolkit), turi „SnowballStemmer“ klasę, Snowball kamienavimo algoritme, kurią galima pritaikyti net 16 kalbų, tačiau lietuvių kalaba nėra viena iš jų. Lematizavimas - tai linksniuojamų žodžio dalių sujungimas taip, kad jas būtų galima atpažinti kaip vieną elementą, vadinamą žodžio lema arba jo žodyno forma. Šis procesas yra panašus į kamienavimą. Jis sujungia tekstą, turintį panašias reikšmes, vienu žodžiu. Jis apibrėžiamas kaip algoritmo metodas, leidžiantis rasti žodžio, kuris yra šakninis žodis, o ne šaknies kamieno lemą. Jis pagrįstas numatoma žodžio prasme, kurią bandoma perteikti.

Lematizavimas išsiskiria iš kitų algoritmų savo sudėtingumu, nes pirmiausiai yra nustatoma žodžio kalbos dalis. Nagrinėjant teksto apdorojimą lietuvių kalbai, sunku rasti daug analogiškos literatūros su šaknies/kamieno nustatymo algoritmais ir jų taikymais. Tai pastebi 2010 metais publikuoto mokslinio darbo [Pet10] autorius, kuris nagrinėja lietuvių kalbos tekstų automatinį temos nustatymą naudojantis ATN įrankiu ir taip pat susiduria su sunkumais dirbant su lietuvišku tekstu, kadangi laisvai prieinamų įrankių nėra norint automatiškai nustatyti šaknį/kamieną žodyje. Išanalizavus sukurtus algoritmus žodžių apdorojimui, buvo sukurtas šaknies/kamieno nustatymo algoritmas lietuvių kalbai, taip pat sukurta programinė įranga ”Stemeris” algoritmo tikslumui tikrinti, algoritmo efektyvumui įvertinti ir autoriaus darbui optimizuoti. Autorius dažnai pamini, kad sunkumų kėlė panašios literatūros nebuvimas.

1.3.2. Teksto klasifikavimas

Tarp dažniausiai sprendžiamų NLP uždavinių yra teksto klasifikavimas (angl. Text Classification), kurio pagrindinė užduotis yra nustatyti iš anksto nurodyta tinkamą grupę. Teksto klasifikavimo problema yra plačiai sprendžiama ir išvystyta anglų kalbai, tačiau lietuvių kalbai dėl jos sudėtingumo reikia papildomų tyrimų norint išspręsti teksto klasifikavimo problemą. [KR19] moksliniame darbe yra nagrinėjamas didelį kategorijų kiekį turinčių draudimo bendrovės klientų užklausos, gautos elektroniniais laiškais. Tyrimui pasirinkta Lietuvoje veikianti draudimo įmonė. Tyrimo tikslas - gaunamus laiškus iš pašto dėžutės suklasifikuoti į kategorijas, kurios atspindėtų laiško temą. Jam pasiekti naudojama duomenų imtis, sudaryta iš beveik 51 tūkstančių skambučių centro užklausų, kurios buvo suskirstytos į 33 kategorijas, bei penki prižiūrimo mokymosi algoritmai, kurie buvo pasirinkti išanalizavus analogišką literatūrą ir joje aprašytus teksto klasifikatorius turėjusius gerus rezultatus. Tyrimui atrinkti algoritmai:

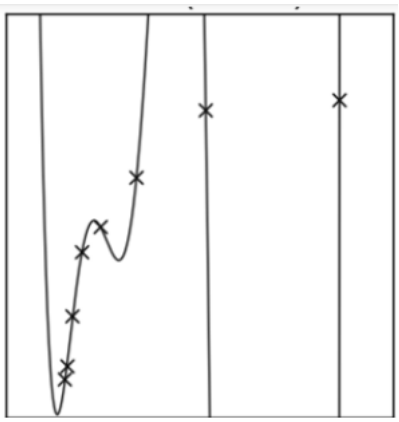
1. Naivaus Bajeso (angl. Naive Bayes) algoritmas.
2. Atraminių vektorių mašinos (angl. Support Vector Machines) algoritmas.
3. Logistinė regresija (angl. Logistic Reegression).
4. Stochastinio gradiento nusileidimo (angl. Stochastic Gradient Descent) algoritmas.

5. Atsitiktinio miško (angl. Random Forest) algoritmas.

Tačiau lyginant visus algoritmus tarpusavyje, šiame tyrime efektyviausiai pasirodė SVM algoritmas. Taip pat pastebėta, kad algoritmo kokybei įtakos turi duomenų apdorojimas. Pašalinus įmonės saugumo pranešimus, laiškų pasirašymus ir ne tokius reikšmingus klasifikavimui lietuviškus žodžius, tikslumas išaugo.

1.3.3. Permokymas (angl. overfitting)

Klasifikuojant ar atliekant kitus mašininio mokymo uždavinius ,svarbu, kokie duomenys yra paduodami, nes dažnai padavus tam tikrus parametrus ir tam tikrą duomenų aibę, dažnai yra susiduriama su problemomis. Persimokymas - tai viena iš pagrindinių problemų mokant neuroninių tinklų modelį. Modelis tampa pernelyg geras mokymosi stadijoje, tačiau kai taikome jį naujiems duomenims, matome, kad modelis negali apibendrinti ir tiksliai numatyti duomenų išvesties. Gaunamas tikslumas yra mažas. Grafiškai šią problemą pavaizduojama [Šal15] darbe:

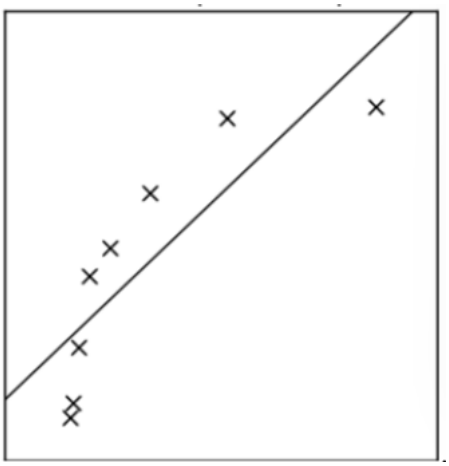


5 pav. Modelis per daug prisitaikęs prie apmokymo aibės

Taigi jei patvirtinimo (angl. validation) metrika yra daug blogesnė nei mokymosi metrika, tai rodo, kad modelis per daug prisitaikęs prie mokymosi duomenų (angl. overfitting). Šią problema spręsti galime pridėdami daugiau duomenų į treniravimo imtį, tuomet modelis turėdamas daugiau duomenų bus tikslesnis, nes daugiau duomenų prideda ir didesnę įvairovę treniravimo imčiai, bei modelis apsimoko tiksliau. Kitas būdas būtų papildyti turimus duomenis (angl. data augmentation), šiuo būdu nepridedame visiškai naujų duomenų, tačiau papildome savo duomenų aibę jau turimais, pamodifikuotais, tarkime jei treniravimo aibei paduodame paveikslėlius, tai būtų tikslinga paduoti tą patį paveikslėlį pasuktą ar apverstą, tuomet modelis gavęs naujus duomenis tiksliau atpažintų nuotrauką paduodamą iš įvairių kampų. Turimam uždaviniui šio sprendimo pritaikymas būtų paėmus turimą komentarą ir žodžius išdėliojus kita tvarka pridėti jį į testavimo aibę. Taip pat, galima supaprastinti turimą modelį, pavyzdžiui sumažinus sluoksnių skaičių ar neuronų. Be to, alternatyvus sprendimo būdas būtų naudoti reguliarizaciją (angl. Regularization), tai yra metodas padedantis sumažinti neuroninio tinklo persimokymą arba dispersiją, sumažinant jo sudėtingumą. Dažniausiai pasitaikantis reguliarizacijos metodas vadinamas L2 reguliarizacija.

1.3.4. Neapsimokymas (angl. underfitting)

Toliau neapsimokymas yra priešingas persimokymui



6 pav. Modelis nepakankamai prisitaikęs prie apmokymo aibės

neapsimokymą atpažinti nesunku pastebėjus, kad jis nesuklasifikuoja duomenų su kuriais buvo apmokytas ir juo labiau nesugeba nustatyti naujai paduodamų duomenų prognozės šiuo atveju taikome priešingus metodus nei buvo taikyta persimokytam modeliui, pavyzdžiui sprendimo būdas galėtų būti modelio parametrų pridėjimas, norint apsunkinti modelį, toks sprendimo būdas yra tinkamas kadangi turint sudėtingus duomenis imtyje jam atitinkamai reikia pakankamai sudėtingo algoritmo, kuris sugebėtų tinkamai klasifikuoti, bei nustatyti tinkamą išvestį.

1.4. Neapykantos kalbos aptikimas angliškam tekstui

Spendžiant neapykantos aptikimo uždavinį neretai susiduriama su iššūkiais, nesvarbu kuriai kalbai teksto klasifikavimas yra taikomas. Tačiau [MYY+19] darbe aptariamos kylančios problemos būtent anglų kalbai. Pirmas iššūkis yra neapykantos kalbos apibrėžimas, kadangi paanalizavus skirtingų šaltinių apibrėžimus neapykantos kalbai yra matomi skirtumai. Taip pat pabrėžiama, kad yra svarbu apibrėžti neapykantą kurstančią grupę, kadangi tokios grupės šlovinimas turi būti laikomas neapykantos klaba. Taigi vien pagyrimų sakinių apdorojimas gali būti sudėtingas procesas. Taip pat pabrėžiami iššūkiai renkant duomenų imtį, kadangi vis daugėja socialinių tinklų, kurie draudžia neapykantos kurstymo sakinius. Tačiau vis dar egzistuoja viešai prieinami duomenų rinkiniai, kurie gali būti naudojami neapykantos kalbos aptikimui anglų kalbai. Tiriami buvo keturi duomenų rinkiniai: Stormfront(šio rinkinio šaltinis yra online forumai), TRAC (duomenų šaltinis yra Facebook), o HatEval ir HatebaseTwitter rinkiniai sudaryti iš Twitter socialinio tinklo komentarų, jiems taikomi BERT, neuroninių tinklų kombinavimą (angl. Neural Ensemble) ir kelių rodinių atraminių vektorių klasifikatorius (angl. Multi-view SVM, mSVM) modeliai. Rezultatų vertinimui palyginti skaičiuojamas tikslumas, bei makrovidutinis F1 balas (angl. macro-averaged F1 score). Taigi atlikus tyrimą išryškėjo, kad geriausi rezultatai buvo pasiekti su HatebaseTwitter duomenų rinkiniu, kuriam buvo pritaikytas neuroninių tinklų kombinavimo modelis pasiekęs

0.9217 tikslumą, bei 0.9118 makrovidutinį F1 balą. Panašiai aukštas rezultatas pasiektas taikant BERT metodą. Šis duomenų rinkinys buvo suskirstytas į tris grupes: į neapykantos, agresyvius, bei komentarus neturinčius nei vieno požymio iš ankstesnių dviejų. Tai nėra nei neapykantos kalbos, nei agresyvūs komentarai. Kadangi taikant tuos pačius metodus gaunami skirtingi rezultatai, skirtingiems duomenų rinkiniam svarbu atidžiau paanalizuoti, kokie duomenys buvo naudojami:

Duomenų rinkinys	Grupės ir jų procentai duomenų rinkinyje
Stormfront	Neapykantos komentarai 11% Ne neapykantos komentarai 86% Neapykantą kurstantis komentaras, kai jis derinamas su aplinkiniais sakiniais 2% Komentarai, kurie yra ne anglų kalba arba kuriuose nėra informacijos, susijusios su ne neapykantą ar neapykantą kurstančia kalba 1%
TRAC	Neagresyvūs komentarai 69% Agresyvūs komentarai 16% Šiek tiek agresyvūs komentarai 16%
HatEval	Neapykantos komentarai 43% Ne neapykantos komentarai 57%
HatebaseTwitter	Neapykantos komentarai 5% Agresyvūs komentarai 76% Neutralūs komentarai 17%

2 lentelė. Duomenų rinkinių palyginimai

Prasčiausi rezultatai gauti iš TRAC duomenų rinkinio. Modelių tikslumas siekia apie 60%. Tyrime pastebėta, kad daugumos duomenų rinkinių ir metodų tikslumas yra nukreiptas į mokymo duomenų klasę, kurioje yra daugiausia komentarų. Tai rodo, kad vertinant tikslumus reikia labiau subalansuotų duomenų rinkinių.

Neapykantos kalbos uždavinio sprendimų yra labai daug, tarkime [MZ17] tyrime pritaikytas atraminių vektorių klasifikatorius (SVM) ir naudojami trys pritaikyti parametrai: surface n-grams, word skip-grams, Brown clusters. Pagrindinė paprasto n-gramų (angl. n-gram) parametro esmė yra suskirstyti gretimus žodžių, raidžių ar simbolių sekas dokumente į pasirinktą n dydžio seką, algoritmui paduodant ne atskirus žodžius, o suskirstyto dydžio n sekas, frazes prognozuoti išvestį, kadangi kartais svarbesni ne pavieniai žodžiai, o jų junginiai. Šie metodai buvo pritaikyti duomenų rinkiniui sudarytam iš 14509 Twitter socialinio tinklo žinučių, kurios buvo suskirstytos į tris grupes: į neapykantos kalbos žinutes, į agresyvias žinutes ir neutralias. Prieš taikant algoritmą, atliktas duomenų apdorojimas. Geriausias rezultatas buvo pasiektas su 4 gramų parametru ir SVM modeliu, pasiekiančiu 78% tikslumą. Didžiausias iššūkis su kuriuo buvo susidurta modelyje prognozuojant išvestį buvo neapykantos kalbos atskyrimas nuo paprastų keiksmažodžių, kurie nėra neapykantos kalba.

Nagrinėjant neapykantos kalbos aptikimą įdomu apžvelgti ne tik anglų klasifikavimo algoritmų pritaikymą, bet ir kitų užsienio kalbų iššūkius dirbant su tekstu. Trijų kalbų (italų, anglų

ir vokiečių) neapykantos aptikimo įvertinimas internete nagrinėjamas 2020 metais publikuotame darbe [CMC+20]. Tyrimui naudojama internete laisvai prieinama duomenų imtis su 16 tūkstančių komentarų anglų kalbai, 4 tūkstančių italų kalbai ir 5009 vokiškų komentarų.

Kalba	Neapykantos kalbos komentarai (%)	Kiti komentarai (%)	Iš viso komentarų
Anglų	5,006 (32%)	10,884(68%)	16,000
Italų	1,296 (32%)	2,704(68%)	4,000
Vokiečių	1,688 (34%)	3,321(66%)	5,009

3 lentelė. Duomenų imties pasiskirstymas

Tikslas yra palyginti įvairių ypatybių, žodžių įterpinių (angl. word embeddings) ir neapykantos kalbos aptikimo duomenų apdorojimo technikas (angl. pre-processing techniques). Tikslui pasiekti naudojami ilgalaikės-trumpalaikės atminties algoritmas (angl. Long Short Term Memory, LSTM), vartiniai pasikartojantys vienetai (angl. Gated Recurrent Unit, GRU) ir dvikryptė ilgalaikė-trumpalaikė atmintis (angl. bidirectional Long Short Term Memory, BiLSTM), Fasttext žodžių įterpiniai. Atlikus bandymus su įvairiais modeliais, parametrais ir duomenų apdorojimo būdais, aptariami geriausi rezultatai visoms kalboms. Anglų kalbai geriausias rezultatas 0,823 naudojant F1 vertinimą su LSTM algoritmu ir „Fasttext“ įterpiniais. Italų kalba geriausias rezultatas 0,805 su F1 vertinimu gaunamas naudojant LSTM ir žodžių įterpimus, kurie buvo apmokyti ant didelių duomenų susidedančių iš itališkų komentarų. Vokiečių geriausias rezultatas yra pasiekiamas naudojant GRU tinklą, naudojant standartinius Fasttext įterpimus. Su tokiais rezultatais galima daryti išvadą, kad taikyti metodus, kurie puikiai pasirodė užsienio kalbai gali būti neefektyvu, kadangi rezultatai nebūtinai bus tokie pat geri.

1.5. Neapykantos kalbos aptikimas lietuviškam teksui

Analogiškame 2021 metų publikuotame moksliniame darbe [Bla21] identifikuojamos smurtinio pobūdžio žinutės. Komentarų duomenų aibėje iš internetinių portalų iš viso surinkta ir sužymėta 8540, iš kurių 734 pažymėti kaip smurtiniai, o iš visų internetinių parduotuvių bei atsiliepimų svetainių surinkta 151222 atsiliepimų. Rezultatai gauti lyginant keturis mašininio mokymosi algoritmus pritaikytus apdorotiems duomenims:

1. Naivus Bajeso algoritmas (angl. Naive Bayes) algoritmas.
2. Atraminių vektorių algoritmas (angl. Support Vector Machines).
3. FastText.
4. Hibridinis klasifikatorius, kuris sujungtas į vieną klasifikatorių iš Naivus Bajeso, Atraminių vektorių, bei FastText algoritmų.

Prieš taikant metodus turimi duomenys yra apdorojami, t.y. pašalinami diakritiniai ženklai, komentaro tekste esantys ”emoji”, bei ”emoticon” paverčiami į žodžius, bei naudojamas kamienavimas. Taip pat stebima, ar komentaro netekstinių parametrų įterpimas į komentaro tekstą turi įtakos galutiniam tikslumui. Pavyzdžiui, yra įterpiamos komentaro ”reakcijos” ir tai ar vartotojas naujienų portale rašo komentarus kaip anonimas. Be to, į komentaro tekstą įterpiamos sentimento žymės. Sentimentas apibrėžiamas kaip natūralusis skaičius nuo 1 (labai negatyvus) iki 5 (labai pozityvus). Tikslumui naudojami statistiniai matai tokie kaip tikslumas, atpažintų smurtinių komentarų dalis, F1 balas. Tyrime, aukščiausius rezultatus tikslumo (0.945), atpažintų komentarų dalimi (0.667) bei F1 balo atžvilgiu (0.608) pasiekė atraminių vektorių algoritmas. Darbe daroma išvada, kad siekiant aukščiausios smurtinio turinio atpažinimo sistemos kokybės, turėtų būti naudojamas atraminių vektorių mašininio mokymosi algoritmas. Geriausiems algoritmams tiriama duomenų apdorojimo įtaka tikslumui, o gautuose rezultatuose matyti, kad neigiamo poveikio duomenų apdorojimas nepadarė bei teigiamą poveikį turėjo tik sentimento žymės įterpimas bei diakritinių ženklų panaikinimas.

2. Tiriamoji dalis

2.1. Naudojami mašininio mokymosi algoritmai

Neapykantos kalbos nustatymui taikomi jau sukurti modeliai, analizuojami jų parametrai ir efektyvumas.

2.1.1. Scikit-Learn bibliotekos algoritmai

Analizuojamas daugiasluoksnis perceptronas (angl. Multilayer perceptron, MLP) bei atraminių vektorių klasifikatorius (angl. Support Vector Machine classifier, SVM). Šis klasifikatorius paimtas, nes nagrinėjamoje literatūroje parodė puikius rezultatus. Bus stebima, ar taip yra iš tikrųjų taikant turimus duomenis šiame darbe. Abiem klasifikatoriams naudojama Scikit-Learn biblioteka, kuri yra populiari taikant mašininio mokymosi algoritmus Python programavimo kalboje. Šioje bibliotekoje yra daug efektyvių mašininio mokymosi ir statistinio modeliavimo įrankių, įskaitant regresiją, grupavimą bei klasifikavimą. Taip pat ši biblioteka palaiko prižiūrimą (angl. Supervised Machine Learning) ir neprižiūrimą (angl. Unsupervised Machine Learning) mašininį mokymąsi.

2.1.2. Keras bibliotekos algoritmai

Keras yra galinga nemokama atvirojo kodo Python biblioteka, skirta kurti ir įvertinti giliojo mokymosi (angl. Deep learning) modeliu. Naudojant šią biblioteką, analizuojamas ilgalaikės-trumpalaikės atminties algoritmas (angl. LSTM).

2.2. Naudojami vektorizacijos būdai

2.2.1. Tokenizacija

Tokenizacija (angl. tokenization) tai kiekvieno sakinio padalijimas į žodžius arba mažesnes dalis, kurios yra žinomos kaip žetonai (angl. tokens). Užbaigus žetonų keitimą, iš duomenų aibės išskiriami visi unikalūs žodžiai. Jei tekstas yra padalintas į žodžius, tada jis vadinamas žodžių tokenizavimu (angl. Word Tokenization), o jei jis suskaidytas į sakinius, tada jis vadinamas sakinio tokenizavimu (angl. Sentence Tokenization). Pagal atliekamą užduotį yra pasirinkamas tinkamas metodas.

2.2.2. Vektorizacija

Vektorizavimas (angl. Text Vectorization) arba kitaip žodžių įterpimas (angl. Word embeddings) yra teksto duomenų konvertavimas į skaitmeninius vektorius procesas.

2.2.2.1. CountVectorizer

CountVectorizer vektorizacijos metodas yra Python scikit-learn bibliotekos įrankis. Jis naudojamas duotą tekstą paversti vektoriumi, remiantis kiekvieno žodžio, pasitaikančio tekste, dažniu. Naudojamas pavyzdys, kurį sudaro 5 sakiniai (”Labas, geros geros dienos”, ”Labas, šiandien šalta”, ”Šiandien lauke šalta ”, ”Lauke labai labai labai gera”, ”Labas, gera tave matyti”), pritaikius algoritmą gaunami vektoriai analogiškai kiekvienam sakiniui.

	dienos	gera	geros	labai	labas	lauke	matyti	tave	šalta	šiandien
0	1	0	2	0	1	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1	1
2	0	0	0	0	0	1	0	0	1	1
3	0	1	0	3	0	1	0	0	0	0
4	0	1	0	0	1	0	1	1	0	0

7 pav. CountVectorizer metodo veikimas

Gavus vektorius kiekvienam sakiniui, šie vektoriai toliau keliauja į modelį, kadangi paduodami duomenis jau nebėra tekstinio formato ir modelis sugeba su jais dirbti.

2.2.2.2. TF - IDF

Analogiškoje literatūroje galima rast TF IDF (angl. Term frequency-inverse document frequency) vektorizacijos ir CountVectorizer vektorizacijos palyginimus. Daug kur TF IDF yra paminimas kaip geresnis. Vektorizacijos metodas jungia dvi sąvokas: terminų dažnis (TF) ir dokumento dažnis (DF). Termino dažnis yra konkretaus termino/žodžio pasikartojimų duomenyse skaičius. Terminų dažnis rodo, kiek svarbus konkretus žodis duomenyse. Šis gautas vektorius sutaptų su CountVectorizer vektoriumi. Toliau apibrėžiamas komentarų dažnumas – tai komentarai, kuriuose yra tam tikras terminas/žodis, skaičius. Komentaro dažnumas rodo, koks dažnas šis terminas. Atvirkštinis komentaro dažnis (IDF) yra termino svoris, juo siekiama sumažinti žodžio svorį, jei žodis pasikartoja visuose komentaruose. IDF apskaičiuojamas:

$$idf_i = log(\frac{n}{df_i})$$

Kur idf_i yra žodžio i IDF balas, df_i yra komentarų, kuriuose yra terminas i, skaičius, o n yra bendras komentarų skaičius. Kuo didesnis žodžio DF, tuo mažesnis IDF. Kai DF skaičius yra lygus n, o tai reiškia, kad žodis yra visuose komentaruose/sakiniuose, IDF bus lygus nuliui, nes žodis pasikartojęs visuose komentaruose neturės daug reikšmės kategorizavimui. TF-IDF galima apskaičiuoti taip: $w_{i,j} = tf_{i,j} * idf_i$. Kur $w_{i,j}$ yra i žodžio TF-IDF balas komentare j, $tf_{i,j}$ yra termino i termino dažnis dokumente j, o idf_i yra i termino IDF balas. Norint pamatyti skirtumą tie patys penki sakiniai yra pritaikomi TF - IDF vektorizacijai:

	dienos	gera	geros	labai	labas	lauke	matyti	\
0	0.428411	0.000000	0.856823	0.000000	0.286912	0.000000	0.000000	
1	0.000000	0.000000	0.000000	0.000000	0.506204	0.000000	0.000000	
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.577350	0.000000	
3	0.000000	0.251365	0.000000	0.934682	0.000000	0.251365	0.000000	
4	0.000000	0.458270	0.000000	0.000000	0.380406	0.000000	0.568014	

	tave	šalta	šiandien
0	0.000000	0.000000	0.000000
1	0.000000	0.609818	0.609818
2	0.000000	0.577350	0.577350
3	0.000000	0.000000	0.000000
4	0.568014	0.000000	0.000000

8 pav. TF - IDF metodo veikimas

2.3. Tikslumo įvertinimo metrikos

Pritaikius algoritmus, svarbu patikrinti, kiek tikslus yra modelis. Tam naudojamos tikslumo metrikos. Modelio tikslumo įvertinimui egzistuoja nemažai būdų. Bus aptariami keli iš jų:

1. Painiavos matrica (angl. Confusion Matrix) dar vadinama klaidų matrica, yra specifinis lentelės išdėstymas, leidžiantis vizualizuoti algoritmo veikimą.

	Teigiama reikšmė (1)	Neigiama reikšmė (0)
Prognozuota teigiama reikšmė (1)	Prognozė teisinga (TP)	Prognozė neteisinga (FP)
Prognozuota neigiama reikšmė (0)	Prognozė neteisinga (FN)	Prognozė teisinga (TN)

4 lentelė. Painiavos matrica

2. Klasifikavimo tikslumas (angl. Classification Accuracy) - tai metrika, kuri apibendrina klasifikavimo modelio našumą, kur teisingų prognozių skaičių padalytas iš bendro prognozių skaičiaus.

3. Statistinėje dvejetainės klasifikacijos analizėje F1 balas arba F1 matas (angl. F1 Score) yra testo tikslumo matas. Jis apskaičiuojamas pagal bandymo tikslumą (angl. precision) $Precision = \frac{TP}{TP+FP}$ ir atšaukimą (angl. recall) $Recall = \frac{TP}{TP+FN}$, o

$$F1score = \frac{2 * Precision * Recall}{Precision + Recall}$$

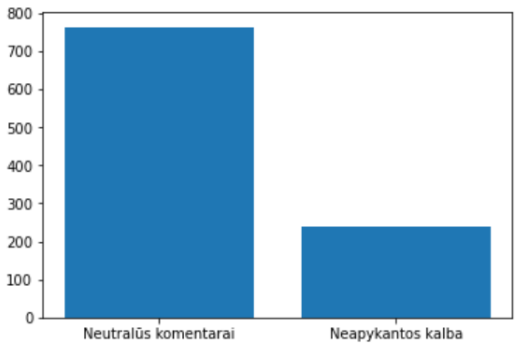
4. Pasikliautinieji intervalai (angl. confidence intervals) - tai intervalų statistika, naudojama tikslumo neapibrėžtumui įvertinti. Skaičiuojant pasikliautinuosius intervalus gaunama tiek apatinės, tiek viršutinės ribos tikimybės. Pasikliautinieji intervalai apskaičiuojami formule:

$$(m - t * \frac{s}{\sqrt{N}}, m + t * \frac{s}{\sqrt{N}}).$$

Kintamasis m yra reikšmių vidurkis, s - imties standartinis nuokrypis N - imties dydis, t - tai laisvai pasirenkamas dydis, pasirenkamas pagal tai kokio pasikliovimo intervalo yra norima

2.4. Duomenų imtis

Mašininio mokymosi metodams kaip įvestis paduodami duomenys - komentarai lietuvių kalba, o išvestį gauname kaip vieną reikšmę. Duomenų aibė sudaryta iš 1000 komentarų, surinktų atsitiktine tvarka rankiniu būdu, iš kurių 237 yra priskiriami neapykantos kalbai ir pažymėti 1, o 763, kurie nėra priskiriami neapykantos kalbai, pažymėti 0. Iš turimos imties 80% priskiriami mokymosi aibei, o likę 20% - testavimo. Norint teisingai atskirti neapykantos kalbą, pirmą reikia ją apibrėžti. Neapykantos kalba yra laikomi tokie sakiniai, kurie skatina neapykantą ar prievartą prieš asmenis ar jų grupes pagal tam tikrus požymius. Taip pat šiuo atveju neapykantos kalbai priskiriami įžeidžiančio pobūdžio teiginiai. Duomenų, sudarytų iš neutralių ir neapykantos kalbos komentarų, pasiskirstymą tarp grupių vaizduoja 9 grafikas.



9 pav. Komentarų pasiskirstymas

Komentarai, sudarantys duomenų aibę, yra paimti tiesiogiai iš šaltinio. Taigi yra susiduriama su netaisyklingos struktūros komentarais. Nagrinėjant analogišką literatūrą, duomenų apdorojimo korekcijos turėjo teigiamą arba neutralią įtaką klasifikavimo tikslumui. Šiam darbui buvo nuspręsta atlikti duomenų apdorojimą ir palyginti gaunamus tikslumus su apdorota ir neapdorota duomenų aibe. Taikomi teksto pakeitimai (angl. Text Pre-Processing) duomenų aibėje:

1. Duomenų aibėje yra daug lietuviškų žodžių, kurie komentaro esmės nesudaro ir yra ne tokie reikšmingi kategorizavimo požiūriu (angl. stop words), pavyzdžiui jungtukai, ištiktukai, prielinksniai, dalelytės, jaustukai. Siekiant pašalinti nereikšmingus žodžius iš komentarų, sukuriama nauja duomenų aibė su šiais žodžiais ir jie pašalinami.

2. Duomenų aibėje, sudarytoje iš neformalių tekstų, dalis komentatorių nenaudoja lietuviškų raidžių, ”zodis” ir ”žodis” yra tą pačią reikšmę turintys žodžiai, tačiau sukurtas modelis šiuos žodžius laiko skirtingą reikšmę turinčius. Norint ištaisyti šį netikslumą, koreguojami komentarai ir pakeičiamos visos lietuviškos raidės jų lotyniškos abėcėlės ekvivalentais.

3. Pašalinami skyrybos ženklai bei komentaro tekste esantys ”emoji”.

4. Visuose žodžiuose iš didžiųjų raidžių padaromos mažosios.

5. Pašalinami visi skaitmenys, esantys komentaruose. Skaičiai neturi reikšmės kategorizavimo požiūriu.

Komentarų apdorojimo sėkmingumas analizuojamas kiekvienam modeliui atskirai. Taigi pirmiausiai žiūrima kokią reikšmę minėti pakeitimai turėjo algoritmams.

Algoritmas	Korekcijos	Užfiksuoti skirtingi žodžiai	Tikslumas
LSTM	Taip	6551	0.7650
LSTM	Ne	8695	0.7400

5 lentelė. LSTM korekcijos rezultatai

Algoritmas	Korekcijos	Užfiksuoti skirtingi žodžiai	Tikslumas
MLP	Taip	6551	0.78
MLP	Ne	8695	0.76

6 lentelė. MLP korekcijos rezultatai

Algoritmas	Korekcijos	Užfiksuoti skirtingi žodžiai	Tikslumas
SVM	Taip	6551	0.75
SVM	Ne	8695	0.735

7 lentelė. SVM korekcijos rezultatai

Įvertinus gautus rezultatus, galima teigti, kad didžiausias tikslumo skirtumas tarp apdorotų ir neapdorotų duomenų yra LSTM modelyje, o mažiausiai įtakos apdorojimas turėjo SVM. Gauti rezultatai rodo, kad apdorojus duomenis modelio tikslumas išauga. Teksto apdorojimas norint gauti geresnius išvesties rezultatus yra būtinas. Galima atsižvelgti į tai, kad modeliams nebuvo taikomi metodai, tokie kaip kamienavimas ar lematizavimas, nagrinėti 2.3 analogiškos literatūros skyriuje. Galima spėti, kad tikslumas taip pat turėtų išaugti. Taip pat akivaizdu, kad apdorojus duomenis algoritmas rado mažiau skirtingų žodžių, tačiau pritaikius šaknies/kamieno nustatymo metodus, skirtingų žodžių rastų duomenų aibėje, turėtų dar sumažėti ir tikslumas būti dar didesnis, jei nesusidurtų su problemomis, tokiomis kaip kamienų netikslus nustatymas - pervertinimas (angl. overstemming), kai žodis sutrumpinamas per daug, arba neįvertinimas (angl. understemming), kai žodis sutrumpinamas nepakankamai.

2.4.1. Vektorizacijų palyginimas

Atlikus duomenų apdorojimą, galima gilintis į skirtingus vektorizacijos metodus. Bus naudojami anksčiau aptarti vektorizacijos būdai TF-IDF, CountVectorizer.

Algoritmas	Vektorizacija	Klasifikavimo tikslumas
SVM	CountVectorizer	0.75
SVM	TF-IDF	0.765
MLP	TF-IDF	0.775
MLP	CountVectorizer	0.78

8 lentelė. Vektorizacijų palyginimas

Rezultatai rodo, kad nėra vektorizacijos metodo, kuris abiems modeliams parodytų puikius rezultatu. SVM klasifikatoriui labiau tiko TF-IDF vektorizacijos būdas, o MLP klasifikatorius geresnius rezultatus parodė su CountVectorizer vektorizacijos būdu. Taikant modelį taip pat svarbu ir tinkamai pasirinkti vektorizavimo būdą.

2.5. MLP parametrų derinimas

Šiame skyriuje derinami (angl. tune) MLP modelio parametrai taip, kad būtų gaunamas geriausias rezultatas. Šiam klasifikatoriui galima pridėti daug parametrų. Bus bandoma rasti geriausius tikslumus:

1. $activation$ - tai parametras, leidžiantis paslėptiesiems sluoksniams pritaikyti aktyvacijos funkcijas.
2. $solver$ - tai parametras, optimizuojantis svorius. Kiekvienas optimizatorius veikia skirtingai.
3. $hidden\_layer\_sizes$ - tai parametras, nustatantis klasifikatoriaus paslėptųjų sluoksnių skaičių ir neuronus tame sluoksnyje.
4. $learning\_rate$ - tai svorių atnaujinimo mokymosi greičio grafikas.
5. $learning\_rate\_init$ - tai mokymo greitis, valdantis žingsnio dydį atnaujinant svorius.
6. max_iter - tai parametras nustatantis maksimalų pasikartojimų (angl. iterations) skaičių
7. $shuffle$ - parametras, įgalinantis maišyti duomenis kiekviename pasikartojime.
8. $alpha$ - tai L2 reguliarizacijos parametras. L2 reguliarizacija spendžia persimokymo problemą mažindama svorius, tačiau jų nepaversdama nuliais.

Pirmas žingsnis - išrinkti geriausią tikslumą duodančias aktyvacijos funkcijų ir svorio optimizavimo parametro kombinacijas.

activation	solver	R0	R1	R2	R3	R4	Mean	Max
identity	sgd	0.765	0.765	0.765	0.765	0.765	0.765	0.765
logistic	sgd	0.765	0.765	0.765	0.765	0.765	0.765	0.765
tanh	sgd	0.75	0.76	0.765	0.765	0.77	0.762	0.770
relu	sgd	0.765	0.765	0.765	0.77	0.76	0.765	0.770
identity	lbfgs	0.77	0.78	0.765	0.765	0.765	0.769	0.780
	adam	0.78	0.775	0.785	0.775	0.77	0.777	0.785
logistic	lbfgs	0.775	0.765	0.775	0.785	0.785	0.777	0.785
	adam	0.775	0.77	0.77	0.785	0.77	0.774	0.785
tanh	adam	0.785	0.775	0.785	0.775	0.77	0.778	0.785
relu	lbfgs	0.78	0.765	0.76	0.785	0.77	0.772	0.785
tanh	lbfgs	0.775	0.755	0.775	0.79	0.765	0.772	0.790
relu	adam	0.785	0.785	0.77	0.795	0.78	0.783	0.795

10 pav. MLP duomenų derinimas aktyvacijos funkcijoms ir svorio optimizavimo parametrui

Lentelėje Ri, kur $i = 0, 1, 2, 3, 4$ yra algoritmo paleidimų skaičius, reiškia, kad algoritmas buvo leistas 5 kartus ir gautas geriausias algoritmo rezultatas su aktyvacijos funkcija "relu" ir svorio optimizavimo parametru "adam".

MLP yra neuroninio tinklo modelis. Bus bandoma nustatyti tokį paslėptojo sluoksnių skaičių, kad jis modeliui duotų geriausią rezultatą. Geriausio rezultato ieškoma tarp vieno ir dviejų dirbtinio neuroninio tinklo sluoksnių, kur viename sluoksnyje gali būti daugiausia 15 neuronų ($[1] - [15]$), o dviejų sluoksnių tinkle 100 neuronų ($[1,1] - [10,10]$).

	R0	R1	R2	R3	R4	Mean	Min	Max
[6, 3]	0.760	0.760	0.760	0.760	0.730	0.754	0.730	0.760
[10, 2]	0.755	0.705	0.740	0.740	0.760	0.740	0.705	0.760
[6, 1]	0.730	0.745	0.745	0.760	0.700	0.736	0.700	0.760
[6, 2]	0.755	0.760	0.760	0.765	0.705	0.749	0.705	0.765
[1, 4]	0.715	0.765	0.760	0.725	0.740	0.741	0.715	0.765
...	...	...	...	...	...	...	...	...
[8, 9]	0.770	0.790	0.780	0.780	0.795	0.783	0.770	0.795
[9, 10]	0.785	0.780	0.795	0.785	0.790	0.787	0.780	0.795
[9, 9]	0.785	0.780	0.795	0.780	0.785	0.785	0.780	0.795
[3, 5]	0.775	0.750	0.775	0.765	0.800	0.773	0.750	0.800
[4, 7]	0.780	0.790	0.800	0.790	0.780	0.788	0.780	0.800

11 pav. MLP duomenų derinimas paslėptojo sluoksnio skaičiui

Skaičiavimai rodo, kad dviejų sluoksnių dirbtinis neuroninis tinkas, kurio pirmame sluoksnyje yra 4, o antrame 7 neuronai, turi aukščiausią tikslumo vidurkį. [3,5] tinkas taip pat pasiekia 0.8 tikslumą, tačiau jis nukrenta žemiau nei [4,7], taigi tolesniems tyrimams naudosime geriausią tinklą vidurkio prasme. Reiktų atkreipti dėmesį, kad neuronų, bei sluoksnių gali būti ir daugiau.

Bus renkamos geriausius rezultatus duodančios "*learning_rate_init*" ir "*learning_rate*" parametrų kombinacijos . Jos viena nuo kitos priklausomos.

		R0	R1	R2	R3	R4	Mean	Min	Max
LR	**rate**								
adaptive	**0.1000**	0.735	0.765	0.775	0.75	0.77	0.759	0.735	0.775
	0.0001	0.765	0.76	0.78	0.775	0.77	0.770	0.760	0.780
	0.0500	0.77	0.76	0.765	0.78	0.78	0.771	0.760	0.780
constant	**0.0001**	0.735	0.775	0.78	0.775	0.78	0.769	0.735	0.780
invscaling	**0.0050**	0.775	0.73	0.755	0.78	0.78	0.764	0.730	0.780
	0.0001	0.775	0.775	0.765	0.775	0.785	0.775	0.765	0.785
	0.0010	0.775	0.775	0.735	0.785	0.775	0.769	0.735	0.785
	0.0500	0.785	0.76	0.765	0.775	0.76	0.769	0.760	0.785
adaptive	**0.0010**	0.78	0.765	0.785	0.785	0.78	0.779	0.765	0.785
constant	**0.0050**	0.78	0.78	0.78	0.79	0.785	0.783	0.780	0.790
invscaling	**0.0100**	0.79	0.78	0.785	0.775	0.785	0.783	0.775	0.790
constant	**0.0010**	0.79	0.765	0.78	0.785	0.775	0.779	0.765	0.790
adaptive	**0.0050**	0.79	0.785	0.78	0.76	0.78	0.779	0.760	0.790
	0.0100	0.785	0.78	0.78	0.79	0.765	0.780	0.765	0.790
constant	**0.0500**	0.78	0.795	0.775	0.785	0.775	0.782	0.775	0.795
	0.1000	0.795	0.76	0.755	0.72	0.76	0.758	0.720	0.795
	0.0100	0.79	0.795	0.775	0.75	0.79	0.780	0.750	0.795
invscaling	**0.1000**	0.81	0.74	0.765	0.75	0.77	0.767	0.740	0.810

12 pav. MLP duomenų derinimas mokymosi intensyvumui ir mokymosi greičio inicijavimui

Maksimalų rezultatą pasiekę parametrai neturi didžiausio vidurkio, bet matyti, kad turi didžiausią potencialą pasiekti aukščiausius tikslumus taip pat, kaip ir vienus iš prasčiausių. Imami didžiausią vidurkį pasiekę parametrai, kurių tikslumas yra stabilesnis.

		R0	R1	R2	R3	R4	Mean	Max
Max Iterations	shuffle							
10000	False	0.775	0.785	0.78	0.78	0.78	0.780	0.785
500	True	0.77	0.79	0.77	0.73	0.785	0.769	0.790
	False	0.775	0.79	0.79	0.78	0.785	0.784	0.790
2000	True	0.765	0.79	0.715	0.785	0.78	0.767	0.790
	False	0.77	0.79	0.785	0.78	0.77	0.779	0.790
5000	True	0.775	0.785	0.79	0.79	0.79	0.786	0.790
	False	0.79	0.775	0.785	0.78	0.78	0.782	0.790
10000	True	0.73	0.79	0.775	0.785	0.79	0.774	0.790
1000	True	0.725	0.79	0.795	0.79	0.765	0.773	0.795
	False	0.78	0.795	0.78	0.79	0.785	0.786	0.795

13 pav. MLP duomenų derinimas maksimaliam iteracijų/epochų skaičiui ir maišymo galimybei

Optimizavus iteracijų ir maišymo galimybės kombinacijas bei išanalizavus lentelės duomenis paimti parametrai su 1000 iteracijų ir su neigiama "shuffle" parametro reikšme. Apžvelgus maksimalias parametrų kombinacijų tikslumų reikšmes matyti, kad visi pasiekė neblogų rezultatų.

	R0	R1	R2	R3	R4	Mean	Max
0.00050	0.78	0.78	0.78	0.78	0.76	0.776	0.780
0.00500	0.77	0.785	0.775	0.78	0.77	0.776	0.785
0.00001	0.77	0.79	0.785	0.765	0.785	0.779	0.790
0.00005	0.785	0.79	0.79	0.79	0.79	0.789	0.790
0.00010	0.79	0.79	0.78	0.785	0.775	0.784	0.790
0.00100	0.785	0.785	0.79	0.78	0.79	0.786	0.790

14 pav. MLP duomenų derinimas alpha reikšmei

Galiausiai turime reguliarizacijos parametro "$alpha$" derinimą. Aukščiausias tikslumas pasiektas su reikšme 0.001, taip pat panašų rezultatą turi ir numatytoji parametro reikšmė lygi 0.0001.

Optimizavus parametrus, analizuojamas tikslumų pokytis tarp MLP klasifikatoriaus su numatomosiomis (angl. default) reikšmėmis ir MLP klasifikatoriaus nustatytomis reikšmėmis, gautomis tyrimo metu, parodžiusias geriausius rezultatus. Rezultatas nustatomas analizuojant 20 klasifikatoriaus paleidimų. Tam naudojami 95% pasikliautinieji intervalai. Klasifikatoriui su numatytomis reikšmėmis pasikliautinasis intervalas yra $(0.782853, 0.787147)$, o klasifikatoriui su optimizuotais parametrais - $(0.782113, 0.788887)$. Parametrų derinimo skirtumo nematyti.

2.6. SVM parametrų derinimas

SVM klasifikatoriaus derinimui pasirenkami hiperparametrai. Hiperparametrai - tai parametrai, kurių reikšmės kontroliuoja mokymosi procesą ir nustato modelio parametrų reikšmes.

Priešdėlis „hyper“ rodo, kad tai yra „aukščiausio lygio“ parametrai, valdantys mokymosi procesą ir iš jo gaunamus modelio parametrus. Tokie parametrai klasifikatoriuje yra:

1. C - tai reguliarizacijos parametras. Reguliarizacijos stiprumas yra atvirkščiai proporcingas C. Turi būti griežtai teigiamas. Taikomas reguliarizacija yra l2 kvadratinė reguliarizacija.

2. Gama branduolio (angl. kernel) parametro koeficientas, naudojamas ”rbf”, ”poly” ir ”sigmoid” branduolio funkcijoms. Šis parametras apsprendžia, kokią įtaką turės duomenų taškai, esantys tam tikru atstumu nuo hiperplokštumos. Jei gamma yra didelė, bus atsižvelgiama į netoliese esančius taškus. Ir jei gamma parametras žemas, toli esantys taškai taip pat turės įtakos.

3. Kernel. Šis parametras stebėjimus sugrupuoja į tam tikrą funkcijų erdvę. Idealiu atveju po šios transformacijos stebėjimai yra lengviau (tiesiškai) atskiriami. Šioms transformacijoms yra keli standartiniai branduoliai, pvz. tiesinis branduolys, daugianario branduolys ir radialinis branduolys.

Parametrų derinimui pasirinkta sklearn bibliotekos funkcija vadinama GridSearchCV, kuri padeda nustatyti idealias hiperparametrų reikšmes konkrečiam modeliui atliekant hiperparametrų derinimą. Iš pradžių neįmanoma žinoti, kokios yra geriausios hiperparametrų reikšmės naudojamiems duomenims. Rankinis hiperparametrų koregavimas reikalauja daug laiko ir išteklių. Todėl proceso pagreitinimui naudojama GridSearchCV. Ši funkcija naudoja kryžminio patvirtinimo (angl. Cross-Validation) metodą, kad patikrintų visas galimas sąraše pateiktų reikšmių kombinacijas ir išanalizuotų kiekvieno iš jų modelį. Dėl to galima gauti kiekvieno hiperparametrų derinio tikslumą ir tinkamai pasirinkti tą, kuris geriausiai veikia. Prieš pradedant parametrų derinimą, tikrinamas modelio tikslumas su numatytomis reikšmėmis. Tikslumui gauti naudojama sklearn bibliotekos $classification\_report()$ funkcija. Gauti rezultatai matomi 15 išvestinėje. Taip pat buvo skaičiuojamas tikslumas, kuris siekė 76.6%

	precision	recall	f1-score	support
0	0.77	1.00	0.87	153
1	0.00	0.00	0.00	47
accuracy			0.77	200
macro avg	0.38	0.50	0.43	200
weighted avg	0.59	0.77	0.66	200

15 pav. SVM tikslumo metrikos su numatytomis reikšmėmis

Pritaikius parametrų atrinkimo funkciją geriausiam tikslumui atraminių vektorių klasifikatoriui geriausi parametrai yra SVC(C=10, gamma=1, kernel=”sigmoid”). SVM klasifikatoriaus tikslumas siekia 77.5%. F1 balo tikslumas su šiais parametrais yra 78%:

	precision	recall	f1-score	support
0	0.78	0.98	0.87	153
1	0.62	0.11	0.18	47
accuracy			0.78	200
macro avg	0.70	0.54	0.53	200
weighted avg	0.74	0.78	0.71	200

16 pav. SVM tikslumo metrikos su derintomis reikšmėmis

Lentelėse 15 ir 16 tikslumai įvertinti keliais skirtingais budais, tokiais kaip $Precision$,$Recall$, bei $F1 - score$, jau minėti tikslumo įvertinimo metrikų skyriuje. Analizuojant skirtumus tarp derintų bei nederintų duomenų, bendri visų 200 testavimo komentarų tikslumai yra aukštesni su optimizuotomis reikšmėmis. Be to nagrinėjant teigiamų ir neigiamų komentarų tikslumus atskirai, pastebėta, kad modelis su numatytomis reikšmėmis nustato neapykantos kalbos komentarus su 0% tikslumu, vadinasi modelis su numatytomis reikšmėmis visiškai neprognozuoja neapykantos kalbos komentarų, o modelis su derintais duomenimis šį tikslumą turi didesnį. Jis sugeba aptikti neapykantos kalbos komentarus. Šią problemą pastebėti lengviau naudojant painiavos matricą.

```
Painiavos matrica su numatytomis reikšmėmis
[[153   0]
 [ 47   0]]

Painiavos matrica su nustatytomis reikšmėmis
[[150   3]
 [ 42   5]]
```

17 pav. Painiavos matrica rezultatų vertinimui

Painiavos matricoje vėlgi akivaizdu tai, kad modelis su nustatytomis optimaliomis parametrų reikšmėmis yra geresnis. Toks modelis sugeba aptikti neapykantos sakinius daug tiksliau, kaip matyti iš matricos toks iš 47 neapykantos kalbos komentarų aptiko 5, o neutralių komentarų teisingai aptiko 150 iš 153. Taigi daroma išvada, kad SVM modeliui reikia parametrų derinimo, ypač svarbiausių parametrų - hiperparametrų. Kadangi neapykantos sakinius SVM modeliui nustatyti yra sunku, vienas iš sprendimų, galinčių padidinti tikslumą, yra duomenų imties papildymas.

2.7. LSTM parametrų derinimas

LSTM modelio derinimui pasirinkta keras bibliotekos Hyperband parametrų optimizavimo funkcija."Keras Tuner" turi keturias optimizavimo funkcijas: RandomSearch, Hyperband, BayesianOptimization ir Sklearn. Naudojama Hyperband funkcija pritaikoma LSTM hiperparametrams, tokiems kaip:

1. $input\_units$ - parametras, nurodantis sluoksnyje esančių neuronų skaičių.

2. $dropout$ - reguliarizacijos parametras, padedantis išspręsti persimokymo problemą.

3. *activation* - parametras, leidžiantis pasirinkti norimą aktyvacijos funkciją.

4. *learning_rate* - parametras, nurodantis modelio mokymosi greitį. Mašininio mokymosi ir statistikos srityse mokymosi greitis yra optimizavimo algoritmo derinimo parametras, kuris nustato žingsnio dydį kiekvienoje iteracijoje tuo pačiu siekdamas minimalios praradimo funkcijos (angl. loss function).

Nors LSTM modelis turi ir daugiau hiperparametru, tačiau šis optimizavimas trunka nemažai laiko. Taigi bus analizuojama, ar optimizavus dalį hiperparametrų yra gaunami geresni rezultatai. LSTM pradinio modelio tikslumas su atsitiktinai parinktais parametrais buvo 0.765. Atlikus Hyperband optimizacija bei gavus rezultatų suvestinę 18, naudojantis $tuner.results\_summary()$ funkcija aišku, kad parametrų optimizacija būtent LSTM modeliui turėjo daug įtakos. Pasiektas tikslumas siekia 79.5 procentus. Šį aukščiausią tikslumą, pasiekę parametrai yra

Hiperparametro pavadinimas	Hiperparametro reikšmė
$input\_units$	32
$dropout$	0.01
$activation$	sigmoid
$learning\_rate$	0.0001

9 lentelė. Tinkamiausi parametrai

Tai galime matyti iš 18 funkcijos išvestinės, kuri parodo net dešimt tiksliausiai pasirodžiusių parametrų kombinacijų, bei jų tikslumą. Svarbiausi yra su aukščiausiu tikslumu:

```
Trial summary
Hyperparameters:
input_units: 32
dropout: 0.01
activation: sigmoid
learning_rate: 0.0001
tuner/epochs: 34
tuner/initial_epoch: 0
tuner/bracket: 1
tuner/round: 0
Score: 0.7950000017881393
```

18 pav. Parametrai turintys aukščiausią tikslumą

Išanalizavus gautą situaciją, darome išvadą, kad parametrų derinimas/optimizavimas yra svarbus procesas norint pasiekti aukštesnius rezultatus atliekant mašininio mokymo algoritmų prognozes.

2.8. Modelių palyginimas

Ankstesniuose skyriuose aukščiau buvo išanalizuotas kiekvienas modelis atskirai. Norint išsirinkti geriausią modelį, bus analizuojami visi modeliai kartu. Lyginant pradinius modelius,

kuriems buvo pritaikytas duomenų apdorojimas, buvo atlikti visi žingsniai, išvardinti 2.4 duomenų imties skyriuje bei naudojama vektorizacija, turėjusi didžiausią tikslumą. MLP klasifikatorius yra tiksliausias, kadangi jis pasiekė 78% tikslumą su CountVectorizer vektorizacijos būdu.

Algoritmas	Vektorizacija	Tikslumas
SVM	TF-IDF	0.765
MLP	CountVectorizer	0.78
LSTM	Tokenizer	0.765

10 lentelė. Modelių palyginimas su neoptimizuotais parametrais

Algoritmas	Vektorizacija	Tikslumas
SVM	TF-IDF	0.775
MLP	CountVectorizer	0.782
LSTM	Tokenizer	0.795

11 lentelė. Modelių palyginimas su optimizuotais parametrais

Toliau kiekvienam algoritmui buvo atlikta parametrų optimizacija ir analizuojamas tikslumas su jais. Rezultatai pateikti 11 lentelėje. Čia išaugo ilgalaikės-trumpalaikės atminties modelio tikslumas, kuriam buvo naudojama tokenizacija, skirtingai nei kitiems dviems algoritmams, kuriems buvo taikyti CountVectorizer ir TF-IDF vektorizacijos būdai. Jis tikslumu pralenkė kitus du algoritmus ir pasiekė 79.5%, o prieš tai buvęs tiksliausias klasifikatorius MLP parametrų optimizacijos metu klasifikavimo tikslumo smarkiai nepadidino. Šiuo metu jis yra antras algoritmas pagal tikslumą. Analogiškoje literatūroje nagrinėtas SVM klasifikatorius nepralenkė LSTM ir MLP algoritmų tikslumo. Atliekant šį tyrimą modeliai, naudoję neuroninius tinklus, pasirodė geriau. Sprendžiant neapykantos kalbos aptikimo uždavinį, norint pasiekti aukštesnį tikslumą svarbus yra duomenų apdorojimas bei parametrų optimizacija. Ateičiai darbams rekomenduotina naudoti neuroninių tinklų modelius. Su jais buvo pasiekta aukščiausias tikslumas.

Rezultatai ir išvados

Tyrimo procesas

1. Nagrinėta analogiška literatūra bei atrinkti metodai tolimesniems eksperimentams.
2. Sudaryta duomenų imtis ir sužymėti komentarai.
3. Realizuoti algoritmai neapykantos kalbos aptikimui, bei optimizuoti jų parametrai.
4. Nustatyti mašininio mokymosi algoritmų tikslumai naudojant įvairias tikslumo įvertinimo metrikas, identifikuotas tiksliausiai pasirodęs LSTM algoritmas.
5. Palyginti CountVectorizer ir TF-IDF vektorizacijos būdai.
6. Pritaikyti duomenų apdorojimo metodai, tokie kaip nereikšmingų žodžių panaikinimas, lietuviškų raidžių pakeitimas į jų lotyniškos abėcėlės ekvivalentus, visų raidžių pavertimas mažosiomis, pašalinami skaitmenys, bei tekste buvę ”emoji”.

Išanalizavus rezultatus gautos išvados:

1. Išanalizavus vektorizacijos technikas kiekvienam modeliui, matyti, kad SVM labiau tiko TF-IDF vektorizacijos būdas, o MPL klasifikatoriui labiau tinkamas buvo CountVectorizer.
2. Tekstas, kuriame duomenys yra apdoroti, leidžia pasiekti didesnį klasifikavimo tikslumą nei neapdorotas tekstas.
3. Neapykantos kalbos nustatymo uždaviniui spręsti neuroniniai tinklai yra efektyvesni nei SVM klasifikatorius.
4. Šiame darbe iš visų tirtų mašininio mokymosi algoritmų tinkamiausias ilgalaikės-trumpalaikės atminties neuroninių tinklų modelis.

Apibendrinus darbo rezultatus ir išvadas, rekomenduojama neapykantos kalbos nustatymo uždaviniui spręsti naudoti LSTM modelį. Rekomenduojama pritaikyti parametrų optimizaciją.

Literatūra

[Bac11] Jurgita Bacevičiūtė. *Neapykantos grupės socialiniame tinkle „Facebook “:„Stop pedofilijai Lietuvoje “„Aš prieš homoseksualų paradą Vilniuje 2010 gegužę “ir „Nekenčiu Seimo “grupių atvejų analizė*. Magistrinis darbas, 2011.

[Bas] Algirdas Bastys. Atraminių vektorių klasifikatorius. `https://kedras.mif.vu.lt/bastys/academic/ATE/biometrika/SVM.pdf`. Peržiūrėta 2022m. gegužę.

[Bis94] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[Bla21] Matas Blagnys. *Smurtinio pobūdžio žinučių identifikavimas vykdant interneto portalų anoniminių nuomonių sentimentų analizę*. Disertacija, Vilniaus universitetas, 2021.

[CMC+20] Michele Corazza, Stefano Menini, Elena Cabrio, Sara Tonelli ir Serena Villata. A multilingual evaluation for online hate speech detection. *ACM Transactions on Internet Technology (TOIT)*, 20(2):1–22, 2020.

[Dzi19] Robert Dzisevič. *Trumpo teksto klasifikacija naudojant neuroninius tinklus*. Disertacija, Vilniaus Gedimino technikos universitetas, 2019.

[Jaf22] Roy Jafari. Classification using MLP - sklearn module. `https://www.youtube.com/watch?v=oPxbrgYuBGY&ab_channel=RoyJafari`, 2022. Peržiūrėta 2022m. gegužę.

[JM15] Michael I Jordan ir Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[Joh] Christopher Williams John McGonagle George Shaikouski. Backpropagation. `https://brilliant.org/wiki/backpropagation/`. Peržiūrėta 2022m. gegužę.

[KKK+17] Diksha Khurana, Aditya Koli, Kiran Khatter ir Sukhdev Singh. Natural language processing: State of the art, current trends and challenges. *arXiv preprint arXiv:1708.05148*, 2017.

[KR19] Karolis Kiaunė ir Simona Ramanauskaitė. Didelį kategorijų kiekį turinčių draudimo bendrovės klientų užklausų, gautų elektroniniais laiškais, lietuviško teksto klasifikavimas. *Jaunųjų mokslininkų darbai*, 49(2):52–59, 2019.

[KSN+20] Divya Khyani, BS Siddhartha, NM Niveditha ir BM Divya. An Interpretation of Lemmatization and Stemming in Natural Language Processing. *Shanghai Ligong Daxue Xuebao/Journal of University of Shanghai for Science and Technology*, 22:350–357, 2020.

[MYY+19] Sean MacAvaney, Hao-Ren Yao, Eugene Yang, Katina Russell, Nazli Goharian ir Ophir Frieder. Hate speech detection: Challenges and solutions. *PloS one*, 14(8):e0221152, 2019.

[Mor22] Anton Morgunov. Keras Tuner: Lessons Learned From Tuning Hyperparameters of a Real-Life Deep Learning Model. `https://neptune.ai/blog/keras-tuner-tuning-hyperparameters-deep-learning-model`, 2022. Peržiūrėta 2022m. gegužę.

[MZ17] Shervin Malmasi ir Marcos Zampieri. Detecting hate speech in social media. *arXiv preprint arXiv:1712.06427*, 2017.

[Pet10] Mažvydas Petkevičius. *Automatinis lietuvių kalbos tekstų temos nustatymas*. Magistrinis darbas, 2010.

[Sax21] Shipra Saxena. Introduction to Long Short Term Memory (LSTM). `https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/?`, 2021. Peržiūrėta 2022m. gegužę.

[Šal15] Justas Šalkevičius. *Maitinimo įstaigų reitingo prognozavimo statistinių modelių tyrimas*. Disertacija, Kauno technologijos universitetas, 2015.

Priedas Nr. 1
Modelių parametrų optimizavimas

```
#activation ir solver parametrų optimizacija
num_repetition = 5
activation_options = ['identity', 'logistic', 'tanh', 'relu']
solver_options = ['lbfgs','sgd','adam']
my_index = pd.MultiIndex.from_product([activation_options,solver_options],
                                      names=('activation', 'solver'))
tune_df = pd.DataFrame(index = my_index,
                       columns=['R{}'.format(i) for i in range(num_repetition)])
n = len(y_test)
for activation_o in activation_options:
    for solver_o in solver_options:
        for rep in tune_df.columns:
            car_mlp = MLPClassifier(hidden_layer_sizes=(5), max_iter=2000,
                                    activation=activation_o,solver=solver_o)

            car_mlp.fit(x_train_vec, y_train)
            y_tune_predict = car_mlp.predict(x_test_vec)
            print(activation_o,solver_o,accuracy_score(y_test, y_tune_predict))
            RSME = ((accuracy_score(y_test, y_tune_predict)))
            tune_df.at[(activation_o,solver_o),rep] = RSME
#rezultatas
tune_df['Mean'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].mean(axis=1)
tune_df['Max'] = tune_df[
     ['R{}'.format(i) for i in range(num_repetition)]].max(axis=1)
tune_df.sort_values('Max')
```

19 pav. MLP modelio parametrų optimizavimas [Jaf22]

```
#susikuriamas hidden_layer_sizes parametro reikšmių sąrašas
PossibleNetStrct = []
PossibleNetString = []
for i in range(1,16):
    netStruct = [i]
    PossibleNetStrct.append(netStruct)
    PossibleNetString.append(str(netStruct))
    print('Network Structure:', netStruct)
for i in range(1,11):
    for j in range(1,11):
        netStruct = [i,j]
        PossibleNetStrct.append(netStruct)
        PossibleNetString.append(str(netStruct))
        print('Network Structure:', netStruct)
#hidden_layer_sizes parametro optimizacija
tune_df = pd.DataFrame(np.nan, index =PossibleNetString,
                       columns = ['R{}'.format(i) for i in range(num_repetition)])

n = len(y_test)
for i,netStr in enumerate(PossibleNetStrct):
    RowName = PossibleNetString[i]
    for rep in tune_df.columns:
        car_mlp = MLPClassifier(hidden_layer_sizes=netStr, activation = 'relu',
                        solver='adam', max_iter=2000)
        car_mlp.fit(x_train_vec, y_train)
        y_tune_predict = car_mlp.predict(x_test_vec)
        RSME = ((accuracy_score(y_test, y_tune_predict)))
        tune_df.at[RowName, rep] = RSME
    print(netStr)
#rezultatas
tune_df['Mean'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].mean(axis=1)
tune_df['Max'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].max(axis=1)
tune_df.sort_values('Max')
```

20 pav. MLP modelio parametrų optimizavimas [Jaf22]

```
#Learning_rate ir learning_rate_init parametrų optimizacija
num_repetition = 5
LR_options = ['constant','invscaling','adaptive']
LRI_options = [0.0001,0.001,0.005,0.01,0.05,0.1]
my_index = pd.MultiIndex.from_product([LR_options,LRI_options],
                                      names=('LR', 'rate'))
tune_df = pd.DataFrame(index = my_index,
                       columns=['R{}'.format(i) for i in range(num_repetition)])
for LR_o in LR_options:
    for LRI_o in LRI_options:
        for rep in tune_df.columns:
            car_mlp = MLPClassifier(hidden_layer_sizes=(4,7), max_iter=2000,
                                    activation='relu',solver='adam',
                                    learning_rate=LR_o, learning_rate_init= LRI_o)
            car_mlp.fit(x_train_vec, y_train)
            y_tune_predict = car_mlp.predict(x_test_vec)
            RSME = ((accuracy_score(y_test, y_tune_predict)))

            tune_df.at[(LR_o,LRI_o),rep] = RSME
        print((LR_o,LRI_o))
#rezultatas
tune_df['Mean'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].mean(axis=1)
tune_df['Max'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].max(axis=1)
tune_df.sort_values('Max')
```

```
#max_iter ir shuffle parametrų optimizacija
max_iterations_options = [500,1000,2000,5000,10000]
shuffle_options = [True,False]
my_index = pd.MultiIndex.from_product([max_iterations_options,shuffle_options],
                                      names=('Max Iterations', 'shuffle'))
tune_df = pd.DataFrame(index = my_index,
                       columns=['R{}'.format(i) for i in range(num_repetition)])
for max_iterations_o in max_iterations_options:
    for shuffle_o in shuffle_options:
        for rep in tune_df.columns:
            car_mlp = MLPClassifier(hidden_layer_sizes=(4,7), max_iter=max_iterations_o,
                                    activation='relu',solver='adam',
                                    learning_rate='constant', learning_rate_init= 0.0050,
                                    shuffle = shuffle_o)
            car_mlp.fit(x_train_vec, y_train)
            y_tune_predict = car_mlp.predict(x_test_vec)
            RSME = ((accuracy_score(y_test, y_tune_predict)))

            tune_df.at[(max_iterations_o,shuffle_o),rep] = RSME
        print((max_iterations_o,shuffle_o))
#rezultatas
tune_df['Mean'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].mean(axis=1)
tune_df['Max'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].max(axis=1)
tune_df.sort_values('Max')
```

21 pav. MLP modelio parametrų optimizavimas [Jaf22]

```
#alpha parametro optimizacija
alpha_options =[0.00001,0.00005,0.0001,0.0005,0.001,0.005]
tune_df = pd.DataFrame(index = alpha_options,
                       columns=['R{}'.format(i) for i in range(num_repetition)])
for alpha_o in alpha_options:
    for rep in tune_df.columns:
            car_mlp = MLPClassifier(hidden_layer_sizes=(4,7), max_iter=1000,
                                    activation='relu',solver='adam',
                                    learning_rate='constant', learning_rate_init= 0.0050,
                                     shuffle = False, alpha=alpha_o)
            car_mlp.fit(x_train_vec, y_train)
            y_tune_predict = car_mlp.predict(x_test_vec)
            RSME = ((accuracy_score(y_test, y_tune_predict)))
            tune_df.at[alpha_o,rep] = RSME
    print(alpha_o)
#rezultatas
tune_df['Mean'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].mean(axis=1)
tune_df['Max'] = tune_df[
    ['R{}'.format(i) for i in range(num_repetition)]].max(axis=1)
tune_df.sort_values('Max')
```

22 pav. MLP modelio parametrų optimizavimas [Jaf22]

```
#spausdinamas modelio rezultatas su neoptimizuotais parametrais
model = SVC()
model.fit(x_train_vec1, y_train)
predictions = model.predict((x_test_vec1))
print(classification_report(y_test, predictions))
#renkami geriausi parametrai
param_grid = {'kernel': ['rbf', 'poly', 'sigmoid'],'C': [10 ,4, 5, 1],
              'gamma': [0.2,1,0.01,0.3]}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(x_train_vec1, y_train)
print(grid.best_estimator_)
#rezultatai su optomizuotais duomenimis
grid_predictions = grid.predict(x_test_vec1)
print(classification_report(y_test,grid_predictions))
```

23 pav. SVM modelio parametrų optimizavimas

```
#sukuriama funkcija modelio kūrimui
def model_builder(hp):
    #nustatomi hiperparametrai modelio optimizavimui
    model=Sequential()
    model.add(Embedding(num_words,32,input_length=max_length))
    model.add(LSTM(hp.Int('input_units',
        min_value=32,
        max_value=256,
        step=32), dropout=hp.Choice(name = 'dropout', values= [0.01, 0.001, 0.1])))
    model.add(Dense(1,activation=hp.Choice(name = 'activation',
                                           values = ['sigmoid', 'tanh', 'relu'])))
    model.compile(loss="binary_crossentropy",
                  optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',
                                                            values=[1e-2, 1e-3, 1e-4])),metrics=["accuracy"])
    return model

#hyperband optimizacijos iniciavimas
tuner = kt.Hyperband(
    model_builder,
    objective='val_accuracy',
    executions_per_trial=4,
    directory="LSTM_opt",
    overwrite = True
)
#optimizacijos vykdymas
tuner.search(x=train_padded,
             y=y_train,
             verbose=2,
             epochs=4,
             batch_size=64,
             validation_data=(test_padded, y_test))
#optimizacijos išvestinė
tuner.results_summary()
```

24 pav. LSTM modelio parametrų optimizavimas [Mor22]