

Szegedi Tudományegyetem

Informatikai Intézet

Számítógépes Algoritmusok és Mesterséges Intelligencia
Tanszék

Nyelvtechnológiai csoport

**Nyelvi modellek összehasonlítása kontextus alapú
szójelentés-összehasonlítás (WiC) problémára**

Szakedolgozat

Készítette:
Fábián Bernát
Programtervező informatikus szakos
hallgató

Témavezető:
Dr. Berend Gábor Iván
egyetemi docens

Informatikai Intézet
Számítógépes Algoritmusok és Mesterséges Intelligencia
Tanszék
Nyelvtechnológiai csoport

Szeged
2025

Tartalomjegyzék

Feladatkiírás	6
Tartalmi összefoglaló	7
1. Motiváció	8
2. A WiC probléma	9
3. Irodalmi áttekintés	9
4. A WiC feladat és jelentősége	9
4.1. A szakdolgozat célja:	10
5. Az alkalmazás célja	10
6. Az alkalmazás célja	11
6.1. Tanító, fejlesztő- és tesztelő adatbázis:	12
7. Eddigi legjobb megoldások	13
8. <i>Mik a nagy nyelvi modellek? A nagy nyelvi modell (angolul Large Language Model, LLM) olyan számítási modell, amely képes nyelv generálására vagy más természetes nyelvi feldolgozási feladatok elvégzésére. Ezek a modellek hatalmas mennyiségű szöveges adat feldolgozásával és mélytanulási technikák alkalmazásával sajátítják el a nyelv megértését és előállítását. Az LLM-ek, mint például az OpenAI GPT-sorozata, a Google PaLM vagy a Meta LLaMA modelljei, különböző architektúrákat használnak, de leggyakrabban a transzformer alapú megközelítést alkalmazzák. Ezek a modellek képesek különféle feladatok elvégzésére, mint például szövegfordítás, összefoglalás, kérdés-válasz és kreatív szövegalkotás. Azonban fontos megjegyezni, hogy az LLM-ek teljesítménye és alkalmazhatósága nagymértékben függ a betanításukhoz használt adatok minőségétől és mennyiségétől. Miért használtam őket a feladat megoldásához? Saját, ipari modellekhez hasonló minőségű készítése körülményes, amelyhez sem időm, sem kapacitásom, sem hardveres erőforrásaim nem voltak. A nagy cégek nagy nyelvi modelljei előnye, hogy hatalmas adathalmazban vannak tanítva, szinte mindenre láttak már példát - talán ezt a Pilehvar féle WiC adathalmazt is látták már, hiszen része a SuperGlue benchmarknak, mellyel számos modellt tesztelnek (de lehet, hogy</i>	

tanítják is a train adathalmazzal).	15
9. 1.1. Chatbot Arena, vagy LMSYS - a kiértékelő platform	15
10. Közösség rangsor	16
10.1. 1.1.1. A beadott szöveg eredete	16
10.2. 1.1.2. A beadott szöveg formátuma	17
10.2.1. 1.1.2.1 Egy mondat formátuma	17
11. Kérdés-válasz szótár	17
11.0.1. 1.1.2.1 A beadott mondatok száma	17
11.1. A nagy nyelvi modellek konfigurása a prompt beküldése előtt	18
12. Klasszifikációs módszer	18
13. "True positive: Sick people correctly identified as sick"	
"False positive: Healthy people incorrectly identified as sick"	
"True negative: Healthy people correctly identified as healthy"	
"False negative: Sick people incorrectly identified as healthy"	18
13.1. Megvizsgált modellek	18
14. gemma-2-2b-it modell letöltése és installálása lokális használatra	20
15. Tesztelés fázis	22
15.1. 1.2. Benchmarking és tesztelés	23
15.2. Konzisztencia	23
16. 2. Saját rendszer fejlesztése	23
16.1. Tervezés	23
16.1.1. Projekt tervezés fázis:	23
16.2. Implementáció	24
16.3. Tesztelés	24
16.4. 2.2.1. A WiC modell működése	26

16.5. 2.2.2. Adatfeldolgozás és annotáció	26
16.6. 2.2.3. Modell teljesítményének értékelése	26
17. 2.2.4. A Flask keretrendszer alkalmazása	26
17.1. 2.2.5. Az API struktúrája	26
17.2. 2.2.6. Backend implementáció	26
17.3. 2.2.7. Frontend lehetőségek	27
18. 2.2.8. Összegzés és jövőbeli fejlesztések	27
19. 2.2.9. Összegzés	27
20. Összefoglalás	28
Nyilatkozat	29
Köszönetnyilvánítás	30
21. 3. Hivatkozások és források	31
Irodalomjegyzék	32

Feladatkiírás

A hallgató feladata a nyelvtechnológia és azon belül a kontextus alapú szójelentés-felismerés problémakörének a megismerése, jelenlegi legjobb megoldások áttekintése, valamint egy angol nyelvű kontextus alapú szójelentés-felismerő rendszer implementálása és kiértékelése. - Ismerje meg az NLP és a nagy nyelvi modellek fogalomrendszerét - Tervezze meg, implementálja és a kontextus alapú szójelentés-felismerő rendszerét, elsősorban a <https://pilehvar.github.io/wic/> nyilvános és ingyenes Word-in-Context adathalmazra tervezve, de minél általánosabb a rendszer, annál jobb. Az adatok letöltése után végezze el a szükséges adattranzformációkat, az adattisztítási feladatokat, szűrje ki a felesleges, vagy használhatatlan adatok körét, illetve készítse el az elemzésre a kész adatállományt. - Értékelje ki a kontextus alapú szójelentés-felismerő rendszerét. - Az elért eredményeiről készítsen statisztikai összesítéseket és grafikus lekérdezéseket.

Elvégzett munkáit a Szakdolgozat keretében dokumentálja.

Tartalmi összefoglaló

- **A téma megnevezése** Egy Word-in-Context (WiC) feladat megoldása. Mohammad Taher Pilehvar WiC adathalmaznak, amely egy validációs, tanító és kiértékelő részekből álló, jó minőségű benchmark adathalmaz, a legmodernebb nyelvi modelleken való kiértékelése és az eredmények elemzése. A feladathoz tartozik egy saját kontextus alapú szójelentés-felismerésre fejlesztett kiértékelő rendszer megtervezése, implementálása Python nyelven, majd tesztelése. Ez egy klasszifikációs probléma, tehát a Hallgató célja, hogy minél nagyobb pontosságot érjen el a célfeladatra.
- **A megadott feladat megfogalmazása** A feladat a kontextus alapú szójelentés-felismerés problémakörének a megismerése, jelenlegi legjobb megoldások áttekintése, valamint egy olyan angol nyelvű kontextus alapú szójelentés-felismerő rendszer implementálása és kiértékelése, amely képes legalább 60%-os pontosságot elérni a teszt adathalmazon, emellett a fedést és a precizitást megadni, tévesztési mátrixot készíteni a tp fp fn értékek eloszlásáról.
- **A megoldás lépései**
 - A jelenleg elérhető legnépszerűbb ingyenes nagy nyelvi modellek megvizsgálása: Claude Sonnet, Grok1, és a Deepseek-r3.
 - Gemma-2-2b-it lokális telepítése, futtatása és kiértékelés
 - A saját implementáció megtervezése, implementálása és tesztelése
 - Nltk wordnet Pythonos implementációját használtam a szinonímák behúzásához, BERT-et a kétirányú mondaértelmezéshez és -összehasonlításhoz. Majd harmadik lépésként a Gemma-2-2b-it, egy kvantált nyelvi modellt Huggingface-ról letölthető változatát töltöttem le és telepítettem, majd ezt hasonlítottam össze az eddigi megoldásokkal.
- **Alkalmazott technológiák, módszerek:** Gemini, GPT modellek, Claude Sonnet, Grok1, és a Deepseek-r3 webes elérhető verziói, LMArena, LMSys, Nltk, Wornet, BERT, Gemma-2-2b-it, Huggingface, Transformers, PyTorch, Matplotlib.
- **Elért eredmények** A feladatmegoldás eredménye egy, a mintafeladathoz készült feldolgozó algoritmus csomag, amely elvégzi a szükséges adattranszformációkat, az adattisztítási feladatokat adat értelmezhető formába alakítását
- **Kulcsszavak** WiC, word-in-context, NLP, nyelvi modell, nagy nyelvi modell, LLM, szövegértelmező modell, információkinyerés, szöveggörnyezet, kontextus, mondat-elemzés, szóvektorok, számítógépes szemantika

1. Motiváció

A WiC (Word-in-Context) probléma megoldása kiemelten fontos mind az akadémiai kutatásban, mind a gyakorlati alkalmazásokban, különösen Szegeden, ahol az egyetem és a helyi vállalatok is aktívan foglalkoznak természetesnyelv-feldolgozási (NLP) megoldásokkal. Az SZTE Informatikai Intézetében folytatott kutatásokban például a WiC modellek segíthetik az automatikus szövegértési rendszerek fejlesztését, ami fontos szerepet játszhat nyelvtechnológiai projektekben, gépi fordításban vagy intelligens kereső-rendszerekben. Ezen túlmenően, a helyi munkahelyeken – különösen IT-területen – egy jól működő WiC modell jelentős előnyt biztosíthat a dokumentációk, ügyfélkommunikáció vagy akár jogi szövegek értelmezésében, ami kulcsfontosságú lehet például a szegedi tudásközpontokban és startup cégeknél is. Egy hatékony WiC modell tehát nemcsak tudományos értékkel bír, hanem gyakorlati alkalmazásokban is közvetlen hasznót hozhat a programozó és IT-területen dolgozók számára.

Bevezetés

2. A WiC probléma

A program fejlesztését inkrementális modellel végeztem, tehát kezdetben csak vázlatosan határoztam meg a kiértékelő rendszer funkcióit. Ennek a modellnek az előnye például a vízesés modellel szemben az, hogy a követelmények nincsenek konkrétan le rögzítve a fejlesztés elején, hanem folyamatosan lehet rajtuk változtatni. Ha eszembe jut egy újabb módszer, amivel jobb eredményt lehet elérni, vagy egy funkció, amely növeli a program érthetőségét, átláthatóságát, akkor nem kell újratervezni a programot. Szakdolgozatom első fejezetében... A második fejezet arról szól, hogy... A harmadik fejezetben... Az utolsó fejezetben...

3. Irodalmi áttekintés

- TF-IDF és hasonlósági mérőszámok (baseline módszerek).
- Lesk algoritmus és egyéb WSD technikák: Rövid magyarázat a működésről.
- Modern neurális módszerek: Transformer alapú modellek (pl. BERT, Gemma 2-2B).
- Benchmark adathalmazok és értékelési metrikák: Pl. accuracy, precision, recall.

A szóbeágyazások tervezésükből adódóan képtelenek modellezni a szavak szemantikájának dinamikus természetét, vagyis azt a tulajdonságot, hogy a szavak potenciálisan különböző jelentéseknek felelhetnek meg. E korlátozás kezelésére tucatnyi specializált jelentésreprezentációs technikát javasoltak, mint például a jelentés- vagy kontextualizált beágyazásokat. Azonban a téma kutatásának népszerűsége ellenére igen kevés olyan értékelési viszonyítási alap létezik, amely kifejezetten a szavak dinamikus szemantikájára összpontosít. Ebben a tanulmányban bemutatom, hogy a meglévő modellek túllépték az erre a célra szolgáló standard értékelési adatkészlet, azaz a Stanford Kontextuális Szóhasonlóság teljesítménykorlátját, és rámutatunk annak hiányosságaira. A megfelelő viszonyítási alap hiányának kezelésére egy nagyszabású, szakértők által összeállított annotációkon alapuló Word in Context (Szó a Kontextusban) adatkészletet, a WiC-et javasoljuk a kontextusérzékeny reprezentációk általános értékelésére. A WiC elérhető a WiC: The Word-in-Context Dataset címen.

4. A WiC feladat és jelentősége

A WiC feladat célja, hogy egy adott szó két különböző mondatbeli előfordulásáról eldöntse, hogy azonos értelemben szerepel-e. A természetes nyelvfeldolgozásban (NLP)

ez kulcsfontosságú, mivel segít a szövegértelmezés és a jelentésalapú keresés fejlesztésében.

4.1. A szakdolgozat célja:

5. Az alkalmazás célja

A készített alkalmazás célja, hogy a kiértékelő algoritmus minél nagyobb pontosságot érjen el a WiC - The Word in Context Dataset teszhalmazon. A szakdolgozatom a **WiC Decision Maker Model** alkalmazására és optimalizálására összpontosít, amely egy Python alapú alkalmazásként valósul meg. A projekt célja egy olyan rendszer fejlesztése, amely a Word-in-Context (WiC) feladatokhoz nyújt támogatást, lehetővé téve a szavak kontextusbeli jelentésének hatékony felismerését és osztályozását.

A Flask keretrendszer lehetőséget biztosít a modell egyszerű integrációjára egy web-alapú alkalmazásba, amely REST API-ként szolgáltat adatokat és fogad bemeneteket. Az alkalmazás magában foglalja a gépi tanulási modellt, amely a WiC adathalmaz feldolgozására és azonosítására épül. A rendszer teljesítményének növelése érdekében különböző optimalizációs stratégiákat és kvantált modelleket is vizsgálok.

A projekt további célkitűzései közé tartozik a felhasználóbarát interfész kialakítása, amely lehetővé teszi a tesztesetek futtatását és az eredmények vizualizációját. Az alkalmazás fejlesztése során olyan technológiákat és eszközöket használok, mint a Hugging Face Transformers, PyTorch és a SQLite adatbázis-kezelő, hogy biztosítsam a rendszer hatékonyságát és skálázhatóságát.

Ennek megvalósítása érdekében az alábbi platformokat vizsgáljuk meg:

- Ollama
- vLLM projekt
- Hugging Face Transformers
- Llama.cpp

] A dolgozat célja egy WiC modell létrehozása, amely különböző módszereket hasonlít össze, beleértve a hagyományos (pl. TF-IDF) és modern neurális megközelítéseket. A szakdolgozatom első felében azt vizsgálom meg, hogy népszerű nagy nyelvi modellek mennyire jól oldják meg a WiC problémát.

Ezután telepítek a gépemre egy nagy nyelvi modellt, melyet lokálisan tesztelek és konfigurálok, minél jobb eredmény elérése érdekében.

Végül saját modellt hozok létre, amely képes legalább 60%-os pontosságot elérni a teszt adathalmazon, emellett a fedést és a precizitást megadni, confusion mátrixot készíteni a tp tn fp fn értékek eloszlásáról. Ennek a részleteiről a Nagy Nyelvi Modellek Eredményei fejezetben írok.

6. Az alkalmazás célja

A készített alkalmazás célja, hogy a kiértékelő algoritmus minél nagyobb pontosságot érjen el a WiC - The Word in Context Dataset teszhalmazon. A szakdolgozatom a **WiC Decision Maker Model** alkalmazására és optimalizálására összpontosít, amely egy Python alapú alkalmazásként valósul meg. A projekt célja egy olyan rendszer fejlesztése, amely a Word-in-Context (WiC) feladatokhoz nyújt támogatást, lehetővé téve a szavak kontextusbeli jelentésének hatékony felismerését és osztályozását.

A Flask keretrendszer lehetőséget biztosít a modell egyszerű integrációjára egy web-alapú alkalmazásba, amely REST API-ként szolgáltat adatokat és fogad bemeneteket. Az alkalmazás magában foglalja a gépi tanulási modellt, amely a WiC adathalmaz feldolgozására és azonosítására épül. A rendszer teljesítményének növelése érdekében különböző optimalizációs stratégiákat és kvantált modelleket is vizsgállok.

A projekt további célkitűzései közé tartozik a felhasználóbarát interfész kialakítása, amely lehetővé teszi a tesztesetek futtatását és az eredmények vizualizációját. Az alkalmazás fejlesztése során olyan technológiákat és eszközöket használok, mint a Hugging Face Transformers, PyTorch és a SQLite adatbázis-kezelő, hogy biztosítsam a rendszer hatékonyságát és skálázhatóságát.

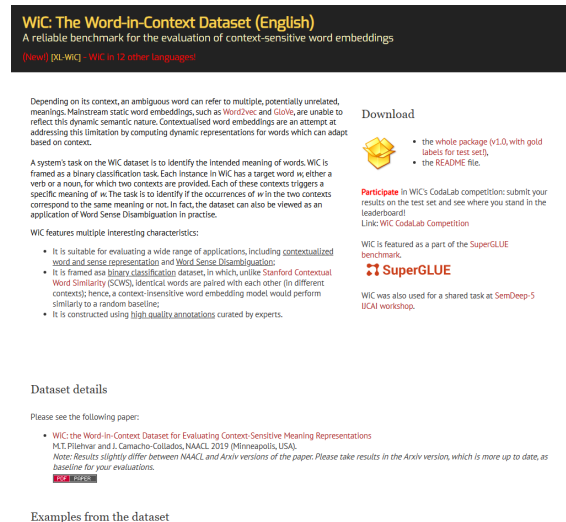
Ennek megvalósítása érdekében az alábbi platformokat vizsgáljuk meg:

- Ollama
- vLLM projekt
- Hugging Face Transformers
- Llama.cpp

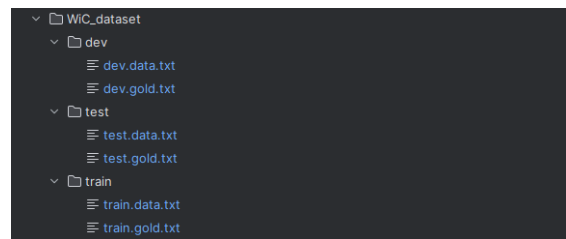
A megoldás módja:

Összehasonlítom a webes és a lokális nyelvi modell futtatás előnyeit és hátrányait.

Nyelvi modellek összehasonlítása kontextus alapú szójelentés-összehasonlítás (WiC) problémára



1. ábra. Wic: The Word-in-Context Dataset (English) főoldala



2. ábra. Az adathalmaz

A Peternity egy konzolos és webes alkalmazás, amelynek célja a WiC (Word in Context) feladatok megoldásának vizsgálata különböző nyelvi modellek segítségével. Az alkalmazás arra is képes, hogy elemezze, mennyire érzékenyek ezek a modellek arra, ha a két példamondat sorrendjét felcseréljük.] A szakdolgozatom első felében elemzem a már meglévő módszereket, megpróbálok minél jobb ingyenes nyelvi modellt találni, azt úgy paraméterezni és konfigurálni, hogy a WiC feladatot minél általánosabban, minél jobb eredménnyel meg tudja oldani.

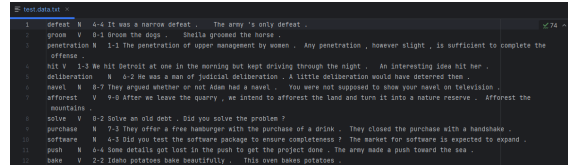
Összehasonlítom a webes és a lokális nyelvi modell futtatás előnyeit és hátrányait.

A Peternity egy konzolos és webes alkalmazás, amelynek célja a WiC (Word in Context) feladatok megoldásának vizsgálata különböző nyelvi modellek segítségével. Az alkalmazás arra is képes, hogy elemezze, mennyire érzékenyek ezek a modellek arra, ha a két példamondat sorrendjét felcseréljük.

6.1. Tanító, fejlesztő- és tesztelő adatbázis:

<https://pilehvar.github.io/wic/>

Nyelvi modellek összehasonlítása kontextus alapú szójelentés-összehasonlítás (WiC) problémára



3. ábra. Példa a tesztelő adat kinézetére

State-of-the-Art

Sentence-level contextualised embeddings	Implementation	Accuracy %
SenseBERT-target	Levine et al (2019)	72.1
KnowBERT-W+W†	Peters et al (2019)	70.9
RoBERTa	Liu et al (2019)	69.9
BERT-large	Wang et al (2019)	69.6
Ensemble	Gari Soler et al (2019)	66.7
ELMo-weighted	Ansell et al (2019)	61.2
Word-level contextualised embeddings	Implementation	Accuracy %
WSD†	Loureiro and Jorge (2019)	67.7
BERT-large	WIC's paper	65.5
Context2vec	WIC's paper	59.3
Elmo	WIC's paper	57.7
Sense representations		
LessLex	Colla et al (2020)	59.2
DeConf	WIC's paper	58.7
SW2V	WIC's paper	58.1
JBT	WIC's paper	53.6
Sentence level baselines		
Sentence Bag-of-words	WIC's paper	58.7
Sentence LSTM	WIC's paper	53.1
Random baseline		50.0

† Use external knowledge resources

4. ábra. This table shows various models and their performance in the WiC (Word-in-Context) NLP task

Key Points to Explain the Table This table ranks different NLP models based on their accuracy on the WiC task. Here's how to frame it effectively for your presentation:

1. Introduce the WiC Task The WiC task requires determining whether a target word has the same meaning in two different contexts. It's a challenging NLP problem because it combines elements of word sense disambiguation (WSD) and contextual embeddings.

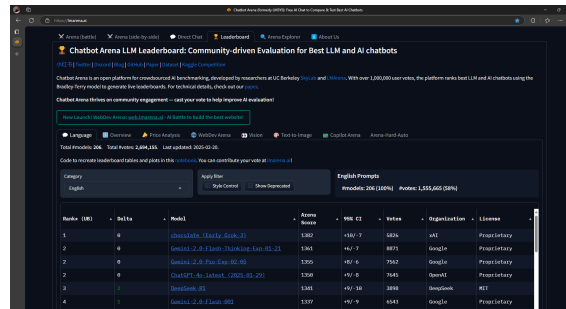
7. Eddigi legjobb megoldások

1. Nagy nyelvi modellek

8. *Mik a nagy nyelvi modellek?* A nagy nyelvi modell (angolul Large Language Model, LLM) olyan számítási modell, amely képes nyelv generálására vagy más természetes nyelvi feldolgozási feladatok elvégzésére. Ezek a modellek hatalmas mennyiségű szöveges adat feldolgozásával és mélytanulási technikák alkalmazásával sajátítják el a nyelv megértését és előállítását. Az LLM-ek, mint például az OpenAI GPT-sorozata, a Google PaLM vagy a Meta LLaMA modelljei, különböző architektúrákat használnak, de leggyakrabban a transzformer alapú megközelítést alkalmazzák. Ezek a modellek képesek különféle feladatok elvégzésére, mint például szövegfordítás, összefoglalás, kérdés-válasz és kreatív szövegalkotás. Azonban fontos megjegyezni, hogy az LLM-ek teljesítménye és alkalmazhatósága nagymértékben függ a betanításukhoz használt adatok minőségétől és mennyiségétől. *Miért használtam őket a feladat megoldásához?* Saját, ipari modellekhez hasonló minőségű készítése körülményes, amelyhez sem időm, sem kapacitásom, sem hardveres erőforrásaim nem voltak. A nagy cégek nagy nyelvi modelljei előnye, hogy hatalmas adathalmazon be vannak tanítva, szinte mindenre láttak már példát - talán ezt a Pilehvar féle WiC adathalmazt is látták már, hiszen része a SuperGlue benchmarknak, mellyel számos modellt tesztelnek (de lehet, hogy tanítják is a train adathalmazzal).

9. 1.1. Chatbot Arena, vagy LMSYS - a kiértékelő platform

A különböző nagy nyelvi modellek összehasonlítására a <https://lmarena.ai/> platformot használtam. Ez az oldal minden ismertebb ingyenes nagy nyelvi modellhez hozzá-



Rank	Model	Elo	Win %	Loss %	Draw %	Organization	License
1	GPT-4o	1200	68%	28%	4%	OpenAI	Proprietary
2	GPT-4o mini	1195	65%	30%	5%	OpenAI	Proprietary
3	Gemini-1.5 Pro	1190	64%	31%	5%	Google	Proprietary
4	Gemini-1.5 Flash	1185	63%	32%	5%	Google	Proprietary
5	Llama 3.3 70B	1180	62%	33%	5%	Meta	Proprietary

5. ábra. Llm-ek rangsora

férést biztosít, pl. GPT-3.5, GPT-4o, GPT-4o mini, Gemini-flash, Gemini-pro, Gemini-flash-pro-thinking, Llama, grok, Deepseek-r1 és r3, Claude Sonnet, Claude Haiku. Az elérhető modellek listája folyamatosan változik, a weboldalon érhető el.

10. Közösség rangsor

A "ranking" dropdown menüben az opciók közül az "English" kategóriát választottam, mert a problémám ehhez a területhez (domainhez) áll a legközelebb. A közösség által a következő sorrendben voltak rangsorolva a nagy nyelvi modellek (2025.02.25-én, a teszt időpontjában):

10.1. 1.1.1. A beadott szöveg eredete

A nagy nyelvi modellek kiértékelésénél nagyon nem mindegy, hogy honnan vesszük a kiértékelő adathalmazt. A https://pilehvar.github.io/wic/package/WiC_dataset.zip megfelelő értékelési referenciaadatkészlet, hiszen szakértők által gondosan annotált példákon alapul. Mivel a nagy nyelvi modellek már be vannak tanítva, és validálva vannak, ezért a kiértékelésükhöz a tanító (train) és a validációs (dev) adathalmazokra nem volt szükség, csak a teszt (test) halmazra.

A elsősorban saját modell lokális tanítására lett kitalálva, emiatt a rajta található dev/train/test adathalmazok minden önálló tokenként tekintenek a következőkre is:

1. vessző,
2. mondatvégi pont,
3. az angol birtokos személyjel 's rag,
4. és a n't tagadószócska.

Ez azt jelentette, hogy ezek a tokenek szóközzel el vannak választva a megelőző és következő tokenről. Én azonban a kutatómunkám első fázisaként a weben, távoli szerveren



```
1 def make_sentence_human_readable(sentence): 2 usage: # Fabian Senet
3 """Replaces contractions for better readability both for humans and for chatbots."""
4 sentence = sentence.replace(" 's", "'s")
5 sentence = sentence.replace(" .", ".")
6 sentence = sentence.replace("n't", "not")
7 return sentence
```

6. ábra. C:\PycharmProjects\Peternity2\modules\wic_sentence_normalizer.py

futó modelleken szerettem volna tesztelni, ahova a mondatok ilyen formában nem feleltek meg a kiértékelésre, mert pont az volt a célom, hogy megvizsgáljam, mennyire értik meg jól a nagy nyelvi modellek az emberi szöveget. Emiatt a listában felsorolt tokeneknek az előfordulásainál átalakítottam a szöveget: kitöröltem az előttük lévő szóközt. Ezzel lényegében visszaalakítottam a mondatokat emberi nyelvre. Erre az alábbi szkriptet használtam:

10.2. 1.1.2. A beadott szöveg formátuma

10.2.1. 1.1.2.1 Egy mondat formátuma

A prompt szótár formátuma az alábbi:

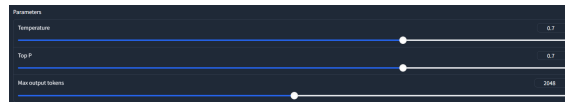
```
'Does the word "defeat" mean the same thing in sentences "It was
'Does the word "groom" mean the same thing in sentences "Groom th
'Does the word "penetration" mean the same thing in sentences "Th
```

11. Kérdés-válasz szótár

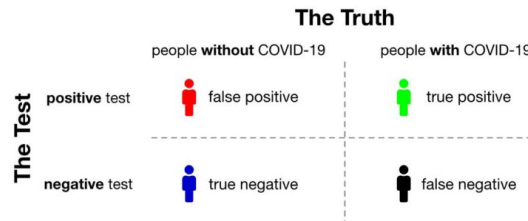
A naiv kérdés-válasz készítési ötlet az lehetne, hogy a vizsgált szótő legyen a kulcs, és a kérdés a válasz. Ez azonban nem célszerű, sőt hibás, ugyanis több mondatpár is lehet az adathalmazban ugyanazzal a szótővel. Fontos, hogy a kérdés-válasz szótárban a kérdéseknek kell lenniük a kulcsoknak, a 'Yes' vagy 'No' válaszoknak pedig az értékneeknek, és nem fordítva. f'Does the word "word" mean the same thing in sentences "sentence_a" and "sentence_b"?'

11.0.1. 1.1.2.1 A beadott mondatok száma

Alapos tesztelés után azt találtam, hogy x példára oldják meg a nagy nyelvi modellek a legjobb pontossággal a feladatot.



7. ábra. Lmsys direct chat állítható paraméterei



8. ábra. Covid példa a TP, TN, FP, FN alkalmazására

11.1. A nagy nyelvi modellek konfigurása a prompt beküldése előtt

12. Klasszifikációs módszer

A chatbotok teljesítményét a TP, TN, FP, FN klasszifikációs módszerrel értékeltem ki. Covid esetén pozitív a teszt, ha a páciens beteg, negatív, ha egészséges. True, ha helyesen ítéltük meg az egészségi állapotát, és false, ha helytelenül ítéltük meg.

A llmek esetében pozitív, ha a szó ugyanazt jelenti mind a két mondatban, negatív, ha eltérő dolgot. True, ha helyesen ítélte meg a llm a szó jelentését, false, ha helytelenül ítélte meg.

13. "True positive: Sick people correctly identified as sick"

"False positive: Healthy people incorrectly identified as sick"

"True negative: Healthy people correctly identified as healthy"

"False negative: Sick people incorrectly identified as healthy"

13.1. Megvizsgált modellek

A Imareana.ai platformon az alábbi paraméterezéssel teszteltem a nagy nyelvi modelleket. a Temperature 0.3-ra állítva, a Precision 1-re állítva, a max output tokens paramétert pedig 4096-ra állítva. Ez elengedhetetlen volt a megfelelő minőségű válasz eléré-

Large Language Models Table

1. táblázat. Hivatalos llm platformok és a rajtuk használt modellek

	A	B	C	D	E
	Model	Version	Max input tokens	Max output tokens	
1	OpenAI	GPT-3.5 Turbo	Each one has 4 "subjects": Normal, Creative, Explanatory and Formal	up to 200,000 tokens in the context window	
2	OpenAI	GPT-4	General	4096 tokens per request	
3	OpenAI	GPT-4o	Fast special reasoning		
4	OpenAI	GPT-4o mini	Fast for everyday tasks		
5	Google	Gemini 1.5			
6	Google	Gemini 1.5 Flash			
7	Google	Gemini 1.5 Pro			
8	Google	Gemini 1.5 Ultra			
9	Google	Gemini 1.5 Flash Thinking Experimental			
10	Google	Gemini 1.5 Flash Thinking Experimental with Apps			
11	Google	Gemini 1.5			
12	Meta	Llama 3.1			
13	Meta	Llama 3.1 Instruct			
14	Meta	Llama 3.1 Instruct 8B			
15	Meta	Llama 3.1 Instruct 70B			
16	Meta	Llama 3.1 Instruct 70B			
17	Meta	Llama 3.1 Instruct 70B			
18	Meta	Llama 3.1 Instruct 70B			
19	Meta	Llama 3.1 Instruct 70B			
20	Meta	Llama 3.1 Instruct 70B			
21	Meta	Llama 3.1 Instruct 70B			
22	Meta	Llama 3.1 Instruct 70B			
23	Meta	Llama 3.1 Instruct 70B			
24	Meta	Llama 3.1 Instruct 70B			
25	Meta	Llama 3.1 Instruct 70B			
26	Meta	Llama 3.1 Instruct 70B			
27	Meta	Llama 3.1 Instruct 70B			
28	Meta	Llama 3.1 Instruct 70B			
29	Meta	Llama 3.1 Instruct 70B			
30	Meta	Llama 3.1 Instruct 70B			
31	Meta	Llama 3.1 Instruct 70B			
32	Meta	Llama 3.1 Instruct 70B			
33	Meta	Llama 3.1 Instruct 70B			
34	Meta	Llama 3.1 Instruct 70B			
35	Meta	Llama 3.1 Instruct 70B			
36	Meta	Llama 3.1 Instruct 70B			
37	Meta	Llama 3.1 Instruct 70B			
38	Meta	Llama 3.1 Instruct 70B			
39	Meta	Llama 3.1 Instruct 70B			
40	Meta	Llama 3.1 Instruct 70B			
41	Meta	Llama 3.1 Instruct 70B			
42	Meta	Llama 3.1 Instruct 70B			
43	Meta	Llama 3.1 Instruct 70B			
44	Meta	Llama 3.1 Instruct 70B			
45	Meta	Llama 3.1 Instruct 70B			
46	Meta	Llama 3.1 Instruct 70B			
47	Meta	Llama 3.1 Instruct 70B			
48	Meta	Llama 3.1 Instruct 70B			
49	Meta	Llama 3.1 Instruct 70B			
50	Meta	Llama 3.1 Instruct 70B			
51	Meta	Llama 3.1 Instruct 70B			
52	Meta	Llama 3.1 Instruct 70B			
53	Meta	Llama 3.1 Instruct 70B			
54	Meta	Llama 3.1 Instruct 70B			
55	Meta	Llama 3.1 Instruct 70B			
56	Meta	Llama 3.1 Instruct 70B			
57	Meta	Llama 3.1 Instruct 70B			
58	Meta	Llama 3.1 Instruct 70B			
59	Meta	Llama 3.1 Instruct 70B			
60	Meta	Llama 3.1 Instruct 70B			
61	Meta	Llama 3.1 Instruct 70B			
62	Meta	Llama 3.1 Instruct 70B			
63	Meta	Llama 3.1 Instruct 70B			
64	Meta	Llama 3.1 Instruct 70B			
65	Meta	Llama 3.1 Instruct 70B			
66	Meta	Llama 3.1 Instruct 70B			
67	Meta	Llama 3.1 Instruct 70B			
68	Meta	Llama 3.1 Instruct 70B			
69	Meta	Llama 3.1 Instruct 70B			
70	Meta	Llama 3.1 Instruct 70B			
71	Meta	Llama 3.1 Instruct 70B			
72	Meta	Llama 3.1 Instruct 70B			
73	Meta	Llama 3.1 Instruct 70B			
74	Meta	Llama 3.1 Instruct 70B			
75	Meta	Llama 3.1 Instruct 70B			
76	Meta	Llama 3.1 Instruct 70B			
77	Meta	Llama 3.1 Instruct 70B			
78	Meta	Llama 3.1 Instruct 70B			
79	Meta	Llama 3.1 Instruct 70B			
80	Meta	Llama 3.1 Instruct 70B			
81	Meta	Llama 3.1 Instruct 70B			
82	Meta	Llama 3.1 Instruct 70B			
83	Meta	Llama 3.1 Instruct 70B			
84	Meta	Llama 3.1 Instruct 70B			
85	Meta	Llama 3.1 Instruct 70B			
86	Meta	Llama 3.1 Instruct 70B			
87	Meta	Llama 3.1 Instruct 70B			
88	Meta	Llama 3.1 Instruct 70B			
89	Meta	Llama 3.1 Instruct 70B			
90	Meta	Llama 3.1 Instruct 70B			
91	Meta	Llama 3.1 Instruct 70B			
92	Meta	Llama 3.1 Instruct 70B			
93	Meta	Llama 3.1 Instruct 70B			
94	Meta	Llama 3.1 Instruct 70B			
95	Meta	Llama 3.1 Instruct 70B			
96	Meta	Llama 3.1 Instruct 70B			
97	Meta	Llama 3.1 Instruct 70B			
98	Meta	Llama 3.1 Instruct 70B			
99	Meta	Llama 3.1 Instruct 70B			
100	Meta	Llama 3.1 Instruct 70B			

9. ábra. Nagy nyelvi modellek összehasonlítása

séhez. A feladat megértéséhez elengedhetetlen, hogy pár az alábbi fogalmakkal tisztában legyek. Maximum Output Tokens és Context Window a Nagy Nyelvi Modellekben A "maximum output tokens" a modell által generált kimeneti szöveg leghosszabb engedélyezett hossza. Ez a paraméter kizárólag a válasz hosszát korlátozza, és nem befolyásolja a bemeneti adatok hosszát közvetlenül.

Fontos jellemzők:

A maximum output tokens értéke a kimeneti szöveg hosszának felső korlátját adja meg.

Ha ez az érték alacsony, a modell nem tud részletes vagy hosszú választ adni.

Ha túl magasra állítjuk, a modell felesleges vagy redundáns szöveget is generálhat. A "context window" az a maximális szöveghossz, amelyet a modell egyszerre képes figyelmebe venni a bemeneti adatok feldolgozása során. Ezt általában tokenekben mérik, ahol egy token lehet egy teljes szó, szótöredék vagy akár egy írásjel is, attól függően, hogy a modell milyen tokenizálási technikát alkalmaz.

Fontos jellemzők:

A context window hossza határozza meg, hogy a modell milyen hosszú szövegeket képes értelmezni és kontextusba helyezni.

Ha a szöveg hossza meghaladja ezt a korlátot, a modell levágja vagy elhagyja a szöveg egy részét, ami információvesztéshez vezethet.

Például a GPT-3 esetében a context window alapértelmezetten 2048 token, míg a GPT-4 esetében ez 8192 vagy akár 32 768 token is lehet egyes verziókban.

A Imarena.ai webalkalmazáson kívül a nyelvi modelleket fejlesztő vállalatok hivatalos oldalain található modell legújabb verzióit is teszteltem, hogy mennyire jól oldják meg a WiC feladatot. Az alábbi táblázat tartalmazza a használt platformokat és a tesztelt modelleket, platformonként.

Rengeteg interneten elérhető nagy nyelvi modellel lehet ingyenesen csevegni (szö-

veges promptot inputként beadni, melyre szöveggel tér vissza). Ezeknek a legfőbb előnye a lokális modell futtatással szemben, hogy nagyon gyorsak, és alig veszik igénybe az eszközünk erőforrásait - megjegyzendő, hogy egy gyenge minőségű általános célú nagy nyelvi modell futtatásához is ipari mennyiségű hardverre és erőforrásra van szükség.

Azonban sok hátrányuk is van. Sajnos azt kellett észrevennem, hogy ezek a személyes, vagy munkahelyi használatra tervezett platformok nem, vagy alig konfigurálhatóak. Emiatt alig javítható az általuk elért eredmény, és ha javul is, az inkább az emlékező képességüknek köszönhető. A modell másik felhasználó számára, illetve másik környezetben ugyanazokat a hibákat követi el, mint a személyes betanítás előtt - szemben a [lmarena.ai](#) oldallal, ahol garantált, hogy minden felhasználó számára ugyanabban a környezetben fut a chatbot. A másik probléma a nagy nyelvi modellek hivatalos platformjaival, hogy az algoritmus működésébe - vagy a mai modelleknél már úgy is mondhatjuk, hogy a nagy nyelvi modell *gondolkodásába* - nincsen megfelelő mértékű betekintése a fejlesztőnek. Ezt a problémát igyekeznek mitigálni pl. a Gemini-flash-thinking verziók, vagy a deepseek-r1 "részletes magyarázatot nyújtó" modelljei, azonban ez sem ad kellő irányítást. **Megoldás** Kvantált nyelvi modell letöltése és lokális futtatása. Megoldásként olyan nagy nyelvi modelleket kerestem, amelyek nyílt forráskódúak és elég kicsik ahhoz, hogy a személyi számítógépemen fussanak. Ennek a megtalálására a Hugging Face platformot használtam. A Hugging Face egy gépi tanulási (ML) és adatelemzési platform és közösség, amely segít a felhasználóknak gépi tanulási modellek építésében, telepítésében és betanításában. A Hugging Face-en számos ingyenes és open-source nyelvi modell található, amelyek elfutnak egy egyszerű asztali gépen. Ezek a kis méretük miatt könnyen félrevezethetők, de a célfeladat nem igényel általános tudást a modell részéről. Itt a hangsúly az angol szavak jelentésének a felismerésén van, erre a célra a kvantált modell tökéletesen megfelel. Hugging Face-re való bejelentkezés után elérhetővé válik 2 nagyon fontos feature: - Bármelyik modellt kipróbálhatjuk a webes felületen keresztül, - Generálhatunk saját Access Tokeneket, amelyekkel letölthetjük a modelleket a saját eszközünkre.

Az elérhető modelleket átnézve és a témavezetőmmel egyeztetve a gemma-2-2b-it modellt választottam. Ez a modell méret/minőség arányban egész jó, számos kvantált (a pontosság rovására kisebb méretűvé tett) variánsa létezik, és nagy fejlesztő közösség áll mögötte. Weben kipróbáltam a modellt. Beadtam neki a test'adathalmazból készített teljes promptot (1400 kérdéssel).

14. gemma-2-2b-it modell letöltése és installálása lokális használatra

A gemma modellt lokálisan egy 2022-es ASUS Vivobook 17 gépről használtam. Az eszköz hardveres tulajdonságai a következők: 256 GB SSD, 128 GB HDD, 8 GB RAM, 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz 3.00 GHz processzor, 4 mag, NVidia Intel CORE I3 processzor. Egy nagy nyelvi modell letöltése és telepítése közel sem egyszerű folyamat, még akkor sem, ha egy kvantált (a pontosság rovására kisebbé zsugo-

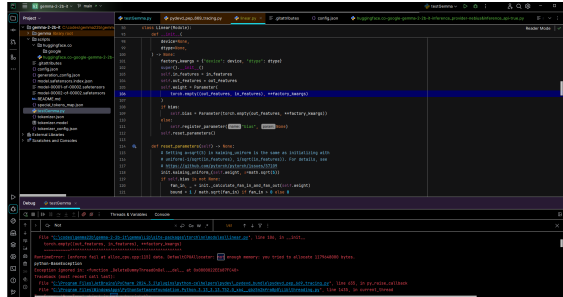
rított) verzióját telepítjük. Egy GPT-4o szintű nagy nyelvi modellt szinte lehetetlen lokálisan futtatni, ugyanis iszonyatosan nagy tárhelyet igényel (több terabájt (forrás:TODO)) és emellett iszonyatosan lassan válaszol. Előfordul, hogy fél óráig kell várni egy válaszra. A gemma-2-2b-it a git clone https://fabernat.<TOKEN_AZONOSITO>@huggingface.co/google/gemma-2-2b-it parancssal futtatva önmagában kb. 5 GB tárhelyet foglal el a gépen. Emellett a használatához szükséges eszközök további kb. 10 GB tárhelyet igényeltek, szóval le kellett törölnöm az alkalmazások felét az eszközömről.

A gemmához szükséges további eszközök: git lfs install - Git Large File Storage downloader: 512 MB-nál nagyobb fájlok fel- és letöltéséhez Github és a lokális verzió között. Ezután a python virtuális környezetet kellett létrehozni az alkalmazáshoz: A 'python -m venv gemma' parancssal. Ezután futtatni kellett az aktiváló szkriptet a 'gemma\Scripts\activate.bat' parancssal. majd a 'python -m pip cache remove *' és a 'python -m pip cache purge' parancsokkal kellett biztosítani a pip cache-ének eltávolítását, majd megtisztítását. Ezután a setuptools telepítése jött a frozenCache%% %% This is file '.tex', %% generated with the docstrip utility. %% %% The original source files were: %% %% fileerr.dtx (with options: 'return') %% %% This is a generated file. %% %% The source is maintained by the LaTeX Project team and bug %% reports for it can be opened at <https://latex-project.org/bugs/> %% (but please observe conditions on bug reports sent to that address!) %% %% Copyright (C) 1993-2023 %% The LaTeX Project and any individual authors listed elsewhere %% in this file. %% %% This file was generated from file(s) of the Standard LaTeX Tools Bundle'. %% —————

————— %% %% It may be distributed and/or modified under the %% conditions of the LaTeX Project Public License, either version 1.3c %% of this license or (at your option) any later version. %% The latest version of this license is in %% <https://www.latex-project.org/lppl.txt> %% and version 1.3c or later is part of all distributions of LaTeX %% version 2005/12/01 or later. %% %% This file may only be distributed together with a copy of the LaTeX %% Tools Bundle'. You may however distribute the LaTeX Tools Bundle' %% without such generated files. %% %% The list of all files belonging to the LaTeX Tools Bundle' is %% given in the file 'manifest.txt'. %% install setuptools Script file ./output.luabridge.lua not found finalizeCache parancssal, majd a torch megfelelő oprendszeres, bites, méretű és verziójú szoftverének az installálása következett a frozenCacheinstall torch torchvision torchaudio --index-url <https://download.pytorch.org/whl/cu118> Script file ./output.luabridge.lua not found finalizeCache parancssal. Ezután a frozenCacheinstall -U transformers Script file ./output.luabridge.lua not found finalizeCache, amely sokkal egyszerűbbé teszi a gemma használatát python kódban, majd végül a frozenCacheinstall accelerate Script file ./output.luabridge.lua not found finalizeCache segítségével az accelerate python könyvtár gyorsabbá teszi a modell kiértékelő folyamatát.

A lényege ugyanaz, mint mondjuk a chatGPT-nek. Beadsz neki egy szöveget és válaszol. Azért hívják modelleknek a modelleket, mert az emberi kommunikációt utánozzák (modellezik)

Előnyük, hogy nyílt forráskódúak, így tetszőlegesen módosíthatóak - habár alapos szakértelem kell hozzájuk. Emellett rendkívül biztonságosak: nincsenek rákötve a világhálóra, semmilyen szinten nem kommunikálnak a külvilággal, emiatt bármilyen privát adatot is nyugodtan be lehet nekik adni. Ezek nem általános célú modellek. Kvantáltak, ami azt jelenti, hogy mesterségesen le van csökkentve a méretük. Emiatt könnyen félrevezethető-



10. ábra. Nem volt elég memória a futás folytatásához

ek és könnyen hallucinálnak is.

A gemmának a hátránya, hogy iszonyatosan nagy tárhelyet igényel és iszonyatosan lassan válaszol, van, hogy 5 percig kell várni egy válaszra, azonban a célfeladatra megfelel. Általános célú feladatokra, például a házi feladat megoldására sokkal rosszabb válaszokat ad, mint pl. egy GPT-4, Claude Sonnet, Grok vagy Deepseek modelljei. A gemma-2-2b-it sem általános célú modell. Kvantált, ami azt jelenti, hogy mesterségesen le van csökkentve a mérete az alapjául szolgáló gemini-flash-pro-hoz képest. Emiatt könnyen félrevezethetőek és könnyen hallucinálnak is. Nekem csak 1 célfeladatra kellett, ezért alkalmasnak találtam.

15. Tesztelés fázis

Első futtatáskor hibát kaptam, mert nem volt a gépen tárhely, amit a program allokalni kellett volna.

"Miért kéne több szó a gépnek, mint egy gyereknek?" - Dr. Berend Gábor

A tervezett vizsgálatok során az alábbi nyelvi modellek kerülnek tesztelésre:

- **GPT-3.5, GPT-4o GPT-4o mini, gemini-flash, gemini-pro, gemini-flash-pro-thinking, Llama, Grok4, Deepseek-r1, Deepseek-r3, Claude Sonnet, Claude Haiku**
- **Gemma2-2b** és ennek kisebb méretű, kvantált változatai.
- **GPT-Neo**, a GPT-3 nyílt forráskódú változata.
- **OPT**, a Facebook által fejlesztett nagy nyelvi modell.
- **BLOOM**, a BigScience projekt többnyelvű LLM-je.
- Egyéb modellek, ha a kutatás során további releváns jelöltek merülnek fel.

15.1. 1.2. Benchmarking és tesztelés

Az összehasonlító teljesítményvizsgálatok az alábbi platformok segítségével történnek:

- LLM Arena
- Large Language Model (LLM) API Playground by Retool

Az eredmények kiértékelése során külön figyelmet fordítottunk az egyes modellek által adott válaszok minőségére, a sorrendiségre való érzékenységekre, valamint a magyar nyelvi feldolgozás pontosságára.

15.2. Konzisztencia

Modellek konzisztenciájának összehasonlítása: Mennyire érzékenyek arra, hogy a 2 mondatot fordított sorrendben adjuk be nekik, azaz az "A ugyanaz, mint B", illetve a "B ugyanaz, mint A" kérdések esetén milyen gyakran egyezik az adott válasz?

16. 2. Saját rendszer fejlesztése

16.1. Tervezés

16.1.1. Projekt tervezés fázis:

Fő szempontjaim voltak, hogy a projekt fájlstruktúrája könnyen értelmezhető legyen, kövesse a konvenciókat, könnyen lehessen benne tájékozódni, és logikusan legyen felépítve. Emiatt előre létrehoztam a szükséges mappákat, amelyekben a kódjaimat tárolni fogom. Az elnevezési szabályaim is rendkívül konzisztensek voltak: minden külső használatra szánt, úgynevezett

```
utility
```

python szkriptem a

```
wic_
```

előtaggal kezdődött. Amelyik pedig belső használatra volt szánva, az nem kapott

```
wic_
```

előtagot. Az ötletet az OpenCV cv2 függvénykönyvtárából vettem, amely többek között Python nyelven is használható. Itt ugyanis minden beimportálható függvény, enum, változó, konstans stb. a cv2. előtaggal kezdődik, amely rendkívül megkönnyíti a cv2 modulok elkülönítését a saját moduloktól, és ezáltal elősegíti kód olvashatóságát. Ezzel lényegében a KISS - keep it simple, stupid alapelvet követtem, amely egy bevált módszer arra, hogy sokszor primitívnek, vagy egyértelműnek tűnő dolgokat is sokszor ismétlünk, primitív módon egymás után felsorolunk, ám ez az olvashatóságot jelentősen javítja. A szkriptjeimen belül is követtem ezt az alapelvet.

A tervezés során számos lépést megtettem, hogy a projektem könnyen átlátható legyen:

- A harmadik féltől származó könyvtárak, csomagok és egyéb függőségek egységesen, requirements.txt-ben lettek definiálva. Ez egy Pythonos konvenció, amelyet én is követtem.
- A python szkript
- A projektben a lehető legkevesebb függőséget alkalmazok, tehát annyira moduláris, amennyire csak lehet. Ez különösen hasznos a projekt refaktorálásakor, illetve akkor, ha csak pár fájlt szeretnénk használni a projektből, mert ez esetben kell az egész projektet lehúzni githubról, csak a használni kívántakat.
- Kivétel erre függvények meghívása a saját függvénykönyvtár fájljaimból, és a konfigurációs beállítások (pl. Elérési utak) importálása a fájlokba

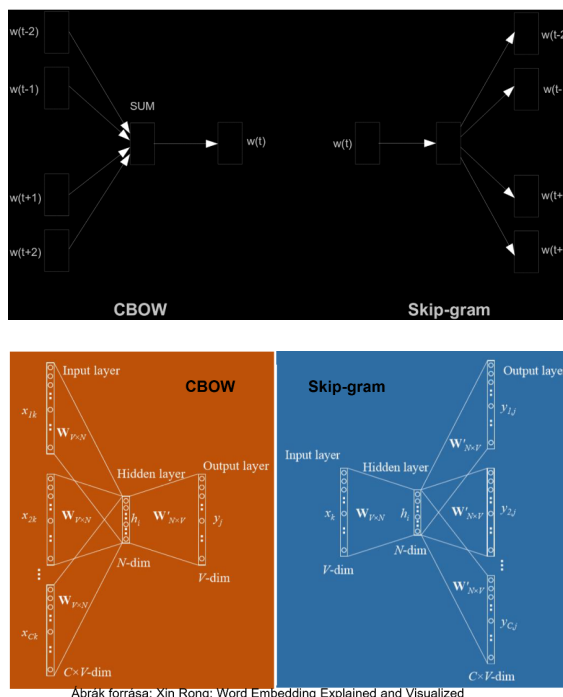
16.2. Implementáció

16.3. Tesztelés

2.1.1. word2vec

$$y(x) = \text{softmax}(W(W1x)) \quad (1)$$

- A word2vec célja, hogy a hasonló jelentésű input szavak hasonló outputot eredményezzenek. CBOW és Skipgram módszerek
- a és b szó jelentése minél hasonlóbb, $y(a)$ és $y(b)$ (eloszlás)vektorok annál inkább hasonlítani fognak
- CBOW: x „környező” szavak reprezentációi alapján akarjuk a „középső” szót előrejelezni $y(x)$ -szel
- Skipgram: x „középső” szó reprezentációja alapján akarjuk a „környező” szavakat előrejelezni $y(x)$ -szel



2.1.2. Vektortérmodell Szöveges tartalmak tömör reprezentációjára a vektortérmodell (VTM) nyújtja a legszélesebb körben használt megoldást. A modell minden egyes dokumentumot egy vektorral ír le, amelyben minden elem az egyes termek (általában szavak) előfordulását jelenti. Termek alatt a reprezentáció egységeit, alapesetben az írásjelek által határolt szavakat (unigram) értjük. Bizonyos módszerek több szóból álló kifejezéseket (n-gram) is alkalmaznak reprezentációként, amellyel általában jobban jellemezhetőek a dokumentumok, de jelentősen megnöveli a dokumentum feldolgozási idő- és tárigényét. A vektortérmodell egyik legelterjedtebb megvalósítása a bag-of-words (BoW) megközelítés, amely a dokumentumok reprezentálására kizárólag a termek előfordulási gyakoriságát veszi figyelembe, figyelmen kívül hagyva azok sorrendjét és kontextusát. Ennek egy továbbfejlesztett változata a TF-IDF (Term Frequency - Inverse Document Frequency) súlyozás, amely nemcsak a termék abszolút előfordulását veszi figyelembe egy adott dokumentumban, hanem azok relatív fontosságát is az egész korpuszon belül. A TF-IDF segítségével a gyakori, de kevésbé informatív szavak (például névelők, kötőszavak) kisebb súlyt kapnak, míg a ritkábban előforduló, de tartalmilag releváns szavak nagyobb szerephez jutnak.

Az utóbbi évek fejlődésével az olyan mélytanulási modellek, mint a BERT (Bidirectional Encoder Representations from Transformers) és más transzformátor alapú nyelvi modellek még fejlettebb szövegreprezentációt kínálnak, figyelembe véve a kontextust és a szavak közötti összefüggéseket.

2.2. WiC Decision Maker Model és Flask Alapú Webalkalmazás

Ez a fejezet bemutatja a WiC (Word-in-Context) döntéshozó modellt, valamint a Python Flask keretrendszer segítségével történő implementációját és alkalmazását.

16.4. 2.2.1. A WiC modell működése

A modell egy előtanított nyelvi modellt használ, amely képes a szóhasználatok közötti finom eltéréseket azonosítani. A bemenet két mondatból és a vizsgált szóból áll, míg a kimenet egy bináris döntés: azonos vagy eltérő jelentés.

16.5. 2.2.2. Adatfeldolgozás és annotáció

Az adatok előkészítése során szükség van megfelelő annotációs technikák alkalmazására. Az adatkészleteket standard WiC benchmarkokból nyerjük, amelyek kézi annotációval lettek ellenőrizve.

16.6. 2.2.3. Modell teljesítményének értékelése

A modell teljesítményét F1-mutatóval és pontossággal mérjük. A Flask alkalmazásban egy REST API teszi lehetővé az automatikus értékelést és interaktív használatot.

17. 2.2.4. A Flask keretrendszer alkalmazása

A Flask egy könnyűsúlyú Python-alapú webkeretrendszer, amely gyors fejlesztést tesz lehetővé. A WiC döntéshozó modellt egy API-n keresztül érhetjük el.

17.1. 2.2.5. Az API struktúrája

A Flask alkalmazás RESTful API-ként működik, amely JSON formátumban kapja és küldi az adatokat. Az alábbi végpontokat tartalmazza:

- `/predict` – Egy POST végpont, amely a bemeneti mondatokra visszaadja az osztályozási eredményt.
- `/health` – Egy GET végpont, amely az alkalmazás állapotát ellenőrzi.

17.2. 2.2.6. Backend implementáció

A Flask szerveren a kérelmfeldolgozás és a modellszólítás az API endpointok mentén történik. A WiC modell egy előre betanított Transformer architektúrán alapszik, amelyet a Hugging Face könyvtár segítségével integrálunk.

17.3. 2.2.7. Frontend lehetőségek

Bár a fő cél az API szolgáltatás, egy egyszerű webes interfész is készíthető a Flask beépített sablonmotorjával (Jinja2), amely interaktív vizsgálati lehetőséget biztosít.

18. 2.2.8. Összegzés és jövőbeli fejlesztések

A Flask alapú webalkalmazás hatékony módot biztosít a WiC feladat automatizált megoldására. A jövőbeli fejlesztések között szerepelhet egy fejlettebb frontend implementáció és nagyobb teljesítményű modellintegráció.

19. 2.2.9. Összegzés

A `WiC_decision_maker_model` célja, hogy átfogó elemzést nyújtson a különböző LLM-ek WiC feladatokban nyújtott teljesítményéről, különös tekintettel a magyar nyelvi feldolgozás sajátosságaira. A projekt hosszú távon hozzájárulhat a nyelvfeldolgozó modellek fejlesztéséhez és finomhangolásához.

20. Összefoglalás

4. Nyilatkozat

Alulírott Fábián Bernát, programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatot a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a TVSZ 4. sz. mellékletében leírtak szerint kezelik.

Szeged, 2025. március 21.

.....
aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani óvónőimnek, tanítóimnak, tanárainak, egyetemi tanárainak, akik végigkísértek utamon tanulmányaim alatt, és hozzásegítettek végső soron, hogy sikeres szakdolgozatot készíthessek. Köszönetet szeretnék továbbá nyilvánítani szüleimnek, testvéreimnek és barátaimnak, hogy mindenben támogattak.

Végül, de nem utolsó sorban köszönetet szeretnék mondani **témavezetőmnek, Berend Gábornak**, hogy konzulensként és témavezetőként segített a szakdolgozatom megírásában.

21. 3. Hivatkozások és források

Az alkalmazás fejlesztése és elemzése során az alábbi forrásokra támaszkodtam:

- WiC adatbázis
- Gemma2-2b-it modell
- Kvantált modellek
- Kvantálás elméleti háttér
 - A nagy nyelvi modellek definíciója:
Nagy nyelvi modell (LLM: Large Language Model) - YouTube
- 2. https://hu.wikipedia.org/wiki/Nagy_nyelvi_modellNagy nyelvi modell - Wikipédia
- 3. Mik a nagy nyelvi modellek (LLM-ek)? - Microsoft Azure
- 4. Mi az a nagy nyelvi modell? - Lexiq
- 5. Nagy Nyelvi Modellek (LLM) - Mi ez és hogyan alakítja a jövőt?
- 6. [PDF] A ChatGPT és más nagy nyelvi modellek (LLM-ek) alkalmazásának ...

Hivatkozások

[1] Irodalomjegyzék

Feladatléírás, valamint forrás a tanító, fejlesztő- és tesztelő adatbázishoz: WiC: The Word-in-Context Dataset Mohammad Taher Pilehvar honlapja: About Mohammad Taher Pilehvar

Online LLM referenciák:

Lokális nyelvi modell futtatáshoz használt toolok: Windows natív C/C++ compiler
CUDA Toolkit 12.8 Update 1 Downloads | NVIDIA Developer google/gemma-2-2b-it · Hugging Face PyTorch

@inproceedingspilehvar-camacho-collados-2019-wic, title = "WiC: the Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations", author = "Pilehvar, Mohammad Taher and Camacho-Collados, Jose", editor = "Burstein, Jill and Doran, Christy and Solorio, Thamar", booktitle = "Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)", month = jun, year = "2019", address = "Minneapolis, Minnesota", publisher = "Association for Computational Linguistics", url = "https://aclanthology.org/N19-1128/", doi = "10.18653/v1/N19-1128", pages = "1267–1273", abstract = "By design, word embeddings are unable to model the dynamic nature of words' semantics, i.e., the property of words to correspond to potentially different meanings. To address this limitation, dozens of specialized meaning representation techniques such as sense or contextualized embeddings have been proposed. However, despite the popularity of research on this topic, very few evaluation benchmarks exist that specifically focus on the dynamic semantics of words. In this paper we show that existing models have surpassed the performance ceiling of the standard evaluation dataset for the purpose, i.e., Stanford Contextual Word Similarity, and highlight its shortcomings. To address the lack of a suitable benchmark, we put forward a large-scale Word in Context dataset, called WiC, based on annotations curated by experts, for generic evaluation of context-sensitive representations. WiC is released in <https://pilehvar.github.io/wic/>."

[2] J. L. Gischer, The equational theory of pomsets. *Theoret. Comput. Sci.*, **61**(1988), 199–224.

[3] J.-E. Pin, *Varieties of Formal Languages*, Plenum Publishing Corp., New York, 1986.