VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Automatinis užduočių apimties vertinimas naudojant natūralios kalbos apdorojimo įrankius

Automatic task estimation using natural language processing

Bakalauro baigiamasis darbas

Atliko:	Jonas Motiejauskas
Darbo vadovas:	Vidas Vasiliauskas
Darbo recenzentas:	Partn. Prof., Dr. Vytautas Ašeris

Vilnius - 2020

Santrauka lietuvių kalba

Šiame darbe nagrinėjamos natūralios kalbos apdorojimo įrankių galimybės pakankamai tiksliai, t.y, bent 80% tikslumu nustatyti darbo vienetų (užduočių) apimtį. Tam įsitikinti atliekamas tyrimas, kuriame lyginamos tradicinė, paremta perceptronais, ir naujesnės, paremtos transformatoriais, mašininio mokymosi architektūros. Jo metu šiose architektūrose sumodeliuojama užduoties vertinimo problema ir tiriama, kuria iš jų remiantis gaunami tiksliausi spėjimo rezultatai. Užduočių apimties nustatymas nėra standartinė natūralios kalbos įrankiais sprendžiama problema, tokia kaip nuotaikos ar semantinė analizė. Tačiau ji gali būti modeliuojama panašiai, kaip kelių klasių klasifikacija, pradiniais duomenimis laikant žodžių seką - užduoties aprašą.

Raktiniai žodžiai: Užduočių vertės vertinimas, BERT, XLNet, Natūralios kalbos apdorojimas

Santrauka anglų kalba (*Summary*)

This dissertation explores the possibility of natural language processing tools solving the task effort estimation problem as accurately as 80%. Research is made to justify this claim, where a classic perceptron based machine learning architecture is compared against newer, transformer-based architectures. In this research, the task effort estimation problem is modeled on each of the selected architectures to check which of them are the most accurate. Unlike sentiment or semantic analysis, task effort estimation is not a standard problem for natural language processing tools. However, it can be modeled similarly as a multi-class classification problem which takes a sequence of words, task description, as input data.

Keywords: Task effort estimation, BERT, XLNet, Natural language processing

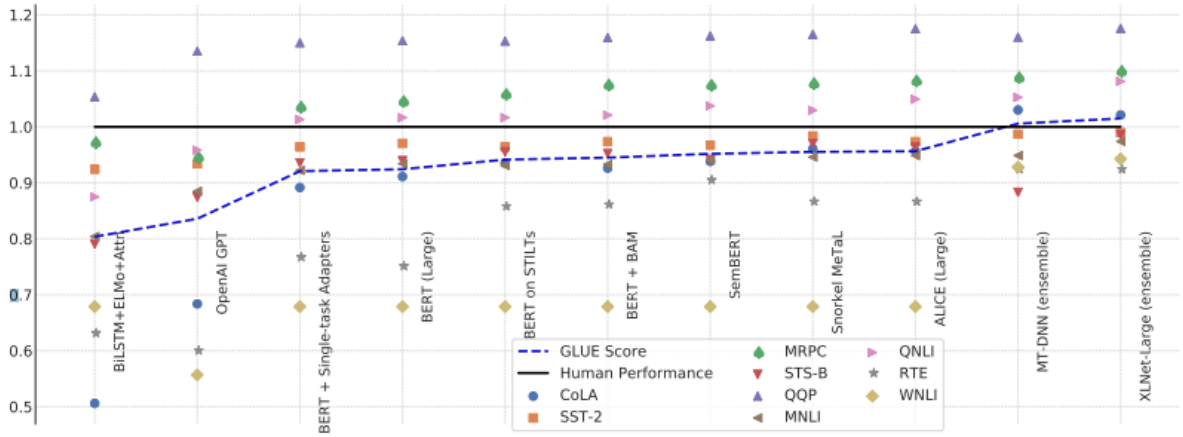
TURINYS

Įvadas

Veiklos trukmės įvertinimas yra projekto laiko valdymo dalis, kurioje bandoma kuo tiksliau nustatyti užduočių trukmė pasirinktais laiko apimties vienetais. Užduotys yra darbo vienetai, reikalingi įgyvendinti projektą ar jo dalį. Laiko apimties vienetai parenkami projekto vadovo. Tradicinis pasirinkimas yra laiko skalės intervalai, tačiau praktikoje naudojama ir natūraliųjų skaičių skalė, „Fibonači“ skalė, marškinėlių dydžiai ir net šunų veislės [ZZ12]. Dėl didelio projektų vėlavimų kiekio, kylančio iš skirtingo vykdytojų efektyvumo, laiko apimties skalės supratimo, emocinės vertintojų būsenos ir kitų nenumatytų veiksnių [Lau20], veiklos trukmės vertinimo tikslumui gerinti taikoma viena ar kelios strategijos. „Project Management Body of Knowledge“ išskiria šešias: ekspertų vertinimas, analogais grįstas vertinimas, parametrinis vertinimas, trijų taškų vertinimas, grupinės sprendimų priėmimo technikos ir papildomas laikas nenumatytiems atvejams [PMI13].

Tikslesnio vertinimo siekimas bei veiklų skaitmenizacija paskatino šio proceso automatizavimo bandymus [BOM07; CDT+18; IDC17]. Bandymai sukonstruoja veiklos trukmės įvertinimo algoritmą, kuris remiasi analogais grįsto užduočių vertinimo strategija. Uždavinys formuluojamas kaip daugiau nei dviejų klasių klasifikacijos problema, pateikiant žodžių seką - uždavinio aprašymą. Alternatyvus būdas formuluoti uždavinį - spręsti regresijos problemą, kai iš aprašymo, modelis nustato tikslų laiko vienetų skaičių. Abiem atvejais modelio apmokymas yra simetriškas validavimui, todėl problemai spręsti tinka didžioji dalis neuronų tinklų tipų. Tai įrodo ir teksto klasifikacijos uždavinio sprendimo paremto giliuoju mašininiu mokymusi būdų apžvalgoje pateikti rezultatai [MKC+20]. Atliktuose bandymuose tikslumą bandoma išgauti giliųjų neuronų tinklų pagalba, juos specializuojant ar pritaikant naujai atrastų architektūrų tipus. Vis dėl to, juose nagrinėjami tinklai - paprastasis (anglų k. FEEDFORWARD) ir pasikartojantis (anglų k. RECURRENT) - industrijoje laikomi pasenusiais, o apžvalgoje didžiausius pasiekimus rodo naujieji - transformatorių technologiją naudojantys tinklai.

Atsiradus transformatoriams [VSP+17] buvo pastebėta, jog natūralios kalbos modeliams apibrėžti tinka neprižiūrimai (anglų k. UNSUPERVISED) iš anksto apmokyti šia technologija pagrįsti neuronų tinklai [RWC+19]. Naujo architektūros tipo potencialas paskatino naujų architektūrų kūrimąsi - „OpenAIGPT“, „OpenAIGPT2“, „BERT“, „XLNet“ ir iš jų išvestomis. Visos jos siekė pagerinti tuometinį neuronų tinklų tikslumo rekordą suprantant natūralią kalbą. Tam nustatyti naudojamas „GLUE“ etalonas. Būtent transformatoriais paremtų architektūrų modeliai „SuperGLUE“ atliktame tyrime pagal „GLUE“ užima aukščiausias vietas [WPN+19].



1 pav. tinklų efektyvumas pagal „GLUE“ etaloną [WPN+19]

Atsižvelgiant į anksčiau atliktus panašius tyrimus bei darant prielaidą, remiantis jų gautomis išvadomis, kad sudėtingesni modeliai tiksliau modeliuoja veiklos trukmės vertinimo problemą, galima teigti, jog sudėtingesnės, transformatorių technologija paremtos architektūros, sugebės pralenkti ankstesniuose tyrimuose tirtus jų pirmtakus.

Remiantis šia prielaida formuojamas darbo tikslas ir uždaviniai:

1. Palyginti, kiek šioje situacijoje „BERT“ ir „XLNet“ architektūromis paremti modeliai yra tikslesni už tradicinį, „Averaged Perceptron“ architektūra paremtą modelį. Tradicinio perceptrono architektūra pasirinkta kaip paprastos architektūros etalonas.

2. Patikrinti, kaip keičiasi tikslumo rezultatai, kečiant veiklos trukmės klasių skaičių.

3. Patikrinti, kiek skiriasi tikslumo rezultatai, turint balansuotą ir atsitiktinai pasirinktų klasių rėžių duomenų modelį.

4. Patikrinti, kokią įtaką tikslumo rezultatams daro duomenų rinkinyje pasikartojančių eilučių pašalinimas.

5. Įsitikinti, kad naujosios natūralios kalbos apdorojimo priemonės gali bent 80% tikslumu įvertinti užduoties apimtį. Šis skaičius pasirinktas imant tyrimų [CDT+18; IDC17] 3 aukščiausių tikslumo įverčių vidurkį.

Tyrime naudojamas duomenų rinkinys transformuojamas pasitelkiant C# programavimo kalbą. „Averaged Perceptron“ tinklas kuriamas ML.NET sistemos pagalba, modelį apmokant vartotojo lygio grafikos procesoriumi „NVidia GeForce GTX 1050m Ti 4gb“. Transformatorių architektūra grįsti modeliai konstruojami Python programavimo kalba, naudojant „PyTorch“ karkasą. Jie apmokomi pasitelkiant „Google Colab“ sistemoje atsitiktinai priskirtus tenzorinius procesorius.

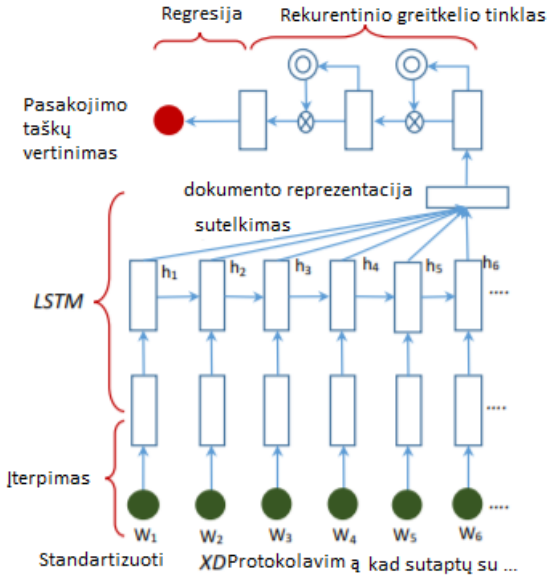
1. Panašių tyrimų apžvalga

1.1. Giliojo mokymosi modelis pasakojimo vienetų vertinimui

[CDT+18] autorių siūlomas modelis skirtas įvertinti vartotojo pasakojimus pasakojimo vienetais. Šis matavimo vienetas vertina užduoties sudėtingumą, o ne tai, kiek laiko užtruks ją padaryti.

1.1.1. Architektūra

Tyrėjai savo darbe pasitelkė du architektūros modelius: „Long Short-Term Memory“ (LSTM) ir „Recurrent Highway Network“ (RHN). Tiek LSTM tiek RHN yra pasikartojantys neuroninių tinklų architektūros modeliai [HS97; PTP+16]. Šie modeliai buvo apjungti sukuriant naują giliojo mokymosi modelį, kuris buvo pavadintas „Long-Deep Recurrent Neural Net“ (LD-RNN).



2 pav. LD-RNN modelis [CDT+18]

1.1.2. Duomenys

Sukurto modelio įvestis yra užduoties pavadinimas ir aprašymas. Rezultatas - pasakojimo vienetai. Tyrimo autoriai surinko duomenų rinkinį tinklo apmokymui, kuris sudarytas iš 23313 užduočių iš 16 atviro kodo projektų. Šis rinkinys buvo panaudotas apmokant modelį bei tikrinant jo patikimumą. 60% duomenų buvo skirti apmokymui, 20% kūrimui/validavimui, 20% testavimui. Duomenys buvo surikiuoti pagal jų sukūrimo laiką.

1.1.3. Rezultatai

Autorių atlikto empirinio tyrimo metu modelis užduotis vertino patikimiau negu pasirinkti baziniai etalonai: atsitiktinis spėjimas, vienetų mediana, vienetų vidurkis. Taip pat autoriai palygino modelį su dviem alternatyviais modeliais, kuriuos jie nagrinėjo:

- LSTM ir „Random Forests“ (RF)
- „Bag of words“ (BoF) ir RF

Jų teigimu, RHN yra efektyvus kuriant išsamią užduočių reprezentaciją, o LSTM efektyvus automatiniam užduoties aprašymo semantinių bruožų mokymuisi. Tai lemia pasakojimo vienetų vertinimo tikslumą.

1.2. Natūralios kalbos apdorojimo ir mašininio mokymosi metodai programų sistemų kūrimo pastangų vertinimui

[IDC17] autoriai siūlo modelį pagal užduoties aprašymą įvertinantį užduoties atlikimo trukmę. Darbe gauto modelio patikimumas lyginimas su klasikiniais parametriniais modeliais, tokiais kaip COCOMO ir Putnam. Šitie modeliai yra funkcijos su baigtiniu parametrų skaičiumi.

1.2.1. Architektūra

Tyrimo autoriai savo darbe naudoja „Term Frequency-Inverse Document Frequency“ (TF-IDF) ir „Distributed Representations of Documents“ (doc2vec) teksto vektorizavimo metodus, taip pat „Gaussian Naive Bayes“ (GNB) ir „Support vector regression“ (SVR) klasifikatorius. Jie kombinuoja vektorizavimo metodus su klasifikatoriais bandydami išgauti geriausią rezultatą.

1.2.2. Duomenys

Tyrime nagrinėjamo modelio išeitis yra įvertinimas laiku, o įvestis yra tekstinė užduoties reprezentacija:

- Interfeisas - komponentas, kuriame atliekama užduotis (duomenų bazės lentelė, klasė, išeities kodo failas ir t.t.);
- Kompleksiškumas - įvertintas projekto vadovo;
- Esybių kiekis - kiek esybių bus pakeista vykdant užduotį;

- Užduoties trukmės įvertinimas - grubus programuotojo vertinimas kiek užtruks įvykdyti užduotį minutėmis;
- Funkcionalumas - trumpas norimo funkcionalumo aprašymas.

Taip pat kiekviena užduotis turi veiksmus, kuriuos reikia atlikti. Veiksmas susideda iš:

- Aprašymas - trumpas veiksmo aprašymas;
- Veiksmo tipas - „Sukūrimas“ arba „Pakeitimas“.

Duomenis tyrimui pateikė įmonė, vykdanti programų sistemų kūrimą bei palaikymą.

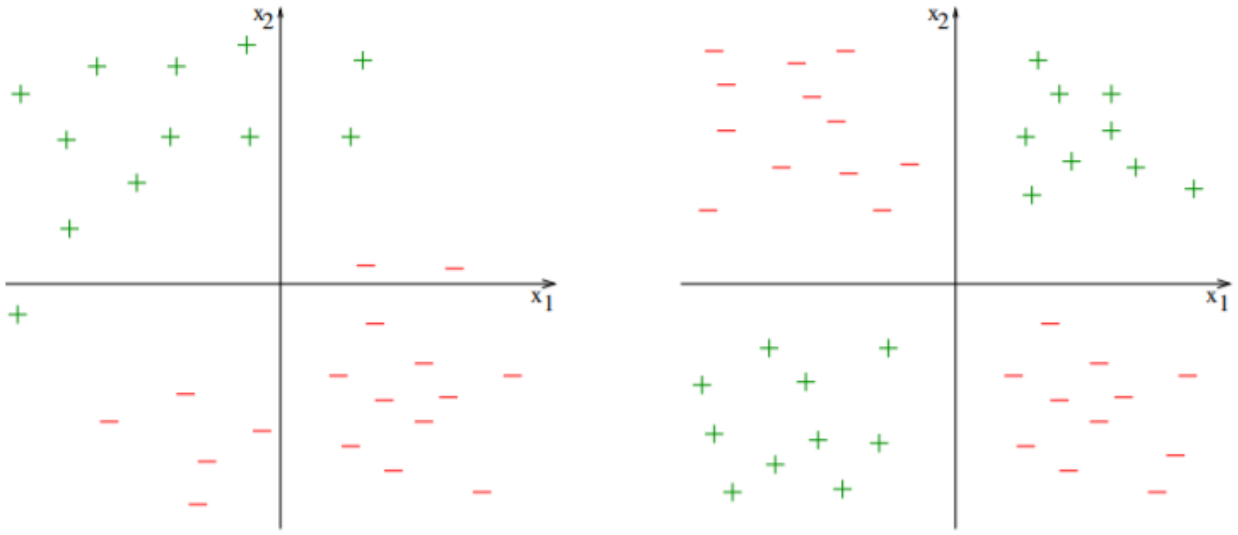
1.2.3. Rezultatai

Tyrimo rezultatai rodo, kad derinant vektorizavimo metodus su regresiniais algoritmais gaunamas patikimesnis programinės įrangos pastangų kiekio vertinimas negu naudojant klasikinius parametrinius modelius [IDC17].

2. „Averaged Perceptron“

Pirmasis perceptrono algoritmas buvo aprašytas [Ros58]. Perceptronas yra tiesinis klasifikatorius, tai reiškia, kad spėjimas daromas remiantis tiesine parametrizuota spėjimo funkcija, bei svorių rinkiniu ir bruožų vektoriumi. Algoritmo tikslas yra iteratyviai keičiant svorius gauti funkciją, kuri gali nuspėti ar įvestis priklauso tam tikrai klasei.

Jeigu duomenys gali būti tiesiškai padalinti plokštumoje pagal priklausymą klasei (vienoje tieses pusėje priklausantys, kitoje nepriklausantys), tai su bet kokiais pradiniais svoriais vykdant iteracijas funkcija konverguos į tiesę, kuri atskiria tuos duomenis pagal priklausomybę klasei [MR13]. Tačiau tuo atveju, kai tiesinis padalinimas neįmanomas atsiranda konvergavimo problemos. Viena iš perceptronų algortimo variacijų, kuri sprendžia šitą problemą yra „Averaged Perceptron“. Šitas algoritmas taip pat koreguoja svorius, tačiau naudoja visų svorių vidurkį iš visų prieš tai vykusių iteracijų.



3 pav. Kairėje tiesiškai plokštumoje padalinami duomenys. Dešinėje nepadalinami [Kal17]

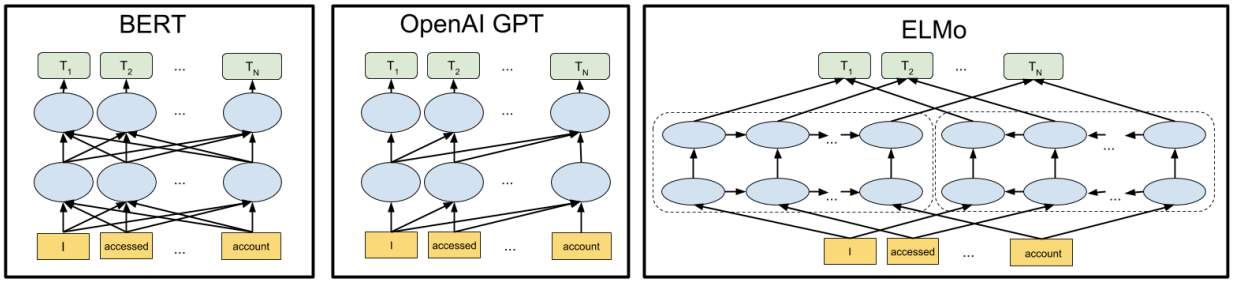
Perceptronais paremti algoritmai yra skirti dvejetainei klasifikacijai, todėl šitas algoritmas negali būti naudojamas tyrimui kaip etalonas, kadangi nagrinėjamos daugiau negu dvi klasės. Uždavinys turi būti išskaldytas į kelis dvejetainės klasifikacijos uždavinius. Viena galimų išskaldymo implementacijų yra „vienas prieš visus“ (One-vs-All). Šios implementacijos atveju yra sukuriamas klasifikatorius kiekvienai klasei, kuris atskiria tą klasę nuo visų likusių [LZ05]. „Averaged Perceptron“ ir „vienas prieš visus“ kombinacija šiame darbe vykdomame tyrime naudojama kaip etalonas.

3. „BERT“

[DCL+18] BERT (Bidirectional Encoder Representations from Transformers) yra Jacob Devlin ir jo kolegų iš „Google“ pasiūlytas, neuronų tinklu paremtas iš anksto apmokytas (pre-trained) modelis, skirtas natūralios kalbos apdorojimo uždaviniams. Modelis puikiai pasirodė atliekant GLUE (General Language Understanding Evaluation) testą, bei SQuAD (Stanford Question Answering Dataset) v1.1 klausimų atsakinėjimo testą.

3.1. Architektūra

BERT modelio architektūra yra daugiasluoksnė ir paremta transformatoriais. Nuo kitų panašių modelių išsiskiria tuo, kad yra abipusė (bidirectional). Tai reiškia, kad transformatorius iš karto perskaito visą teksto įvestį, kitaip negu kryptinių modelių atveju, kai tekstas skaitomas iš kairės į dešinę arba iš dešinės į kairę (architektūrų palyginimas žemiau esančiame paveikslėlyje). Tokiu būdu modelis išmoksta žodžio kontekstą remdamasis tiek dešine tiek kaire to žodžio aplinka.



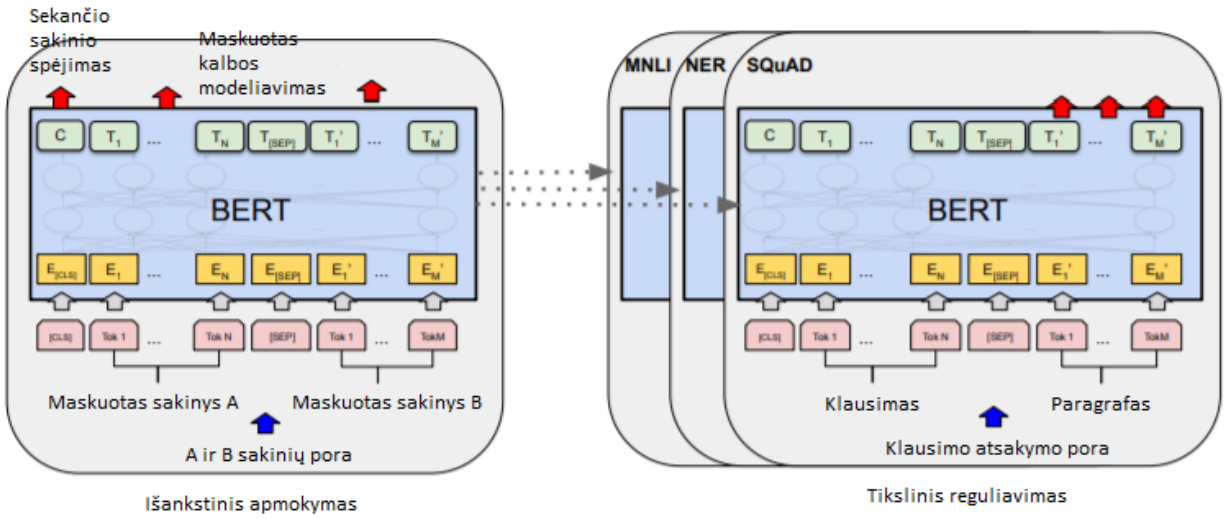
4 pav. Giliai abipusės architektūros palyginimas su kryptine ir paviršutiniškai abipuse [DC18]

[DCL+18] nagrinėjami dviejų dydžių modeliai:

- BERT Base
  - 12 transformatorių blokų;
  - 768 paslėpti sluoksniai;
  - 12 dėmesio procesorių (anglų k. self-attention head).
- BERT Large
  - 24 transformatorių blokai;
  - 1024 paslėpti sluoksniai;
  - 16 dėmesio procesorių (anglų k. self-attention head).

3.2. Apmokymo algoritmas

BERT apmokymas susideda iš dviejų dalių: išankstinio apmokymo (pre-training) ir tikslinio reguliavimo (fine-tuning).



5 pav. Bendra BERT išankstinio apmokymo ir tikslinio reguliavimo schema [DCL+18]

3.2.1. Išankstinis apmokymas

Vykdant išankstinį apmokymą vykdomos dvi užduotys:

- Maskuotas kalbos modeliavimas (Masked LM);
- Sekančio sakinio spėjimas.

Vykdant maskuotą kalbos modeliavimą įvesties duomenyse skirtiems apmokymui, tam tikra žodžių dalis yra maskuojama ir tada bandoma nuspėti paslėptas reikšmes. [DCL+18] visuose eksperimentuose yra maskuojama 15% visų žodžių. Tokiu būdu gaunamas iš anksto apmokytas abipusis modelis.

Kita užduotis iš anksto apmokant BERT modelį yra sekančio sakinio spėjimas. Ši užduotis yra svarbi, nes tokie uždaviniai, kaip atsakinėjimas į klausimus ir natūralios kalbos interpretavimas, yra priklausomi nuo gebėjimo suprasti sąryšius tarp sakinių. Apmokant modelį spėti sekantį sakinį, vienam duomenų vienetui parenkami du sakiniai A ir B. 50% atvejų B yra sakinys, kuris seka A. Likusiais atvejais B yra atsitiktinis sakinys, kuris neseka A.

Modelio autoriai išankstiniam modelio apmokymui naudoja duomenis iš BookCorpus (800 milijonų žodžių) [ZKZ+15] ir angliško Wikipedia puslapio (2,5 milijardo žodžių).

3.2.2. Tikslinis reguliavimas

Išankstinis apmokymas vykdomas naudojant bendrinius duomenis. Norint modelį pritaikyti konkrečiai užduočiai, turi būti vykdomas tikslinis reguliavimas koreguojant visus parametrus bei naudojant užduočiai specifinius duomenis, pavyzdžiui:

- Sakinių poros, perfrazavimo uždaviniui;
- Hipotezių ir prielaidų poros, išvadų iš prielaidų išvedimo uždaviniui;
- Klausimų ir teksto poros, atsakymų į teksto suvokimo klausimus uždaviniui.

Toks modelio apmokymo būdas yra naudingas tuo, kad modelio reguliavimas užtrunka kur kas mažiau negu pilnas modelio apmokymas.

4. „XLNet“

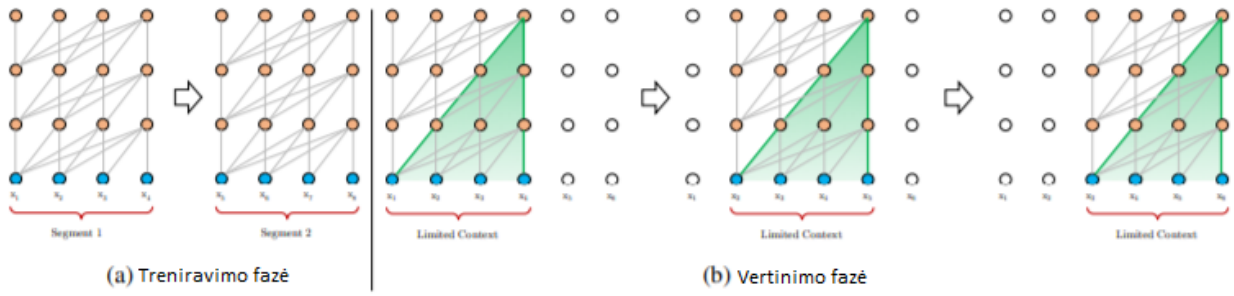
[YDY+19] XLNet yra Zhilin Yang ir jo kolegų siūlomas iš anksto apmokytas modelis natūralios kalbos apdorojimo uždaviniams. Autorių teigimu, šis modelis išsprendžia problemas su kuriomis susiduria BERT:

- BERT, vykdydamas maskuotą kalbos modeliavimą, nepaiso priklausomybių tarp maskuotų žodžių.
- Vykdant tikslinį reguliavimą, duomenys nėra maskuojami, kitaip negu vykdant maskuotą kalbos modeliavimą. Šitas skirtumas sukelia išankstinio apmokymo ir tikslinio reguliavimo neatitikimus.

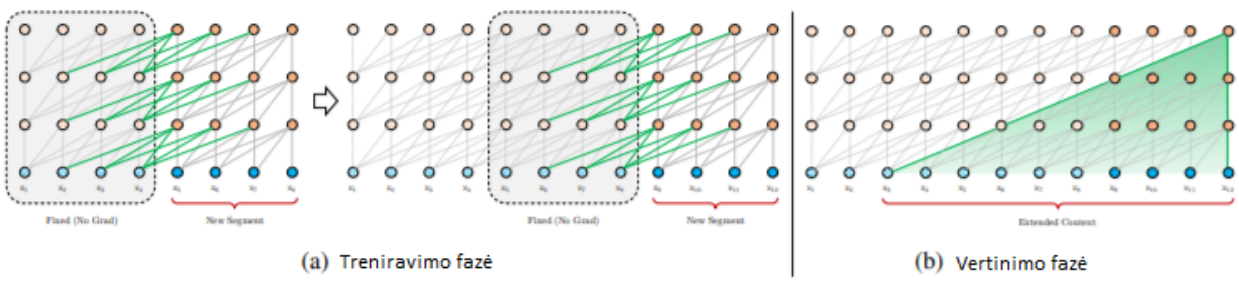
Modelis puikiai pasirodė vykdant eksperimentus su 20 užduočių, tokių kaip atsakymas į klausimus, natūralios kalbos interpretavimas, sentimentų analizė ir dokumentų reitingavimas, susitvarkė geriau negu BERT.

4.1. Architektūra

XLNet architektūra remiasi Transformer-XL [DYY+19] transformatoriais, kurie naudoja dėmesio procesorius kiekviename įvesties duomenų segmente bei rekurentinius mechanizmus, kad išmoktų priklausomybes iš sekančių segmentų. Žemiau esančiuose paveikslėliuose matomas tradicinių transformatorių modelių ir Transformer-XL modelių skirtumas.



6 pav. Tradicinis transformatorių modelis, kurio segmento ilgis yra 4 [DYY+19]



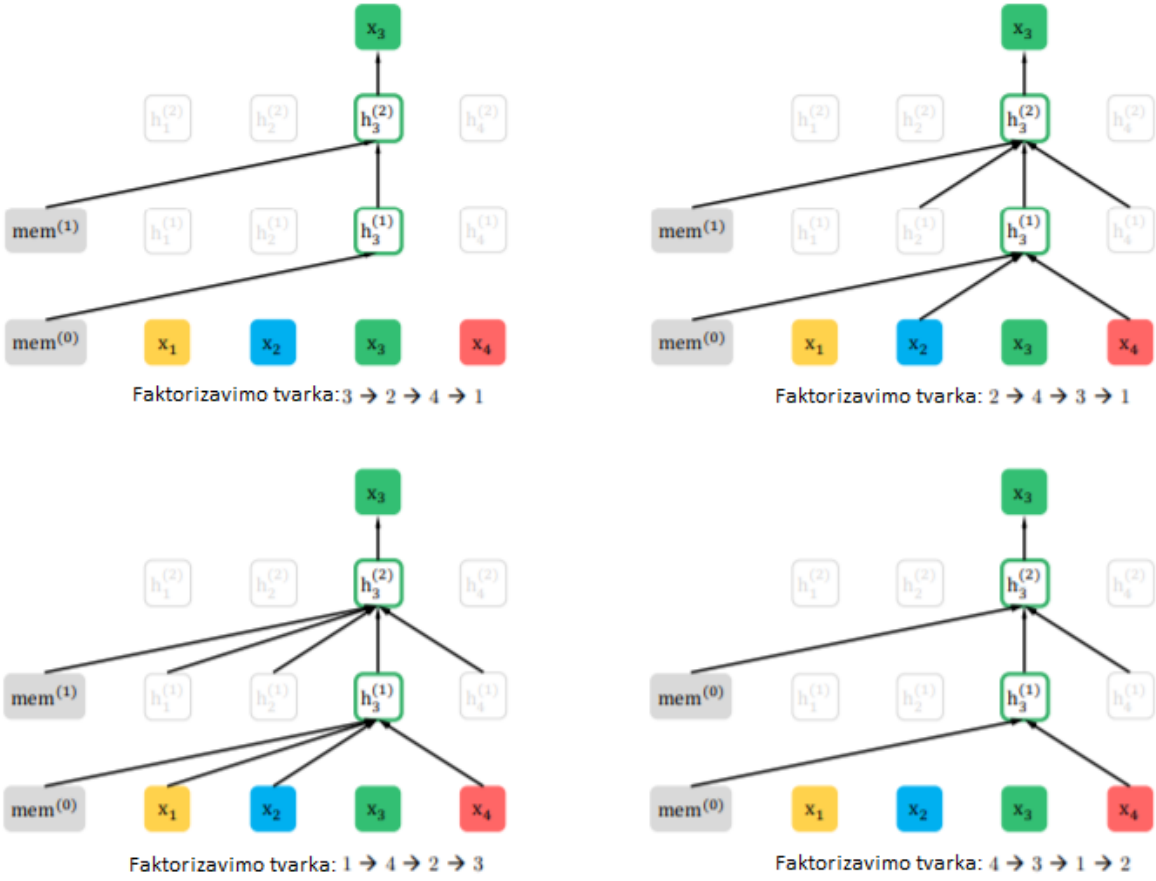
7 pav. Transformer-XL modelis, kurio segmento ilgis yra 4 [DYY+19]

4.2. Apmokymo algoritmas

XLNet atveju, taip pat kaip ir BERT modelio, yra vykdomas išankstinis apmokymas bei tikslinis reguliavimas. Šiame skyriuje tikslinio reguliavimo procedūra nebus aprašoma, kadangi yra panaši į BERT modelio.

4.2.1. Išankstinis apmokymas

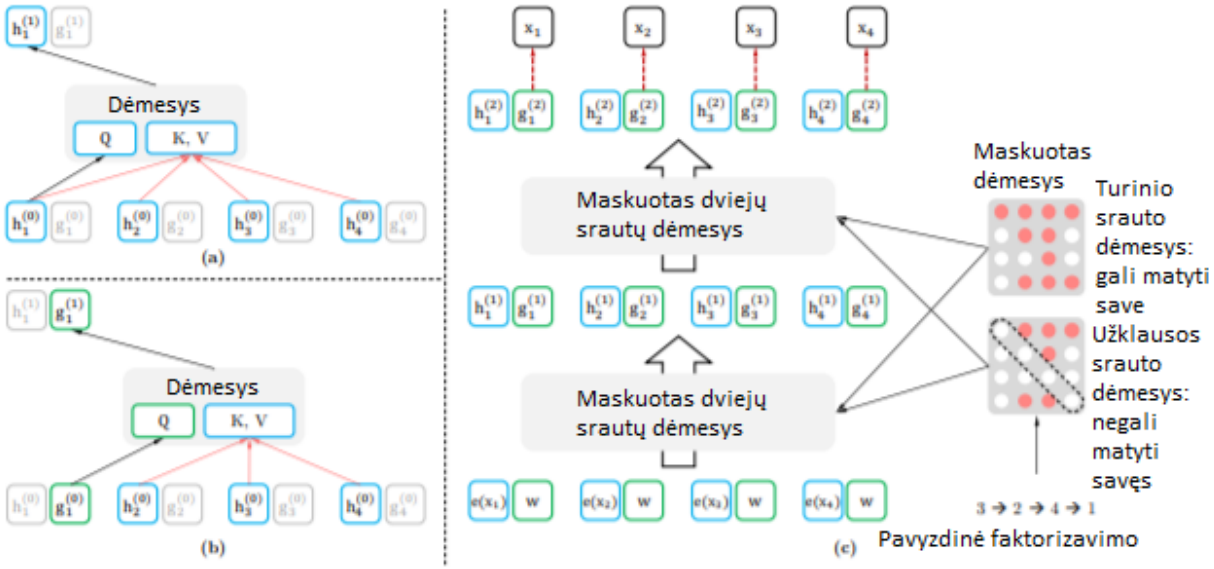
XLNet išankstinis apmokymas susideda tik iš kombinacinio kalbos modeliavimo, kuris sprendžia problemas su kuriomis susiduria BERT: priklausomybių tarp maskuotų žodžių nepaisymas bei išankstinio apmokymo ir tikslinio reguliavimo neatitikimai. Kombinacinis kalbos modeliavimas sprendžia ir kitą problemą. XLNet yra autoregresinis modelis, kas pagal apibrėžimą reiškia, kad tai turėtų būti kryptinis modelis ir būtų apribotas konteksto iš kairės į dešinę arba iš dešinės į kairę. Kombinacinio kalbos modeliavimo metu kontekstas yra išmokstamas naudojant įvairias žodžių faktorizacijos kombinacijas ir tokiu būdu užtikrinamas abipusiškumas. Kombinacinio kalbos modeliavimo schema pavaizduota žemiau esančiam paveikslėlyje.



8 pav. Kombinacinio kalbos modeliavimo schema bandant atspėti x_3 naudojant skirtingą faktorizavimo eiliškumą [YDY+19]

Tačiau toks modeliavimas sukelia naujų problemų: kad mokymosi metu modelis galėtų spėti konkretų žodį, jis turi žinoti tik jo poziciją (šiuo atveju taip nėra, kadangi žodžiai nėra maskuojami). Reikalingas pozicijos ir turinio duomenų atskyrimas. Šiai problemai spręsti XLNet pasitelkia dviejų srautų dėmesio modulius:

- Turinio srauto dėmesys, kuris koduoja konteksto ir kontekstui priklausančių žodžių pozicijų informaciją.
- Užklausos srauto dėmesys, kuris koduoja turinio informaciją ir spėjamo žodžio poziciją.



9 pav. Dviejų srautų dėmesio schema [YDY+19]

Sekančio sakinio spėjimas nenaudojamas išankstiniam XLNet apmokymui, kadangi tyrimo metu buvo išsiaiškinta, kad jis nesuteikia papildomo patikimumo modeliui.

5. Duomenų rinkinys

Tyrimo metu pasirinktas duomenų rinkinys yra surinktas ir pagamintas įmonės „Software in Partnership“ [JC19]. Tai 12299 įrašų apimties masyvas, apibūdinantis įmonės veiklos užduotis bei nagrinėjantis jų trukmę. Įmonės veikla susijusi su programinės įrangos užsakomosiomis paslaugomis. Nemažą duomenų bagažą surinkti padėjo tai, kad įmonė anksti pradėjo naudoti judrumo (anglų k. Agile) metodus.

Pasirinktas duomenų masyvas, toliau vadinamas - SiP, lyginant su kitais natūralios kalbos apdorojimo uždaviniais, pavyzdžiui, nuotaikos analizė, yra mažas. Egzistuoja sakinio nuotaikos nustatymo uždavinio duomenų rinkinių, viršijančių puse milijono įrašų [ZZL15], kuris už pasirinktą SiP rinkinį yra beveik 50 kartų didesnis. Vis dėlto, darbo uždavinių trukmės nustatymo problemą yra sudėtingiau išspręsti bendru atveju, todėl automatizavimo sprendimuose mažo duomenų rinkinio problema turi būti išspręsta kitais mechanizmais.

SiP duomenų rinkinys nagrinėja pastangas veiklai darančius faktorius. Rinkinio įrašai pateikti griežtu formatu:

- Užduoties numeris (TaskNumber) - skaitinis, unikalus užduoties identifikatorius duomenų rinkinio kontekste.
- Aprašas (Summary) - santrauka į kurią galimai įeina užduoties priėmimo kriterijai, vykdymo būdai ir kita užduočiai svarbi informacija.
- Prioritetas (Priority) - skaitinė vertė, nusakanti projekto skubumą. Mažesnis skaitinis rodiklis reiškia didesnį skubumą.
- Kūrėjas (RaisedByID) - skaitinis, unikalus žmogaus, iškėlusio šią užduotį, identifikatorius. Tai gali būti ne tik vidinis įmonės darbuotojas, bet ir klientas.
- Atsakingas asmuo (AssignedToID) - įmonės darbuotojo, atsakingo už užduotis įgyvendinimą, identifikatorius.
- Įgaliotinis (AuthorisedByID) - įmonės vadybininko, turinčio galią patvirtinti užduoties pabaigą kliento vardu, identifikatorius.
- Statusas (StatusCode) - nurodo užduoties įvykdymo etapą. Standartiškai per savo gyvavimo laikotarpį užduoties etapai keičiami šia tvarka: sukurta (CREATED); įvertinta (ESTIMATED); pasirašyta (AUTHORISE); vykdoma (CHRONICLE); įvykdyta (COMPLETED); išleista (RELEASED); užbaigta (FINISHED);

- Projektas (ProjectCode) - užduoties saitas su kitomis užduotimis, atliekamos bendram tikslui įgyvendinti. Neretai, bet nebūtinai, susijęs su kokrečiu klientu.
- Projekto dalis (ProjectBreakdownCode) - kodas, nusakantis projekto pogrupį, kuriai priklauso tam tikra užduotis.
- Kategorija (Category) - identifikatorius, nusakantis tipą. Užduotis gali būti vadybinė, operacinė arba programinės įrangos kūrimo.
- Pakategorė (SubCategory) - kategorijai būdinga reikšmė, nusakanti tikslesnį užduoties darbo tipą. Pavyzdžiui, vadybinė - personalo įdarbinimo užduotis.
- Įvertinimas valandomis (HoursEstimate) - dešimtainė skaitinė reikšmė, atspindinti užduoties trukmės vertinimą valandų išraiška. Atliekama prieš vykdant užduotį.
- Trukmė valandomis (HoursActual) - dešimtainė skaitinė reikšmė, atspindinti bendrą valandų skaičių, kurio vykdymo personalui prireikė įvykdyti šią užduotį.
- Programuotojas (DeveloperID) - identifikatorius darbuotojo, atsitiktinai pasirinkto iš užduotį vykdžiusių programuotojų.
- Programavimo trukmė valandomis (DeveloperHoursActual) - dešimtainė skaitinė reikšmė, atspindinti bendrą valandų skaičių užduoties programavimo dalims.
- Bendras užduoties efektyvumas (TaskPerformance) - dešimtainis skaitinis užduoties perviršis arba nuvertinimas, pasiektas prie jos dirbusių visų programuotojų kartu sudėjus.
- Programuotojo efektyvumas (DeveloperPerformance) - dešimtainis skaitinis užduoties perviršis arba nuvertinimas pasirinktam nagrinėjamam programuotojui.

Šia forma pateikto duomenų rinkinio nagrinėti natūralios kalbos apdorojimo algoritmais negalima, todėl reikalingas duomenų transformavimas. Visų pirma paruošiama standartinei teksto klasifikavimo problemai reikalinga įvestis - žodžių seka. Ją veiklos vertinimo uždavinyje atstoja laisvu formatu parašytas veiklos aprašas. Duomenų rinkinyje visi įrašai turi užduoties aprašą, todėl tuščių eilučių filtruoti nebūtina. Siekiant sumažinti triukšmą aprašo tekstinėje informacijoje, iš aprašų ištrinami skyrybos ženklai ir semantiniai, konteksto nesuteikiantys žodžiai. Šių žodžių, dar vadinamų nutraukimo žodžiais (anglų k. stop word), pašalinimas tai populiari duomenų apdorojimo technika, skirta padidinti mašininio kalbos supratimo rezultatų tikslumą [RU11].

Kitas parametras, reikalingas sukonstruoti veiklos vertinimo uždavinio modelį yra jo klasifikatoriai. Tyrime pasirenkama nenagrinėti užduoties vertinimo ar atlikimo efektyvumo ir klasifikuoti uždavinius pagal tikrąją programavimo trukmę valandomis. Vis dėlto, ne visos duomenų rinkinyje esančios užduotys yra įvykdytos. Tai reiškia, kad reikia išfiltruoti tas eilutes, kurių „programavimo trukmė valandomis“ stulpelis yra tuščias. Po filtravimo proceso duomenų rinkinyje lieka 12256 įrašai. Likusiose eilutėse lieka dešimtainis skaičius, rodantis užduoties vykdymo trukmę. Kadangi klasifikavimo uždaviniui netinka skaitinė išraiška, valandos klasifikuojamos. Iš pradžių veiklos trukmės apimčiai aklai išskiriamos trys klasės, atitinkančios marškinėlių dydžių modelį - maža, vidutinė, didelė. Į mažą užduoties kategoriją patenka įrašai, kurių trukmė mažesnė nei penkios valandos imtinai. Tai tokie uždaviniai, kuriems atlikti reikia maždaug pusės dienos. Kita klasė, vidutinės apimties veikla, yra 5 - 15 val. imtinai apimties darbai. Visi kiti uždaviniai klasifikuojami į trečiąją, didžiųjų, kategoriją.

Atlikę duomenų apdorojimo operacijas [žr. priedą nr. 1], gauname naują duomenų rinkinį D1, kurio struktūra yra:

- Užduoties apimties klasė - maža, vidutinė, didelė.
- Aprašymas - užduoties kontekstą nustatanti tekstinė informacija.

5.1. Pasikartojančių duomenų pašalinimas

Duomenų kopijavimas ir manipuliavimas yra bandymas padidinti pradinių duomenų skaičių, siekiant tolygaus duomenų pasiskirstymo mašininio mokymosi klasifikavimo problemose [KKP+06]. Kita vertus, panašūs ar vienodi duomenys gali lemti modelio perpildymą (anglų k. Overfitting). Pasikartojančia eilute pirminiam modeliui vadinsime tokią, kurios programavimo trukmė valandomis ir aprašymas, sutampa su jau modelyje esančia eilute.

Įsitikinkime, jog modelis D1 turi pasikartojančių eilučių. Tai atlikti sukonstruojame modelį D2 [žr. priedą nr. 2], kurio metu atliekame tas pačias operacijas, kaip ir pirminiam modeliui ir papildomai išfiltruojame pasikartojančias eilutes. Iš pirminių 12256 eilučių D2 modelyje lieka unikalios 10009. Ta pati operacija pakartojama ir kitiems, tyrimo metu sukonstruotiems duomenų masyvams D4, D6 ir D8 [priedai nr. 4, 6, 8].

5.2. Skirtingas klasių skaičius

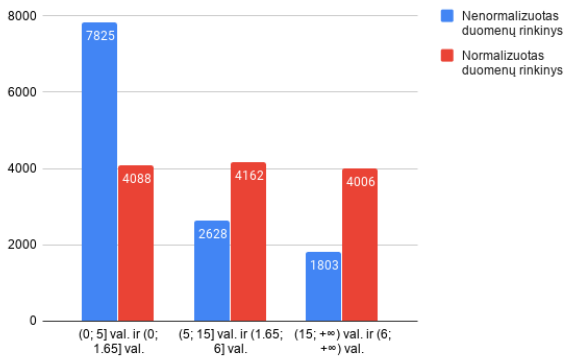
Pradinis duomenų modelis D1 apibrėžtas trims duomenų klasėms. Šis pasirinkimas nėra apribotas SiP duomenų rinkinio ir yra padarytas tyrimo autoriaus nuožiūra. Veiklos trukmės vertinimo

problema taip pat griežtai neriboja klasių kiekio. Tikslinga patikrinti kaip mašininio mokymosi tikslumas keičiasi su iš tų pačių duomenų suformuotu rinkiniu, turinčiu daugiau arba mažiau klasių. Mažiau nei 3 klasių pasirinkti negalima, nes pakeistume formuojamos problemos tipą į binarinę klasifikaciją. Imame paprasčiausią variantą, kuris vis dar atitinka marškinėlių dydžių abstrakciją - 4 klases. Šią duomenų modelio transformaciją kombinuojame su kitomis dvejomis ir gauname D3, D4, D7 ir D8 [priedai nr. 3, 4, 7, 8]. Reikia pastebėti, kad didinant klasių skaičių tam pačiam kiekiui duomenų, gauname retesnį duomenų pasiskirstymą, kas beveik visada reiškia prastesnius mokymosi rezultatus.

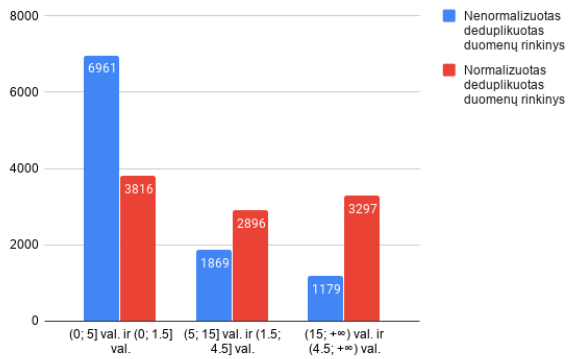
5.3. Normalizavimas

Dar vienas standartinis procesas ruošiant duomenų rinkinį mašininiam mokymui yra duomenų normalizavimas. Normalizuoto rinkinio klasėse elementai pasiskirsto tolygiai [RU11]. Tokiu būdu sumažinamas mokymosi metu įgaunamas duomenų šališkumas. Tai reiškia, kad modelis nesitiki vienos užduočių klasės daugiau, negu kitų. Šio metodo trūkumas yra tas, kad veiklos apimties klasių rėžiai gali būti nustatyti tik turint visus duomenis. Tai reiškia, kad galime gauti reikšmes, kurios nebūtinai sutampa su tomis, kurias apibrėžia projektų valdytojai. Metodas taip pat iškelia sunkumų atliekant tęstinį modelio apmokymą su vis naujais duomenimis, nes nauji duomenys gali neatitikti jau nustatyto pasiskirstymo.

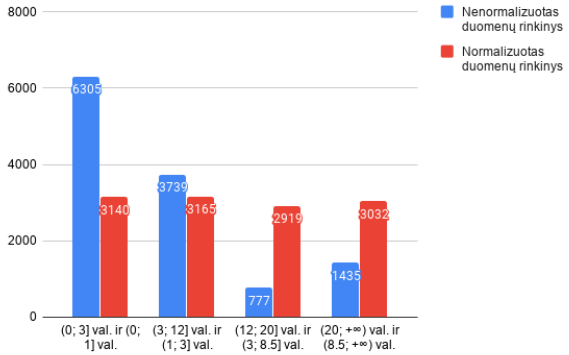
Normalizuota pirminio duomenų rinkinio versija turėtų tą pačią struktūrą. Skirtųsi užduoties apimties klasę apibrėžiantys intervalai. Kadangi pradinio modelio sudaryme klasių rėžiai pasirinkti atsitiktinai, modelis nėra subalansuotas. Normalizavimo procesą atliekame duomenų apdorojimo kode, konstruodami duomenų rinkinius D5, D6, D7 ir D8 [priedai nr. 5 - 8]. Tikėtina, kad normalizuojant trijų ir keturių klasių duomenų rinkinius, rėžius gausime skirtingose vietose. Žemiau esančiuose paveikslėliuose matome, kaip kinta duomenys, po normalizavimo proceso.



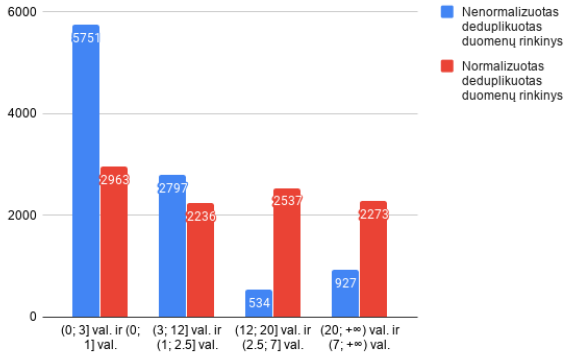
10 pav. D1 ir D5 rinkinių grafinis pavaizdavimas



11 pav. D2 ir D6 rinkinių grafinis pavaizdavimas



12 pav. D3 ir D7 rinkinių grafinis pavaizdavimas



13 pav. D4 ir D8 rinkinių grafinis pavaizdavimas

6. Tyrimo metodika

Tyrimo eksperimentai atliekami sukonstruojant tris mašininio mokymosi modelius, atitinkamai „BERT“, „XLNet“ ir „Averaged Perceptron“ architektūroms. Dėl resursų limito, pasirinkta nagrinėti bazinius transformatoriais technologija grįstus modelius - „bert-base-uncased“ ir „xlnet-base-cased“. „Averaged Perceptron“ atveju yra naudojama vieno prieš visus strategija tam, kad būtų įmanoma sumodeliuoti kelių klasių klasifikacijos problemą.

6.1. Automatinis mašininio mokymosi srautas

Šiame poskyryje aptariamas modelius konstruojančių kodų veikimo principas [žr. priedus 9-11]. Skirtingiems modeliams, kodas yra parašytas atskirose aplinkose, naudojant kitas technologijas, tačiau bendrai jų mokymosi srautų etapai ir etapų eiliškumas sutampa.

Visų pirma pasirinkti duomenų rinkiniai yra atsitiktinai 9/1 santykiu išskiriami į mokymosi ir validavimo rinkinius.

Prieš pradedant mokymo procesą, mokymosi rinkinio bruožai (anglų k. features) turi būti paversti į vienodo dydžio vektorius. Atlikę nedidelį tyrimą nustatome, kad 40 yra didžiausias žodžių skaičius viename aprašyme. „BERT“, „XLNet“ atvejais papildomai yra užkoduojami sakinio pradžios ir pabaigos specialūs ženklai. Šiais atvejais vektoriaus dydį pasirenkame 64. „Averaged Perceptron“ sraute pasinaudojama pagalbine teksto pavertimo į vektorius funkcija, kuri automatiškai parenka maksimalų vektoriaus dydį, pagal didžiausią aprašymo žodžių skaičių.

Kitame etape paruošiamas apmokymo procesas. Eksperimentų būdu pamatuota, kad transformatoriais paremtų modelių apmokymui užtenka 4 mokymo iteracijų, o perceptrono modeliui - 10. Pasirinkus didesnį iteracijų skaičių pastebima validavimo rezultatų mažėjimo tendencija. Tai gali reikšti, kad modelis yra perpildytas (anlgų k. overfitted), t.y. įgauną vertinimo šališkumą mokymosi duomenims. Kitas, partijos dydžio, parametras parenkamas pagal atminties kiekio limitą vykdymo aplinkoje. Dar vienu eksperimento metodu nustatoma, kad didžiausias galimas partijos dydis yra 32, nes atliekant tyrimą su sekančiu logišku dydžiu 48, išsenka vykdymo aplinkos atmintis.

Galiausiai yra vykdomas mokymosi ciklas, kurio metu gaunamas veiklos trukmės vertinimo modelis. mokymosi metu skaičiuojama mokymosi ir validavimo nuostoliai ir epochos metu pasiektas tikslumas. Gauti galutiniai tikslumai užrašomi į rezultatų lentelę.

7. Tyrimo rezultatai

1 lentelėje pavaizduotos reikšmės, gautos tiriant veiklos trukmės įvertinimo uždavinį mašininio mokymosi pagalba. Kiekvienam eksperimentui, paimtas trijų mokymo bandymų vidurkis. Iš trijų skirtingų tinklų tipų, visose kategorijose geriausiai pasirodė „BERT“ vidutiniškai surinkęs 59,8%. Toliau seka „XLNet“ su 58,9% ir „Averaged Perceptron“ tinklas 56,5%. Didžiausias 73% tikslumas pasiektas taip pat „BERT“ su D1 duomenų rinkiniu.

	„Averaged Perceptron“	**„BERT“**	**„XLNet“**
3 klasių duomenų rinkinys (D1)	0,65	0,73	0,71
3 klasių unikalių duomenų rinkinys (D2)	0,68	0,69	0,72
4 klasių duomenų rinkinys (D3)	0,60	0,64	0,55
4 klasių unikalių duomenų rinkinys (D4)	0,56	0,63	0,60
3 klasių normalizuotas duomenų rinkinys (D5)	0,61	0,59	0,58
3 klasių normalizuotas unikalių duomenų rinkinys (D6)	0,52	0,54	0,59
4 klasių normalizuotas duomenų rinkinys (D7)	0,48	0,51	0,49
4 klasių normalizuotas unikalių duomenų rinkinys (D8)	0,42	0,45	0,47

1 lentelė. Užduoties apimties vertinimo tikslumas išgautas skirtingomis mašininio mokymosi architektūromis

Veiklos trukmės uždavinio duomenų rinkiniuose D1, D2, D5 ir D6 padidinus klasių skaičių iki 4, pastebimas tikslumo sumažėjimas 10,08%. Labiausiai krito „XLNet“ rezultatai (12,25%), o mažiausiai „BERT“, kurio rezultatai vidutiniškai sumažėjo 8%. Likusio modelio efektyvumas krito lygiai 10%.

Subalansavus duomenų rinkinį, užduočių apimties vertinimų tikslumas vidutiniškai krito apie 12,58%. Labiausiai nuo normalizavimo proceso nukentėjo „BERT“ modelis vidutiniškai praradęs 15%. Vidutinis nuokrypis tarpusavyje lyginant normalizuotus duomenų rinkinius yra 4,5%, o nenormalizuotus - 7%.

Pašalinus pasikartojančias duomenų rinkinio eilutes, duomenų rinkinių balansas suprastėjo. Vidutinis standartinis nuokrypis po transformacijos padidėjo nuo 80 iki 325. Tai atsispindi ir vidutiniam 4,5% tikslumo sumažėjime balansuotuose rinkiniuose, lyginant duomenis su pasikartojančiomis eilutėmis ir be jų.

Išvados

1. Atliktas tyrimas rodo tai, kad naujausi natūralios kalbos apdorojimo įrankiai nėra efektyvesni už tradicinius mašininio mokymosi modelius, spręsti automatinę užduočių apimties vertinimo problemą. Beveik visais tyrimo atvejais transformatoriais pagrįstos architektūros tik nežymiai pagerino tradicine architektūra gautus rezultatus. Tikėtina to priežastis yra ta, kad natūralios kalbos apdorojimui reikalingas kalbos modeliavimas veiklos trukmės įvertinimui neturi daug įtakos. Tai reiškia, kad veiklos trukmės vertinimas nėra natūralios kalbos apdorojimo problema.

2. Visais atvejais padidinus veiklos trukmės klasių skaičių, algoritmo tikslumas mažėjo. Taigi, projekto valdymo procesų, organizuojančių užduočių apimtį į daug klasių, automatizavimo efektyvumo potencialas yra mažesnis.

3. Subalansuotų duomenų rinkinio rezultatų suprastėjimą galimai lėmė tas faktas, kad sukonstruotų modelių tikslumo tikrinimui buvo naudotas tas pats duomenų skirstinys kaip ir apmokymui. Užduočių trukmės vertinimo automatizavimui tai didelės įtakos neturi, nes validavimui gali būti naudojamas atskiras, jau subalansuotas duomenų rinkinys. Taip pat, šio proceso pranašumą patvirtina ir rezultatai, kurie su normalizuotais rinkiniais yra pastovesni tarp skirtingų modelių matavimo, nei jų nesubalansuoti atitikmenys.

4. Pasikartojančių eilučių pašalinimo technika, pritaikyta pradiniam duomenų modeliui, didelės įtakos neturėjo. Taip pat nebuvo ir krypties (tikslumo sumažėjimo ar padidėjimo), kurią nulemtų ši technika. Galima daryti prielaidą, kad rezultatų pokytį, su atitinkamais duomenų rinkiniais lėmė skirtingas duomenų balansavimas, nes pasikartojantys duomenys pastumia automatiškai nustatomus rėžius. Tokiu atveju, pasikartojančių eilučių pašalinimas arba pridėjimas, galėtų būti viena iš technikų idealiam duomenų rinkinio balansui pasiekti.

5. Nei vienu tyrimo atveju, veiklos tikslumas nepasiekė užsibrėžtos 80% ribos. Kadangi to nepavyko pasiekti jungiant skirtingas mašininio mokymosi technikas su skirtingais duomenų apdorojimo metodais, galima daryti išvadą, kad rezultatus nulėmė viena pagrindinių mašininio mokymosi problemų - per mažas modelių apmokymui skirtas duomenų kiekis. Dėl duomenų rinkinių nesutapimų, nėra visiškai tikslinga tiesiogiai lyginti darbe gautus rezultatus su kitais tyrimais. Tačiau remiantis tuo faktu, kad transformatoriais paremti tinklai neatneša daug naudos, galima daryti išvadą, jog su tais pačiais, kituose tyrimuose naudojamais duomenimis, būtų pasiekti panašūs rezultatai.

Šaltiniai

[BOM07] P. L. Braga, A. L. I. Oliveira ir S. R. L. Meira. Software effort estimation using machine learning techniques with robust confidence intervals. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, p. 352–357, 2007.

[CDT+18] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose ir Tim Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7):637–656, 2018.

[DC18] Jacob Devlin ir Ming-Wei Chang. Open sourcing bert: state-of-the-art pre-training for natural language processing. *Google AI Blog, November*, 2, 2018.

[DCL+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee ir Kristina Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DYY+19] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le ir Ruslan Salakhutdinov. Transformer-xl: attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[HS97] Sepp Hochreiter ir Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[IDC17] Vlad-Sebastian Ionescu, Horia Demian ir Istvan-Gergely Czibula. Natural language processing and machine learning methods for software development effort estimation. *Studies in Informatics and Control*, 26(2):219–228, 2017.

[YDY+19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov ir Quoc V Le. Xlnet: generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, p. 5754–5764, 2019.

[JC19] Derek M Jones ir Stephen Cullum. A conversation around the analysis of the sip effort estimation dataset. *arXiv preprint arXiv:1901.01621*, 2019.

[Kal17] Shivaram Kalyanakrishnan. The perceptron learning algorithm and its convergence, 2017.

[KKP+06] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas ir k.t. Handling imbalanced datasets: a review. *GESTS International Transactions on Computer Science and Engineering*, 30(1):25–36, 2006.

[Lau20] S. Lauesen. It project failures, causes and cures. *IEEE Access*, 8:72059–72067, 2020.

[LZ05] Yi Liu ir Yuan F Zheng. One-against-all multi-class svm classification using reliability measures. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Tom. 2, p. 849–854. IEEE, 2005.

[MKC+20] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu ir Jianfeng Gao. Deep learning based text classification: a comprehensive review. *arXiv preprint arXiv:2004.03705*, 2020.

[MR13] Mehryar Mohri ir Afshin Rostamizadeh. Perceptron mistake bounds. *arXiv preprint arXiv:1305.0208*, 2013.

[PMI13] PMI, red. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, Newtown Square, PA, 5 leid., 2013. ISBN: 978-1-935589-67-9.

[PTP+16] Trang Pham, Truyen Tran, Dinh Phung ir Svetha Venkatesh. Faster training of very deep networks via p-norm gates. *2016 23rd International Conference on Pattern Recognition (ICPR)*, p. 3542–3547. IEEE, 2016.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RU11] Anand Rajaraman ir Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei ir Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser ir Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[WPN+19] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy ir Samuel Bowman. Superglue: a stickier benchmark for general-purpose language understanding systems. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox ir R. Garnett, redaktoriai, *Advances in Neural Information Processing Systems 32*, p. 3266–3280. Curran Associates, Inc., 2019. URL: http://papers.nips.cc/paper/8589-superglue-a-stickier-benchmark-for-general-purpose-language-understanding-systems.pdf.

[ZKZ+15] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba ir Sanja Fidler. Aligning books and movies: towards story-like visual explanations by watching movies and reading books. *Proceedings of the IEEE international conference on computer vision*, p. 19–27, 2015.

[ZZ12] Shahid Kamal Tipu Ziauddin ir Shahrukh Zia. An effort estimation model for agile software development. *Advances in computer science and its applications (ACSA)*, 2(1):314–324, 2012.

[ZZL15] Xiang Zhang, Junbo Zhao ir Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, p. 649–657, 2015.

Priedas nr. 1

Trijų klasių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).ToArray();

var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;

foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}
var classes = transformed.Select(t =>
{
    if (t.Item1 <= 5) return ("0", t.Item2);
    if (t.Item1 <= 15) return ("1", t.Item2);
    else return ("2", t.Item2);
});

await File.WriteAllLinesAsync("D1.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 2

Trijų klasių unikalių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).Distinct().ToArray();

var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}

var classes = transformed.Select(t =>
{
    if (t.Item1 <= 5) return ("0", t.Item2);
    if (t.Item1 <= 15) return ("1", t.Item2);
    else return ("2", t.Item2);
});

await File.WriteAllLinesAsync("D2.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 3

Keturių klasių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).ToArray();

var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}

var classes = transformed.Select(t =>
{
    if (t.Item1 <= 3) return ("0", t.Item2);
    if (t.Item1 <= 12) return ("1", t.Item2);
    if (t.Item1 <= 20) return ("2", t.Item2);
    else return ("3", t.Item2);
});

await File.WriteAllLinesAsync("D3.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 4

Keturių klasių unikalių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).Distinct().ToArray();

var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}

var classes = transformed.Select(t =>
{
    if (t.Item1 <= 3) return ("0", t.Item2);
    if (t.Item1 <= 12) return ("1", t.Item2);
    if (t.Item1 <= 20) return ("2", t.Item2);
    else return ("3", t.Item2);
});

await File.WriteAllLinesAsync("D4.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 5

Trijų klasių normalizuoto duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).ToArray();
var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}
Array.Sort(transformedClean);
var ab = transformedClean.Count() / 3;
var bc = ab * 2;
var classes = transformedClean.Select(t =>
{
    if (t.Item1 <= transformedClean[ab].Item1) return ("0", t.Item2);
    if (t.Item1 <= transformedClean[bc].Item1) return ("1", t.Item2);
    else return ("2", t.Item2);
});

await File.WriteAllLinesAsync("D5.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 6

Trijų klasių normalizuoto unikalių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');

var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).Distinct().ToArray();
var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}
Array.Sort(transformedClean);
var ab = transformedClean.Count() / 3;
var bc = ab * 2;
var classes = transformedClean.Select(t =>
{
    if (t.Item1 <= transformedClean[ab].Item1) return ("0", t.Item2);
    if (t.Item1 <= transformedClean[bc].Item1) return ("1", t.Item2);
    else return ("2", t.Item2);
});

await File.WriteAllLinesAsync("D6.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 7

Keturių klasių normalizuoto duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');
var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).ToArray();
var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}
Array.Sort(transformedClean);
var ab = transformedClean.Count() / 4;
var bc = ab * 2;
var cd = ab * 3;
var classes = transformedClean.Select(t =>
{
    if (t.Item1 <= transformedClean[ab].Item1) return ("0", t.Item2);
    if (t.Item1 <= transformedClean[bc].Item1) return ("1", t.Item2);
    if (t.Item1 <= transformedClean[cd].Item1) return ("2", t.Item2);
    else return ("3", t.Item2);
});

await File.WriteAllLinesAsync("D7.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 8

Keturių klasių normalizuoto unikalių duomenų rinkinio apdorojimo kodas

```
// reikalingas dotnet-stop-words 1.0.4 NuGet paketas
var rawDataset = (await File.ReadAllTextAsync("raw-dataset.txt")).Split('\n');
var transformed = rawDataset.Select(full => full.Split(','))
    .Where(x => x.Length > 12)
    .Where(x => double.TryParse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture, out var _))
    .Select(x => (double.Parse(x[12], NumberStyles.Any, CultureInfo.
    InvariantCulture),
    new string(x[1].ToLowerInvariant().Trim('"').Trim('\t')
    .Where(c => !char.IsPunctuation(c)).ToArray())
    )).Distinct().ToArray();
var engstopwords = StopWord.StopWords.GetStopWords("en");
var transformedClean = transformed;
foreach (var stopword in engstopwords)
{
    transformedClean = transformedClean.Select(x =>
    {
        var arr = x.Item2.Split(' ');
        return (x.Item1, string.Join(' ', arr.Where(y => !y.Equals(stopword)))
    );
    }).ToArray();
}
Array.Sort(transformedClean);
var ab = transformedClean.Count() / 4;
var bc = ab * 2;
var cd = ab * 3;
var classes = transformedClean.Select(t =>
{
    if (t.Item1 <= transformedClean[ab].Item1) return ("0", t.Item2);
    if (t.Item1 <= transformedClean[bc].Item1) return ("1", t.Item2);
    if (t.Item1 <= transformedClean[cd].Item1) return ("2", t.Item2);
    else return ("3", t.Item2);
});
await File.WriteAllLinesAsync("D8.txt", classes.Select(t => $"{t.Item1}\t{t.
    Item2}"));
```

Priedas nr. 9

„BERT“ modelį konstruojantis kodas

```
import tensorflow as tf

import torch

device = torch.device("cuda")

sentences = data.summary.values
labels = data.hrsactual.values

from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=
    True)

max_len = 0

for sent in sentences:
    input_ids = tokenizer.encode(sent, add_special_tokens=True)
    max_len = max(max_len, len(input_ids))

input_ids = []
attention_masks = []

for sent in sentences:
    encoded_dict = tokenizer.encode_plus(
                        sent,
                        add_special_tokens = True,
                        max_length = 64,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                   )

    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

input_ids = torch.cat(input_ids, dim=0)
```

```
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

from torch.utils.data import TensorDataset, random_split

dataset = TensorDataset(input_ids, attention_masks, labels)
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler
batch_size = 32

train_dataloader = DataLoader(
            train_dataset,
            sampler = RandomSampler(train_dataset),
            batch_size = batch_size
        )

validation_dataloader = DataLoader(
            val_dataset,
            sampler = SequentialSampler(val_dataset),
            batch_size = batch_size
        )

from transformers import BertForSequenceClassification, AdamW, BertConfig

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 3, # kai kuriais atvejais 4
    output_attentions = False,
    output_hidden_states = False,
)

model.cuda()

params = list(model.named_parameters())


for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
77
78
79 for p in params[5:21]:
80     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
81
82
83 for p in params[-4:]:
84     print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
85
86 optimizer = AdamW(model.parameters(),
87                   lr = 2e-5,
88                   eps = 1e-8
89                 )
90
91 from transformers import get_linear_schedule_with_warmup
92
93 epochs = 4
94
95 total_steps = len(train_dataloader) * epochs
96
97 scheduler = get_linear_schedule_with_warmup(optimizer,
98                                             num_warmup_steps = 0,
99                                             num_training_steps = total_steps)
100
101 import numpy as np
102
103 def flat_accuracy(preds, labels):
104     pred_flat = np.argmax(preds, axis=1).flatten()
105     labels_flat = labels.flatten()
106     return np.sum(pred_flat == labels_flat) / len(labels_flat)
107
108 import time
109 import datetime
110
111 def format_time(elapsed):
112     elapsed_rounded = int(round((elapsed)))
113     return str(datetime.timedelta(seconds=elapsed_rounded))
114
115 import random
116 import numpy as np
```

```
117
118 seed_val = 42
119
120 random.seed(seed_val)
121 np.random.seed(seed_val)
122 torch.manual_seed(seed_val)
123 torch.cuda.manual_seed_all(seed_val)
124
125 training_stats = []
126 total_t0 = time.time()
127
128 for epoch_i in range(0, epochs):
129     t0 = time.time()
130     total_train_loss = 0
131     model.train()
132     for step, batch in enumerate(train_dataloader):
133         if step % 40 == 0 and not step == 0:
134             elapsed = format_time(time.time() - t0)
135         b_input_ids = batch[0].to(device)
136         b_input_mask = batch[1].to(device)
137         b_labels = batch[2].to(device)
138         model.zero_grad()
139         loss, logits = model(b_input_ids,
140                              token_type_ids=None,
141                              attention_mask=b_input_mask,
142                              labels=b_labels)
143         total_train_loss += loss.item()
144         loss.backward()
145         torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
146         optimizer.step()
147         scheduler.step()
148
149     avg_train_loss = total_train_loss / len(train_dataloader)
150     training_time = format_time(time.time() - t0)
151
152     print("")
153     print("  Average training loss: {0:.2f}".format(avg_train_loss))
154     print("  Training epcoh took: {:}".format(training_time))
155
156     t0 = time.time()
```

```
157
158     model.eval()
159
160     total_eval_accuracy = 0
161     total_eval_loss = 0
162     nb_eval_steps = 0
163
164     for batch in validation_dataloader:
165         b_input_ids = batch[0].to(device)
166         b_input_mask = batch[1].to(device)
167         b_labels = batch[2].to(device)
168         with torch.no_grad():
169             (loss, logits) = model(b_input_ids,
170                                    token_type_ids=None,
171                                    attention_mask=b_input_mask,
172                                    labels=b_labels)
173         total_eval_loss += loss.item()
174         logits = logits.detach().cpu().numpy()
175         label_ids = b_labels.to('cpu').numpy()
176         total_eval_accuracy += flat_accuracy(logits, label_ids)
177
178     avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
179     print("  Accuracy: {0:.2f}".format(avg_val_accuracy))
180
181     avg_val_loss = total_eval_loss / len(validation_dataloader)
182     validation_time = format_time(time.time() - t0)
183
184     print("  Validation Loss: {0:.2f}".format(avg_val_loss))
185     print("  Validation took: {:}".format(validation_time))
186
187     training_stats.append(
188         {
189             'epoch': epoch_i + 1,
190             'Training Loss': avg_train_loss,
191             'Valid. Loss': avg_val_loss,
192             'Valid. Accur.': avg_val_accuracy,
193             'Training Time': training_time,
194             'Validation Time': validation_time
195         }
196     )
```

Priedas nr. 10

„XLNet“ modelį konstruojantis kodas

```
!pip install transformers
import tensorflow as tf
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
    SequentialSampler
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from transformers import XLNetModel, XLNetTokenizer,
    XLNetForSequenceClassification
from transformers import AdamW
from tqdm import tqdm, trange
import pandas as pd
import io
import numpy as np
import matplotlib.pyplot as plt

device = torch.device("cuda")

tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased', do_lower_case=
    True)
tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]

attention_masks = []

for seq in input_ids:
  seq_mask = [float(i>0) for i in seq]
  attention_masks.append(seq_mask)

train_inputs, validation_inputs, train_labels, validation_labels =
    train_test_split(input_ids, labels, random_state=2018, test_size=0.1)
train_masks, validation_masks, _, _ = train_test_split(attention_masks,
    input_ids, random_state=2018, test_size=0.1)

train_inputs = torch.tensor(train_inputs)
validation_inputs = torch.tensor(validation_inputs)
train_labels = torch.tensor(train_labels)
validation_labels = torch.tensor(validation_labels)
```

```
train_masks = torch.tensor(train_masks)
validation_masks = torch.tensor(validation_masks)

batch_size = 32

train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=
    batch_size)

validation_data = TensorDataset(validation_inputs, validation_masks,
    validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler
    , batch_size=batch_size)

model = XLNetForSequenceClassification.from_pretrained("xlnet-base-cased",
    num_labels=3) # gali buti ir 4 klases
model.cuda()

param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'gamma', 'beta']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if not any(nd in n for nd in
    no_decay)],
     'weight_decay_rate': 0.01},
    {'params': [p for n, p in param_optimizer if any(nd in n for nd in
    no_decay)],
     'weight_decay_rate': 0.0}
]

optimizer = AdamW(optimizer_grouped_parameters, lr=2e-5)

def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)

train_loss_set = []

```

```
epochs = 4

for _ in trange(epochs, desc="Epoch"):
  model.train()

  tr_loss = 0
  nb_tr_examples, nb_tr_steps = 0, 0

  for step, batch in enumerate(train_dataloader):
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask, b_labels = batch
    optimizer.zero_grad()
    outputs = model(b_input_ids, token_type_ids=None, attention_mask=
   b_input_mask, labels=b_labels)
    loss = outputs[0]
    logits = outputs[1]
    train_loss_set.append(loss.item())
    loss.backward()
    optimizer.step()

    tr_loss += loss.item()
    nb_tr_examples += b_input_ids.size(0)
    nb_tr_steps += 1

  print("Train loss: {}".format(tr_loss/nb_tr_steps))

  model.eval()

  eval_loss, eval_accuracy = 0, 0
  nb_eval_steps, nb_eval_examples = 0, 0

  for batch in validation_dataloader:
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask, b_labels = batch
    with torch.no_grad():
      output = model(b_input_ids, token_type_ids=None, attention_mask=
   b_input_mask)
      logits = output[0]

    logits = logits.detach().cpu().numpy()
```

```
    label_ids = b_labels.to('cpu').numpy()

    tmp_eval_accuracy = flat_accuracy(logits, label_ids)

    eval_accuracy += tmp_eval_accuracy
    nb_eval_steps += 1

print("Validation Accuracy: {}".format(eval_accuracy/nb_eval_steps))
```

Priedas nr. 11

„Averaged Perceptron“ architektūra pagrįstą modelį konstruojantis kodas

```
// reikalingas Microsoft.ML 1.4.0 NuGet paketas

public class ModelInput
{
    [ColumnName("taskeffortclass"), LoadColumn(0)]
    public float TaskEffortClass { get; set; }

    [ColumnName("summary"), LoadColumn(1)]
    public string Summary { get; set; }
}

public class ModelOutput
{
    [ColumnName("PredictedLabel")]
    public Single Prediction { get; set; }
    public float[] Score { get; set; }
}

// Sukurto modelio validavimui skirta pagalbine funkcija
public static ModelOutput Predict(ModelInput input)
{

    MLContext mlContext = new MLContext();
    string modelPath = "";
    ITransformer mlModel = mlContext.Model.Load(modelPath, out var
    modelInputSchema);
    var predEngine = mlContext.Model.CreatePredictionEngine<ModelInput,
    ModelOutput>(mlModel);
    ModelOutput result = predEngine.Predict(input);
    return result;
}

// Modelio kurimui skirta klase
public static class ModelBuilder
{
```

```
34      public static void CreateModel()
35      {
36          IDataView trainingDataView = mlContext.Data.LoadFromTextFile<
        ModelInput>(
37                                          path: TRAIN_DATA_FILEPATH,
38                                          hasHeader: true,
39                                          separatorChar: '\t',
40                                          allowQuoting: true,
41                                          allowSparse: false);
42
43          IEstimator<ITransformer> trainingPipeline = BuildTrainingPipeline(
        mlContext);
44
45          Evaluate(mlContext, trainingDataView, trainingPipeline);
46
47          ITransformer mlModel = TrainModel(mlContext, trainingDataView,
        trainingPipeline);
48
49          SaveModel(mlContext, mlModel, MODEL_FILEPATH, trainingDataView.Schema)
        ;
50      }
51
52      public static IEstimator<ITransformer> BuildTrainingPipeline(MLContext
        mlContext)
53      {
54          var dataProcessPipeline = mlContext.Transforms.Conversion.
        MapValueToKey("TaskEffortClass", "TaskEffortClass")
55                                      .Append(mlContext.Transforms.Text.
        FeaturizeText("Summary", "Summary"))
56                                      .Append(mlContext.Transforms.CopyColumns("
        Features", "Summary"))
57                                      .Append(mlContext.Transforms.NormalizeMinMax
        ("Features", "Features"))
58                                      .AppendCacheCheckpoint(mlContext);
59          var trainer = mlContext.MulticlassClassification.Trainers.OneVersusAll
        (
60          mlContext.BinaryClassification.Trainers.AveragedPerceptron(
61          labelColumnName: "TaskEffortClass", numberOfIterations: 10,
        featureColumnName: "Features"), labelColumnName: "TaskEffortClass")
62                                      .Append(mlContext.Transforms.Conversion.
```

```
    MapKeyToValue("PredictedLabel", "PredictedLabel"));

        var trainingPipeline = dataProcessPipeline.Append(trainer);

        return trainingPipeline;
    }

    public static ITransformer TrainModel(MLContext mlContext, IDataView
    trainingDataView, IEstimator<ITransformer> trainingPipeline)
    {
        ITransformer model = trainingPipeline.Fit(trainingDataView);
        return model;
    }

    private static void Evaluate(MLContext mlContext, IDataView
    trainingDataView, IEstimator<ITransformer> trainingPipeline)
    {
        var crossValidationResults = mlContext.MulticlassClassification.
    CrossValidate(trainingDataView, trainingPipeline, numberOfFolds: 5,
    labelColumnName: "0");
    }

    private static void SaveModel(MLContext mlContext, ITransformer mlModel,
    string modelRelativePath, DataViewSchema modelInputSchema)
    {
        mlContext.Model.Save(mlModel, modelInputSchema, GetAbsolutePath(
    modelRelativePath));
        Console.WriteLine("The model is saved to {0}", GetAbsolutePath(
    modelRelativePath));
    }

    public static double CalculateStandardDeviation(IEnumerable<double> values
    )
    {
        double average = values.Average();
        double sumOfSquaresOfDifferences = values.Select(val => (val - average
    ) * (val - average)).Sum();
        double standardDeviation = Math.Sqrt(sumOfSquaresOfDifferences / (
    values.Count() - 1));
        return standardDeviation;
```

```
92      }
93
94      public static double CalculateConfidenceInterval95(IEnumerable<double>
        values)
95      {
96          double confidenceInterval95 = 1.96 * CalculateStandardDeviation(values
        ) / Math.Sqrt((values.Count() - 1));
97          return confidenceInterval95;
98      }
99  }
```