

LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

**DAUDZVALODĪGU JĒDZIENTELPU PIELIETOJUMS
NODOMU NOTEIKŠANĀ**

MAĢISTRA DARBS

Autors: **Viktorija Leimane**

Studenta apliecības Nr.: v116047

Darba vadītājs: Dr.sc.comp. Kaspars Balodis

RĪGA 2022

ANOTĀCIJA

Daudzvalodīga lietotāja nodomu noteikšana ir nozīmīga virtuālo asistentu darbībā, un klientu apkalpošanas automatizācija kļūst arvien izdevīgāka un aktuālāka. Viens veids noteikt nodomu ir attēlojot ievades teksta virknes daudzdimensionālā vektoru telpā jeb jēdzientelpā, kuru izmanto nodomu klasifikācijas modeļi, lai piegādātu lietotājiem tiem nepieciešamo informāciju. Darbā tiks apmācīti dažādi mašīnmācīšanās modeļi un salīdzinātas dažādas pieejas: ievades teksta attēlojums uz daudzvalodīgu tekstu korpusu apmācītas jēdzientelpas un ievades teksta mašīntulkošana uz angļu valodu un attēlojums uz angļu valodas korpusa apmācītas jēdzientelpas.

Atslēgas vārdi: daudzvalodīgas jēdzientelpas, nodomu noteikšana

ABSTRACT

Multilingual intention detection is important for virtual assistants, and customer-service automation is becoming more cost-effective and relevant. One way of detecting intent is mapping input text strings to a multidimensional vector space, which is used by intent classification models to supply users with the information they need. This thesis focuses on training a variety of machine learning models and comparing different approaches, such as mapping input text to multilingual word embeddings and machine translating input text to English and using English corpus-based word embeddings for intent detection.

Keywords: multilingual word embeddings, intent detection

SATURS

APZĪMĒJUMU SARAĶSTS	4
IEVADS	5
1 DABISKO VALODU APSTRĀDE	6
2 JĒDZIENTELPA	7
3 MODEĻI	12
3.1 Continuous Bag-of-Words	12
3.2 Continuous Skip-gram Model	13
4 NODOMU NOTEIKŠANA	14
IZMANTOTĀ LITERATŪRA UN AVOTI	16
PIELIKUMS	17

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 daudzdimensionāli 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.

IEVADS

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

Uzņēmumiem tas galvenokārt ir izdevīgi, jo ļauj samazināt personālizdevumus. Tas savukārt samazina barjeru iekļūšanai un dalībai starptautiskā tirgū, kas nozīmē lielāku konkurenci un piedāvāto pakalpojumu daudzveidību. Lietotājiem, kuru dzimto valodu pārvalda mazs cilvēku skaits kā tas ir, piemēram, latviešu valodā, ir pieejami pakalpojumi, kuru tulkojumus būtu ekonomiski nerentabli nodrošināt ar algotu profesionālu personālu.

Darbā apskatītā metode nodrošina automatizāciju divos veidos:

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

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.

1. DABISKO VALODU APSTRĀDE

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 kolekcija) pieejamībai, NLP uzdevumi arvien biežāk un efektīvāk tiek risināti ar mašīnmācīšanās modeļiem [1]. Dabiskās valodas apstrādei ir liels biznesa potenciāls, jo tas ļauj uzņēmumiem palielināt peļņu samazinot izdevumus, no kuriem lielākais parasti ir darbs.

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 [2].

2. JĒDZIENTELPA

Jēdzientelpa ir vārdu vai frāžu attēlojums daudzdimensionālā vektoru telpā. Jēdzientelpas no teksta korpusa iegūst ar neironu tīkliem, kuri uztver kontekstu no tuvākajiem vārdiem tekstā.

Vienizcēluma kodējums (*one hot encoding*) 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. Piemēram, ja vārdu krājumā ir četri vārdi: karalis, karaliene, sieviete, vīrietis, karaliene tiktu kodēta kā $[0, 1, 0, 0]$ [3].

Vektora garuma sasaiste ar vārdu krājuma izmēru ir trūkums, jo vārdu vektori ir cieši savienoti (coupled) ar korpusu un statistiski, 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 raksturīgs sparsity (neblīvums/izretinātība): 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].

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].

Vārdus ir daudz grūtāk salīdzināt nekā skaitļus, tādēļ mēs piešķiram vārdiem vektorus. Tomēr vārdi apraksta objektus ar noteiktām kvantificējamām īpašībām, piemēram, vieglāks/smagāks (svars), lētāks/dārgāks (cena). Šādai reprezentācijai ir jēga, jo dažādus objektus var salīdzināt savā starpā pēc īpašību vērtības jeb izteiktības pakāpes, piemēram, velosipēds ir vieglāks nekā mašīna. Tādā veidā vārda attēlojums tiek sadalīts pa visiem vektora elementiem, un katrs elements pievieno nozīmi daudziem vārdiem (2.1 attēls). Zinot, ka objektu īpašību skaitliska reprezentācija palīdz tos salīdzināt, atklājas jēga kvantitatīvi izteikt semantiku, tādējādi vārdi tiek attēloti veidā, kas izsaka to nozīmi caur kontekstu.

Cilvēkiem uztverama jēdzientelpu analogija ir krāsas nosaukums un tam atbilstošais vektors RGB krāsu modelī ar R, G un B koordinātēm no 0 līdz 255, piemēram, red = (255, 0, 0). Ar krāsu



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

jēdzientelpām ir iespējams veikt saskaitīšanu un atņemšanu, kam ir fizikāla nozīme [4].

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.

$$\text{purple} - \text{red} = \text{blue}$$

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

```
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 nolasīt 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)
```

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 nozīmē. Jēdzientelpas ietver gan sintaktiskas (2.2 tabula), gan semantiskas (2.1 tabula) attiecības starp vārdiem. Jāuzsver, ka tādas semantiskas attiecības kā valsts–galvaspilsēta (2.2) nav eksplicīti uzdotas, jēdzientelpu modelis tās ir novērojis tikai balstoties uz vārdu atrašanās vietām teksta korpusā. Iespēja trenēt modeli uz neanotētiem datiem kā šajā gadījumā samazina modeļa izmaksas valodām, kurās anotēti dati ir mazāk pieejami, un daudzkārt palielina potenciālās treniņu kopas apjomu, kas parasti ļauj sasniegt lielāku precizitāti.

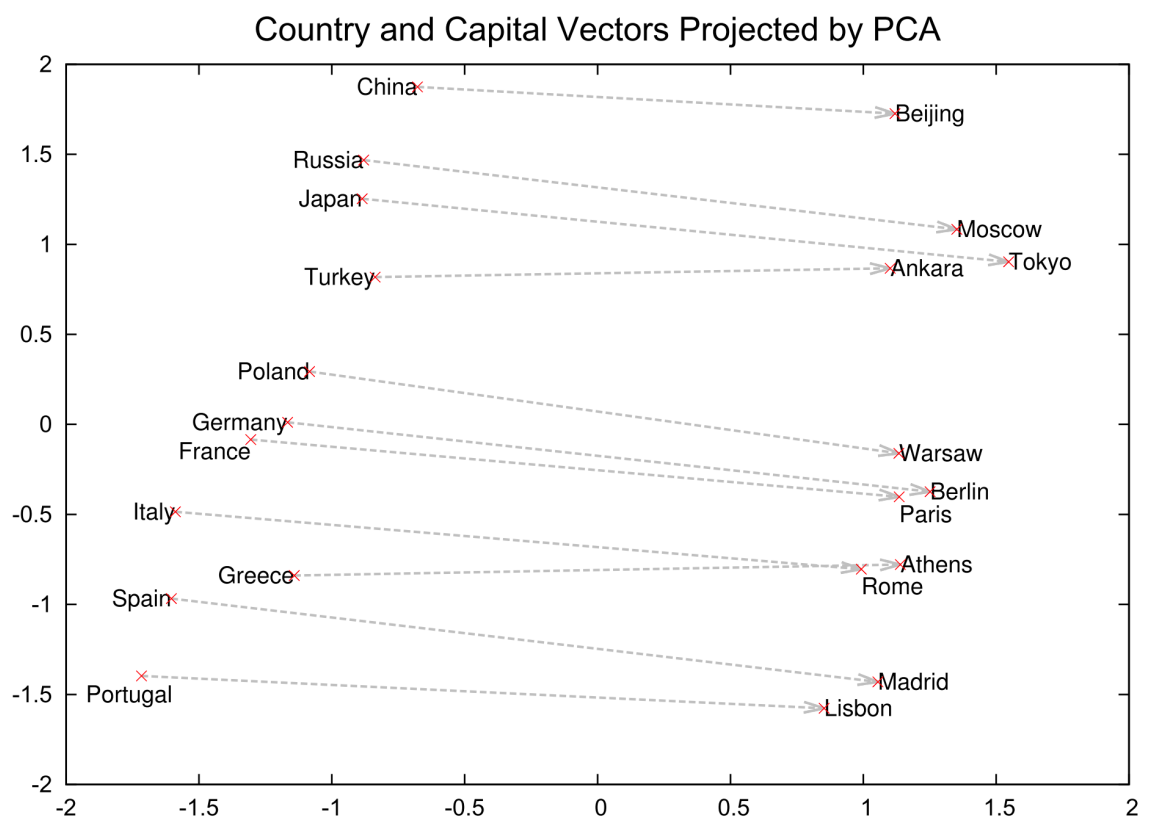
Spēja noteikt sintaktiskas un semantiskas vārdu attiecības ir īpaši būtiska virtuālo asistentu jomā, jo, pirmkārt, semantiski līdzīgiem nodomiem ir līdzīgi vektori, tātad tie vienādi klasificēsies, otrkārt, informācija par sintakses attiecībām noder, jo lietotāji ievada jautājumus brīvā formā un tas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

2.1. tabula

Semantisko attiecību piemēri [5]	
attiecība	piemērs
valsts–galvaspilsēta	Parīze - Francija + Itālija = Roma
valsts–valūta	dolāri - ASV + Latvija = eiro
vīrietis–sieviete	karalis - vīrietis + sieviete = karaliene

2.2. tabula

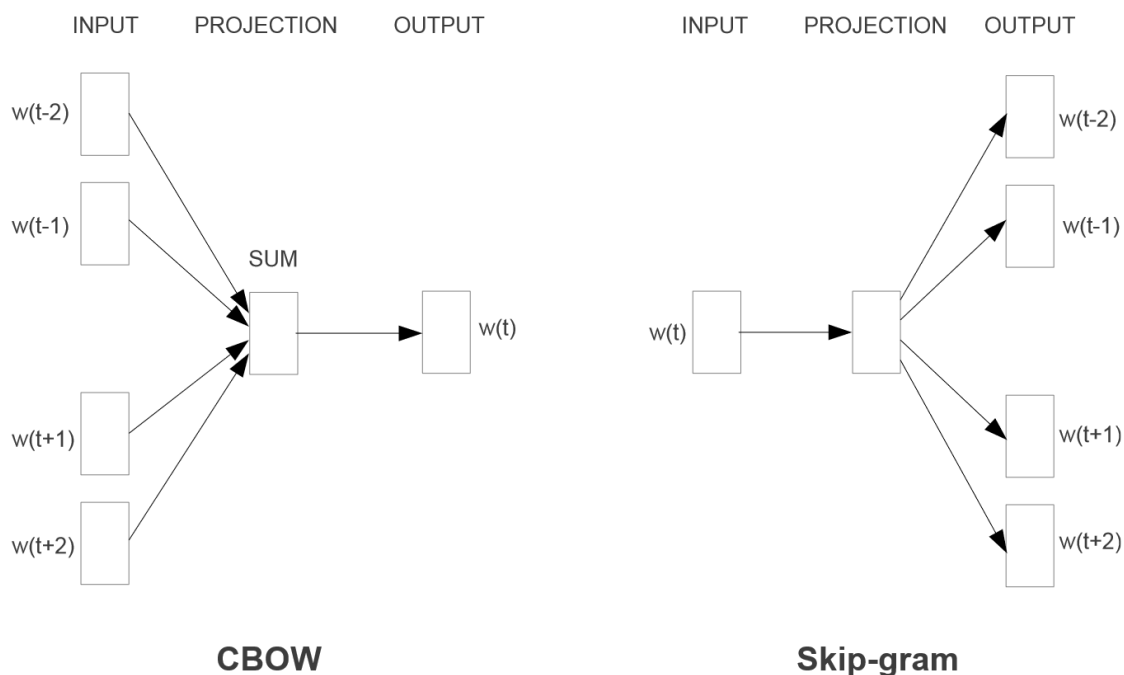
Sintaktisko attiecību piemēri [5]	
attiecība	piemērs
daudzskaitlis	pele - peles
pagātnē	staigā - staigāja
salīdzināmā pakāpe	labs - labāks



2.2. att. Divdimensionāla PCA projekcija uzrāda attiecības starp valstu un galvaspilsētu jēdzien-
telpām [3]

3. MODEĻI

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 (3.1 attēls).



3.1. att. CBOW un Skip-gram modeļu arhitektūra [5].

3.1. Continuous Bag-of-Words

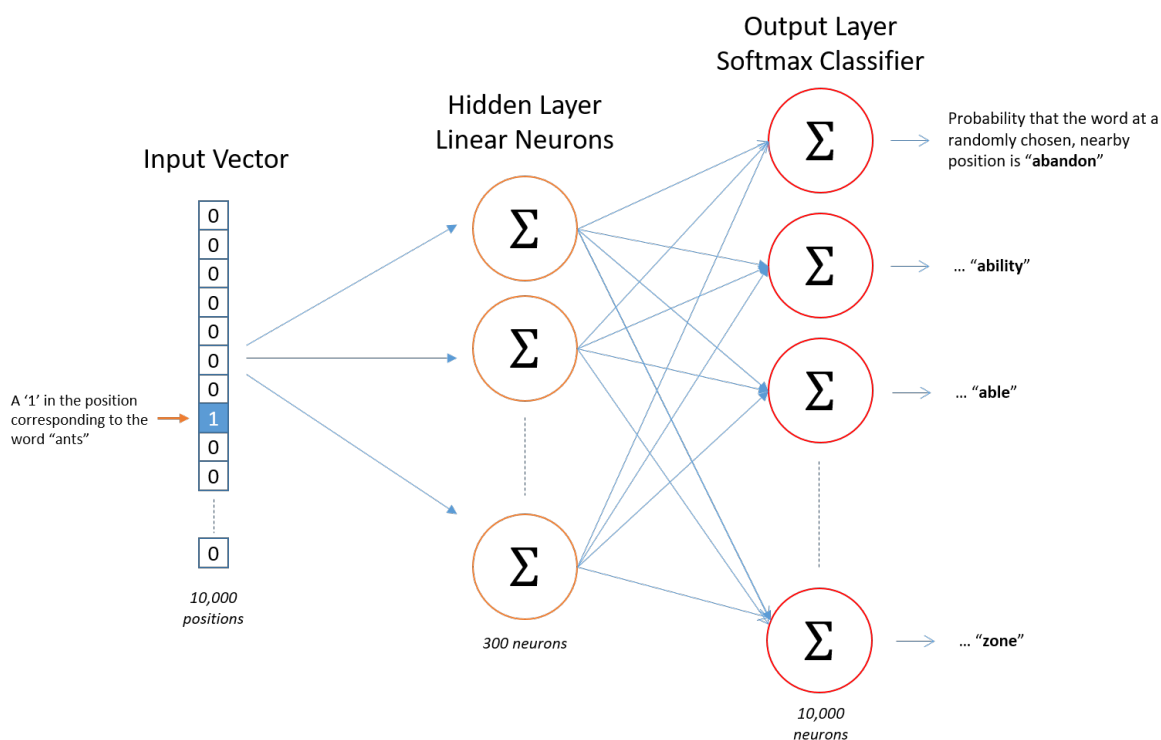
Bag-of-Words (BOW) apzīmē vārdu grupu nesaglabājot kārību. Vienā izlasē (bag) vārda tuvums mērķa vārdam konkrētā izlasē nav tik svarīgs, atkārtojot procesu uz korpusa no konteksta tāpat tiks sīkāk (granulētāk) izšķirti svāri tuvākajiem vārdiem, piemēram, Rīga un Latvija būs tuvumā 1000 reizes biežāk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

atrodas līdzīgos kontekstos, ir ar līdzīgu nozīmi [5]. Tāpat kā BOW modeli, CBOW vārdu secība neietekmē projekciju. Nepārtrauktība (*continuity*) modelī rodas no tā, ka izmanto nepārtrauktu izklaidētu konteksta reprezentāciju jeb svāri starp ievades un projekcijas slāņiem tiek lietoti visiem vārdiem. [5].

3.2. Continuous Skip-gram Model

Koncepts ir uztrenēt neironu tīklu ar slēpto slāni (*hidden layer*) un iegūt slēptā slāņa svarus, kas patiesībā arī ir vārdu vektori. Uzdevums ir no ievades vārdu pa vārdam paredzēt apkārt esošos vārdus. Kaimiņu vārdu skaits – loga izmērs (*window size*) – ir hiperparametrs (3.2 attēls).



3.2. att. **Skip-gram modeļa arhitektūra.** Ievades vektors – vārda vienizcēluma kodējums $1 \times n$ – kur n ir vārdu skaits; slēptais slānis – $n \times l$, kur l ir loga izmērs; Softmax slānis – $1 \times l$; Izvades slānis – $1 \times n$ [6].

4. NODOMU NOTEIKŠANA

Nodoms ir mērķis, kas lietotājam ir padomā, rakstot jautājumu. Piemēram, lietotājs raksta: “kad ir nākošais vilciens uz Rīgu?”, un mērķis ir pārlūkot vilcienu sarakstu. 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 [7]. Tāpēc klienta nodoma noteikšana ir svarīga, lai nodrošinātu patīkamu lietotāja pieredzi.

Nodomu noteikšana ir klasifikācijas uzdevums: attēlot lietotāja brīvā valodā rakstītu pieprasījumu uz visiespējamāko nolūku. Nodomu noteikšanai lieto neironu tīklu klasifikatoru, kas mācās no 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*) [8].

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 izvēlēta 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 [7].

Tipiski soļi nodomu noteikšanas biznesa pielietojumā:

1. Atrast visbiežākos pieprasījumu tipus;
2. Sagatavot atbildes veidni (*template*);
3. Nodomu noteikšanas sistēma identificē, vai lietotāja pieprasījums pieder iepriekšdefinētajiem tiptiem 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 [7].

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] 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>.
- [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] Allison Parrish. *Understanding word vectors*. 2017. URL: <https://gist.github.com/aparrish/2f562e3737544cf29aaf1af30362f469>.
- [5] 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>.
- [6] Chris McCormick. *Word2Vec Tutorial - The Skip-Gram Model*. 2016. URL: <http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>.
- [7] Pēteris Paikens, Artūrs Znotiņš un Guntis Bārzdīņš. „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.
- [8] Kaspars Balodis un Daiga Deksnē. „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>.

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 [4], 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"])
```