KAUNO TECHNOLOGIJOS UNIVERSITETAS
MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

Andrius Dapševičius

KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO MODELIS

Baigiamasis magistro projektas

Vadovai:
doc. dr. Vytautas Janilionis
prof. dr. Rimantas Gatautis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO MODELIS

Magistro projektas
Didžiųjų verslo duomenų analitika (kodas 621G12002)

Vadovai
Doc. dr. Vytautas Janilionis

Prof. dr. Rimantas Gatautis

Recenzentai
Doc. dr. Tomas Ruzgas

Doc. dr. Evaldas Vaičiukynas

Projektą atliko
Andrius Dapševičius

KAUNAS, 2017

Baigiamųjų projektų rengimo,
gynimo ir saugojimo tvarkos aprašo
4 priedas

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Matematikos ir gamtos mokslų fakultetas
(Fakultetas)

Andrius Dapševičius
(Studento vardas, pavardė)

Didžiųjų verslo duomenų analitika (621G12002)
(Studijų programos pavadinimas, kodas)

„Baigiamojo projekto pavadinimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužė 26 d.
Kaunas

Patvirtinu, kad mano, **Andriaus Dapševičiaus**, baigiamasis projektas tema „Kenksmingų interneto tinklalapių identifikavimo metodas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

TURINYS

PAVEIKSLŲ SĄRAŠAS

LENTELIŲ SĄRAŠAS

ALGORITMŲ SĄRAŠAS

Dapševičius, Andrius. Kenksmingų interneto tinklalapių identifikavimo modelis: Magistro baigiamasis projektas baigiamasis projektas / vadovai doc. dr. Vytautas Janilionis, prof. dr. Rimantas Gatautis; Kauno technologijos universitetas, Matematikos ir gamtos mokslų fakultetas.

Mokslo kryptis ir sritis: fiziniai mokslai, matematika

Reikšminiai žodžiai: kenksmingi tinklalapiai, klasifikavimas, PageRank, atraminių vektorių klasifikatorius, C4.5.

Kaunas, 2017. 60 p.

SANTRAUKA

Interneto tinklalapiai gali būti klasifikuojami į dvi kategorijas. Vieni siekia pajamas gauti tiekiant naudingas paslaugas naudotojams. Antra grupė yra naudojama siekiant pasipelnyti kenksmingais naudotojui metodui. Šie tinklalapiai yra kenksmingi tinklalapiai.

Šiame darbe sukurta metodika ir programinis įrankis skirtas kenksmingų interneto tinklalapių identifikavimui. Naudojamos tinklalapių savybės yra jų adresai bei interneto svetainių grafo informacija, taip pasinaudojant kenksmingų tinklalapių tendencija turėti nuorodas į kitus kenksmingus tinklalapius.

Mokyti trys skirtingi mašininio mokymo modeliai, sprendimų medis, atraminių vektorių mašinų modelis, logistinės regresijos modelis. Geriausias suformuotas modelis pasiekė 97% tikslumą. Modelis panaudotas kuriant programinį įrankį kuris yra pritaikytas naudoti kuriant kitas programas ar jų sistemas.

Atkreipiamas dėmesys į mašininiu mokymu kuriamų kenksmingų tinklalapių identifikavimo modelius ir problemas, kurios pastebėtos kuriant modelį.

Dapševičius, Andrius. Malicious web page identification model: Master‘s thesis in business big data analytics / supervisors assoc. prof. Vytautas Janilionis, prof Rimantas Gatautis. The Faculty of Matematics and Natural Sciences, Kaunas University of Technology.

Research area and field: physical sciences, mathematics.

Key words: malicious web pages, classification, PageRank, support vector classifier, C4.5

Kaunas, 2017. 60p.

SUMMARY

Web pages can be classified into two categories. One category is for websites that profit from some service provided for the customer, and the other category is for web pages that profit by harming the user. Web pages that fall into this category are malicious web pages.

This paper describes a tool and methodology for malicious web page identification. Identification is based upon addresses of the web pages and their position in the web graph, which allows to consider a tendency of malicious websites to contain links to other malicious websites.

Three machine learning models were being trained – a decision tree, support vector machine, logistic regression model. The best model reached 97% accuracy. Model was used to create a software tool that is designed to be used in other applications and application systems.

Also, this paper raises some concerns regarding use machine learning models for malicious website identification because of problems that were noticed during creation of the model.

1. ĮŽANGA

Šiuolaikinės technologijos lemia spartų interneto tinklapių atsiradimą. Tinklalapiai kuriami siekiant gauti kokią nors naudą. Tačiau dalis jų yra kuriama tikintis pasipelnyti naudotojams kenksmingais būdais, tai yra infekuojant jų kompiuterius, vagiant asmens duomenis. Tokie tinklalapiai yra kenksmingi ir juos atpažinti yra svarbu, taip apsaugant naudotojus.

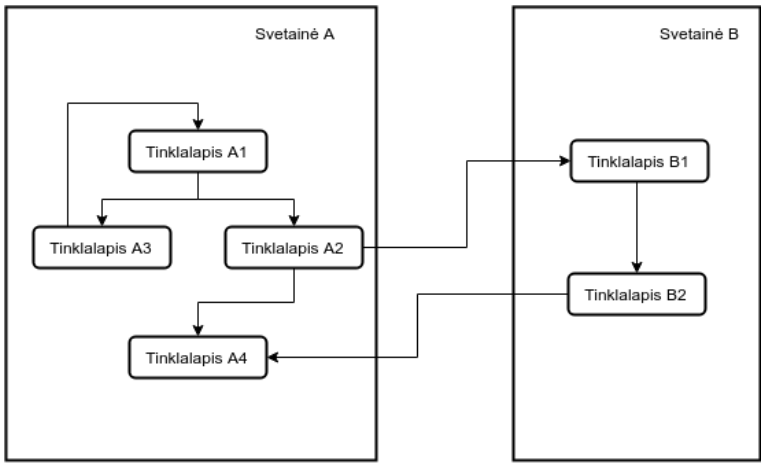
Darbo tikslas - pasiūlyti ir įgyvendinti metodiką tinklalapių klasifikavimui į kenksmingus bei nekenksmingus pasinaudojant jų adresu bei svetainių grafo suteikiama informacija. Tai leidžia tiksliau klasifikuoti kenksmingus tinklalapius remiantis prielaida, kad kenksmingi tinklalapiai svetainių grafe yra susiję, tai yra kenksmingi tinklalapiai turi nuorodas į kenksmingus tinklalapius, nekenksmingi tinklalapiai turi daugiau nuorodų į nekenksmingus tinklalapius.

Pirmoje darbo dalyje yra pateikiama klasifikavimo metodų bei kenksmingų tinklalapių tyrinėjimo literatūros apžvalga. Antroje dalyje pateikiama metodika kenksmingų ir nekenksmingų tinklalapių klasifikavimui, suformuojant mašininio mokymo modelius. Trečiojoje dalyje yra aprašomas ir apmokomas modelis, pateikiama programinė įranga šio modelio praktiniam panaudojimui, kritiškai įvertinamas praktinis įrankio panaudojimas.

2. LITERATŪROS APŽVALGA

2.1. KENKSMINGI TINKLALAPIAI

Internetas gali būti nagrinėjamas įvairiomis perspektyvomis. Jis gali atlikti informacijos šaltinio paskirtį, leisti praleisti laisvalaikį socialiniuose tinkluose, apsipirkinėti įvairiose parduotuvėse neišeinant iš namų.



2.1 pav. Interneto svetainių ir tinklapių struktūra

Tačiau žvelgiant iš techninės pusės, internetas yra daugybė jame pasiekiamų svetainių. Svetaines sudaro vienas ar daugiau tinklalapių 2.1 pav.. Šie tinklalapiai talpina tam tikrą turinį bei nuorodas į kitus tinklalapius, taip suformuojant tinklą – orientuotą grafą kurį sudaro tinklalapiai bei naudotojams naviguoti skirtos nuorodos vedančios į kitus tinklalapius.

Svetainėms kurti ir talpinti yra reikalingi žmogiškieji resursai turinio sukūrimui, programinės įrangos paruošimui. Taip pat ir infrastruktūra jų talpinimui. Tai sudaro jų talpinimo kaštus, kuriuos padengia svetainė turi tiesiogiai ar netiesiogiai padengti, nesvarbu ar tai yra e-parduotuvė ar įmonę reprezentuojantis tinklalapis skirtas pritraukti klientus. Šie pajamų generavimo metodai yra paremti naudos suteikimu. Naudojami metodai gali būti tiesioginiai - apmokestinami parduodami produktai ar prieiga prie turinio, ar reklama paremtas monetizacijos modelis – tinklalapiuose talpinamos nuorodos į kitas svetaines, siūlomi produktai. Šie metodai yra naudingi vartotojams, jie gauna informaciją, sužino apie naujus produktus, savo noru perka prekes.

Dalis svetainių yra paremtos naudotojams žalingais monetizacijos modeliais [1], kai pasinaudojant naudotojo naivumu, techniniu neišprusimu ar jo apgaule išviliojami asmeniniai duomenys, įdiegiama kenkėjiška programinė įranga į jo kompiuterį. Straipsnyje [1] analizuojami įvairūs metodai kuriais pasinaudojant apgaunami naudotojai, jie patenka į kenksmingus tinklalapius. Atakose naudotojas yra išskiriamas kaip silpniausia sistemos grandis [1]. Tai išskiriama ne tik dėl galimy-

bės apgauti pateikiant klaidinančią informaciją, bet ir dėl naudotojų turimų teisių, naudotojas gali prieiti prie visų sistemos resursų ir įdiegti bet kokią programinę įrangą, net jei ji yra kenksminga. Be naudotojo pagalbos tą padaryti itin sunku. Sistemos naudotoju stengiamasi pasinaudoti semantinėse atakose.

Straipsnyje išskiriamos semantinės atakos yra [1]:

a) **Phishing** – Sukčiavimas apsimetant, naudojami el. laiškai, žinutės, kuriomis apsimetama patikimu šaltiniu ir siekiama apgaule išvilioti naudotojo asmeninius duomenis, kuriais pasinaudojus bus galima pasipelnyti.
b) **File Masquerading** – Failo maskavimas, platinami failai kurie yra maskuojami kaip naudotojui įprasti ir saugiai atrodantys failai. Failai galimai maskuojami kaip dokumentų failai, sisteminiai failai, kuriuos naudotojas įpratęs matyti ir pasitikėti.
c) **Application Masquerading** – Apsimetimas aplikacija, apsimetama vartotojui naudinga aplikacija.
d) **Web Pop-Up** – Iššokantieji langai, apsimetantys klaidų pranešimais, klausimynais, ar informacija kuri gali sudominti naudotoją.
e) **Malvertisement** – Kenksmingos reklamos.
f) **Social Networking** – socialiniais tinklais plintančios atakos, tai gali būti pasidalintos nuorodos, failai.
g) **Removable Media** – Fiziniais įrenginiais paremtos atakos.
h) **Wireless** – belaidžiais įrenginiais paremtos atakos.

Didelė dalis šių atakų (a), d), e), f)) naudojasi tarpinėmis svetainėmis. Tai yra ramstinės svetainės per kurias yra nukreipiamas duomenų srautas ir paskirstomas į kitas kenksmingas svetaines, kurios yra naudojamos galutinei naudotojų monetizacijai [2]. Šios atakos pasižymi tuo, kad naudotojai jas gali atskirti per URL adresą, kuris nuves į kenkėjišką svetainę [1].

Semantines atakas atpažinti siūlomi mašininio mokymosi modeliai [1]. Jie gali būti pritaikyti atpažinti atakas pagal įvairias savybes: el. laiškų turinį, svetainių URL adresus, tinklalapių turinį [1]. Straipsnyje [3] siūlomas „WebCop“ metodas skirtas atpažinti žalingus internetinius tinklalapius ir kenksmingų tinklapių grupes, kaimynystes.

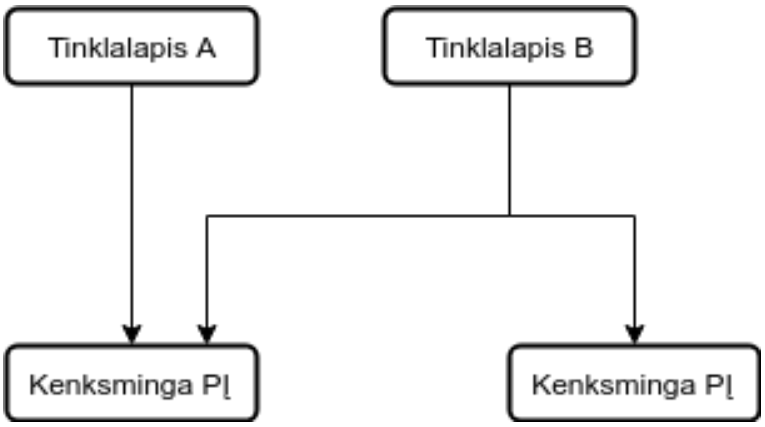
Šiuo straipnsiu siekiama [3]:

- pateikti sistemą gebančią atpažinti kenksmingas bei saugias interneto kaimynystes kurias sudaro svetainių nuorodos,
- pateikti tikslingą, iš apačios aukštyn, veikiantį kenkėjiškos programinės įrangos aptikimo metodą,
- papildomą būdą atpažinti pirmojo tipo klaidas kenkėjiškos programinės įrangos identifikavimo modulyje,
- pateikti naują būda atrasti kenkėjišką programinę įrangą.

Straipsnyje aprašoma žalingos programinės įrangos plitimo problema. Vienas iš būdų kaip į kompiuterius patenka tokia programinė įranga yra kenksmingų vykdomųjų failų parsisiuntimas iš

interneto svetainių. Kenkimo programų plitimą siekiama apriboti naudojant prevencines priemones. Siekiama identifikuoti svetaines platinančias kenksmingus failus ir neleisti naudotojams jas pasiekti, ar juos kitaip perspėti apie potencialiai kenksmingą svetainę.

Straipsnyje identifikuotą kenkėjiškos programinės įrangos plitimo problemos sprendimui siūloma WebCop sistema kuri atpažįsta kenkėjiškas ir joms gretimas svetaines. Pavyzdžiui jei turima struktūra atitinkanti 2.2 pav. tinklalapius. Šiuo atveju tinklalapiai A ir B talpina nuorodas tik į kenksmingą programinę įrangą, dėl to galima laikyti, kad vartotojas negaus jokios naudos iš jų, tik bus skatinamas įsidiegti kenksmingą programinę įrangą. Taip pat, keliaujant interneto grafu nuo apačios, pradedant kenksminga programine įrangą galima aptikit ją platinančias svetaines, jas taip pat pažymėti kaip kenksmingas.



2.2 pav. Kenksmingos programinės įrangos platinimo svetainės struktūros pavyzdys

WebCop sistemą sudaro WebCop modulis bei dvi duomenų bazės. Viena duomenų bazėje saugomi interneto svetainių tarpusavio ryšių informacija, tai yra interneto svetainių grafas. Antra duomenų bazė laiko informaciją apie identifikuotas kenkėjiškas programas bei jų šaltinius internete. Ši sistema veikia keliais etapais:

1. nuorodų į platinimo svetaines aptikimas;
2. žalingų kaimynysčių aptikimas;
3. naujos, potencialiai žalingos, programinės įrangos aptikimas.

Pirmas žingsnis, platinimo svetainių atpažinimas, vykdomas jungiant identifikuotų kenkėjų duomenų bazės duomenis su interneto grafo viršūnėmis. Sekančiame žingsnyje išrenkamos svetainės gretimos platinimo svetainėms. Tai yra vienos nuorodos atstumu nuo kenksmingų svetainių esančios svetainės. Šie interneto tinklalapiai yra laikomi žalingos programinės įrangos katalogais. Naudotojų prieiga prie šių tinklalapių turi būti apribota, siekiant juos apsaugoti. Svetainės esančios dviejų nuorodų atstumu ir talpinančios nežinomas taikomąsias programas identifikuojamos kaip potencialiai kenksmingos. Laikoma, kad tai nauja, neidentifikuota kenkėjiška programinė įranga.

WebCop sistema naudoja du duomenų rinkinius – kenksmingos programinės įrangos registrą bei interneto grafą. Interneto grafo duomenų rinkinys formuotas naudojant interneto paieškos variklio duomenis. Kenkėjiškų programų registras kuriamas naudojantis Windows operacinės sistemos naudotojų teikiamais duomenimis.

2.1 lentelė

WebCop straipsnyje pateikiami matavimų rezultatai

Matavimas	Vertė
Aptiktos nežalingų programų platinimo svetainės	1460
Aptiktos žalingų programų platinimo svetainės	10853
Aptikti nežalingų programų katalogai	2850883
Aptikti žalingų programų katalogai	391893

Lentelėje 2.1 pateikiami straipsnyje atliktų matavimų rezultatai.

Šis metodas yra unikalus keliais aspektais – tinklalapių grafas analizuojamas iš apačios aukštyn (angl. *bottom-up*) vietoje skenavimo iš viršaus žemyn (angl. *top-down*) ir naudojamas egzistuojantis kenkėjiškos programinės įrangos registras vietoje bandymo ją identifikuoti skenavimo metu. Tai leidžia pasinaudoti interneto grafo struktūra, ne tik atskirų svetainių bei tinklalapių savybėmis.

Interneto grafo savybėmis naudojamasi ir straipsnyje [2]. Čia aprašomas metodas kenksmingos programinės įrangos aptikimui remiantis interneto svetainių grafo topologine struktūra. Interneto svetainės yra ranguojamos naudojant PageRnka algoritmą. Taip aptinkamos svetainės, kurios yra svarbios kenksmingų svetainių struktūroje.

Kertinės svetainės straipsnyje [2] yra apibrėžiami kaip dedikuoti serveriai, kurie yra būtini kenksmingų internetinių svetainių veikimui. Šie serveriai talpina svetaines, kurios pasiekiamos tik iš kenkėjiškų svetainių ir yra skirti valdyti įvairias kenksmingas veiklas nukreipiant naudotojų srautus. Šiuos tinklo ramsčius aptikti ir sustabdyti yra svarbu dėl vykdomo veiklos masto – jie gyvuoja daug ilgiau nei įprastos kenksmingos svetainės, apdorojami dideli duomenų kiekiai, dėl to jų sustabdymas labiau sutrikdo žalingą veiklą. Sprendžiama šių ramsčių aptikimo problema.

Svetainės, kurias stengiamasi aptikti pasižymi tam tikromis topologinėmis savybėmis grafe. Interneto svetaines galima skaidyti į dvi klases - kenksmingos ir nekenksmingos svetainės. Iš kenksmingų svetainių yra išskiriamas papildomas kenksmingų svetainių tipas. Tai yra ramstiniuose serveriuose talpinamos svetainės, kurių tikslas yra nukreipti naudotojus į kitas kenksmingas svetaines, kur bus bandoma monetizuoti naudotojus. Tai vaizduojama 2.3 pav.. Čia tinklapiai A, B, C yra kenksmingi tinklapiai kurie yra skirti pritraukti naudotojų dėmesį. Jie gali būti įterpiami į nekenksmingas svetaines įsilaužus ar naudojant apgaulingas reklamas [1]. Naudotojai iš jų yra nukreipiami į tarpinį, srauto paskirstymo tinklapį. Šis tinklapis vėl naudotojus automatiškai nukreipia į tolimesnius tinklalapius, kur bus bandoma iš jų pasipelnyti, ar verčiant ką nors nusipirkti, bandant

išvilioti asmeninius duomenis ar kita. Ši ramstinių serverių paskirtis, srauto paskirstymas lemia tai, kad ramstiniai serveriai neturi jokių tiesioginių ryšių iš nekenksmingų svetainių. Visi ryšiai ateina tik iš kenksmingų svetainių. [2]. Tokia topologinė struktūra interneto grafe leidžia juos ranguoti naudojant PageRank algoritmą ir tikėtis aukšto rango.

2.3 pav. Vartotojų srauto paskirstymo schema

PageRank algoritmas ranguoja interneto tinklalapius pagal kiekvieno tinklalapio svarbą. Svarba vertinama pagal nuorodų į tinklalapį kiekį. Laikoma, kad svarbesni tinklalapiai yra dažniau minimi kitose svetainėse. Taip pat svarbių svetainių paminėjimai yra verti daugiau nei nesvarbių svetainių. Taip imituojamas socialinio tinklo prestižo statusas [4].

Vertinant svetainės PageRank įvertį internetas laikomas orientuotu grafu $G = (V, E)$ kur V yra grafo viršūnės, tai yra tinklalapiai, o E yra orientuotos briaunos, tai yra nuorodos tarp tinklalapių [4]. Supaprastintas PageRank įvertis yra apibrėžiamas kaip [5]:

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{2.1}$$

kur $R(u)$ yra PageRank įvertis svetainei u, B_u yra aibė svetainių kurios turi nuorodas į svetainę u, N_v yra svetainės v nuorodų į kitus tinklalapius skaičius, o c normalizavimo faktorius. Šis įvertis yra teisingas, tačiau gali būti lengvai iškreipiamas grafe esančių ciklų. Jie lemtų greitą kilimą range. Tas sprendžiama įtraukiant slopinimo faktorių d [5]. Gaunamas galutinis PageRank įvertis yra:

$$R(u) = (1 - d) + d \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{2.2}$$

slopinimo faktorius d gali būti nustatomas bet kokiam skaičiui tarp 0 ir 1. $d = 0.85$ naudojama straipsnyje [5]

PageRank įvertis priklauso nuo kitų svetainių PageRank įverčio, dėl to naudojamas iteracinis metodas jo skaičiavimui. Rezultatai taip pat priklauso nuo pradinio, nulinės iteracijos, PageRank vertės nustatymo. Ji gali būti parenkama pagal ekspertinę nuomonę, jei siekiama gauti įprastinį rangavimą, kuris tinkamas naudoti interneto paieškos varikliuose. Iteracinis algoritmas konverguoja greitai, 322 milijonų nuorodų duomenų rinkinys pasiekia gerą rezultatą per 52 iteracijas [5].

PageRank algoritmo rezultatas labai priklauso nuo nulinės iteracijos PageRank įverčių parinkimo [2]. Tuo pasinaudojant formuojami keli PageRank įverčiai. Vienas yra įprastinis įvertis,

kai įvertis nustatomas į 1 nekenksmingoms svetainėms, į 0 visoms kitoms svetainėms [2]. Taip suranguojamos nekenksmingos svetainės. Šiuo atveju ramstinės kenksmingos svetainės turės itin žemą rangą, nes neegzistuos nuorodos iš nekenksmingų svetainių. Formuojant antrą PageRank įvertį vieneto reikšmės priskiriamos kenksmingoms svetainėms. Taip ramstinės svetainės gaus itin aukštą rangą [2]. Taip gaunamas rangas interneto tinkle ir kenksmingų svetainių tinkle. Skirtumas tarp rangų leidžia atpažinti ramstines svetaines, taip netiesiogiai pasinaudojant jų izoliuotumu nuo nekenksmingų svetainių interneto grafe.

Naudotas duomenų rinkinys suformuotas iš kelių šaltinių. Kenksmingos svetainės surinktos iš Microsoft suteikto duomenų rinkinio, „WarningBird“ projekto duomenų, „Twitter“ socialinio tinklo duomenų, „Alexa“ svetainių katalogo duomenų. Iš šių šaltinių surinkta apie 5,5 milijono svetainių URL, kurie buvo panaudoti tolimesniam interneto svetainių rinkimui naudojant paiešką internete [2]. Paieška vykdyta 7 mėnesius naudojant 20 virtualių mašinų [2].

Duomenys kategorizuoti suformuojant svetainių klasterius ir juos peržiūrint. Keli klasteriai identifikuoti kaip nekenksmingos svetainės. Kenksmingos svetainės žymėtos jas tikrinant naudojant Microsoft Forefront kenksmingos programinės įrangos aptikimo įrankį [2]. Didelė dalis URL (78.51%) nebuvo priskirti jokiai klasei, jiems klasė priskiriama tyrimo metu, identifikuojant kenksmingas svetaines.

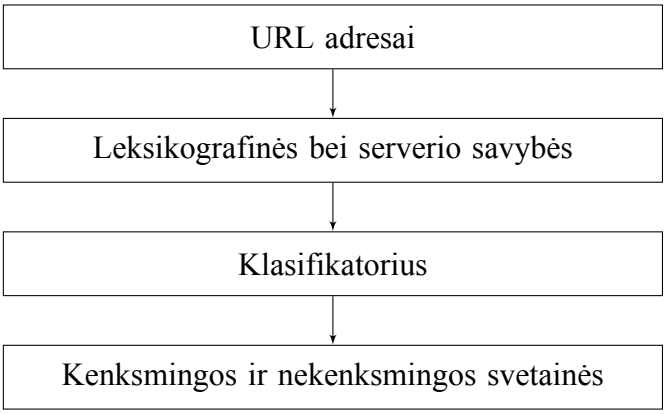
Naudojant turimą duomenų rinkinį dalis žinomų ramstinių serverių yra naudojama kitų ramsčių aptikimui. Atsitiktinai parenkama dalis žinomų ramsčių (1%, 5%, 10%, 50%, 90%), kurie bus naudojami kaip nulinė PageRank įverčio skaičiavimo iteracija. jJiems kenksmingų svetainių formavimo žingsnyje nustatomas pradinis $R(u) = 1$ ir tikrinama kiek kitų žinomų ramsčių yra aptinkama.

Naudojant 5% pradiniam apmokymui aptikta 48,59% kitų ramstinių serverių. Tai reiškia, kad įmanoma aptikti 7 kartus daugiau ramstinių serverių nei buvo panaudota apmokyti pradiniam modeliui. Šiuo atveju pirmojo tipo klaidos pasiekia 2,36%. Tai yra atvejis pasiekęs didžiausią klaidą ir skirtumą tarp pradinio duomenų rinkinio dydžio ir aptiktų ramstinių serverių kiekio. Metodas gali būti taikomas iteracijomis, panaudojant dalį aptiktų ramstinių serverių kitų serverių radimui. Tai leidžia aptikti žymiai daugiau ramstinių serverių, tačiau klaidos vis dar yra pakankamai žemos, leidžiančios praktinį panaudojimą [2].

Naudojant šį metodą pasinaudota interneto svetainių grafo topologija, naudoti papildomi duomenys - tai yra domeno informacija, svetaines talpinančių serverių duomenys [2]. Toks duomenų rinkinys negali būti panaudotas dėl turimo kompiuterinių resursų kiekio – duomenys rinkti 7 mėnesius, naudota 20 įrenginių. Norint tai atlikti naudojant vieną kompiuterį prireiktų daugiau nei 10 metų, tai sukelia sunkumų formuojant tokį duomenų rinkinį. Straipsnyje [6] naudojamas paprastesnis duomenų rinkinys ir siūlomas sprendimų medžio naudojimas kenkėjiškoms svetainėms identifikuoti pagal įvairias jų savybes.

Internetinės svetainės naudoja įvairius pajamų generavimo metodus. Dalis jų yra neteisėti ir kenkia svetainių naudotojams. Tokių svetainių identifikavimas ir blokavimas leidžia apsaugoti

naudotojus nuo kenkėjiškos programinės įrangos. Įprasti metodai naudoja kenksmingų svetainių sąrašus, tačiau šie sąrašai yra baigtiniai, naujos svetainės į juos pridedamos ne iš karto [6]. Alternatyva blokavimo sąrašų sudarymui yra mašininio mokymo modelis, kuris gali atpažinti kenksmingas svetaines pagal jų adresą.



2.4 pav. Straipsnyje analizuojamo metodo schema

Metodo pritaikymo schema yra vaizduojama paveiksle (2.4 pav.). Svetainių klasifikavimui naudojamas C4.5 sprendimų medis.

Sprendimų medis yra vienas iš klasifikavimo modelių [7]. Klasifikavimas vykdomas sukuriant testus, kurie atitinka medžio viršūnes. Įrašai yra tikrinami pagal viršūnes atitinkančius testus, pagal testo rezultatą einama per medį kol pasiekiama kurią nors klasę atitinkanti medžio viršūnė [6]. Algoritmas C4.5 yra paremtas ankstesniu, ID3 medžio formavimo algoritmu [7].

Sprendimų medis formuojamas dviem etapais. Pirma suformuojamas medis, vėliau jis yra supaprastinamas siekiant sumažinti jo sudėtingumą ir galimai padidinti jo tikslumą [7]. Medį sudarantys formuojami testai. Testas tikrina vieną iš atributų. Atributas ir jo vertė parenkami siekiant maksimizuoti informacijos išlošį gaunamą padalinant duomenų rinkinį į du segmentus [4].

Informacijos išlošis skaičiuojamas kaip duomenų rinkinio entropijos skirtumas prieš ir po padalinimo. Viso duomenų rinkinio S entropija gaunama [7]

$$entropija(S) = \sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \times log_2\Big(\frac{freq(C_j, S)}{|S|}\Big) bitai. \quad \textbf{(2.3)}$$

kur $freq(C_j, S)$ yra S aibės elementų skaičius kurie priklauso klasei C_i. Rinkinį T padalinus į rinkinius $T_1, T_2, ..., T_n$ naudojant testą X kuris padalina rinkinį į n grupių skaičiuojama pasverta entropija $entropija_X(T)$ [7]

$$entropija_x(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times entropija(T_i) \quad \textbf{(2.4)}$$

dydis

$$ilošis(X) = entropija(T) - entropija_X(T) \quad \textbf{(2.5)}$$

nusako kiek informacijos gaunama, dalinant rinkinį T naudojant testą X. Šis dydis yra skaičiuojamas visiems galimiems testams. Radus didžiausią vertę kuriamas testas ir pridedamas į formuojamą medį. Toliau tai kartojama su gautomis n medžio šakų, taip suformuojant visą medį, iki kol pasiekiamos konkrečios klasės medžio viršūnėse ir nebėra atributų pagal kuriuos būtų galima atlikti testus [7].

Algoritmas C4.5 yra vystomas iš ID3 [4]. Šis, atnaujintas metodas prideda papildomas savybes. Medžio formavimo metu testuose atsižvelgiama į neegzistuojančias reikšmes, pridedama galimybė naudoti ne tik testus su diskrečiomis reikšmėmis, bet ir su tęstinėmis reikšmėmis. Taip pat pridedamas ir medžio mažinimo žingsnis, kuris atliekamas suformavus visą medį. Tai ne tik supaprastina medį bet ir mažina persimokymą [7]. Taikant metodą interneto svetainių klasifikavimio atveju yra svarbus tik šio algoritmo minimizavimo aspektas, nes tęstinės bei neegzistuojančios reikšmės nepasitaiko duotame duomenų rinkinyje.

Medžio minimizavimui galimi keli metodai. Vieni naudoja papildomą duomenų rinkinį, pagal kurį aptinkamos medžio šakos suformuotos dėl pirminio mokymo duomenų rinkinio persimokymo. Alternatyvus metodas, kuris nereikalauja papildomų duomenų mokymui, yra naudojamas C4.5 algoritme, jis stengiasi prognozuoti klaidų kiekį šakoje, laikant, kad duomenys yra pasiskirstę pagal binominį skirstinį. Tada skaičiuojama, kuriuo atveju klaidų yra daugiau, ar kai medžio šaka pakeista lapu atitinkančiu tam tikrą klasę ar kai paliekama egzistuojanti šaka.

Sprendimų medžiai turi ir tam tikrų apribojimų [7]. Laikant, kad visi tiriamos aibės elementai yra išsidėstę Euklido erdvėje, sprendimų medis klases sukurs išskirdamas stačiakampius regionus erdvėje. Tai reiškia, kad ne stačiakampio formos klasteriai bus imituojami kelių stačiakampių, taip prarandant dalį tikslumo ir didinant sudėtingumą.

Siūloma alternatyva [7] yra dirbtinių neuronų tinklai, jie gali pasiekti didesnį tikslumą nei sprendimų medžiai, rezultatas yra gana nesudėtingai pasiekiamas [7], tačiau jiems suformuoti reikia žymiai daugiau skaičiavimo resursų.

Klasifikavimui naudojamos leksikografinės svetainių adreso savybės bei serverio savybės. Naudojamos svetainės serverio savybės:

1. Duomenų centro vieta;
2. Domeno savininko kontaktinė informacija;
3. Domeno registracijos data;
4. Domeno informacijos atnaujinimo data;
5. Svetainės spartinančiųjų atmintinių informacijos saugojimo laikas;
6. Domeno valstybės kodas;
7. Ryšio su serveriu pralaidumas.

Naudojamos leksikografinės savybės yra formuojamos iš svetainės universalaus adreso (URL). Jis yra dalinamas į segmentus, kuriuos skiria įvairūs simboliai bei skyrybos ženklai. Šie segmentai tampa dvireikšmėmis adreso savybėmis.

Modeliui apmokyti naudotas duomenų rinkinys surinktas iš kelių skirtingų šaltinių. Naudoti 5000 URL iš kurių 1676 yra kenkėjiškų svetainių adresai.

Apmokant modelį naudota kryžminė patikra, 10% duomenų skiriama testavimo duomenų imčiai. Klasifikavimo kokybės metrikos naudojamos metodo kokybei įvertinti yra jautrumas, tikslumas, specifiškumas, AUC metrika. Kryžminė patikra naudota dėl mažo duomenų rinkinio. Ji aprašoma algoritme nr. 1.

1 algoritmas. Kryžminės validacijos algoritmas

1: **visiems** validacijos žingsniams i **kartoti**
2: Atskirti i-tąjį validacijos duomenų rinkinį
3: Apmokyti modelį su likusiais duomenimis
4: Įvertinti modelį su i-tąja validacijos imtimi
5: Išsaugoti validacijos rezultatus
6: Pateikti vidutines validacijų reikšmes

Naudotos klasifikavimo kokybės metrikos yra:

$$jautrumas = \frac{TP}{TP + FN} \cdot 100 \tag{2.6}$$

$$tikslumas = \frac{TP}{jautrumas + specifiškumas} \cdot 100 \tag{2.7}$$

$$specifiškumas = \frac{TN}{TN + FP} \cdot 100 \tag{2.8}$$

kur *TN* yra tikrų neigiamų, *TP* tikrų teigiamų, *FP* klaidingai teigiamų, *FN* klaidingai neigiamų klasifikavimų skaičius.

2.2 lentelė

Straipsnyje apžvelgiamų klasifikatorių kokybė

Metrika / Kategorija	Tikslumas	Jautrumas	Specifiškumas	AUC
Nekenksmingos	98.3%	96.4%	96.5%	0.985
Kenksmingos	92.9%	96.4%	96.5%	0.985

Matavimų rezultatai pateikiami lentelėje 2.2. Rezultatai yra itin geri. Tai gali būti dėl duomenų rinkinio formavimo specifikos. Mokymo duomenų rinkiniui naudojamos svetainės iš egzistuojančių juodųjų sąrašų. Modelis sugeba išmokti tik jau aptiktų svetainių savybių, naudojant jį

po tam tikro laiko tarpo, kai kenksmingos svetainės yra atnaujintos atsižvelgiant į egzistuojančius juoduosius sąrašus modelio rezultatai turėtų prastėti. Taip pat naudojamas duomenų rinkinys yra nedidelis, tai leidžia medžiui gerai prisitaikyti prie jame esančių duomenų.

Straipsnyje lyginami įvairūs metodai skirti svetainių klasifikavimui į kenksmingas ir nekenksmingas. Formuluojama problema yra svetainių klasifikavimas į dvi grupes, naudojant metodus [8]

a) K-artimiausų kaimynų (K-Nearest neighbours);
b) Atraminių vektorių klasifikatorius (angl. *Support Vector Machine*);
c) Paprastasis Bayes klasifikatorius (angl. *Naive Bayes*);
d) K-vidurkių (K-Means);
e) Affinity propagation metodas.

Staripsnyje naudojami metodai su mokytoju ir be. Jų rezultatas yra lyginamas siekiant aptikti geriausiai tinkantį parinktai problemai spręsti [8].

K-Artimiausų kaimynų (K-Nearest neighbours) metodas yra paremtas trimis elementais [4]:

a) Mokymo duomenų rinkinys;
b) Elementų atstumo matas;
c) Analizuojamų kaimynų kiekis k.

Elementų klasifikacija vyksta balsavimo principu - išrenkama k elementų, kurie yra artimiausi pagal parinktą artumo metriką, dominuojanti klasė išrinktame rinkinyje priskiriama klasifikuojamam elementui.

Pagrindiniai šio metodo trūkumai yra tinkamo kaimynų skaičiaus k parinkimas. Per didelis k lems gretimų klasių įtraukimą į balsavimo procesą, per mažas gali lemti blogą rezultatą esant triukšmui pradiniuose, apmokymo, duomenyse. Tai sprendžiama pridedant balso svorį [4] kuris yra atvirkščiai proporcingas elemento atstumui nuo balsuojančio elemento. Taip pat svarbus atstumo mato parinkimas. Dažnai naudojamas Euklido atstumas, tačiau jis nėra tinkamas kai yra daug matmenų, ar jų reikšmės yra itin skirtingos [4]. Tada verta naudoti kitus matus arba normalizuoti atributų vertes.

Šio metodo privalumas yra tai, kad skaičiavimai atliekami tingiai. Pirminio modelio formavimo metu nėra atliekami jokie skaičiavimai, jie vykdomi tik klasifikuojant naujus duomenis [4].

Atraminių vektorių klasifikatoriaus (Support Vector Machine) metodas yra vienas iš patikimiausių klasifikavimo metodų [4]. Jis pasižymi geru rezultatu net turint mažą kiekį mokymo duomenų bei nenukenčia dėl didelio matmenų kiekio [4]. Šio metodo esmė yra duomenų atvaizdavimas kitoje erdvėje kurioje bus galima aptikti hiperplokštumą kuri skiria įrašų klases.

Atraminių vektorių klasifikatorius aprašomas[8]:

$$h(x) = b + \sum_{n=1}^{N} y_i \alpha_i K(x, x_i), \tag{2.9}$$

kur $h(x)$ yra elemento x atstumas nuo klases skiriančios hiperplokštumos, b yra papildomas svorio koeficientas, α yra hiperplokštumos paraštės korekcijos koeficientas mokymo imčiai, N yra matmenų skaičius, K yra branduolio funkcija kuri transformuoja elementus į atraminių vektorių erdvę, x yra klasifikuojamas įrašas.

Apmokant atraminių vektorių klasifikatorių yra svarbu parinkti tinkamą branduolio funkciją, nuo jos itin priklauso modelio rezultatų tikslumas. Mokymas vykdomas parenkant branduolio funkciją ir pskaičiuojant tokius koeficientus α_i su kuriais gaunamas didžiausias riba tarp hiperplokštumos ir teisingai klasifikuotų mokymo duomenų erdvėje.

Šio metodo trūkumas yra sudėtingas modelio apmokymas, apmokant modelį reikia atlikti itin daug skaičiavimų. Tačiau po modelio formavimo jis gali būti panaudojamas nereikalaujant daug resursų [4].

Paprastas Bajeso klasifikatorius yra paremtas Bajeso teorema [8]. Šio metodo privalumas yra paprastas modelio formavimas, nėra hiperparametrų kuriuos reikėtų derinti. Taip pat jo sudėtingumas mokymosi metu laiko atžvilgiu yra tiesinis, mokymosi imties dydžiui. Tai leidžia jį pritaikyti dideliems duomenų rinkiniams [4]. Taip pat rezultatas yra pakankamai geras, ypač įvertinant reikalingą pastangų kiekį norint apmokyti modelį [4].

Modelis formuojamas naudojantis Bajeso teorema ir sąlyginėmis tikimybėmis. Jei klasių aibė yra $i = 0, 1$ tik dviejų elementų, $P(i|x)$ yra tikimybė, kad įrašas su verčių vektoriumi $x = (x_1, ..., x_p)$ priklauso klasei i, tai bet kokia monotoniška $P(i|x)$ gali būti naudojama klasifikavimui [4]. Ši funkcija gali būti išreiškiama kaip santykis $P(1|x)/P(0|x)$. Tai gali būti išreiškiama kaip [4]:

$$\frac{P(1|x)}{P(0|x)} = \frac{f(x|1)P(1)}{f(x|0)P(0)} \tag{2.10}$$

Šią funkciją klasifikavimui galima naudoti radus $f(x|i)$ funkciją. Paprastojo Bajeso klasifikavimo metode laikoma, kad visi x vektoriaus elementai nepriklauso vienas nuo kito tada $f(x|i)$ prastinama į [4]

$$f(x|i) = \prod_{j=1}^{p} f(x_j|i) \tag{2.11}$$

kur visos $f(x_j|i)$ funkcijos yra įvertinamos atskirai, taip supaprastinant problemą į daug vienmačių uždavinių.

Šis metodas leidžia panaudoti nedidelį duomenų kiekį modelio apmokymui, apmokymas vyksta greitai ir paprastai, nereikalauja komplikuotų iteracinių schemų modelio formavimui [4]. Tačiau vienas iš reikalavimų labiausiai ribojančių šį metodą yra visų klasifikuojamo objekto požymių nepriklausomumas tarpusavyje. Tačiau tai galima spręsti naudojant pirminį duomenų apdorojimą, pašalinant stipriai koreliuotas reikšmes [4]. Taip pat metodas gali modeliuoti tik tiesines

ribas tarp klasių, netiesiniams uždaviniams spręsti reikalingos papildomos modelio modifikavijos [8].

K-vidurkių algoritmas yra iteracinis metodas skirtas skaidyti duomenų rinkinį į naudotojo parinktą klasterių skaičių k [4]. Algoritmo metu siekiama minimizuoti visų imties elementų atstumą iki jiems priskirtų klasterių centrų. Algoritmo iteracija susideda iš dviejų žingsnių:

Duomenų priskyrimas. Visi objektai yra priskiriami jiems artimiausiam centroidui. Esant vienodiems atstumais centroidas parenkamas atsitiktinai. Pirmą kartą vykdant šį žingsnį centroidai parenkami atsitiktinai. Šio žingsnio rezultatas yra duomenys, suskaidyti į k grupių.

Centrų tikslinimas. Perskaičiuojamos grupių vidutinės parametrų vertės, parenkamas naujas centroidas esantis arčiausiai tikrojo grupės centro. Šie algoritmo žingsniai yra kartojami kol centroidai nebesikeičia [4]. Atstumui įvertinti naudojamas Euklido atstumas [8]:

$$||x_n - \mu_k||^2 = \sqrt{\sum_{i=1}^{D}(x_{ni} - \mu_{ki})^2} \tag{2.12}$$

kur μ_k yra klasterio centroidas, D yra duomenų rinkinio dydis, x yra vienas iš duomenų rinkinio elementų.

K vidurkių metodas turi trūkumų. Galutiniai klasteriai priklauso nuo pirminių taškų pasirinkimo. Netinkamai parinkti taškai gali lemti neoptimalų sprendimą. Taip pat k-vidurkių metodas nesugeba išskirti klasterių kurie nėra vienas nuo kito aiškiai atskirtos sferos erdvėje.

Dėl vidurkio funkcijos naudojimo centroidų parinkimui rezultatas gali būti stipriai iškreiptas kelių išskirčių. Tai gali būti sprendžiama naudojant papildomą duomenų valymą, pirma suskaidant į daugiau klasterių siekiant, kad išskirtims būtų priskiriami maži klasteriai, ir vėliau jas prijungiant prie didesnių klasterių [4].

Affinity propagation yra vienas iš klasterizavimo metodų [8]. Šio metodo įvestis yra panašumų matrica, kurioje saugoma visų duomenų įrašų porų panašumai $s[i, j]$ kur $i, j = (1, 2, ..., N)$. Naudojant šį metodą siekiama surasti kiekvieno elemento duomenų aibėje atstovą. Tą galima analizuoti kaip žinučių tarp duomenų įrašų perdavimą [9]. Kiekvienai elementų porai i ir j egzistuoja dvi žinutės – atsakomybė (responsibility), $r[i, j]$ siunčiama iš taško i į j, kuri parodo sukauptus įrodymus apie elemento j galimybę atstovauti elementą i klasteryje. Antra žinutė yra tinkamumas (availability) $a[i, j]$, tai žinutė siunčiama iš elemento j elementui i kuri parodo j elemento tinkamumą atstovauti elementą i. Iš pradžių visos a ir r reikšmės nustatomos į nulį ir yra iteratyviai atnaujinamos pagal [9]:

$$r[i, j] = (1 - \lambda)\rho[i, j] + \lambda r[i, j]a[i, j] = (1 - \lambda)\alpha[i, j] + \lambda a[i, j] \tag{2.13}$$

kur λ yra konstanta naudojama sumažinti svyravimus, galimos reikšmės yra $0 \geq \lambda < 1$, o $\rho[i, j]$ ir $\alpha[i, j]$ yra propaguojama atsakomybė ir propaguojamas tinkamumas [9]. Šie dydžiai gaunami pagal [9]:

$$\rho[i, j] = \begin{cases} s[i, j] - max_{k \neq j}\{a[i, k] + s[i, k]\} & (i \neq j) \\ s[i, j] - max_{k \neq j}\{s[i, k]\} & (i = j) \end{cases} \tag{2.14}$$

ir [9]:

$$\alpha[i,j] = \begin{cases} min\{0, r[i,j] + \sum_{k \neq i,j} max\{0, r[k,j]\}\} & (i \neq j) \\ \sum_{k \neq i,j} max\{0, r[k,j]\} & (i = j) \end{cases} \quad \textbf{(2.15)}$$

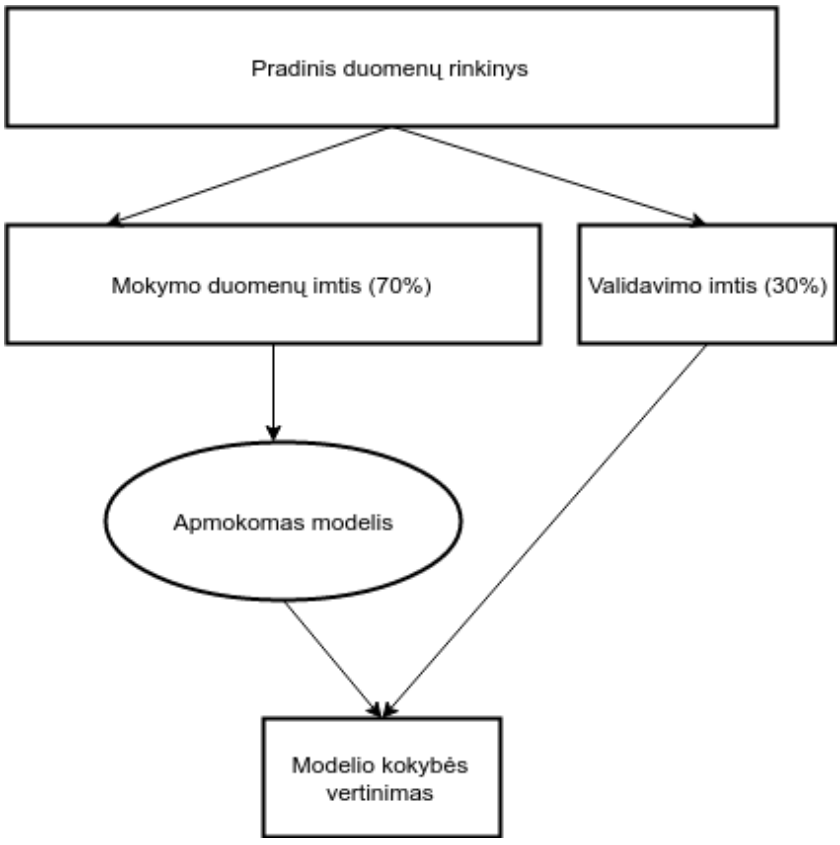
žinutės yra gaunamos iš atitinkamų propaguojamų žinučių. Galiausiai atstovas elementui i yra apibrėžiamas kaip [9]

$$argmax\{r[i,j] + a[i,j] : j = 1, 2..., N\} \quad \textbf{(2.16)}$$

Šis metodas leidžia kontroliuoti suformuojamų klasterių kiekį per λ vertę [8]. Šis metodas pasižymi geresne greitaveika lyginant su K-vidurkių metodu [9].

Pritaikant šiuos metodus svetainės turi būti supaprastinamos iki tam tikrų savybių vektoriaus. Tyrimo metu nauodtas svetainių turinys, jų URL adresai [8]. Svetainių turinys, kuris yra anglų kalba buvo analizuojamas siekiant išgauti jų semantinę prasmę. Tam naudota Term Frequency - inverse document frequency (TFIDF) metodika. Taip pat į svetainių savybes įtraukta ir jų HTML kodo struktūra, jų adreso semantinė reikšmė, nuorodos į kitus tinklalapius, atvaizduoto tinklalapio vaizdas.

Duomenų rinkinys surinktas iš dviejų pagrindinių šaltinių, Alexa katalogo, iš kurio išgautos nekenksmingos svetainės, bei iš Phishtank žalingų svetainių registro. Surinkta 100000 svetainių informacija.



2.5 pav. Modelio apmokymo ir validacijos proceso schema

Modeliai apmokyti naudojant 70% pradinių, sužymėtų duomenų apmokymui bei 30% testavimui 2.5 pav.. Duomenų rinkinys atsitiktinai skaidomas į dvi dalis, mokymo ir validacijos. Mokymo duomenys naudojami modelio apmokymui, o validacijos duomenys naudojami įvertinti modelį. Taip yra išvengiama modelio persimokymo, galima teisingai suderinti modelio hiperparametrus. Kenksmingų ir nekenksmingų svetainių santykis išlaikytas vienodas testavimo ir mokymo imtyse.

2.3 lentelė

Strapsnyje pateikiami modelių rezultatai

Metrika \ Modelis	KNN	LS	RS	NB
Tikslumas, naudojant 50 įrašų	74%	80%	79%	77%
Tikslumas, naudojant 100 įrašų	75%	82%	83%	78%
Tikslumas, naudojant 500 įrašų	79%	86%	92%	78%
Tikslumas, naudojant 5000 įrašų	91%	93%	97%	84%
Tikslumas, naudojant 100,000 įrašų	95%	93%	98%	89%
AUC	0.66	0.93	0.91	0.88

kur modeliai:

KNN - K-artimiausių kaimynų

LS - Tiesinio branduolio atraminių vektorių klasifikatorius

RS - Radial Basis Function branduolio atraminių vektorių klasifikatorius

NB - Paprastasis Bayes klasifikatorius

Modelių rezultatai pateikiami 2.3 lentelėje. Modelis pasiekęs geriausius rezultatus yra tiesinio branduolio atraminių vektorių klasifikatorius. Deja šiame straipsnyje nėra vertinami sprendimų medžiai, dėl to nėra aišku, koks būtų jų rezultatas naudojant šiuos duomenų ir savybių rinkinius. Analizuojant egzistuojančią literatūrą aptikta, kad geri rezultatai klasifikuojant kenksmingas ir nekenksmingas svetaines yra pasiekiami naudojant du metodus: C4.5 sprendimų medį [6], atraminių vektorių klasifikatorių [8]. Šie metodai bus mėginami bandant išrinkti geriausiai tinkantį parinktam panaudos atvejui.

Duomenų rinkiniai yra formuojami jungiant įvairių juodųjų sąrašų duomenis bei naudojant privačius duomenų rinkinius kurie taip pat atitinka juoduosius sąrašus [6, 8, 3].

Tinklalapių parametrai yra formuojami iš jų adresų (URL) [6, 8]. Interneto grafo struktūra naudojama straipsnyje [2] stengiantis identifikuoti klasterių centrus. Siūloma bandyti šią metriką pritaikyti ir klasifikavimo uždavinyje. Taip pat naudojami papildomi duomenys apie talpinančius serverius, įvairius svetainių metaduomenis, tačiau šie duomenys nėra viešai pasiekiami, ir jų surin-

kimui naudojami itin dideli skaičiavimo resursai – [2] straipsnio atveju naudota 20 virtualių mašinų, renkančių duomenis 7 mėnesius. Dėl to naudojami tik URL bei svetainių grafo PageRank įverčiai.

Modeliai formuojami naudojant įvairius metodus, tačiau dalinimas į mokymo, testavimo ir validavimo imtis yra paprastas ir tinkamas, turint pakankamai didelį duomenų rinkinį. Kryžminė validacija gali būti naudojant esant mažesniems rinkiniams [6]. Modeliai vertinami naudojant įvairias metrikas: tikslumą, validumo vertinimas naudojant AUC, specifiškumą, jautrumą.

2.2. DARBO TIKSLAS IR UŽDAVINYS

Kenksmingi tinklalapiai skirstomi į skirtingas kategorijas pagal jų vykdomą [1] ir pagal jų paskirtį žalingų tinklalapių tinkle [2]. Jų aptikimui naudojami įvairūs metodai [8]. Šiame darbe bus tobulinamas tokių tinklalapių aptikimas, naudojant interneto grafo bei tinklalapių URL informaciją.

Darbo tikslas – pasiūlyti metodiką ir sukurti įrankį kenksmingų ir nekenksmingų tinklalapių klasifikavimui, kur kenksmingi tinklalapiai yra kenksmingų tinklalapių sąrašuose aptinkami adresai.

Sprendžiami uždaviniai:

a) savybių vektoriaus formavimo ir klasifikavimo modelių parinkimas;
b) kenksmingų interneto tinklalapių identifikavimo modelio sukūrimas;
c) kenksmingų interneto tinklalapių identifikavimo modelio programinė realizacija;
d) kenksmingų interneto tinklalapių identifikavimo modelio pritaikymas analizuojant realų duomenų rinkinį.

3. TYRIMŲ METODAI

Darbo galutinis rezultatas yra įrankis, kurio įvestis yra tinklalapio adresas, o rezultatas – klasifikacija ir jos kokybės vertinimas. Tinklapio adresas (URL) yra pakankamas identifikuoti tinklalapius. Taip pat naudojantis juo galima pasiekti patį tinklapį, taip įrankiui paliekama galimybė pridėti papildomas savybes, tokias kaip tinklalapio turinio peržiūra.

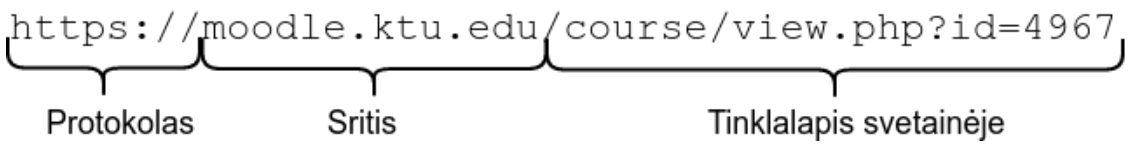
Klasifikavimo modeliai kaip įvestį naudoja parametrų vektorių, dažniausiai tai yra n ilgio vektorius $X = (x_1, x_2, ..., x_n)$ žymintis klasifikuojamo objekto savybes. Savybės išreiškiamos skaitinėmis vertėmis. Klasifikatoriaus tikslas - kiekvienam vektoriui X priskirti klasę Y. Dėl to viena pirmų užduočių yra nagrinėjamo objekto, tinklalapio adreso, transformavimas į savybių vektorių X.

3.1. SAVYBIŲ VEKTORIAUS X FORMAVIMAS

Tinklalapio savybių vektorius X gaunamas iš tinklalapio adreso bei svetainės PageRank įverčių. Tinklalapio adresas yra simbolių eilutė. Visą tinklapio adresą atitinkantis vektorius X yra sudarytas iš savybių:

a) srities varde esantys žodžiai;
b) užklausoje esantys žodžiai;
c) taško simbolių skaičius adrese;
d) specialiųjų simbolių skaičius adrese (Naudojami simboliai: `-`, `.`, `_`, `~`, `:`, `/`, `?`, `#`, `[`, `]`, `@`, `!`, `$`, `&`, `'`, `(`, `)`, `*`, `+`, `,`, `;`, = ir tarpo simbolis);
e) kenksmingų svetainių PageRank įvertis;
f) nekenksmingų svetainių PageRank įvertis.

Dėl žodžių naudojimo savybių vektoriuje jo ilgis priklauso nuo naudojamo duomenų rinkinio. Kiekvienam žodžiui rinkinyje naudojama po vieną lauką vektoriuje, dėl to jis gali tapti ganėtinai ilgas.



3.1 pav. Tinklalapio adreso (URL) struktūra

Tinklalapio adreso pavyzdys yra pateiktas 3.1 pav.. Čia matoma ir jo struktūra. Adreso pradžioje yra protokolas, dažniausiai aptinkami `http://` ar `https://`. Antras segmentas yra sritis. Kreipiantis į tinklalapį naudojamas sričių vardų serveris, kuris susieja simbolinį adresą, sritį,

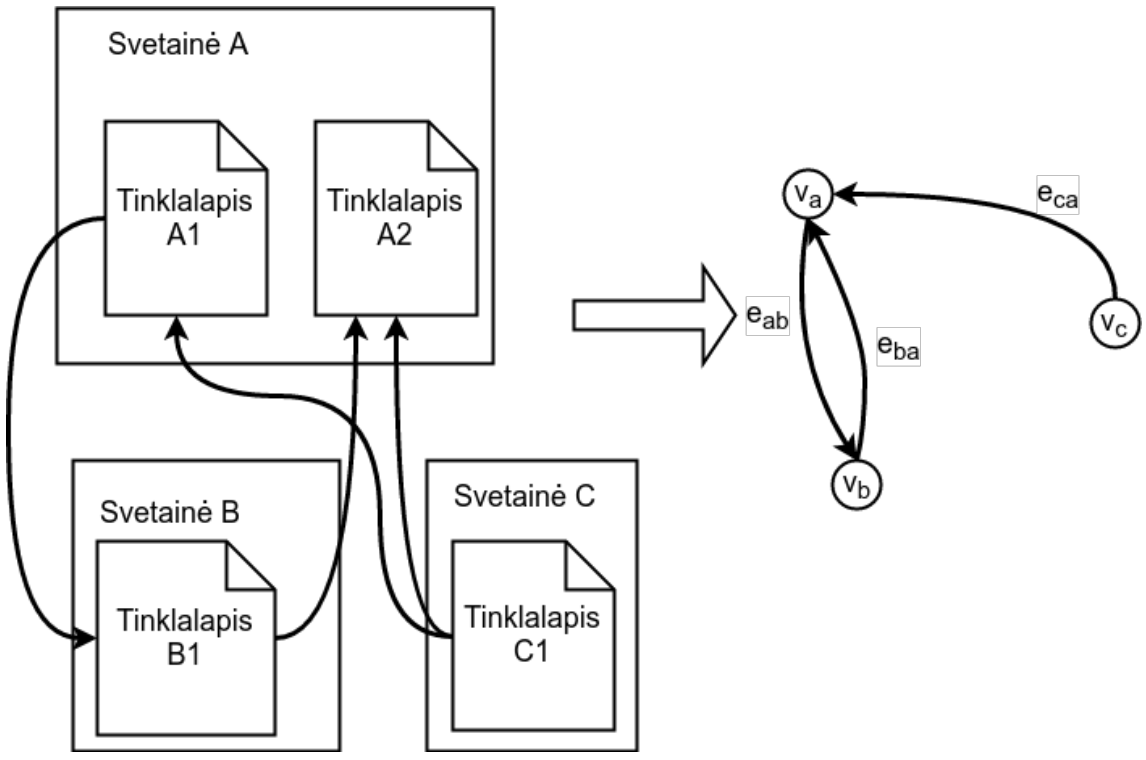
su skaitiniu adresu, pagal kurį galima kreiptis į svetainę talpinantį serverį. Sritis parenkama taip, kad atspindėtų svetainės turinį, būtų lengvai įsimenama naudotojams, bei būtų lengvai randama paieškos varikliuose. Sričių vardus suteikia sričių registrai, už tam tikrą mokestį. Kaina nustatoma pagal raktažodžių populiarumą. Paskutinis adreso segmentas yra užklausa. Ji leidžia svetainė talpinančiam serveriui pateikti tą dokumentą kuris atitinka tinklalapį kurį bando pasiekti naudotojas.

Išskiriamos savybės iš tinklalapio adreso yra:

a) srities varde esantys žodžiai;
b) užklausoje esantys žodžiai;
c) taško simbolių skaičius adrese;
d) specialiųjų simbolių skaičius adrese.

Žodžiai adrese yra išskiriami kaip dviejų reikšmių savybės, 0 jei žodis neegzistuoja adrese, 1 jei žodis yra adrese. Laikoma, kad žodžius skiria specialieji simboliai.

PageRank įverčiai gaunami iš orientuoto svetainių grafo $G = (V, E)$, kur V yra šio grafo viršūnės o E lankai. Grafo formavimas iš interneto svetainių yra pavaizduotas 3.2 pav. .



3.2 pav. Svetainių grafo formavimas

Grafo viršūnės V yra svetainės, o lankai atitinka ryšius tarp svetainių. Tai yra nuorodos tarp svetainių aptinkamos svetainių tinklalapiuose. Šie lankai turi kryptį. Jei svetainės v_a tinklalapyje yra nuoroda į svetainės v_b tinklalapį laikoma, kad egzistuoja lankas e_{ab} iš viršūnės v_a į viršūnę v_b. Visų lankų svoris yra vienodas, tai yra jei keli svetainės v_a tinklalapiai turi nuorodas į svetainės v_b tinklalapius, vis vien laikoma, kad yra tik vienas lankas e_{ab}. Galimi abipusiai svetainių ryšiai,

tai yra jei egzistuoja lankas e_{ab} lankas e_{ba}, kuris rodo ryšį iš viršūnės v_b į viršūnę v_a taip pat gali egzistuoti.

Svetainių grafas naudojamas skaičiuoti PageRank įverčiams. PageRank įvertis čia apibrėžiamas [5]:

$$R(v) = (1-d) + d \sum_{u \in B_v} \frac{R(u)}{N_u}, \tag{3.1}$$

kur $R(v)$ yra viršūnės v PageRank įvertis, d - slopinimo faktorius, B_v viršūnės u kurios turi lankus e_{uv} su viršūne v. N_u yra viršūnės u išėjimo laipsnis, tai yra skaičius lankų kurie veda iš viršūnės u. Slopinimo faktorius d gali turėti reikšmes tarp 0 ir 1.

PageRank įvertis negali būti tiesiogiai apskaičiuojamas pagal 3.1 formulę. Naudojamas iteracinis metodas algoritmas nr. 2 skaičiavimui:

2 algoritmas. Iteracinis PageRank įverčio skaičiavimo algoritmas

1: $R_0 \leftarrow S$
2: **kartoti**
3: **visiems** v_i grafe G **kartoti**
4: $r \leftarrow 0$
5: **visiems** $u \in B_{v_i}$ **kartoti**
6: $r \leftarrow r + \frac{R_i(u)}{N_u}$
7: $R_{i+1}(v_i) \leftarrow (1-d) + dr$
8: $\delta \leftarrow \|R_i - R_{i+1}\|$
9: **kol** $\delta < e$

Šis būdas apskaičiuoti PageRank priklauso nuo nuo kelių parametrų lentelėje 3.1 . Vienas iš jų yra S_0, nulinės iteracijos įvertis. Šios iteracijos metu nustatytos reikšmės bus naudojamos tolimesnių viršūnių įverčiui rasti. Nustatant šį vektorių galima koreguoti kurios viršūnės yra svarbios skaičiavimų pradžioje. Šio vektoriaus keitimas leidžia pritaikyti PageRank įverčio rezultatą. Parametras d yra slopinimas, nuo jo priklauso metodo konvergavimo greitis. Viskas vykdoma kol pasiekiamas pasirinktas konvergavimo lygmuo, kurį galima aptikti pagal tai, jog iteruojant įvertis kinta nežymiai. Minimali iteracijos įverčio pakitimo riba nustatoma kintamuoju e.

3.1 lentelė

Iteracinio PageRank įverčio skaičiavimo algoritmo parametrai

Parametras	Reikšmė
G	Grafas kurio įverčiai skaičiuojami
S	Pirminės įverčio reikšmės
d	Slopinimo faktorius
e	Siekiama klaida tarp iteracijų

PageRank įvertis skaičiuojamas du kartus, naudojant skirtingas pradines reikšmes. Skaičiuojant nekenksmingų svetainių įvertį mokymo imties svetainių viršūnėms priskiriama nenulinė reikšmė, o visoms kitoms svetainėms nustatoma nulinė pradinė reikšmė. Skaičiuojant kenksmingų svetainių įvertį nenulinės reikšmės priskiriamos kenksmingų tinklalapių svetainių viršūnėms. Priskiriamos vertės yra vienas iš modelio mokymo metu derinamų hiperparametrų.

Gavus PageRank įverčius, jie yra naudojami suformuoti vektoriui X, taip baigiant tinklalapių nuorodų transformavimą į klasifikavimo modeliams pritaikytą savybių vektorių.

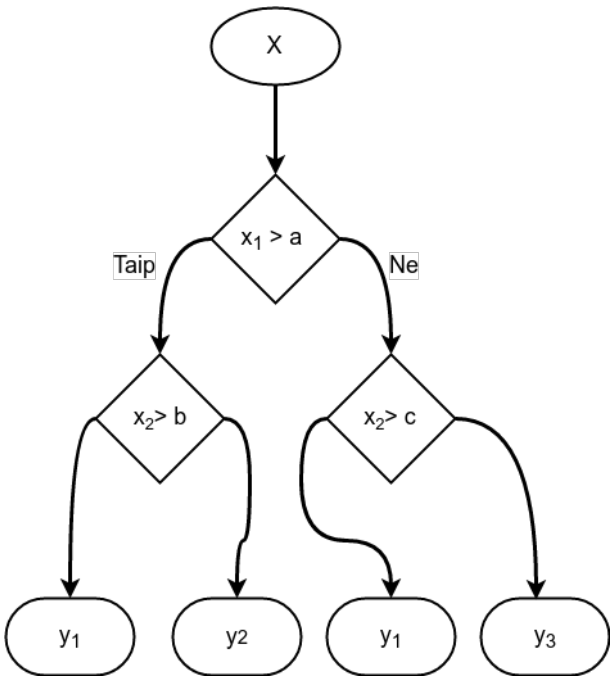
3.2. KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO MODELIS

Lyginami du klasifikavimo modeliai:

a) C4.5 sprendimų medžio modelis;
b) atraminių vektorių modelis.

3.2.1. C4.5 sprendimų medžio modelis

C4.5 sprendimų medis yra sprendimų medžio tipo klasifikatorius kurio formavimui naudojamas C4.5 algoritmas. Sprendimų medis yra programa, suformuota iš tam tikrų testų, kurie tikrina nagrinėjamo objekto parametrus, ir pagal juos parenkama tam tikra klasė.



3.3 pav. Sprendimų medžio pavyzdys

Sprendimų medžio pavyzdys klasifikuojantis vektorių X į klases $Y = (y_1, y_2, y_3)$ pateikiamas 3.3 pav. . Jį sudaro sąlyginių viršūnių rinkinys T.

Sprendimų medį sudarančios sąlyginės viršnūnės yra formuojamos modelio mokymo metu. Mokymui naudojamas duomenų rinkinys H. Medžio formavimo metu parenkami testai kurie maksimizuoja funkciją:

$$santykinisIlois(t_i) = \frac{ilois(t_i)}{santykinEntropija(t_i)} \tag{3.2}$$

Santykinis išlošis priklauso nuo informacijos išlošio, kuris būtų maksimizuojamas naudojant ID3 algoritmą bei santykinės entropijos. Santykinė entropija naudojama siekiant sumažinti informacijos išlošio iškreipimą kai testas skaido rinkinį į daug skirtingų klasių [7]

$$ilois(t_i) = entropija(H) - entropija_{t_i}(H) \tag{3.3}$$

informacijos išlošis vertina duomenų rinkinio klasių atsiskyrimą - skaičiuojama kiek informacijos suteikia padalinimas naudojant nagrinėjamą testą

$$entropija_{t_i} = \sum_{j=1}^{n} \frac{|H_i|}{H} \times entropija(H_i), \tag{3.4}$$

$$entropija(S) = -\sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \times log_2\big(\frac{freq(C_j, S)}{|S|}\big), \tag{3.5}$$

$$santykinEntropija(S) = \sum_{j=1}^{n} \frac{|H_i|}{H} \times log_2\big(\frac{|H_i|}{H}\big), \tag{3.6}$$

kur $freq(C_j, S)$ žymi klasės C_j pasikartojimo kiekį rinkinyje S. Aptikus testą kuris maksimizuoja santykinį išlošį 3.2 testas yra įtraukiamas į medį. Mokymo duomenų rinkinys H yra skaidomas pagal testą į kelis rinkinius[7]. Šie duomenų rinkiniai priskiriami medžio šakoms atitinkančioms testo rezultatus ir rekursiškai tęsiamas medžio formavimas.

C4.5 medžio formavimo algoritmas palaiko diskrečius parametrus bei tolydžius kintamuosius. Sąlyginių viršūnių testai vykdomi diskretiems parametrams sukuria po vieną šaką kiekvienai galimai parametro reikšmei.

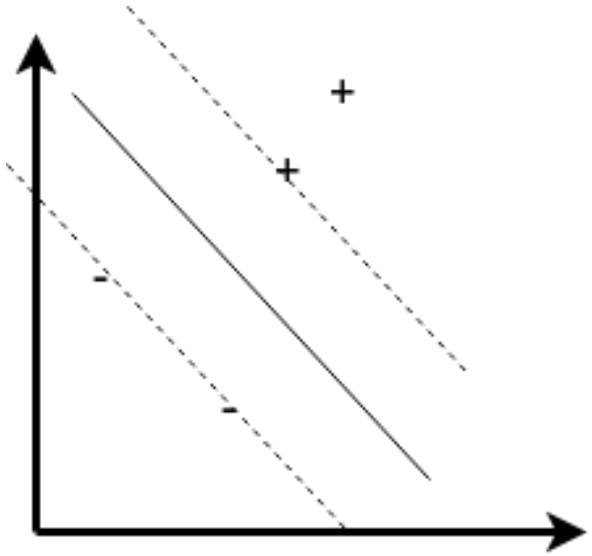
Sąlyginių viršūnių testai tolydiems parametrams vykdomi parenkant ribinę vertę Z, su kuria bus lyginama parametro vertė [7]. Kadangi apmokymui naudojamas baigtinis duomenų rinkinys, jame yra baigtinis kiekis galimų tolydaus parametro A reikšmių $\{v_1, v_2, ..., v_m\}$. Ribinei vertei Z priskiriama viena iš egzistuojančių parametro A verčių v_i. Parenkama vertė kuri maksimizuoja santykinį išlošį 3.2.

Suformavus medį vykdomas medžio mažinimas. Dalis šakų yra sukeliančios persimokymą - tai yra jos yra per daug pritaikytos mokymo imčiai [7]. Šios šakos yra aptinkamos skaičiuojant tikėtiną klaidų dažnį medžiui be tam tikro testo ir su potencialiai pertekliniu testu.

Šis metodas neturi hiperparametrų kuriuos būtų galima derinti.

3.2.2. Atraminių vektorių modelis

Atraminių vektorių klasifikatorius naudojamas klasifikuoti vektorius į dvi klases [8]. Tai atliekama atskiriant objektus erdvėje naudojant tam tikrą hiperplokštumą. Tai vaizduojama 3.4 pav.. Objektų erdvė yra transformuojama naudojant branduolio funkciją, kuri leidžia parinkti tokią erdvę kurioje įmanomas klasių atskyrimas naudojant hiperplokštumą.



3.4 pav. Supaprastinta atraminių vektorių klasifikatoriaus schema

.

Atraminių vektorių klasifikatoriaus modelis išreiškiamas[8]:

$$h(x) = b + \sum_{n=1}^{N} y_i \alpha_i K(x, x_i), \tag{3.7}$$

kur $h(x)$ yra atstumas nuo hiperplokštumos skiriančios klases, b yra papildomas svorio koeficientas, y_i yra i-tojo mokymo rinkinio elemento klasė, o x_i yra i-tasis mokymo imties elementas. N yra mokymo imties elementų skaičius, K yra naudojama branduolio funkcija kuri parinkta optimaliai erdvės transformacijai, x yra klasifikuojamas objektas.

Modelis yra formuojamas apibrėžiant [10]:

$$W(\alpha) = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j) \tag{3.8}$$

ir sprendžiant atraminių vektorių kalsifikatoriaus kvadratinio optimizavimo uždavinį:

$$\max_{\alpha} W(\alpha) \text{kur} \begin{cases} \sum_i \alpha_i = 0 \\ A_i \leq \alpha_i \leq B_i \\ A_i = \min(0, Cy_i) \\ B_i = \max(0, Cy_i) \end{cases} \tag{3.9}$$

Kur C yra hiperparametras naudojamas modelio formavime. Naudojant mažas C vertes galima pritaikyti modelį triukšmingiems pradiniams duomenims, o didelės vertės galimos kai visi mokymo duomenų rinkinio pavyzdžiai turi teisingas klases.

Šio modelio naudojama branduolio funkcija gali būti laikoma vienu iš hiperparametrų kurie yra derinami modelio mokymo metu. Nagrinėjami trys branduolio tipai [11]:

a) tiesinis,
b) polinominis,
c) radialinės bazės.

Tiesinis branduolio tipas yra naudojamas kai funkcija K yra nenaudojama, tokiu atveju atraminių vektorių klasifikatorius yra apibrėžiamos kaip

$$W(\alpha) = \sum_i \alpha_i y_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j x_i x_j. \tag{3.10}$$

Šis modelio variantas gali būti naudojamas kai objektai nagrinėjamoje erdvėje gali būti atskirti tiesia plokštumas, nėra reikalingos papildomos modifikacijos.

Polinominis branduolys gali būti formuojamas naudojant [11]

$$K(u, v) = (1 + u \cdot v)^d, \tag{3.11}$$

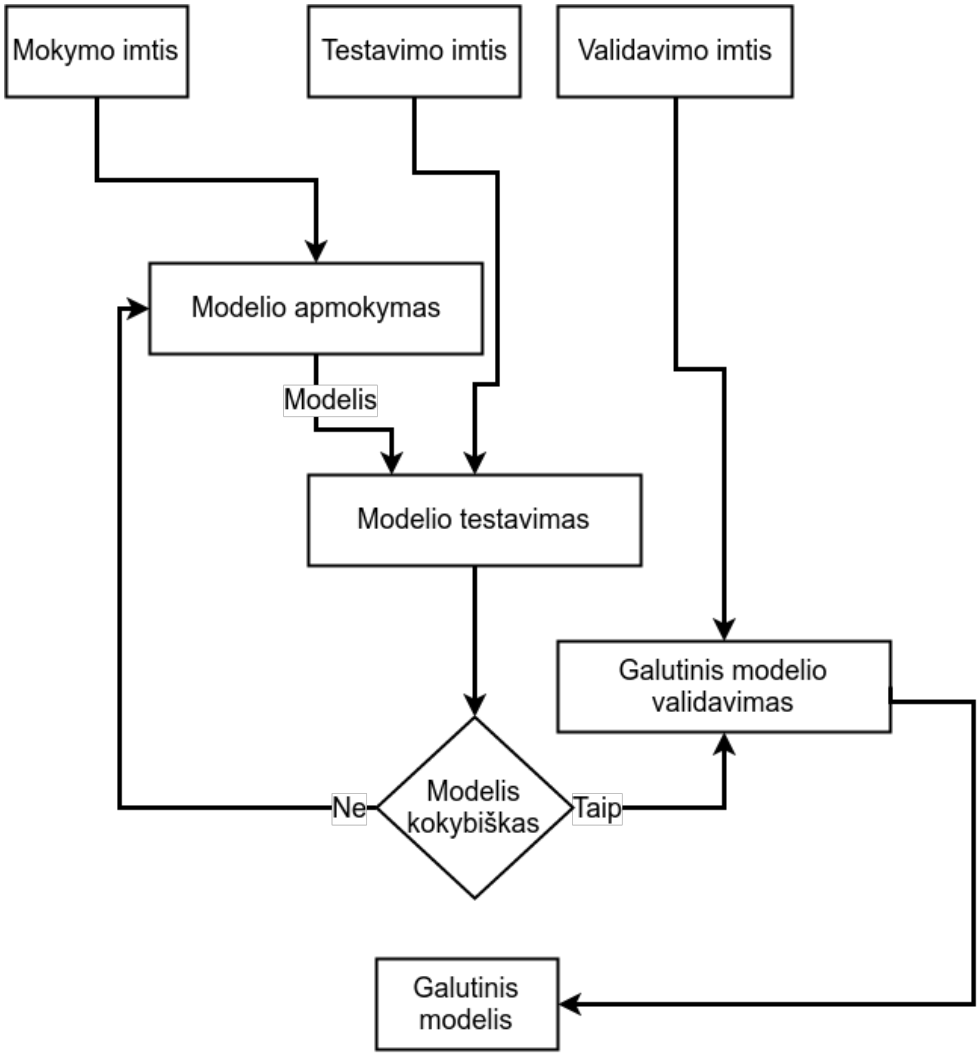
radialinės bazės branduolys išreiškiamas

$$K(u, v) = exp\Big(- \frac{||u - v||^2}{2\sigma^2} \Big). \tag{3.12}$$

Šis modelis turi du hiperparametrus - tai konstanta C, bei parenkama branduolio funkcija. Branduolio funkcija taip pat prideda papildomus hiperparametrus. Polinominio branduolio atveju tai yra laipsnio parametras d, radialinės bazės branduolio atveju tai yra parametras σ.

3.3. KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO MODELIO APMOKYMAS

Kenksmingų interneto tinklalapių identifikavim modelio apmokymo schema yra pateikiama 3.5 pav..



3.5 pav. Modelio formavimo schema

Modeliams mokyti naudojamas duomenų rinkinys yra dalinamas į tris imtis:

a) mokymo imtis (70% viso duomenų rinkinio);
b) testavimo imtis (25% viso duomenų rinkinio);
c) validavimo imtis (5% viso duomenų rinkinio).

Didžiausia, mokymo imtis yra naudojama modelio mokymui. Siekiant aptikti modelių persimokymą naudojama validavimo imtis. Modelį apmokius yra įvertinama jo kokybė naudojant parinktas metrikas, jei įverčiai stipriai skiriasi mokymosi ir testavimo imčiai modelis yra persimokęs, dėl to reikia koreguoti modelio hiperparametrus. Jei gauta modelio kokybė netenkina, taip pat koreguojami hiperparametrai.

Suformavus modelį naudojant mokymo ir testavimo imtis jis yra tikrinamas paskutinį kartą. Tam naudojama validavimo imtis. Modelio rezultatai naudojami įvertinti modelio klasifikavimo kokybę.

Modelio klasifikavimo kokybės įverčiai yra tikslumas, AUC įvertis, specifiškumas, jautrumas. Šios metrikos leidžia įvertinti modelį net ir esant klasių disbalansui. Klasifikavimo rezultatų lentelę sudaro keturi dažniai:

a) teisingai teigiamų dažnis (žymima TP);
b) klaidingai teigiamų dažnis (žymima FP);
c) teisingai neigiamų dažnis (žymima TN);
d) klaidingai neigiamų dažnis (žymima FN).

Atsižvelgiant į ROC kreivę (angl. *receiver operating characteristic curve*), galima nustatyti klasių intervalus kurie yra optimalūs modelio naudojimui. Ši kreivė rodo TP dažnio santykį su FP dažniu, keičiant ribinę klasifikatoriaus vertę, pagal kurią yra parenkama klasė. AUC metrika yra skaičiuojama kaip plotas, esantis po šia kreive. Kitos klasifikavimo kokybės metrikos apskaičiuojamos:

$$\text{jautrumas} = \frac{TP}{TP + FN} \tag{3.13}$$

$$\text{tikslumas} = \frac{TP}{\text{jautrumas} + \text{specifiškumas}} \tag{3.14}$$

$$\text{specifiškumas} = \frac{TN}{TN + FP} \tag{3.15}$$

Šių kokybės metrikų galimos reikšmės yra intervale $[0, 1]$, siekiama, kad jos būtų kuo arčiau 1.

4. METODŲ TAIKYMAS IR REZULTATAI

Darbe siūlomos metodikos taikymas realiam uždaviniui sudarytas iš dviejų žingsnių - modelio apmokymas bei apmokyto modelio naudojimas.

4.1. DUOMENŲ RINKINYS

Analizuojamas duomenų rinkinys yra sudarytas iš trijų komponentų:

a) svetainių grafo duomenys;
b) kenksmingų tinklalapių sąrašas;
c) nekenksmingų tinklalapių sąrašas.

4.1.1. Svetainių grafas

Interneto grafo duomenys naudojami iš [12] projekto teikiamų duomenų rinkinių. Naudojamas grafą sudaro 22 milijonai viršūnių bei 123 milijonų briaunų. Grafas yra pateikiamas WebGraph formatu. Šis formatas yra optimizuotas grafo užimamos vietos diske atžvilgiu. Jame yra saugomi grafo viršūnių numeriai bei briaunos. Viršūnes atitinkančios svetainės yra randamos naudojant pagalbinį failą talpinantį svetainės adresą bei jos eilės numerį grafe. Grafas gali būti nuskaitomas naudojant atviro kodo biblioteką sukurtą pagal [13] straipsnį.

4.1.2. Kenksmingų tinklalapių sąrašas

Kenksmingi tinklapiai surinkti naudojant egzistuojančius juoduosius sąrašus [14], [15] [16] [17]. Taip suformuotas 48761 kenksmingų tinklalapių sąrašas. Jų tipas, pagal vykdomą ataką, nėra išskiriamas. Šių tinklalapių duomenų rinkiniai yra naudojami kuriant juoduosius sąrašus. Visi rinkiniai sujungiami į vieną tekstinį failą, kuriame vienoje eilutėje suagoma nuoroda į vieną kenksmingą tinklalapį.

4.1.3. Nekenksmingų tinklalapių sąrašas

Nekenksmingų tinklapių sąrašas yra formuojamas iš [12] projekto teikiamų duomenų. Naudojamas tinklalapių grafų duomenų rinkinio tinklalapių sąrašas. Šiame duomenų rinkinyje yra apie 1.7 milijardo tinklalapių. Suspausti šie duomenys diske sudaro apie 20GB informacijos. Išskleidus tai sudarytų apie 133GB duomenų. Tai yra gana daug norint juos apdoroti. Viso sąrašo naudojimas sukeltų per didelį klasių disbalansą, dėl to parenkama dalis įrašų. Įrašai yra išrenkami naudojant

scenarijų:

```
zcat index-000* | parallel --pipe \
 "awk 'BEGIN {srand()} { if (rand() <= 0.0005798) {print $1}}\'" > \
   /media/ssd/final_data/whitelist.txt
```

Šiuo atveju naudojamos atviro kodo programos zcat, parallel [18], awk. Šios programos parinktos dėl galimybės failų turinį apdoroti kaip duomenų srautą. Pirmas programos fragmentas skaito visų failų turinį, antras fragmentas skaldo failų turinį į fragmentus, kurie apdorojami lygiagrečiai. Kiekvienas lygiagretus procesas išrenka eilutes. Eilutės perkėlimo į galutinį failą tikimybė $P = 0.0005798$. Šis tikimybė siekiant suformuoti duomenų rinkinį, kurio 5% sudaro kenksmingos svetainės, 95% nekenksmingos svetainės. Lygiagretus vykdymas naudojamas atsitiktinių skaičių generavimui vykdymo metu. Naudojant vieną procesą kompiuterio resursai būtų naudojami neefektyviai, šiuo atveju yra išnaudojami visi turimi skaičiavimo resursai - nauji procesai yra kuriami tol kol pilnai panaudojami visi procesoriaus branduoliai. Galiausiai gaunamas nekenksmingų svetainių sąrašas.

Dalis tinklalapių naudoja papildomą parametrų kodavimą URL-kodavimo principu. Šis kodavimas tinklalapių adresuose neleistinus simbolius ar rezervuotus simbolius keičia specialiais procento simboliu prasidedančiais kodais. Adresai yra dekoduojami naudojant prieduose pateiktas programas.

4.1.4. Duomenų paruošimas mokymui

Suformuoti duomenų rinkiniai yra skaidomi ir paruošiami apmokymui. Duomenų paruošimo programos išeities kodas yra pateikiamas prieduose (1 priedas.). Programoje įgyvendintas PageRank įverčio skaičiavimas.

PageRank įvertis skaičiuojamas iteracijomis. Vienos iteracijos išeities kodas:

```
def pagerank(transposedGraph: Array[Array[Int]],
  outDegrees: Array[Int],
  rank: Array[Float],
  dampening: Float = 0.85f): Array[Float] = {

  val spring = 1 - dampening
  val newRank = new Array[Float](transposedGraph.length)

  var i = 0
  while (i < transposedGraph.length) {
    var sum = 0.0f
    val links = transposedGraph(i)

    var j = 0
    while (j < links.length) {
      val current = links(j)
      sum += rank(current) / outDegrees(current)
      j += 1
```

```
19      }
20
21      newRank(i) = spring + dampening * sum
22
23      i += 1
24    }
25
26    newRank
27  }
```

Sukurta funkcija atlieka vieną pagerank iteraciją. Jai reikalingi parametrai yra grafo duomenys, ankstesnis PageRank įvertis, slopinimo koeficiento vertė. Skaičiavimas vykdomas naudojant biblioteka [12] nuskaitytą grafą. Grafas yra išsaugomas atmintyje naudojant skaičiavimams pritaikytą formatą. Grafas yra saugojamas priešingos krypties nei yra pateikiamas duomenų faile. Grafas yra saugomas kaip masyvas iš masyvų `transposedGraph`. Išorinio masyvo indeksai i atitinka grafo viršūnių indeksus. Vidiniai masyvai talpina viršūnes kurios turi briauną į i-tąją viršūnę. Taip pat reikalingi viršūnių išėjimo laipsniai. Jie naudojami dažnai, dėl to yra iš anksto suskaičiuojami ir saugomi kitame masyve.

Ši funkcija įgyvendinta naudojant procedūrinį programavimo stilių, kas nėra įprasta rašant išeities kodą Scala kalba. Šis stilius parinktas dėl geresnio vykdymo greičio, naudojant funkcinį stilių išeities kodą galima supaprastinti iki

```
1 transposedGraph.map(
2     (1 - dampening) + dampening * _.map(
3       link => rank(link) / outDegrees(link)
4     ).sum
5   )
```

. Tačiau tokios funkcijos vykdymo greitis yra apie 10 kartų lėtesnis. Todėl netinka didiesiems duomenims.

Duomenims saugoti naudojami masyvai, nes naudojamas duomenų kiekis yra tam tinkamas. Iteravimas per įprastinius masyvus yra vykdomas žymiai greičiau nei naudojant kitas duomenų struktūras dėl geros procesoriaus optimizacijos naudojant spartinančiąsias atmintines. Naudojant pasirinktą duomenų rinkinį naudojama apie 5GB darbinės atminties.

4.2. MODELIO MOKYMAS

Modelių mokymas buvo vykdomas programinės įrangos paketu Vowpal Wabbit [11].

4.2.1. Atraminių vektorių klasifikatoriaus mokymas

Naudojant atraminių vektorių klasifikatorių sistemoje Vowpal Wabbit naudojami parametrai nurodyti 4.1 lentelėje.

4.1 lentelė

Atraminių vektorių klasifikatoriaus hiperparametrai

Komandinės eilutės parametras	Galimos vertės	Parametro reikšmė
`passes`		Modelio mokymo iteracijų skaičius
`loss_function`		Mokymo metu naudojama nuostolio funkcija:
	hinge	nuostolio funkcija $l(y) = \max(0, 1-t*y)$ kur y yra įvertis o $t = \pm 1$ tikroji klasė
	logistic	logistinė nuostolio funkcija
`l2`		l2 parametrų reguliarizacija
`bit_precision`		Bitų, naudojamų savybių lentelei, skaičius
`kernel`		Naudojama branduolio funkcija:
	poly	polinominis branduolys
	linear	tiesinis branduolys
	rbf	radialinės bazės branduolys
`degree`		polinominio branduolio laipsnis

Mokant atraminių vektorių klasifikatorių stebimas nuostolių funkcijos įvertis. Pagal jį vertinamas modelio mokymo kokybė.

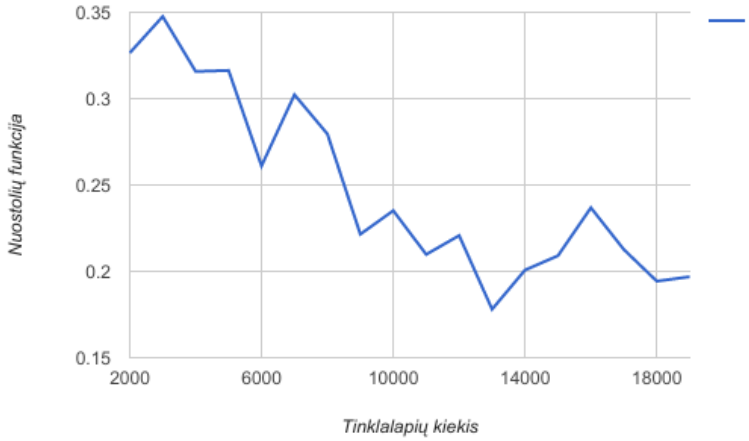
Modeliai pradėti mokyti nuo polinominio branduolio. Svarbūs parametrai derinant šį modelį yra polinominės funkcijos laipsnis bei parametrų reguliarizavimo koeficientas.

4.2 lentelė

Pirminio polinominio branduolio modelio parametrai

L2 reguliarizacija	10^{-7}	10^{-5}	0,1	10^{-7}	10^{-5}	0,1
Branduolio funkcijos laipsnis	5	5	5	3	3	3

Modelio mokymas toliau tyrinėtas naudojant kitas branduolio funkcijas ir derinant jų parametrus. Modelio parametrų reguliarizacijos parametras L2 nekeistas.



4.1 pav. Tiesinio branduolio modelio nuostolių ir tinklalapių skaičiaus naudoto mokymui priklausomybė

4.2.2. Logistinės regresijos modelio mokymas

Vowpal Wabbit sistema visiems modeliams naudoja vienodą duomenų formatą bei panašius parametrus. Tai leidžia lengvai bandyti įvairius modelius. Dėl to po prastų atraminių vektorių klasifikatoriaus rezultatų pasirinktas vienas iš paprasčiausių modelių, kuriuo tikėtina pasiekti gerus rezultatus, nenaudojant didelio resursų kiekio modelio formavimui.

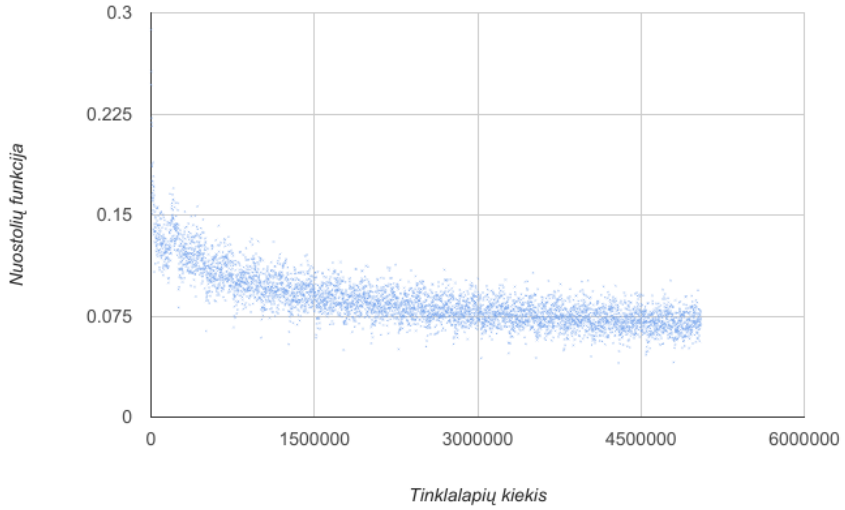
4.3 lentelė

Logistinės regresijos modelio hiperparametrai

Komandinės eilutės parametras	Galimos vertės	Parametro reikšmė
`passes`		Modelio mokymo iteracijų skaičius
`loss_function`		Mokymo metu naudojama nuostolio funkcija
	hinge	nuostolio funkcija $l(y) = \max(0, 1-t*y)$ kur y yra įvertis o $t = \pm 1$ tikroji klasė
	logistic	logistinė nuostolio funkcija
`l2`		l2 parametrų reguliarizacija
`bit_precision`		Bitų, naudojamų savybių lentelei, skaičius

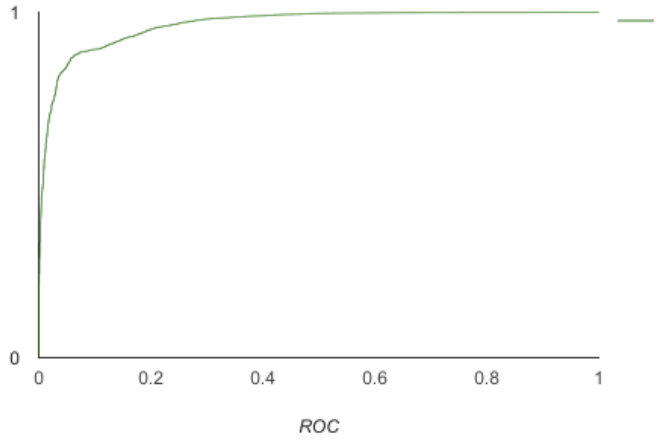
Mokant logistinės regresijos modelį naudojamas mažesnis savybių lentelės dydis nurodomas naudojant `bit_precision` parametrą. Tai leidžia optimaliau atlikti mokymą sumažinant naudojamą atminties kiekį. Naudojami 27 bitai.

Modelio mokymo nuostoliai pateikiami 4.2 pav.. Šiame grafike matoma, kad nuostolio funkcijos rezultatai yra žymiai geresni nei mokant atraminių vektorių klasifikatorius. Mokomas modelis konverguoja, pasiekiama gana pastovi nuostolio funkcijos reikšmė, artima nuliui.



4.2 pav. Logistinės regresijos modelio nuostolių priklausomybė nuo mokymui naudojamų tinklalapių skaičiaus

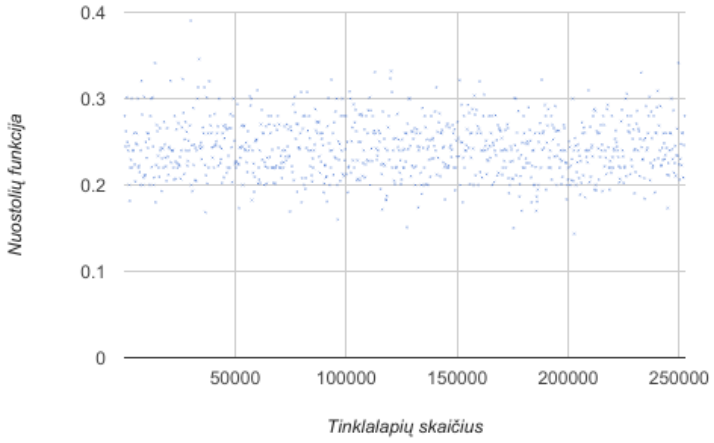
Gautas modelis itin gerai prognozuoja testavimo rinkinio duomenis. Suformuotas logistinės regresijos modelis naudoja 145832 matmenų.



4.3 pav. Logistinės regresijos modelio ROC kreivė

Lyginant šio modelio rezultatus pateiktus [8] straipsnyje šis logistinės regresijos modelis pralenkia atraminių vektorių klasifikatorių, kuris šiame straipsnyje buvo išskiriamas kaip geriausias klasifikatorius.

4.2.3. Sprendimų medžio modelio formavimas



4.4 pav. Sprendimų medžio modelio nuostolių priklausomybė nuo mokymui naudojamų tinklalapių skaičiaus

Sprendimų medžio modelio mokymo parametrai pateikiami lentelėje 4.4 .

4.4 lentelė

Sprendimų medžio modelio hiperparametrai

Komandinės eilutės parametras	Galimos vertės	Parametro reikšmė
`passes`		Modelio mokymo iteracijų skaičius
`loss_function`		Mokymo metu naudojama nuostolio funkcija
	hinge	nuostolio funkcija $l(y) = \max(0, 1-t*y)$ kur y yra įvertis o $t = \pm 1$ tikroji klasė
	logistic	logistinė nuostolio funkcija
`l2`		l2 parametrų reguliarizacija
`bit_precision`		Bitų, naudojamų savybių lentelei, skaičius
`log_multi`		Siekiamas sprendimų medžio komplekšiškumas laiko atžvilgiu, $O(log(n))$ kur n yra šio parametro vertė

Sprendimų medžio modelio mokymas vyksta sėkmingai, stebint nuostolio funkcijos vertes paveiksle 4.4 pav..

Nepaisant gero nuostolio funkcijos rezultato šis modelis persimokė su visomis naudotomis hiperparametrų vertėmis. Šis modelis visada prognozuoja tik vieną iš reikšmių, tai yra klasifikacija nevykdoma, modelis visada grąžina, kad analizuojamas vektorius priklauso klasei 1 (tinklalapis kenksmingas).

4.2.4. Modelių mokymo rezultatai

4.5 lentelė

Modelių kokybės įverčiai

Matas	Logistinės regresijos klasifikatorius
Tikslumas	0,97552
Jautrumas	0,92588
Specifiškumas	0,98325
AUC	0.95456

Kokybiško atraminių vektorių klasifikatoriaus bei sprendimų medžio apmokyti nepavyko. Tikėtina, kad tai lemia naudojamo duomenų tipo specifika. Visi klasifikavimo kokybės įverčiai pateikiami lentelėje 4.5 . Duomenys turi itin daug matmenų - kiekvienas naujas žodis, simbolių grupė yra koduojama kaip naujas matmuo. Tačiau suformuotas itin tikslus logistinės regresijos modelis. Lyginant logistinės regresijos modelio rezultatus su pateiktais įvairių modelių rezultatais straipsnyje [8], kurie pateikiami lentelėje 4.6 visą mokymą galima laikyti sėkmingu. Nors apmokytas tik vienas modelis iš trijų jis yra itin tikslus ir lenkia atraminių vektorių mašinų klasifikatorių, kuris buvo įvertintas kaip geriausias.

4.6 lentelė

Logistinės regresijos modelio kokybės metrikos lyginamos su [8]

	Logistinės regresijos modelis	K-artimiausių kaimynų modelis	Radialinės bazės funkcijos branduolio atraminių vektorių klasifikatorius	Tiesinio branduolio atraminių vektorių klasifikatorius
Tikslumas	0.97552	0.91	0.97	0.92

Šis modelis toliau naudojamas klasifikavimo įrankio kūrime.

4.3. MODELIO NAUDOJIMAS KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO PROGRAMOS KŪRIMUI

Apmokytas modelis yra paruoštas naudojimui, tačiau jis nėra itin patogus. Naudojamos programinės įrangos modelis gali būti panaudojamas per komandinės eilutės sąsają, pateikiant duomenis mokymo metu naudotu formatu. Modelio grąžinamas rezultatas taip pat nėra lengvai suvokiamas. Dėl to yra kuriama taikomoji programa kuri supaprastina modelio panaudojimą realiose, praktinėse situacijose.

Kuriama sistema vadinama toliau vadinama **kenksmingų interneto tinklalapių identifikavimo programa**.

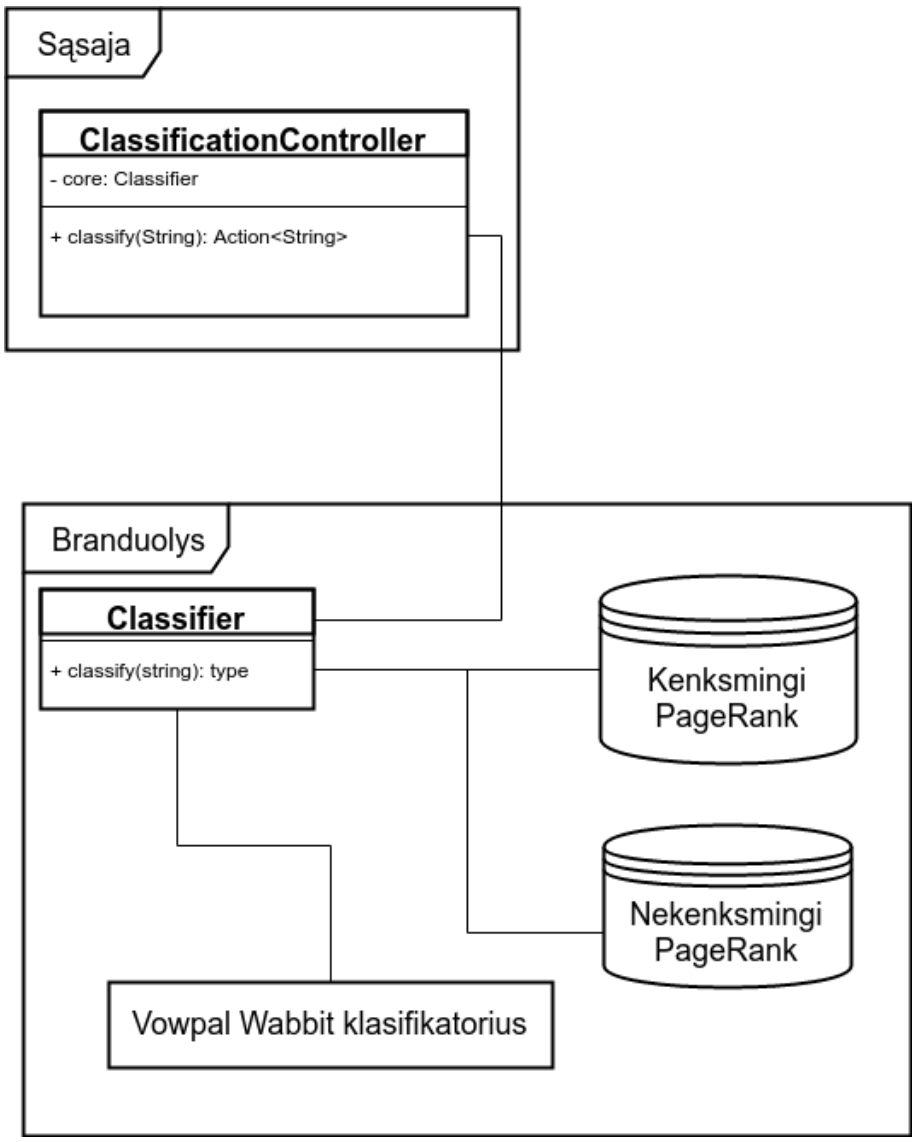
4.7 lentelė

Kenksmingų interneto tinklalapių identifikavimo programos sąsaja

Kelias	Metodas	Parametrai	Rezultatas
`/classify`	`GET`	`url` klasifikuojamo tinklalapio adresas	slankaus kablelio skaičius - klasifikavimo rezultatas

Kenksmingų interneto tinklalapių identifikavimo programa pritaikyta naudoti http protokolu, naudojama REST architektūra. Sąsaja pateikta 4.7 lentelėje. Tinklalpio kurį norima klasifikuoti adresas yra pateikiamas klasifikavimo programai, grąžinamas rezultatas yra klasifikavimo modelio pateiktas įvertis.

Kenksmingų interneto tinklalapių identifikavimo programos architektūra pateikiama 4.5 pav.



4.5 pav. Kenksmingų interneto tinklalapių identifikavimo programos architektūra

Kenksmingų interneto tinklalapių identifikavimo programa susideda iš dviejų komponentų:

a) sąsaja,
b) branduolys.

Sąsaja yra komponentas kurio funkcija yra pateikti branduolio suteikiamą funkcionalumą naudotojams. Šiuo atveju tai yra http protokolu veikianti sąsaja. Šis komponentas atsakingas už užklausos perdavimą branduolio komponentui, branduolio komponente esančio klasifikatoriaus rezultato pa-

teikimui naudotojui tinkamu formatu. Programinio įrankio išeities kodas pateikiamas prieduose (2 priedas).

Branduolio komponente vykdomas modelio pritaikymas. Jį sudaro klasė `Classifier-Controller` bei duomenų saugyklos kenksmingų ir nekenksmingų svetainių PageRank įverčiams.

Kenksmingų interneto tinklalapių identifikavimo programoje atskirti komponentai siekiant supaprastinti programos modifikacijas.. Atskiriant branduolio elementą kuriame yra vykdoma verslo logika, matematiniai skaičiavimai sistema yra paruošiama greitam plėtimui, lengvam migravimui tarp palaikomų ir naudojamų vartotojo sąsajos protokolų. Šiuo atveju vartotojo sąsaja nepriklauso nuo branduolio, tik nuo klasės `ClassifierController` implementacijos.

Lyginant su alternatyvia sistema [19] yra matomi kenksmingų interneto tinklalapių identifikavimo programos privalumai:

a) galima naudoti alternatyvi sąsaja, pakeitus sąsajos komponento implementaciją;
b) klasifikavimo modelis gali būti keičiamas;
c) nereikalauja interneto prieigos.

Pirminė programos versija yra kurta siekiant įrodyti jos veikimo koncepciją. Dėl to kai kurios sistemos vietos yra optimizuotos programavimo laiko minimizavimo atžvilgiu. Plėtojant programinę įrangą vertėtų programą skaidyti į kelis atskirtus komponentus, pagal naudojamų resursų tipą. PageRank įverčiai turi būti iškeliami į duomenų bazę, mašininio mokymo modelis iškeliamas į atskirą vykdomąją aplikaciją kuri gali būti plečiamas esant reikalui ar skaičiavimo resursų trūkumui.

4.4. PRAKTINIS KENKSMINGŲ INTERNETO TINKLALAPIŲ IDENTIFIKAVIMO PROGRAMOS TAIKYMAS

Sukurtas įrankis bandytas panaudoti naudojant kelias nekenksmingas interneto svetaines. Dalis iš bandytų tinklalapių pateikiama lentelėje 4.8 .

4.8 lentelė

Kenksmingų interneto tinklalapių identifikavimo programos bandymo rezultatai

Tinklalapis	Tinklalapio adresas	Kenksmingas
KTU titulinis tinklalapis	http://ktu.edu/	Ne
Wikipedia straipsnis apie dramblius	https://lt.wikipedia.org/wiki/Drambliai	Ne
Socialinis tinklas Facebook	https://facebook.com	Taip
Google paieškos variklis	https://google.com	Taip
15 min laikraštis	https://15min.lt	Taip

Toks kenksmingų interneto tinklalapių identifikavimo programos bandymas parodo trūkumų. Tam tikri tinklalapiai kaip Google, 15min kurie nėra kenksmingi, yra klasifikuojami klaidingai. Tačiau validavimo imties klasifikavimo kokybės metrikos rodo gerus rezultatus - **tikslumas 0.976, specifiškumas 0.982, jautrumas 0.932**.

Programos trūkumas yra tai, kad iš logistinės regresijos modelio negalima paprastai gauti priežasčių, dėl ko tinklalapis buvo klasifikuotas kaip kenksmingas ar nekenksmingas. Tačiau kenksmingų interneto tinklalapių identifikaivmo programa turi ir privalumų. Kenksmingų tinklalapių klasifikavimas vykdomas pagal mokymo duomenų rinkinį, tai leidžia pakeitus kenksmingų tinklalapių sąrašą identifikuoti kitos klasės tinklalapius, pavyzdžiui pagal jų tematiką. Taip pat sistemos veikimas yra aiškus - komercinės sistemos nepateikia tikslaus klasifikavimo metodo. Šią programą galima lengvai modifikuoti, ar keičiant sistemos sąsają ar klasifikavimo modelį. Taikant programą praktikoje būtina atsižvelgti į šiuos trūkumus bei privalumus.

IŠVADOS

1. Apžvelgus literatūrą parinkti klasifikavimo modeliai ir pasiūlytas PageRank įverčio naudojimas kenksmingų interneto tinklalapių identifikavimo modelyje;
2. Suformuotas modelis, kurio tikslumas 97,5 %;
3. Sukurta kenksmingų interneto tinklalapių identifikavimo programa, kuri gali būti lengvai modifikuojama, tiksliai identifikuoja kenksmingus tinklalapius, pagal validavimo imtį;
4. Ateityje siūloma tobulinti programą į klasifikuojamų tinklalapių požymius įtraukiant tinklalapių turinį bei serverių informaciją apibūdinančius parametrus.

Literatūra

1. HEARTFIELD, R.; LOUKAS, G. A taxonomy of attacks and a survey of defence mechanisms for semantic social engineering attacks. *ACM Computing Surveys (CSUR)*. 2016, t. 48, nr. 3, psl. 37.
2. LI, Z.; ARLWAIS, S.; XIE, Y.; YU, F.; WANG, X. Finding the Linchpins of the Dark Web: a Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. In: *IEEE Symposium on Security and Privacy, 2013*. IEEE, 2013.
3. STOKES, J. W.; ANDERSEN, R.; SEIFERT, C.; CHELLAPILLA, K. WebCop: Locating Neighborhoods of Malware on the Web. In: USENIX, 2010.
4. WU, X. ir kt. Top 10 algorithms in data mining. *Knowledge and Information Systems*. 2008, t. 14, nr. 1, psl. 1–37. ISSN 0219-3116.
5. PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. *The PageRank Citation Ranking: Bringing Order to the Web.* Stanford InfoLab, 1999. Technical Report. Stanford InfoLab.
6. MAŠETIC, Z.; SUBASI, A.; AZEMOVIC, J. Malicious Web Sites Detection using C4.5 Decision Tree. *Southeast Europe Journal of Soft Computing*. 2016.
7. QUINLAN, R. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.
8. KAZEMIAN, H.; AHMED, S. Comparisons of machine learning techniques for detecting malicious webpages. *Expert Systems with Applications*. 2015, t. 42, nr. 3, psl. 1166–1177. ISSN 0957-4174.
9. FUJIWARA, Y.; IRIE, G.; KITAHARA, T. Fast Algorithm for Affinity Propagation. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*. Barcelona, Catalonia, Spain: AAAI Press, 2011, psl. 2238–2243. IJCAI'11. ISBN 978-1-57735-515-1.
10. BORDES, A.; ERTEKIN, S.; WESTON, J.; BOTTOU, L. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*. 2005, t. 6, nr. Sep, psl. 1579–1619.
11. LANGFORD, J.; LI, L.; STREHL, A. *Vowpal Wabbit*. 2007.
12. MEUSEL, R.; LEHMBERG, O.; BIZER, C.; VIGNA, S. *Web Data Commons - Hyperlink Graphs*. 2017. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `http://webdatacommons.org/hyperlinkgraph/index.html`.

13. BOLDI, P.; VIGNA, S. The webgraph framework I: compression techniques. In: *Proceedings of the 13th international conference on World Wide Web*. 2004, psl. 595–602.

14. KORET, J. *Malware URLs*. 2014. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `http://malwareurls.joxeankoret.com/`.

15. WWW.MALWAREDOMAINLIST.COM. *Malware Domain List*. 2009. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `https://www.malwaredomainlist.com/`.

16. PHISHTANK. *PhisTank | Join the fight against phishing*. 2017. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `http://www.phishtank.com/`.

17. MALWAREDOMAINS@GMAIL.COM. *Malware prevention through DNS Redirection*. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `http://malware-domains.com/`.

18. TANGE, O. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*. 2011, t. 36, nr. 1, psl. 42–47.

19. GOOGLE. Google safe browsing apis. [interaktyvus], [žiūrėta 2017m. birželio 1d.]. Prieiga per: `https://developers.google.com/safe-browsing`.

PRIEDAI

1 priedas. MODELIO MOKYMO DUOMENŲ PARUOŠIMO PROGRAMOS IŠEITIES KODAS

```
package andriusdap.orbweaver.executor

import scala.language.postfixOps

object UrlBuilder {

  private def dropPrefix(s: String): String = {
    val banned =
      Vector("https://www.", "http://www.", "https://", "http://", "www.")

    banned
      .find(s startsWith)
      .map(prefix => s.substring(prefix.length))
      .getOrElse(s)
  }

  case class URL(host: Vector[String],
                 query: Vector[String],
                 dots: Int,
                 specialSymbols: Int)

  private def split(s: String): URL = {
    val separators = Array('-', '.', '_', '~', ':', '/', '?', '#', '[', ']',
      '@', '!', '$', '&', ''', '(', ')', '*', '+', ',', ';', '=', ' ')
    val slashed = s.toLowerCase.split("/")
    val host = slashed.headOption
      .map(_.split(separators).toVector)
      .getOrElse(Vector.empty)
    val path = slashed.tail.flatMap(param => param.split(separators)).toVector

    val dotCount = s.count(_ == '.')
    URL(
      host,
      path,
      dotCount,
      s.length - (host.map(_.length).sum + path.map(_.length).sum) - dotCount)
  }

  def build(s: String): URL =
    split(dropPrefix(s))
}
```

```
package andriusdap.orbweaver.executor

import java.lang.Long
import java.util.function.Consumer

import it.unimi.dsi.big.webgraph.BVGraph

import scala.collection.mutable.ArrayBuffer
```

```
9  import scala.io.{Codec, Source}
10 import scala.reflect.io.File
11
12 class Pagerank(webgraph: String) {
13
14   val graph = BVGraph.load(webgraph)
15   val transposedGraph = transpose(graph)
16   val outDegree = outDegrees(graph)
17
18   def seed(length: Int, seedFile: File, seedValue: Float): Array[Float] = {
19     val values = Array.fill[Float](length)(0.0f)
20     Source
21       .fromFile(seedFile.jfile)
22       .getLines()
23       .map(_.toInt)
24       .foreach(values(_) = seedValue)
25     values
26   }
27
28   def output(pagerank: Array[Float], s: String): Unit = {
29     File(s).writeAll(pagerank.zipWithIndex.map {
30       case (rank, index) => s"$index $rank\n"
31     }: _*)
32   }
33
34   case class PagerankResult(pagerank: Array[Float], convergence: Seq[Double])
35
36   def pagerank(seedFile: File,
37                passes: Int,
38                dampening: Float,
39                error: Double,
40                seedValue: Float): PagerankResult = {
41     val seedRank = seed(transposedGraph.length, seedFile, seedValue)
42
43     val (convergence, pagerank) =
44       (1 to passes).foldLeft(Seq[Double]() -> seedRank) {
45         case ((diffs, currentRank), pass) =>
46           if (!diffs.lastOption.exists(_ < error)) {
47             val newValue =
48               singlePass(transposedGraph, outDegree, currentRank, dampening)
49             (diffs :+ distance(currentRank, newValue)) -> newValue
50           } else {
51             (diffs, currentRank)
52           }
53       }
54     PagerankResult(pagerank, convergence)
55   }
56
57   def distance(current: Array[Float], previous: Array[Float]): Double = {
58     var i = 0
59     var sum = 0.0d
60     while (i < current.length) {
61       val diff = current(i) - previous(i)
62       sum += diff * diff
63       i += 1
64     }
65
66     Math.sqrt(sum)
67   }
68
```

```
69  def singlePass(transposedGraph: Array[Array[Int]],
70                 outDegrees: Array[Int],
71                 rank: Array[Float],
72                 dampening: Float = 0.85f): Array[Float] = {
73
74    val spring = 1 - dampening
75    val newRank = new Array[Float](transposedGraph.length)
76
77    var i = 0
78    while (i < transposedGraph.length) {
79      var sum = 0.0f
80      val links = transposedGraph(i)
81
82      var j = 0
83      while (j < links.length) {
84        val current = links(j)
85        sum += rank(current) / outDegrees(current)
86        j += 1
87      }
88
89      newRank(i) = spring + dampening * sum
90
91      i += 1
92    }
93
94    newRank
95  }
96
97  def transpose(graph: BVGraph): Array[Array[Int]] = {
98    require(
99      graph.numNodes() < Int.MaxValue,
100     s"Cannot handle more than maxint nodes, received ${graph.numNodes()}")
101
102   val transposedGraph =
103     Array.fill(graph.numNodes().toInt)(new ArrayBuffer[Int]())
104
105   graph
106     .nodeIterator(0)
107     .forEachRemaining(
108       new Consumer[Long] {
109         override def accept(t: Long): Unit = {
110           val successors = graph.successorBigArray(t)
111
112           var i = 0
113           while (i < successors.length) {
114             var j = 0
115
116             val asInt = t.toInt
117             while (j < successors(i).length) {
118               val current = successors(i)(j).toInt
119               transposedGraph(current).append(asInt)
120
121               j += 1
122             }
123             i += 1
124           }
125         }
126       }
127     )
128   transposedGraph.map(_.toArray)
```

```
129   }
130
131   def outDegrees(graph: BVGraph): Array[Int] = {
132     val outDegrees = new Array[Int](graph.numNodes().toInt)
133     graph
134       .nodeIterator(0)
135       .forEachRemaining(
136         new Consumer[Long] {
137           override def accept(t: Long): Unit = {
138             outDegrees(t.toInt) = graph.outdegree(t).toInt
139           }
140         }
141       )
142
143     outDegrees
144   }
145 }
```

```
1  package andriusdap.orbweaver.executor
2
3  import scala.io.{Codec, Source}
4  import scala.reflect.io.File
5  import scala.util.Random
6
7  object App {
8    Random.setSeed(5l)
9
10   val dir = "/media/ssd/final_data/"
11
12   val tmpDir = "/media/ssd/temp_dir/"
13
14   val hostgraph = dir + "hostgraph"
15   val blacklist = dir + "blacklist.txt"
16   val whitelist = dir + "decoded_whitelist.txt"
17   val hostIndex = dir + "index.host"
18
19   def l[T](message: String, f: () => T) = {
20     val before = System.currentTimeMillis()
21
22     print(s"$message ... ")
23
24     val result = f.apply()
25     val after = System.currentTimeMillis()
26     println(f" ... ${(after - before) / 1000.0f}%2.2f s")
27
28     result
29   }
30
31   sealed trait Label {
32     def print = {
33       this match {
34         case Benign => "Benign"
35         case Malicious => "Malicious"
36       }
37     }
38   }
39
40   object Label {
41     def fromString: String => Label = {
```

```
42        case "Benign" => Benign
43        case "Malicious" => Malicious
44        case _ => ???
45      }
46    }
47
48    case object Malicious extends Label
49
50    case object Benign extends Label
51
52    case class Feature(url: String, label: Label)
53
54    object Feature {
55      def parse(line: String): Feature = {
56        val fields = line.split(" ")
57        Feature(fields.dropRight(1).mkString(" "), Label.fromString(fields.last))
58      }
59    }
60
61    def dumpLabelFiles(test: File,
62                       train: File,
63                       validate: File): (File, File, File) = {
64      val testLabelFile = File(tmpDir + "test_labels.txt")
65      val trainLabelFile = File(tmpDir + "train_labels.txt")
66      val validateLabelFile = File(tmpDir + "validate_labels.txt")
67      if (Seq(testLabelFile, trainLabelFile, validateLabelFile).exists(
68            !_.exists)) {
69        Seq(
70          testLabelFile -> test,
71          trainLabelFile -> train,
72          validateLabelFile -> validate
73        ).foreach {
74          case (labels, source) =>
75            val writer = labels.bufferedWriter()
76            Source
77              .fromFile(source.jfile)
78              .getLines()
79              .map(l => Feature.parse(l).label)
80              .map(toWabbitValue)
81              .foreach(l => writer.write(s"$l\n"))
82
83            writer.close()
84        }
85      }
86
87      (trainLabelFile, testLabelFile, validateLabelFile)
88    }
89
90    def main(args: Array[String]): Unit = {
91      val (test, train, validate) =
92        l("split files", () => splitFile(blacklist, whitelist))
93      val (malicious, benign) = l("building seeds", () => getSeeds(train))
94      val (maliciousPagerankFile,
95           benignPagerankFile,
96           maliciousPagerankConvergenceFile,
97           benignPagerankConvergenceFile) =
98        l("building pageranks", () => buildPageranks(malicious, benign))
99      val (trainingFile, testingFile, validationFile) = l(
100       "dump wabbit files",
101       () =>
```

```
102         dumpWabbitFiles(test,
103                         train,
104                         validate,
105                         maliciousPagerankFile,
106                         benignPagerankFile))
107       val (trainingLabels, testingLabels, validationLabels) =
108         l("dump label files", () => dumpLabelFiles(test, train, validate))
109     }
110
111     def dumpWabbitFiles(test: File,
112                         train: File,
113                         validate: File,
114                         maliciousPagerankFile: File,
115                         benignPagerankFile: File): (File, File, File) = {
116       val trainingFile = File(tmpDir + "training_file.vw")
117       val testingFile = File(tmpDir + "testing_file.vw")
118       val validationFile = File(tmpDir + "validation_file.vw")
119
120       if (Seq(trainingFile, testingFile, validationFile).exists(!_.exists)) {
121         val maliciousRanks = readRankFile(maliciousPagerankFile)
122         val benignRanks = readRankFile(benignPagerankFile)
123         Seq(
124           train -> trainingFile,
125           test -> testingFile,
126           validate -> validationFile
127         ).par.foreach {
128           case (from, to) =>
129             val features = extractFeatures(from, maliciousRanks, benignRanks)
130             dump(to, features)
131         }
132       }
133       (trainingFile, testingFile, validationFile)
134     }
135
136     def dump(file: File, set: Iterator[FeatureSet]): Unit = {
137       val buffer = file.bufferedWriter()
138       set
139         .map {
140           case FeatureSet(host,
141                           query,
142                           dots,
143                           spec,
144                           maliciousRank,
145                           benignRank,
146                           label) =>
147             val result = toWabbitValue(label)
148             s"$result " +
149               s"|path ${host.mkString(" ")}" +
150               s" |query ${query.mkString(" ")}" +
151               s" |numerical dots:$dots.0 specialSymbols:$spec.0" +
152               s" |ranks benign:$benignRank malicious:$maliciousRank" +
153               "\n"
154         }
155         .foreach(buffer.write)
156       buffer.close()
157     }
158
159     def toWabbitValue(label: Label): Int = {
160       label match {
161         case Malicious => 1
```

```
162       case Benign => -1
163     }
164   }
165
166   def extractFeatures(
167       train: File,
168       maliciousRanks: Map[String, Float],
169       benignRanks: Map[String, Float]): Iterator[FeatureSet] = {
170     val minMalicious = maliciousRanks.values.min
171     val minBenign = benignRanks.values.min
172
173     Source.fromFile(train.jfile).getLines().map(Feature.parse).map { f =>
174       val host = strip(f.url)
175       val url = UrlBuilder.build(f.url)
176       FeatureSet(
177         url.host,
178         url.query,
179         url.dots,
180         url.specialSymbols,
181         maliciousRanks.getOrElse(host, minMalicious),
182         benignRanks.getOrElse(host, minBenign),
183         f.label
184       )
185     }
186   }
187
188   case class FeatureSet(host: Vector[String],
189                         query: Vector[String],
190                         dots: Int,
191                         specialSymbols: Int,
192                         maliciousRank: Float,
193                         benignRank: Float,
194                         flag: Label)
195
196   def readRankFile(benignPagerankFile: File): Map[String, Float] = {
197     Source
198       .fromFile(benignPagerankFile.jfile)
199       .getLines()
200       .map { s =>
201         val split = s.split(" ")
202         split(0) -> split(1).toFloat
203       }
204       .toMap
205   }
206
207   def buildPageranks(malicious: File, benign: File): (File, File, File, File) = {
208     val maliciousPagerankFile = File(tmpDir + "malicious_pagerank.txt")
209     val benignPagerankFile = File(tmpDir + "benign_pagerank.txt")
210
211     val maliciousPagerankConvergenceFile = File(
212       tmpDir + "malicious_pagerank_convergence.txt")
213     val benignPagerankConvergenceFile = File(
214       tmpDir + "benign_pagerank_convergence.txt")
215
216     if (Seq(maliciousPagerankFile,
217             benignPagerankFile,
218             maliciousPagerankConvergenceFile,
219             benignPagerankConvergenceFile).exists(!_.exists)) {
220
221       val maliciousPagerankFileBuffer = maliciousPagerankFile.bufferedWriter()
```

```
222       val benignPagerankFileBuffer = benignPagerankFile.bufferedWriter()
223       val maliciousPagerankConvergenceFileBuffer =
224         maliciousPagerankConvergenceFile.bufferedWriter()
225       val benignPagerankConvergenceFileBuffer =
226         benignPagerankConvergenceFile.bufferedWriter()
227
228       val pagerankAlgo =
229         l("initializing pagerank", () => new Pagerank(hostgraph))
230       val maliciousPagerank = l(
231         s"malicious pagerank",
232         () => pagerankAlgo.pagerank(malicious, 20, 0.95f, 0.1, 100))
233       val benignPagerank = l(
234         s"benign pagerank",
235         () => pagerankAlgo.pagerank(benign, 50, 0.75f, 0.1, 1))
236
237       val hosts = Source
238         .fromFile(hostIndex)
239         .getLines()
240         .map { line =>
241           line.split("\t")(0)
242         }
243         .toSeq
244
245       maliciousPagerank.pagerank.zip(hosts).foreach {
246         case (rank, host) =>
247           maliciousPagerankFileBuffer.write(s"$host $rank\n")
248       }
249
250       benignPagerank.pagerank.zip(hosts).foreach {
251         case (rank, host) =>
252           benignPagerankFileBuffer.write(s"$host $rank\n")
253       }
254
255       maliciousPagerank.convergence.foreach(
256         convergence =>
257           maliciousPagerankConvergenceFileBuffer.write(s"$convergence\n")
258       )
259
260       benignPagerank.convergence.foreach(
261         convergence =>
262           benignPagerankConvergenceFileBuffer.write(s"$convergence\n")
263       )
264
265       maliciousPagerankFileBuffer.close()
266       benignPagerankFileBuffer.close()
267       maliciousPagerankConvergenceFileBuffer.close()
268       benignPagerankConvergenceFileBuffer.close()
269     }
270     (maliciousPagerankFile,
271      benignPagerankFile,
272      maliciousPagerankConvergenceFile,
273      benignPagerankConvergenceFile)
274   }
275
276   def getSeeds(train: File): (File, File) = {
277     val malicious = File(tmpDir + "maliciousSeed.txt")
278     val benign = File(tmpDir + "benignSeed.txt")
279     if (Seq(malicious, benign).exists(!_.exists)) {
280       val hosts = Source
281         .fromFile(hostIndex)
```

```
282         .getLines()
283         .map { line =>
284           val pair = line.split("\t")
285           (pair(0), pair(1).toInt)
286         }
287         .toMap
288
289       val trainItem = train.lines().map(Feature.parse).map {
290         case Feature(url, label) =>
291           Feature(strip(url), label)
292       }
293       val maliciousBuffer = malicious.bufferedWriter()
294       val benignBuffer = benign.bufferedWriter()
295
296       trainItem
297         .flatMap { item =>
298           hosts.get(item.url).map(_ -> item.label)
299         }
300         .foreach {
301           case (id, label) =>
302             val file = label match {
303               case Benign => benignBuffer
304               case Malicious => maliciousBuffer
305             }
306
307             file.write(s"$id\n")
308         }
309       benignBuffer.close()
310       maliciousBuffer.close()
311     }
312
313     (malicious, benign)
314   }
315
316   def strip(url: String): String = {
317     val prefixes = Seq(
318       "https://www.",
319       "http://www.",
320       "https://",
321       "http://"
322     )
323
324     val prefixLength = prefixes.find(url.startsWith).map(_.length).getOrElse(0)
325     val withoutPrefix = url.drop(prefixLength)
326
327     val suffixFrom = withoutPrefix.indexOf("/")
328
329     val clean = if (suffixFrom == -1) {
330       withoutPrefix
331     } else {
332       withoutPrefix.slice(0, suffixFrom)
333     }
334     clean
335   }
336
337   def splitFile(blacklist: String, whitelist: String): (File, File, File) = {
338     val test = File(tmpDir + "test.txt")
339     val train = File(tmpDir + "train.txt")
340     val validate = File(tmpDir + "validate.txt")
341
```

```
342     if (Seq(train, test, validate).exists(!_.exists)) {
343       val blacklistLines = Source
344         .fromFile(blacklist)
345         .getLines()
346         .map(s => Feature(s, Malicious))
347         .toVector
348       val whitelistLines = Source
349         .fromFile(whitelist, "ISO-8859-1")
350         .getLines()
351         .map(s => Feature(s, Benign))
352         .toVector
353
354       val testBuffer = test.bufferedWriter()
355       val trainBuffer = train.bufferedWriter()
356       val validateBuffer = validate.bufferedWriter()
357
358       Random.shuffle(blacklistLines ++ whitelistLines).foreach { line =>
359         val target = Random.nextFloat() match {
360           case a if a < 0.7 => trainBuffer
361           case a if a < (0.7 + 0.25) => testBuffer
362           case a => validateBuffer
363         }
364
365         target.write(s"${line.url} ${line.label.print}\n")
366       }
367
368       testBuffer.close()
369       trainBuffer.close()
370       validateBuffer.close()
371     }
372
373     (test, train, validate)
374   }
375 }
```

2 priedas. MODELĮ NAUDOJANČIO KLASIFIKAVIMO PROGRAMINIO ĮRANKIO IŠEITIES KODAS

```
1  package andriusdap.orbweaver.controllers
2
3  import andriusdap.orbweaver.core.Classifier
4  import com.google.inject.Singleton
5  import play.api.mvc._
6
7  import scala.language.postfixOps
8
9  @Singleton
10 class ClassifierController {
11   def classify(source: String) = Action {
12     request =>
13       Results.Ok(Classifier.classify(source))
14   }
15 }
```

```
1  package andriusdap.orbweaver.core
2
```

```
import andriusdap.orbweaver.executor.App.FeatureSet
import andriusdap.orbweaver.executor.{App, UrlBuilder}

import scala.io.Source
import scala.language.postfixOps
import scala.sys.process._

object Classifier {
  lazy val maliciousRanks = loadPageranks(
    "/media/ssd/temp_dir/malicious_pagerank.txt")

  lazy val benignRanks = loadPageranks(
    "/media/ssd/temp_dir/benign_pagerank.txt")

  val model = "/media/ssd/vw/model.vw"

  lazy val minMalicious = maliciousRanks.values.min
  lazy val minBenign = benignRanks.values.min

  def loadPageranks(source: String): Map[String, Float] = {
    Source.fromFile(source).getLines().map{c =>
      val s = c.split(" ")
      s(0) -> s(1).toFloat
    }.toMap
  }

  def classify(source: String): String = {
    val host = App.strip(source)
    val url = UrlBuilder.build(source)
    val malicious = maliciousRanks.getOrElse(host, minMalicious)
    val benign = benignRanks.getOrElse(host, minBenign)

    val sample = App.toSample(FeatureSet(
      url.host,
      url.query,
      url.dots,
      url.specialSymbols,
      malicious,
      benign,
      App.Benign
    ))

    val result = Process("echo", Seq(sample)) #| Process(
      "/media/ssd/vw2/test.sh"
    ) lineStream_!

    val head = result.head
    head
  }
}
```

```
#!/usr/bin/env bash
vw -t -i /media/ssd/vw2/best_model.vw -p /dev/stdout --quiet
```