E-mail spam szűrő alkalmazás

Haladó programozás / Mesterséges intelligencia beadandó

Ács Dávid – QOUJAO

Kisjuhász Vilmos – QUCS64

Az alkalmazás GitHub Link-je

https://github.com/kvilmos/Spam_Filter

1. Bevezetés

A projekt célja egy egyszerű, de hatékony MI spam-szűrő alkalmazás készítése, amely képes az e-maileket "spam" vagy "ham" (nem spam) kategóriákba sorolni. A spam-szűrés ma már elengedhetetlen az e-mail szolgáltatásokban, mivel egyre több kéretlen üzenet árasztja el a felhasználói fiókokat.

Az alkalmazás egy Naive Bayes algoritmuson alapuló gépi tanulási modellt használ, amely képes az e-mailek szövegének elemzésére és kategorizálására. A projektben a **Scikit-learn** könyvtár segítségével épül fel a modell, a felhasználói felületet pedig a **Streamlit** könyvtár valósítja meg. A beszámolóban részletezésre kerül az alkalmazás elkészítésének lépései, a modellezés folyamata, valamint azokat a nehézségek, amelyekkel a fejlesztés során szembesültünk. A modell magyar és angol e-mailek szűrésére is elkészültek.

A fejlesztői környezet, amit használtunk: **Visual Studio Code**

A program futtatásához szükség lesz az alábbi csomagokra:

python -m pip install **pandas**

python -m pip install **openpyxl**

python -m pip install **scikit-learn**

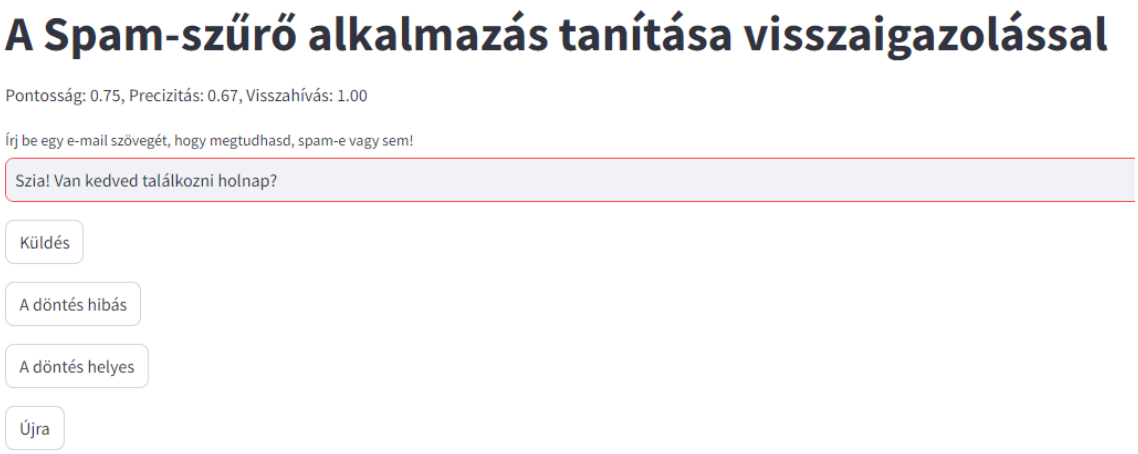
python -m pip install **streamlit**

python -m pip install **nltk**

python -m pip install **matplotlib**

python -m pip install **seaborn**

Az indító parancs: streamlit run [Elérési út]\ spam_szurov2_hun.py



A program indításakor a böngészőben egy weboldal nyílik meg ahol meg tudunk adni egy üzenetet.

Majd a program eldönti spam e vagy sem.

2. Az alkalmazás megvalósítása

2.1 Adatok előkészítése

Az e-mail szövegek gépi tanulási modellbe történő betáplálása előtt az adatok előfeldolgozására volt szükség. A projekt során egy létező adatbázist használtam, amely spam és ham e-mailek szövegét tartalmazta, és egy megkülönböztető adatot hogy spam e vagy sem, ezt külön oszlopokban. A Scikit-learn **TfidfVectorizer** osztályát alkalmaztam az e-mailek szövegének felbontására. Ez a lépés kulcsfontosságú, mivel a gépi tanulási algoritmusok csak numerikus adatokat képesek feldolgozni.
Az adatok előkészítésének része volt a következő lépések elvégzése:
• A szövegek tisztítása, például a felesleges karakterek eltávolítása.
• A szövegek átalakítása kisbetűssé, valamint a stop szavak (gyakori, de jelentéktelen szavak, mint pl. „és”, „a/az”) eltávolítása.
• Az adatok "TF-IDF" (**Term Frequency-Inverse Document Frequency**) formátumban történő átalakítása, amely figyelembe veszi a szavak fontosságát és gyakoriságát az e-mailekben. Tehát a modellünk úgy fog üzemelni, hogy a szavak **gyakoriságát** megnézi egy spam és ham e-mailben és ez alapján dönti el, hogy az adott e-mail spam-e.

Az „adatbázisra” hivatkozva később az excel táblázatot értjük. Egyszerűbb megoldás a jelenlegi feladathoz, mint egy tényleges adatbázis. Úgy döntöttünk a célnak megfelel, nagyobb hangsúlyt fektetünk a modell működésének megismerésére és fejlesztésére. Persze némi változtatással ugyanígy működne adatbázis rendszerrel is.

```
# Stop szavak letöltése
nltk.download('stopwords')

# Adatok beolvasása
data = pd.read_excel(r'C:\Users\student\Desktop\ver1\emails.xlsx')
data['Label'] = data['Label'].map({'spam': 1, 'ham': 0})

def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\W', ' ', text)
    text = re.sub(r'\s+', ' ', text)
    text = text.strip()
    return text

data['Text'] = data['Text'].apply(preprocess_text)

X = data['Text']
y = data['Label']
vectorizer = TfidfVectorizer(stop_words=stopwords.words('english'))
X = vectorizer.fit_transform(X)
```

2.2 Gépi tanulási modell építése

A **Naive Bayes** algoritmust választottuk a spam-szűrő modell alapjául, mivel a Naive Bayes modellek jól teljesítenek a szövegalapú osztályozási feladatokban. Az algoritmus gyorsan tanul és nem igényel nagy mennyiségű számítási kapacitást, így ideális választás egy egyszerűbb spam-szűrő számára. A modellt az előre megadott adatainkon tanítottuk be, majd teszteltük. Tehát az Excelben előre volt megadva pár e-mail és ez persze a használat és a tanítás során idővel bővül.

```
X = vectorizer.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = MultinomialNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

A modell teljesítményének értékelésére a pontosság, a precizitás és a visszahívás mutatóit használtuk. Ezek az értékek segítettek meghatározni, hogy a modell mennyire képes jól felismerni a spam üzeneteket és milyen mértékben kerülte el a hamis megoldást. Ez nagyon jól megfigyelhető miközben használjuk az alkalmazást mivel a **Streamlit** felületen feltüntettük ezeket az adatokat, és miközben használjuk változnak.

- **Pontosság (Accuracy)**

A pontosság megmutatja, hogy a modell összességében milyen arányban adott helyes választ.

Képlet:

$$\text{Pontosság} = \frac{\text{Helyesen osztályozottak száma}}{\text{Összes minták száma}}$$

Ha például az accuracy értéke 0.9, akkor a modell az összes eset 90%-ában helyesen döntött.

- **Precizitás(Precision)**

A precizitás azt mutatja meg, hogy az összes pozitívnak osztályozott minta közül hány volt ténylegesen pozitív. Tehát milyen mértékben nem téved a modell.

Képlet:

$$\text{Precizitás} = \frac{\text{Igaz pozitív (TP)}}{\text{Igaz pozitív (TP)} + \text{Hamis pozitív (FP)}}$$

Például egy 0.8-as precizitás azt jelenti, hogy az összes pozitív osztályozás 80%-a valóban helyes volt.

- **Visszahívás(Recall)**

A visszahívás megmutatja, hogy a modell a tényleges pozitívok (például valódi spamek) hány százalékát találta meg.

Képlet:

$$\text{Visszahívás} = \frac{\text{Igaz pozitív (TP)}}{\text{Igaz pozitív (TP)} + \text{Hamis negatív (FN)}}$$

Például egy 0.85-ös visszahívás azt jelenti, hogy a tényleges pozitívok (például valódi spamek) 85%-át sikerült felismerni.

2.3 Naive Bayes grafikon

Ezt a grafikon csak érdekességképpen került az alkalmazásba, hogy vizualizálhassuk pontosan milyen szavak is a leggyakoribbak a spam és ham üzenetekben. Láthatjuk tehát egy szó „spam/ham" arányát és gyakoriságát. A grafikon nem veszi figyelembe a stop szavakat csak azokat a szavakat, amik ténylegesen rangsorolva voltak.

3. Az alkalmazás felhasználói felülete

A felhasználói felületet a **Streamlit** segítségével készült el, amely egy Python alapú webalkalmazás-fejlesztő rendszer. A Streamlit a gépi tanulási modellekkel lehetővé teszi a projekt gyors megvalósítását és az interaktív felület kialakítását. Az alkalmazás biztosít egy felületet, ahol beírhatunk egy e-mailt vagy egy szöveget és a modell eldönti hová tartozik. Tanítani is tudjuk a modellt azzal, hogy megmondjuk neki a válasza helyes-e. Ezzel bekerül az adatbázisba a bevitt szöveg és a modell egyre nagyobb adatbázisból tud tanulni. Extraként egy kis grafikont is kapunk, ami az adatbázisban lévő spam és ham szavak eloszlását mutatja.

4. Fejlesztési nehézségek

4.1 Adatok előfeldolgozása

Maga a szöveg tisztítása nem okozott túl sok problémát, egyszerűen a tanult módon kicseréltük a felesleges karaktereket és szóközöket. Viszont olyan hibába ütköztünk, hogy túl sok fölösleges/semleges szó, amit tartalmaz a rendszer ezért utána néztünk hogyan szűrhetnénk ezt ki. Az nltk „**stopwords**" könyvtárának importálásával lehetséges, ami kiválogatja a semleges szavakat, mint például a „the" vagy „and". Ez fontos mivel ezek eléggé semleges szavak és egy spam és ham-ben is előfordulhatnak egyaránt. Így jobban figyel a modell az „egyedi" szavakra, mint a FREE, CONGRATULATIONS, ami már inkább specifikusan egy spamre utal. Általában ezek nem megtalálhatóak egy ham emailben. Úgyszint pl. „Hi" „Wanna", „meet", ezek inkább ham e-mailekben előforduló szavak. Az újabb magyar változatban természetesen a magyar listát telepítettem és az alapján végzi a szűrést. Pl: Ő, Ők, és.

4.2 A felhasználói visszajelzések feldolgozása

A felhasználói visszajelzési rendszer integrálása kisebb újragondolást jelentett, mivel biztosítani kellett, hogy a rendszer felismerje a felhasználói helyesbítéseket, és ezek alapján módosítsa az eredeti modellt. A visszajelzések alapján a program eltárolja a felhasználó által

megadott új címkét, és lehetőség van a modell újra tanítására, ha elegendő visszajelzés gyűlik össze. Először ez a funkció nem volt benne csak szimplán felismerte a modell az e-mailt és elrakta magának. Így sokkal jobban hasznosíthatóvá vált, mivel tudunk neki visszajelzést adni, ha hibázna, és ezáltal tanul.

4.3 Streamlit integrálása

A Streamlit alapú felhasználói felület Colabon történő futtatása külön kihívás volt, mivel Colab nem támogatja közvetlenül a Streamlit alkalmazások futtatását, végül arra a döntésre jutottunk, hogy a fejlesztői környezetem VS Code lesz. Hosszú távon érdemes a projektet helyi gépen futtatni, például Visual Studio Code-ban, mivel az stabilabb és közvetlen hozzáférést biztosít a Streamlit alkalmazáshoz. Persze így szükség volt a kiegészítő csomagok letöltésére lokálisan, hogy minden rendben működjön. (Ezek megtalálhatóak a bevezetésnél)

A streamlit okozott még egy hosszú fejtörést mivel nem tudtuk elképzelni miért nem működnek a gombok rendesen úgy ahogy szeretnénk.

Egy korábbi kód így nézett ki a gombok leprogramozásában: (Ez a kód **hibás!**)

```
if st.button("Küldés") and user_input:
    processed_input = preprocess_text(user_input)  # A felhasználó által megadott szöveg előfeldolgozása
    input_vector = vectorizer.transform([processed_input])  # TF-IDF mátrixba alakítás
    predicted_label = "spam" if model.predict(input_vector)[0] == 1 else "ham"  # Predikció

    # Kiírjuk a döntést a felhasználónak
    st.write(f"A modell döntése: {predicted_label}")

    # Piros gomb a visszajelzéshez
    if st.button("A döntés hibás", key='feedback'):
        st.write("Köszönjük a visszajelzést! A válasz módosítva.")
        # Helyes címke beállítása
        correct_label = 'ham' if predicted_label == 'spam' else 'spam' # Itt javítani kell valamits
    else:
        # Ha a gombot nem nyomják meg, a címke a modell döntése marad
        correct_label = predicted_label

    new_feedback = pd.DataFrame([[user_input, correct_label]], columns=["Text", "Label"])

    # Rögzítjük az új visszajelzést az Excel fájlban
    try:
        existing_data = pd.read_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx')
        updated_data = pd.concat([existing_data, new_feedback], ignore_index=True)
    except FileNotFoundError:
        updated_data = new_feedback

    updated_data.to_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx', index=False)
```

Az elképzelés az volt, hogy ha a felhasználó megnyomja a „döntés hibás” gombot akkor a program lecseréli a megoldást (spam vagy ham) a döntés ellenkezőjére. Tehát kijavítja az adatot és úgy menti el. Ez természetesen ilyen formában nem működött, és egy pár órás próbálgatás és kódcserélgetés, arra jutottunk, hogy érdemes lenne állapotváltozókat be iktatni a rendes működéshez.

A kód az *st.session_state* segítségével tárolja az állapotváltozókat, így a felhasználói interakciók során egyes információk megőrződnek.

Az "Újra" gomb megnyomásakor a st.session_state.clear() törli az összes korábban használt session állapotot (pl. predicted_label). Ezzel a felhasználó tiszta kezdőlappal folytathatja a következő e-mail tesztelését.

"A döntés hibás" és "A döntés helyes" gombok lehetővé teszik a felhasználó számára, hogy visszajelzést adjon a modell döntéséről. Ezek kerülnek az adatbázisba.

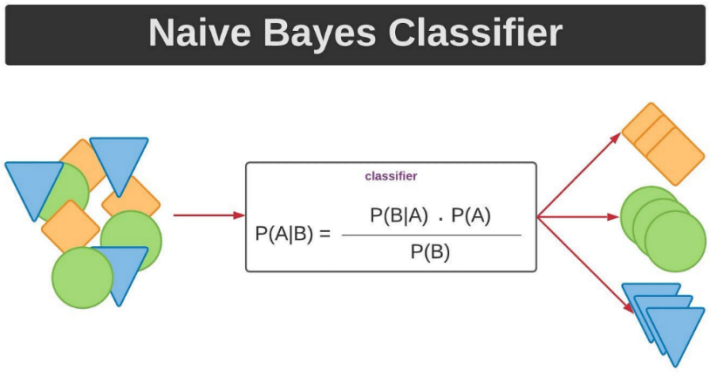
Az alábbi képen látható a már **működő** kód:

```
if st.button("Küldés") and user_input:
    processed_input = preprocess_text(user_input)
    input_vector = vectorizer.transform([processed_input])
    st.session_state.predicted_label = "spam" if model.predict(input_vector)[0] == 1 else "ham"
    st.write(f"A modell döntése: {st.session_state.predicted_label}")

if st.session_state.predicted_label is not None:
    if st.button("A döntés hibás"):
        correct_label = 'ham' if st.session_state.predicted_label == 'spam' else 'spam'
        st.write("Köszönjük a visszajelzést! A válasz módosítva.")
        new_feedback = pd.DataFrame([[user_input, correct_label]], columns=["Text", "Label"])
        try:
            existing_data = pd.read_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx')
            updated_data = pd.concat([existing_data, new_feedback], ignore_index=True)
        except FileNotFoundError:
            updated_data = new_feedback
        updated_data.to_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx', index=False)

    if st.button("A döntés helyes"):
        st.write("Köszönjük a visszajelzést!")
        new_feedback = pd.DataFrame([[user_input, st.session_state.predicted_label]], columns=["Text", "Labe
        try:
            existing_data = pd.read_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx')
            updated_data = pd.concat([existing_data, new_feedback], ignore_index=True)
        except FileNotFoundError:
            updated_data = new_feedback
        updated_data.to_excel(r'D:\Suli\5.félév\Haladoprog\Beadando\ver1\emails.xlsx', index=False)
        # Újra gomb
    if st.button("Újra"):
        st.session_state.clear()  # Az összes session állapot törlése
        st.write("Új e-mailt adhatsz meg.")
```

5. A gépi tanulás alapelvei és a modell tanítása

5.1 Naive Bayes algoritmus

A Naive Bayes egy osztályozó gépi tanulási algoritmus, amely a Bayes-tételen alapul. Nagyon hatékony, különösen akkor, ha olyan szöveges adatokkal foglalkozik, mint például: érzelmek elemzése, spamészlelés és szövegosztályozás.



Ezt az algoritmust „naivnak” nevezik, mert feltételezik, hogy az összes adatkészlet-változó független, ami nem mindig van így

5.2 A modell teljesítménye

A modell pontossága, precizitása és visszahívása egyaránt fontos mutatók voltak a teljesítmény értékelésében. A pontosság segít megérteni, hogy a modell milyen arányban hoz helyes döntéseket, míg a precizitás és a visszahívás a spam és ham osztályok közötti különbségek felismerésében játszik szerepet. Az első tesztek során (már csak 5 e-mail alapján) a modell 80-85%-os pontosságot ért el, de az eredmény javítható a visszajelzések beépítésével. Persze ez a pontosság annál jobban tért el minél szokatlanabb adatokat adtunk neki. Miután ezeket is megtanulta egyre nagyobb hatékonysággal működött.

6. Összefoglalás

Az alkalmazás egy Naive Bayes alapú gépi tanulási modellt használ a spam e-mailek detektálására. A szövegeket először előfeldolgozza, például normalizálja és tisztítja, majd a szavakat a TF-IDF vektorizáló transzformálja, hogy a szavak fontosságát figyelembe vegye. A feldolgozott adatokat egy Naive Bayes modell tanítja, amely képes a "spam" és "ham" kategóriák közötti különbség azonosítására. A Streamlit felhasználói felületen a felhasználó e-mail szöveget adhat meg, amire a modell jósol, és visszajelzést is kér a döntés pontosításához, melyet később felhasznál a modell továbbtanítására.