Miskolci Egyetem Alkalmazott Matematikai

Gépészmérnöki és Informatikai Kar Intézeti Tanszék

Általános Informatikai Intézeti Tanszék

**Természetes nyelvű szövegek témakör szerinti elemzése gépi tanulás segítségével**

**Szakdolgozat**

**Készítette:**

**Név:** Farkas Ádám

**Neptunkód:** FE019W

**Szak:** Mérnökinformatikus BSc

Korszerű web technológiák szakirány

|  |  |  |  |
| --- | --- | --- | --- |
| **Miskolci Egyetem**  **Gépészmérnöki és Informatikai Kar** |  | | **Általános Informatikai Intézeti Tanszék**  3515 Miskolc-Egyetemváros |
| Szak**: Mérnökinformatikus** | |  | Szakdolgozat azonosító: **IAL/FE019W/BSc/2023** |
| Szakirány: korszerű web-technológiák | |  | **Intézmény azonosító: FI 87515** |
|  | |  |  |

**SZAKDOLGOZAT FELADAT**

**Farkas Ádám**

BSc mérnökinformatikus jelölt részére

|  |  |
| --- | --- |
| A tervezés tárgyköre: | **természetes nyelvfeldolgozás** |
| A feladat címe: | **Természetes nyelvű szövegek témakör szerinti elemzése gépi tanulás segítségével** |
| **A feladat részletezése:**  Az aktuálisan elérhető idegen nyelvű szövegek lehetővé teszik, hogy azokon gépi tanulási módszerek felhasználásával a tartalomra vonatkozó elemzéseket hajtsunk végre. A dolgozat célja egy olyan idegennyelv tanulását segítő eszköz fejlesztésének előkészítése, amely egy szövegkorpusz automatizált feldolgozásával statisztikai kimutatásokat, a szavak témakör szerint csoportosítását, illetve egy tartalmi összefoglalót is megfogalmaz belőle. A dolgozat ezen, alapvetően klaszterezési probléma megoldásával foglalkozik. Bemutatja a természetes nyelvi elemzés matematikai hátterét, hasonló alkalmazási területeken a készen elérhető megoldásokat. Az elemzések Python programozási nyelv, és a hozzá elérhető adatelemző függvénykönyvtárak segítségével készülnek. | |

|  |  |
| --- | --- |
| Tervezésvezető:  Piller Imre | Tanszék, beosztás:  Alkalmazott Matematikai Intézeti Tanszék, egyetemi tanársegéd |
| Konzulens(ek): | Cég, beosztás: |
| A szakdolgozat kiadásának időpontja: | 2023.09.15. |
| A szakdolgozat beadásának határideje: | 2023.11.17. |
| Miskolc, 2023.12.05. | **Prof. Dr. Kovács László**  tanszékvezető egyetemi tanár |

1. A szakmai gyakorlat helye: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. A szakmai gyakorlat vezetőjének neve: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. A szakdolgozat módosítása: szükséges (a módosítást külön lap tartalmazza)   
 nem szükséges (a megfelelő rész aláhúzandó)

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tervezésvezető aláírása

4. A tervezést ellenőriztem: (1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(3) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(4) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dátum, tervezésvezető aláírása

5. A szakdolgozat beadható

nem adható be

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

konzulens aláírása tervezésvezető aláírása

6. A szakdolgozat ….. szövegoldalt,

….. db rajzot,

….. db CD mellékletet

….. egyéb mellékletet tartalmaz.

7. A szakdolgozat bírálatra: bocsátható

nem bocsátható

A bíráló neve, címe: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

tanszékvezető aláírása

8. Osztályzat: a bíráló javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a tanszék javaslata: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság döntése: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Miskolc, \_\_\_\_\_\_\_\_\_\_\_\_\_

a Záróvizsga Bizottság elnökének aláírása

# **Köszönetnyilvánítás**

Szeretnék köszönetet mondani Piller Imrének, a tervezésvezetőmnek, aki rengeteg hasznos tanáccsal, és figyelmes, segítőkész hozzáállással egyengette a dolgozatom útját. Szeretném megköszönni barátnőmnek a végtelen türelmet és támogatást, amelyet kaptam tőle a dolgozat megírása közben. Szeretnék még köszönetet mondani a családomnak, akik nélkül sosem jutottam volna el oda, ahol ma tartok.

**EREDETISÉGI NYILATKOZAT**

Alulírott ……………………………………………………….; Neptun-kód:…………………

a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős ……………. szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy

………………………………………………………………………………………………

című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom

felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

* szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
* tartalmi idézet hivatkozás megjelölése nélkül;
* más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy

plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc,.............év ………………..hó ………..nap

…….……………………………….…

Hallgató

Tartalom

[**Prof. Dr. Kovács László** 2](#_Toc164375209)

[**Köszönetnyilvánítás** 4](#_Toc164375210)

[1 Bevezető 7](#_Toc164375211)

[2 Az NLP problémakör áttekintése 9](#_Toc164375212)

[2.1 Az NLP 9](#_Toc164375213)

[2.2 Alapvető feladatok az NLP-ben 11](#_Toc164375214)

[2.3 A deep learning és az NLP 12](#_Toc164375215)

[2.4 Kihívások és jövőbeli irányok 14](#_Toc164375216)

[2.5 NLP a gyakorlatban 15](#_Toc164375217)

[2.6 Összehasonlító nyelvelemzés 17](#_Toc164375218)

[3 Python fejlesztőkörnyezet összeállítása 18](#_Toc164375219)

[3.1 Lehetséges programozási nyelvek 18](#_Toc164375220)

[3.2 Fontosabb Python fejlesztőkörnyezetek 22](#_Toc164375221)

[3.3 Függvénykönyvtárak 24](#_Toc164375222)

[4 A Spyder konfigurációja 29](#_Toc164375223)

[4.1 A Spyder 29](#_Toc164375224)

[4.2 Technikai részletek 34](#_Toc164375225)

[5 Szövegkorpuszok összegyűjtése és statisztikai elemzése 35](#_Toc164375226)

[5.1 A szövegek kiválasztása 35](#_Toc164375227)

[5.2 A statisztikai vizsgálatok célja 35](#_Toc164375228)

[5.3 A választott szövegek előfeldolgozása 36](#_Toc164375229)

[5.4 Statisztikai vizsgálatok 40](#_Toc164375230)

[6 A könyv szövegének tartalmi összefoglalója 45](#_Toc164375231)

[6.1 A tartalmi vizsgálat célja 45](#_Toc164375232)

[6.2 LSI-SVD 46](#_Toc164375233)

[6.3 TextRank 47](#_Toc164375234)

[7 Témakörök szerinti elemzés 50](#_Toc164375235)

[7.1 Szövegosztályozás gépi tanulással 50](#_Toc164375236)

[7.2 A vizsgált szövegkorpusz 57](#_Toc164375237)

[8 Irodalomjegyzék 60](#_Toc164375238)

[9 Mellékletek 64](#_Toc164375239)

Bevezető

Szakdolgozatom témájaként egy gépi tanulással elkészített elemzést választottam. Az aktuálisan elérhető idegen nyelvű szövegek lehetővé teszik, hogy azokon gépi tanulási módszerek felhasználásával a tartalomra vonatkozó elemzéseket hajtsunk végre. A dolgozatom célja annak bemutatása gyakorlati példákon keresztül, hogy egy szövegkorpusz automatizált feldolgozásával hogyan tudunk statisztikai kimutatásokat készíteni, a benne szereplő szavakat témakör szerint csoportosítani, illetve egy tartalmi összefoglalót is megfogalmazni. Ezekből az elemzésekből egy regény tartalmára, cselekményére, és a főszereplők kapcsolatára vonatkozó következtetéseket is fogok részletezni. A vizsgált regénysorozatról való eddigi ismereteimet próbálom majd igazolni a statisztikai és a gépi tanulásos vizsgálatok eredményeivel. A dolgozat alapja lehet egy olyan idegennyelv tanulását segítő eszköz fejlesztésének, amely a fenti szövegelemzési vizsgálatokat egy felhasználó által megadott tetszőleges szövegre is el tudja végezni, és a kapott eredményeket képes megjeleníteni az alkalmazáson belül.

A munkám ennek a klaszterezési problémának a megoldásával foglalkozik. Bemutatja a természetes nyelvi elemzés matematikai hátterét, és hasonló alkalmazási területeken a készen elérhető megoldásokat. Az elemzéseket Python programozási nyelven, és az ehhez elérhető adatelemző függvénykönyvtárak segítségével készítettem el. Az alábbiakban egy rövid leírást adok arról, hogy pontosabban mivel is fogok foglalkozni a dolgozatban.

A gépi tanulás és a mesterséges intelligencia területe az utóbbi néhány évben hatalmas népszerűségre tett szert. Ha gazdasági szempontból nézzük, ez a folyamat ugyan a 2010-es évek második felében még nem hatott látványosan a termelékenység növekedésére [1], viszont nem hagyhatjuk figyelmen kívül például a tudományos, kulturális életre gyakorolt hatását sem. Ez a folyamatos fejlődés termesztésen kihatott a digitális formátumban már meglévő szövegekkel kapcsolatos leíró, elemzési és összehasonlítási feladatokra is. Ezek a vizsgálatok közelebb vihetnek minket az egyes nyelvek, illetve nyelvek közötti kapcsolatok jobb megértéséhez.

A ma digitálisan rendelkzésre álló szövegeknek a nagysága sokszor már szinte áttekinthetetlen, és manuális, vagy egyszerű digitalizált formában történő elemzésük nemcsak időigényes, de általában teljeskörűen lehetetlen is. Ezt a folyamatot fejleszti tovább a gépi tanulás alkalmazása, amely lehetővé teszi ilyen nagy mennyiségű szövegbázis rendszerezett és minden eddiginél gyorsabb feldolgozását, illetve statisztikai, nyelvtani, és valamilyen szinten már a jelentésbeli elemzését is.

A szakdolgozatom középpontjában egy olyan eszköz kifejlesztése állt, amely amellett, hogy statisztikai elemzésekkel segíti az idegen nyelvek tanulását, a szövegeket automatizált módon témakörök mentén is csoportosítja. A vizsgálatom fontos része az a klaszterezési probléma, melynek megoldásával megkíséreltem létrehozni egy olyan rendszert, amely a rendelkezésre álló szövegkorpuszokat, jelen esetben a közismert Harry Potter könyvsorozat első regényének szövegét [36] tematikus struktúrába szervezi. Mindezek mellett igyekeztem elkészíteni egy, a könyv tartalmába, fontosabb mozzanataiba betekintést nyújtó összefoglalót is.

A könyv szövegének statisztikai alapú vizsgálatát négy különböző nyelven alkottam meg (angol, német, francia, spanyol). A tartalmi és a klaszterezési elemzéseket az angol nyelvű változaton készítettem el.

Ahogy már említettem, elemzéseim során a Python programozási nyelvet alkalmaztam, illetve felhasználtam az ehhez elérhető adatelemző függvénykönyvtárakat. Ezzel az erős technológiai bázissal megkíséreltem egy olyan szoftver létrehozását, amely segítséget nyújt azoknak, akik szeretnének mélyebben elmerülni a nyelvtanulás hagyományostól eltérő lehetőségeiben, illetve a különféle nyelvek mélyrehatóbb tanulmányozásában.

A jövőre nézve szerintem fontos alkalmazási terület lehet még az összehasonlító nyelvészet, amelyet szintén nagymértékben segíthet a természetes nyelvű szövegek gépi tanulás és mesterséges intelligencia segítségével történő elemzése. Erről a felhasználási területről is igyekszem egy kisebb ízelítőt adni a következő fejezetben.

Az NLP problémakör áttekintése

Ha az a célunk, hogy digitális formátumba hozott szövegeket elemezzünk, akkor adódik a kérdés, hogy milyen módszertan alapján tudjuk ezt elkezdeni. Kiindulhatunk abból, hogy van lehetőség hagyományos nyelvészeti elemzésekkel vagy programozási eszközökkel, algoritmusokkal (pl. kiválasztások, szűrések, keresések) statisztikai vizsgálatokat végezni a megadott szövegeken, ami jó kiindulási alap. Viszont az is fontos kérdés, hogy a szöveg milyen formában áll rendelkezésünkre, hogyan tudunk belőle releváns, gyakorlatban is hasznosítható információkat kinyerni. Ha például csak egy nyers, rendezetlen, elírásokkal teli szöveg, vagy egy weboldal szövege áll rendelkezésünkre, amelyben még akár html tag-ek is szerepelnek, akkor könnyen belátható, hogy ezen szövegek elemzésekor problémákba fogunk ütközni. Nem lesznek általánosíthatók az általunk kívánt információk (pl mondatok száma, szavak gyakorisága, stb) kinyerésére szolgáló erőfeszítéseink, mert fals információkat kaphatunk a nem a szöveghez tartozó karakterek vagy pontatlanságok miatt.

Természetesen az alapvető statisztikai elemzéseken túl szeretnénk továbbmenni a szövegvizsgálatainkkal, és ha már olyan jellegű információt is ki akarunk nyerni, mint például egy szöveg témaköre, nyelvtani struktúrája, vagy a közvetített érzelmek, akkor jön képbe az NLP (Natural Language Processing).

Az NLP

Az elemzéseim megalapozásához fontos volt, hogy a természetes nyelvek feldolgozásának (NLP, vagyis Natural Language Processing) témakörét megértsem. Ez a terület jelenleg már megkerülhetetlen a technológiai fejlődés és az adatmennyiség hihetetlen mértékű növekedésének köszönhetően. Az NLP a mesterséges intelligencia és a nyelvészet határmezsgyéjén mozog. Célja, hogy áthidalja az emberi és a gépi kommunikáció közötti szakadékot. Ebben a fejezetben szeretnék áttekintést adni az NLP alapelveiről, fejlődéséről és lehetséges jövőbeli alakulásáról.

A természetes nyelv megértéséhez az emberi kommunikáció szerteágazó bonyolultságával kell megküzdenünk. Mindezt igen nehéz feladat számítógépes formába átültetni. A korai NLP-rendszerek nagymértékben támaszkodtak különféle nyelvi szabályokra és kézzel készített nyelvtanokra. Chomsky generatív nyelvtana, amelyet a Syntactic Structures című korai művében adott közre [2], elméleti alapot adott a mondatok szintaktikai struktúráinak elemzésére és megértésére. Ez a megközelítés csak korlátozottan alkalmazható az emberi nyelv hatalmas változatosságának kezelésére.

A statisztikai módszerek és a gépi tanulás felé való elmozdulás jelentős löketet adott az NLP fejlődésének. A fontos eredmények között találjuk a rejtett Markov-modellek (HMM) bevezetését a beszédrészek címkézésére [3], valamint a statisztikai gépi fordítás témakörében elért előretörést. Ezeből a példákból láthatóvá vált, hogy az adatvezérelt módszerek felhasználhatók összetett nyelvi feladatok megoldására is. Mindez hozzájárult a gépi tanulás elterjedéséhez az NLP-ben.

Tehát az NLP célja, hogy támogassa a számítógépeket abban, hogy megértse, tolmácsolja és feldolgozza a természetes emberi nyelveket. Tehát mint már említettem, végeredményben az a feladata, hogy megkönnyítse az emberek és a gépek közötti kommunikációt, és a szavakat kontextuson belül tudja értelmezni [38].

A dolgozatomban az NLP-t arra fogom használni, hogy a Harry Potter első könyvének szövegéből következtetéseket vonjak le. Mint már volt róla szó, a vizsgálat során először statisztikai elemzéseket fogok végrehajtani a szövegen, és megvizsgálom, hogy a kapott eredmények megerősítik, vagy esetleg cáfolják-e a regény ismeretében meglévő eddigi tudásomat. Ezután megkísérlem a szöveget témakörök szerint csoportosítani, először felügyelt gépi tanulással, majd felügyelet nélküli tanulási modell segítségével is. Ezután rövidebb összegzéseket fogok létrehozni a könyv teljes tartalmáról, illetve külön-külön is az egyes fejezetekről.

Alapvető feladatok az NLP-ben

Ha a lényegi elemeket nézzük, az NLP többféle szövegelemzési feladatra is alkalmas. Az első a tartalom kategorizálása. Ennek során azonosítjuk a szöveg fő kulcsszavait, és ezek alapján csoportosíthatók a különféle szövegek. A második fő terület összegzések, kivonatok írása a szövegből. Egy nagy terjedelmű szövegből a mesterséges intelligencia képes egy rövidebb összegzést előállítani. A harmadik nagyobb téma a szentimentális elemzés. Az NLP akár azt is képes azonosítani, hogy az adott szöveg létrehozásakor az író milyen hangulatban volt, vagy mi az általános vélekedése, világnézete. Ez megállpítható a szóhasználatból és a különböző nyelvtani szerkezetekből. A negyedik fontos terület a szintaktikai vagy mondattani elemzés. Ennek olyan szövegeknél lehet nagy szerepe, amelyeknek nem ismert a keletkezési dátuma. A mondattani szerkezeteket és a szöveg szavait együtt elemezve meg lehet állapítani, hogynagyjából mikor, és hol keletkezhetett az adott szöveg [38]. Az alábbiakban ezeknek a feladatoknak egy részét, illetve más, speciális alkalmazási területeket fogok áttekinteni.

* + 1. Szövegosztályozás és érzelemelemzés

Az NLP egyik alapvető feladata a szövegosztályozás, ahol a cél egy dokumentum előre meghatározott osztályokba sorolása. Pang és Lee "Opinion Mining and Sentiment Analysis" című 2008-as munkájukban a szövegosztályozás egyik részterületét, a hangulatelemzést vizsgálták [4]. Ez a feladat a szövegben kifejezett érzelmek meghatározását jelenti. Minél pontosabban tudjuk meghatározni a szövegek, szavak mögött megbúvó érzelmi töltetet, annál közelebb kerülünk például a felhasználói vélemények jobb megértéséhez az értékelésekben, a közösségi médiában és a vásárlói visszajelzésekben, hogy csak néhány példát említsek.

* + 1. Nevesített entitások felismerése (NED)

A megnevezett entitások, például emberek, szervezetek vagy helyszínek neveinek azonosítása alapvető a lényeges információk megszerzéséhez. Erre remek példa a A CoNLL-2003, amely egy megosztott feladat részeként kiadott, nyelvfüggetlen nevesített entitásfelismerési adathalmaz [5]. Az adatok nyolc fájlból állnak össze, amelyek az angol és a német nyelvet fedik le. Mindkét nyelvhez tartozik egy gyakorlófájl, egy fejlesztőfájl, egy tesztfájl és egy nagy fájl megjegyzések nálküli adatokkal.

Az ezirányú kitatások ösztönözték a NED fejlődését, különböző megközelítésekkel. Többek között a szabályalapú rendszerek, illetve olyan gépi tanulási modellek tartoznak ide, mint például a feltételes véletlen mezők (CRF, vagyis Conditional Random Field) és újabban már a deep learning módszerek is ide sorolhatók.

* + 1. Gépi fordítás

A szövegek egyik nyelvről a másikra történő fordításának lehetősége már régóta az NLP egyik fontos célja. A neurális gépi fordítás (NMT vagyis Neural Machinr Translation) megjelenése, amelyet Sutskever et al. (2014) és Bahdanau et al. (2014) mutatott be, paradigmaváltást jelentett a kérdéskörben [6]. Az NMT egyik legfontosabb újítása, a figyelemmechanizmusok bevezetése lehetővé tette, hogy a modellek a bemeneti adathalmaz meghatározott részeire összpontosítsanak, ami jelentősen javította a fordítás minőségét.

A deep learning és az NLP

* + 1. Szóbeágyazások

A szóbeágyazások megjelenése forradalmasította a szavak NLP-rendszerek általi reprezentációját [7]. A beágyazások a szavak közötti szemantikai kapcsolatokat rögzítik, ez pedig lehetővé teszi a modellek számára a kontextus és a jelentés megértését. Például a Word2Vec, a GloVe és a fastText technikákat lehet érdemes kiemelni, amelyek a különböző NLP-alkalmazások szerves részévé váltak, és alkalmazásukat a gyakorlatban is látni fogjuk az általam hozott szövegek elemzése során.

* + 1. Rekurrens neurális hálózatok (RNN) és hosszú-rövidtávú memória (LSTM)

A nyelvek szekvenciális jellege vezetett ahhoz a felfedezéshez, hogy a rekurrens neurális hálózatok (RNN vagyis Recurrent Neural Network) jól felhasználhatók az NLP-feladatok megoldásához.

Hochreiter és Schmidhuber 1997-ben publikált LSTM-architektúrája [8] megoldotta az eltűnő gradiens problémáját, amely lehetővé téve az RNN-ek számára, hogy megragadják a szekvenciák hosszú távú függőségeit. A réshosszra való viszonylagos érzéketlensége az előnye más RNN-ekkel, rejtett Markov-modellekkel, illetve más szekvenciatanulási módszerekkel szemben. Célja, hogy az RNN számára olyan rövid távú memóriát biztosítson, amely több ezer időlépésre képes, tehát nevezhető „hosszú rövidtávú memóriának”. Többek között az idősorokon alapuló adatok osztályozására alkalmazható, illetve feldolgozására és előrejelzésére, például a kézírásbeszédfelismerés, gépi fordítás, beszédaktivitás-felismerés, robotvezérlés, videojátékok és az egészségügy területén. Érdemes megemlíteni, hogy az LSTM használata sikeresen beépíthető olyan feladatokba is, mint a nyelvi modellezés és a gépi fordítás.

* + 1. Transzformátorok és figyelemfelkeltő mechanizmusok

A Transzformer architektúra Vaswani és társai [9] általi bevezetése (2017) fontos mérföldkő volt az NLP fejlődésében. A Transzformerek az önfigyelési mechanizmusokra támaszkodva felülmúlták a korábbi modelleket a kontextusfüggő információk megragadásában. A Transzformer architektúra lett az alapja az olyan korszerű és közismertté váló modelleknek, mint a BERT [10] a kontextuális szóbeágyazások és a GPT (Generative Pre-trained Transformers) [11] a nyelvi modellezés területén.

Kihívások és jövőbeli irányok

* + 1. Etikai megfontolások az NLP-ben

Ahogy az NLP-technológiák egyre jobban elterjednek, úgy természetesen felmerülnek az elfogultsággal, a méltányossággal és a magánélet, illetve érzékeny vagy személyes adatok védelmével kapcsolatos etikai aggályok. Például érdemes megemlíteni a szóbeágyazásokban megjelenő részrehajlást, tehát megfelelő odafigyeléssel kell kezelnünk a nyelvi adatokban akaratlanul is megjelenő társadalmi elfogultságokat, előítéleteket, kategorizálásokat [12]. Ha belegondounk, egyértelmű, hogy az NLP-rendszereket felelősen kell fejlesztenünk és figyelmesen gondoznunk, ha a lehetőségekhez mérten igazságos és elfogulatlan eredményeket szeretnénk biztosítani.

* + 1. Multimodális NLP

A többféle modalitás, például a szöveg, a képek és a beszédhang integrálása a kutatások egyre növekvő területéhez tartozik. A legújabb munkák, mint például a CLIP, a modalitások közötti összevont tanulásban rejlő lehetőségeket tárgyalják, a fenti példában a szövegek és a hozzájuk tartozó képek összeillesztése területén [13]. A multimodális NLP hozzájárul a különféle típusú tartalmak átfogóbb megértéséhez és generálásához, és alkalmazásai lefedik többek között a képfeliratozást segítő technológiákat.

Mint az végigkövethettük, a természetes nyelvfeldolgozás figyelemre méltó átalakuláson ment keresztül az utóbbi évtizedekben, és átölelte a szabályalapú rendszereket, az adatvezérelt megközelítéseket, a deep learning megjelenésével bezárólag. A nyelvészeti elméletekre, statisztikai módszerekre és kortárs neurális architektúrákra támaszkodva az NLP egyre sikerekesebben alkalmazható az emberi nyelvek megértésében és generálásában. Azonban továbbra is rengeteg kihívás áll fenn, és folyamatos interdiszciplináris együttműködésre van szükség az etikai aggályok kezelése, az alkalmazkodóképesség javítása és a multimodális megértés fejlesztése érdekében. Ahogy az NLP tovább fejlődik, például az egészségügyi ellátástól az oktatásig számos területen kifejtett hatása azt bizonyítja, hogy a mesterséges intelligencia egyre bővülő területének egyik fontos bázisává vált.

NLP a gyakorlatban

* + 1. Az elemzések vázlata

A természetes nyelvi feldolgozás és az adatelemzés metszéspontjában elhelyezkedő szövegelemzés az utóbbi években jelentős teret nyert. Ez a tudományág, mint már láthattuk, a strukturálatlan szöveges adatokból értelmes meglátások, minták és tudás kinyerésével foglalkozik.

A Python egy sokoldalú és nagy teljesítményű programozási nyelv, a szövegelemzés területén dolgozó szakemberek és kutatók egyik legelterjedtebb eszközévé vált. A Python szövegelemzésre való felhasználása robusztus és hatékony megközelítést kínál a hatalmas mennyiségű szöveges adat feldolgozásához, elemzéséhez és értékes információk kinyeréséhez. A Python-hoz átfogó könyvtárak és eszközök állnak rendelkezésre, az érzelmek elemzésétől és a nevesített entitásfelismeréstől kezdve a dokumentum klaszterezésen és a témamodellezésen át, így ideális választásnak tartottam az elemzéseim elkészítéséhez.

* + 1. Az alkalmazott megközelítés

Dipanjan Sarkar „Text Analytics with Python: A Practitioner’s Guide to Natural Language Processing” [14] című könyvét hasznosítottam a az NLP témakörének gyakorlati szempontú megismeréséhez. A könyv a későbbiekben részletezésre kerülő statisztikai és gépi tanulásos szövegelemzésekhez is remek fogódzóként szolgált.

Mint azt a bevezetőben már megállapítottam, a modern társadalom számára a hatalmas információmennyiség kezelése egyre komolyabb, és egyre több haszonnal kecsegtető kihívást jelent, és az NLP kiemelkedő játszik ebben a folyamatban. Sarkar könyvének gyakorlatias megközelítése révén rálátásom nyílt a szövegekből való információkinyerés, dokumentumösszegzés és témamodellezés alapvető fogalmaira.

A könyv részletezi a különböző technikákat, amelyek lehetővé teszik az információk hatékony kinyerését a szöveges adatokból. Az NLP terén elért eddigi eredmények felhasználása segíti az információk gyorsabb és hatékonyabb feldolgozását, elősegítve a releváns tartalmak azonosítását a rengeteg redundáns és lényegtelen adat között. A szinguláris értékbontás és az alacsony rangú mátrixközelítés matematikai elveit felhasználva bemutatható, hogyan lehet integrálni az NLP algoritmusokat egy adott szövegkorpusz mélyebb feldolgozásába.

A feldolgozás során, mint azt hamarosan látni fogjuk, részletesen elemeztem az NLP problémakör három stratégiáját: a statisztikai elemzéseket, a témák szerinti modellezést és automatikus dokumentum összefoglalást kulcskifejezésekkel és párbeszédekkel. Az NLP-ben alkalmazott fejlett kulcskifejezés összegyűjtési módszerek, illetve a téma szerinti klaszterezési technikák segítik a nagy adatmennyiségben való tájékozódást és a releváns információk azonosítását. Azt is részletezni fogom, hogyan lehetséges finomítani a témák szerint kialakított klasztereket, és hogyan lehet meghatározni szövegfüggően az optimális témák számát.

Természetesen a kapott eredmények értelmezése, megértése és felhasználása is kritikus szempont. Az automatizált dokumentum-összefoglalás kapcsán ismertetett látens szemantikai elemzés és a TextRank módszerek gyakorlati példákon keresztül segítik majd megérteni, hogyan alkalmazhatók ezek a módszerek valós adatokon, és milyen hasznos eredmények érhetők el velük.

Bár már valamivel régebbi forrás, viszont a dolgozatomhoz szintén hasznos vezérfonalat nyújtott a "Natural Language Processing with Python" című könyv [15], amely a fentebb említett mű mellett kiváló alapot szolgáltat a feladatmegoldásom során gyakran használt NLTK (Natural Language Toolkit) nevű könyvtár eszközkészletének megértéséhez.

Kutatásom továbbá kiterjedt a különböző témamodellezési technikák fejlesztésére is, beleértve a latens szemantikus indexelést, a latens Dirichlet-kiosztást, és a nemrégiben bevezetett nem-negatív mátrix-faktorizációt a Gensim és a Scikit-Learn keretrendszerek segítségével. Az NLTK-ról és az utóbb említett könyvtárakról a következő fejezetben lesz szó részletesebben.

Összehasonlító nyelvelemzés

A gépi tanulás és a természetes nyelvi feldolgozás (NLP) számos átalakító eszközt ad nekünk az összehasonlító nyelvi elemzéshez. Ez új lehetőséggel szolgál a kutatók számáraba különböző adathalmazok nyelvi árnyalatainak mélyebb vizsgálatára. Ezek a technológiák a minták kinyerésének automatizálásával megkönnyítik a nyelvhasználat hasonlóságainak és különbségeinek azonosítását, és egyszerűsítik az összehasonlító elemzés folyamatát. Két fontosabb, új eredményt szereték itt megemlíteni, amelyek bizonyítják az imént megfogalmazott állításokat.

Az első, fontos tanulmány 2013-ból származik, amelyben Mikolov és társai bemutatták a Word2Vec-et, egy úttörőnek számító modellt a természetes nyelvi feldolgozás témakörében [16]. A tanulmány a szóreprezentációk vektortérben történő hatékony becslésével foglalkozik. Részletesen bemutatják a Word2Vec skip-gram és „continuous bag of words” (CBOW) architektúráját. Kiderül, hogy a modell hogyan ragadja meg a szemantikai kapcsolatokat a szavak többdimenziós vektortérbe való leképezésével. Ez lehetővé teszi a szemantikai hasonlóság mérését és felszínre hozza a nyelvi árnyalatokat. A tanulmány jelentősen befolyásolta a későbbi kutatásokat a témakörben, és alapját képezte az NLP hatékonyabb alkalmazásának. Ide tartozik például az összehasonlító nyelvi elemzés és az időbeli nyelvi változások folyamatának megértése is.

A másik érdekes publikációt Gulordava és munkatársai adták közre 2018-ban. Ők az emberi nyelv hierarchikus szerkezetét vizsgálták rekurrens neurális hálózatok (RNN) segítségével [17]. A tanulmány mélyebben elemzi a neurális hálózatoknak azt a képességét, hogy szintaktikai struktúrákat tanuljanak és reprezentáljanak. Ezen kívül bemutatnak olyan módszereket, amelyekkel e hálózatok „szondázhatók”, azzal a céllal, hogy megértsük a hierarchikusan kódolt nyelvi információkat. Bár a munka elsősorban a szintaktikai struktúrákra összpontosít, emellett azonban értékes betekintést nyújt a fejlett neurális hálózati modellek segítségével végzett összehasonlító nyelvi elemzés területére is. Kutatásuk segítségével könnyebben megérhetővé válik, hogy a deep learning modellek hogyan rögzítik és dolgozzák fel a különböző nyelvekben a hierarchikus nyelvi jellemzőket.

Python fejlesztőkörnyezet összeállítása

Szerencsére az elmúlt évtizedek során programozási nyelvek rendkívül sokszínű tárháza vált elérhetővé, amelyek különféle feladatokra és projektekre lettek kialakítva. Mielőtt rátérnék az általam választott Python fejlesztőkörnyezet bemutatására és részletezésére, szeretnék egy átfogó áttekintést nyújtani néhány ismertebb, szélesebb körben használatos programnyelvről, amelyek hasznosnak bizonyulhatnak gépi tanulással kapcsolatos feladatok, elemzések elvégzéséhez.

Lehetséges programozási nyelvek

A lista összeállításához figyelembe vettem, hogy mely nyelvek rendelkeznek kiterjedt eszközkészlettel gépi tanulásos, NLP feladatok elvégzéséhez. Az alábbi nyelveket találtam célravezetőnek.

* + 1. A Python

A Python nyelv a sokoldalúságával és könnyen érthető szintaxisával emelhető ki a többi programnyelv közül. Alkalmazási területei közé tartozik például a webfejlesztés, de általánosan adatelemzési és gépi tanulási feladatokra használják. Az olvasható, letisztult kód és a gyors fejlesztési idő miatt ideális lehet mind a kezdő, mind a tapasztaltabb fejlesztők számára. Gazdag rendelkezésre álló könyvtárkészletét és széleskörű közösségi támogatását érdemes még kiemelni, amelyek fontos érvek voltak a Python használata mellett.

Egyes statisztikák és elemzők szerint a teljesítménye nem mindig a legkiemelkedőbb, különösen erőforrásigényesebb alkalmazásokban [18] [19] ]20]

* + 1. Az R nyelv

Az R kifejezetten magas szintű statisztikai analízis és adatvizualizálás céljából készült. Mindenki számára, aki mélyebben meg akarja érteni a gépi tanulással vagy statisztikával kapcsolatos matematikai számításokat, ez megfelelő nyelv lehet.

Az R lehetővé teszi a gyors prototípusalkotást és az adathalmazokkal való munkát az gépi tanulásos modellek felépítése érdekében. Ha például hatalmas bekezdéseket szeretnénk szavakra vagy mondatokra bontani, és mintákat keresni, az R hatékonyabb, mint a Python.

Az R emellett kiterjedt könyvtár- és eszközgyűjteménnyel rendelkezik, amelyek segítik a gépi tanulási folyamatokat. Ezek a fejlett adatelemző csomagok a modellezés előtti és utáni szakaszokra is kiterjednek, és olyan speciális feladatokra készültek, mint a modell validáció vagy adatvizualizáció (pl. a Tidyr ’adattisztításra’, a Gplot2 vizualizációra használható). [21]. Említést érdemel még a quanteda, ami NLP eszközöket biztosít az R nyelvhez, például szövegmodelleket, statisztikai számításokat, aatvizualizációt (textplots), és hangulat analízis eszközöket.

Mindezek ellenére az R hátrányai közé tartozik, hogy egy nehéz nyelv, ami megnehezíti, hogy szakértőket találjunk egy projekthez vagy csapathoz. Minden új csomag, amit használatba veszünk, tanulást igényel, illetve az R-ről nem áll rendelkezésre alapos dokumentáció. Emellett az R következetlenül működhet, mivel algoritmusai harmadik féltől származnak [21].

* + 1. A C#

A C# közismert objektum-orientált programozási nyelvhez tartozik egy NLP célokat szolgáló függvénykönyvtár, a catalyst nlp[[1]](#footnote-1). A spaCy (később lesz róla szó) kialakítását követve, előre betanított modelleket, a szó- és dokumentumbeágyazások betanítását, illetve rugalmas entitásfelismerő eszközöket kínál. Például szinte tökéletesen reguláris kifejezés-mentes tokenizáló algoritmust ígér, ami teljesítmény szempontból hasznos. Említésre méltók az előre betanított nyelvi modellek, a rövidítés- és érzékfelismerő eszközök, a nyelvfelismerés beágyazott FastText és StarSpace modulokkal, illetve előre felépített nyelvi modulok és lemmatizáció (a spaCy keresési táblázataival).

* + 1. A Matlab

A MATLAB számos felületet kínál az adattudományi elemzésekhe az adatok eléréséhez és előfeldolgozásához, a gépi tanulás és a prediktív modellek létrehozásához. Lehetőségünk van kipróbálni sokféle gépi tanulásos módszert a Statistics and Machine Learning Toolbox segítségével, például klaszteranalízist (felügyelet nélküli), osztályozási modelleket (felügyelt gépi tanulás). Fejlett adatvizualizációs képességekkel és letisztult kóddal rendelkezik, ami hasznossá teszi ahhoz, hogy megismerjük a legfontosabb gépi tanulási és AI modellezési technikákat.

Alkalmazások használatával vagy mindössze néhány sornyi MATLAB-kóddal már statisztikai, gépi és deep learning technikákat alkalmazhatunk algoritmusok tervezéséhez, adatok előkészítésére és címkézésére, illetve kódgenerálásra és beágyazott rendszerekre való telepítésre [39].

* + 1. A Javascript

A nodejs egy nyílt forráskódú, keresztplatformos Javascript futtatókörnyezet, amely szerver oldali manipulációkat tesz lehetővé. A nodejs-hez többféle NLP modell elérhető. Péládul az NLP.js tartalmaz egy kifejezés nyelvét kitaláló eszközt, szótávolságot számító modellt (Levenshtein távolság, amely számszerűsíti, mennyire különbözik egymástól két szó), szótövesítőt és tokenizálót több nyelvhez. Elérhető emellett mondatok hangulatelemzése, nevesített entitásfelismerés és kezelés, többnyelvű támogatás, és hasonló karakterláncok elfogadása, így a bevezetett szövegnek nem kell teljesen pontosnak lennie. Találunk még itt többek között NLP osztályozót, a kifejezések szándékok szerinti osztályozására [40].

Emellett elérhető még másfajta, nodejs környezetben működő NLP csomagok is, péládul a compromise és a natural.

* + 1. Összehasonlítás

Ha a Python főbb jellemzőit szeretnénk összehasonlítani a többi felsorolt programnyelv tulajdonságaival, akkor a következő fontosabb megállapításokra juthatunk:

Ha az R nyelvvel hasonlítjuk össze, akkor a Python általánosabb feladatokra, így gépi tanulásra, de akár webfejlesztésre is ideális lehet. Az R inkább statisztikai vizsgálatokra specializálódott, jó vizualizációs képességekkel.

Ha a JavaScript viszonylatában vizsgálódunk, akkor abban lehet a fő különbséget meglátni, hogy a Python inkább a szerveroldali fejlesztéshez alkalmasabb, illetve a gépi tanulásos, és tudományos számításokhoz testhezállóbb. A JavaScript pedig eredetileg kliensoldali webfejlesztésre lett optimalizálva.

Ezeknél az összehasonlításoknál fontos hangsúlyozni, hogy természetesen minden programnyelvnek megvannak a maga előnyei és ideális alkalmazási területei. Az optimális választás mindig a konkrét feladat vagy projekt céljától és a követelményektől függ. Emellett egy adott feladat megoldására többféle programozási nyelv és NLP könyvtár is alkalmas lehet.

* + 1. A Python előnyei

Konklúzióként az alábbi előnyök sorolhatók fel, amelyek arra ösztönöztek, hogy a Pythont válasszam a szakdolgozatom szövegelemzési feladatainak programozási nyelveként: a Pythonban írt kódok könnyen értelmezhetők és olvashatók, és ez elősegíti a hatékony munkát és a karbantarthatóságot. Emellett a Python gazdag és sokoldalú könyvtárkészlettel rendelkezik, többek között az NLP területén, amely jelentősen felgyorsítja a fejlesztést. A Pythonnal aktív közösség foglalkozik, könnyen hozzáférhetők dokumentációk, segítségek és fejlesztői erőforrások. Természetesen, mint láttuk, a fentebb felsorolt más programnyelvű eszközök (pl. Matlab, R) is kiválóan alkalmasak ugyanilyen feladatokra.

Azonban nekem a jól olvasható kód, a rugalmasság, az NLP könyvtárak széles választéka és a gyors fejlesztési idő fontos előnynek bizonyultak. Ezek a tulajdonságok ösztönöztek arra, hogy a Pythont válasszam a feladatomban felszínre kerülő gépi tanulásos problémák megoldásához, implementálásához, illetve a megoldások különböző adatkészletekkel és paraméterekkel való tesztelésében [20].

Fontosabb Python fejlesztőkörnyezetek

Most, hogy a választott programozási nyelv már adott, a Python fejlesztőkörnyezetünk (IDE) kiválasztása szintén fontos lépés a hatékony fejlesztéshez, és számos lehetőség közül választhatunk a projektek jellegétől függően. Az alább összegyűjtött Python IDE-k mindegyikében találhatunk előnyös és hátrányos tulajdonságokat. Igyekeztem kiemelni, hogy miért esett végül a választásom a Spyder-re. De lássuk az általam vizsgált fejlesztői környezetek legfontosabb aspektusait.

* + 1. DataCamp Workspace

A DataCamp Workspace egy olyan Python fejlesztőkörnyezet, amelyet kifejezetten az online tanuláshoz terveztek. A platform interaktív kurzusokat és gyakorlatokat ad, amelyekhez beépített fejlesztői környezet áll rendelkezésre. Ennek előnye, hogy a felhasználók azonnal alkalmazhatják a tanultakat a gyakorlatban, anélkül, hogy saját fejlesztőkörnyezetet kellene telepíteniük. Könnyen elérhetők a kurzusokhoz kapcsolódó kódok és gyakorlatok, és azonnal láthatók a változtatások eredményei.

Hátránya az offline használat korlátozottsága. Továbbá a platformot elsősorban oktatási célokra fejlesztették ki, a fejlesztőknek kevésbé van lehetőségük testre szabni a környezetet az egyéni igényeikhez. Fontos szempont az is, hogy az online környezetek általában kisebb teljesítményűek lehetnek, különösen nagyobb projektek vagy adatfeldolgozási feladatok esetén.

* + 1. Jupyter Notebook

A Jupyter Notebook egy interaktív környezet, amely lehetővé teszi a kód, a szöveg és a képek kombinálását egy dokumentumban. Széles körben alkalmazzák tudományos kutatásokhoz és adatelemzési projektek során. Az interaktivitás révén a felhasználók lépésről lépésre fejleszthetik és dokumentálhatják a kódot, miközben megmarad a projekt is. Többek között a Python, R és Julia programozási nyelveket támogatja a platform, így sokoldalúan használható tudományos tevékenységekre. A Jupyter Notebook nem mindig optimális nagyobb projektek létrehozására.

* + 1. Google Colaboratory

A Google Colaboratory egy felhőalapú Jupiter Notebook környezet, amely a Google fiókkal rendelkező felhasználóknak ingyenesen elérhető. Ideális a közös munkához és a projektek megosztásához, mivel a felhasználók egyszerűen megoszthatják a Colab dokumentumaikat, és közösen dolgozhatnak rajtuk. A Colab GPU-támogatása lehetővé teszi a gyors és hatékony gépi tanulási projektek futtatását a felhőben. Azonban használata korlátozott a Google Drive-hoz való kötöttség miatt.

* + 1. Atom

Az Atom egy ingyenes, nyílt forráskódú szövegszerkesztő, amelyet a GitHub fejlesztett ki. Könnyen testreszabható és bővíthető, és számos kiterjesztéssel szolgál a Python fejlesztők számára. A közösség által fejlesztett kiegészítők révén személyre szabható a felhasználók igényeinek megfelelően, és segíti a hatékony kódszerkesztést [22].

* + 1. Thonny

A Thonny egy könnyen használható fejlesztőkörnyezet, kifejezetten a kezdő programozók számára tervezték meg. Ez az egyszerűsített környezet tartalmazza a Python interpretert és egy alapvető kódszerkesztőt, így minimálisra csökkenti a konfigurálási folyamatot. A Thonny ideális választás lehet azoknak, akik gyorsan el szeretnének kezdeni a Python programozást, anélkül, hogy sok időt töltenének beállításokkal és konfigurációval. Azonban kevésbé alkalmazható nagyobb projektekhez [23].

* + 1. JupyterLab

A JupyterLab a Jupyter Notebook kiterjesztése, további funkcionalitásokkal és testreszabási lehetőségekkel. Ez a környezet lehetővé teszi a felhasználók számára, hogy több ablakban és lapon dolgozhassanak egyszerre, optimalizálva ezzel a munkafolyamatot. A beépített fájlkezelő, kiterjesztett widgetek és tematikus kiegészítők sorolhatók még az eszköztárához. Nem támogat minden Jupyter Notebook kiegészítőt [24].

* + 1. Spyder

A Spyder (Scientific Python Development Environment) egy olyan tudományos számítási fejlesztőkörnyezet, amelyet kifejezetten adatvizualizációs és adatelemzési feladatokhoz terveztek. Az előnyei közé tartozik a könnyű telepítés és konfiguráció, valamint a gyors és interaktív adatvizualizáció lehetősége. A Spyder tartalmazza a SciPy, NumPy és a Matplotlib könyvtárakat, ami megkönnyíti az adatelemző munkát. Az Ipython konzol mellett a beépített változóinspektor és a változók értékeinek vizualizációja segíti a változók állapotának folyamatos nyomon követését és a hibakeresést. Emellett a Spyder támogatja a Jupyter Notebookot, így a projektjeinkben kombinálhatjuk a két környezet előnyeit. A Spyder további funkcionalitásokkal bővíthető a kiterjeszthető plugin rendszer segítségével, így az egyéni igényeinkhez testreszabható fejlesztőkörnyezetet kapunk.

A Spyder előnyei közé tartozik a könnyű használat, az adatelemző eszközök gazdagsága és az IPython konzol által biztosított interaktivitás. Ideális választás lehet azok számára, akik az adatelemzésben és a tudományos számításokat szeretnének végezni, és egy olyan fejlesztőkörnyezetet keresnek, amely támogatja mind a fejlesztési, mind az elemzési folyamatokat. Ugyanakkor fontos azt is megemlíteni, hogy a Spyder közössége és bővíthetősége kisebb lehet más, nagyobb közösséggel rendelkező IDE-khez képest.

Mindezt összefoglalva, azért esett végül a választásom a Spyder használatára, mert elég kompaktnak és jól felépítettnek éreztem a moduljait, amelyekkel hatékonyan tudtam végigkövetni, illetve tesztelni a fejlesztés során fellépő köztes állapotokat. Végül az elkészült elemzések ereményeit is integráltan kézhez kaptam. Ezt a folyamatos interaktivitást rendkívül produktívnak éreztem. A sokféle rendelkezésre álló modul ellenére szerintem kellően letisztult a felhasználói felület. Végig azt éreztem, hogy jól kezemben tudom tartani a különböző funkciókat, és teljeskörű támogatást kapok a kódjaim karbantartásához, teszteléséhez, és az eredmények kiértékeléséhez. Konklúzióként kijelenthető, hogy a Spyder a tudományos jellegű Python fejlesztés terén hasznos eszköz, különösen, ha nagyobb mennyiségű adat elemzésére, interaktív feldolgozására és vizualizációjára van szükség [24].

Függvénykönyvtárak

Most pedig térjünk rá azoknak a Python függvénykönyvtáraknak a bemutatására, amelyeket vizsgálataim során alkalmaztam. Ezek a könyvtárak kifejezetten az NLP területén végzett kutatómunkát támogatják, stabil és kiterjedt matematikai hátteret adnak a kívánt szövegelemzések elvégzéséhez.

* + 1. Matplotlib [28]

A Matplotlib fejlett matematikai algoritmusokat alkalmaz látványos és informatív ábrák létrehozásához, adatvizualizációhoz. Tartalmaz algoritmusokat adatpontok interpolációjához, görbeillesztéshez és statisztikai vizualizációhoz. A könyvtár integrációja a NumPy-al hatékony numerikus műveletekre ad lehetőséget, ami növeli a kigenerált vizualizációk matematikai hasznosságát.

* + 1. Pandas [29]

A Pandas a statisztikai modellezés és az idősor elemzés integrálásával bővíti matematikai eszköztárunkat. A könyvtár tartalmaz gördülő statisztikákhoz, exponenciálisan súlyozott mozgóátlagokhoz és autokorrelációhoz készült függvényeket, ezzel időfüggő adatok elemzését teszi lehetővé. Emellett a Pandas matematikai optimalizációs algoritmusokat használ a hatékony adatkiigazításhoz és aggregáláshoz, optimalizálva a számítási teljesítményt.

* + 1. SpaCy [30]

A spaCy fejlett matematikai modelleket, például transzformátor-architektúrákat használ. Figyelemmechanizmusok és mély neurális hálózatok alkotják a nevesített entitások felismerésének és a beszédrészek címkézési képességeinek alapját. A könyvtár matematikai kifinomultsága kiterjed a beágyazásokra is, ami lehetővé teszi a szavak és kifejezések folyamatos vektortérben történő reprezentációját, így képessé téve arra, hogy megragadja a szemantikai kapcsolatokat.

* + 1. NLTK (Natural Language Toolkit) [15]

Az NLTK túlmutat az alapvető nyelvfeldolgozáson, mivel olyan matematikai eszközökkel rendelkezik, mint az entrópia-alapú jellemzőválasztás információelmélete és az olyan valószínűségi modellek, mint a Naive Bayes osztályozó. A különböző távolságmérők implementációja lehetővé teszi a hasonlóságelemzést és a klaszterezést a szövegbeli különbözőség matematikai mérőszámai alapján. Az NLTK sokoldalúságát az adja, hogy statisztikai technikákat használ a szövegosztályozáshoz és a hangulatelemzéshez.

* + 1. NumPy [31]

A NumPy matematikai arzenálja kiterjed a lineáris algebrára, illetve szinguláris értékek felbontását, mátrix faktorizációt és sajátérték-számításokat tartalmaz. Fourier-transzformációs funkciói megkönnyítik a frekvenciatartományok elemzését, míg a numerikus integrációs és differenciálási eszközök hozzájárulnak a tudományos szimulációkhoz. A NumPy átfogó matematikai függvénykészlete megalapozza a nagy léptékű tudományos számításokhoz szükséges numerikus hatékonyságot.

* + 1. Gensim [32]

A Gensim továbbfejleszti matematikai eszköztárát az online tanuláshoz és a szöveges adatok áramlásához szükséges inkrementális algoritmusokkal. Megvalósítja a latens szemantikai elemzés (LSA) és a latens Dirichlet-allokáció (LDA) algoritmusait, amelyekkel közelebb kerülünk a szemantikai kapcsolatok szövegből való kinyeréséhez. Mátrix-faktorizációs technikákat foglal magában a témamodellezéshez és a hasonlóságelemzéshez, illetve matematikai optimalizálási módszereket használ a modellképzéshez. A Gensim képes nagy szövegkorpuszok látens struktúráinak megragadására, ami értékes eszközzé teszi a rejtett szemantikai kapcsolatok feltárása kapcsán.

* + 1. Scikit-learn

A Scikit-learn matematikai alapja magában foglalja a modellképzés és paraméterhangolás optimalizálási technikáit. A többdimenziós skálázáson és spektrális beágyazáson alapuló sokrétű tanulási algoritmusok implementációja geometriai perspektívát ad a többdimenziós adatelemzéshez. A scikit-learn továbbá integrálja a hipotézisvizsgálat és a modellértékelés statisztikai módszereit, így átfogó csomagot nyújt a gépi tanulással foglalkozó szakemberek számára.

* + 1. NetworkX [34]

A NetworkX eszköztárába tartoik a fejlett gráfelméleten alapuló moduláris optimalizáció és a spektrális klaszterezésen alapuló közösségfelismerő algoritmusok. Péládul a Katz-centralitás és a sajátvektor-centralitás segítségével képes számszerűsíteni egy hálózatban található csomópontok fontossági súlyát. A könyvtár matematikai gazdagsága kiterjed a hálózati motívumok azonosítására és az irányított gráfok maximális folyamának megtalálására szolgáló algoritmusokra is. Mindez mutatja a könyvtár komplex hálózatelemzésben való hasznosságát.

* + 1. SciPy [35]

A SciPy tovább bővíti matematikai eszköztárunkat optimalizációs algoritmusokkal, például a korlátozott és korlátlan minimalizálási technikákkal. A könyvtár jelfeldolgozó modulja szűrési és spektrumanalízis módszereket integrál, illetve statisztikai funkciói hipotézistesztek és valószínűségi eloszlások széles skáláját ölelik fel. A SciPy a tudományos számítások és adatelemzés átfogó eszközeként szerepel a palettán.

Összefoglalva, ezek a Python-könyvtárak együttesen hatékony környezetet adnak a tudományos számításokhoz, az adatelemzéshez és a természetes nyelvi feldolgozáshoz, felhasználva a rendelkezésre álló matematikai eszközöket és algoritmusokat. Az idézett irodalmi hivatkozások további betekintést nyújtanak e könyvtárak fejlesztésébe és alkalmazásába.

A Spyder konfigurációja

Mielőtt rátérnék a Spyder bemutatására, szeretném megemlteni, hogy a munkám kezdetén az Anaconda AI projekt mededzselő platformjával. Az Anaconda repository-jában megtalálható a Conda nevű csomag menedzselő rendszer, amelyet a munkám kezdetén párhuzamosan használtam a pip-pel, a Python package menedzselő csomagjával. A Conda más programnyelveket is támogat a Pythonon kívül.

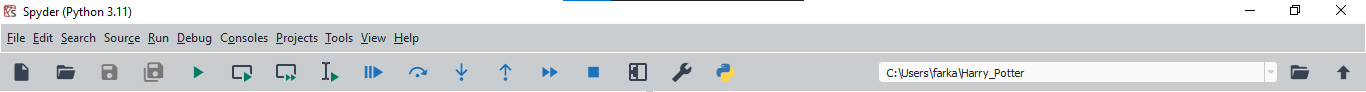
Az Anaconda Navigator[[2]](#footnote-2) nevű, asztali számítógépekre letölthető Anaconda disztribúcióval lehetőségünk nyílik többféle, gépi tanulást is támogtó open source szoftver használatára, és egy lokális fejlesztői környezet összeállítására. Segítségével menedzselhetők az általunk választott alkalmazások, csomagok, és környezetek, egy grafikus felületen. Az Anaconda weboldalán rendelkezésre állnak csokorba szedve a legfontosabb adatelemzési, adatvizualizációs, AI és gépi tanulásos, NLP open source könyvtárak. A függvénykönyvtárakról szóló alfejezetet nagyrészt ebből inspirálódva rendszereztem.

Az Anaconda Navigator letöltése után a felületén felkínált lehetőségek közül telepítettem a Spyder 5.4.3 verzióját, Python 3.11 verzióval. Térjünk át ennek a bemutatására.

A Spyder

A fejlesztéshez a már korábban bemutatott Spyder fejlesztőkörnyezetet használtam. Ez egy kifejezetten Python programozáshoz tervezett integrált fejlesztőkörnyezet (IDE). A Spyder hivatalos weboldala[[3]](#footnote-3) releváns információkat nyújt a legújabb verziókról, a funkciókról és az átfogó dokumentációról. Egyik előnye, hogy kompatibilis Windows, macOS és Linux típusú rendszerekkel is.

Tehát, miután az Anaconda Navigator befejezte a telepítést, elindítottam a Spyder alkalmazást. Amikor megnyílik a Spyder IDE, egy egységes és intuitív kezelőfelület tárul elém. A Spyder kezelőfelülete különálló panelekbe rendeződik, beleértve az Editor (Szerkesztő), a Console (Konzol), a Variable Explorer (Változó követés) és még sok mást. Rövid idő alatt felfedeztem a felületeket és az elrendezéseket. A Spyder felhasználói felületét úgy alakították ki, hogy átfogó, mégis a felhasználó érdekét is szem előtt tartó környezetet

biztosítson a Python-fejlesztéshez. Ehhez a más egyéb népszerű szövegszerkesztőkhöz,

*1. ábra: a Spyder irányítópanelje*

illetve fejlesztőkörnyezetekhez hasonló látványt nyújtó menüszalag formátum, rögtön az „otthonosság” érzetét keltheti a felhasználóban. A funkciók ikonokkal való megfeleltetése átláthatóságot és gyorsabb kezelhetőséget kölcsönöz, ezzel is támogatva a fejlesztőt.

A kódok megírásához az Editor ablaktáblában létrehoztam a szükséges Python szkripteket. A szkripjeim és függvénykönyvtáraim megírása közben a Console ablaktábla használatával tudtam a kódot futtatni és tesztelni. A Spyder hasznos hibakeresési eszközöket ad a debugger funkcionalitásban, ezt a kék színű nyilak segítségével lehet működtetni (futtatás, breakpointoknál megállás, függvényekbe be- és azokból kilépés).

Az alábbiakban szeretném kifejteni az egyes menüpontok pontos funkcióját[[4]](#footnote-4):

Editor: A Spyder integrált fejlesztőkörnyezetének (IDE) lényege a többnyelvű szerkesztőablak, amelyben a forrásfájlok létrehozását, megnyitását és módosításait tudjuk kezelni. Például automatikus kitöltés, valós idejű elemzés, szintaxis kiemelés, valamint vízszintes és függőleges felosztás segíti a munkát.

Key component: A baloldali sáv mutatja a kódsorok számait és az aktuális fájlban található kódelemzési figyelmeztetéseket. Ha rákattintunk egy sor számára, kiemeli az adott sorban lévő szöveget, és ha a sorszámtól jobbra kattintunk, breakpoint kerül az adott sorra, amit hibakeresés közben hasznosíthatunk.

A görgetősávok megkönnyítik a függőleges és vízszintes navigációt az adott fájlban.

A (jobb gombbal történő kattintással elérhető) kontextusmenü a rákattintott elemhez tartozó műveleteket mutatja be.

Az opciók menü, amelyet a jobb felső sarokban található háromsávos ikon jelöl, tartalmazza a szerkesztővel kapcsolatos hasznos beállításokat és műveleteket.

A Szerkesztő ablaktábla tetején található helymeghatározó sáv az aktuális fájl teljes elérési útvonalát jelzi.

A fülsáv az összes jelenleg megnyitott fájl nevét mutatja. Ezenkívül a bal oldalon található a lapok böngészése gomb, amely minden megnyitott lapot megjelenít, így gyorsan tudunk váltani közöttük. Ez előnyös funkció akkor, amikor már sok fájl van megnyitva.

A szerkesztőben a lapok segítségével böngészhetünk és navigálhatunk a megnyitott fájlok között. Ha rákattintunk a lapszalag bal oldalán található ’Browse tabs’ gombra, akkor a megjelenik a megnyitott fájlok listája, az aktuális aktív lap bejelölésével. Lehetséges átrendezni a fájlok sorrendjét ’drag and drop’ módszerrel.

File Switcher funkció: A szerkesztőn belül a fájlváltó megkönnyíti a navigációt és átmenetet a különböző megnyitott fájlok között. Ez a funkció hasznos a több megnyitott fájl közötti hatékony keresésben. A Fájl ‣ Fájlváltó menü vagy a Ctrl-P parancsikon segítségével érhető el, és tartalmaz egy keresési funkciót. A megnyitott fájl nevének bármely töredékének megadásával az Enter megnyomásával gyorsan átválthatunk rá, ha létezik.

Split panels funkció: A szerkesztő rugalmasan osztható vízszintesen és függőlegesen számos különálló panelre, ha szükséges.

Run File: Egy teljes szerkesztési fájl végrehajtása a Run ‣ Run menüpont, a Run file eszköztár gomb vagy az F5 billentyű segítségével. A legutóbb végrehajtott fájl újbóli futtatásához a Run ‣ Last szkript újbóli futtatása parancsot kell használni.

Sor/kiválasztás futtatása: Az aktuális sor vagy több kijelölt sor végrehajtása az eszköztár vagy a Run menü, Run selection or current linefuttatása parancsával, valamint az F9 billentyűvel. Az aktuális sor futtatása után a kurzor automatikusan a következő sorra lép, így soronként lépkedhetünk a kódon. A Run file-tól eltérően a végrehajtott sorok az IPython konzolon jelennek meg.

Keresés és csere: Az aktuális fájlban lévő szöveg kereséséhez használható a Search ‣ Find text or Ctrl-F / Cmd-F parancs, a szöveg cseréjéhez pedig a Search ‣ Replace text vagy Ctrl-R / Cmd-R parancsot. A keresési karakterlánc beírása a Editor alatti eredménypanelen kiemeli az egyes találatokat, és összeszámolja azokat. A találatok között a keresés/helyettesítés panelen található Find Previous és Find Next gombokkal, a Search menü megfelelő bejegyzéseivel, vagy az F2 és F3 billentyűkkel navigálhatunk.

Mindezek mellett fontos funkció a Debugger. A Spyderen belüli hibakeresést megkönnyíti az IPython konzolon belül a továbbfejlesztett ipdb hibakereső Python csomaggal való integráció[[5]](#footnote-5). Ezzel az integrációval közvetlenül a Spyder felhasználói felületéről tudjuk a breakpointok éa a végrehajtési lépések figyelését megvalósítani. A Debugger futtatását teljes mértékben vezérelhetjük a Debug menüből, a Debug eszköztárból és a konfigurálható billentyűparancsikonok segítségével, a standard ipdb konzolparancsok mellett. A futtatás közben a szerkesztő egy nyíllal jelzi, hogy a debugger éppen melyik kódsoron áll meg.

A Matplotlib ábrák generálása teljes mértékben támogatott a hibakereső aktív működése közben, beleértve az összes különböző grafikus háttértárat. Használhatjuk a %matplotlib hívást, hogy interaktív háttértárra váltson (pl. %matplotlib qt5), hogy a grafikonokat egy külön ablakban vizsgálja át, nagyítsa és állítsa be, vagy váltson vissza az alapértelmezett inline-ra (%matplotlib inline), ahhoz, hogy közvetlenül a Plots ablaktáblában jelenítse meg őket. Ha a hibakeresés során nem szeretném megjeleníteni a plotokat, ki tudom kapcoslni a Preferences ‣ IPython console ‣ Debugger ‣ Preferences ‣ IPython console ‣ Debugger alatt található "Process execute events while debugging" opciót.

A következő hasznos funkció, amely miatt felhasználóbarátnak a Spyder, az IPythone Console. Az IPython Console lehetővé teszi parancsok végrehajtását és az adatokkal való interakciót az IPython interpreteren belül. Egy új IPython konzol indításához ki kell választani a New console (default settings) menüpontját, vagy a Ctrl-T (macOS-en Cmd-T) billentyűparanccsal lehet megoldani.

Ugyanebből a menüből leállítható az éppen futó kód a Interrupt kernel parancsával, törölhető a konzol névtere az Remove all variables paranccsal, vagy újraindíthatunk egyet az Restart kernel parancsával. Mivel minden konzol különálló eljárásban kerül végrehajtásra, ez nem befolyásolja a többi megnyitott konzolt, és így könnyedén tesztelhető a kód tiszta környezetben, anélkül, hogy megszakítanánk az elsődleges munkamenetet. Bármely IPython Console, akár külső, akár a Spyder által indított, támogatja az automatikus kódkiegészítést, a valós idejű függvényhívási tippeket, illetve teljes GUI integrációt a Spyder Debuggerrel.

A következő hasznos IPython Console funkció a ’Reload changed modules’. Amikor interaktív munkamenetben dolgozunk, a Python csak egyszer tölt be egy modult a forrásfájlból, az első importáláskor. A Spyder User Module Reloader (UMR) automatikusan újratölti a modulokat közvetlenül a meglévő IPython konzolokon, amikor azok egy módosítás után újra importálásra kerülnek. Az UMR engedélyezésével a kernel újraindítása nélkül tudjuk tesztelni a kódban bekövetkezett változásokat. Az UMR alapértelmezés szerint be van kapcsolva, és aktiválásakor egy piros Reloaded modules: üzenetet jelenít meg a konzolon, amely felsorolja az általa frissített fájlokat.

Szintén lényeges funkcionalitás a Plots, amelyen az elemzéseket és futási eredményeket tekinthetjük meg ábrák formájában. Ez a munkamenet során létrehozott statikus ábrákat és képeket mutatja. Megjelennek rajta az IPython konzolból származó, a szerkesztőben lévő kód által létrehozott vagy a Variable Explorer által generált ábrák. A Plots ablakban megjelenített ábrák az aktuálisan aktív Console laphoz tartoznak. Ha konzolt váltunk, a megjelenített ábrák listája ennek megfelelően változik. Az ablak jobb felső sarkában található opciók menüben a ’Mute inline plotting’ opció alapértelmezés szerint be van kapcsolva, ami megakadályozza, hogy a plot-jaink megjelenjenek a konzolon. Ha kikapcsoljuk ezt az opciót, az ábrák mind a Plots ablaktáblában, mind a Konzolban megjelennek. A szintén alapértelmezés szerint bekapcsolt Fit plots to Window (Illeszkedés az ablakhoz) opció az ábrákat az ablaktáblának megfelelő méretűre méretezi. Ha kikapcsoljuk, akkor az ábrák a natív méretükben jelennek meg, és lehetővé teszi, hogy a panel tetején található zoom gombok segítségével manuálisan méretezzük őket. Az ablak tetején található eszköztár funkciói lehetővé teszik az ábrákkal való interakciót. Például az előre és a hátra nyilakkal egymás után végigböngészhetjük a diagramok listáját.

Az utolsó általam gyakran használt funkció a Variable Explorer, ami kiválóan használható a program futása közben létrejött változók nevétnek, típusának, méretének és értékének követésére egy táblázatban. A Variable Explorer lehetővé teszi, hogy interaktívan böngészhessünk és kezelhessünk a kód által létrehozott objektumokat. Megjeleníti az aktuálisan kiválasztott IPython Console munkamenet névtér tartalmát (beleértve az összes globális objektumot, változót, osztálypéldányt és egyebeket), és lehetővé teszi, hogy hozzáadjuk, eltávolítsuk és szerkesszük az értékeiket.

A Spyder Variable Explorer beépített támogatást nyújt listák, karakterláncok, szótárak, NumPy tömbök, Pandas DataFrame-ek, sorozatok és egyebek szerkesztéséhez, valamint képes egyetlen kattintással ábrázolni és megjeleníteni őket. Ha egy string változó hosszabb negyven karakternél, akkor dupla kattintással megjeleníthetjük az értékét egy szövegszerkesztőben, hogy könnyebben módosíthassuk azt. A listák esetében a fő Variable Explorer az első tíz érték előnézetét jeleníti meg. A listákhoz hasonlóan a Numpy tömbök esetében is a Variable Explorer megjeleníti az értékek előnézetét. Ha duplán kattintunk rájuk, megnyílik egy olyan ablak, amely a tömb értékeit egy "hőtérképen" jeleníti meg, ahol minden egyes értéket kiszínez egy rácscellában a numerikus mennyisége alapján. A DataFrame-ek a Numpy tömbökhöz hasonlóan egy kijelzőn jelennek meg, ahol megjeleníthetjük vagy elrejthetjük a "hőtérkép" színeit, megváltoztathatjuk a formátumot, valamint manuálisan vagy automatikusan átméretezhetjük a sorokat és oszlopokat. Ezenkívül a Spyder 4-től kezdődően a Variable Explorer DataFrame-ellenőrzője MultiIndex-támogatással rendelkezik, ide tartoznak a többszintű és többdimenziós indexek is.

Technikai részletek

A Spyder használata közben döntenem kellett, hogy amikor telepítenem kellett függőségeket a fejlesztőkörnyezetbe a statisztikai és a gépi tanulásos elemzésekhez, hogyan telepítsem a különböző csomagokat. Az Anaconda Navigator grafikus kezelőfelületén is nyílt volna erre lehetőség, viszont hosszútávon, ahogy haladtam előre a szoftver fejlesztésével, praktikusabbnak láttam, ha a pip parancssoros Python package manager-t használom új függőségek hozzáadására. Ezeket a parancsokat az IPython konzolon könnyebb volt közvetlenül kiadni, mint minden ilyen alkalommal újra megnyitni az Anaconda Navigatort és ott keresgélni. Egy ideig még vegyesen használtam a pip-et a conda managerrel, viszont a fenti ok miatt a hibakeresések során is egyszerűbb volt a pip-et használni a package-ek törlésére, frissítésére és újak telepítésére.

Szövegkorpuszok összegyűjtése és statisztikai elemzése

A szövegek kiválasztása

A statisztikai és gépi tanulásos vizsgálataim elvégzéséhez a Harry Potter könyvsorozat első kötetének, a Harry Potter és a Bölcsek Köve [36] című regénynek szövegét választottam ki. Ez már kellően nagy szövegkorpusznak bizonyult a releváns feldolgozások számára. A könyvsorozatot már több, mint 70 különböző nyelvre lefordították[[6]](#footnote-6). Az elemzésekhez kiválasztottam a kötet angol, német, francia és spanyol nyelvű, interneten elérhető változatait pdf formátumban.

A statisztikai vizsgálatok célja

A négy különböző nyelvű szöveg statisztikai elemzésének az volt a célja, hogy a könyvben előforduló szókincs osztályozása és szűrési lehetőségei, illetve a szavak gyakoriság szerinti összegyűjtése segítse a nyelvtanulást. Persze attól függően, milyen témában, nehézségben, illetve mélységben szeretne valaki nyelvet tanulni, szükséges szöveget választani. A dolgozat példáján keresztül ez olyan módon valósuhat meg, hogy egy haladó szinten álló tanuló a kívánt szöveget (pl. egy hosszabb regényt) inputként megadva eredményül kapja a szöveg mondatainak, sorainak, szavainak számát. Ezen információkból saját szószedetet, szótárat alkothat, illetve összehasonlíthatja, hogy különböző nyelveknek melyek a legfontosabb kifejezései a különféle szövegekben. A szöveg leggyakoribb és leghosszabb szavai is szolgálhatják ugyanezt a célt.

Tehát, a következő statisztikai vizsgálatokat végeztem el mind a 4 nyelvű változatra: a szövegek sorainak a száma, a szavak száma, a szavak hosszának előfordulása, az első 100 szó listázása gyakoriság szerint, illetve ugyanezek a vizsgálatok fejezetenként. Ezekből az információkból több következtetést is levonhatunk a könyvre vonatkozóan:

A szöveg sorainak és szavainak száma segít felmérni a könyv hosszát és általános terjedelmét. A mondatok számából kiderülhet, hogy inkább jobban tagoltak, vagy összetettek a mondatok. Az egyes szavak hosszának előfordulása meghatározhatja a szókincs gazdagságát és a ritkábban előforduló kifejezések elárulhatják, hogy mennyire vannak jelen szakterület-specifikus vagy nehéz szavak a szövegben. Így jobban meghatározhatjuk egy szöveg értelmezhetőségét. A szavak eloszlásának fejezetenkénti változása információt nyújt a fejezetek különböző témáiról vagy fókuszairól.

Ha ezek az információk több különböző nyelvű változat esetén is rendelkezésre állnak, az további előnyökkel jár a nyelvtanulás szempontjából. Például a különböző nyelvű változatok segíthetnek a szóhasználat és a kifejezések összehasonlításában, ami elősegítheti a szókincs bővítését és hatékonyabbá teheti a nyelvtanulást. A különböző nyelveken írt változatok összehasonlítása lehetővé teszi a nyelvtani szerkezetek és kifejezések jobb megértését, ami segíthet a nyelvtanulóknak, hogy jobban megértsék egy adott nyelv sajátosságait. Végül a különböző nyelvű változatok megismerése által a nyelvtanulók bepillantást nyerhetnek a különböző kultúrákba és nyelvi kontextusokba is, amelyek esetleg nem lennének olyan szembetűnőek, ha szeparáltan vizsgálnánk a különböző nyelvű változatokat.

A választott szövegek előfeldolgozása

A szövegkorpusz meghatározása után megkezdődhetett a szöveg elemezhető formába alakítása (normalizálás). Először is a begyűjtött pdf-eket txt formátumba konvertáltam[[7]](#footnote-7). Ezután a kapott txt fájlokat megvizsgálva arra a következtetésre jutottam, hogy a kapott négy különböző nyelvű szövegnek korántsem egységes a tagolása, így az általános előfeldolgozási lépések mellett speciális, csak egy-egy adott dokumentumra alkalmazható formázásra is szükség lesz. Az előfeldolgozást először még egyszerű Python kóddal, illetve a beépített ’re’ modul segítségével kezdtem el, amely reguláris kifejezésekkel való operációkat, szöveg helyettesítéseket tartalmaz.

* + 1. Az angol verzió

Az angol szövegben sok helyen előfordult, hogy bizonyos sorok végén elválasztásra kerültek bizonyos szavak. Ezeket a szövegrészeket egy reguláris kifejezés segítségével helyettesítettem, amely a szóvégi elválasztásjeleket kitörli, és az aktuális sor végén és a következő sor (vagy azutáni sor, ha van még üres sor véletlenül a két sor között) elején szereplő szórészeket egyesíti. Azt a kivételes esetet, hogy vannak-e olyan elválasztásjelet tartalmazó, és pont egy sor végén elválasztott, de valójában egyetlen összetett szónak számító szavak, amelyeket nem szabad az elválasztásjel kitörlésével egyesíteni, kézzel, Visual Studio Code segítségével néztem végig, és manuálisan egyesítettem ezeket a kötőjel megtartásával.

A következő lépés az eredeti pdf-ből áthozott, szövegben maradt felesleges, oldalszámokaA képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírást is tartalmazó header

*ábra – felesleges header szöveg és oldalszám*

*A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás*eltávolítása volt, amilyen az 1. ábrán is látható. Előbbieket, illetve a szövegben maradt felesleges gondolatjelejeket (x. ábra) is reguláris helyettesítő kifejezések alkalmazásával távolítottam el.

*ábra – felesleges gondolatjelek*

Ezután a dokumentumban maradt felesleges üres sorok kiszedése következett (*remove\_empty\_lines.py*). A szöveg sorainak szétválasztása után kiszűrtem az üres sorokat, majd újra összeillesztettem a szöveget az újsor karakterek mentén. Az utolsó előkészítési lépés a könyv egyik szereplőjének, Hagridnak a tájszólására irányult. A karakter kiejtését más alakkal jelölte a szerző, ezzel utalva a kiejtésbeli különbségekre (you – yeh, your – yer, to – ter). A tájszólásos alakok manuális kikeresgélése és eltávolítása után készen állt a szöveg a további normalizálási lépésekre, és a vizsgálatokra.

Fontos lépés az előfeldolgozásban a szükségtelen, speciális karakterek eltávolítása a szavak elemzéséhez. Erre két megoldást szállítok a szoftveremben. Az első egy blacklist típusú szűrés (*remove\_special\_characters.py*), amely csak a speciális karaktereket szűri ki a szövegből, de az olyan betűket, amelyek csak bizonyos nyelvekben fordulnak elő (pl. ä, ç, è, ê, ñ, í) benne hagyja a szövegben. A másik megközelítés a whitelist alapú szűrés (*remove\_characters.py*). Itt minden más karaktert kiszűrök a szövegből, amely nem valamilyen betű az ASCII[[8]](#footnote-8) készletből, tehát minden olyan betűt, amely nem az angol ábécé betűje.

A statisztikai elemzéseket mind a négy nyelvű szövegváltozatra elvégeztem, ezért ezesetben a blacklist módszert alkalmaztam, mert meg kellett őriznem az angoltól eltérő változatok speciális betűit is, és csak a szükségtelen karakterek, írásjelek eltávolítására volt szükségem.

* + 1. A német verzió

A német verziót összehasonlítva a többivel, már gondoltam a fejezetekre bontás majdani szükségességére, és mivel a német szövegben hiányzott a ’fejezet’ szó az egyes részek elejéről, ezért ezeket kézzel pótoltam a kiindulási szövegben (KAPITEL szavakkal). Így a többi nyelvű szöveggel megegyezővé vált a struktúra. Ez hozzájárult ahhoz, hogy a kiindulási szövegkorpusz amennyire csak lehet, egységes legyen.

A német szövegben az eredeti pdf-ben minden oldalon előfordult footer szövegben a könyv címe és az oldalszám. Ezeket hasonlóan az angol verzióhoz, reguláris kifejezéssel vettem ki. Az üres sorok és a szükségetelen karakterek eltávolítása (pl a párbeszédeket jelző » nyilak is) után ez a szöveg is készen állt az elemzésre.

* + 1. A francia verzió

A francia változatban felmerült az a probléma, hogy a — (EM DASH nevű unicode) karakter megjelent az összetett, kötőjellel elválasztott szavak között kötőjelként (viszont előfordult a normál kötőjel karakter is ugyanerre a célra), illetve a párbeszéd kezdetének a jele is ez volt a sorok elején. Ezeket egységesítettem, tehát az EM DASH unicode karaktereket lecseréltem normál kötőjelekre (- mínusz karakter, vagyis HYPHEN-MINUS unicode-ban). Végül az üres sorok és a szükségetelen karakterektől való megszabadulás (blacklist) után ez a szöveg is normalizálásra került.

A francia szöveghez kapcsolódóan érdemes megjegyezni, hogy a francia nyelvben szintén előfordulnak az angolhoz hasonló rövidített, aposztrófos alakok. Úgy döntöttem, hogy jelen dolgozat keretei között figyelmen kívül hagyom ezeket a szóalakokat, mivel egyelőre csak R programozási nyelvre találtam volna használható csomagot ennek a problémakörnek a kezelésére.

* + 1. A spanyol verzió

A spanyol szövegben némi kézi korrekcióval kellett kezdenem a szöveg normalizálását. Néhány helyen ugyanis a pdf -> txt konverzió széttördelte a fejezetek elejét, ez szerencsére csak néhány helyen fodrult elő. Ezeket még időben észrevettem és korrigáltam. A spanyol változatban a párbeszédek elejét és végét is a — karakter (EM DASH) jelezte. Mivel a kötőjellel elválasztott összetett szavakban is megjelent ez a karakter, így ezeket kiszűrtem a szövegből. Ezután az üres sorokat és a maradék speciális karaktereket kellett még eltávolítanom, például a kérdést és a felkiáltást jelző speciális, csak a spanyol nyelvben előforduló karaktereket is (¿¡). Mindezek után mind a négy teljes szöveg elkészült a statisztikai vizsgálatok elvégzéséhez.

Statisztikai vizsgálatok

* + 1. A mondatok elemzése

Az 5.2. fejezetben részletezett előfeldolgozás után a megtisztított, normalizált szövegek készen álltak a statisztikai elemzésekre.

Az első érdekesség akkor adódott, amikor a szövegekben szereplő mondatok számát próbáltam meghatározni az ehhez létrehozott *get\_sentences* modullal. A modul egy NLTK-ból importált tokenizáló függvényt használ fel, és a szövegben szereplő mondatok listáját kapjuk meg vele[[9]](#footnote-9). Viszont az angol nyelvű, eredeti szövegben, amelyben az unicode APOSTROPHE (’) típusú idézőjelek helyett RIGHT SINGLE QUOTATION MARK (’) típusú karakterek szerepelnek idézőjelként a párbeszédek elején és végén, az NLTK modul *sent\_tokenize* metódusa számos esetben nem ismeri fel a mondathatárokat az ilyen jelek után, és egyben maradnak több mondatból álló szekvenciák. Ez jelentősen torzította az eredményeket, ugyanis az eredti szöveg mondatainak számára így 4931 adótott. Viszont az ilyen típusú idézőjelek normál aposztróf jelre történő kicserélése után a megfelelő szétválasztással az angol mondatok száma 6436 lett.

Hasonló probléma adódott a német szöveg esetében is, ahol a párbeszédek elején és végén megtalálható ’»’ és ’«’ jelek okoztak hasonló problémát. Amennyiben ezek benne maradnak a szövegben, a mondaotk számára 5073 jön ki, ami még szintén gyanúsan kevés a többi nyelvű szöveg mondatainak számához képest. Miután eltüntettem a fenti karaktereket, 6208 lett az eredmény a német szövegre, ami lényegesen közelebb állhat a valósághoz. Az eddigi tapasztalatokból kiindulva a francia és a spanyol szövegekben is elkezdtem keresni ilyen lehetséges mondatalkotási anomáliákat, heurisztikus módszerrel. Találtam is példát ilyen problémára a francia szövegben (x. ábra). A párbeszédeknél, ha a sor végén nem egy egyszerű írásjel van, hanem három pont, akkor ezt nem ismeri fel a *sent\_tokenize* metódus. A fenti példákból látszik, hogy nem feltétlenül egyszerű kérdés egy nagy méretű szövegkorpuszból a mondatok kinyerése. Azt gondolhatnánk, hogy esetünkben *A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás*ugyanannak a regénynek négy különböző nyelvű verziójánál a mondatok száma meg kell

*ábra*

egyezzen. Valószínűleg ez így is van, és mindössze az NLTK metódus nem működik megfelelően erre a regény típusú szövegkorpuszra. Azonban az is lehetséges, hogy a különféle fordításokban valamiért mégis adódhat némi eltérés a mondatok száma között. Ennek kiderítése további kutatásokat igényelne, jelen dolgozat keretei között nem tértem ki rá. A modatok száma az egyes szövegekben a lenti táblázatban szerepel. Látható, hogy a francia éa a spanyol változat számai majdnem megegyeznek, az angol szöveg száma már több, mint 300-zal kevesebb az előbbi kettőnél, a német könyv adatánál pedig már több, mint 500 mondat a differencia negatív irányban. Ezekből az adatokból megállapítható, hogy a német és az angol szövegben még bőven maradhattak mondatokká szét nem választott szövegrészek, illetve az is lehetséges, hogy a francia és a spanyol változatokban bizonyos mondatokat még több részre bontott az algoritmus.

|  |  |
| --- | --- |
| angol | 6436 |
| német | 6208 |
| francia | 6756 |
| spanyol | 6749 |

A mondatok számából arra a következtetésre jutottam, hogy a vizsgálat metódusának további finomítása, illetve a szövegek ’tisztább’ formában történő beszerzése lenne szükséges annak eldöntéséhez, hogy mennyire feleltethetők meg egymásnak a négy szöveg mondatai és felépítése (az előfeltételezésem szerint sok különbségnek nem kellene előjönnie, mivel csak germán és újlatin nyelveket érint a vizsgált szövegkorpusz).

* + 1. A szavakra vonatkozó statisztikák

Ha megvizsgáljuk a négy vizsgált nyelv szövegét (1. melléklet), a szavak számának alakulásából azt a következtetést vonhatjuk le, hogy az angol és a német változat szavainak száma majdnem teljesen megegyezik, 80 ezer szó környékén van mindkét mennyiség. A legkevesebb szó a spanyol szövegben található (77774 szó), ami azt sejteti, hogy a spanyol a legkevésbé terjengős a négy vizsgált nyelv közül. A francia nyelvű szövegre éppen az ellenkezője lehet igaz, ugyanis több mint 83 ezer szóból áll ez a szöveg, amivel kiemelkedik a többi közül.

A sorok számát természetesen befolyásolja, hogy milyen tördelésben szerepeltek a szövegek az eredeti pdf dokumentumokban. Ha megvizsgáljuk a statisztikai eredményeket, kiderül, hogy átlagosan az angol szövegben a legrövidebbek a sorok, mert 8278 ezer sorból áll a szövege. A német verzióban is hasonló hosszúságú egy-egy sor, ezt a 7887 sor bizonyítja. A francia és a spanyol nyelvű könyvek sorai az eredmények alapján hosszabbak (6446 és 6134 sor).

Ha a szavak hosszát és ezek eloszlását vizsgáljuk, látható, hogy az angol szövegben nagyjából egyenletesen csökken az egyes szóhosszúságok előfordulási gyakorisága (2. melléklet). 3 és 4 betűs szóból találunk a szövegében a legtöbbet, és a leghosszabb szavak, amelyek kötőjeles összetett szavak, 19 karakteresek. Ehhez képest a többi nyelv esetében már nem ilyen egyenletes az eloszlás. A német szövegben messze 3 betűs szóból találunk a legtöbbet, jóval több mint 20 ezer darabot. Itt a második leggyakoribb szóhosszúság az 5 karakternyi, csak ezt követik a 4 és a 6 karakteres szavak. A német verzióban meglehetősen kevés a 2 karakter hosszú szó, kb. 7500 db. Ehhez képest az angolban csaknem kétszer ennyi kétbetűs kifejezés szerepel, az újlatin szövegekben pedig jóval több, mint kétszerese az ilyen rövid szavak száma a németének. A német, francia és spanyol változatokban is előfordulnak extrém hosszú kifejezések, amelyek majdnem mindegyike kötőjeles összetett szó, és 1-1 alkalommal fordulnak elő a szövegekben. Ezeke legtöbbje a Harry Potter könyvre specifikus (pl. az ’Ő, akit nem nevezünk nevén’ kifejezés kötőjelekkel elválasztva). Ezekből az eredményekből azt a következtetést vonhatjuk le, hogy nagyobb terjedelmű szövegkorpusz vizsgálata esetén világosan kirajzolódik az egyes nyelvek néhány nyelvtani sajátossága.

A következő, szempontunkból érdekes statisztika a szövegekben leggyakrabban szereplő szavak listája. A 6-9. mellékeletek mutatják az egyes nyelveken előforduló leggyakoribb szavak listáját. A végleges adatok kinyerése előtt még szükség volt egy fontos NLP lépésre, a gyakran előforduló, viszont a vizsgálataink szempontjából kevésbé fontos, vagy jelentéktelen kötőszavak, névmások és hasnonló szavak (stopwords) kiszűrésére (a, the, and, me, her, stb). Ezek torzítják a szógyakorisági vizsgálatokat, és elveszik a figyelmet a valóban lényeges kifejezésekről a szövegben. A stopword listát szabadon megválaszthatjuk, tetszőlegesen elvehetünk belőle általunk fontosnak gondolt szavakat, és hozzá is adhatunk a listához bármilyen kifejezést. Az elemzéseimben az NLTK beépített stopword listáit használtam, amely mind a négy vizsgált nyelvre rendelkezésre áll.

A szógyakoriságokból az első fontos kiolvasható eredmény a regényben szereplő karakterek előfordulási gyakorisága. A főszereplő természetesen az adatok alapján is Harry, őt követik a hozzá közel álló karakterek, Ron, Hagrid és Hermione. Látható, hogy egy ilyen statisztikából kideríthető, hogy általánosan kik a legfőbb szereplők egy regényben, vagy egy másfajta hosszabb elbeszélő szövegben.

Konklúzióként megállapítható, hogy az NLP módszerek segítségével alkotott statisztikák segítségével elérhetők a fejezet elején kitűzött nyelvtanulási célok. Ezek tehát a támogatás saját szószedetek, szótárak készítéséhez. Továbbá a különféle nyelvek legfontosabb kifejezéseinek összegyűjtése, az adott alkotás általános terjedelme, a mondatok tagoltsága, összetettsége, a szókincs gazdagsága, a szakkifejezések és nehezebb szavak elkülönítése a ritkább gyakoriság alapján.

A könyv szövegének tartalmi összefoglalója

A tartalmi vizsgálat célja

A következő vizsgálatom egy érdekes alkalmazási területe a gépi tanulásos módszereknek. Célként a Harry Potter és a Bölcsek Köve angol nyelvű szövegének tartalmi összefoglalójának elkészítését tűztem ki. Az eredmények segítségével betekinthetünk egy-egy hosszabb szöveg, hagyományos módszerekkel csak hosszú idő alatt elkészíthető, kulcsmozzanatokat és cselekményeket tartalmazó kivonatába. Ennek felhasználási területe lehet például egy olyan kutatási témában való elmélyedés felgyorsítása, amelynek szakirodalmát akár csak hosszú hónapok vagy évek olvasásával tudnánk teljeskörűen feldolgozni. Ilyen esetben az összegyűjtött szövegkorpusz gépi tanulásos kivonatolásával lehetőség nyílik a probléma megoldására.

A *summarization.py* scriptben történik meg az előkészített szöveg összefoglalójának generálása. Az angol szöveg normalizálása ezesetben némileg más lépésekből állt: whitelist típusú karakterszűrés, a szöveg kisbetűssé tétele, a szó eleji és szóvégi üres szóközök eltávolítása, a szöveg tokenizálása vagyis szavakra bontása NLTK segítségével, a haszontalan szavak (stopwords) eltávolítása, és a megszűrt szavak visszaalakítása összefüggő szöveggé (*normalize\_document\_for\_summarization.py*).

Ezután a numpy segítségével vektorizált szöveg mondatait is normalizáltam. Az így eredményül kapott mondatokat a TF-IDF modell segítségével vektorizáltam. A TF-IDF modell úgy működik, hogy két mérőszám kombinációját, a kifejezés gyakoriságát (TF) és az inverz dokumentumfrekvenciát (IDF) rendeli az egyes szavakhoz. Ezt a technikát eredetileg a felhasználói lekérdezések alapján a keresőmotorok találatainak rangsorolására szolgáló metrikaként fejlesztették ki, és mára a keresőmotorok az információk és a szövegjellemzők kinyerésének elterjedt eszközévé vált [14, 212.o].

Az eredményül kapott mátrix alkalmassá válik arra, hogy segítségével összeállítsuk a könyv kivonatát, összefoglalóját. Az erre a célra szolgáló módszer, amelyet kipróbáltam, a látens szemantikai analízis nevet viseli (LSA). A módszer lényege, hogy bármely dokumentumban létezik egy látens struktúra a kifejezések között, amelyek kontextuálisan kapcsolódnak egymáshoz, és ezért ugyanabban a szinguláris térben is korrelálniuk kell egymással [14, 442.o]. Ennek két változatát próbáltam ki a minél pontosabb összefoglaló elkészítése érdekében.

LSI-SVD

* + 1. Az LSI-SVD elméleti háttere

Az első módszer a ’low-rank singular value decomposition’ (LSI SVD). Az erre szolgáló algoritmus a következőképpen működik:

Előkészületként meg kell választanunk az összefoglaló kívánt mondatainak számát. Ezután szükség vank a témák számának meghatározására (k). Az algoritmus menete a következő:

1. A mondatvektorok kinyerése a V mátrixból (k sor).

2. Az első k szinguláris érték kinyerése az S-ből.

3. Küszöbérték-alapú megközelítéssel eltávolítjuk az olyan szinguláris értékeket, amelyek a legnagyobb szinguláris érték felénél kisebbek, ha vannak ilyenek. Ez egy heurisztika, és lehetőség van ezzel az értékkel ’játszani’ az elemzés finomhangolásához.

4. Szorozzuk meg a V négyzetből minden kifejezés mondatoszlopát megszorozzuk az S-ből hozzájuk tartozó szinguláris értékkel, és minden szorzatból gyököt vonunk, így megkapjuk témánként a mondatok súlyát.

5. Kiszámítjuk a mondatsúlyok összegét a témák között, és a végső pontszám négyzetgyökét vesszük, így megkapjuk az egyes témakörökre vonatkozó fontossági pontszámokat.

6. Miután megkaptuk a pontszámokat, csökkenő sorrendbe rendezzük ezeket, és kiválasztjuk az n legmagasabb pontszámú mondatot, majd ezek sorrendje alapján megalkotjuk a végső összefoglalónkat.

* + 1. LSI-SVD a gyakorlatban

Először a teljes könyv tatalmának összefoglalóját készítettem el. Az algoritmust leteszteltem úgy, hogy változtattam a témakörök számát. Azt szerettem volna megtudni ezzel a heurisztikus tesztelési módszerrel, hogy mennyi lehet az optimális témakörök száma, melyik hozza ki a legelfogadhatóbb összefoglaló szöveget. A mondatok számát 500-nak választottam meg, hogy kellően nagy merítés álljon a program rendelkezésére az összegző szöveg megalkotásához, tehát hogy ne torzítsák túlzottan az eredményeket a túl gyakran előforduló kifejezések (pl. a ’said’ szó a párbeszédekben).

Összehasonlítottam az eredményül kapott 500 mondatot különböző témakör számokkal (*summarization\_LSI\_SVD\_1\_topic,* stb.). 1, 2, 3, 4, 5, 6, és 10 témakör beállításával próbálkoztam. Az összehasonlítást a könyv utolsó fejezete alapján végeztem el a teljes szöveg alapján történő generálás esetén is elsősorban a szöveg legvégét vizsgáltam. Tehát összevetettem az egyes témakörszámok szerinti generált szövegeket egymással, és különbségeket, tendenciákat kerestem. Az eredmények vizsgálatából számomra az az összefüggés derült ki, hogy minél nagyobbra állítottam a témakörök számát, annál több kulcsmozzanat és fontos mondat került bele a kivonat szövegébe. Szerintem valóban logikus, hogy egy viszonylag hosszú regény esetében sok témakört, illetve koncepciót tudunk elkülöníteni, és ennek megfelelően jobban teljesít az algoritmus a különböző cselekménybeli fontos részletek felszínre hozásában. 6 és 10 témakör esetén találtam a legszínvonalasabbnak az összefoglaló szöveget, ezeknél éreztem azt leginkább, hogy rálátást kapok a regény legfontosabb történéseire.

A vizsgálatot megismételtem az angol nyelvű könyv egyes fejezeteire is külön-külön. Itt szintén az utolsó fejezetek összevetése alapján (más fejezeteket is átfutottam szúrópróbaszerűen) arra jutottam, hogy a témakörök számát itt érdemes limitálni a fejezetek rövidebb mivolta miatt. 1, 2 és 3 témakörrel adta a legrészletesebb eredményt az algoritmus. Szerintem az 1 vagy 2 témakörrel készített összefoglalók szebben kiemelték az egyes fejezetek mondanivalóját és cselekményébe is több betekintést engednek, mint a teljes szöveg kivonata, amit előzetesen el is lehetett várni ettől a vizsgálattól. Tehát általában hosszabb és a cselekmény szempontjából lényegesebb mondatokat kaptunk ereményül.

TextRank

* + 1. A TextRank elméleti háttere

A másik LSA módszer, amelyet alkalmaztam a szövegkorpuszra, a TextRank. A TextRank összefoglaló algoritmus a magjában a Google által webhelyek és oldalak rangsorolására használt népszerű PageRank algoritmust használja. A Google keresőmotorja ezt használja, amikor a kereséseink alapján releváns weboldalakat ad találatként. Tehát a PageRank alapalgoritmusa egy gráf-alapú pontozás vagy rangsorolás, amelyben az oldalakat pontszámozása vagy rangsorolása a fontosságuk alapján történik. A weboldalak további linkeket tartalmaznak, amelyek további, akár még több linkkel rendelkező oldalakra mutatnak, és ez a felépítés az egész interneten folytatódik. Ezt a struktúrát ábrázolni tudjuk egy gráf-alapú modellben, ahol a csúcsok a weboldalakat, az élek pedig a köztük lévő linkeket jelölik. Ezt fel lehet használni egy szavazási vagy ajánlási rendszer kialakítására, így, ha a gráfban az egyik csúcsról a másikra mutató linket találunk, az gyakorlatilag egy szavazat leadását jelenti. A csúcsok fontossága nem csak a szavazatok vagy élek száma alapján dől el, hanem a saját és a hozzá kapcsolódó csúcsok fontossága alapján is. Ez segít árnyaltabban meghatározni az egyes pontok pontszámát vagy rangsorát [14, 445.o].

Az algoritmus képletében a Vi csúcsokra léteznek PR(Vi) értékek, amely a PageRank pontszámot jelzik. A (Vi) azon oldalak halmazát jelöli, amelyek erre a csúcsra/oldalra mutatnak, Out(Vi) pedig azokat az oldalakat, melyekre a Vi csúcs/oldal mutat. A d pedig a csillapítási tényező, és általában 0 és 1 közötti értéket vesz fel (ideális esetben 0,85-re van beállítva).

Tehát, amikor egy szöveges dokumentumot összegezünk, az algoritmusban szereplő csúcsokat mondatok, kulcsszavak vagy kifejezések alkotják majd, az alapján, hogy milyen összegzést szeretnénk elvégezni. Lehet, hogy a fenti csomópontok között több kapcsolat is létezik. A TextRank annyiban tér el az eredeti PageRank algoritmushoz képest, hogy van egy súlykoefficiens (pl. wij) a két csúcsot összekötő él (Vi és Vj) Ez a súly jelezi a közöttük lévő kapcsolat erősségét. Ez alapján a kibővítés alapján kapjuk meg a TR-t, ami egy csúcs súlyozott PageRank pontszámát jelöli.

A TextRank algoritmus lépései a következők:

1. Tokenizáljuk és kinyerjük a mondatokat az összefoglalni kívánt dokumentumból.

2. Eldöntjük, hogy hány mondatot (k) szeretnénk szerepeltetni az összefoglalóban.

3. Építünk egy dokumentum-kifejezés jellemzőmátrixot például a TF-IDF vagy a Bag of Words (szózsák).

4. Kiszámítjuk a dokumentum-hasonlósági mátrixot úgy, hogy a mátrixot a transzponáltjával szorozzuk.

5. Felhasználjuk ezeket a dokumentumokat (esetünkben mondatokat) csúcsokként, illetve az egyes dokumentumpárok közötti hasonlóságokat súlyként vagy a korábban említett pontszám-együtthatótként, és átadjuk ezeket a PageRank algoritmusnak.

6. Megkapjuk az egyes mondatok pontszámát.

7. Rangsoroljuk a mondatokat a kapott pontszámok alapján, és visszaadjuk a legmagasabb pontszámokat elérő mondatokat.

* + 1. TextRank a gyakorlatban

Mint láttuk, a TextRank alkalmazásához a szokásos szövegelőfeldolgozási lépések elvégzése mellett csak az összefoglalóban szerepeltetendő mondatok számát kell megadnunk. A vizsgálatot elvégeztem az angol nyelvű szöveg egészére, és külön-külön az egyes fejezetekre is.

A teljes szöveg összefoglalójához 500 mondatos terjedelmet választottam, hogy össze lehessen vetni az LSI SVD algoritmussal. Ha összevetjük a két összefoglalót, már a szövegek elején feltűnő a különbség. A 10. és 11. mellékleteken a szövegnek ugyanabból a részéből láthatunk kivonatokat. A TextRank algoritmus egyértelműen koherensebb, bővebb mondatokat választott ki, mint az LSI SVD, így alkalmasabb hosszabb szövegek tömör összefoglalójának elkészítésére. Természetesen mivel még itt is csak látens szemantikai analízisről van szó, ezért ez a modell sem alkalmas arra, hogy egyértelműen megragadjon jelentésbeli, olyan tényleges összefüggéssel járó elemeket a szövegben, amelyeket már csak például egy deep learning modellel lennénk képesek elérni.

Amennyiben hasonló módon összevetjük a két használt algoritmus által az egyes fejezetekről készített összefoglalókat, akkor ugyanarra a következtetésre juthatunk, hogy a TextRank módszer, szintén 40 modatos fejezetenkénti mondathosszal, lényegretörőbb, összefüggőbb összefoglalót ad. Bár itt már kevesebb a különbség a két módszer között, és az LSI SVD által generált szöveg is használhatóbb eredményt ad. Utóbbi algoritmus hátránya egyértelműen a hosszabb, több fejezetnyi terjedelem esetében ütközik ki egyértelműen.

Témakörök szerinti elemzés

Szövegosztályozás gépi tanulással

A szövegosztályozást, tehát különböző szövegek témakörök szerinti csoportosítását angolul hívhatjuk úgy, hogy text categorization, vagy úgy is, hogy text classification. Szempontunkból azért kifejezőbb a classification kifejezés, mert egyrészt ugyanazt jelenti, mint a kategorizálás (tehát osztályozás), de azt a jelentést is magában hordozza a szó, hogy olyan felügyelt gépi tanulásos módszerrel történő vizsgálatokra fókuszálunk, amelyek osztályozást alkalmaznak a szövegek csoportokba sorolására. Ez az eljárás nemcsak a szövegekre terjed ki, hanem egy általános problémára kínál megoldást (pl. zenék, képek, videók osztályozása is ide tartozik) [14] (277. o).

* + 1. Az osztályozás formális definíciója

A szövegosztályozási rendszer lényege az, hogy szöveges dokumentumokat előre megadott osztályokba, csoportokba sorolunk be. Az adott szövegek lényeges tulajdonságai alapján sorolja be a rendszer ezeket az egyes témakörökhöz tartozó csoportokba. Ha szeretnénk matematikailag leírni a folyamatot, akkor azt mondhatjuk, hogy adott számú *D* dokumentumot szeretnénk egy, a *C* halmazban *(C = {c1, c2, c3, … , cn})*  lévő kategóriába *(Cx)* besorolni, egy szövegosztályozási rendszer segítségével (*T*). Így a hozzárendelés függvénye a következő: *T : D → Cx.*

* + 1. Az osztályozás típusai

Elméletben az osztályozás két fő típusa különböztethető meg: a tartalom alapú, és az igény alapú csoportosítás. Előbbi esetben különböző súlyokat adunk egy szövegben előforduló témáknak, és ez alapján próbáljuk osztályozni a tartalmát. Utóbbinál kifejezetten a felhasználó szabja meg, hogy egy adott dokumentumhalmazból milyen jellegű osztályozást szeretne megvalósítani. Hamar belátható, hogy a gépi tanulás segítségével automatizált szövegosztályozás hatékony segítséget tud nyújtani abban az esetben, amikor például már szövegek millióit szeretnénk rövid időn belül ketegóriákba sorolni. Az alábbiakban szeretném röviden bemutatni, hogy jelenleg milyen gépi tanulásos technikák állnak rendelkezésünkre ennek kivitelezéséhez.

A két fő technika, amelyet megkülönböztethetünk, a felügyelt és a felügyelet kélküli gépi tanulás. Ezek mellett fontos megemlíteni a megerősítéses tanulás és a részben felügyelt tanulás módszereit.

A felügyelet nélküli gépi tanulás módszerének, illetve az erre épülő algoritmusoknak az a lényege, hogy előre felcímkézett tanulási adatminták nélkül is képes a modellalkotásra.

A metódus inkább a mintakeresésre és az adatokból kinyerhető látens alstruktúrákra fókuszál a prediktív analízis helyett. Általában szöveges vagy szám formátumú adatpontokkal rendelkezünk, amelyekből kinyerjük a lényeges információkat (feature engineering). Ezután a kinyert információkra alkalmazzuk az algoritmusunkat, amely megpróbál lényeges mintázatokat találni az adatok között (klaszterezés, összefoglalás témamodellek alapján). Erre a dokumentum klaszterezésre vissza fogunk térni a tartalmi összefoglalónál.

A felügyelt gépi tanulás ezzel szemben úgy működik, hogy előre elkészített és kategorizált mintaadatokkal „tanítjuk” be a modellünket. Először szintén a lényeges tulajdonságok kinyerésével kezdjük el a folyamatot, és minden adatpont rendelkezni fog a saját tulajdonsághalmazával és az ahhoz tartozó osztállyal vagy címkével. Ezután az algoritmus többféle mintát tanul meg, amelyek a mintaadatokhoz tartozó egyes osztályokra lesznek jellemzőek. Ennek a folyamatnak a végén kapunk egy betanított modellt. Ezt alkalmazni tudjuk más, kívülről érkező teszt adatokra, és mivel a modellt már betanítottuk, az meg tudja határozin, hogy melyik osztályokba sorolhatók be az ismeretlen dokumentumok.

A felügyelt gépi tanulás esetében az osztályozás mellett (amikor a kapott eredmény egy kategória vagy osztály) beszélhetünk regresszióról is, amelynek eredménye egy folytonos numerikus változó (pl. hőmérséklet, tőzsdei árfolyamok). Jelen vizsgálataim során természetesen osztályozást fogok végezni a vizsgált dokumentumokon.

* + 1. A felügyelt gépi tanulásos osztályozás elmélete

Ha már rendelkezünk a mintadokumentumokkal és a hozzájuk tartozó címkékkel (training set), ezeket jelölhetjük *TS*-el. Ez a következőképpen néz ki:

*TS* = {(*d*1, *c*1), (*d*2, *c*2), … , (*dn*, *cn*)}, ahol *d*1, *d*2, … ,*dn* a dokumentumok listája. Az ezekhez tartozó címkéket jelölhetjük úgy, hogy *c1, c2, … , cn*, vagyis *cx* ∈ *C* = {*c*1, *c*2, … , *cn*}. A *cx* jelöli az *x* dokumentum címkéjét, a *C* pedig az összes lehetséges kategória halmazát képviseli, minden dokumentum ezek közül kaphatja meg az egyik címkét. Miután rendelkezünk a minta dokumentum-címke párokkal, definiálhatunk egy *F* felügyelt tanulási algoritmust, amelynek a *TS*-en való alkalmazásával építhetünk egy osztályozási modellt (*γ***),** és ebből azt mondhatjuk, hogy*F*(*TS*) = *γ,* tehát az *F* algoritmusunk bemenetként megkapja a *TS* párokat, és ebből megkapjuk a *γ* modellt. Ez adja a tanulási folyamatot. Ez a modell aztán megkap egy, a rendszer számára előzetesen ismeretlen dokumentumot (*ND*), és meghatározza annak osztályát (*cND*)úgy, hogy *cND* ∈ *C*. Ezt előrejelzési folyamatnak nevezzük, és definiálhatjuk úgy, hogy *γ* : *TD* → *cND*.

Végeredményben két fő folyamatot láthatunk, a tanulási és az előrejelzési lépést. A felügyelt gépi tanulás során tehát a legfontosabb a tanulási minta megfelelő megalkotása, amely manuálisan történik, viszont utána kevés erőfeszítésre és kézi felügyeletre van csak szükségünk az új dokumentumok osztályozásához. A tanulási minta alkalmazása mellett érdemes lehet egy vaidációs adathalmazt is feldolgozni a modellünknek, amely lehetővé teszi, hogy a modell ne illeszkedjen túlzottan az eredetileg kapott adathalmazhoz („túltanulás”), és így jobban általánosítható legyen új szövegek osztályozására. A modell tovább javítható például a saját belső paramétereinek tanulási algoritmushoz igazodó állításával, vagy a mintaadatoknak egy tanulási és egy validálási csoportra való szétválasztásával. A besorolások minőségére többféle mérőszámot fogunk használni.

Háromféle szövegosztályozási típust különböztethetünk meg. A bináris osztályozás során két darab kategória létezik, és minden szöveget besorolunk valamelyik kategóriába a kettő közül. A második típus az, ahol kettőnél több címke közül választ egyet az algoritmus a vizsgált dokumentumoknak. A harmadik lehetőség az, hogy szintén több osztály áll rendelkezésre, viszont egy szöveg több címkét is kaphat. A vizsgálat során fontos szem előtt tartani, hogy minél több csoportba szeretnénk a dokumentumainkat besorolni, annál nehezebb lesz pontos osztályozó modellt létrehozni.

A felhasznált adatkinyerési modellek

* + 1. A hagyományos modellek

Adatkinyerés alatt itt azt a folyamatot értem, amellyel egy szövegkorpuszból kiválogatjuk az osztályozási elemzések szempontjából lényeges szavakat (angol kifejezéssel feature engineering, amit nagyjából jellemző tervezésként/meghatározásként lehetne fordítani). Először a hagyományos, megszámlálás alapú jellemző kinyerési módszereket szeretném megemlíteni.

Ide tartozik a „Bag of Words” (szózsák) modell, amely a szöveges dokumentumokat egy olyan numerikus vektorként ábrázolja, ahol minden egyes dimenzió egy adott szó a szövegkorpuszból, és az érték lehet a dokumentumban való gyakorisága, előfordulása (1 vagy 0), vagy akár súlyozott értékek. A modell minden dokumentumot szó szerint a saját szavaiból álló halmazként reprezentál, és figyelmen kívül hagyja a szórendeket, szekvenciákat és a nyelvtant.

A „Bag of N-Grams” modell lényegében egy kiterjesztése az előbbinek. Az N-gram jelentése egy szövegből kinyert szó-tokenekből álló olyan gyűjtemény, ahol a tokenek egybefüggőek és sorrendben következnek egymás után. A bi-gramok a 2. rendű n-grammokat jelölik (két szó), a tri-gramok a 3. rendű n-grammokat (három szó) stb.

A TF-IDF (Term Frequency-Inverse Document Frequency) modell két mérőszámból tevődik össze. Az egyik a kifejezés-gyakoriság, a másik pedig az inverz dokumentum-gyakoriság (a 6. fejezetben volt már róla szó).

A hagyományos szógyakoriság alapú jellemző kinyerés hatékonyságát ellensúlyozza, hogy eredményként csak egy rendezetlen szóhalmazt kapunk, emiatt használatakor sajnos elveszítjük a szövegünk szemantikáját, struktúráját, szekvenciáját, és az egymáshoz közel eső szavak közötti kontextust.

* + 1. A fejlettebb modellek

Láthattuk, hogy a főként szógyakoriságon alapuló módszereknek vannak korlátai, amikor gépi tanulásos vizsgálatokról van szó. Rendelkezésünkre állnak jobban kidolgozott modellek, amelyek ún. beágyazásokkal, vagyis a szavak vektoros reprezentációjával érik el azt, hogy az olyan fontos információkat se veszítsünk el, mint a szöveg szemantikája, struktúrája, belső kontextusai.

A modellek, amelyeket kipróbáltam, neurális hálón alapuló prediktív nyelvi modellek. Ezek működési elve az, hogy megpróbálja előrejelezni a szavakat a szomszédos szavak alapján úgy, hogy átnézi a korpuszban található szósorozatokat. A folyamat során elosztott ábrázolások megtanulása után sűrű szóbeágyazásokat kapunk eredményül.

Az első ilyen modell a Word2Vec, amelyet a Google hozott létre 2013-ban. Ez egy prediktív deep learning alapú modell. Segítségével magas minőségű, elosztott, és folytonos sűrűségű vektoros szóábrázolásokat alkothatunk, amely képes a kontextus- és a jelentésbeli hasonlóságok megragadására. Általában megadhatjuk a szóbeágyazási vektorok méretét, és a vektorok száma lesz a szókincs mérete. Az így kapott sűrű vektortérnek a dimenzionalitása sokkal alacsonyabb lesz, mint a hagyományos szózsák modellekkel épített nagydimenziós ritka vektorterek. Itt a Continuous Bag of Words (CBOW) és a Skip-Gram architektúrákat érdemes megemlíteni.

A GloVe modell a Global Vectors rövidítése. Ez egy felügyelet nélküli gépi tanulási módszer, és a Word2Vec-hez hasonlóan sűrű szóvektorokat nyerhetünk ki vele a szövegekből. Viszont más technikai alapokon működik, és a betanítása egy összesített globális szó-szó együttes előfordulási mátrix alapján történik. Végül egy tartalmas részstruktúrákkal rendelkező vektorteret kapunk.

A harmadik vizsgált modell a FastText, amelyet a Facebook vezetett be 2016-ban, a Word2Vec modell továbbfejlesztéseként. Mikolov és társai "Enriching Word Vectors with Subword Information" című cikkén alapszik ez a metódus. A FastText alkalmazható szóreprezentációk tanulására, és robosztus, gyors és pontos szövegosztályozást tesz lehetővé. A Word2Vec-től eltérően, ahol a szavak egy-egy entitásként vannak kezelve, a FastText modell minden szót egy-egy karakter N-gram halmazként kezel.

A felhasznált osztályozási modellek

Az alábbiakban felsorolt osztályozási modellek felügyelt gépi tanulásos elven működnek, és adatpontokat osztályoznak vagy címkéznek be azon információk alapján, amelyeket addig begyűjtöttek. A felügyelt jelleg miatt minden esetben szükségünk van betanulási adatokra. Ez két részből áll: egy bemeneti adatpontból, ami általában egy jellemző vektor, és a hozzá tartozó kimenet, tehát a besorolás eredménye.

Ebben a modellezési szakaszban három fázist járnak végig az osztályozó algoritmusok: a betanulás (training), a kiértékelés (evaluation) és a hangolás (tuning) lépéseit. A betanulás alatt az algoritmus megpróbál általános mintákat képezni a megadott jellemzőkészletekből és a hozzájuk tartozó kategóriákból. Ennek a végén kapunk egy osztályozási modellt. A kiértékelés alatt egy validációs adathalmazon teszteljük a kapott modell teljesítményét, a tényleges osztálycímkékkel való összevetéssel. A hangolás, vagy másképp optimalizálás azt jelenti, hogy a modellünk hiperparamétereit állítjuk. Ezeket, a modell komplexitását, tanulási képességét meghatározó paramétereket nem lehet közvetlenül az adatokból megtanulni. Még a modell futtatása és betanítása előtt be kell őket állítani, hogy végül a modell a lehető legjobb teljesítményt, előrejelzési pontosságot adja.

Az alábbiakban csak egy rövid áttekintést adok azokról az osztályozási algoritmusokról, amelyeket a tesztelés során felhasználtam. Az egyes felügyelt gépi tanulásos modellek részletesebb elemzése önmagában is meghaladná egy dolgozat kereteit. A vizsgálatok során az egyes modelleknek a scikit-learn csomagban való megvalósításait fogom használni.

A Multinomial Naïve Bayes (a továbbiakban MNB) algoritmus kifejezetten alkalmas olyan feladatokra, amelyekben több, mint két darab osztályba kell sorolnunk szövegeket. Az algoritmus a Bayes-tételt valósítja meg, viszont van benne egy „naív” feltételezés, mégpedig az, hogy minden jellemző független a többitől. Ez természetesen nem lesz mindig igaz, viszont ennek ellenére más algoritmusokhoz képest gyorsan tanul, és még akkor is jól működik, ha nincs elegendő képzési adatunk [14, 300. o.]. A modellek gyakran nem teljesítenek jól, ha sok jellemzővel rendelkeznek, ezt a jelenséget a dimenzió átkának szokták nevezni. A Naïve Bayes úgy oldja meg ezt a problémát, hogy szétválasztja az osztályváltozóval kapcsolatos feltételes jellemzőeloszlásokat, így az egyes eloszlások egymástól függetlenül egyetlen dimenziós eloszlásként becsülhetők.

Következő algoritmusunk, a logisztikus regresszió (a továbbiakban LR) gyakorlatilag egy statisztikai modell, amely a szigmoid matematikai függvényt használja a lehetséges kategóriák előrejelzésére. Az egyes témakörök valószínűsége 0 és 1 közé fog esni, ezt az esélyt dönti el a szigmoid függvény a modellben.

Térjünk is át a Support Vector Machines (a továbbiakban SVM) nevű modellre. Bináris osztályozás esetén ez az algoritmus alapvetően pontokként ábrázolja a térben a betanulási adatmintákat. Ez olyan módon történik, hogy az adathalmazt két, lineárisan szétválaszható részre bontja egy széles határ vagy távolság mentén, és az új adatpontok az alapján kapják meg az egyik vagy a másik osztályt, hogy a határoló sáv melyik oldalára esnek. Nekünk azonban most nem ilyen egyszerű lineáris osztályozási folyamatra van szükségünk, hanem 2-nél több témakörrel dolgozunk. Több osztály esetén a modell minden osztályhoz egy bináris osztályozót tanít be, amellyel képes szétválasztani minden csoportot az összes többitől. Az előrejelzés során minden osztályozónak kiszámítódnak a pontszámai (a határoktól, vagyis a hipersíkoktól való távolságok) és a maximális pontszám lesz a választott osztály.

A negyedik tárgyalt modell, a Random Forest (véletlenszerű döntési fa) igyekszik kiküszöbölni a döntési fa elv azon hátrányát, hogy kellően sok adat esetén túl nagy és mély fákat kapunk, amelyek hajlamosak a túlillesztésre. Ez azt jelenti, hogy a modell tényleges „tanulás” helyett csak memorizálja a a tanulási mintákat, és túl konkrét szabályokat generál. Ezért túlzottan a betanulási adatokhoz fog illeszkedni a modell, és emiatt rosszabb teljesítményt fog adni a teszt adatokon. A véletlenszerű erdő egy olyan együttes modell, amely több döntési fa osztályozót is illeszt az adathalmaz többféle alcsoportjára, és átlagolást használ az előrejelzési pontosság javítására, illetve szabályozza a túlillesztést.

Az utolsó felügyelt gépi tanulásos modell, amelyet be fogok mutatni, a Gradient Boosting Machines (a továbbiakban GBM). Ez az eljárás általában egy additív modellt épít fel, előre szakaszosan szekvenciált módon. Ezzel tetszőleges differenciálható veszteségfüggvényt (olyan függvény, amely segít meghatározni, milyen pontosan teljesít egy gépi tanulási algoritmus) lehetséges optimalizálni. A modell előnye, hogy jól használható regressziós és osztályozási problémák megoldására is. A Scikit-Learn csomag a GBRT-t (Gradient Boosted Regression Trees), egy ilyen általánosított modellt használ.

A vizsgált szövegkorpusz

Az osztályozáshoz kiterjesztettem az eredetileg adott, Harry Potter első kötetet szövegét az összes többi Harry Potter könyv szövegére. Így véleményem szerint már elegendő anyag lesz az osztályozási gépi tanulásos algoritmusok hatékonyságának, és persze a témaválasztások alkalmasságának elemzéséhez. Ehhez igyekeztem a köteteknek már *txt* formátumban meglévő szövegeit beszerezni, lehetőleg egy forrásból, azért, hogy egységes legyen az osztályozandó korpusz [41].

A *prepare\_for\_classification.py* fájl segítségével végeztem el a 7 kötet szövegének fejezetekre bontását, és elemezhető formára hozását. Először előkészítettem a fejezetek és a témakörök tárolásához szükséges adatstruktúrákat.

Az osztályozási algoritmusok későbbi ellenőrzéséhez szükség volt arra, hogy a 7 könyv minden egyes fejezetét manuálisan különböző kategóriákba csoportosítsam. Ehhez Michael Siebel 2020-as cikkét vettem alapul, ahol a szerző szintén a Harry Potter kötetekre végzett el NLP elemzéseket R programozási nyelv segítségével [42]. A szerző négy témakört határozott meg, amelyek nagyjából lefedik a könyvsorozat történéseit. Ezek a következők: a muglik világa, a Roxforton kívüli varázsló közösség, Voldemort történetszála, az utolsó pedig a Roxfortban (és a kviddicspályán) zajló események. Ezeket a témákat használhatónak találtam, ugyanis általában valóban a könyvek különböző részeire jellemző a felbukkanásuk (a muglik világa a könyvek elején, a varázslóközösségek és intézmények témára nincs általános szabály, Voldemort történetszála inkább az utolsó kötetekben kerül előtérbe, a Roxfort eseményei pedig a könyvek középső részén vannak többségben.

Tehát a könyvek összes fejezetéhez szerintem leginkább passzoló témekörök számait rögzítettem egy cél listában, amellyel össze lehet majd hasonlítani az osztályozási algoritmusok eredményeit. A témakörök számait egy kulcs-érték párokat tartalmazó lista segítségével fogja a rendszer összerendelni a témakörök neveivel.

Következő lépésként beolvastam a könyvek szövegeit a fájlokból, és szétválasztottam őket a fejezetek szerint. A keletkezett fejezeteket eltároltam egy listában. Ebből fog keletkezni az osztályozandó szövegkorpusz.

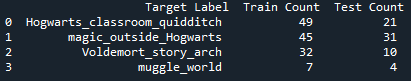
Ezután a fejezetek szövegeit a pandas adatelemző könyvtár segítségével adattáblába szerveztem, majd az egyes fejezetekre elvégeztem a normalizálást. Ez, mint a statisztikai elemzéseknél már láttuk, azt jeleni, hogy a szövegeket megtisztítottam a nem releváns, vagy redundáns információktól, amelyek torzíthatják az osztályozó algoritmusok eredményeit, tehát elemezhető formába hoztam a szövegeket. A statisztikai vizsgálatoknál használt szövegekhez képest itt annyival könnyebb volt a helyzetem, hogy minden mondat egy-egy sorban szerepel a dokumentumokban, és nem tartalmaznak például üres sorokat. A normalizálás ezesetben a következő lépésekből állt:

Még a szövegfájlok beolvasása után átalakítottam Hagrid dialektusának nagy részét (pl.: an’ -> and, don’ -> do not, yeh -> you). Azért csak nagy részét, mert vannak Hagrid szövegeiben olyan szavak is, amelyek nem az előbb említett példákhoz hasonló, gyakran előforduló névmások vagy segédigék, hanem például normál igék (pl. askin’ -> asking lenne helyes). Ezek kikeresgélése az összes kötetben, és felvételük az átalakító algoritmusba túl sok időt vett volna igénybe, és úgy gondolom, hogy lényegesen nem torzítják a vizsgálati eredményeket ezek az elemzés szempontjából „elvesztett” szavak. Akkor már talán okozhatna ez problémát, ha részletesebb lebontásban próbálnánk témakörök szerint elemezni a szövegeket, például akár az egyes karakterek szinjén.

Ezután az aposztrófos kifejezések normál kifejezéssé alakítása (pl.: you’re -> you are), a szövegek kisbetűssé tétele, szótövesítés, a speciális karakterek eltávolítása, és az elemzés szempontjából haszontalan szavak (stopwords) eltávolítása is megtörtént.

Az eredményül kapott, megtisztított szövegkorpusz alkalmassá vált arra, hogy azt elkülönítsem egy betanulási és egy teszt adatcsoportra. Ezt a scikit-learn *train\_test\_split* metódusa segítségével valósítottam meg. A metódus alapbeállítása szerint összekeveri a fejezetek szövegeit, és az összekevert listát két csoportra választja szét. Emellett van benne egy *random\_state* paraméter, amellyel az összekeverés véletlenszerűségét határozhatjuk meg. Amennyiben nem adjuk meg ezt a paramétert, akkor minden híváskor más szétválasztási eredményt kapunk. Viszont, ha egy integer értéket adunk meg neki, akkor egy adott egész számmal besorolt véletlenszám-generátort kapunk, így minden híváskor ugyanazt az eredményt kapjuk.

A teszt csoport méretét feleakkorának határoztam meg, mint a betanulási halmazt, tehát 2/3 – 1/3 felosztást valósítottam meg. 133 betanulási, és 66 teszt fejezet keletkezett. Érdemes megvizsgálni, hogy a kézzel kiosztott kategóriák szerint hogyan oszlanak el a fejezetek, lebontva a betanulási és a teszt szövegekre. Ez a xy. ábrán látható. Mint látható, a mugli világhoz jóval kevesebb fejezetet sikerült bekategorizálnom, mint a másik három

**témához.

*x. ábra: a fejezetek témakörök szerinti eloszlása*

Végül a betnaulási és a teszt adattáblákat elmentettem egy numpy típusú bináris fájlba, így nem kellett minden későbbi használatkor lefuttatni az elkülönítési metódust.

Az osztályozási eredmények

Az alábbiakban szeretném elemezni az előkészített szövegkorpuszon elvégzett osztályozási feladat ereményeit. A teszteredmények 0 és 1 közé eső számok lesznek, és azt kell figyelembe vennünk, hogy minél közelebb esik a kapott érték 1-hez, annál pontosabban teljesített a csoportosítási modellünk. Az egyes osztályozási algoritmusokat először az alapbeállításként megadott, vagy a dokumentáció által javasolt paraméterekkel teszteltem.

Az eredmények kiértékelése előtt szeretném kifejteni, hogy mit jelent pontosan a kereszt-validációs vizsgálat, amelyet a betanulás során alkalmaztam. Abban az esetben, amikor a becslések különböző, manuálisan beállítandó paraméterei (hiperparaméterek) kiértékelődnek (ilyen például az SVM és a LR modell esetén a C-paraméter), fennáll a túlillesztés veszélye a tesztkészletben, mert a paraméterek addig módosíthatók, amíg az osztályozó optimálisan nem teljesít. Így a tesztkészlettel kapcsolatos jellemzők „szivároghatnak” a modellbe, és az értékelési mérőszámok már nem az általános teljesítményt mutatják. Ennek a problémának a megoldására elkülöníthetjük az adathalmaz egy másik részét ún. „validációs halmazként”. Így a képzés a betanítási halmazon zajlik le, majd a validációs csoporton már megtörténik egy kiértékelés, és amikor úgy tűnik, hogy sikeres lesz a kísérlet, elvégezhető a végső értékelés a teszt adathalmazon. Viszont rendelkezésre álló adatok három halmazra történő felosztásával drasztikusan csökkentjük a modell betanításához felhasználható minták számát, és az eredményeket függővé tesszük attó, hogy milyen véletlenszerű tanulási és validációs halmazpárt választunk ki.

A probléma megoldása a keresztellenőrzésnek (CV) nevezett eljárás. A végső értékeléshez még ezesetben is el kell különítenünk egy tesztkészletet, de az keresztellenőrzések elkészítésekor már nincs szükség az érvényesítő halmazra. A k-szoros CV-nek nevezett alapmegközelítésben a képzési halmazt *k* db kisebb halmazra osztjuk fel.

Ezek után tehát vizsgáljuk meg először a szózsák alapú adathalmazra kapott eredményeket (12. melléklet). Mint láthatjuk, ötös osztású (alapértelmezett) kereszt-validációval is ellenőriztem a kapott adatok pontosságát. A keresztvalidációs és a tesztelési pontosságok eredményeit az alábbi táblázatban foglaltam össze (3 tizedesjegy pontossággal):

|  |  |  |
| --- | --- | --- |
|  | Bag of Words | TF-IDF |
| SVM with Stohastic Gradient Descent | **0,621** | 0,606 |
| Support Vector Machines | 0,591 | 0,606 |
| Random Forest | 0,591 | 0,561 |
| Multinomial Naive Bayes | 0,576 | 0,424 |
| Logistic Regression | 0,576 | **0,636** |
| Gradient Boosting Machines | 0,545 | 0,515 |
| Average test accuracy | 0,583 | 0,558 |

*x. táblázat: szövegosztályozási eredmények különböző gépi tanulási és adatkinyerési modellekkel*

Legjobban az SVM SGD, SVM, illetve a RF modellek teljesítettek. Azonban a továbbiakban azt várjuk, hogy a TF-IDF adatkinyerési metódussal jobb eredményeket fogunk kapni, ebben az esetben ugyanis kevesebb olyan jellemző marad az adathalmazban, ami a vizsgálat szempontjából kevésbé fontos.

A TF-IDF-el kinyert adatokon alapuló csoportosítás eredményeit a 13. mellékleten láthatjuk. Az átlagos teszt pontosság ugyan valamelyest csökkent, viszont itt már 3 modell eredménye is 0,6 felé esik, és eddig a legmagasabb eredményünket is itt kaptuk, a LR modell esetében (0,636).

Irodalomjegyzék

[1] Szalavetz Andrea. Mesterséges intelligencia és technológiavezérelt termelékenységemelkedés. Külgazdaság, 2019, 63.7-8: 53-79.

[2] Noam Chomsky: Syntactic Structures. Mouton & Co. 1957.

[3] Church, Kenneth. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In Proceedings of the Second Conference on Applied Natural Language Processing (pp. 136-143).

[4] Pang, Bo; Lee, Lillian. (2008). Opinion Mining and Sentiment Analysis. Foundations and Trends® in Information Retrieval, 2(1-2), 1-135.

[5] Tjong Kim Sang, Erik F.; De Meulder, Fien. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 (pp. 142-147).

[6] Sutskever, Ilya; Vinyals, Oriol; Le, Quoc V. (2014). Sequence to Sequence Learning with Neural Networks. In Advances in Neural Information Processing Systems 27 (pp. 3104-3112).

[7] Mikolov, Tomas; Chen, Kai; Corrado, Greg; Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781.

[8] Hochreiter, Sepp; Schmidhuber, Jürgen. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.

[9] Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Łukasz; Polosukhin, Illia. (2017). Attention is All You Need. In Advances in Neural Information Processing Systems 30 (pp. 5998-6008).

[10] Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

[11] Radford, Alec; Narasimhan, Karthik; Salimans, Tim; Sutskever, Ilya. (2018). Improving Language Understanding by Generative Pretraining. arXiv preprint arXiv:1701.00160.

[12] Bolukbasi, Tolga; Chang, Kai-Wei; Zou, James Y.; Saligrama, Venkatesh; Kalai, Adam T. (2016). Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In Advances in Neural Information Processing Systems 29 (pp. 4349-4357).

[13] Radford, Alec; Chen, Jonathon; et al. (2021). Learning Transferable Visual Models From Natural Language Supervision. arXiv preprint arXiv:2103.00020.

[14] Dipanjan Sarkar: Text Analytics with Python. A Practitioner’s Guide to Natural Language Processing. Second Edition. Apress, Bangalore, Karnataka, India, 2019

[15] Steven Bird, Ewan Klein, and Edward Loper: Natural Language Processing with Python. O’Reilly, 2009

[16] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781 (2013)

[15] Parisotto, Emilio; Salakhutdinov, Ruslan. (2017). Neural Map: Structured Memory for Deep Reinforcement Learning. arXiv preprint arXiv:1702.08360.

[16] Gabriel Wittum, Michael Hoffer, Babett Lemke, Robert Jabs, Arne Nägel: Automated methods for the comparison of natural languages. Computing and Visualization in Science, 2020. 23:7, <https://doi.org/10.1007/s00791-020-00325-2>

[17] Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, Marco Baroni: Colorless green recurrent networks dream hierarchically. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018.

Programnyelvekhez:

[18] https://careerfoundry.com/en/blog/data-analytics/best-machine-learning-languages/#what-skills-are-important-for-machine-learning

[19] https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

[20] https://www.springboard.com/blog/data-science/best-language-for-machine-learning/

[21] <https://neptune.ai/blog/programming-languages-machine-learning>

Fejlesztőkörnyezetekhez:

[22] https://github.com/atom/atom

[23] https://thonny.org/

[24] <https://www.upgrad.com/blog/python-ides-for-data-science-machine-learning/>

Library-khez:

[25] https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb

[26] https://sunscrapers.com/blog/9-best-python-natural-language-processing-nlp/#textblob

[27] <https://bitbytesoft.com/top-nlp-libraries-python/#CoreNLP>

[28] John D. Hunter: Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90-95.

[29] WesMcKinney: Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference, 51-56.

[30] a spaCy hivalos weboldala: <https://spacy.io/usage/spacy-101>

[31] Stefan van der Walt, S. Chris Colbert, Gael Varoquaux: The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering, 13(2), 22-30.

[32] Radim Řehůřek, Petr Sojka: Software Framework for Topic Modelling with Large Corpora. Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 45-50.

[33] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[34] Aric A. Hagberg, Daniel A. Schult, Pieter J. Swart: Exploring network structure, dynamics, and function using NetworkX. Proceedings of the 7th Python in Science Conference, 11-15.

[35] Eric Jones, Travis Oliphant, Pearu Peterson: SciPy: Open Source Scientific Tools for Python. 2001.

[36] J. K. Rowling: Harry potter and the philosopher's stone. Bloomsbury Childrens Books. 2014.

[37] <https://wordsrated.com/harry-potter-stats/>

[38] <https://www.regens.com/hu/-/natural-language-processing-what-is-it-and-how-can-you-put-it-to-use->

[39] <https://www.mathworks.com/help/stats/machine-learning-in-matlab.html>

[40] <https://github.com/axa-group/nlp.js>

[41] <https://github.com/idc9/stor390/tree/master/notes/natural_language_processing/rowling>

[42] <https://siebelm.github.io/Harry_Potter_1/#Q1>

[43] <https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation>

A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás Mellékletek

**A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás**1. melléklet – szavak és sorok száma a különböző nyelvű szövegekben

2. melléklet – az angol szöveg szóhosszúságai és ezen szavak száma

A képen szöveg, képernyőkép, diagram, sor látható

Automatikusan generált leírás3. melléklet – a német szöveg szóhosszúságai és ezen szavak száma

A képen szöveg, képernyőkép, diagram, Diagram látható

Automatikusan generált leírás

4. melléklet – a francia szöveg szóhosszúságai és ezen szavak száma

A képen szöveg, képernyőkép, Diagram, diagram látható

Automatikusan generált leírás

5. melléklet – a spanyol szöveg szóhosszúságai és ezen szavak száma

A képen szöveg, képernyőkép látható

Automatikusan generált leírás6. melléklet – az angol szöveg leggyakoribb szavai és előfordulásuk száma

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

7. melléklet – a német szöveg leggyakoribb szavai és előfordulásuk száma

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

8. melléklet – a francia szöveg leggyakoribb szavai és előfordulásuk száma

A képen szöveg, képernyőkép látható

Automatikusan generált leírás

9. melléklet – a spanyol szöveg leggyakoribb szavai és előfordulásuk száma

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

10. melléklet – LSI\_SVD összefoglaló szöveg minta

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

11. melléklet – TextRank összefoglaló szöveg minta

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

12. melléklet – szózsák modell alapú osztályozási eredmények többféle osztályozási algoritmussal

1. https://github.com/curiosity-ai/catalyst [↑](#footnote-ref-1)
2. https://www.anaconda.com/download [↑](#footnote-ref-2)
3. <https://www.spyder-ide.org/> [↑](#footnote-ref-3)
4. https://docs.spyder-ide.org/current/panes/ [↑](#footnote-ref-4)
5. https://pypi.org/project/ipdb/ [↑](#footnote-ref-5)
6. https://harrypotter.fandom.com/wiki/Harry\_Potter\_in\_translation [↑](#footnote-ref-6)
7. A konvertálást a <https://www.pdf2go.com/pdf-to-text> oldal segítségével végeztem el. Utolsó elérés időpontja: 2023.11.29. [↑](#footnote-ref-7)
8. American Standard Code for Information Interchange [↑](#footnote-ref-8)
9. A metódus egy felügyelet nélküli algoritmus alapján egy modellt épít a rövidítő szavak, kollokációk és a mondatkezdeti szavak megtalálására, majd felhasználja ezt a modellt a mondathatárok elkülönítéséhez. A jelenlegi megközelítés a dokumentáció szerint számos európai nyelv esetében jól működik (jelenleg a PunktSentenceTokenizer modult használja ehhez a metódus). Forrás: <https://www.nltk.org/api/nltk.tokenize.PunktSentenceTokenizer.html#nltk.tokenize.PunktSentenceTokenizer>

   ill. [14, 121. o.] [↑](#footnote-ref-9)