LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

DAUDZVALODĪGU JĒDZIENTELPU PIELIETOJUMS NODOMU NOTEIKŠANĀ

MAĢISTRA DARBS

Autors: **Viktorija Leimane**

Studenta apliecības Nr.: vl16047

Darba vadītājs: Dr.sc.comp. Kaspars Balodis

RĪGA 2023

ANOTĀCIJA

Daudzvalodīga lietotāja nodomu noteikšana ir būtiska virtuālo asistentu darbībā, un klientu apkalpošanas automatizācija kļūst arvien izdevīgāka un aktuālāka.

Lietotāju nodomi tiek noteikti vispirms attēlojot lietotāja ievadīto tekstu daudzdimensionālā vektoru telpā jeb jēdzientelpā. Tad mašīnmācīšanās modelis klasificē vektorā izteikto nodomu, lai piegādātu lietotājiem nepieciešamo informāciju.

Darbā tiek izmantots anotēts nodomu noteikšanas korpuss, kas satur lietotāja ievadu un nodomu pārus angļu, latviešu, krievu, igauņu un lietuviešu valodās. Pētījumā tiek salīdzināta nodomu noteikšanas precizitāte divām daudzvalodu jēdzientelpām (ģenerētas ar mBERT un XLM-RoBERTa modeļiem) un divu veidu ievaddatu valodām (oriģinālvalodā un mašīntulkojumā uz angļu valodu), testējot nodomu klasifikācijas modeli, kas apmācīts:

- uz tās pašas valodas korpusa;
- uz visu piecu valodu korpusa;
- tikai uz angļu valodas korpusa.

Rezultāti liecina par to, ka daudzvalodīgas jēdzientelpas un apmācības uz daudzvalodu korpusiem var uzlabot nodomu noteikšanas precizitāti, bet atkarībā no valodas var būt atšķirīgi rezultāti.

Atslēgas vārdi: daudzvalodīgas jēdzientelpas, nodomu noteikšana, mBERT, XLM-RoBERTa

ABSTRACT

Multilingual user intent recognition is essential in the operation of virtual assistants, and automated customer service becomes increasingly cost-effective and relevant. User intents are determined by mapping input text strings to a multidimensional vector space or word embeddings. Then based on the word embedding a machine learning model classifies the intent to deliver the necessary information to users.

This work uses an annotated corpus for intent determination, containing user input and intent pairs in English, Latvian, Russian, Estonian, and Lithuanian. The study compares the accuracy of intent detection for multilingual word embeddings generated by mBERT and XLM-RoBERTa models, as well as three different intent detection approaches for each language (in the original language and machine translated into English), testing the intent classification model trained on: the same language corpus the corpus of all five languages only the English language corpus.

- the same language corpus;
- the corpus of all five languages;
- only the English language corpus.

The results indicate that multilingual word embeddings and training on multilingual corpora can improve intent detection accuracy, but results may vary depending on the language.

Keywords: multilingual word embeddings, intent detection, mBERT, XLM-RoBERTa

SATURS

Apzīmējumu saraksts 4

Ievads 5

1 Jēdzientelpa 6

1.1 Izkliedētā reprezentācija 7

1.2 Sintaktiskās un semantiskās attiecības 9

1.3 Daudzvalodīga jēdzientelpa 12

1.4 Jēdzientelpu korpusi 12

1.5 Jēdzientelpu modeļu apmācība 16

1.5.1 Continuous Bag-of-Words 16

1.5.2 Continuous Skip-gram Model 17

2 Nodomu noteikšana 19

2.1 Nodomu noteikšanas korpusi 20

3 Daudzvalodu nodomu noteikšana 23

3.1 Apmācība un testēšana vienā valodā 23

3.2 Apmācība uz visām valodām, testēšana vienā valodā 24

3.3 Apmācība angļu valodā, testēšana valodās, kas nav angļu 25

3.4 Mašīntulkošana uz angļu valodu 26

3.5 Citas pieejas 27

4 Rezultāti 29

4.1 Chatbot datu kopa 30

4.2 Askubuntu datu kopa 36

4.3 Webapps datu kopa 41

Secinājumi 46

Pateicības 47

Izmantotā literatūra un avoti 48

Pielikums 53

APZĪMĒJUMU SARAKSTS

NLP (*natural language processing*) – dabisko valodu apstrāde.

Jēdzientelpa (*word embeddings*) – vārdu vai frāžu attēlojums daudzdimensionālā vektoru telpā.

Word2Vec (*word to vector*) – jēdzientelpas implementācija, kurā individuālus vārdus aizstāj daudzimensionāli vektori.

PCA (*Principal Component Analysis*) – galveno komponentu analīze.

GPT (*Generative Pre-trained Transformer*) – dziļo neironu tīklu modelis, kas spēj producēt tekstu, kas līdzīgs cilvēka rakstītam.

Transformeris (*transformer*) – dziļās mācīšanās modelis ar uzmanības (attention) mehānismu, kas spēj novērtēt ievades daļas nozīmīgumu.

Pārpielāgošana (*overfitting*) – pārmērīga pielāgošanās kādam konkrētai datu kopai, zaudējot spēju ģeneralizēt uz citām datu kopām.

Pietrenēšana (*fine-tuning*) – metode, kurā iepriekš apmācīts modelis tiek pietrenēts jaunam uzdevumam.

XLM (*Cross-lingual Language Model*) – valodas modelis, kas apmācīts uz daudzvalodu datiem, lai apgūtu jēdzientelpas, ko var pielietot vairākām valodām.

Multilingual BERT (*Bidirectional Encoder Representations from Transformers*) –

XLM-R (*Cross-lingual Language Model pre-trained with XLM*) –

XLM-Roberta (*Robustly Optimized BERT Pretraining Approach*) –

IEVADS

Dabiskā valodas apstrāde (NLP – *natural language processing*) ir starpdisciplināra datorlingvistikas un mākslīgā intelekta nozare, kas strādā pie tā, lai datori varētu saprast cilvēka dabiskās valodas ievadi. Dabiskās valodas pēc būtības ir sarežģītas, un daudzi NLP uzdevumi ir slikti piemēroti matemātiski precīziem algoritmiskajiem risinājumiem. Palielinoties korpusu – liela apjoma rakstītas vai runātas dabiskās valodas kolekciju – pieejamībai, NLP uzdevumi arvien biežāk un efektīvāk tiek risināti ar mašīnmācīšanās modeļiem [1].

Arvien lielāku daļu tirgus pārņem pakalpojumu industrija, un pakalpojumi arvien biežāk tiek piedāvāti starptautiski. Tam ir nepieciešams lietotāju dzimtās valodas atbalsts gan valstu valodu regulējumu, gan tirgus nišas ieņemšanas un tirgus konkurences dēļ.

Dabiskās valodas apstrādei ir liels biznesa potenciāls, jo tas ļauj uzņēmumiem palielināt peļņu samazinot izdevumus, no kuriem lielākais parasti ir darbs. Tas savukārt samazina barjeru iekļūšanai un dalībai starptautiskā tirgū, kas nozīmē lielāku piedāvāto pakalpojumu daudzveidību un konkurenci, tātad zemākas izmaksas patērētājam. Lietotājiem, kuru dzimto valodu pārvalda mazs cilvēku skaits kā tas ir, piemēram, latviešu valodā, kļūst pieejami pakalpojumi, kuru tulkojumus būtu ekonomiski nerentabli nodrošināt ar algotu profesionālu personālu.

Darbā apskatītā metode nodrošina automatizāciju divos veidos:

- daudzvalodīgs modelis aizvieto profesionālu tulkotāju;
- virtuālais asistents aizvieto klientu apkalpošanas speciālistu.

Darbs ir sadalīts teorētiskajā un praktiskajā daļā. Teorētiskajā daļā ir īsi aprakstīti mūsdienu modeļi un pieejas. Praktiskajā daļā ir veikti eksperimenti ar mērķi pielietot daudzvalodīgus modeļus un salīdzināt tos ar esošiem risinājumiem.

Pētījuma jautājums: Kādas ir efektīvākās metodes un daudzvalodīgi jēdzientelpu modeļi daudzvalodu nodomu noteikšanai?

1. JĒDZIENTELPA

Jēdzientelpa (*word embedding*)[1] ir vārdu vai frāžu attēlojums daudzdimensionālā vektoru telpā. Jēdzientelpu pamatā ir ideja, ka vārdiem, kuriem ir līdzīga nozīme un kurus lieto līdzīgos kontekstos, daudzdimensiju telpā jābūt savstarpēji tuvākiem, bet vārdiem ar atšķirīgu nozīmi un kontekstiem jābūt tālākiem, piemēram, vārds "suns" būs tuvāk vārdiem "kaķis" un "mājdzīvnieks" nekā vārds "koks".

To, cik tuvu ir vārdi jēdzientelpā, var noteikt, izmantojot kosinusa līdzības (*cosine similarity*) metriku [2]. Ja vienam vārdam atbilst vektors $\vec{a}$, bet otram – vektors $\vec{b}$, tad kosinusa līdzību $K.L.$ var atrast šādā veidā:

$$K.L. = \cos(\varphi) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|},$$

kur φ ir leņķis starp vektoriem $\vec{a}$ un $\vec{b}$, bet $||$ apzīmē vektora garumu jeb moduli: $|\vec{a}| = \sqrt{\vec{a} \cdot \vec{a}}$.

Noderīgs sākumpunkts izpratnei par vārdu attēlošanu ar skaitļu vektoriem un šo attēlojumu pielietojamības robežām ir vienizcēluma kodējums (*one hot encoding*). Vienizcēluma kodējums dabiskās valodas apstrādē ir vektors, kurā katrs vektora elements ir sasaistīts ar vārdu krājuma elementu. Līdz ar to katrs vārds ir vektors, kurā atbilstošais elements ir 1 un visi pārējie elementi ir 0 [3]. Piemēram, ja vārdu krājumā ir četri vārdi: karalis, karaliene, sieviete, vīrietis, karaliene tiktu kodēta kā [0, 1, 0, 0].

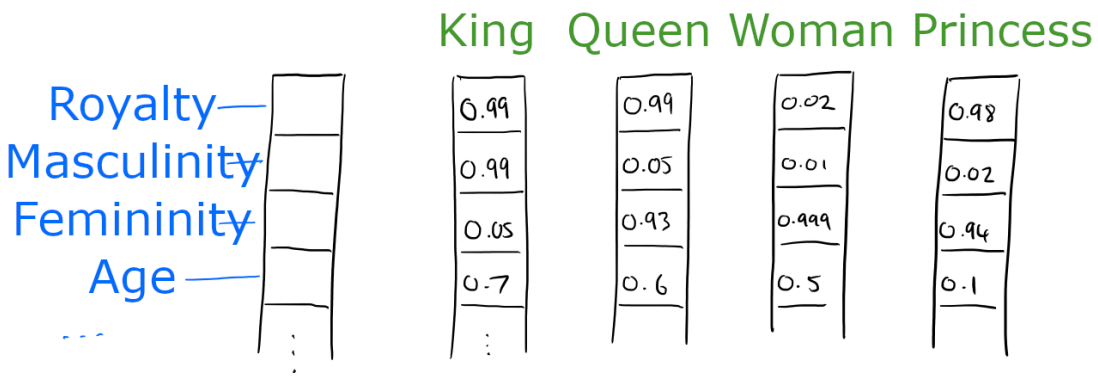
Vairākas NLP metodes izmanto vienizcēluma kodējumu, attēlojot vārdus kā indeksus vārdnīcā. Šai izvēlei ir priekšrocības, piemēram, ir novērots, ka vienkāršie modeļi, kas apmācīti uz milzīgu datu apjomu, pārspēj sarežģītas sistēmas, kas apmācītas ar mazāku datu apjomu [4]. Taču vektora garuma sasaiste ar vārdu krājuma izmēru ir trūkums, jo vārdu vektori ir cieši savienoti (*coupled*) ar korpusu un statiski, piemēram, pievienot jaunu vārdu nozīmē katram esošajam vārdu vektoram pievienot papildus nulli, tātad nāktos pārtrenēt visu modeli. Tāpat palielinoties dimensiju skaitam telpa pieaug tik strauji, ka daudzdimensiju telpām raksturīga izretinātība (*sparsity*): vienizcēluma kodējuma vektorā ir tikai viens nenulles elements un korpusos mēdz būt miljardiem vārdu. Visbeidzot vienizcēluma kodējums nesatur kontekstuālu vārdu nozīmi, nav korelācijas starp vārdiem ar līdzīgu nozīmi un lietojumu [3].

[1] Oficiālais latviskojums vārdam "word embedding" ir "vārdlietojuma kartējums". Autors dod priekšroku "jēdzientelpa" un apzināti izvēlas lietot to darbā, lai palielinātu "jēdzientelpa" izplatību korpusos, uz kuriem tiks trenēti valodu modeļi.

1.1. Izkliedētā reprezentācija

Atšķirībā no dabisko valodu apstrādes metodēm, kas katru vārdu uztver kā vienu atsevišķu vienību un tādēļ vienīgā iespējamā darbība ar vārdiem ir pārbaudīt vienādību, katras jēdzientelpas vektora vērtības ietekmē vārdi tiem apkārt jeb reprezentācija ir izkliedēta (*distributed representation*) un būtībā jēdzientelpas uztver attiecības starp vārdiem. Rezultātā vārdam atbilstošais vektors satur semantisku un sintaktisku informāciju par vārdu. No tā izriet praktiskā implikācija – ar vektoriem var veikt lineārās algebras operācijas, piemēram, saskaitīt un atņemt [3].

Vārdiem kā tādiem ir grūti piekārtot konkrētu skaitli, taču vārdi apraksta objektus ar noteiktām kvantificējamām īpašībām, piemēram, vieglāks/smagāks (masa), lētāks/dārgāks (cena). Šādai reprezentācijai ir jēga, jo dažādus objektus var salīdzināt savā starpā viendimensionālā telpā pēc konkrētas īpašības vērtības jeb izteiktības pakāpes, piemēram, velosipēds ir vieglāks nekā mašīna. Tomēr ar vienu dimensiju nepietiek, lai viennozīmīgi izteiktu vārdu nozīmi, piemēram, ir daudz vārdu, kas apraksta objektus, kuru masa var būt 5 kg: suns, kaķis, soma, maiss ar rīsiem, hantele utml. Tādējādi, lai arī vārdi var tikt salīdzināti pēc vienas konkrētas īpašības, vārdu viendimensionāla reprezentācija nespēj pilnībā ietvert vārda nozīmi. Kvantitatīva semantikas izteikšana kļūst iespējama tikai daudzdimensiju telpā, kur vārda attēlojums tiek sadalīts pa visiem vektora elementiem, un katrs elements pievieno nozīmi daudziem vārdiem, kā redzams 1.1 attēlā. Tas nozīmē, ka vārdu nozīme tiek izteikta caur kontekstu.



1.1. att. **Vārdu vektoru piemērs, kur katra dimensija ir novērtēta ar svariem un atbilst hipotētiskai vārda nozīmes niansei [3].**

Cilvēkiem uztverama jēdzientelpu analoģija ir krāsas nosaukums un tam atbilstošais vektors RGB krāsu modelī ar R (sarkans), G (zaļš) un B (zils) koordinātēm no 0 līdz 255, piemēram, sarkans = (255, 0, 0). Ar krāsu jēdzientelpām ir iespējams veikt saskaitīšanu un atņemšanu, kam ir fizikāla nozīme [5].

Atrast tuvākās krāsas sarkanam.

```
closest(colors, colors['red'])
# red (229, 0, 0)
# fire engine red (254, 0, 2)
# bright red (255, 0, 13)
# tomato red (236, 45, 1)
# cherry red (247, 2, 42)
```

Operācijas ar vektoriem darbojas gan krāsu nosaukumiem semantiski, gan skaitliskiem vektoriem krāsu telpā. Piemēram, tuvākais vektors violeta un sarkana starpībai ir zils, kas atbilst cilvēku intuīcijai par RGB krāsām.

$$purple - red = blue$$

$$(126, 30, 156) - (229, 0, 0) = (-103, 30, 156)$$

```
closest(colors, subtractv(colors['purple'], colors['red']))
# cobalt blue (3, 10, 167)
# royal blue (5, 4, 170)
# darkish blue (1, 65, 130)
# true blue (1, 15, 204)
# royal (12, 23, 147)
```

Tā saskaitot zaļu un zilu rodas kaut kas pa vidu – tirkīzs.

$$blue + green = turquoise$$

$$(3, 67, 223) + (21, 176, 26) = (24, 243, 249)$$

```
closest(colors, addv(colors['blue'], colors['green']))
# bright turquoise (15, 254, 249)
# bright light blue (38, 247, 253)
```

```
# bright aqua (11, 249, 234)
# cyan (0, 255, 255)
# neon blue (4, 217, 255)
```

No vektoru operācijām var izdarīt secinājumus par semantiskajām attiecībām starp vārdiem, piemēram, rozā sarkanam ir tas pats, kas gaiši zils zilam.

$$pink - red + blue = lightblue$$

$$(255, 129, 192) - (229, 0, 0) + (3, 67, 223) = (29, 196, 415)$$

```
closest(colors, addv(subtractv(colors['pink'], colors['red']), colors['blue
                                        ']))
# neon blue (4, 217, 255)
# bright sky blue (2, 204, 254)
# bright light blue (38, 247, 253)
# cyan (0, 255, 255)
# bright cyan (65, 253, 254)
```

Kā analoģiju izkliedētai reprezentācijai var apsvērt arī ģeogrāfiskā platuma un garuma koordinātas kā vektora attēlojumu vietu nosaukumiem. Divu ģeogrāfisku punktu tuvums koordinātēs var norādīt uz līdzīgu klimatu, vēsturi, kultūru un citiem faktoriem. Piemēram, Rīga (56°57′N 24°6′E) ir līdzīgāka Viļņai (54°41′N 25°19′E) nekā Riodežaneiro (22°54′40″S 43°12′20″W). Tāpat jēdzientelpas ir veids, kā attēlot vārdus kā vektorus daudzdimensiju telpā, kur vārdi ar līdzīgu nozīmi vai kontekstu atrodas tuvāk viens otram. Tāpat kā krāsas var attēlot kā vektorus RGB telpā un vietas var attēlot kā vektorus platuma-garuma telpā, vārdus var attēlot kā vektorus semantiskā telpā, kas atspoguļo to attiecības ar citiem vārdiem.

1.2. Sintaktiskās un semantiskās attiecības

Izrādās, tādas pašas sakarības, kādas ir krāsu nosaukumiem un to attēlojumiem krāsu telpā, ir spēkā jebkuram vārdam. Vārdi, kuri bieži atrodas līdzīgos kontekstos, ir tuvāki pēc nozīmes. Jēdzientelpas ietver gan sintaktiskas (1.2 tabula), gan semantiskas (1.1 tabula) attiecības starp vārdiem. Jāuzsver, ka tādas semantiskas attiecības kā valsts–galvaspilsēta (1.2) nav uzdotas tiešā

veidā, jēdzientelpu modelis tās ir novērojis tikai balstoties uz vārdu atrašanās vietām teksta korpusā. Iespēja trenēt modeli uz neanotētiem datiem kā šajā gadījumā samazina modeļa trenēšanas izmaksas valodām, kurās anotēti dati ir mazāk pieejami, un daudzkārt palielina potenciālās treniņu kopas apjomu, kas parasti ļauj sasniegt lielāku precizitāti.

Spēja noteikt sintaktiskas un semantiskas vārdu attiecības ir īpaši būtiska virtuālo asistentu jomā, jo, pirmkārt, semantiski līdzīgiem nodomiem ir līdzīgi vektori, tātad tie tiks vienādi klasificēti, otrkārt, informācija par sintakses attiecībām noder, jo lietotāji ievada jautājumus brīvā formā un tas ir it īpaši svarīgi fleksīvām valodām kā latviešu.

Fleksīva valoda ir valoda, kurā vārdi var mainīties atkarībā no to funkcijas teikumā, t.i., no dzimtes, skaitļa, locījuma, laika un citiem gramatiskiem faktoriem. Piemēram, lietvārds latviešu valodā var būt sešos dažādos locījumos, un jēdzientelpā viena vārda dažādi locījumi tiks pārstāvēti kā dažādi vārdi. Tas nozīmē, ka modelis var nepareizi interpretēt tos kā atsevišķus vārdus ar atšķirīgu nozīmi, it īpaši vārdus retāk izmantotās gramatiskās formās. To var mēģināt novērst ar lemmatizāciju (*lemmatization*), kas samazina unikālo vārdu formu skaitu, grupējot vārdus pamatformās. Piemēram, ”mājas” un ”māju” varētu reducēt līdz pamatformai ”māja”. Atkarībā no valodas sarežģītības lemmatizācija var izrādīties neparocīgāka nekā korpusa palielināšana.

1.1. tabula

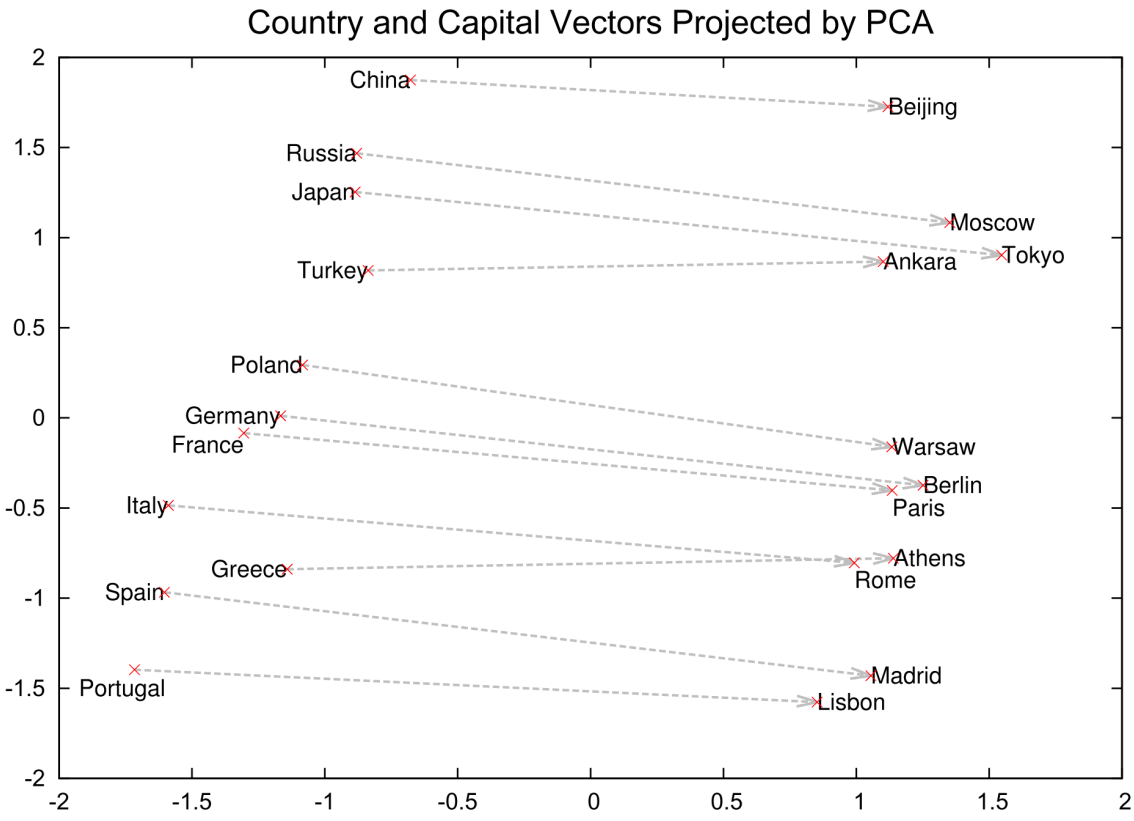
Semantisko attiecību piemēri [4]

attiecība	piemērs
valsts–galvaspilsēta	Parīze - Francija + Itālija = Roma
valsts–valūta	dolāri - ASV + Latvija = eiro
vīrietis–sieviete	karalis - vīrietis + sieviete = karaliene

1.2. tabula

Sintaktisko attiecību piemēri [4]

attiecība	piemērs
daudzskaitlis	pele - peles
pagātne	staigā - staigāja
salīdzināmā pakāpe	labs - labāks



1.2. att. **Divdimensionāla PCA projekcija uzrāda attiecības starp valstu un galvaspilsētu jēdzientelpām [3]**

1.3. Daudzvalodīga jēdzientelpa

Daudzvalodīgas jēdzientelpas no vienvalodīgām jēdzientelpām atšķiras ar to, ka uztver attiecības starp vārdiem no dažādām valodām.

Tā kā vienvalodīgas jēdzientelpas tiek trenētas tikai uz vienas valodas, tās nespēj notvert attiecības starp vārdiem dažādās valodās, un spēj raksturot tikai attiecības starp vārdiem vienā valodā, piemēram, vārds ”suns” tiek reprezentēts kā vektors, kas jēdzientelpā ir tuvs citiem ar suņiem saistītiem vārdiem kā ”kucēns” un ”riet”, bet nav sasaistīts ar suņiem saistītiem vektoriem citās valodās.

Turpretī daudzvalodīgas jēdzientelpas tiek trenētas uz paralēliem datiem - vienādas nozīmes tekstiem dažādās valodās. Tas ļauj notvert starpvalodu (*cross-lingual*) sakarības starp līdzīgas nozīmes vārdiem kopējā jēdzientelpā, piemēram, vārdi ”suns” un ”dog” (”suns” angļu valodā) tiek reprezentēti kā tuvi vektori kopējā jēdzientelpā, kas norāda uz līdzīgu nozīmi. Daudzvalodīgas jēdzientelpas īpaši noder valodām ar mazākām treniņdatu kopām, jo palīdz tulkot un atgriezt informāciju starp valodām (*cross-language information retrieval*) – piemēram, atgriezt kādām vaicājumam angļu Vikipēdijas lapu, ja tai nav latviešu Vikipēdijas ekvivalenta. Pat tādām populārām valodām kā spāņu un hindi Vikipēdijas rakstu skaits ir būtiski mazāks par rakstu skaitu angļu valodas versijā (attiecīgi 26% un 2.3%), kā arī vairākums no šiem rakstiem ir īsāki, nekā to angliskās versijas. Turklāt gandrīz pusei no ne-angļu rakstiem nav attiecīgā analoga angļu versijā, visdrīzāk tāpēc, ka tie raksti ir veltīti specifiskai kulturālai vai ģeogrāfiskajai informācijai. Tas parāda nepieciešamību ņemt vērā saturu vairākās valodās, lai korekti un līdzvērtīgi pārstāvētu mazās valodas mašīnapmācības procesā [6].

1.4. Jēdzientelpu korpusi

Lielo teksta korpusu un mašīnmācīšanās modeļu precizitātes vēsture ir cieši saistīta. Agrīnie mašīnmācīšanās algoritmi balstījās uz nelielām manuāli veidotām datu kopām, kas ierobežoja to efektivitāti. Viens no pirmajiem korpusiem bija *Standard Sample of Present-Day American English*, plašāk pazīstams kā *The Brown Corpus*, kas tika izdots 1964-1965. gadā un sastāvēja no apmēram viena miljona vārdu angļu teksta no dažādiem avotiem [7], tas ir mazs apjoms teksta salīdzinot ar mūsdienās pieejamo.

Procesoru jaudas palielināšanās kopā ar datoru un interneta savienojuma pieejamību plašākai sabiedrībai ir radījuši labvēlīgu vidi izveidot un uzglabāt lielu daudzumu digitālo datu, tostarp teksta formā. Lieliem teksta korpusiem ir bijusi izšķiroša loma efektīvu mašīnmācīšanās modeļu izstrādē. Mašīnmācīšanās modeļu efektivitāte ir proporcionāla tiem pieejamo apmācības datu lielumam un kvalitātei.

Pirms lielu teksta korpusu pieejamības mašīnmācīšanās modeļi aprobežojās ar mazām un salīdzinoši vienkāršām datu kopām, tādēļ bija grūti sasniegt augstu precizitāti dabiskās valodas apstrādes uzdevumos. Mūsdienās lielos teksta korpusos kā Common Crawl un Wikipedia ir miljardiem vārdu vairākās valodās, kas ļauj modeļiem iemācīties ģenerēt cilvēkiem līdzīgu valodu.

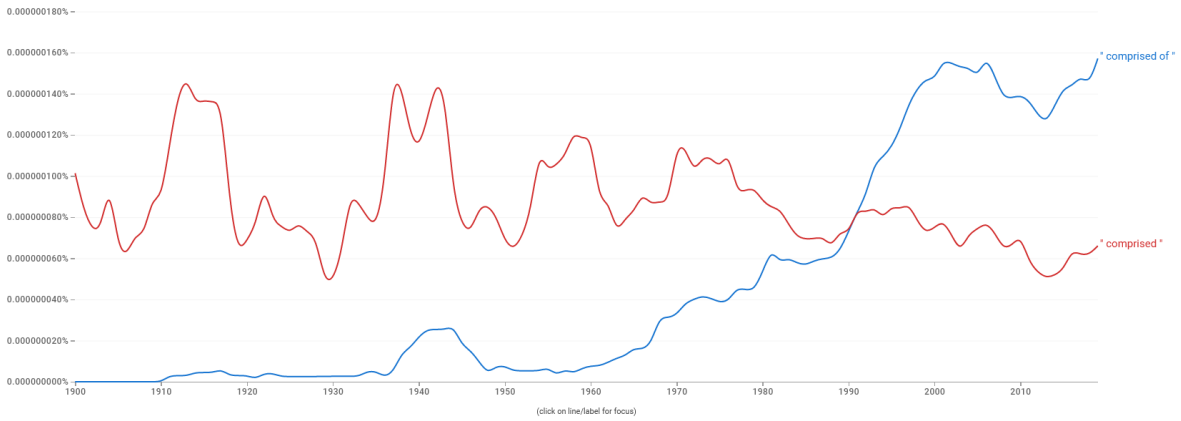
Taču vai visas valodas ir līdzvērtīgi pārstāvētas korpusos? Viegli iztēloties, ka tādi lielie korpusi kā Common Crawl (kopš 2008. gada ievākti petabaiti datu no interneta mājaslapām, tostarp Vikipēdijas un Reddit) līdzvērtīgi pārstāv visu Zemes iedzīvotāju valodas. Taču valodu reprezentācija korpusos ir saistīta ar rakstītā teksta datu pieejamību šajā valodā, un neprecīzi atspoguļo cilvēku skaitu, kuri runā šajā valodā. Piemēram, valodai, kurā runā liels skaits cilvēku, korpusā var būt maz marķieru, ja šajā valodā ir maz digitāli pieejama rakstīta teksta [8].

Tomēr dažādi faktori traucē visiem rakstīt tekstus internetā, kas vēlāk nokļūst korpusos, piemēram, rakstītneprasme, nabadzība, ierīču un interneta nepieejamība, karš utml. Tā rezultātā korpusos ir disproporcionāli pārstāvēti gados jaunāku lietotāju no attīstītajām valstīm drukāti teksti, piemēram, GPT-2 apmācības dati tika ievākti no Reddit, un pēc Pew Internet Research pētījuma 67% Reddit lietotāju Amerikas Savienotajās Valstīs ir vīrieši un 64% vecumā no 18 līdz 29 gadiem [8].

Līdzīgi 87% Vikipēdijas ierakstu veicēji ir vīrieši. Gandrīz puse dzīvo Eiropā un viena piektā daļa Ziemeļamerikā, salīdzinot ar 9.7% un 4.8% pasaules iedzīvotāju [9]. Analizējot labojumus Vikipēdijas rakstos no 2001. līdz 2010. gadam, 1% visbiežākie ierakstu veicēji uzrakstīja 77% satura [10].

Korpusos tam ir vairākas praktiskas implikācijas gan sintaksē, gan semantikā. Piemēram, Vikipēdijas autors Brians Hendersons (*Bryan Henderson*) 15 gados veica 90 tūkstošus labojumu, kur lielākā daļa izmaiņu ir no "comprised of" uz "comprised", kaut gan abas formas tiek pieņemtas un citos rakstiskos avotos "comprised of" ir izplatītāks (1.1 attēls). Tāpat BERT biežāk asociē

cilvēkus ar invaliditāti ar negatīva sentimenta vārdiem un vairāki darbi to sasaista ar treniņu datu kopu īpašībām [8]. Pētīt negatīvu sentimentu valodu korpusos ir svarīgi, jo kompānijas reputācija ciestu, virtuālajam asistentam sniedzot atbildes ar negatīviem steriotipiem klientu apkalpošanas jomā.



1.3. att. **Uz x ass attēloti gadi, uz y ass – cik procentu no visiem vārdiem, kas ietverti angļu valodā rakstīto grāmatu korpusā (English 2019), ir "comprised of" (zilā līnija) un "comprised" (sarkanā līnija)? [11]**

Ar to tiek pierādīts, ka tekstu nav radījuši nejauši izvēlēta izlase cilvēku, tāpēc teksts nav neitrāls. Tiek paredzēts, ka virtuālos asistentus izmantos plašāks cilvēku loks nekā šobrīd internetā publicēto tekstu autori, tāpēc ir svarīgi, lai treniņdatos ir atbilstoši pārstāvēta potenciālo lietotāju valoda.

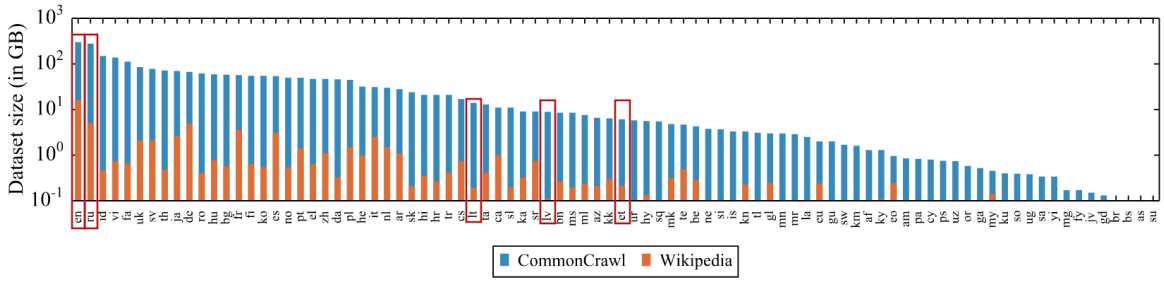
Latviešu marķieru daļa Common Crawl 100 korpusā ir atkarīga no daudziem faktoriem, tostarp latviešu satura daudzuma tīmeklī un korpusa konstruēšanā izmantotās izlases metodikas. Iespējams, ka latviešu teksta saturs korpusā ir pārāk vai nepietiekami pārstāvēts, salīdzinot ar tā izplatību tīmeklī vai proporcionāli latviešu valodā runājošo īpatsvaram.

Kas padara valodu par maz-resursu? Mazāks skaits vārdu un teikumu datu kopās, tātad mazāks skaits tokenu uz kuriem trenēt daudzvalodu jēdzientelpu modeli. Piemēram, Common Crawl-100 korpusā, uz kura trenēts XLM-R modelis, svahili un urdu valodās ir 275M un 730M tokenu attiecīgi, darbā izmantotajās - lietuviešu, latviešu, igauņu - ir 1835M, 1198M un 843M tokenu attiecīgi (1.3 tabula), tātad šīs datukopas kontekstā tās var uzskatīt par maz-resursu.

1.3. tabula

Common Crawl-100 valodas un statistika: valodu saraksts ar marķieru (*tokens*) skaitu (miljonos) un datu izmēru gibibaitos (GiB) katrai valodai

ISO kods	Valoda	Marķieri (M)	Izmērs (GiB)
en	angļu	55608	300.8
ru	krievu	23408	278.0
lt	lietuviešu	1835	13.7
lv	latviešu	1198	8.8
et	igauņu	843	6.1
ur	urdu	730	5.7
sw	svahili	275	1.6

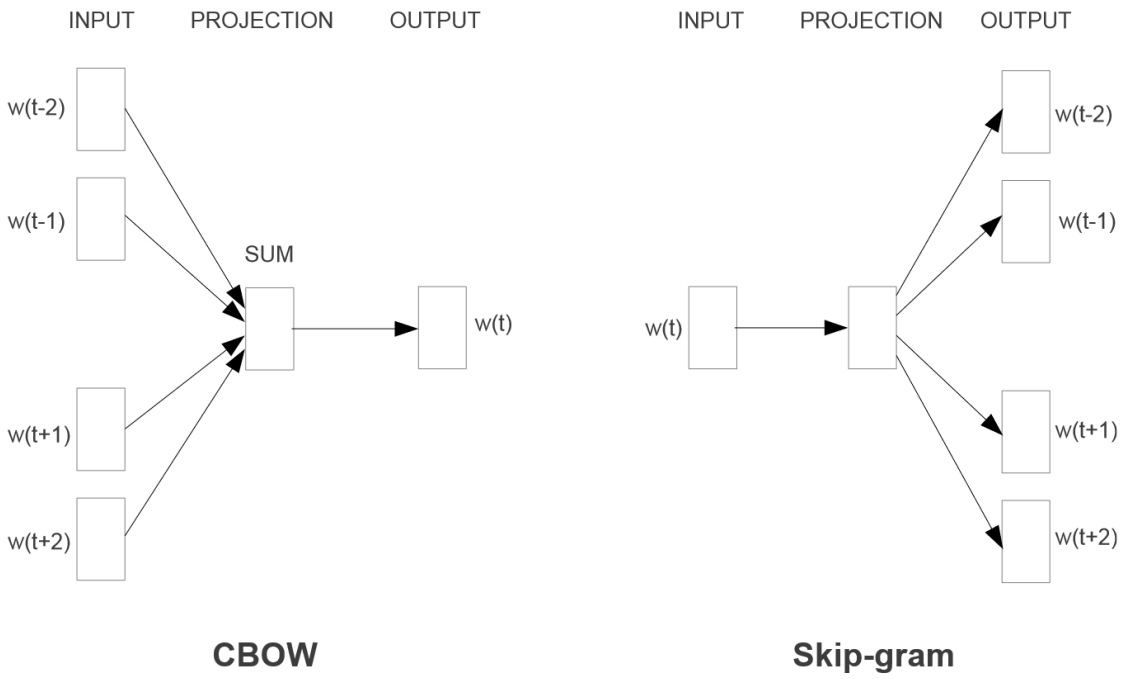


1.4. att. **Datu apjoms GiB (logaritmiskā skalā) valodām Wiki-100 korpusā, ko izmanto mBERT un XLM-100, un Common Crawl-100, ko izmanto XLM-R. Common Crawl-100 palielina datu apjomu par vairākām kārtām, jo īpaši maz-resursu valodās [12] Grafikā ir iekļautas piecas konkrētas šajā darbā izmantotās valodas (atzīmētas ar sarkaniem taisnstūriem) parādīšanās secībā: angļu, krievu, lietuviešu, latviešu, igauņu.**

1.5. Jēdzientelpu modeļu apmācība

Jēdzientelpas no teksta korpusa iegūst ar neironu tīkliem, kuri uztver kontekstu no tuvākajiem vārdiem tekstā.

Continuous Bag-of-Words (CBOW) un *Continuous Skip-gram Model* ir divas neironu tīklu modeļu arhitektūras jēdzientelpu izveidei balstoties uz teksta korpusa. Metožu priekšrocība ir tajā, ka nav nepieciešama anotēta treniņu datu kopa, trenēšanai izmanto lielus teksta korpusus. CBOW modelī apkārt esošos vārdus izmanto vidū esošā vārda paredzēšanai. Skip-gram modelī vārda vektoru izmanto konteksta paredzēšanai (1.5 attēls).



1.5. att. **CBOW un Skip-gram modeļu arhitektūra [4].**

1.5.1. Continuous Bag-of-Words

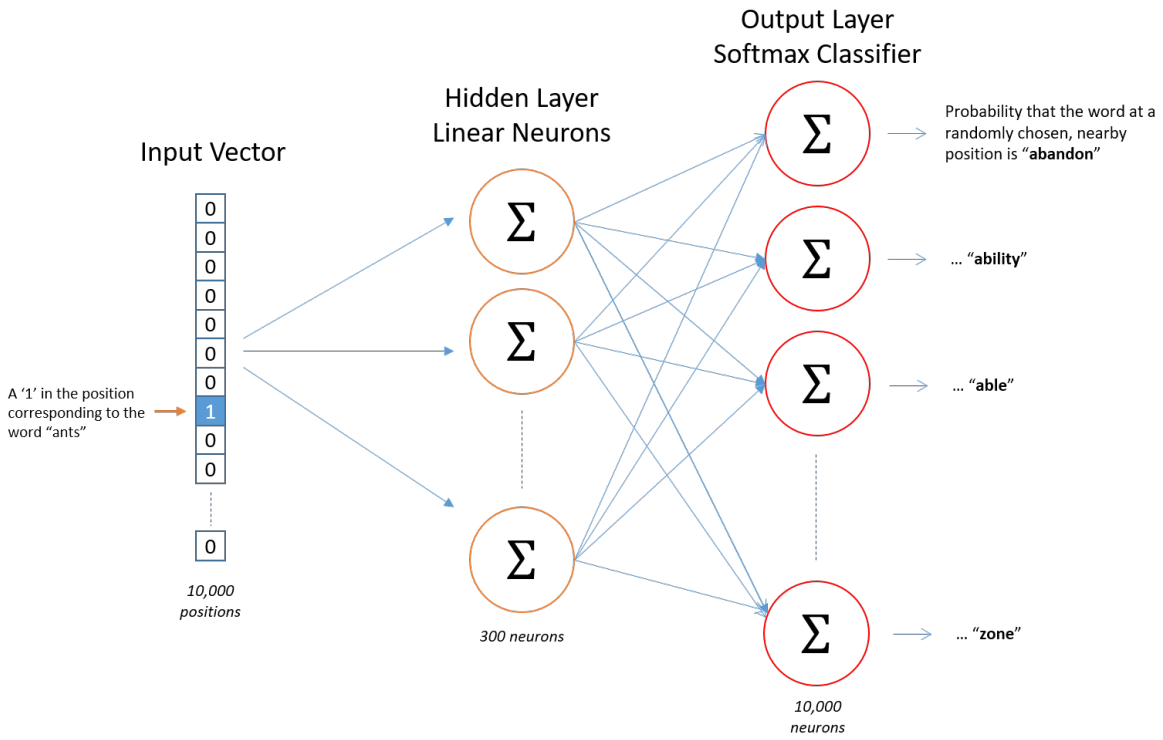
Bag-of-Words (BOW) apzīmē vārdu grupu nesaglabājot kārību. Vienā izlasē (bag) vārda tuvums mērķa vārdam konkrētā izlasē nav tik svarīgs, atkārtojot procesu uz korpusa no konteksta tāpat tiks sīkāk (granulētāk) izšķirti svari tuvākajiem vārdiem, piemēram, Rīga un Latvija būs tuvumā 1000 reizes biežāk nekā Rīga un sniegs.

CBOW (Continuous Bag-of-Words) metodē neironu tīkls mēģina uzminēt esošo (vidējo) vārdu no n iepriekšējiem un n nākošajiem vārdiem. Procesu atkārtojot, vārdiem, kas bieži parādās vienā kontekstā, būs līdzīgi vektori. Pēc izkliedētības (*distributional*) hipotēzes vārdi, kas atrodas līdzīgos kontekstos, ir ar līdzīgu nozīmi [4]. Tāpat kā BOW modelī, CBOW vārdu secība neietekmē projekciju. Nepārtrauktība (*continuuity*) modelī rodas no tā, ka izmanto nepārtrauktu izkliedētu konteksta reprezentāciju jeb svari starp ievades un projekcijas slāņiem tiek lietoti visiem vārdiem. [4]. CBOW neironu tīkla galvenās daļas ir kartējuma (*embedding*) slānis un tam sekojošs blīvais (*dense*) slānis, kurā katrs slāņa mezgls ir pilnībā savienots ar citu mezglu nākamajā slānī (1.5 attēls).

1.5.2. Continuous Skip-gram Model

Continuous Skip-gram metode ir līdzīga CBOW, tikai tās uzdevums ir paredzēt nevis vienu vārdu no apkārt esošajiem, bet otrādi – no ievades vārdu pa vārdam paredzēt apkārt esošos vārdus. Šīs metodes ideja ir uztrenēt neironu tīklu ar slēpto slāni (*hidden layer*) un iegūt slēptā slāņa svarus, kas patiesībā arī ir vārdu vektori. Vērā ņemamu kaimiņu vārdu skaits – loga izmērs (*window size*) – ir hiperparametrs (1.6 attēls). Modeļa trenēšanas laikā kaimiņu vārdu svara koeficienti nav vienādi: jo tālāk tie atrodas no aplūkotā ievades vārda, jo mazāk ietekmē tā nozīmi, tāpēc tālāk esošajiem vārdiem tiek piešķirti mazāki svara koeficienti [4].

Daudzvalodu valodu modeļi, piemēram, mBERT un XLM, tiek apmācīti uz vairākām valodām, un tas nodrošina efektīvu starp-valodu zināšanu pārnesi. Tie ir ievērojami uzlabojuši veikumu starpvalodu izpratnes (cross-lingual understanding) (*cross-lingual understanding*) uzdevumos, tostarp starpvalodu dabiskās valodas izvedumos (*cross-lingual language inference*), kas ietver sevī semantiskās līdzības noteikšanu starp teikumiem dažādās valodās [12]. Piemēram, salīdzinot teikumu ”the dog is sleeping on the mat” angļu valodā un teikumu ”suns guļ uz paklāja” latviešu valodā, var noteikt, ka šie divi teikumi ir semantiski līdzvērtīgi, neskatoties uz to ka tie ir rakstīti dažādās valodās.



1.6. att. **Skip-gram modeļa arhitektūra. Ievades vektors – vārda vienizcēluma kodējums** $1xn$ **– kur** n **ir vārdu skaits; slēptais slānis –** nxl**, kur** l **ir loga izmērs; Softmax slānis –** $1xl$**; Izvades slānis –** $1xn$ **[13].**

2. NODOMU NOTEIKŠANA

Nodoms ir mērķis, kas lietotājam ir padomā, rakstot jautājumu. Nodomu noteikšana ir lietotāja ievades teksta klasifikācija tam piešķirot visvarbūtīgāko nodomu no iepriekš definētu nodomu kopas [14]. Piemēram, klasificējot lietotāja nodomu kā vilciena atiešanas laiks, čatbots var sniegt lietotājam nepieciešamo atbildi no vilcienu grafika (2.1 tabula). Dabiskā valodā ir vairāki veidi kā izteikt vienu un to pašu nodomu (2.2 tabula).

2.1. tabula

Lietotāja ievada un nodoma piemērs

Ievads	Cikos ir nākošais vilciens no Rīgas uz Siguldu?
Nodoms	vilciena atiešanas laiks

2.2. tabula

Lietotāja dažādi ievadi ar vienu nodomu [15]

Nodoms	Ievads
switchLightOn	Ieslēdz gaismu
switchLightOn	Istabā ir pārāk tumši, vai vari to izlabot?

Pirms mašīnmācīšanās nodomi tika noteikti ar šabloniem (*pattern-based recognition*), bet izveidot un uzturēt lielu skaitu šablonu ir darbietilpīgi. Advancētāka pieeja nodomu noteikšanai ir apmācīt neironu tīklu klasifikatoru uz anotētas datu kopas – lietotāju ievades tekstiem un atbilstošajiem klientu apkalpošanas speciālista identificētajiem lietotāja nodomiem. Ierobežotās apmācību kopas dēļ dialogsistēmas/virtuālie asistenti var atbildēt uz ierobežotu jautājumu klāstu, piemēram, aptverot bieži uzdotos jautājumus (FAQ – *Frequently Asked Questions*) [14].

Lai arī jaunāko valodas modeļu, piemēram, GPT-3, izvades teksti lietotājam rada iespaidu par tekošu valodu, pastāv neparastās ielejas (*uncanny valley*) efekts, kurā novērotā plūstošā atbildes valoda rada ekspektācijas, kuras virtuālie asistenti nevar attaisnot un izraisa neapmierinātību [16]. Tāpēc klienta nodoma noteikšana ir svarīga, lai nodrošinātu patīkamu lietotāja pieredzi.

Jāpiebilst, ka labuma gūšanai no nodomu noteikšanas automatizācijas nav nepieciešams pārklāt 100% lietotāju pieprasījumu. Veiksmīgas izmantošanas piemērs telekomunikāciju industrijā

validācijā izmantoja 1732 klientu pieprasījumu datu kopu anotētu ar attiecīgajiem nolūkiem. Šajā gadījumā divi visbiežākie nodomi ir rēķina atlikšana (356 pieprasījumi; 21% datu kopas) un nokavēta rēķina maksājuma apstiprināšana (207 pieprasījumi; 12% datu kopas). Trīs mēnešus ilgā eksperimentālā pētījuma tika apstrādāti 14000 lietotāju pieprasījumi. Sākotnējos testos nodomu noteikšana un izvēlētā atbildes veidne bija precīza 90% gadījumu, eksperimenta gaitā iegūtie dati ļāva uzlabot nodomu noteikšanu par 2%, tātad klientu apkalpošanas speciālistiem bija jāveic izmaiņas tikai 8% pieprasījumu rēķinu kategorijā [16].

Tipiski soļi nodomu noteikšanas pielietojumam uzņēmējdarbībā:

1. Atrast visbiežākos pieprasījumu tipus;
2. Sagatavot atbildes veidni (*template*);
3. Nodomu noteikšanas sistēma identificē, vai lietotāja pieprasījums pieder iepriekš definētajiem tipiem un izdod potenciālo atbildi;
4. Klientu apkalpošanas speciālists izvērtē un koriģē atbildi pirms nosūtīšanas;
5. Automātiski uzlabot nodomu noteikšanas sistēmu, balstoties uz speciālista veiktajām korekcijām [16].

2.1. Nodomu noteikšanas korpusi

Viens no svarīgākajiem korpusiem tieši nodomu noteikšanā ir aviokompāniju ceļojumu informācijas sistēmu (ATIS – *Airline Travel Information Systems*) datu kopa. Tā ir audioierakstu un manuālu transkriptu datu kopa, kas sastāv no cilvēku sarunām ar automatizētām aviolīniju ceļojumu informācijas sistēmām. ATIS datu kopa nodrošina lielu ziņojumu un ar tiem saistīto nodomu skaitu, ko plaši izmanto kā novērtējuma (*benchmark*) datu kopu klasifikatoru apmācībai nodomu noteikšanā [17].

SNIPS – *Spoken Natural Language Interaction for Personal Assistant*) datu kopā ir 16 000 ievadi, kas sadalīti septiņos nodomos: SearchCreativeWork, GetWeather, BookRestaurant, PlayMusic, AddToPlaylist, RateBook, SearchScreeningEvent [18].

Lietotāja dabiskās valodas ievads tiek pārveidots jēdzientelpā, un tad pēc tam klasifikators nosaka lietotāja nodomu.

Dabisko valodu saprašanas (Natural Language Understanding (NLU)) uzdevumi iedalās divās daļās: nodomu noteikšana un parametru *slots*) iegūšana no ievadiem *slot filling*) (2.3 tabula) [18]. Šajā darbā tiks apskatīts pirmais solis – nodomu noteikšana.

2.3. tabula

"Weather" datu kopa, uz kuras tika apmācīts virtuālais asistents, kas apstrādā lietotāja vaicājumus par laikapstākļiem [18]

Nodoms	Parametrs	Piemēri
ForecastCondition	region	Is it cloudy in Germany right now?
	country	Is South Carolina expected to be sunny in 2 hours?
	datetime	Is there snow in Paris?
	locality	Should I expect a storm near Mount Rushmore?
	condition	
	point of interest	

Korpusi ir paredzēti izmantošanai dabiskās valodas izpratnes uzdevumiem – nodomu noteikšanai un entītiju atpazīšanai.

Ask Ubuntu korpuss sastāv no 162 jautājumiem un atbildēm, kas iegūti vietnē `https://askubuntu.com`. Šis korpuss ir anotēts ar pieciem dažādiem nolūkiem: MakeUpdate, SetupPrinter, ShutdownComputer, SoftwareRecommendation un None. Papildus šiem nolūkiem korpusā ir iekļauti arī trīs entītiju veidi: printeris, programmatūra un versija, tomēr entītiju noteikšana ir ārpus šī darba tvēruma *scope*) [19]. Pateicoties konkrētiem nodomiem un entītiju veidiem, izstrādātāji var koncentrēties uz precīzāku modeļu izveidi konkrētiem uzdevumiem, piemēram, printera iestatīšanai vai programmatūras atjaunināšanai.

Ask Ubuntu korpuss ir noderīgs resurss Ubuntu un citu Linux operētājsistēmu lietotāju pieredzes uzlabošanai. Tā kā Ubuntu ir pieejams daudzās valodās virutālā asistenta spēja atbildēt uz Ubuntu jautājumiem lietotāja izvēlētajā valodā var ievērojami palielināt operētājsistēmas pieejamību un lietojamību. Piemēram, trenējot virtuālos asistentus uz šī korpusa izstrādātāji var

izveidot lietotājiem draudzīgas saskarnes *interfaces*).

Tīmekļa lietojumprogrammu korpuss (*Web Applications Corpus*), turpmāk "webapps" korpuss ir vērtīgs resurss virtuālo asistentu izstrādei, kas var palīdzēt lietotājiem orientēties un novērst ar tīmekļa lietojumprogrammām saistītas problēmas. Ar 89 jautājumiem un atbildēm, kas iegūtas vietnē `https://webapps.stackexchange.com`, šis korpuss aptver dažādas tēmas un scenārijus, ar kuriem lietotāji var saskarties, izmantojot tīmekļa lietojumprogrammas. Korpusam ir astoņas anotācijas – ChangePassword, DeleteAccount, DownloadVideo, ExportData, FilterSpam, FindAlternative, SyncAccounts un None –, kā arī trīs entītiju veidi: WebService, OS un Browser [19]. Uz šī korpusa apmācīti mašīnmācīšanās modeļi var noteikt lietotāju nodomus un sniegt atbildes, tādā veidā uzlabojot lietotāja pieredzi ar tīmekļa lietojumprogrammām.

Chatbot korpuss tika iegūts no Telegram virtuālā asistenta Minhenes sabiedriskajam transportam. Tas satur 206 jautājumus, kas tika anotēti divos nolūkos – DepartureTime un FindConnection –, kā arī piecus entītiju veidus: StationStart, StationDest, Criterion, Vehicle un Line [19]. Šis korpuss ir īpaši noderīgs, lai izstrādātu virtuālos asistentus, kas palīdz lietotājiem orientēties sabiedriskajā transportā, jo tas nodrošina skaidru un kodolīgu jautājumu un entītiju kopumu šajā jomā.

Askubuntu, Webapps un Chatbot korpusi tika izveidoti, lai novērtētu komerciāli pieejamo NLU servisu precizitāti Braun et al. rakstā [19], un ir pieejami `https://github.com/sebischair/NLU-Evaluation-Corpora`. Darbā tika izmantots uz šī korpusa balstīta datu kopa, kas tika izmantota nodomu noteikšanai [14] rakstā un ir pieejama `https://github.com/tilde-nlp/NLU-datasets`. Šī datu kopa ir iztulkota latviešu, krievu, igauņu un lietuviešu valodās, kā arī attīrīta no entītijām, atbildēm uz lietotāju jautājumiem un lietotājvārdiem.

3. DAUDZVALODU NODOMU NOTEIKŠANA

Daudzvalodu nodomu noteikšana ir lietotāja vaicājumu nolūka identificēšana dažādās valodās. Šajā sadaļā tiks dots ieskats trīs pieejās daudzvalodu nodomu noteikšanas modeļu apmācībai un testēšanai, tās pamatojot ar pieejamo teorētisko literatūru par šo tēmu:

1. apmācība vienā valodā, un testēšana tajā pašā valodā, piemēram, apmācība latviešu valodā, un testēšana arī latviešu valodā;

2. apmācība visās valodās kopā, testēšana vienā valodā, piemēram, apmācībā izmantojot datu kopu, kurā angļu, latviešu, krievu, igauņu, lietuviešu datu kopas ir apvienotas vienā, testēšana latviešu valodā;

3. apmācība angļu valodā, testēšana ne-angļu valodā.

Katrai no trijām pieejām ir savas priekšrocības un ierobežojumi, un pieejas izvēle ir atkarīga no konkrētajām uzdevuma prasībām. Apmācība un testēšana vienā un tajā pašā valodā var nodrošināt augstāku precizitāti, savukārt, apmācot visas valodas kopā, var izveidot vienu modeli vairākām valodām, taču dažās valodās var būt zemāka precizitāte. Apmācība angļu valodā un testēšana valodās, kas nav angļu valoda, ir noderīgas, ja ir ierobežoti resursi citām valodām un ir sagaidāms, ka modelis labi darbosies angļu valodā. Pieejas izvēlei jābūt balstītai uz uzdevuma īpašajām prasībām un resursiem.

3.1. Apmācība un testēšana vienā valodā

Modeļa apmācība un testēšana vienā un tajā pašā valodā ir piemērota, ja paredzams, ka nodomu noteikšanas modelis konkrētajā valodā darbosies ar labi precizitāti. Vairāki pētījumi ir parādījuši, ka apmācība un testēšana vienā un tajā pašā valodā var nodrošināt lielāku nodomu noteikšanas modeļu precizitāti.

Pētījumā uz XTREME datu kopas [20] tika parādīts, ka, apmācot mBERT modeli [21] ne-angļu valodas nodomu noteikšanai uz datiem tajā pašā valodā var sasniegt par 17–20% augstāku precizitāti, nekā apmācot uz datiem angļu valodā. Pētījumā aplūkotas 40 valodas no 12 saimēm, un secināts, ka rezultāti ir labāki indo-eiropiešu valodu saimei, tāpēc ka citām saimēm var pastāvēt tokenizācijas grūtības.

H. Li un citi [22] izmantoja savā pētījumā paštaisītu datu kopu ar izteikumiem sešās valodās (angļu, spāņu, franču, vācu, hindi un taju). Vidējā precizitāte ne-angļu valodām sasniedza 78% gadījumā, kad apmācība notika vienā valodā, 80% – gadījumā, kad apmācība notika visās valodās, un 66% – kad apmācība notika tikai angļu valodā. Tika parādīts, ka XLM modelis [12] uz šīs datu kopas ļauj sasniegt par 10–11% augstāku precizitāti, nekā XLU modelis [23].

Citā pētījumā autori izmantoja daudzvalodu (angļu, japāņu) modeli un parādīja, ka tas ir efektīvāks, ja tikai daļa no modeļa tiek izmantota abās valodās, bet otra daļa ir specifiska katrai valodai. Apmācot un testējot šo modeli vienā un tajā pašā valodā, nodomu klasifikācijas precizitāte uzlabojās par 0.5–2%, salīdzinot ar modeļa apmācību uz visām valodām [24]. Pētījumā izmantotā datu kopa sastāvēja no dialoga replikām un jautājumiem angļu un japāņu valodās [25].

Priekšrocības šādai pieejai ir iespēja ieviest nepārtrauktu attīstību (*continuous integration*), kurā ienākošie lietotāju ievades teksti un nodomi tiek izmantoti papildus apmācībai, izolējot efektus vienā valodā. Tomēr ļoti daudzās valodās datu kopas, ko var izmantot neironu tīkla apmācībā, ir salīdzinoši mazas, kā parādīts 1.4 att.. Izplatīta stratēģija nepietiekama apjoma treniņdatu problēmas risināšanai ir ievākt vairāk datu un apmācīt katru vienvalodu nodomu noteikšanas modeli atsevišķi, taču tas ir dārgi un resursietilpīgi. Bet izmantojot vienu daudzvalodu modeli (pieeja, kas aprakstīta nākamajā apakšnodaļā) zināšanas no liel-resursu valodas tiek pārnestas uz maz-resursu mērķvalodu [26].

3.2. Apmācība uz visām valodām, testēšana vienā valodā

Otrā pieeja ir modeļa apmācība daudzvalodu datu kopā, kas ietver visas interesējošās valodas, un tā testēšana konkrētā valodā. Šī pieeja ir noderīga, ja paredzams, ka modelis dažādās valodās darbosies pietiekami labi un mērķis ir izveidot vienu nodomu noteikšanas modeli, kas spēj apstrādāt vairākas valodas. Cilvēki sagaida precīzu mijiedarbību ar virtuālajiem asistentiem neatkarīgi no viņu lietotās valodas, tomēr mērogot (*scaling*) nodomu noteikšanu uz vairākām valodām ir izaicinājums. Tipisks daudzvalodu nodomu noteikšanas risinājums ir transformeru daudzvalodu modeļu izmantošana, piemēram, mBERT un XLM-RoBERTa. Atšķirībā no monolingvāliem modeļiem, daudzvalodu modeļi tiek apmācīti uz daudzvalodu datu kopām.

2022. gada pētījumā [27] tika parādīts, ka modeļu precizitāti var uzlabot, mākslīgi palielinot

treniņkopas izmēru. Tomēr treniņkopas palielināšana palielināja arī pārklājumu ar testa kopu, kas pārvērtēja (*overestimating*) patieso precizitāti. Izveidotais daudzvalodu modelis sasniedza 93.4% vidējo precizitāti nodomu noteikšanā uz MASSIVE datu kopas, kas satur paralēlus anotētus datus (angļu izteikumi ar tulkojumiem 51 valodā) [28].

Citā pētījumā tika secināts, ka pie nemainīgas modeļa arhitektūras palielinot apmācībā izmantoto valodu skaitu līdz pat 100 modeļa efektivitāte mazāk populārām valodām sākumā pieaug, bet pēc tam sāk samazināties. Tas tiek dēvēts par "daudzvalodības lāstu" (*curse of multilinguality*), ko var novērst, palielinot kopējo neironu skaitu. Šā pētījuma ietvaros salīdzinot mBERT un XLM-RoBERTa jēdzientelpu daudzvalodu klasifikāciju, tika secināts, ka XLM-RoBERTa ir precīzāks līdz pat 23% maz-resursu valodās (svahili un urdu) [12].

Vēl vienā pētījumā parādīts, ka, izmantojot kopīgu modeli angļu, hindi un bengali valodu mBERT jēdzientelpām precizitāte palielinās par $\sim 2\%$, salīdzinot ar atsevišķu apmācību katrā valodā, t.i., individuāliem nodomu noteikšanas modeļiem [29].

Šīs pieejas (apmācība uz visām valodām) priekšrocības ir mazākas modeļa apmācības izmaksas (viens modelis visiem datiem), kā arī maz-resursu valodas var gūt labumu no starp-valodu zināšanu pārneses (*cross-lingual knowledge transfer*), kas raksturīga kopīgam modelim [27]. Taču var parādīties vairākas negatīvas sekas, ja daudzvalodu jēdzientelpu modeli apmāca uz datu kopas, kurā kāda valoda, piemēram, angļu, ir disproporcionāli pārstāvēta (*over represented*) un testējot uz maz-resursu (*low-resource*) valodas, piemēram, latviešu. Pirmkārt, modelim var būt zemāka precizitāte latviešu valodā, kas nozīmē nepareizi klasificētus lietotāja nodomus un lietotāju neapmierinātību ar virtuālo asistentu. Otrkārt, modelis, kas apmācīts uz disbalansētas datu kopas var ciest no katastrofiskas aizmiršanas (*catastrophic interference*) fenomena, kurā modelis "aizmirst" maz-resursu valodu kad tiek iepazīstināts ar jauniem datiem citās valodās, kas noved pie zemas precizitātes un nepieciešamības apmācīt modeli no jauna.

3.3. Apmācība angļu valodā, testēšana valodās, kas nav angļu

Trešā pieeja ietver modeļa apmācību angļu valodā un tā testēšanu valodā, kas nav angļu valoda. Lasītājam var rasties šaubas, kā modelis kaut ko var paredzēt valodā, uz kuras nav apmācīts. Daudzvalodu modeļi tika apmācīti uz 100 dažādām valodām (precīzāk mBERT – 104 valodas,

XLM-Roberta – 100 valodas), kas iemācīja tiem izveidot jēdzientelpas dažādās valodās. Šie modeļi izmanto no daudzvalodu korpusa iemācīto kopējo reprezentāciju (*shared representations*), lai reprezentētu tekstu kā vektoru, kas tālāk tiek izmantots klasifikācijai. Būtībā modelis iemācās reprezentēt tekstu veidā, kas vispārina valodai raksturīgās nianses, tādējādi ļaujot tam strādāt dažādās valodās.

Daudzvalodu modeļi spēj sasniegt pietiekami labus rezultātus, jo ir "iemācījušies" uztvert valodas lietojumu plašā valodu diapazonā. Tomēr klasificējot ne-angļu tekstu var samazināties precizitāte, jo apmācībā modelim nebija pieejami valodai raksturīgie dati. Piemēram, pētījumā ar MASSIVE datu kopu [28] autori apmācīja daudzvalodu modeļus XLM-Roberta un mT5 uz ļoti lielas (1 miljons rindiņu) datu kopas. Veikti dažādi eksperimenti: gan apmācot tikai uz angļu valodas (šajā gadījumā rezultātu variance bija ļoti liela), gan uz visām valodām (iegūti par 25-37% labāki rezultāti, nekā apmācot tikai uz angļu valodas). Arī citā pētījumā [30] parādīts, ka, neatkarīgi no izmantotā modeļa (mBERT, XLM-R, mT5), apmācība uz visām valodām ļauj sasniegt par 1–5% augstāku teikumu klasifikācijas precizitāti, nekā apmācība tikai uz angļu valodas.

Šī pieeja ir noderīga, ja ne-angļu valodām ir ierobežoti treniņkopas resursi un ja paredzams, ka modelis labi darbosies angļu valodā. Tomēr šī pieeja paredz, ka valodu struktūra ir pietiekami līdzīga, lai modelis varētu pārnest zināšanas no angļu valodas uz citām valodām.

3.4. Mašīntulkošana uz angļu valodu

Mašīntulkošana izvēlēta lai apietu šķērsli kurā maz-resursu valodās ir mazāk datu uz kā apmācīt modeli. Ideja ir nevis gaidīt līdz tiks savākti pietiekami daudz datu, bet nodrošināt iespēju ienākt maz-resursu valodas lietotāju tirgū un sākt nodomu noteikšanu jau no pirmās dienas, lietotāju ievadus mašīntulkojot angļu valodā un izmantojot jau eksistējošos klasificēšanas modeļus angļu valodās.

Pieļaujamā nodomu noteikšanas precizitāte ir katra uzņēmuma biznesa plāna ziņā. Nav obligāti, lai tā sasniegtu 100%, jo ir iespējams lietot human-in-the-loop pieeju, kurā noteikto nodomu pārbauda klientu apkalpošanas speciālists vēl pirms lietotājam tiek nosūtīta atbilde [16]. Arī nepilna automatizācija ir noderīga, jo strādniekam novērtēt vaicājuma atbilstību konkrētam nodomam ir vieglāk, nekā izvērtēt kuram no daudziem nodomiem tas atbilst.

Taču mašīntulkošana rada papildu trokšņus un kļūdas, kas var ietekmēt ievades datu kvalitāti un klasifikācijas modeļu veiktspēju. Turklāt mašīntulkošanas rezultātā var tikt zaudētas svarīgas nianses un katrai valodai raksturīgā semantiskā informācija, kas var vēl vairāk pasliktināt ievades datu kvalitāti un apgrūtināt precīzu nodomu noteikšanu.

Mani darbā interesē tieši noskaidrot pielietojamības robežas maz-resursu valodām un kādi kompromisi (*trade-offs*) uzlabo modeļa precizitāti; šajā gadījumā vai modeļa veiktspēja ar mašīntulkošanas troksni atsver nepietiekamos datos. Tāpēc ir svarīgi novērtēt modeļu veiktspēju gan ar oriģinālajiem, gan mašīntulkotajiem ievades datiem un salīdzināt rezultātus, lai labāk izprastu modeļu stiprās puses un ierobežojumus daudzvalodu nodomu noteikšanas uzdevumos.

Lai rezultāti būtu reproducējami, mašīntulkošana no latviešu, krievu, igauņu, lietuviešu valodām uz angļu tika veikta ar [31] rakstā aprakstītajiem modeļiem (3.1 tabula).

3.1. tabula

Modeļi, kas mašīntulkoja attiecīgo valodu uz angļu valodu.

Valoda	Mašīntulkošanas modelis
lv	`https://huggingface.co/Helsinki-NLP/opus-mt-tc-big-en-lv`
ru	`https://huggingface.co/Helsinki-NLP/opus-mt-ru-en`
et	`https://huggingface.co/Helsinki-NLP/opus-mt-tc-big-et-en`
lt	`https://huggingface.co/Helsinki-NLP/opus-mt-tc-big-lt-en`

3.5. Citas pieejas

Visbeidzot, būtu svarīgi izpētīt, kā virtuālais asistents tiek galā ar jauktu valodu vaicājumiem (*code switching*), kas ir izplatīti daudzvalodu vidēs, piemēram, kombinācija latviešu-angļu. Tas ietver pieeju izpēti vairāku valodu identificēšanai un atdalīšanai vienā izteikumā, kā arī efektīvi pielietotas daudzvalodu jēdzientelpas. Piemēram, pētījumā datu kopa, kas sastāv tikai no jauktu angļu un hindi valodu vaicājumiem, bija ar 2% zemāku precizitāti (F1 score) nekā angļu, hindi un jauktu valodu datu kopa, ar ELMO jēdzientelpām [32].

Vēl viens jauktu valodu paņēmiens ir aizvietot izvēlētus vārdus ar to tulkojumiem maz-resursu valodām. Pētījumā [26] tika salīdzināta (a) modeļa apmācība tikai uz angļu valodas datu

kopas un testēšana uz datiem spāņu valodā un (b) modeļa apmācība uz jauktiem angļu un spāņu valodu vaicājumiem. Izmainot pieeju no (a) uz (b), nodomu noteikšanas precizitāte uzlabojās no 73.7% uz 86.5% ar multilingvālām BERT jēdzientelpām un no 60.8% uz 83.9% ar XLM jēdzientelpām. Jaukti vaicājumi tika ģenerēti automātiski aizvietojot vārdus, kas izvēlēti balstoties uz uzmanības slāņa (*attention layer*) aprēķinātajiem rādītājiem (*scores*) uz angļu valodas modeļa, ar to tulkojumiem bilingvālā vārdnīcā [26].

4. REZULTĀTI

Šajā nodaļā ir izklāstīti pētījuma rezultāti, kuru mērķis ir salīdzināt dažādu lietotāju nodomu noteikšanas metožu precizitāti angļu, latviešu, krievu, igauņu un lietuviešu valodās, izmantojot mBERT un XLM-RoBERTa jēdzientelpas un dažādu valodu datus. Nodaļā rezultāti tiek analizēti pēc precizitātes un tiek apskatīti modeļu apmācības grafiki. Tāpat nodaļā ir apskatīta iegūto rezultātu ietekme uz virtuālo asistentu sistēmu izstrādi un uzlabošanu.

4.1 tabula sniedz pārskatu par atšķirībām starp dažādām lietotāju nodomu noteikšanai izmantotajām metodēm, pamatojoties uz apmācībā izmantoto valodu un to, vai korpuss ir bijis mašīntulkots uz angļu valodu. Papildus 4.2 tabulā ir īsi definēti 1a, 1b, 2a, 2b, 3a, 3b metožu atšifrējumi par piemēru ņemot latviešu valodas korpusu, ļaujot interpretēt turpmākās tabulas ar rezultātiem.

4.1. tabula

Apkopojums galvenajām atšķirībām starp 1a, 1b, 2a, 2b, 3a, 3b metodēm attiecībā uz apmācības valodu un ievaddatu valodu.

Abreviatūra	Atšifrējums
1	Treniņdati un testa dati vienā valodā
2	Treniņdati visās valodās, testa dati vienā valodā
3	Treniņdati angļu valodā, testa dati ne-angļu valodās
a	mašīntulkoti angļu valodā
b	oriģinālvalodā

4.2. tabula

Piemērs 1a, 1b, 2a, 2b, 3a, 3b metožu atšifrējumam latviešu valodas datiem.

Abreviatūra	Atšifrējums
1a	Apmācība un testēšana uz latviešu valodas korpusa, kas mašīntulkots uz angļu valodu
1b	Apmācība un testēšana uz latviešu valodas korpusa
2a	Apmācība uz visām valodām, kur ne-angļu dati mašīntulkoti uz angļu valodu, testēšana uz latviešu valodas korpusa, kas mašīntulkots uz angļu valodu
2b	Apmācība uz visām valodām, testēšana uz latviešu valodas korpusa
3a	Apmācība uz angļu valodas korpusa, testēšana uz latviešu valodas korpusa, kas mašīntulkots uz angļu valodu
3b	Apmācība uz angļu valodas korpusa, testēšana uz latviešu valodas korpusa

4.1. Chatbot datu kopa

Tabulā 4.3 ir sniegta informācija par unikālo nodomu skaitu *Chatbot* treniņkopā un testa kopā, kā arī to summa.

4.3. tabula

Unikālo nodomu skaits *Chatbot* treniņkopā un testa kopā.

Nodoms	Treniņkopā	Testa kopā	Σ
FindConnection	57	71	128
DepartureTime	43	35	78
Σ	100	106	206

Nodomu noteikšanas precizitāte *Chatbot* datu kopā ir apkopota pēc izmantotā jēdzientelpu modeļa: mBERT (4.4 tabula) un XLM-RoBERTa (4.5 tabula). Salīdzinot šīs jēdzientelpas, redzams, ka angļu valodā uz XLM-RoBERTa jēdzientelpām precizitāte ir augstāka, bet lietuviešu valodā – zemāka nekā mBERT jēdzientelpām. Kopumā mBERT un XLM-RoBERTa modeļi uzrāda līdzīgus rezultātus.

Igauņu valodā ir vislielākais kritums stratēģijā, kas apmāca tikai uz angļu valodas datiem un testē igauņu valodā, tas atbilst paredzētajam, jo igauņu valoda ir vismazāk pārstāvēta korpusos, uz kuriem apmācīti mBERT un XLM-RoBERTa modeļi. Lai gan intuitīvi varētu šķist, ka modelis, kas apmācīts tikai angļu valodā, slikti spēs klasificēt nodomus citā valodā, rezultāti pierāda pretējo – 3b kolonnā redzami salīdzinoši labi rezultāti uz testa kopām krievu un latviešu valodās (85.85–93.40%). Tas tādēļ, ka daudzvalodu jēdzientelpas angļu datu kopās ir līdzīgas tādu pašu teikumu jēdzientelpām citās valodās.

Vidēji, izmantojot latviešu valodas mašīntulkojumus uz angļu valodu, var sasniegt lielāku precizitāti nekā oriģinālvalodā (1ab–2ab) abās jēdzientelpās, taču krievu valodas korpusa tulkojums uz angļu valodu precizitāti samazināja. Gadījumā, kad modelis apmācīts tikai uz angļu valodas korpusa, precizitāte būtiski uzlabojos testa kopu mašīntulkojot uz angļu valodu, igauņu un lietuviešu valodās precizitātei vidēji pieaugot par 30% un 12% attiecīgi, izņemot krievu valodu, kur precizitāte mazliet samazinās – par 3.8% mBERT un 1% XLM-RoBERTa jēdzientelpām. Šo izņēmumu krievu valodā varētu izskaidrot tās salīdzinoši labā pārstāvētība korpusos, uz kuriem apmācīti mBERT un XLM-RoBERTa modeļi, salīdzinot ar latviešu, igauņu un lietuviešu valodām (1.4 attēls).

Maz-resursu valodās (latviešu, lietuviešu, igauņu) tipiski precizitāte ieguva no apmācības uz datiem, kas mašīntulkoti angļu valodā (1ab). Visās valodās visstabilākā veiktspēja ir metodei, kas apmāca modeli uz visām valodām apvienotām vienā korpusā (2ab), kas nebūtiski atšķiras no tā, vai dati pirms tam ir mašīntulkoti uz angļu valodu.

Att. 4.1 parāda precizitāti apmācības laikā, lietojot mBERT jēdzientelpu un treniņdatus latviešu valodā. Redzams, ka apmācības laiks (epochs) bija nepietiekams (att. 4.1, pa labi), jo trenēšanās pārtrūka pie treniņkopas precizitātes 90%, kad vairums citos grafikos tas sasniedza 100%. Tas vieš cerības, ka palielinot epochs skaitu 1a un 1b scenārijos, izdotos iegūt augstāku precizitāti. Mašīntulkoto datu gadījumā trenēšanās laiks bija pietiekams (att. 4.1, pa labi), un arī precizitāte uz testa kopas ir mazliet augstāka – 91.51% salīdzinot ar 88.67%.

4.4. tabula

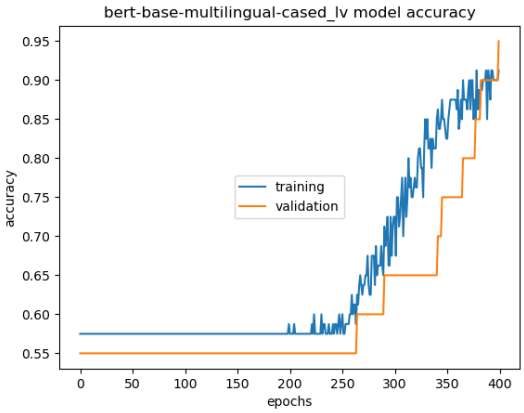
Nodomu noteikšanas precizitāte uz *Chatbot* datukopas ar mBERT modeli, %

languages	1a	1b	2a	2b	3a	3b
en		91.27		89.15		95.05
lv	89.62	94.34	92.69	94.58	93.87	86.08
ru	91.51	92.45	93.16	93.16	92.45	91.98
et	90.57	87.26	91.27	89.39	92.45	62.26
lt	93.87	94.81	91.51	94.10	91.51	74.53
avg	91.39	92.03	92.16	92.08	92.57	81.98
avg lv-lt-et	91.35	92.14	91.82	92.69	92.61	74.29

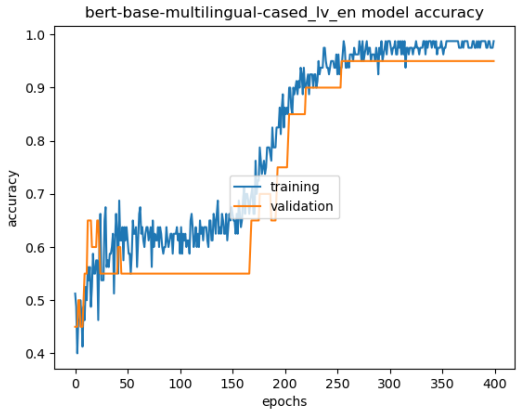
4.5. tabula

Nodomu noteikšanas precizitāte uz *Chatbot* datukopas ar XLM-RoBERTa modeli, %

languages	1a	1b	2a	2b	3a	3b
en		95.28		95.28		96.46
lv	90.33	88.92	93.63	94.34	93.40	87.74
ru	92.45	91.75	91.75	94.58	91.98	91.98
et	88.44	92.69	91.75	88.68	89.86	72.17
lt	90.57	91.04	93.63	94.81	89.39	74.76
avg	90.45	91.93	92.69	93.54	91.16	84.62
avg lv-lt-et	89.78	90.88	93.00	92.61	90.88	78.22

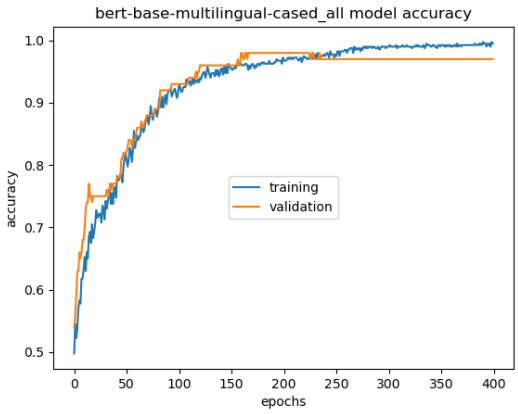


(a) **mBERT latviešu treniņdatu kopa**

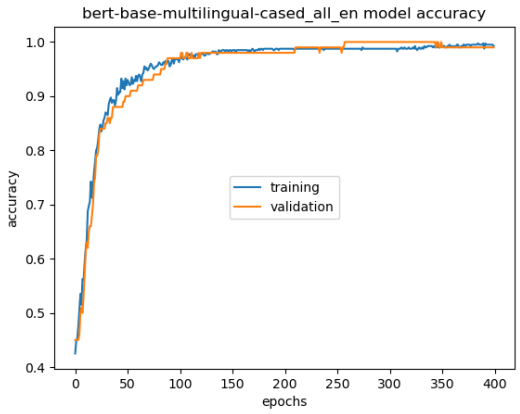


(b) **mBERT mašīntulkoto latviešu treniņdatu kopa**

4.1. att. **caption**

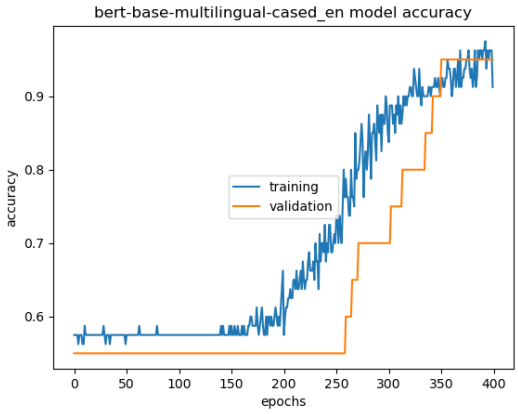


(a) **mBERT apvienotā treniņdatu kopa**

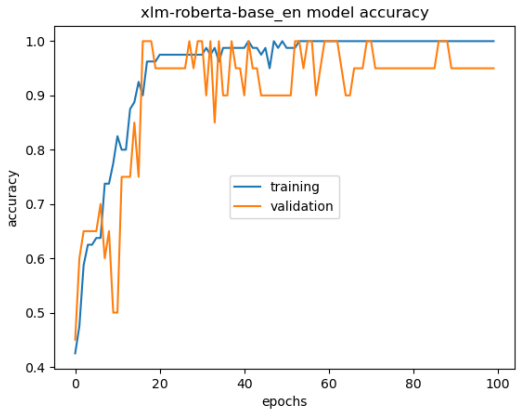


(b) **mBERT apvienotā mašīntulkoto treniņdatu kopa**

4.2. att. **caption**

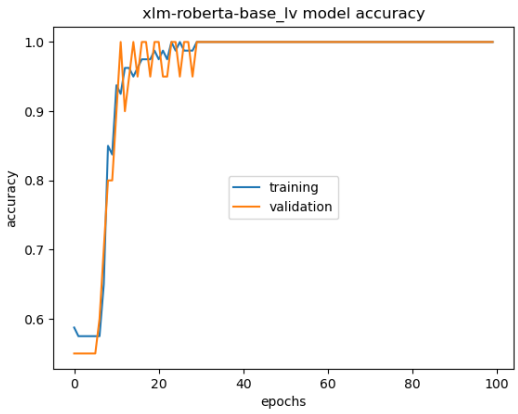


(a) **mBERT angļu treniņdatu kopa**

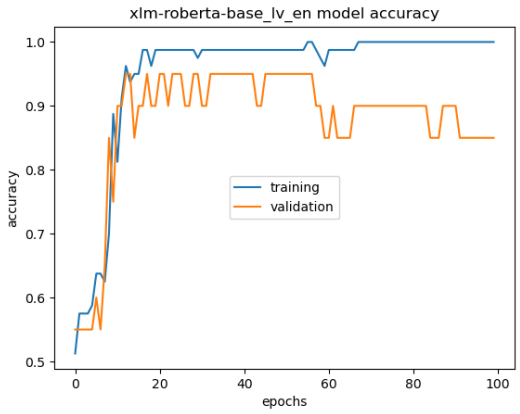


(b) **XLM-RoBERTa angļu treniņdatu kopa**

4.3. att. **caption**

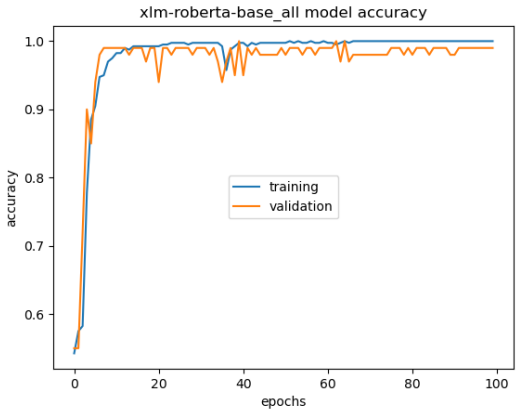


(a) **XLM-RoBERTa latviešu treniņdatu kopa**

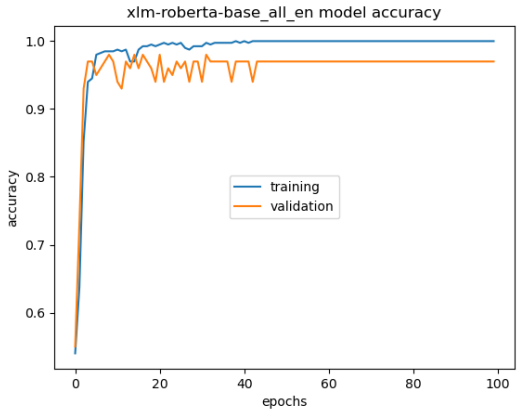


(b) **XLM-RoBERTa mašīntulkoto latviešu treniņdatu kopa**

4.4. att. **caption**



(a) **XLM-RoBERTa apvienotā treniņdatu kopa**



(b) **XLM-RoBERTa apvienotā mašīntulkoto treniņdatu kopa**

4.5. att. **caption**

4.2. Askubuntu datu kopa

4.6. tabula

Unikālo nodomu skaits ”askubuntu” treniņkopā un testa kopā.

Nodoms	Treniņkopā	Testa kopā	Σ
Software Recommendation	17	40	57
Make Update	10	37	47
Shutdown Computer	13	14	27
Setup Printer	10	13	23
None	3	5	8
Σ	53	109	162

Sliktākā nodomu noteikšanas precizitāte Askubuntu datukopā ir 45.87%, tas nozīmē, ka modelis modelis pareizi identificēs lietotāja nodomu mazāk nekā puse gadījumos. Lai gan tā ir salīdzinoši zema precizitāte, ir svarīgi atzīmēt, ka datu kopā ir pieci nodomi un, ja nodomi tiktu izvēlēti nejauši, paredzamā precizitāte būtu tikai 20%. Tāpēc modelis ar 45.87% precizitātes līmeni joprojām darbojas ievērojami labāk nekā random.

Askubuntu datu kopā nodomu noteikšanas precizitāte ir zemāka nekā Chatbot datu kopā tādeļ, ka ir vairāk nodomu ar mazāk piemēriem katram nodomam.

Datu kopā *Askubuntu* 3b metodei vienmēr bija viszemākā nodomu noteikšanas precizitāte gan mBERT, gan XLM-RoBERTa modeļiem. Turpretim visaugstākā veiktspēja nodomu noteikšanas precizitātē tika sasniegta ar 2b metodi, kam sekoja 3a un 1b metode gan mBERT, gan XLM-RoBERTa modeļiem.

4.7. tabula

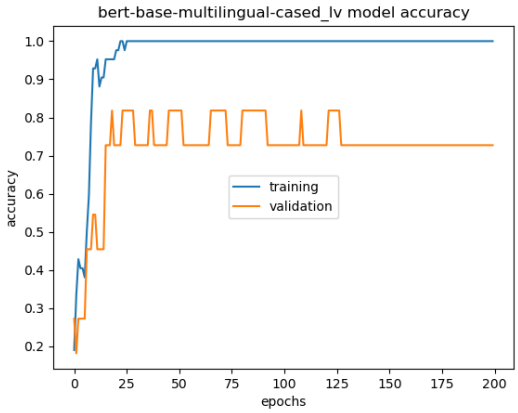
Nodomu noteikšanas precizitāte uz *Askubuntu* datukopas ar mBERT modeli, %.

languages	1a	1b	2a	2b	3a	3b
en		82.57		83.26		82.57
lv	79.36	82.34	80.50	82.34	79.36	54.13
ru	78.90	78.67	75.92	79.59	79.59	64.22
et	80.96	83.26	79.82	77.06	80.05	57.80
lt	80.28	78.67	80.73	77.52	78.44	55.50
avg	79.87	81.10	79.24	79.95	79.36	62.84
avg lv-lt-et	80.20	81.42	80.35	78.98	79.28	55.81

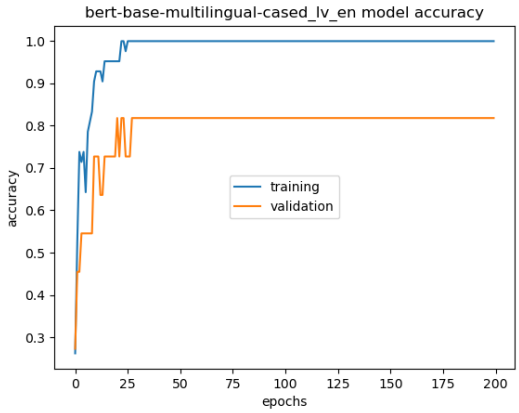
4.8. tabula

Nodomu noteikšanas precizitāte uz *Askubuntu* datukopas ar XLM-RoBERTa modeli, %

languages	1a	1b	2a	2b	3a	3b
en		73.85		78.90		71.10
lv	68.58	64.91	78.44	75.92	72.94	44.95
ru	66.06	70.87	76.61	75.92	66.51	51.83
et	63.53	73.17	75.92	77.98	67.66	52.06
lt	69.50	61.70	77.52	75.00	67.89	44.72
avg	66.92	68.90	77.12	76.74	68.75	52.94
avg lv-lt-et	67.20	66.59	77.29	76.30	69.50	47.25

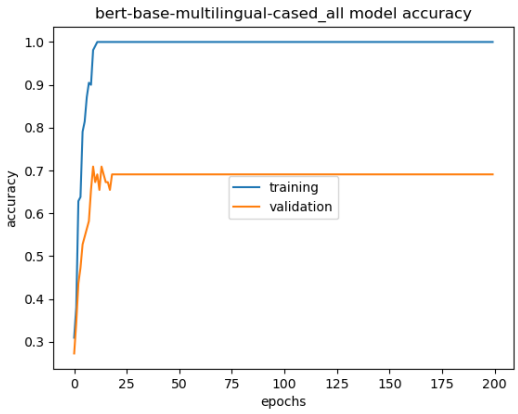


(a) **mBERT latviešu treniņdatu kopa**

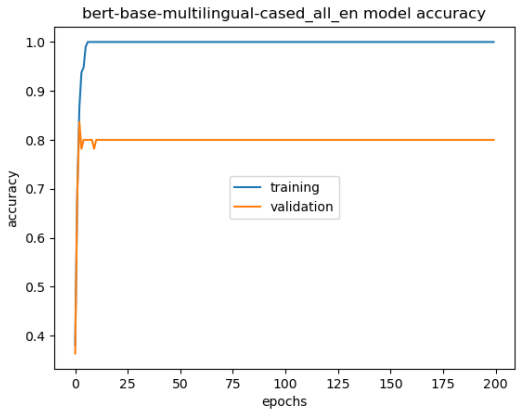


(b) **mBERT mašīntulkoto latviešu treniņdatu kopa**

4.6. att. **caption**

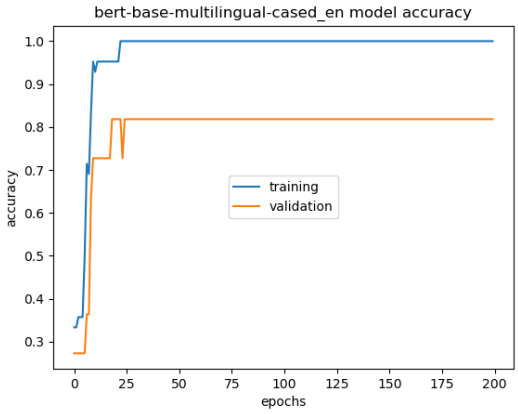


(a) **mBERT apvienotā treniņdatu kopa**

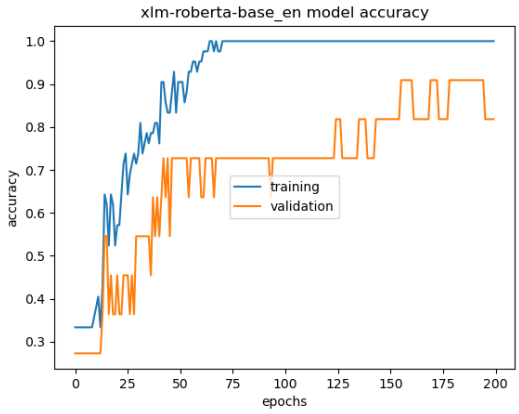


(b) **mBERT apvienotā mašīntulkoto treniņdatu kopa**

4.7. att. **caption**

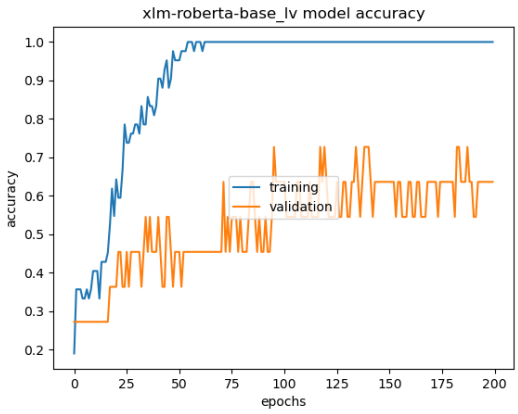


(a) **mBERT angļu treniņdatu kopa**

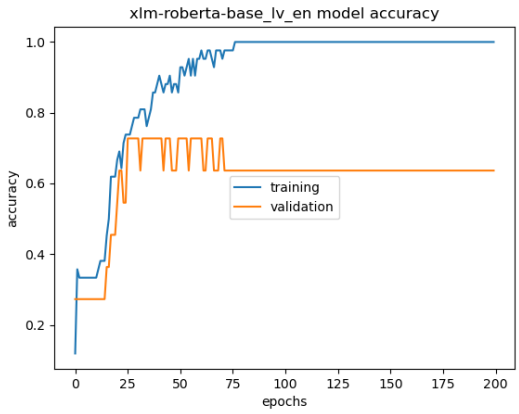


(b) **XLM-RoBERTa angļu treniņdatu kopa**

4.8. att. **caption**

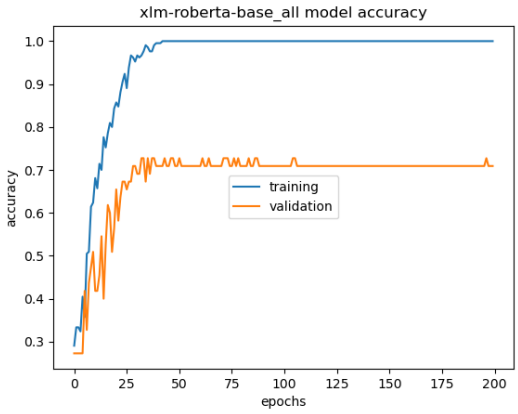


(a) **XLM-RoBERTa latviešu treniņdatu kopa**

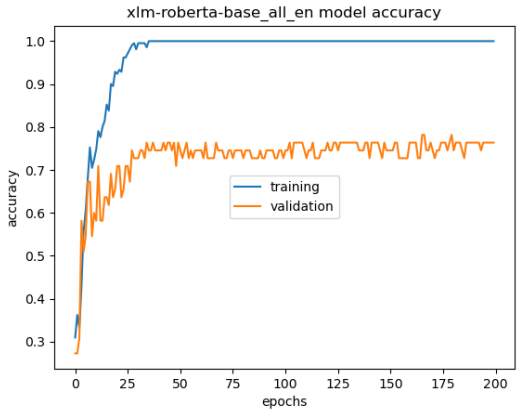


(b) **XLM-RoBERTa mašīntulkoto latviešu treniņdatu kopa**

4.9. att. **caption**



(a) **XLM-RoBERTa apvienotā treniņdatu kopa**



(b) **XLM-RoBERTa apvienotā mašīntulkoto treniņdatu kopa**

4.10. att. **caption**

4.3. Webapps datu kopa

Webapps datu kopā ir nodoms ar tikai vienu piemēru, kas izraisa "ValueError: The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2." Tāpēc nodomi ar mazāk nekā trīs piemēriem tika apvienoti vienā nodomā "Other". Tas atbilst reālam pielietojumam industrijā, kur nodomi nav vienlīdzīgi pārstāvēti – piemēram, starp 115 dažādiem nodomiem divi visbiežākie nodomi kopā pārstāv 33% datu kopas [16] – un ir svarīgi spēt atsijāt nodomus, kurus jāapstrādā klientu apkalpošanas speciālistam – cilvēkam.

Salīdzinot abas tabulas, redzams, ka XLM-RoBERTa vairumā gadījumu ir zemāka precizitāte nekā nekā mBERT, bet visaugstākajos rezultātos – 2a un 2b metodēs XLM-RoBERTa ir pārāks. Šajā gadījumā spēja vienā metodē sasniegt konsekventi labus rezultātus ir svarīgāka nekā viduvēji rezultāti visās, jo pastāv iespēja izvēlēties un lietot vislabāko.

Angļu valodā 1b XLM-RoBERTa ir ar precizitāti 42.37%, kas varētu būt nejaušība, jo 3b arī apmāca un trenē uz tieši tāda paša korpusa un ir 64.41%, tāpēc to var neņemt vērā. Ar mBERT jēdzientelpu šāda problēma angļu valodai nebija novērota (apmācot modeli ar 1b metodi tika iegūta precizitāte 64.41%).

Atšķirībā no pārējām datu kopām nodomu noteikšanas precizitāte igauņu valodā nav sliktāka, salīdzinot ar citām maz-resursu valodām – igauņu valodai maksimālā precizitāte sasniedz gandrīz 75%, bet latviešu un lietuviešu valodām tā ir attiecīgi 73% un 80%. Tādā veidā maz-resursu valodas uz šīs datu kopas ir ar līdzīgu precizitāti kā krievu valodai, kura ir daudz plašāk pārstāvēta tekstu korpusos.

4.9. tabula

Unikālo nodomu skaits "webapps" treniņkopā un testa kopā. Ar treknrakstu iezīmētas nodomi, kuri ir pietiekami pārstāvētas, pārējie nodomi tika apvienoti vienā jaunā nodomā: "Other"

Nodoms	Treniņkopā	Testa kopā	Σ
Find Alternative	**7**	**16**	23
Delete Account	**7**	**10**	17
Filter Spam	**6**	**14**	20
Sync Accounts	**3**	**6**	9
Change Password	2	6	8
None	2	4	6
Export Data	2	3	5
Download Video	1	0	1
Σ	30	59	89

4.10. tabula

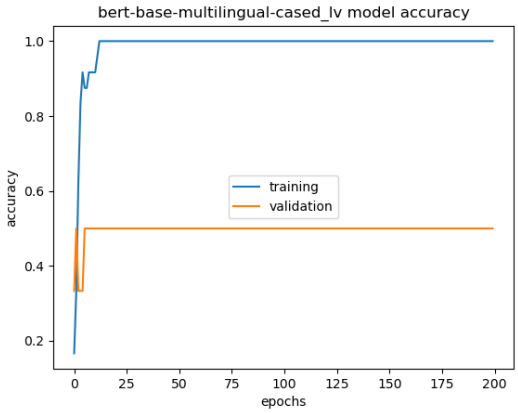
Nodomu noteikšanas precizitāte uz *Webapps* datukopas ar mBERT modeli, %.

languages	1a	1b	2a	2b	3a	3b
en		66.53		64.83		63.14
lv	69.07	62.29	71.19	66.10	62.71	33.47
ru	66.95	68.64	75.00	72.46	62.71	44.49
et	60.17	59.32	74.58	66.10	64.41	38.56
lt	62.71	58.90	72.03	66.95	61.02	33.90
avg	64.72	63.14	73.20	67.29	62.71	42.71
avg lv-lt-et	63.98	60.17	72.60	66.38	62.71	35.31

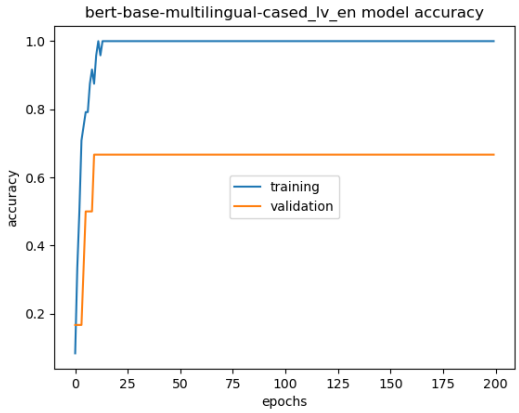
4.11. tabula

Nodomu noteikšanas precizitāte uz *Webapps* datukopas ar XLM-RoBERTa modeli, %.

languages	1a	1b	2a	2b	3a	3b
en		50.85		66.53		58.05
lv	37.29	50.00	66.95	70.34	44.49	29.24
ru	41.53	51.27	64.41	62.29	44.07	33.90
et	37.29	46.61	64.83	65.68	47.88	30.51
lt	58.47	47.46	69.07	65.68	58.05	27.54
avg	43.64	49.24	66.31	66.10	48.62	35.85
avg lv-lt-et	44.35	48.02	66.95	67.23	50.14	29.10

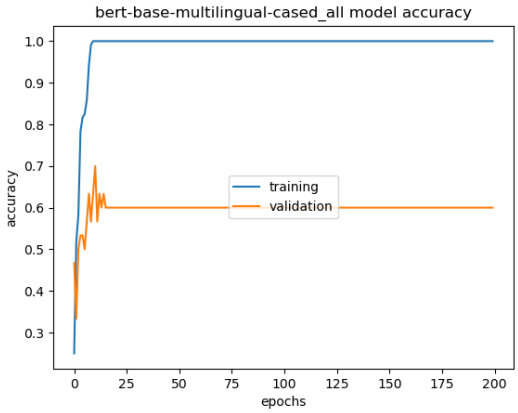


(a) **mBERT latviešu treniņdatu kopa**

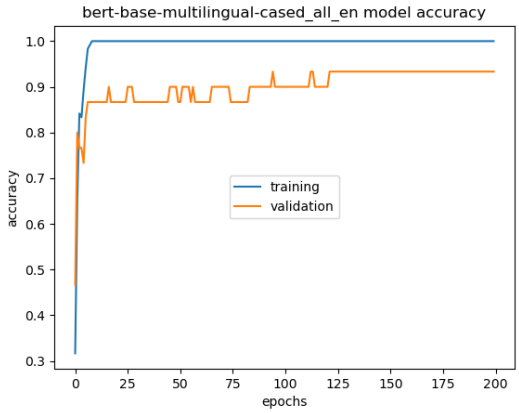


(b) **mBERT mašīntulkoto latviešu treniņdatu kopa**

4.11. att. **caption**

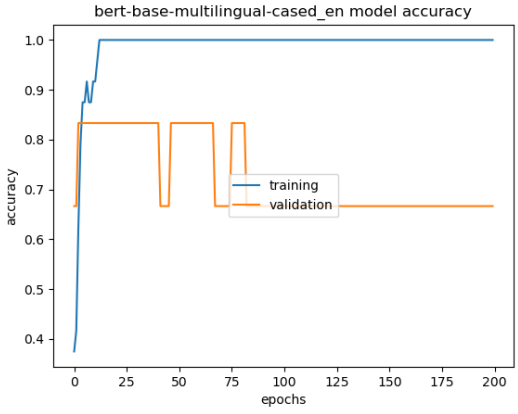


(a) **mBERT apvienotā treniņdatu kopa**

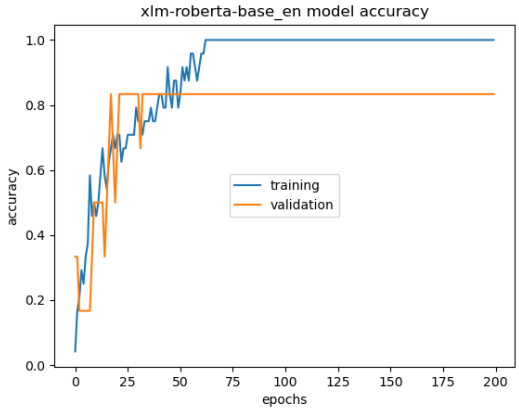


(b) **mBERT apvienotā mašīntulkoto treniņdatu kopa**

4.12. att. **caption**

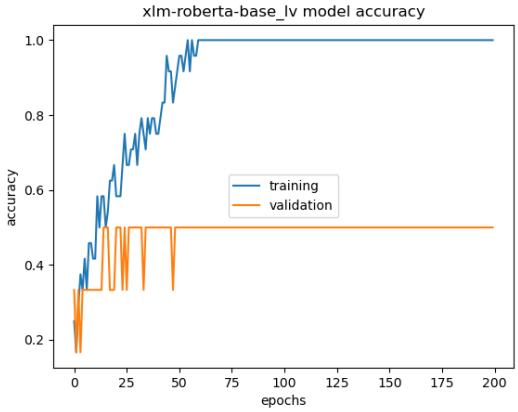


(a) **mBERT angļu treniņdatu kopa**

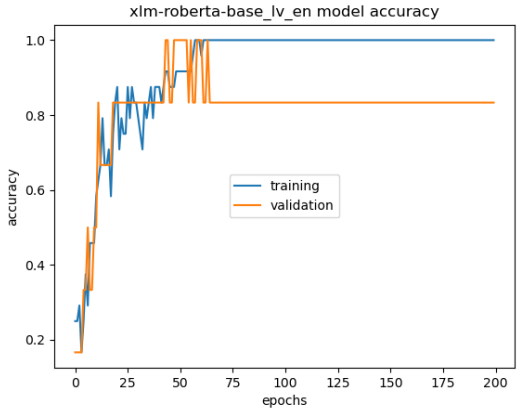


(b) **XLM-RoBERTa angļu treniņdatu kopa**

4.13. att. **caption**

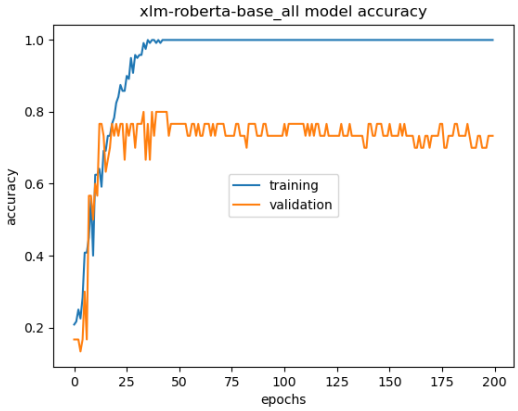


(a) **XLM-RoBERTa latviešu treniņdatu kopa**

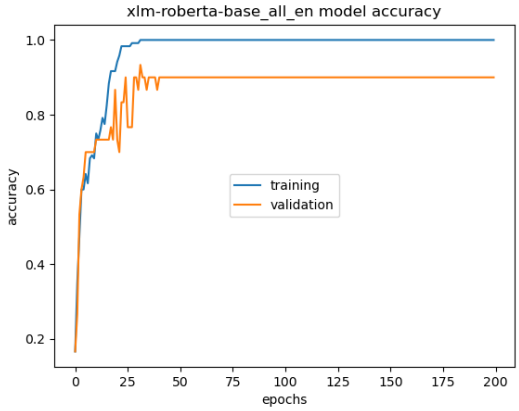


(b) **XLM-RoBERTa mašīntulkoto latviešu treniņdatu kopa**

4.14. att. **caption**



(a) **XLM-RoBERTa apvienotā treniņdatu kopa**



(b) **XLM-RoBERTa apvienotā mašīntulkoto treniņdatu kopa**

4.15. att. **caption**

SECINĀJUMI

Testējot multilingual BERT un XLM-RoBERTa daudzvalodu jēdzientelpas lietotāju nodomu noteikšanā piecās dažādās valodās tika konstatēts, ka (modelim) bija visaugstākā precizitāte (valodās) valodās ar ()%. (Modelis) arī darbojās labi ar kopējo precizitāti ()%. Rezultāti liecina, ka daudzvalodu vārdu iegulšanas izmantošana var būt efektīva nodomu noteikšanai vairākās valodās, un modeļa izvēle var būtiski ietekmēt klasifikācijas uzdevuma precizitāti.

Modeļu precizitāte katrai valodai bija atšķirīga, ar zemāko precizitāti (valodā) valodā, un augstāko precizitāti (valodā) valodā, kas bija sagaidāms ņemot vērā datu kopas uz kurām tika apmācīti multilingual BERT un XLM-RoBERTa modeļi.

Kopumā paredzams, ka nolūku klasifikācijas modeļu precizitāte būs augstāka oriģinālajiem ievades datiem latviešu, igauņu, krievu un lietuviešu valodā, salīdzinot ar to pašu ievades datu mašīnu, kas tulkota angļu valodā. Tas ir tāpēc, ka mašīntulkošana rada papildu trokšņus un kļūdas, kas var ietekmēt ievades datu kvalitāti un klasifikācijas modeļu veiktspēju. Turklāt mašīntulkošanas rezultātā dažkārt var tikt zaudētas svarīgas nianses un katrai valodai raksturīgā semantiskā informācija, kas var vēl vairāk pasliktināt ievades datu kvalitāti un apgrūtināt precīzu nolūku klasificēšanu.

Tomēr precīza modeļu veiktspēja var atšķirties atkarībā no vairākiem faktoriem, piemēram, mašīntulkošanas kvalitātes, ievades vaicājumu sarežģītības un izmantoto daudzvalodu vārdu iegulšanas specifiskajām īpašībām. Tāpēc ir svarīgi novērtēt modeļu veiktspēju gan ar oriģinālajiem, gan mašīntulkotajiem ievades datiem un salīdzināt rezultātus, lai labāk izprastu modeļu stiprās puses un ierobežojumus daudzvalodu nolūku klasifikācijas uzdevumiem.

PATEICĪBAS

Paldies darba vadītājam par mītingiem divu gadu garumā un labajiem padomiem.

Paldies Kirilam un Paulīnai par darba pārlasīšanu, un Emīlam par code review.

Paldies Aleksandrai Elbakjanai par sci-hub.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Venkat N. Gudivada un Kamyar Arbabifard. „Chapter 3 - Open-Source Libraries, Application Frameworks, and Workflow Systems for NLP". *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*. Izdevis Venkat N. Gudivada un C.R. Rao. 38. sējums. Handbook of Statistics. Elsevier, 2018, 31.—50. lpp. DOI: https://doi.org/10.1016/bs.host.2018.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0169716118300221.

[2] Pratap Dangeti. *Statistics for Machine Learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with Python and R*. Packt Publishing, 2017. ISBN: 9781788295758.

[3] Adrian Colyer. *The amazing power of word vectors*. 2016. URL: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/.

[4] Tomás Mikolov u. c. „Efficient Estimation of Word Representations in Vector Space". (2013). arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781v3.

[5] Allison Parrish. *Understanding word vectors*. 2017. URL: https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469.

[6] Dwaipayan Roy, Sumit Bhatia un Prateek Jain. „A Topic-Aligned Multilingual Corpus of Wikipedia Articles for Studying Information Asymmetry in Low Resource Languages". *Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)*. 2020. URL: https://aclanthology.org/2020.lrec-1.289.pdf.

[7] W. Nelson Francis un H Kucera. *Brown Corpus Manual: Manual of information to accompany a Standard Sample of Present-Day American English, for use with digital computers*. Brown University, 1964. URL: http://icame.uib.no/brown/bcm.html.

[8] Emily M. Bender u. c. „On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?": *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, 610.—623. lpp. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. URL: https://doi.org/10.1145/3442188.3445922.

[9] Wikimedia Foundation. *Community Insights: Community Insights 2020 Report: Thriving Movement*. https://meta.wikimedia.org/wiki/Community_Insights/Community_Insights_2020_Report/Thriving_Movement#Community_and_Newcomer_Diversity. 2020.

[10] Brian C. Britt un Sorin Adam Matei. *Structural differentiation in social media: adhocracy, entropy, and the "1 % effect"*. Lecture notes in social networks. Springer, 2017. ISBN: 978-3-319-64425-7, 3319644254, 978-3-319-64424-0. URL: http://gen.lib.rus.ec/book/index.php?md5=416b9349cbf6cff824e540feb4228cb6.

[11] Google Ngram Viewer Team. *Google Books Ngram Viewer*. https://books.google.com/ngrams/. Aplūkots 2023-05-06.

[12] Alexis Conneau u. c. „Unsupervised Cross-lingual Representation Learning at Scale". *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020. g. jūl., 8440.—8451. lpp. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747.

[13] Chris McCormick. *Word2Vec Tutorial - The Skip-Gram Model*. 2016. URL: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.

[14] Kaspars Balodis un Daiga Deksne. „FastText-Based Intent Detection for Inflected Languages". *Information* 10.5:161 (2019). ISSN: 2078-2489. DOI: 10.3390/info10050161. URL: https://www.mdpi.com/2078-2489/10/5/161.

[15] Snips. *Snips Natural Language Understanding Documentation: Key Concepts & Data Model*. https://snips-nlu.readthedocs.io/en/latest/data_model.html#intent. Aplūkots 2023-05-10.

[16] Pēteris Paikens, Artūrs Znotiņš un Guntis Bārzdiņš. „Human-in-the-Loop Conversation Agent for Customer Service". *Natural Language Processing and Information Systems*. Izdevis Elisabeth Métais u. c. Cham: Springer International Publishing, 2020, 277.—284. lpp. ISBN: 978-3-030-51310-8. DOI: https://doi.org/10.1007/978-3-030-51310-8_25. URL: https://link.springer.com/chapter/10.1007/978-3-030-51310-8_25.

[17] Charles T. Hemphill, John J. Godfrey un George R. Doddington. „The ATIS Spoken Language Systems Pilot Corpus". *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley*. 1990. URL: https://catalog.ldc.upenn.edu/docs/LDC93S4B/corpus.html.

[18] Alice Coucke u. c. *Snips Voice Platform: an embedded Spoken Language Understanding system for private-by-design voice interfaces*. https://arxiv.org/abs/1805.10190. Aplūkots 2023-05-10. 2018. arXiv: 1805.10190 [cs.CL].

[19] Daniel Braun u. c. „Evaluating Natural Language Understanding Services for Conversational Question Answering Systems". *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Saarbrücken, Germany: Association for Computational Linguistics, 2017. g. aug., 174.—185. lpp. DOI: 10.18653/v1/W17-5522. URL: https://aclanthology.org/W17-5522.

[20] Junjie Hu u. c. „XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization". *Proceedings of the 37th International Conference on Machine Learning*. 2020. URL: https://arxiv.org/abs/2003.11080.

[21] Jacob Devlin u. c. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *NAACL-HLT 2019: Minneapolis, MN, USA*. 2019, 4171.—4186. lpp. URL: https://aclanthology.org/N19-1423.pdf.

[22] Haoran Li u. c. „MTOP: A Comprehensive Multilingual Task-Oriented Semantic Parsing Benchmark". *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2021, 2950.—2962. lpp. URL: https://aclanthology.org/2021.eacl-main.257.

[23] Sebastian Schuster u. c. „Cross-Lingual Transfer Learning for Multilingual Task Oriented Dialog". *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*. 2019, 3795.—3805. lpp. URL: https://doi.org/10.18653/v1/N19-1380.

[24] Ryo Masumura u. c. „Multi-task and Multi-lingual Joint Learning of Neural Lexical Utterance Classification based on Partially-shared Modeling". *Proceedings of the 27th International Conference on Computational Linguistics*. 2018, 1137.—1155. lpp. URL: https://aclanthology.org/C18-1304.pdf.

[25] Satoshi Sekine un Chikashi Nobata. „Definition, dictionaries and tagger for extended named entity hierarchy". *Proc. Language Resources and Evaluation Conference*. 2004.

[26] Zihan Liu u. c. „Attention-Informed Mixed-Language Training for Zero-Shot Cross-Lingual Task-Oriented Dialogue Systems". *Proceedings of the AAAI Conference on Artificial Intelligence* 34.05 (2020), 8433.—8440. lpp. DOI: 10.1609/aaai.v34i05.6362. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6362.

[27] Maxime De bruyn u. c. „Machine Translation for Multilingual Intent Detection and Slots Filling". *Proceedings of the Massively Multilingual Natural Language Understanding Workshop (MMNLU-22)*. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, 2022. g. dec., 69.—82. lpp. URL: https://aclanthology.org/2022.mmnlu-1.8.

[28] Jack FitzGerald u. c. „MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages". *Arxiv.org* Jun (2022). URL: https://arxiv.org/abs/2204.08582.

[29] Mauajama Firdaus, Asif Ekbal un Erik Cambria. „Multitask learning for multilingual intent detection and slot filling in dialogue systems". *Information Fusion* 91 (2023), 299.—315. lpp. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2022.09.029. URL: https://www.sciencedirect.com/science/article/pii/S1566253522001671.

[30] Linting Xue u. c. „mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer". *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2021, 483.—498. lpp. URL: https://aclanthology.org/2021.naacl-main.41.

[31] Jörg Tiedemann un Santhosh Thottingal. „OPUS-MT – Building open translation services for the World”. *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*. Lisboa, Portugal: European Association for Machine Translation, 2020. g. nov., 479.—480. lpp. URL: https://aclanthology.org/2020.eamt-1.61.

[32] Pratik Jayarao un Aman Srivastava. „Intent Detection for code-mix utterances in task oriented dialogue systems”. *2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*. 2018, 583.—587. lpp. DOI: 10.1109/ICEECCOT43722.2018.9001577.

PIELIKUMS

Kods

Koda piemērs literatūras ievadā. Krāsu dati "xkcd.json" `https://github.com/dariusk/corpora/blob/master/data/colors/xkcd.json`.

Ideja un hex_to_int un closest funkcijas [5], pārējās pārrakstītas ātrdarbībai ar numpy.

```
import numpy as np
import json

def hex_to_int(s):
    s = s.lstrip("#")
    return int(s[:2], 16), int(s[2:4], 16), int(s[4:6], 16)



def distance(coord1, coord2):
  """Euclidean distance between two points
  """
  return np.sqrt(np.sum(np.subtract(coord1, coord2)**2))


def subtractv(coord1, coord2):
  """coord1 - coord2
  """
  return np.subtract(coord1, coord2)


def addv(coord1, coord2):
  """coord1 + coord2
  """
  return np.sum([coord1, coord2], axis=0)


def closest(space, coord, n=10):
  closest = []
  for key in sorted(space.keys(),
            key=lambda x: distance(coord, space[x]))[:n]:
```

```
    closest.append(key)
  return closest


color_data = json.loads(open("xkcd.json").read())

colors = dict()
for item in color_data['colors']:
    colors[item["color"]] = hex_to_int(item["hex"])
```

Darba rezultātu iegūšanai izmantotais kods pieejams arī: `https://github.com/chararchter/intent-detection`.

Datu korpusa mašīntulkošana angļu valodā.

```
from typing import List

from transformers import pipeline

from model import read_file


def get_source_text(dataset_type: str, source_language: str, dataset_name:
                                        str) -> List[str]:
    """ Wrapper for get_data that provides file path.

    :param dataset_type: "test" or "train"
    :param source_language: "lv", "ru", "et", "lt"
    :param dataset_name: "chatbot", "askubuntu" or "webapps"
    :return: array of file contents for specified file
    """
    return read_file(f"NLU-datasets\{dataset_name}\{source_language}\{
                                            dataset_name}_{dataset_type}_q.
                                            txt")
```

```
def translate_to_file(dataset_type: str, source_language: str, dataset:
                                    List[str], dataset_name: str,
                                    model_name: str):
    """ Write the translated text to file.
    utf-8 encoding is specified in case the source text wasn't translated
                                        and still has the source language
                                         characters.

    :param dataset_type: "test" or "train"
    :param source_language: "lv", "ru", "et" or "lt"
    :param dataset: dataset of source_language and type e.g. "lv_train"
    :param dataset_name: "chatbot", "askubuntu" or "webapps"
    :param model_name: model name e.g. "opus-mt-tc-big-et-en"
    """
    pipe = pipeline("translation", model=model_name)
    for line in dataset:
        print(line)
        output = pipe(line)
        print(output)
        if "error" in output:
            print(output)
            write_to_file(source_language, dataset_type, dataset_name, "
                                                error, original line:" +
                                                line)
        else:
            write_to_file(source_language, dataset_type, dataset_name,
                                                output[0]["
                                                translation_text"])


def write_to_file(source_language: str, dataset_type: str, dataset_name:
                                    str, output: str):
    """ Write the translated text to file.
    utf-8 encoding is specified in case the source text wasn't translated
                                        and still has the source language
```

```
                                        characters.

    :param source_language: "lv", "ru", "et" or "lt"
    :param dataset_type: "test" or "train"
    :param dataset_name: "chatbot", "askubuntu" or "webapps"
    :param output: translated text or error and sentence in original
                                        language
    """
    with open(f"{dataset_name}_{source_language}_{dataset_type}.txt", "w",
                                        encoding="utf-8") as f:
        f.write(output + "\n")


def translate_to_english(dataset: List[str], model_name: str,
                                    source_language: str, dataset_type:
                                    str):
    pipe = pipeline("translation", model=model_name)
    translated_text = pipe(dataset)
    print(translated_text)
    write_to_file(source_language, dataset_type, translated_text)


model = dict()
model = {
    "lv": ".\opus-mt-tc-big-lv-en",
    "ru": ".\opus-mt-ru-en",
    "et": ".\opus-mt-tc-big-et-en",
    "lt": ".\opus-mt-tc-big-lt-en",
}

for language, model_name in model.items():
    for dataset_name in ["chatbot", "webapps", "askubuntu"]:
        for dataset_type in ["test", "train"]:
            dataset = get_source_text(dataset_type, language, dataset_name)
            translate_to_file(
```

```
                source_language=language, dataset_type=dataset_type,
                                                    dataset_name=
                                                    dataset_name, dataset
                                                    =dataset,
                model_name=model_name
            )
```

Webapps datu kopas nepietiekami pārstāvēto anotāciju pārgrupēšana ”Other” kategorijā.

```
from collections import Counter

from model import get_source_text


def get_labels(dataset: str = "webapps") -> dict:
    """ Initialize data dictionary with train_labels and test_labels
    """
    data = dict()
    for dataset_type in ["test", "train"]:
        data.update(
            {f"{dataset_type}_labels": get_source_text(dataset_type=
                                                dataset_type, dataset=
                                                dataset, labels=True)})
    return data


def clean_data(data: list) -> list:
    # Define the list of valid categories
    valid_categories = ['Find Alternative', 'Filter Spam', 'Sync Accounts',
                                            'Delete Account']

    # Iterate through the list and check each element
    for i in range(len(data)):
        if data[i] not in valid_categories:
            data[i] = 'Other'
    return data
```

```
def count_categories(data: list):
    # Count the occurrences of each category
    category_counts = Counter(data)

    for category, count in category_counts.items():
        if count > 2:
            print(f'{category}: {count}')


labels = get_labels()
print(labels)

count_categories(labels["train_labels"])
print("######")
count_categories(labels["test_labels"])

# keepers: 'Find Alternative', 'Filter Spam', 'Sync Accounts', 'Delete
                                        Account'

labels["train_labels"] = clean_data(labels["train_labels"])
labels["test_labels"] = clean_data(labels["test_labels"])

for dataset_type in ["test", "train"]:
    with open(f"webapps_{dataset_type}_ans.txt", "w", encoding="utf-8") as
                                            f:
        f.writelines(line + "\n" for line in labels[f"{dataset_type}_labels
                                                "])
```

Modeļa arhitektūra

```
from typing import List

import matplotlib.pyplot as plt
import tensorflow as tf
```

```
from keras.layers import Dense, Conv1D, Dropout, GlobalMaxPooling1D,
                                        MaxPooling1D
from keras.models import Sequential
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, TFBertModel, AutoTokenizer,
                                        TFAutoModel


def get_embeddings_tokenizer_model(model_name: str):
    model_name = f"./{model_name}"
    if "roberta" in model_name:
        return AutoTokenizer.from_pretrained(model_name), TFAutoModel.
                                                from_pretrained(model_name)
    else:
        return BertTokenizer.from_pretrained(model_name), TFBertModel.
                                                from_pretrained(model_name)


def read_file(path: str) -> List[str]:
    """ Read path and append each line without \n as an element to an array
                                            .
    Encoding is specified to correctly read files in Russian.
    Example output: ['FindConnection', 'FindConnection', ..., '
                                            FindConnection']
    """
    with open(path, encoding='utf-8') as f:
        array = []
        for line in f:
            array.append(line.rstrip("\n"))
        return array


def get_source_text(dataset_type: str, dataset: str, source_language: str =
                                        None, labels: bool = False,
                    machine_translated: bool = False) -> List[str]:
```

```
    """ Wrapper for read_file that provides file path.
    Prompts in all languages are in the same order, therefore they use the
                                        same label files. So please be
                                        careful
    to use the correct argument for labels, as label=True returns labels
                                        regardless of specified
                                        source_language
    Usage examples:
    prompts: read_source_text("test", "et", False)
    labels: read_source_text("test")
    :param dataset_type: "test" or "train"
    :param dataset: "chatbot", "askubuntu" or "webapps"
    :param source_language: "lv", "ru", "et", "lt"
    :param labels: does the file being read contain labels
    :param machine_translated: has the data been machine translated to
                                        English?
    :return: array of file contents for specified file
    """
    if labels:
        return read_file(f"NLU-datasets\{dataset}\{dataset}_{dataset_type}
                                        _ans.txt")
    elif machine_translated:
        return read_file(f"machine-translated-datasets\{dataset}_{
                                        source_language}_{
                                        dataset_type}.txt")
    else:
        return read_file(f"NLU-datasets\{dataset}\{source_language}\{
                                        dataset}_{dataset_type}_q.txt
                                        ")


def get_dataset(datasets: dict, dataset: str = "chatbot") -> dict:
    """
    :param datasets: test/train key and languages as values
    :param dataset: "chatbot", "askubuntu" or "webapps"
```

```
    :return: dictionary with dataset type, language and optional labels and
                                            '_en' as keys and list of input
                                           data as values
    """
    results = dict()
    for key, value in datasets.items():
        results.update({f"{key}_labels": get_source_text(dataset_type=key,
                                                dataset=dataset, labels=True)
                                                })
        for lang in value:
            results.update({f"{key}_{lang}": get_source_text(dataset_type=
                                                    key, dataset=dataset,
                                                    source_language=lang)})
            if lang != "en":
                results.update({f"{key}_{lang}_en": get_source_text(
                                                        dataset_type=key,
                                                        dataset=dataset,
                                                                source_language
```

```
    return results


def split_train_data(x: list, y: list, validation_size: int = 0.2):
    """ Split training set in training and validation
    :param x: data
```

```
    :param y: labels
    :param validation_size: what fraction of data to allocate to training?
    :return:
    """
    return train_test_split(x, y, test_size=validation_size, stratify=y,
                                        random_state=42)


def split_validation(datasets: dict, data: dict) -> dict:
    """ Split training dataset in training and validation
    :param datasets: dictionary with dataset type as key and list of
                                        languages as value
    :param data: dictionary with test/train, language and labels as key and
                                        data as values
    :return: updated data dictionary where each train key is split in train
                                        and validation
    """
    for key, value in datasets.items():
        if key == "train":
            for lang in value:
                data[f"{key}_{lang}"], \
                    data[f"{key}_{lang}_validation"], \
                    data[f"{key}_{lang}_labels"], \
                    data[f"{key}_{lang}_labels_validation"] =
                                                        split_train_data(
                                                        data[f"{key}_{
                                                        lang}"],
                                                                        data
```

```
                if lang != "en":
                    data[f"{key}_{lang}_en"], \
                        data[f"{key}_{lang}_en_validation"], \
                        data[f"{key}_{lang}_en_labels"], \
                        data[f"{key}_{lang}_en_labels_validation"] =
                                                                split_train_data
                                                                (data[f"{key}
                                                                _{lang}_en"],
    return data


def plot_performance(training_data, validation_data, broad_dataset: str,
                                        dataset: str, x_label: str = '
                                        accuracy'):
    plt.plot(training_data, label='training')
    plt.plot(validation_data, label='validation')
    ax = plt.gca()
    ax.set_xlabel('epochs')
    ax.set_ylabel(x_label)
```

```
    plt.title(f"{dataset} model {x_label}")
    plt.legend(loc="center")
    plt.savefig(f"graphs/{broad_dataset}_{dataset}-{x_label}.png")
    plt.show()


def create_model(sentence_length: int, num_classes: int = 2, hidden_size:
                                    int = 768):
    model = Sequential()
    model.add(tf.keras.Input(shape=(sentence_length, hidden_size)))
    model.add(Dense(64, activation='relu'))
    model.add(Conv1D(128, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(256, kernel_size=3, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Conv1D(512, kernel_size=3, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dropout(0.1))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    return model


def create_adam_optimizer(lr=0.001, beta_1=0.9, beta_2=0.999, weight_decay=
                                    0, epsilon=0, amsgrad=False, clipnorm
                                    =1.0):
    # sgd is worse than adam
    return tf.keras.optimizers.Adam(learning_rate=lr, beta_1=beta_1, beta_2
                                        =beta_2, epsilon=epsilon, amsgrad
                                        =amsgrad,
                                    weight_decay=weight_decay, clipnorm=
                                                                    clipnorm
                                                                    )
```

```
def get_classification_model(learning_rate: float, sentence_length: int,
                                    num_classes: int, clipnorm: float = 1
                                    .0):
    optimizer = create_adam_optimizer(lr=learning_rate, clipnorm=clipnorm)
    classification_model = create_model(sentence_length=sentence_length,
                                        num_classes=num_classes)

    classification_model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return classification_model


def training(data, lang: str, learning_rate: float, sentence_length: int,
                                    batch_size: int, epochs: int,
             model_name: str, broad_dataset: str, num_classes: int = 2):
    train_data = data[f"train_{lang}"]
    # TODO: stack attributes in different levels: test/train, language and
                                        machine translated yes/no
    # t = data["train"][lang][[identifier]]
    train_labels = data[f"train_{lang}_labels"]
    validation_data = data[f"train_{lang}_validation"]
    validation_labels = data[f"train_{lang}_labels_validation"]

    print(f"train_data.shape {train_data.shape}")  # (num_samples,
                                        sentence_length, hidden_size) (80
                                        , 20, 768)
    print(f"validation_data.shape {validation_data.shape}")  # (num_samples
                                        , sentence_length, hidden_size) (
                                        80, 20, 768)
    print(f"train_labels.shape {train_labels.shape}")  # (num_samples,
                                        num_classes) (80, 2)
    print(f"validation_labels.shape {validation_labels.shape}")  # (
```

```
                                        num_samples, num_classes) (20, 2)

    classification_model = get_classification_model(learning_rate,
                                        sentence_length, num_classes)

    history = classification_model.fit(
        train_data,
        y=train_labels,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(validation_data, validation_labels)
    )

    plot_performance(
        history.history['accuracy'],
        history.history['val_accuracy'],
        broad_dataset=broad_dataset,
        dataset=f"{model_name}_{lang}",
        x_label='accuracy'
    )

    plot_performance(
        history.history['loss'],
        history.history['val_loss'],
        broad_dataset=broad_dataset,
        dataset=f"{model_name}_{lang}",
        x_label='loss'
    )

    return classification_model


def test_classification_model(model, data: dict, lang: str, batch_size: int
                                    ) -> float:
    test_data = data[f"test_{lang}"]
```

```
    test_labels = data["test_labels"]

    test_loss, test_accuracy = model.evaluate(test_data, test_labels,
                                              batch_size=batch_size)
    print('Test Loss: {:.2f}'.format(test_loss))
    print('Test Accuracy: {:.2f}'.format(test_accuracy))
    return test_accuracy
```

Eksperimentālo mērījumu iegūšana

```
from typing import Iterable, Tuple

import pandas as pd
import tensorflow as tf
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.framework.ops import EagerTensor

from model import training, \
    test_classification_model, get_source_text, split_train_data,
                                              get_embeddings_tokenizer_model


class MyModel:
    def __init__(self, batch_size: int, learning_rate: float, epochs: int,
                                              sentence_length: int, model_name:
                                               str,
                 num_classes: int, dataset: str = "chatbot", languages=("en
                                                    ", "lv", "ru", "et",
                                                     "lt")):
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.sentence_length = sentence_length
        self.model_name = model_name
        self.data = dict()
```

```
        self.datasets = dict()
        self.results = pd.DataFrame()
        self.num_classes = num_classes
        self.dataset = dataset
        self.hidden_size = 768
        # allows to run model one language at the time
        self.languages = [languages] if isinstance(languages, str) else
                                                languages
        self.non_eng_languages = list(set(self.languages) - {"en"})
        self.non_eng_languages = [language + "_en" for language in self.
                                                non_eng_languages]
        # explicit better than implicit, making sure the order of languages
                                                 is consistent across board
        self.non_eng_languages = ['lv_en', 'ru_en', 'lt_en', 'et_en', '
                                                lt_en']
        self.csv_file_name = f"{self.dataset}_{self.model_name}_results.csv
                                                "

        self.tokenizer, self.model = get_embeddings_tokenizer_model(self.
                                                model_name)

        self.init_dataset()
        self.init_data()
        self.init_results()

    def init_dataset(self):
        self.datasets = {
            "test": self.languages,
            "train": self.languages
        }

    def init_results(self):
        self.results['hyperparameters'] = [self.model_name, self.batch_size
                                                , self.sentence_length, self.
                                                learning_rate,
```

```
                                        self.epochs]
        self.results['languages'] = self.languages

    def init_data(self):
        self.get_dataset()

        self.labels_to_categorical()

        self.split_validation()

        self.convert_to_embeddings()

        self.merge_all_data('train_all', ['train_en', 'train_lv', 'train_ru
                                          ', 'train_et', 'train_lt'])
        self.merge_all_data('train_all_labels',
                            ['train_en_labels', 'train_lv_labels', '
                                                          train_ru_labels
                                                          ', '
                                                          train_et_labels
                                                          ',
                             'train_lt_labels'])
        self.merge_all_data('train_all_validation',
                            ['train_en_validation', 'train_lv_validation',
                                                          '
                                                          train_ru_validation
                                                          ', '
                                                          train_et_validation
                                                          ',
                             'train_lt_validation'])
        self.merge_all_data('train_all_labels_validation', ['
                                          train_en_labels_validation',
                                          'train_lv_labels_validation',
                                                        '
```

```
                                                '

self.merge_all_data('train_all_en', ['train_en', 'train_lv_en', '
                                     train_ru_en', 'train_et_en',
                                     'train_lt_en'])
self.merge_all_data('train_all_en_labels',
                    ['train_en_labels', 'train_lv_en_labels', '
                                                          train_ru_en_labels
                                                          ', '
                                                          train_et_en_labels
                                                          ',
                     'train_lt_en_labels'])
self.merge_all_data('train_all_en_validation',
                    ['train_en_validation', 'train_lv_en_validation
                                                          ', '
                                                          train_ru_en_validati
                                                          ',
                     'train_et_en_validation',
                     'train_lt_en_validation'])
self.merge_all_data('train_all_en_labels_validation',
                    ['train_en_labels_validation', '
                                                          train_lv_en_labels_v
                                                          ',
```

```
                        'train_ru_en_labels_validation',
                        'train_et_en_labels_validation', '
                                                        train_lt_en_labels_
                                                        '])

    print(self.data)

def is_translated(self, translated: bool = False) -> Tuple[list, list,
                                        str, str]:
    """ Return appropriate parameters based on whether the data has
                                            been machine translated to
                                            English
    :param translated: has the data been machine translated to English?
    :return: which list of languages to use, initialized result set and
                                             column name for results
                                            dataframe
    """
    if translated:
        return self.non_eng_languages, [None], "translated", "_en"
    else:
        return self.languages, [], "untranslated", ""

def train_and_test_on_same_language(self, translated: bool = False):
    """ Each language has its own model, e.g., training on Latvian,
                                            testing on Latvian
    :param translated: has the data been machine translated to English?
    """
    temp_languages, temp_results, col_name, discard = self.
                                            is_translated(translated)
    for language in temp_languages:
        classification = training(self.data, language, self.
                                                learning_rate, self.
                                                sentence_length, self.
                                                batch_size,
                                    self.epochs, self.model_name, self.
```

```
                                                                dataset
                                                                ,
                                                                self
                                                                .
                                                                num_classe
                                                                )

            temp_results.append(test_classification_model(classification,
                                                          self.data, language, self
                                                          .batch_size))
        self.results[f"1_{col_name}"] = temp_results
        self.results.to_csv(self.csv_file_name, index=False)

    def train_on_all_languages_test_on_one(self, translated: bool = False):
        """ One model trained on all language datasets, tested on each
                                                  language separately
        e.g., training on all, testing on Latvian
        :param translated: has the data been machine translated to English?
        """
        temp_languages, temp_results, col_name, identifier = self.
                                                  is_translated(translated)
        classification = training(self.data, f"all{identifier}", self.
                                                  learning_rate, self.
                                                  sentence_length,
                                  self.batch_size, self.epochs, self.
                                                                model_name
                                                                ,
                                                                self
                                                                .
                                                                dataset
                                                                ,
                                                                self
                                                                .
                                                                num_classes
```

```
                                                                    )

        for language in temp_languages:
            temp_results.append(test_classification_model(classification,
                                                          self.data, language, self
                                                          .batch_size))
        self.results[f"2_{col_name}"] = temp_results
        self.results.to_csv(self.csv_file_name, index=False)

    def train_on_english_test_on_non_english(self, translated: bool = False
                                             ):
        """ Trained on English only, tested on non-English
        e.g., training on English, testing on Latvian
        :param translated: has the data been machine translated to English?
        """
        temp_languages, temp_results, col_name, discard = self.
                                                  is_translated(translated)
        classification = training(self.data, "en", self.learning_rate, self
                                                  .sentence_length,
                                  self.batch_size, self.epochs, self.
                                                                    model_name
                                                                    ,
                                                                    self
                                                                    .
                                                                    dataset
                                                                    ,
                                                                    self
                                                                    .
                                                                    num_classes
                                                                    )

        for language in temp_languages:
            temp_results.append(test_classification_model(classification,
                                                          self.data, language, self
                                                          .batch_size))
```

```
        self.results[f"3_{col_name}"] = temp_results
        self.results.to_csv(self.csv_file_name, index=False)

    def get_dataset(self):
        """ Initialize data dictionary with values read from files
        """
        for key, value in self.datasets.items():
            self.data.update({f"{key}_labels": get_source_text(dataset_type
                                                =key, dataset=self.
                                                dataset, labels=True)})
            for lang in value:
                self.data.update(
                    {f"{key}_{lang}": get_source_text(dataset_type=key,
                                                      dataset=self.
                                                      dataset,
                                                      source_language=
                                                      lang)}
                )
                if lang != "en":
                    self.data.update({f"{key}_{lang}_en": get_source_text(
                        dataset_type=key,
                        dataset=self.dataset,
                        source_language=lang,
                        machine_translated=True
                    )})

    def labels_to_categorical(self):
        """ Convert string labels to categorical data
        """
        label_encoder = LabelEncoder()
        for key in self.data.keys():
            if "labels" in key:
                # Encode string labels to integer labels
                self.data[key] = label_encoder.fit_transform(self.data[key]
                                                             )
```

```
            # Convert integer labels to categorical data
            print(f"Num classes: {len(label_encoder.classes_)}")
            self.data[key] = to_categorical(self.data[key], num_classes
                                            =len(label_encoder.
                                            classes_))

def split_validation(self):
    """ Split training dataset in training and validation
    """
    for key, value in self.datasets.items():
        if key == "train":
            for lang in value:
                self.data[f"{key}_{lang}"], \
                    self.data[f"{key}_{lang}_validation"], \
                    self.data[f"{key}_{lang}_labels"], \
                    self.data[f"{key}_{lang}_labels_validation"] =
                                                        split_train_data
                                                        (self.data[f"
                                                        {key}_{lang}"
                                                        ],

                if lang != "en":
```

```
                    self.data[f"{key}_{lang}_en"], \
                        self.data[f"{key}_{lang}_en_validation"], \
                        self.data[f"{key}_{lang}_en_labels"], \
                        self.data[f"{key}_{lang}_en_labels_validation"]
                                                                =
                                                               split_train_data
                                                               (
                        self.data[f"{key}_{lang}_en"],
                        self.data[f"{key}_labels"])

    def convert_to_embeddings(self):
        """ Loop through the data dictionary and call get_word_embeddings
                                                on each key that isn't a
                                                label
        """
        for key, value in self.data.items():
            if "labels" not in key:
                self.data[key] = self.get_word_embeddings(value)

    def get_word_embeddings(self, vectorizable_strings: list) ->
                                            EagerTensor:
        """ Convert input to word embeddings
        """

        encoded_input = self.tokenizer(
            vectorizable_strings,
            padding='max_length',
            max_length=self.sentence_length,
            truncation=True,
            return_tensors='tf'
        )
        return self.model(encoded_input)["last_hidden_state"]

    def merge_all_data(self, new_key_name: str, keys_to_merge: Iterable):
        new_values = tf.concat([self.data[key] for key in keys_to_merge],
```

```
                                                axis=0)
            self.data.update({new_key_name: new_values})


# learning_rate=0.0001 is too small
if __name__ == "__main__":
    model = MyModel(batch_size=24, learning_rate=0.001, epochs=200,
                                                sentence_length=20, model_name="
                                                xlm-roberta-base")
    model.train_and_test_on_same_language(translated=True)
    model.train_and_test_on_same_language(translated=False)
    model.train_on_all_languages_test_on_one(translated=True)
    model.train_on_all_languages_test_on_one(translated=False)
    model.train_on_english_test_on_non_english(translated=True)
    model.train_on_english_test_on_non_english(translated=False)

    model = MyModel(batch_size=24, learning_rate=0.001, epochs=200,
                                                sentence_length=20,
                    model_name="bert-base-multilingual-cased", num_classes=
                                                            2)
    model.train_and_test_on_same_language(translated=True)
    model.train_and_test_on_same_language(translated=False)
    model.train_on_all_languages_test_on_one(translated=True)
    model.train_on_all_languages_test_on_one(translated=False)
    model.train_on_english_test_on_non_english(translated=True)
    model.train_on_english_test_on_non_english(translated=False)


    model = MyModel(batch_size=16, learning_rate=0.0001, epochs=200,
                                                sentence_length=20,
                    model_name="bert-base-multilingual-cased", num_classes=
                                                            5, dataset="
                                                            askubuntu")
    model.train_and_test_on_same_language(translated=True)
    model.train_and_test_on_same_language(translated=False)
```

```
model.train_on_all_languages_test_on_one(translated=True)
model.train_on_all_languages_test_on_one(translated=False)
model.train_on_english_test_on_non_english(translated=True)
model.train_on_english_test_on_non_english(translated=False)

model = MyModel(batch_size=16, learning_rate=0.0001, epochs=200,
                                        sentence_length=20,
                model_name="xlm-roberta-base", num_classes=5, dataset="
                                                    askubuntu")
model.train_and_test_on_same_language(translated=True)
model.train_and_test_on_same_language(translated=False)
model.train_on_all_languages_test_on_one(translated=True)
model.train_on_all_languages_test_on_one(translated=False)
model.train_on_english_test_on_non_english(translated=True)
model.train_on_english_test_on_non_english(translated=False)


model = MyModel(batch_size=8, learning_rate=0.0001, epochs=200,
                                        sentence_length=20,
                model_name="bert-base-multilingual-cased", num_classes=
                                                    5, dataset="
                                                    webapps")
model.train_and_test_on_same_language(translated=True)
model.train_and_test_on_same_language(translated=False)
model.train_on_all_languages_test_on_one(translated=True)
model.train_on_all_languages_test_on_one(translated=False)
model.train_on_english_test_on_non_english(translated=True)
model.train_on_english_test_on_non_english(translated=False)

model = MyModel(batch_size=8, learning_rate=0.0001, epochs=200,
                                        sentence_length=20,
                model_name="xlm-roberta-base", num_classes=5, dataset="
                                                    webapps")
model.train_and_test_on_same_language(translated=True)
model.train_and_test_on_same_language(translated=False)
```

```
model.train_on_all_languages_test_on_one(translated=True)
model.train_on_all_languages_test_on_one(translated=False)
model.train_on_english_test_on_non_english(translated=True)
model.train_on_english_test_on_non_english(translated=False)
```

Maģistra darbs „**Daudzvalodīgu jēdzientelpu pielietojums nodomu noteikšanā**” izstrādāts Latvijas Universitātes **Datorikas fakultātē**.

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

Autors: _____
(Autora paraksts un datums)

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Darba vadītājs: _____
(Vadītāja paraksts un datums)

Recenzents: _____
(Dr.sc.comp Normunds Grūzītis)

Darbs iesniegts maģistratūras sekretariātā: _____
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķis: _____
(Metodiķa paraksts)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)