Haladó programozás beadandó

Visszakeresés alapú magyar nyelvű chatbot

Dokumentáció

A képen rajzfilm, clipart, illusztráció, tervezés látható

Automatikusan generált leírás

# Tartalomjegyzék

Tartalom

[Tartalomjegyzék 2](#_Toc181789931)

[Bevezetés 2](#_Toc181789932)

[Könyvtárak importálása 3](#_Toc181789933)

[Stopword lista 4](#_Toc181789934)

[Szöveg tokenizálása 5](#_Toc181789935)

[Intents file létrehozása és beillesztése 5](#_Toc181789936)

[Szöveg vektorizálása 7](#_Toc181789937)

[Model elkészítése 8](#_Toc181789938)

[Felhasználói bevitel jósolása 8](#_Toc181789939)

[Válaszok lekérése 9](#_Toc181789940)

[Tesztelés 10](#_Toc181789941)

[Plusz 10](#_Toc181789942)

[Felhasznált irodalom 11](#_Toc181789943)

# Bevezetés

Ebben a projektben egy magyar nyelvű visszakeresés alapú chatbotot készítettem el egyszerűbb Machine Learning módszerekkel. Az ezt követő pontokban nagy vonalakban leírtam, hogyan működik és mennyire hatékony lett, valamint miképpen lehetne továbbfejleszteni jövőben. Azért választottam ezt a saját témát, mert érdekel, hogy állnak össze egyszerűbb és bonyolultabb chatbotok, valamint érdekesnek és kellemes kihívásnak tartottam a témát.

Vésszakereséses alapú chatbot: egy előre generált „intents” (szándékok) .JSON fájlt használ fel az adatokhoz, amiből tanul és edzi a modellt, majd a stopwordok (leggyakrabban használt szavak) és az adatok segítségével értelmezni tudja és megjósolja, hogy egy mondat vagy szó milyen osztályba tartozik és az alapján választ ki a válaszok közül. A hatékonysága attól függ, hogy mennyi stopword-ünk van, az intents file mennyire jól lett kialakítva, vagyis mennyire bővelkedik szándékokkal (mintákkal és válaszokkal), valamint mennyire jól van beállítva a modellünk.

Az én modellem hatékonysága nagyjából 50-60% lett, mert saját .JSON fájlom és a stopword listám limitált.

A feladat elkészítéséhez a *Google Colab*-ot használtam az intents.json file elkészítéséhez pedig *Visual Studio Code*-ot.

# Könyvtárak importálása

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Itt importáltam be a megvalósításhoz szükséges könyvtárakat.

A SpaCy egy nyilt forráskodú természetes nyelv feldolgozó könyvtár, másnéven NLP, amelyet nagyszabású chatprogramok készítésére használnak. A string-et azért importáltam, mert későbbiekben szükséges lesz stringekkel dolgozni. A string-hez hasonló okokért a json-t is importáltam. A random-ot a szándékok címkéinek (pl. köszönés, elköszönés) megjósolása utáni véletlenszerű válasz megadásához szükséges. A többi importálás mind a scikit-learn egyes részihez tartozik, amelyek az adatok modellezésének elkészítéséhez szükségesek. A fentebbi ábárn röviden leolvashatóak, hogy melyiket mire használjuk.

# Stopword lista

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Következőnek létrehoztam a stopwords listát, amely olyan gyakori szavakat jelent, amelyek a természetes nyelvi feldolgozási feladatok során önmagukban nem hordoznak jelentős jelentést, mint pl. kötőszavak vagy elöljárószavak. Ezeket a chatbot az osztályozásnál nem veszi figyelemben, ezzel pedig javítja a modell pontosságát. Célja a modell bonyolultságának csökkentés ezáltal, hogy az értelmesebb szavakra, például főnevekre és igékre összpontosít, amelyek segítenek azonosítani a témát vagy a szándékot.

Felül látható, hogy nyelvspecifikus modellek nélküli toknizátort használok a következőkben a spacy segítségével, mert a magyar tokenizátort több próbálkozás után sem sikerült letöltenem.

# A képen szöveg, szoftver, Multimédiás szoftver, képernyőkép látható Automatikusan generált leírásSzöveg tokenizálása

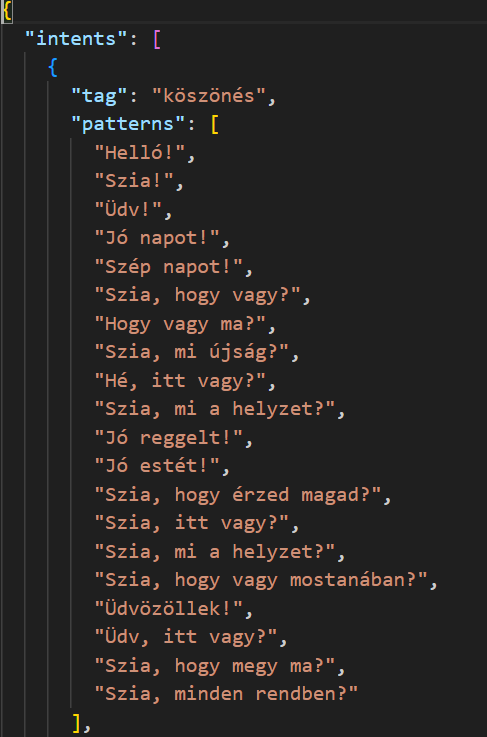
Itt történik az előbb is említett tokenizálás (Tokenization), amely során egy függvény segítségével szavakká alakítjuk a szöveget, hogy a nyers szöveget a gép számára érthető formátumba alakítsuk át. A bejövő szöveget kisbetűssé alakítjuk és a tokenizálóba hívjuk meg, aztán pedig kiszűrjük belőle a stopszavakat és az írásjeleket egy list comprehension-nel és visszaadjuk ezt a listát. A függvény alatt egy példát is láthatunk. Ez az előfeldolgozás szükséges lesz később a szöveg vektorizálásához, a jósoláshoz és a teszteléshez.

Tokenizálás: tokenizálás az a folyamat, amikor egy mondatot vagy szöveget egyedi egységekre (tokenekre: szavak, karakterek) osztanak fel.

# Intents file létrehozása és beillesztése

Ezzel a két sorral nyitom meg az „intents.json” fájlt. Ezt előtte minden alkalommal fel kell tölteni a jegyzetfüzetbe a google colab-on keresztül.

Így néz ki pár sora:



A képen szöveg, képernyőkép, Betűtípus látható

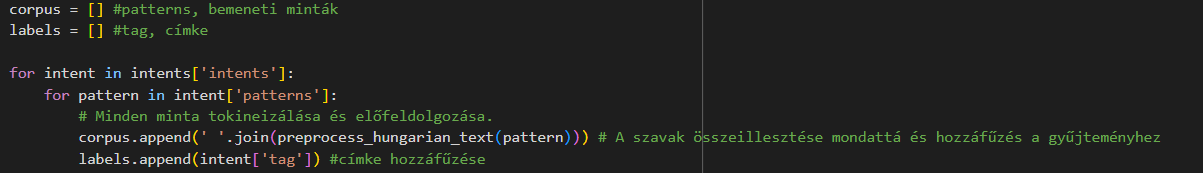
Automatikusan generált leírás

Ezen a köszönés „tag” (címke) látható 20 db „pattern”-nel, amelyek a különböző típusú felhasználói bevitelek, és ezekhez 6 db „responses”-t ( válaszokat) írtam.

# Szöveg vektorizálása

A szöveget olyan formátumba kell alakítanunk, amelyet a gépi tanulási modellek megértenek (numerikus vektorokká kell alakítani).

Itt a BoW (Bag of Words) megközelítést használtam. Ez egy szójegyzéket készít az adatkészletben talákható összes egyedi szóról, majd az egyes mondatokat úgy reprezentálja, hogy megszámolja, hogy az egyes szavak milyen gyakran fordulnak elő az adott mondatban.

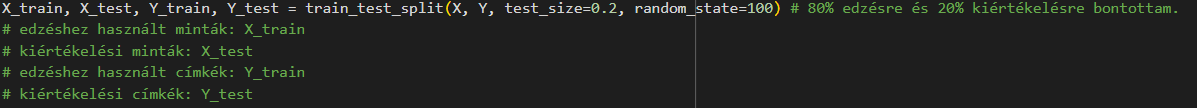
„Korpusz”-t (gyűjteményt) kell létrehozni, ez fogja elraktározni az intents file összes bemeneti mintáját (patterns). A labels-be mentjüke el a címkéket.

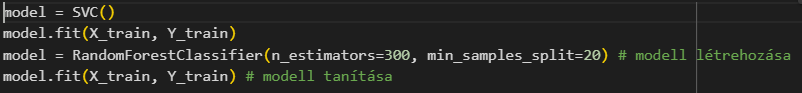
A képen szöveg, képernyőkép látható

Automatikusan generált leírásVektorizáljuk a korpuszt a CountVectorizer-rel, tehát a korpuszból egy Bag of Words márixot hozunk létre. Ezzel hozzuk létre a szókincsünket, valamint a mintánkat numerikusan (nullásokkal és egyesekkel) is ábrázolni tudjuk, amiből láthetó, hogy az adathalmazunk mennyire effektív. Ha sok az 1-es akkor jók a bemeneti minták.

Végül a felhasználói bemenetet is vektorizáljuk, de ezt majd a *Felhasználói bevitel jóslása* közben belül fog megtörténni.

# Model elkészítése

 Először fel kell osztania az adatainkat véletlenszerű „train” (képzési,tanító) sorozatokra és „test” (teszt) részhalmazokra a scikit-learn beépített train\_test\_split függvény segítségével, hogy a modell ne ugyanazon az adatokon kerüljön értékelésre, amelyekre betanították, hogy elkerülje a túlillesztést. A modell segítégével az adatokat osztályozni tudjuk, majd kiértékelni az osztályokat. A tanítókészlet lesz a modellünk később, míg a tesztelési a tesztelésünk kiértékelésére használjuk.

Aztán be kell tanítani a betanítási modellt RandomForestClassifier segítségével:

Végül értékeljük a nem látott adatokon, azaz a tesztkészleten a modellt:

A képen szöveg, Betűtípus, képernyőkép, sor látható

Automatikusan generált leírás

# Felhasználói bevitel jósolása

Ebben a részben egy függvény segítségével prediktáljuk meg a felhasználó bevitelének szándékát azzal, hogy vektorizálja a felhasználó bemenetét, így a modell is dolgozni tud vele, hiszen így már numerikus formában jelenik meg, ezután megjósolja a címkét és visszaalakítja az A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leíráseredeti alakra (szöveges: pl, vicc).

# Válaszok lekérése

Itt egy függvény segítségével véletlenszerű választ kérünk le - a random.choice segíségével - az intents.json fájlból az előbb kiszámított szándék alapján.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Ha talál egyezést az intents.json fájlban, akkor a „Bocsánat, de nem értettem” választ adja vissza, ha pedig sikerül akkor a modell minőségéhez mérten kapunk választ.

# Tesztelés

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás Egy előltesztelős ciklusban (while loop) teszteljük a chatbot-ot, ahol a folyamatosan felszólítjuk a felhasználót a bevitelre. A bevitel után megkezdődik a szándék megjósolása, majd a válasz kiíratása. Ehhez még szükség volt egy intent\_map nevezetű szótár (dictionary comprehension pontosan), amely keresőtáblaként működik. A numerikus címkékből az eredeti címkékké való visszaalakításnak az végrehajtásához szükséges. Addig fut a ciklus, ameddig meg a kilépés szót meg nem adja a felhasználó.

# Plusz

Itt egy bonyolúltabb folyamat ment végbe, amit hiperparaméter-hangolási folyamatnak nevezünk. A *GridSearchCV* használatának segítségével olyan beállítási paramétereket (hipermaraméterek) tudunk megekeresni, amelyek egy gépi tanulási modell tanulási folyamatát vezérlik, és nem magukból az adatokból tanulják meg őket.

n\_estimators: A fák száma a véletlenszerű erdőben.

max\_depth: Az egyes fák maximális mélysége.

min\_samples\_split: A belső csomópont felosztásához szükséges minimális minták száma.

min\_samples\_leaf: A minták minimális száma ahhoz, hogy egy levélcsomóponton legyen.

GridSearchCV inicializálásához ezeket a paramétereket használtam:

estimator: A hangolandó modell (jelen esetben a modell, ami egy RandomForestClassifier).

param\_grid: A keresendő hiperparaméterek rácsja.

cv: A keresztellenőrzési hajtások száma (jelen esetben 5).

n\_jobs: A párhuzamos feldolgozáshoz használandó CPU magok száma (-1 azt jelenti, hogy az összes elérhető magot használja).

verbose: A kimenet bőbeszédűségét vezérli (a 2 a részletes kimenetet jelenti).A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

Amit tudni kell róla, hogy nagyon hosszú a futásideje, nagyjából 5-6 óra a Google Colab-ban.

# Felhasznált irodalom

<https://www.youtube.com/watch?v=t933Gh5fNrc>

<https://spacy.io/models/en>

<https://github.com/huspacy/huspacy>

<https://community.intersystems.com/post/step-step-guide-create-customized-chatbot-using-spacy-python-nlp-library>

<https://thepitch.hu/magyar-stop-words-lista/>

<https://www.tutorial.hu/egyeb/hungarian-stop-words/>

<https://github.com/stopwords-iso/stopwords-hu>

<https://scikit-learn.org/stable/>