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 fejlesztése, 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 a barátnőmnek a végtelen türelmet és támogatást, amelyet kaptam tőle a dolgozat megírása közben. Szeretnék még köszönetet mondani a családomnak, akik nélkül sosem jutok 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

# Tartalomjegyzék

1. Bevezető 7

2. Az NLP problémakör áttekintése 9

3. Python fejlesztőkörnyezet összeállítása 17

# **1. 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 egy olyan idegennyelv tanulását segítő eszköz fejlesztése, amely egy szövegkorpusz automatizált feldolgozásával statisztikai kimutatásokat készít, emellett a benne szereplő szavakat témakör szerint csoportosítja, illetve egy tartalmi összefoglalót is megfogalmaz belőle. 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 az egyre fokozódó, robbanásszerű 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 digitális korszak előrehaladtával az idegen nyelvű szövegek óriási mennyisége vált elérhetővé. Ez a számítógépek által rendelkezésünkre bocsátott egyre hatalmasabb számítási kapacitás felhasználásával már eddig is lehetőséget teremtett a különböző témájú szövegkorpuszok mélyebb megismerésére, rendszerezésére. Ugyanakkor ezeknek a 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 megoldása révén egy olyan rendszer hozható létre, amely a rendelkezésre álló szövegkorpuszokat, jelen esetben a közismert Harry Potter könyvsorozat első regényének szövegét [36] tematikus struktúrába szervezi. Mindezek mellett egy, a könyv tartalmába, fontosabb mozzanataiba betekintést nyújtó összefoglalót is elkészítészítettem.

A könyv szövegének statisztikai alapú vizsgálatát négy különböző nyelven alkottam meg. 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.

# **2. Az NLP problémakör áttekintése**

**2.1. Az NLP**

Az elemzéseim megalapozásához fontos volt, hogy a természetes nyelvek feldolgozása (Natural Language Processing, a továbbiakban NLP) témakörét próbáljam alaposabban megérteni. Ez a terület jelenleg már megkerülhetetlen a technológiai fejlődés és az adatmennyiség hihetetlen mértékű növekedésének köszönhetően. Az NLP a mesterséges intelligencia és a nyelvészet metszetében áll. Célja nem kevesebb, mint az emberi és a gépi kommunikáció közötti szakadék áthidalása. 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.

**2.2. Alapvető feladatok az NLP-ben**

**2.2.1. Szövegosztályozás és érzelemelemelemzés**

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.

**2.2.2 Nevezett 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.

**2.2.3 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.

**2.3. Deep learning és NLP**

**2.3.1. 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.

**2.3.2. 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. Természetesen meg kell említenem, hogy az LSTM használata sikeresen beépíthető olyan feladatokba is, mint a nyelvi modellezés és a gépi fordítás.

**2.3.3. 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.

**2.4. Kihívások és jövőbeli irányok**

**2.4.1 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-rendszerek felelős fejlesztésére és figyelmes gondozására van szükség, ha a lehetőségekhez mérten igazságos és elfogulatlan eredményeket szeretnénk biztosítani.

**2.4.2 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.

**2.5. NLP a gyakorlatban**

**2.5.1. 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.

**2.5.2. Az alkalmazott megközelítés**

Dipanjan Sarkar „Text Analytics with Python: A Practitioner’s Guide to Natural Language Processing” [14] című könyvét kiválóan tudtam hasznosítani 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 az általam fejlesztett alkalmazáshoz 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.

**2.6. 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 hozzájárul annak megértéséhez, hogy a deep learning modellek hogyan rögzítik és dolgozzák fel a különböző nyelvekben a hierarchikus nyelvi jellemzőket.

# **3. Python fejlesztőkörnyezet összeállítása**

**3.1. Programozási nyelvek [18] [19] [20] [21]**

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. A listát nem más nyelvvel, mint a Pythonnal kezdem.

**3.1.1. 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 (lásd a hivatkozásokat).

**3.1.2. Az R nyelv**

Az R programozási nyelv legfontosabb felhasználási területei közé küzé tartozik a statisztikai analízis és adatvizualizáció. A statisztikai modellezés és a gazdag vizualizációs lehetőségek révén segíti az adatelemzési feladatokat. Hátrányai közé sorolható például, hogy a szintaxisa néha bonyolult lehet, és inkább a tisztán statisztikai területeken találhatók meg a fő alkalmazási területei.

**3.1.3. A C++**

A C++ nyelv legerősebb tulajdonságai közé tartozik a hatékonyság és a teljesítmény, ezeken a területeken emelkedik ki elsősorban. Leginkább rendszerprogramozásban és játékfejlesztésben alkalmazzák, és rendkívül erőteljes és sokoldalú nyelvnek számít. Mindezek ellenére a C++ alaposabb megtanulásához szükséges idő hosszabb lehet, illetve a manuális memóriakezelés nagy komplexitást eredményezhet.

**3.1.4. A Java**

A Java platformfüggetlen, így ideális választás nagy rendszerek és vállalati alkalmazások fejlesztésére. Robosztus és biztonságos. Hátránya, hogy a kód hosszabb, terjengősebb lehet, és a bonyolult lekérdezések néha kihívást jelenthetnek.

**3.1.5. A JavaScript**

A JavaScript a kliensoldali webfejlesztés alapnyelve. Dinamikus funkciókat hoz a böngészőkbe. A különböző böngészők támogatása miatt előfordulhatnak kompatibilitási problémák.

**3.1.6. 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.

A C++ nyelvhez képest a Python egyszerűbb, gyorsabb benne a fejlesztés, emellett sokoldalú is. Ezzel szemben előbbi inkább teljesítményorientált, komplex és erőteljes hardverközeli fejlesztést tesz lehetővé.

Ha a Java nyelvvel kellene szembeállítanom a Pythont, akkor azt emelném ki, hogy a Python egyszerű szintaxissal rendelkezik, és emellett széles körű alkalmazhatóságot kínál. A Java nagy előnye szintén a platformfüggetlenség, ami vállalati alkalmazások, robosztus rendszerek fejlesztésénél hasznos, viszont mint már volt róla szó, terjengősebb a szintaktikáj, mint a Pythonnak.

Ha a JavaScript viszonylatában vizsgálódunk, akkor abban lehet a fő különbséget meglátni, hogy a Python inkább a szerveroldali fejlesztéshez alkalmasabb, illetve a gépi tanulás, és a ttudományos 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 is alkalmas lehet.

**3.1.7. 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 a webfejlesztéstől kezdve, adatelemzésen át egészen a gépi tanulásig sokféle területen használható. Továbbá gazdag és sokoldalú könyvtárkészlettel rendelkezik, többek között az NLP területén, amely jelentősen felgyorsítja a fejlesztést. A Python szerencsére aktív közösséggel rendelkezik, könnyen hozzáférhetők dokumentációk, segítségek és fejlesztői erőforrások.

Ezek a tulajdonságok együttesen teszik a Pythont az egyik legkedveltebb választássá a fejlesztők és tudósok körében. Mint már említettem, a jól olvasható kódja és a gyors fejlesztési idő nagy előnyt biztosít, amikor gépi tanulásos problémákat, feladatokat kell megoldani, implementálni, illetve a megoldások különböző adatkészletekkel és paraméterekkel való tesztelésében.

**3.2. Fontosabb Python fejlesztőkörnyezetek [22] [23] [24]**

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.

**3.2.1. 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.

**3.2.2. 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.

**3.2.3. PyCharm**

A PyCharm-ot a JetBrains kifejezetten Python fejlesztéshez tervezte. Fontos funkciói közé tartozik a kódok ellenőrzése, a refaktorálás és a kódelemzés. A PyCharm támogatja például a Django és a Flask keretrendszereket. Az integrált debugger és a beépített unit tesztelési eszközök segítik a hibakeresést és a kódminőség fenntartását. A teljes verzió díjköteles, bár létezik ingyenes verzió is.

**3.2.4. 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.

**3.2.5. Visual Studio Code**

A Visual Studio Code egy könnyűsúlyú, mégis erőteljes kódszerkesztő, amely jól kiterjeszthető és testreszabható. Támogatja a Python programozási nyelvet, és számos kiterjesztés közül választhatunk, amelyek segítenek a produktivitás növelésében. Gyors és reszponzív, és alkalmas kisebb projektekhez és nagyobb alkalmazásokhoz egyaránt. Hátrányként említhető, hogy levésbé hangsúlyozza a Python-specifikus fejlesztést.

**3.2.6. 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.

**3.2.7. 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.

**3.2.8. 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.

**3.2.9. DataSpell**

A DataSpell egy integrált fejlesztőkörnyezet, amely támogatja a nagy adathalmazokon végzett feldolgozásokat és az adatvizualizációt. Ez a környezet magában foglalja a Jupyter Notebookot és olyan egyéb eszközöket, amelyek az adatelemzésben és gépi tanulási projektekben nyújtanak segítséget. A DataSpell gyors kódellenőrző és hibakereső eszközökkel rendelkezik, elősegítve ezzel a hatékony fejlesztést. Hátránya a kisebb ismertség és szűkebb közösség.

**3.2.10. 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.

**3.3. Függvénykönyvtárak [25] [26] [27]**

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.

**3.3.1. 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.

**3.3.2. 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.

**3.3.3. 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.

**3.3.4. 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.

**3.3.5. 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.

**3.3.6. 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.

**3.3.7. scikit-learn [33]**

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.

**3.3.8. 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.

**3.3.9. 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.

# **4. A Spyder konfigurációja**

**4.1. Az Anaconda**

Az Anaconda egy népszerű AI és Data Science toolokat biztosító, Python-ban való fejlesztést segítő platform. Fontos tulajdosága, hogy támogatja az open source programokat, környezeteket, illetve elérhetőséget biztosít vállalati szintű Data Science eszközök kipróbálásához. Az Anaconda Cloud egy felhő alapú tárhelyet is biztosít a lokális környezetünk biztonságos tárolásához.

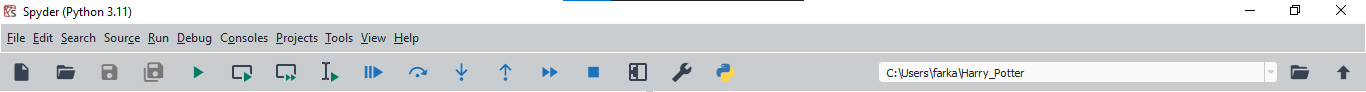
Az Anaconda repository-jában találjuk meg emellett a Conda nevű csomag menedzselő rendszert. Ennek segítségével gyorsan és hatékonyan lehetséges telepíteni, frissíteni a munkámhoz szükséges adatfeltáráshoz, vizualizációhoz és gépi tanuláshoz használatos csomagokat, és futtatni az ezekre támaszkodó Python progamokat (más programnyelveket is támogat a Conda).

Az Anaconda Navigator[[1]](#footnote-1) nevű, asztali számítógépekre letölthető Anaconda disztribúcióval lehetőségünk nyílik többféle, gépi tanulást is támogtó open source szoftver használatára, és egy lokális fejlesztői környezet összeállítására. Segítségével könnyen menedzselhetők az általunk választott alkalmazások, csomagok, és környezetek, egy grafikus felületen.

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.

**4.2. A Spyder**

A fejlesztéshez a már korábban bemutatott Spyder fejlesztőkörnyezetet használtam. Ez egy kifejezetten Python programozáshoz tervezett integrált fejlesztőkörnyezet (IDE). A Spyder hivatalos weboldala[[2]](#footnote-2) 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, 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 felhasználót.

A kódok megírásához az Editor ablaktáblában létrehoztam a szükséges Python szkripteket. A különféle funkciók, plédául a szintaxis-kiemelést, a kódkiegészítést és a beépített súgó, hozzájárulnak a hatékony és élvezetes kódolási élményhez. 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:

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[[3]](#footnote-3). 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.

**4.3. 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.

# **5. Szövegkorpuszok összegyűjtése és statisztikai elemzése**

**5.1. 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. Ugyan a könyvsorozatot már 80 különböző nyelvre lefordították [], viszont én kiválasztottam a kötet angol, német, francia és spanyol nyelvű, interneten elérhető változatait pdf formátumban.

**5.2 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 mik a legfontosabb kifejezései a különféle szövegekben. A szöveg leggyakoribb és leghosszabb szavai is ugyanezt a célt szolgálhatják.

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.

**5.2. A választott szövegek előfeldolgozása**

A szövegkorpusz meghatározása után megkezdődhetett a szöveg elemezhető formába alakítása (normalizálás). Először is a begyűjtött pdf-eket txt formátumba konvertáltam[[4]](#footnote-4). 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, ami reguláris kifejezésekkel való operációkat, szöveg helyettesítéseket tartalmaz.

**5.2.1. 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 képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírásA következő lépés az eredeti pdf-ből áthozott, szövegben maradt felesleges, oldalszámokat 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 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[[5]](#footnote-5) 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.

**5.2.2. 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, hogy a kiindulási szövegkorpusz amennyire csak lehet, egységes legyen.

A német szövegben az eredeti pdf-ben footer szövegben fordult elő minden oldalon 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.

**5.2.3. 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.

**5.2.4. 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.

**5.3. Statisztikai vizsgálatok**

**5.3.1. 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[[6]](#footnote-6). 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.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás 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* metó-

*ábr*a

dus. A fenti példákból látszik, hogy nem feltétlenül egyszerű kérdés egy nagy méretű szövegkorpuszból a mondatok kinyerése. Azt gondolhatnánk, hogy esetünkben ugyanannak a regénynek négy különböző nyelvű verziójánál a mondatok száma meg kell 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 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.

|  |  |
| --- | --- |
| 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).

**5.3.2. 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.

# **6. A könyv szövegének tartalmi összefoglalója**

**6.1. a 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.

**6.2. LSI SVD**

**6.2.1. 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.

**6.2.2. Az 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.

**6.3. TextRank**

**6.3.1. 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. Építsünk egy dokumentum-kifejezés jellemzőmátrixot olyan modellek segítségével, mint 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.

**6.3.2. A 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 hosszabb, több fejezetnyi terjedelem után ütközik ki egyértelműen.

# **7. Témakörök szerinti elemzés**

# **8. Összefoglaló**

# **9. Summary**

# **10. 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://www.datacamp.com/tutorial/data-science-python-ide

[23] https://www.projectpro.io/article/best-python-ide-for-data-science-and-machine-learning/812#:~:text=PyCharm%20is%20one%20of%20the,syntax%20highlighting%2C%20and%20error%20checking.

[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/

# **A képen szöveg, képernyőkép, diagram, Diagram látható Automatikusan generált leírásA képen szöveg, képernyőkép, diagram, Diagram látható Automatikusan generált leírás11. Mellékletek**

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

1. https://www.anaconda.com/download [↑](#footnote-ref-1)
2. <https://www.spyder-ide.org/> [↑](#footnote-ref-2)
3. https://pypi.org/project/ipdb/ [↑](#footnote-ref-3)
4. A konvertálást a <https://www.pdf2go.com/pdf-to-text> oldal segítségével végeztem el. Utolsó elérés időpontja: 2023.11.29. [↑](#footnote-ref-4)
5. American Standard Code for Information Interchange [↑](#footnote-ref-5)
6. A metódus gy 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-6)