Magyar nyelvű Szentiment Analízis Projekt

Név

2025. május 2.

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.

```
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 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

6. 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