

Magyar nyelvű Szentimentanalízis Projekt

Név

2025. május 11.

Tartalomjegyzék

1. Projekt Áttekintése	4
2. Módszertan	4
3. Dataset	4
3.1. huBERT bemutatása	4
3.2. A huBERT alkalmazási lehetőségei	5
4. Implementáció, technológiák	5
4.1. Alapvető Python könyvtárak	5
4.2. Adatgyűjtés és Feldolgozás	6
4.3. Adatbázis Kapcsolatok	6
4.4. Webes Felület	6
4.5. Machine Learning és NLP	6
4.5.1. PyTorch Könyvtárak	6
4.5.2. NLP-specifikus Könyvtárak	7
4.6. Adatelemzés	7
4.7. Konténerizáció	7
4.8. Függőségek	7
5. Alapvető szentimentanalízis modell elkészítése	7
5.1. 1. lépés: Az előre tanított BERT modell betöltése és a tanítás felparaméterezése	8
5.1.1. Batch size	8
5.1.2. Max length	8
5.1.3. Num classes és Label map	9
5.1.4. Epochs	9
5.1.5. Learning rate	9
5.2. 2. lépés: Adatok betöltése, előfeldolgozása, tokenizáció	9
5.2.1. Tokenizáció	10
5.3. 3. lépés: Tanítás, elkészült modell mentése	11
5.4. A modell tanítása közben keletkezett kimenet	12
6. Az elkészült modell felhasználása	13
6.1. Futtatás Python fájlból	14
6.2. Futtatás webes felülettel	14
6.3. Webes szolgáltatások	16

6.4. Adatbázis modell	20
6.5. Adatbázis kapcsolatok	20
6.6. Docker és Docker Compose	20
6.7. Docker fájlok	21
7. Kommenteken végzett szentimentanalízis	23
7.1. Web scraping	24
7.2. Docker Compose beállításai	25
7.3. Elért eredmények	27
7.4. Egyéb funkciók	29
8. Források	29

1. Projekt Áttekintése

A projekt célja egy magyar nyelvű szentimentanalízis modell fejlesztése Pythonban, amely a HuSST adatkészletet használja. A modellel szembeni elvárás, hogy képes legyen a szövegeket negatív, semleges és pozitív kategóriákba sorolni.

2. Módszertan

A cél megvalósításához a huBERT (*Hungarian Universal Bidirectional Encoder Representations from Transformers*) betanított neurális hálót fogom felhasználni alapmodellként. Az előre betanított neurális háló nagyon jó kiindulási alapként szolgál, mivel magyar nyelvű adatokon tanították, tehát általános magyar nyelvtudással rendelkezik. Képes a szövegek értelmezésére és feldolgozására, viszont általánosságban elmondható, hogy ezeket az alapmodelleket további tanítással kell kiegészíteni ha specifikusan egy bizonyos célra szeretnénk használni az alap tudását.

Jelen feladatban a HuSST adathalmazmal fogok további tanítást végezni a modellen. A HuSST, mint korábban említésre került, magyar nyelvű kijelentéseket tartalmaz és az azokhoz tartozó címkéket. A címke lehet negatív, semleges vagy pozitív. Ezek alapján kerül besorolásra az adott szöveg.

3. Dataset

A bevezetőben ismertetett két forrást fogom használni a projekt megvalósításához.

- huBERT base model (Hungarian Universal Bidirectional Encoder Representations from Transformers)
- HuSST dataset (Hungarian Stanford Sentiment Treebank)

3.1. huBERT bemutatása

A huBERT egy magyar nyelvű, transzformátor alapú nyelvi modell, amelyet a SZTAKI fejlesztett ki. A modell a BERT architektúrát követi, és kifejezetten a magyar nyelv sajátosságainak kezelésére optimalizálták. A tanítást az

úgynevezett *Common Crawl* adatbázis magyar nyelvű részén végezték szűrések és deduplikációk után, valamint a magyar Wikipédián alapulva. A modell 111 millió paraméterrel rendelkezik.

3.2. A huBERT alkalmazási lehetőségei

A huBERT modellt különféle magyar nyelvű NLP (*Natural Language Processing*) feladatokhoz használhatjuk:

- Szövegosztályozás
- Névfelismerés (NER (*Named Entity Recognition*))
- Szövegrészletezés (Chunking)
- Kérdésmegválaszolás
- Szöveggenerálás

4. Implementáció, technológiák

A projekt megvalósítása során Python nyelven dolgozom a gépi tanulás és a webes felület implementációjához. A modell fejlesztéséhez a PyTorch keretrendszert, az adatkezeléshez és előfeldolgozáshoz a pandas és numpy könyvtárakat, míg a tokenizáláshoz a Hugging Face transformers könyvtárat használok. A megoldás konténerizálását Docker segítségével oldom meg. A webes felület Flask webserverral készül, míg az adatok tárolása PostgreSQL adatbázisban történik.

A modell és a ráépülő webes rendszer Pythonban készül a következő könyvtárak felhasználásával:

4.1. Alapvető Python könyvtárak

- `os`: Operációs rendszer szintű műveletek (fájlkezelés, környezeti változók)
- `json`: JSON adatok szerializálása és deszerializálása
- `re`: Reguláris kifejezések a szövegfeldolgozáshoz (regex)

- `time`: Időzíti műveletek és késleltetések
- `logging`: Alkalmazás naplózásának konfigurálása
- `zlib`: Adattömörítés és kicsomagolás

4.2. Adatgyűjtés és Feldolgozás

- `requests`: HTTP kérések küldése és fogadása
- `BeautifulSoup`: HTML és XML dokumentumok elemzése
- `concurrent.futures`: Párhuzamos feldolgozás megvalósítása
- `urllib.parse`: URL címek kezelése

4.3. Adatbázis Kapcsolatok

- `psycopg2`: PostgreSQL adatbázis-kezelőhöz való csatlakozás
- `SQLAlchemy`: ORM (*Object-Relational Mapping*) rendszer
- `datetime`: Dátum és időkezelés

4.4. Webes Felület

- `Flask`: Mikrokeretrendszer webalkalmazás fejlesztéséhez
- `flask_login`: Felhasználói munkamenetek kezelése
- `werkzeug.security`: Jelszavak biztonságos tárolása és ellenőrzése

4.5. Machine Learning és NLP

4.5.1. PyTorch Könyvtárak

- `torch`: Tenzorműveletek és GPU támogatás
- `torch.nn`: Neurális hálók építéséhez szükséges modulok
- `torch.optim`: Optimalizálási algoritmusok (Adam, SGD)
- `torch.utils.data`: Adatbetöltés és előfeldolgozás

4.5.2. NLP-specifikus Könyvtárak

- `transformers`: Előtanított nyelvi modellek kezelése
- `datasets`: Nagy nyelvi adathalmazok betöltése és kezelése
- `sklearn.metrics`: Osztályozási metrikák számítása

4.6. Adatelemzés

- `jupyter`: Interaktív notebook környezet
- `jupyterlab`: Fejlettebb notebook felület
- `pandas`: Adatok táblázatos kezelése és elemzése
- `numpy`: Numerikus számítások és tömbműveletek
- `tqdm`: Folyamatjelző sáv iterációkhoz

4.7. Konténerizáció

- `Docker`: Alkalmazás konténerbe csomagolása
- `Docker Compose`: Többkonténeres alkalmazások kezelése

4.8. Függőségek

A projekt függőségeit a `requirements.txt` fájl tartalmazza.

5. Alapvető szentimentanalízis modell elkészítése

A szentimentanalízis modell elkészítése több fő lépésből áll, ezek bemutatása fog következni.

5.1. 1. lépés: Az előre tanított BERT modell betöltése és a tanítás felparaméterezése

Első lépésként a kiválasztott nyelvhez illeszkedő előre tanított neurális háló betöltésére van szükség. Jelen esetben a magyar nyelvfeldolgozáshoz a SZTAKI-HLT/hubert-base-cc modellre esett a választás. A konkrét megvalósítás szemléltetése érdekében beilleszttem az alábbi kódrészletet, ahol a felparaméterezés látható.

```
1 # Configuration - Using a publicly available Hungarian model
2 MODEL_NAME = "SZTAKI-HLT/hubert-base-cc" # Public Hungarian
  BERT model
3 BATCH_SIZE = 16
4 MAX_LENGTH = 128
5 EPOCHS = 3
6 LEARNING_RATE = 2e-5
7 NUM_CLASSES = 3 # negative, neutral, positive
8 LABEL_MAP = {"negative": 0, "neutral": 1, "positive": 2} #
  Create label mapping
9 DEVICE = torch.device("cuda" if torch.cuda.is_available()
  else "cpu")
```

Listing 1. Modell konfiguráció

5.1.1. Batch size

A kódrészletben a *Batch size* paraméter határozza meg, hogy egy *Epoch-ban* hány minta legyen felhasználva a tanításhoz. A jelenlegi 16-os *batch size* azt jelenti, hogy ekkora csomagokban fog zajlani a tanítás. Ez az érték kiegyensúlyozott a memóriahasználát és a tanítási sebesség között.

5.1.2. Max length

A *Max length* paraméter 128-as értéke azt állítja be, hogy legfeljebb 128 tokenet tartalmazhasson egy minta. Pontosabban kifejezve, egy adott bemenet hossza legfeljebb 128 tokenből állhat, ahol egy token például egy szónak, szórészletnek, vagy írásjelnek feleltethető meg. A tokenek konkrét hossza és a tokenizáció menete a eltérő lehet különböző nyelvek között. Jelenleg a rövidebb szövegeket *padding* egészíti ki, a hosszabbak csonkolásra kerülnek.

5.1.3. Num classes és Label map

A *Num classes* és a *Label map* a tanítás során használni kívánt kategóriákat határozza meg. Esetünkben három kategória létezik: a negatív, semleges, és pozitív. A szöveges címkéket numerikus értékekre képezi le, ami szükséges a neurális háló számára.

5.1.4. Epochs

Az *Epochs* a tanítási iterációk számát határozza meg. Egy epoch azt jelenti, hogy a teljes tanító adathalmazon egyszer végighaladt a modell. Túl sok epoch túltanításhoz (*overfitting*) vezethet, míg kevesebb epoch alultanítást (*underfitting*) eredményez. A jelenleg megadott érték elegendő lehet egy megfelelő tanításhoz, erőforrások hiányában nem növelem, mert azzal jelentősen növekedne a tanításhoz szükséges idő is.

5.1.5. Learning rate

A *Learning rate* a tanulási ráta, ami meghatározza, hogy mennyit változzon a modell súlya egy lépésben. Túl magas érték instabil tanításhoz vezet, míg túl alacsony érték lassú konvergálást eredményez. Jelenleg egy általánosan elfogadott érték került beállításra.

5.2. 2. lépés: Adatok betöltése, előfeldolgozása, tokenizáció

Az előző fejezetben ismertetett paraméterek beállítása után elkezdhetjük az adatok betöltését. Egyfelől az előre betanított huBERT neurális hálót, másfelől a tanításhoz szükséges HuSST címkézett adatokat.

1. Adathalmaz betöltése a Hugging Face `datasets` könyvtárával
2. Szövegek tokenizálása a huBERT tokenizálóval
3. PyTorch DataLoader-ek létrehozása a tanításhoz

A HuSST tanító, validációs és teszt adathalmazból áll a korábban ismertett felépítéssel: egy magyar kijelentéshez vagy negatív, vagy semleges, vagy pozitív címke tartozik. Szemléltetésképp egy részlet a tanítási adathalmazból:

```

1 [
2   {
3     "text": "Azonban hiányzik belőle az a nagyság és
4     hősiesség, ami Stevensont és a korábbi Disney-meséket
5     jellemzi.",
6     "label": "negative"
7   },
8   {
9     "text": "Informatív, ha sok beszédes részt
10    tartalmaz egy dokumentumfilm.",
11    "label": "neutral"
12  },
13  {
14    "text": "Ha szeretsz időnként moziba menni, é
15    rdemes a Wasabi-val kezdeni.",
16    "label": "positive"
17  }
18 ]

```

Listing 2. Minta a HuSST adathalmazból

5.2.1. Tokenizáció

A HuSST tokenizációját a HuBERT előre tanított tokenizálója végzi el. Ennek segítségével helyesen lesznek tagolva a szavak a tanításhoz használt szöveg betöltésekor.

```

1 # Initialize tokenizer
2 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
3
4 # Create datasets
5 train_dataset = HungarianSentimentDataset(
6     train_texts, train_labels, tokenizer, MAX_LENGTH
7 )
8 # Create dataloaders
9 train_loader = DataLoader(
10     train_dataset, batch_size=BATCH_SIZE, shuffle=True
11 )

```

Listing 3. Tokenizáció

5.3. 3. lépés: Tanítás, elkészült modell mentése

A folyamat végső lépéseként elkezdhető az új modell betanítása a korábban bemutatottak segítségével.

A *train epoch* függvény felelős a modell egy epoch-on keresztüli tanításáért. A függvény először a modellt tanítási módba állítja, majd inicializálja a veszteség és az előrejelzések nyilvántartását. A tanítási ciklus a megadott adatokon halad végig, ahol minden köteget adatra három fő lépést hajt végre: az adatok mozgatása a megfelelő eszközre (*CPU/GPU, jelenleg csak CPU áll rendelkezésre a tanításhoz*), a forward és backward propagáció végrehajtása, valamint a paraméterek frissítése az optimizer segítségével. A veszteségfüggvény *CrossEntropyLoss* értékelésével és a gradiensek visszaszámításával a modell súlyait finomhangolja.

```
1 # 3. Training functions
2 def train_epoch(model, data_loader, optimizer, device):
3     model.train()
4     total_loss = 0
5     correct_predictions = 0
6
7     for batch in tqdm(data_loader, desc="Training"):
8         input_ids = batch['input_ids'].to(device)
9         attention_mask = batch['attention_mask'].to(device)
10        labels = batch['label'].to(device)
11
12        optimizer.zero_grad()
13        outputs = model(input_ids, attention_mask)
14        loss = nn.CrossEntropyLoss()(outputs, labels)
15        loss.backward()
16        optimizer.step()
17
18        total_loss += loss.item()
19        _, preds = torch.max(outputs, dim=1)
20        correct_predictions += torch.sum(preds == labels)
21
22    accuracy = correct_predictions.double() / len(data_loader
23    .dataset)
24    avg_loss = total_loss / len(data_loader)
25    return avg_loss, accuracy
```

Listing 4. Tanítási folyamat

A folyamat utolsó szakaszában a modell tényleges betanítása történik meg a korábban definiált komponensek felhasználásával. A tanítási folyamat

során a modell többször feldolgozza a teljes tanító adathalmazt, miközben a veszteségfüggvény minimalizálására törekszik. Minden futás után értékelni lehet a modell teljesítményét a validációs halmazon, ami lehetővé teszi a túlilleszkedés (*overfitting*) felismerését. Az elkészült modellt a diszldre kerül mentésre, hogy predikciókat lehessen vele végezni a későbbiekben. A mentés során nem csak a modell súlyait, hanem a tokenizálót és a konfigurációs paramétereket is érdemes elmenteni.

5.4. A modell tanítása közben keletkezett kimenet

```

1 C:\Temp\py\hu-sent\venv\Scripts\python.exe C:\Temp\py\hu-sent
  \learn.py
2
3 Epoch 1/3
4 -----
5 Training: 100%|=====| 583/583 [2:23:42<00:00, 14.79s/it]
6 Train Loss: 0.6952, Accuracy: 0.6944
7 Evaluation: 100%|=====| 73/73 [05:27<00:00, 4.49s/it]
8 Val Loss: 0.5807, Accuracy: 0.7399
9 Saved new best model
10
11 Classification Report:
12 Training: 0%|          | 0/583 [00:00<?, ?it/s]
13      precision    recall  f1-score   support
14
15  negative         0.75         0.91         0.82         697
16  neutral          0.76         0.48         0.59         435
17  positive         0.40         0.55         0.46          33
18
19      accuracy                0.74        1165
20  macro avg          0.64         0.65         0.62        1165
21  weighted avg       0.75         0.74         0.73        1165
22
23 Epoch 2/3
24 -----
25 Training: 100%|=====| 583/583 [2:14:58<00:00, 13.89s/it]
26 Train Loss: 0.4737, Accuracy: 0.8001
27 Evaluation: 100%|=====| 73/73 [05:00<00:00, 4.12s/it]
28 Val Loss: 0.6953, Accuracy: 0.6884
29
30 Classification Report:
31      precision    recall  f1-score   support

```

```

32
33     negative      0.79      0.80      0.80      697
34     neutral      0.68      0.50      0.58      435
35     positive      0.18      0.82      0.30      33
36
37     accuracy
38     macro avg      0.55      0.71      0.56      1165
39     weighted avg    0.74      0.69      0.70      1165
40
41
42 Epoch 3/3
43 -----
44 Training: 100%|=====| 583/583 [2:11:09<00:00, 13.50s/it]
45 Train Loss: 0.3142, Accuracy: 0.8765
46 Evaluation: 100%|=====| 73/73 [04:58<00:00, 4.08s/it]
47 Val Loss: 0.7588, Accuracy: 0.7425
48 Saved new best model
49
50 Classification Report:
51           precision    recall  f1-score   support
52
53     negative      0.80      0.86      0.83      697
54     neutral      0.74      0.56      0.64      435
55     positive      0.27      0.70      0.39      33
56
57     accuracy
58     macro avg      0.60      0.71      0.62      1165
59     weighted avg    0.76      0.74      0.74      1165
60
61
62 Training complete. Best validation accuracy: 0.7425
63
64 Process finished with exit code 0

```

6. Az elkészült modell felhasználása

A betanított, elkészült modellt egy egyszerű *Python* fájlban is tudjuk használni, vagy *Jupyter Notebookban*. Ezek a módszerek alapvetően jók tesztelésre vagy további fejlesztésekhez, de nem túl felhasználóbarátok.

A könnyű használhatóság érdekében fontosnak tartottam valamilyen felhasználói grafikus interfész implementálását, ehhez a legegyszerűbben elkészíthető, multiplatform megoldást választottam: a webes felületet. A felület

segítségével felhasználói oldalon bármilyen eszközzel használható a rendszer, amelyen van böngésző. A webszervert és a modellt *Docker* konténerben lehet futtatni *docker compose* segítségével.

6.1. Futtatás Python fájlból

Első körben a kipróbálás és tesztelés legegyszerűbb módja, a predikcióhoz készült *Python* fájl futtatása. Ennek egy példája látható a következő sorokban.

```
(venv) C:\Temp\py\hu-sent>python prediction.py
Text: Ez a film fantasztikus volt!
Sentiment: positive
```

```
Text: Nem tetszett a könyv.
Sentiment: negative
```

```
Text: Átlagos élmény volt, semmi különös.
Sentiment: negative
```

```
Text: Süt a nap.
Sentiment: neutral
```

```
Text: Esik az eső.
Sentiment: neutral
```

```
Text: Szép időnk van ma.
Sentiment: positive
```

6.2. Futtatás webes felülettel

A webes verzió futtatásához *Docker compose* segítségével el kell indítani az alábbi szolgáltatásokat:

- **Adatbázis szerver** (PostgreSQL): Felhasználói adatok és előzmények tárolása
- **API szerver** (Python Flask): A szentiment elemzés végrehajtása REST API-n keresztül

- **Webszerver (Flask):** Felhasználói felület megjelenítése

Futtatáskor a *Docker* az alábbi kimenetet adja, amiből meggyőződhetünk róla, hogy minden szolgáltatás megfelelően el tudott indulni és elérhető. Ha mindez megtörtént, böngészővel tudunk csatlakozni a kiszolgáló IP címén, az 5000-es porton futó webszerverhez.

```
1 (venv) C:\Temp\py\hu-sent\sentimentapp>docker compose up
2 time="2025-05-03T12:16:32+02:00" level=warning msg="C:\\Temp
   \\py\\hu-sent\\sentimentapp\\docker-compose.yml: the
   attribute 'version' is obsolete, it will be ignored,
   please remove it to avoid potential confusion"
3 [+] Running 3/3
4 Container sentimentapp-db-1 Created
   0.0s
5 Container sentimentapp-sentiment-api-1 Created
   0.0s
6 Container sentimentapp-web-client-1 Created
   0.0s
7 Attaching to db-1, sentiment-api-1, web-client-1
8 db-1 |
9 db-1 | PostgreSQL Database directory appears to
   contain a database; Skipping initialization
10 db-1 |
11 db-1 | 2025-05-03 10:16:34.041 UTC [1] LOG:
   starting PostgreSQL 13.20 (Debian 13.20-1.pgdg120+1) on
   x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14)
   12.2.0, 64-bit
12 db-1 | 2025-05-03 10:16:34.041 UTC [1] LOG:
   listening on IPv4 address "0.0.0.0", port 5432
13 db-1 | 2025-05-03 10:16:34.041 UTC [1] LOG:
   listening on IPv6 address ":::", port 5432
14 db-1 | 2025-05-03 10:16:34.043 UTC [1] LOG:
   listening on Unix socket "/var/run/postgresql/.s.PGSQL
   .5432"
15 db-1 | 2025-05-03 10:16:34.049 UTC [27] LOG:
   database system was interrupted; last known up at
   2025-05-02 18:35:34 UTC
16 db-1 | 2025-05-03 10:16:34.196 UTC [27] LOG:
   database system was not properly shut down; automatic
   recovery in progress
17 db-1 | 2025-05-03 10:16:34.198 UTC [27] LOG:
   redo starts at 0/160C470
```

```

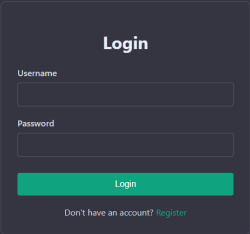
18 db-1 | 2025-05-03 10:16:34.198 UTC [27] LOG:
    invalid record length at 0/160C558: wanted 24, got 0
19 db-1 | 2025-05-03 10:16:34.198 UTC [27] LOG:
    redo done at 0/160C520
20 db-1 | 2025-05-03 10:16:34.212 UTC [1] LOG:
    database system is ready to accept connections
21 sentiment-api-1 | * Serving Flask app 'app'
22 sentiment-api-1 | * Debug mode: off
23 sentiment-api-1 | WARNING: This is a development server. Do
    not use it in a production deployment. Use a production
    WSGI server instead.
24 sentiment-api-1 | * Running on all addresses (0.0.0.0)
25 sentiment-api-1 | * Running on http://127.0.0.1:5000
26 sentiment-api-1 | * Running on http://172.18.0.3:5000
27 sentiment-api-1 | Press CTRL+C to quit
28 sentiment-api-1 | 127.0.0.1 - - [03/May/2025 10:17:09] "HEAD
    /health HTTP/1.1" 200 -
29 web-client-1 | * Serving Flask app 'app'
30 web-client-1 | * Debug mode: on
31 web-client-1 | WARNING: This is a development server. Do
    not use it in a production deployment. Use a production
    WSGI server instead.
32 web-client-1 | * Running on all addresses (0.0.0.0)
33 web-client-1 | * Running on http://127.0.0.1:5000
34 web-client-1 | * Running on http://172.18.0.4:5000
35 web-client-1 | Press CTRL+C to quit
36 web-client-1 | * Restarting with stat
37 web-client-1 | * Debugger is active!
38 web-client-1 | * Debugger PIN: 118-356-955
39 sentiment-api-1 | 127.0.0.1 - - [03/May/2025 10:17:39] "HEAD
    /health HTTP/1.1" 200 -

```

6.3. Webes szolgáltatások

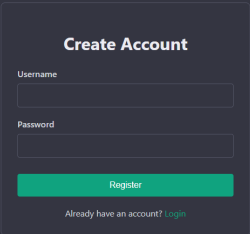
A rendszer rendelkezik felhasználókezeléssel, regisztrációs és bejelentkezési felülettel, valamint a felhasználók chatelőzményeinek tárolásával.

Néhány kép működés közben:



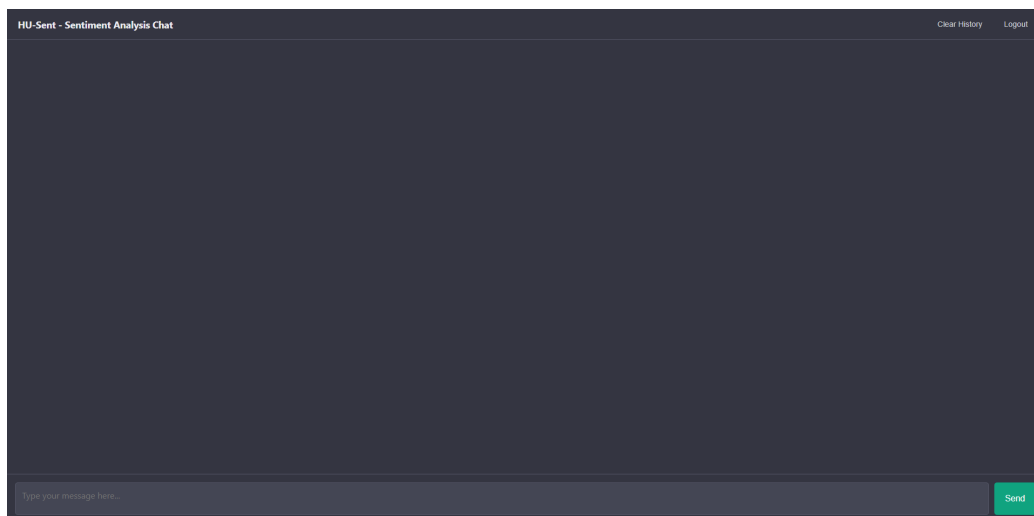
The image shows a login form centered on a dark blue background. The form is a light gray rectangle with a white border. At the top, the word "Login" is written in bold. Below it, there are two input fields: "Username" and "Password". Each field has a small icon of a person and a lock respectively. Below the fields is a green button with the word "Login" in white. At the bottom, there is a link that says "Don't have an account? [register](#)".

1. ábra. Bejelentkezési felület

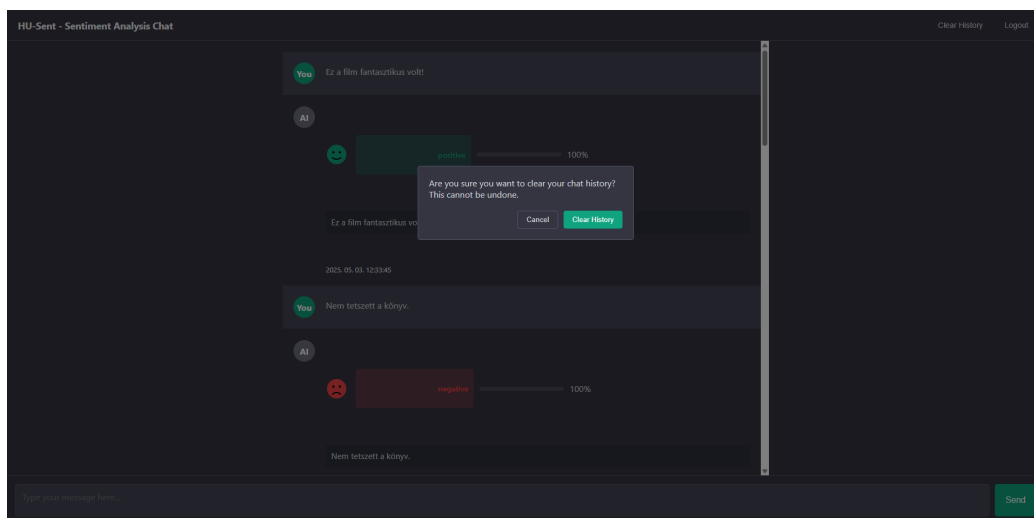


The image shows a "Create Account" form centered on a dark blue background. The form is a light gray rectangle with a white border. At the top, the words "Create Account" are written in bold. Below it, there are two input fields: "Username" and "Password". Each field has a small icon of a person and a lock respectively. Below the fields is a green button with the word "Register" in white. At the bottom, there is a link that says "Already have an account? [Login](#)".

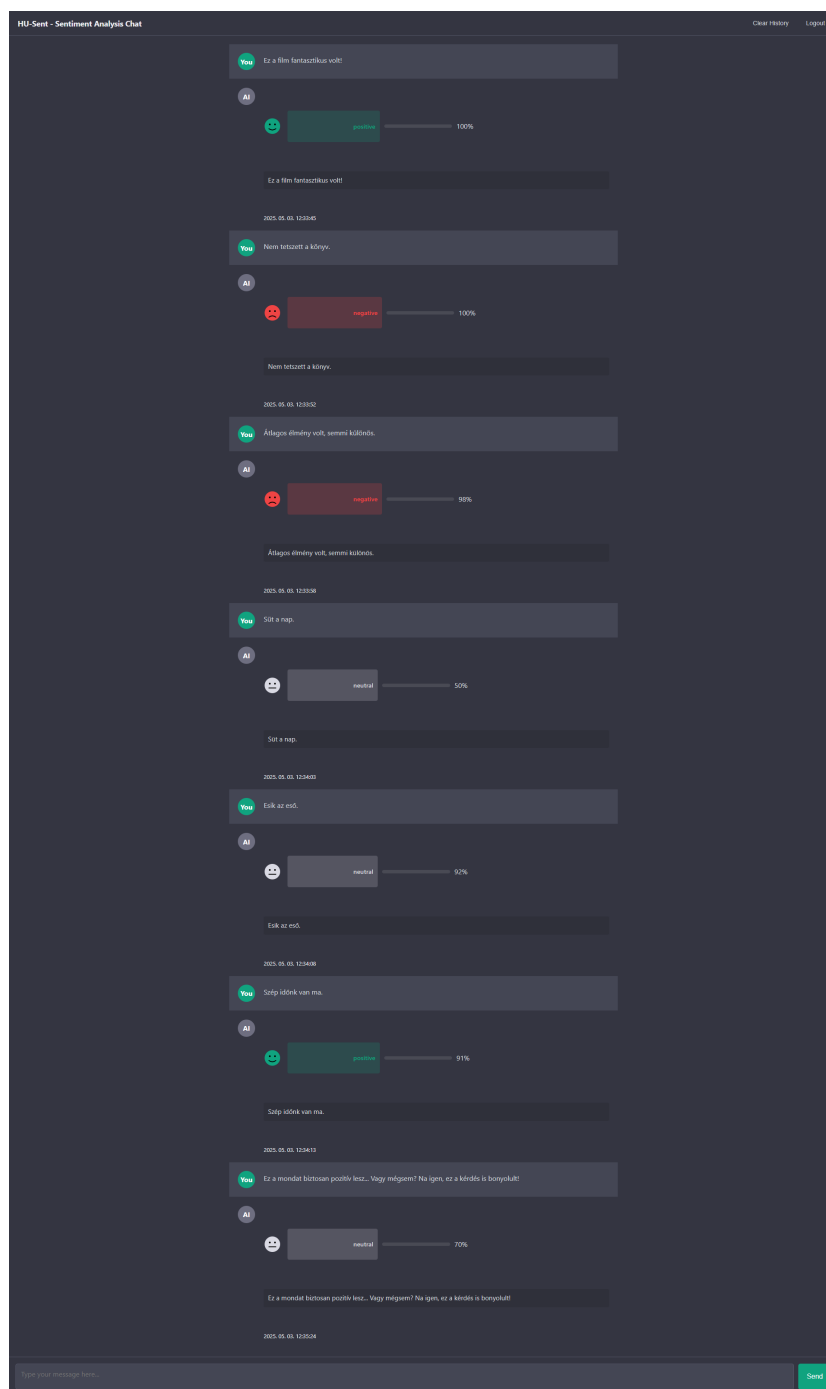
2. ábra. Regisztrációs felület



3. ábra. Üres chatfelület



4. ábra. Előzmények törlése



5. ábra. Chatfelület néhány példával

6.4. Adatbázis modell

A rendszer két fő táblát használ:

1. táblázat. Felhasználói tábla (**User**)

Mező	Típus	Leírás
id	Integer	Elsődleges kulcs
username	String(100)	Egyedi felhasználónév
password_hash	String(200)	Titkosított jelszó
created_at	DateTime	Regisztráció időpontja
chats	Relationship	A felhasználó chatüzenetei

2. táblázat. Chat tábla (**Chat**)

Mező	Típus	Leírás
id	Integer	Elsődleges kulcs
user_id	Integer	Külső kulcs a User táblához
message	String(1000)	Felhasználó üzenete
sentiment	String(20)	Érzelmi értékelés eredménye
confidence	Float	A modell bizonyossága
created_at	DateTime	Üzenet időpontja

6.5. Adatbázis kapcsolatok

- Egy felhasználó (**User**) több chatüzenettel (**Chat**) rendelkezhet
- Minden chatüzenet pontosan egy felhasználóhoz tartozik

6.6. Docker és Docker Compose

Ahogy az korábban említve lett, a webes szentimentanalízis alkalmazás *Docker* konténerizációval készült, a könnyű telepítés, hordozhatóság és reprodukálhatóság érdekében. A konténerizált környezet kiváló lehetőségeket nyújt a fejlesztés, tesztelés és üzemeltetés területén. A függőségek konténerekbe csomagolásával megszűnnek a kompatibilitási problémák és az alkalmazás

bármely Docker-t támogató platformon futtatható. A szolgáltatások szükség szerint akár skálázhatók is.

A szolgáltatások a Docker által biztosított belső hálózaton kommunikálnak egymással, a *depends_on* direktívák pedig biztosítják a megfelelő indítási sorrendet. Jelenleg:

- Elsőként az adatbázis szerver
- Másodikként a szentiment API
- Végül a webszerver indul

A kötetek (*volumes*) használata garantálja az adatmegőrzést a konténerek újraindítása esetén is. Perzisztenciára jelenleg csak az adatbázis esetében van szükség.

A *Docker Compose* segítségével a három szükséges konténer kezelése egybe van szervezve, és a beállításaik tárolva vannak. Amennyiben a *Docker* fájlok és a *Docker Compose* fájl jól van elkészítve, a program indításához mindössze a `docker compose up -build` parancsot kell kiadni, és automatikusan lefutnak a folyamatok.

6.7. Docker fájlok

```
1  version: '3.8'
2
3  services:
4    db:
5      image: postgres:13
6      environment:
7        POSTGRES_USER: sentiment_user
8        POSTGRES_PASSWORD: sentiment_pass
9        POSTGRES_DB: sentiment_db
10     volumes:
11       - postgres_data:/var/lib/postgresql/data
12     ports:
13       - "5432:5432"
14     healthcheck:
15       test: ["CMD-SHELL", "pg_isready -U sentiment_user -
16         d sentiment_db"]
17       interval: 5s
17       timeout: 5s
18       retries: 5
```

```

19
20     sentiment-api:
21         build:
22             context: .
23             dockerfile: model/Dockerfile
24         ports:
25             - "5001:5000"
26         environment:
27             - MODEL_NAME=SZTAKI-HLT/hubert-base-cc
28             - MAX_LENGTH=128
29             - DATABASE_URL=postgresql://sentiment_user:
sentiment_pass@db:5432/sentiment_db
30         depends_on:
31             db:
32                 condition: service_healthy
33         healthcheck:
34             test: ["CMD-SHELL", "wget -q --spider http
://127.0.0.1:5000/health || exit 1"]
35             interval: 30s
36             timeout: 10s
37             retries: 3
38             restart: unless-stopped
39
40     web-client:
41         build:
42             context: .
43             dockerfile: client/Dockerfile
44         ports:
45             - "5000:5000"
46         depends_on:
47             sentiment-api:
48                 condition: service_healthy
49             db:
50                 condition: service_healthy
51         environment:
52             - API_URL=http://sentiment-api:5000/predict
53             - DATABASE_URL=postgresql://sentiment_user:
sentiment_pass@db:5432/sentiment_db
54             - SECRET_KEY=your-secret-key-here
55         restart: unless-stopped
56
57     volumes:
58         postgres_data:

```

Listing 5. docker-compose.yml

```

1 FROM python:3.9-slim
2
3 WORKDIR /app
4
5 # Install necessary system dependencies
6 RUN apt-get update && apt-get install -y gcc python3-dev
  curl wget && rm -rf /var/lib/apt/lists/*
7
8 # Copy the requirements.txt specific to the model service
9 COPY model/requirements.txt /app/requirements.txt
10
11 # Install dependencies
12 RUN pip install --no-cache-dir -r /app/requirements.txt
13
14 # Copy the rest of the model code to the container's /app
  directory
15 COPY model /app
16
17 # Set the command to run the application (ensure the app.
  py is at /app/app.py)
18 CMD ["python", "/app/app.py"]

```

Listing 6. Sentiment API Dockerfile

```

1 FROM python:3.9-slim
2
3 WORKDIR /app
4
5 COPY client/requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 COPY client/ /app
9
10 RUN mkdir -p instance
11
12 CMD ["python", "app.py"]

```

Listing 7. Webserver Dockerfile

7. Kommenteken végzett szentimentanalízis

A projekt egyik ötlete egy szentimentanalízisre szolgáló webes chatfelület kialakítása, míg a másik megközelítés weboldalakról származó szövegek és

kommentek automatikus gyűjtése valamint azok elemzése, majd egy webes felületen az eredmények megjelenítése.

7.1. Web scraping

Az ötlet megvalósításához szükség van egy adatforrásra, ahonnan nagy mennyiségben hozzászólásokat lehet gyűjteni. Erre a célra egy fórum oldal felhasználását tűztem ki, ahol körülbelül 50.000 komment érhető el. A hozzászólások letöltéséhez egy erre a *web scrapingre* szánt Python scriptet készítettem, ami *HTTP* kérésekkel tölti le a weboldalról az adatokat. Az oldal eredeti formája *AJAX* kéréseket használ a kommentek betöltéséhez, így ezeket az *API* hívásokat Pythonban elkészítve, majd a válaszként érkezett *HTML* kódot megtisztítva *JSON* fájlokat tudtam előállítani. A *JSON* az alábbi mezőkkel rendelkezik:

```
1  {
2  "id_placeholder": {
3    "metadata": {
4      "id": 0,
5      "title": "Title Placeholder",
6      "description": "Description Placeholder",
7      "url": "https://example.com",
8      "tags": [
9        "tag1",
10       "tag2"
11      ],
12      "download_link": null
13    },
14    "comments": [
15      {
16        "id": "comment_id",
17        "user": {
18          "username": "Username Placeholder",
19          "profile_url": "https://example.com/user",
20          "avatar": "https://example.com/avatar.jpg",
21          "avatar_local": "path/to/avatar.jpg"
22        },
23        "timestamp": "Timestamp Placeholder",
24        "text": "Comment text placeholder",
25        "upvotes": 0,
26        "replies": [],
27        "has_more_replies": false,
28        "reply_count": 0
```



```

29     }
30   ]
31 }
32 }

```

7.2. Docker Compose beállításai

A teljes rendszer indításához több konténerre is szükség van. Elsőként a *PostgreSQL* adatbázist kell elindítani, majd a szkriptet ami importálja a *JSON* fájlokban rögzített adatokat az adatbázisba. Utána a szentimentanalízis *API-t*, következésként a szkriptet ami elküldi az *API-nak* az adatbázisban rögzített kommenteket, végül a webes felületet szolgáltató webalkalmazást ami az adatbázissal kommunikál. A perzisztencia érdekében egy *docker volume* csatolására is szükség van.

```

1  services:
2  # Main comment database (only service with volume)
3  db:
4    image: postgres:13
5    environment:
6      POSTGRES_USER: comment_user
7      POSTGRES_PASSWORD: comment_password
8      POSTGRES_DB: comments_db
9    volumes:
10     - postgres_data:/var/lib/postgresql/data
11    ports:
12     - "5432:5432"
13    healthcheck:
14     test: ["CMD-SHELL", "pg_isready -U comment_user -d
15      comments_db"]
16     interval: 5s
17     timeout: 5s
18     retries: 5
19  importer:
20    build:
21     context: .
22     dockerfile: ./importer/Dockerfile
23    environment:
24     DATABASE_URL: "postgresql://comment_user:
25      comment_password@db:5432/comments_db"
26    depends_on:
27     db:

```

```

27         condition: service_healthy
28     restart: "no"
29
30 sentiment_api:
31     build:
32         context: .
33         dockerfile: ./model/Dockerfile
34     ports:
35         - "5001:5000"
36     environment:
37         - MODEL_NAME=SZTAKE-HLT/hubert-base-cc
38         - MAX_LENGTH=128
39     healthcheck:
40         test: [ "CMD-SHELL", "curl -f http://localhost:5000/
health || exit 1" ]
41         interval: 10s
42         timeout: 5s
43         retries: 3
44     depends_on:
45         importer:
46             condition: service_completed_successfully
47
48 sentiment_processor:
49     build:
50         context: .
51         dockerfile: ./sentiment_processor/Dockerfile
52     environment:
53         DATABASE_URL: "postgresql://comment_user:
comment_password@db:5432/comments_db"
54         SENTIMENT_API_URL: "http://sentiment_api:5000/predict"
55     depends_on:
56         sentiment_api:
57             condition: service_healthy
58     restart: unless-stopped
59
60 webapp:
61     build:
62         context: ./webapp
63         dockerfile: Dockerfile
64     environment:
65         DATABASE_URL: "postgresql://comment_user:
comment_password@db:5432/comments_db"
66     ports:
67         - "5000:5000"
68     depends_on:

```

```

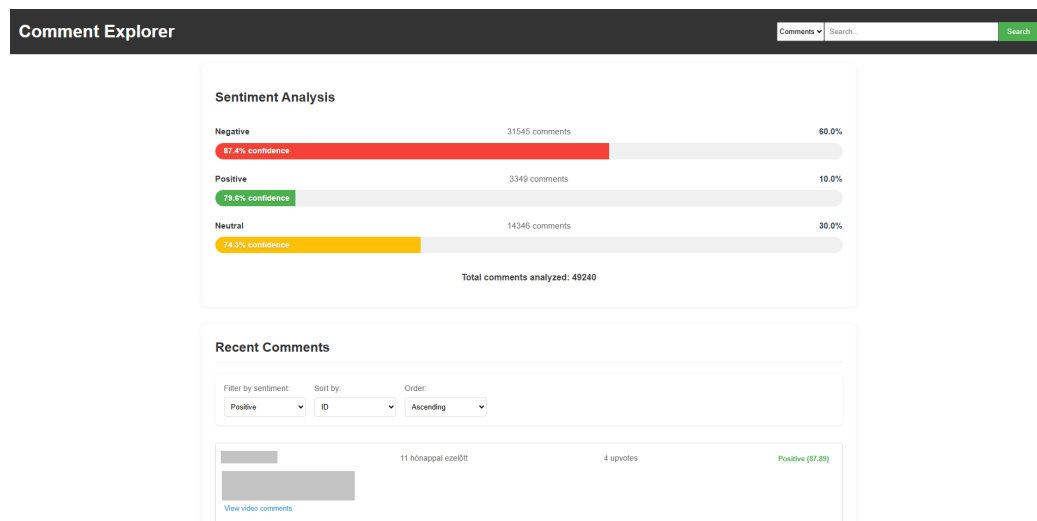
69 db:
70     condition: service_healthy
71     sentiment_processor:
72     condition: service_started
73
74 volumes:
75     postgres_data:

```

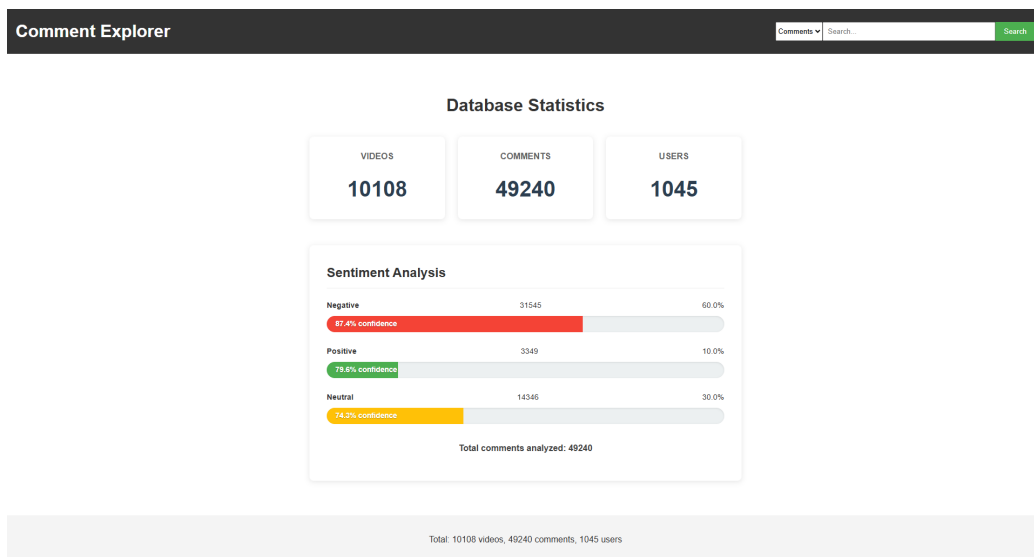
7.3. Elért eredmények

A kijelölt oldalról körülbelül 10.000 *json* fájl került előállításra és 50.000 komment kiértékelésre. A fórumon lévő kommenteket szentimentanalízis alá vetve a következő eredményeket láthatjuk:

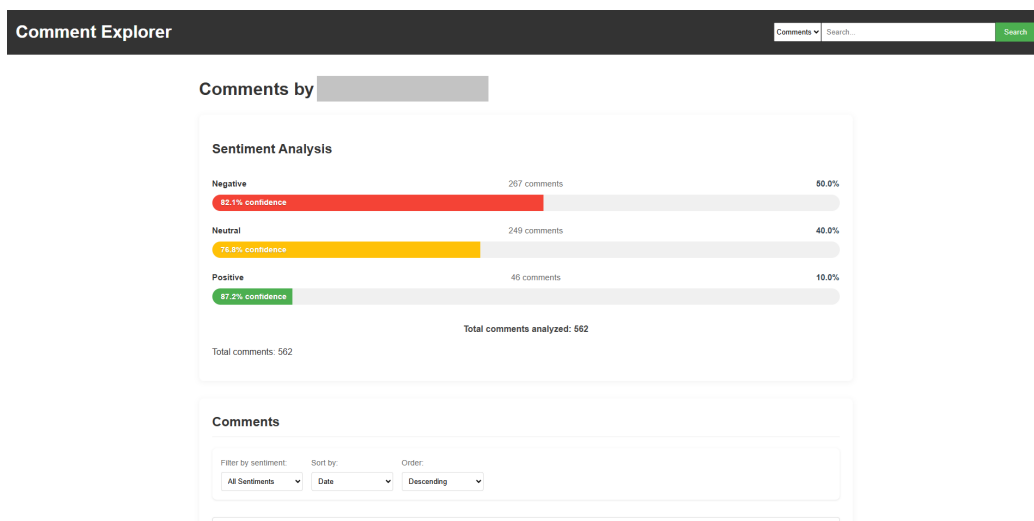
- **Negatív kommentek száma:** 31 500 (*87% konfidencia*)
- **Semleges kommentek száma:** 14 300 (*74% konfidencia*)
- **Pozitív kommentek száma:** 3 300 (*80% konfidencia*)



6. ábra. Komment szentimentanalízis



7. ábra. Komment szentimentanalízis statisztika



8. ábra. Felhasználó szentimentanalízise

7.4. Egyéb funkciók

- Keresés a kommentek és felhasználók között
- Találatok szűrése és rendezése
- Felhasználó szentiment elemzése a kommentjei alapján

8. Források

A dokumentumot az alább felsorolt források segítségével készítettem el.

Hivatkozások

- [1] SZTAKI-HLT. *hubert-base-cc*. Hugging Face.
<https://huggingface.co/SZTAKI-HLT/hubert-base-cc>
- [2] NYTK. *HuSST Dataset*. Hugging Face.
<https://huggingface.co/datasets/NYTK/HuSST>
- [3] SZTAKI-HLT. *huBERT - Hungarian BERT Model*. BME-HLT.
<https://hlt.bme.hu/hu/resources/hubert>
- [4] Orosz György. *Awesome Hungarian NLP Resources*. GitBook.
<https://oroszgy.gitbook.io/awesome-hungarian-nlp-resources>
- [5] Orosz György. (2023). *Awesome Hungarian NLP*. GitHub.
<https://github.com/oroszgy/awesome-hungarian-nlp>
- [6] Laki László J., Yang Zijian Győző. (2022). *huBERT - Hungarian BERT*. Acta Universitatis Óbuda.
https://acta.uni-obuda.hu/Laki_Yang_134.pdf
- [7] Docker Inc. *Install Docker Engine on Debian*. Docker Documentation.
<https://docs.docker.com/engine/install/debian/>
- [8] Docker Inc. *Linux post-installation steps for Docker Engine*. Docker Documentation.
<https://docs.docker.com/engine/install/linux-postinstall/>

- [9] Wikipedia contributors. *BERT (language model)*. Wikipedia.
[https://en.wikipedia.org/wiki/BERT_\(language_model\)](https://en.wikipedia.org/wiki/BERT_(language_model))
- [10] Shaikh, Rayyan. *A Comprehensive Guide to Understanding BERT: From Beginners to Advanced*. Medium.
<https://medium.com/@shaikhrayyan123/a-comprehensive-guide-to-understanding-bert-from-beginners-to-advanced-237969>
- [11] NYTK. *sentiment-hts5-hubert-hungarian*. Hugging Face.
<https://huggingface.co/NYTK/sentiment-hts5-hubert-hungarian>