Magyar nyelvű Szentiment Analízis Projekt

Név

2025. május 3.

Tartalomjegyzék

1. Projekt Áttekintés

Ez a projekt célja egy magyar nyelvű szentiment analízis modell fejlesztése Pythonban, amely a HuSST adatkészletet használja. A modell felé elvárás, hogy képes legyen szövegeket negatív, semleges és pozitív kategóriákba sorolni.

2. Módszertan

A cél megvalósításához a huBERT 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 a tudását.

Jelen feladatban a HuSST adathalmazzal fogok további tanítást végezni a modellen. A HuSST mint korábban említsre került, magyar nyelvű kijelentéseket tartalmaz és az azokhoz tartozó címkét. 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 Wikipedia alapján. 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
- Névvelentismerés (NER *(Named Entity Recognition)*)
- Szövegrészletezés (Chunking)
- Kérdésmegválaszolás
- Szöveggenerálás

4. Implementáció

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`: Időzítési 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 idő kezelése

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`: Tenzor műveletek 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

- `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 szentiment analízis modell elkészítése

A szentiment analízis 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 konrét megvalósítása szemléltetése érdekében beillesztem az alábbi kódrészletet, ahol a felparaméterezés látható.

```
# Configuration - Using a publicly available Hungarian model
MODEL_NAME = "SZTAKI-HLT/hubert-base-cc"  # Public Hungarian BERT model
BATCH_SIZE = 16
MAX_LENGTH = 128
EPOCHS = 3
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 memóriahasználat és a tanítási sebesség között.

5.1.2. Max length

A *Max length* paraméter 128-as értéke azt állítja be, hogy legfeljebb 128 tokent 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 közt. 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 elégséges 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 ismertetett felépítéssel: egy magyar kijelentéshez vagy negatív, vagy semleges, vagy pozitív címke tartozik. Szemléltetésképp egy részlet a tanítási adathalmazból:

```
[
    {
        "text": "Azonban hiányzik belőle az a nagyság és
    hősiesség, ami Stevensont és a korábbi Disney-meséket
    jellemzi.",
        "label": "negative"
    },
    {
        "text": "Informatív, ha sok beszédes részt
    tartalmaz egy dokumentumfilm.",
        "label": "neutral"
    },
    {
        "text": "Ha szeretsz időnként moziba menni, é
    rdemes a Wasabi-val kezdeni.",
        "label": "positive"
```

```
    }
]
```

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.

```
# Initialize tokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

# Create datasets
train_dataset = HungarianSentimentDataset(
    train_texts, train_labels, tokenizer, MAX_LENGTH
)
# Create dataloaders
train_loader = DataLoader(
    train_dataset, batch_size=BATCH_SIZE, shuffle=True
)
```

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 kötegelt 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, és a paraméterek frissítése az optimizer segítségével. A veszteségfüggvény *CrossEntropyLoss* értékelésével és a gradiensek visszaszámításával a modell súlyait finomhangolja.

```
# 3. Training functions
def train_epoch(model, data_loader, optimizer, device):
    model.train()
```

```
    total_loss = 0
    correct_predictions = 0

    for batch in tqdm(data_loader, desc="Training"):
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask)
        loss = nn.CrossEntropyLoss()(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        _, preds = torch.max(outputs, dim=1)
        correct_predictions += torch.sum(preds == labels)

    accuracy = correct_predictions.double() / len(data_loader
    .dataset)
    avg_loss = total_loss / len(data_loader)

    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 diszlre 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.

6. Az elkészült modell felhasználása

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

A könnyű használhatóság érdekében fontosnak tartottam valamilyen fel-

használói grafikus interfész implementálását, ehhez a legegyszerűbben elkészíthető, multi-platform megoldást választottam: a web-es felületet. A felület segítségével felhasználói oldalon bármilyen eszközről 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 Web-es felülettel

A Web-es 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, 5000-es porton futó webszerverhez.

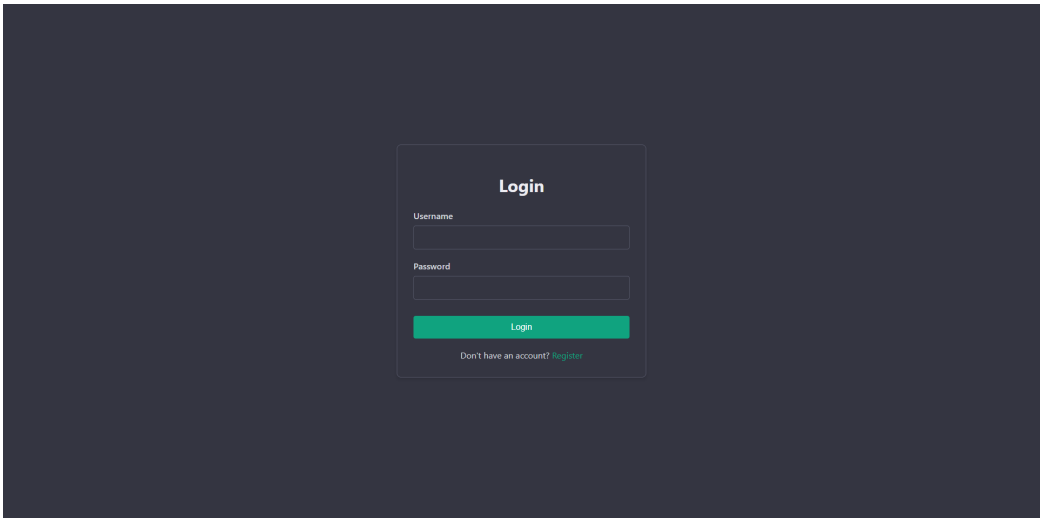
```
(venv) C:\Temp\py\hu-sent\sentimentapp>docker compose up
time="2025-05-03T12:16:32+02:00" level=warning msg="C:\\Temp
    \\py\\hu-sent\\sentimentapp\\docker-compose.yml: the
    attribute `version` is obsolete, it will be ignored,
    please remove it to avoid potential confusion"
[+] Running 3/3
  Container sentimentapp-db-1             Created

                         0.0s
  Container sentimentapp-sentiment-api-1  Created

                         0.0s
  Container sentimentapp-web-client-1     Created

                         0.0s
Attaching to db-1, sentiment-api-1, web-client-1
db-1             |
db-1             | PostgreSQL Database directory appears to
    contain a database; Skipping initialization
db-1             |
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
db-1             | 2025-05-03 10:16:34.041 UTC [1] LOG:
    listening on IPv4 address "0.0.0.0", port 5432
db-1             | 2025-05-03 10:16:34.041 UTC [1] LOG:
    listening on IPv6 address "::", port 5432
db-1             | 2025-05-03 10:16:34.043 UTC [1] LOG:
    listening on Unix socket "/var/run/postgresql/.s.PGSQL
    .5432"
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
```

```
db-1             | 2025-05-03 10:16:34.196 UTC [27] LOG:
    database system was not properly shut down; automatic
    recovery in progress
db-1             | 2025-05-03 10:16:34.198 UTC [27] LOG:
    redo starts at 0/160C470
db-1             | 2025-05-03 10:16:34.198 UTC [27] LOG:
    invalid record length at 0/160C558: wanted 24, got 0
db-1             | 2025-05-03 10:16:34.198 UTC [27] LOG:
    redo done at 0/160C520
db-1             | 2025-05-03 10:16:34.212 UTC [1] LOG:
    database system is ready to accept connections
sentiment-api-1  |  * Serving Flask app 'app'
sentiment-api-1  |  * Debug mode: off
sentiment-api-1  | WARNING: This is a development server. Do
    not use it in a production deployment. Use a production
    WSGI server instead.
sentiment-api-1  |  * Running on all addresses (0.0.0.0)
sentiment-api-1  |  * Running on http://127.0.0.1:5000
sentiment-api-1  |  * Running on http://172.18.0.3:5000
sentiment-api-1  | Press CTRL+C to quit
sentiment-api-1  | 127.0.0.1 - - [03/May/2025 10:17:09] "HEAD
     /health HTTP/1.1" 200 -
web-client-1     |  * Serving Flask app 'app'
web-client-1     |  * Debug mode: on
web-client-1     | WARNING: This is a development server. Do
    not use it in a production deployment. Use a production
    WSGI server instead.
web-client-1     |  * Running on all addresses (0.0.0.0)
web-client-1     |  * Running on http://127.0.0.1:5000
web-client-1     |  * Running on http://172.18.0.4:5000
web-client-1     | Press CTRL+C to quit
web-client-1     |  * Restarting with stat
web-client-1     |  * Debugger is active!
web-client-1     |  * Debugger PIN: 118-356-955
sentiment-api-1  | 127.0.0.1 - - [03/May/2025 10:17:39] "HEAD
     /health HTTP/1.1" 200 -
```

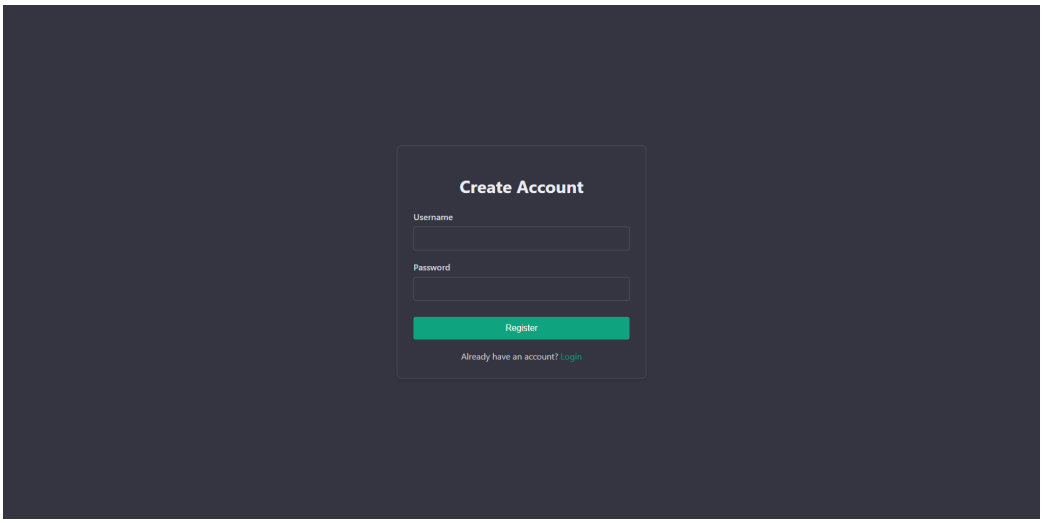
6.3. Web-es szolgáltatások

A rendszer rendelkezik felhasználó kezeléssel, regisztrációs és bejelentkező felülettel, a felhasználók chat előzményeinek tárolásával.

Néhány kép működés közben:

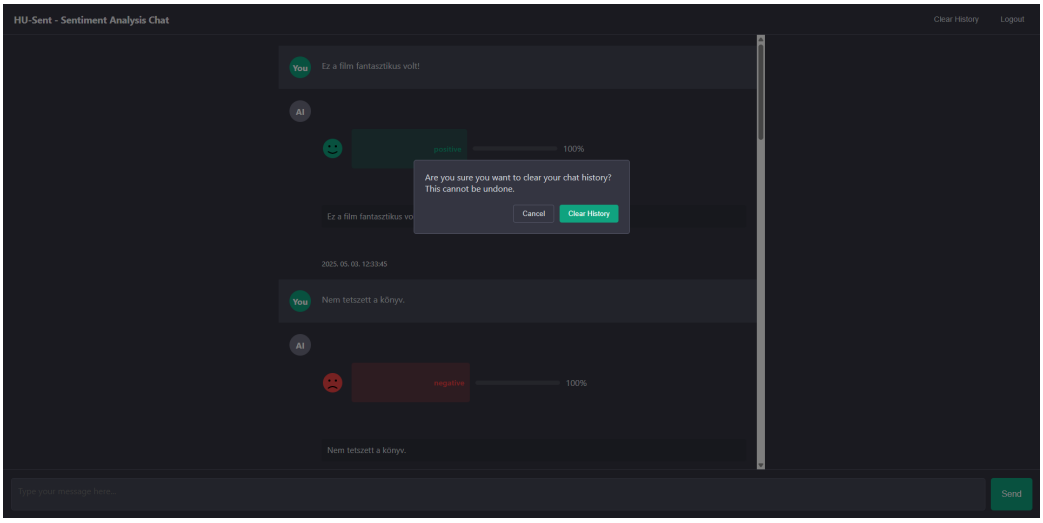


1. ábra. Bejelentkezési felület

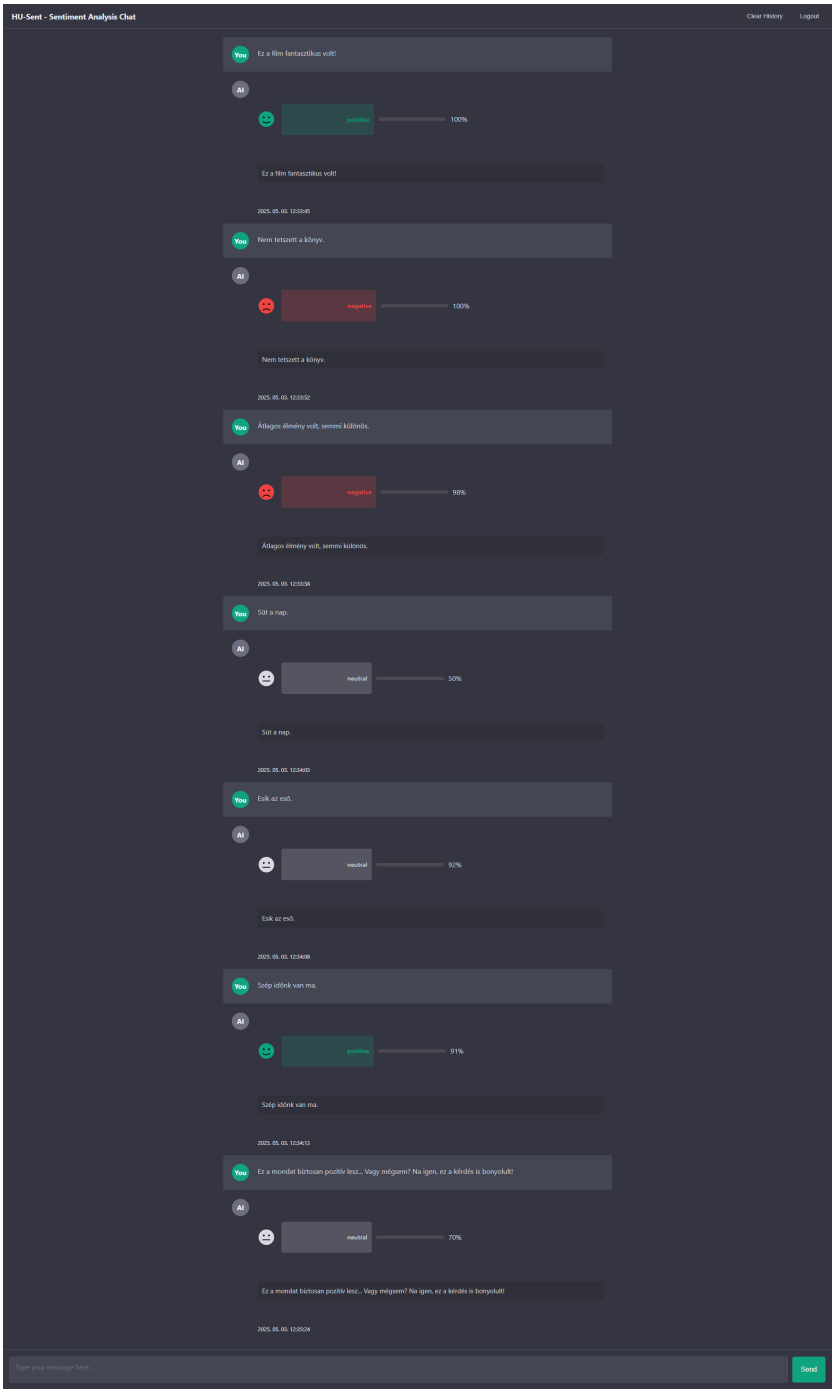


2. ábra. Regisztrációs felület

3. ábra. Üres chat felület



4. ábra. Előzmények törlése



5. ábra. Chat felület 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 chat üzenetei

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	Modell bizonyossága
`created_at`	DateTime	Üzenet időpontja

6.5. Adatbázis kapcsolatok

- Egy felhasználó (`User`) több chat üzenettel (`Chat`) rendelkezhet
- Minden chat üzenet pontosan egy felhasználóhoz tartozik

6.6. Docker és Docker Compose

Ahogy az korábban említve lett, a webes szentiment analízis alkalmazás *Docker* konténerizációval készült, a könnyű telepítés, hordozhatóság, 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 sentiment 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 van egybe szervezve, és a beállításaik tárolva. 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 -
   d sentiment_db"]
16         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 Dcokerfile

```
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. Források

A dokumentumot az alább felsorolt források segítségével készítettem el.

Hivatkozások

[1] SZTAKI-HLT. (2022). *hubert-base-cc*. Hugging Face.
`https://huggingface.co/SZTAKI-HLT/hubert-base-cc`

[2] NYTK. (2022). *HuSST Dataset*. Hugging Face.
`https://huggingface.co/datasets/NYTK/HuSST`

[3] SZTAKI-HLT. (2022). *huBERT - Hungarian BERT Model*. BME-HLT.
`https://hlt.bme.hu/hu/resources/hubert`

[4] Orosz György. (2023). *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/`