/2020.

SZAKDOLGOZAT

Kecse Ábel

2020

Pécsi Tudományegyetem
Műszaki és Informatikai Kar
Mérnökinformatikus Szak

SZAKDOLGOZAT

Web oldalakon szereplő szövegek hasonlóságának keresése
és kimutatása python program segítségével

Készítette: Kecse Ábel
Témavezető: Témavezető Neve
Konzulens: Dr. Iványi Péter

Pécs

SZAKDOLGOZAT FELADAT

..... Kecse Ábel .....
hallgató részére

A záróvizsgát megelőzően szakdolgozatot kell benyújtania, amelynek témáját és feladatait az alábbiak szerint határozom meg:

Téma:

Web oldalakon szereplő szövegek hasonlóságának keresése és kimutatása python program segítségével

Feladat:

- Nyelvi összehasonlító algoritmusok keresése
- Algoritmusok implementálása python-ban
- Teszt oldalak létrehozása, algoritmusok ellenőrzése
- Algoritmusok ellenőrzése valós web oldalakon

A szakdolgozat készítéséért felelős tanszék: ……

Külső konzulens: ……
munkahelye:…….

Témavezető: …Iványi Péter…….
munkahelye:…MIK, RSZT Tsz…….

Pécs, 2019. szeptember 24.

Dr. Iványi Péter
szakvezető

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye. A felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Egyéb jelentősebb segítséget nem vettem igénybe.
Az elkészült szakdolgozatban talált eredményeket a feladatot kiíró intézmény saját céljaira térítés nélkül felhasználhatja.

Pécs, 2020. január 6.

.....
hallgató aláírása

Köszönetnyilvánítás

Köszönöm a témavezetőmnek a témakiírást és a támogatását a szakdolgozat elkészítése során.

1.1. Tartalomjegyzék

Table of Contents

A szakdolgozatom témája: weboldalakon szereplő szövegek hasonlóságának keresése és kimutatása python program segítségével. Több mint 5000 ezer viagra, kamagra és hasonló vényköteles vagy illegális potencianövelő szerek árusításával foglalkozó weboldal kielemzése és egymással összevetése a cél.

A nyelvi összehasonlító algoritmusok főbb lépéseit bemutatom és az egyes lépések megvalósítási módjait. Az összehasonlítás egyes lépéseit a legelőnyösebb módszerrel probálom implementálni, indoklom a választásom a módszerek előnyei és hátrányainak mérlegelése után.

A weboldalak összehasonlítása több lépéses folyamat, sok elemre lehet bontani, egy absztrakt szinten a következő lépésekből áll:

1. weboldalak beolvasása, lementése
2. plaint text kigyűjtése:

A html tag-eken belüli szövegeket szűrjük ki, a script, style tagek tartalmát eldobjuk, a képeket, videókat, ehhez hasonló tartalmakat szintén kiszűrjük a BeautifulSoup nevű könyvtárral.

3. dokumentumok vektorizálása és eltárolása

3.a dokumentumok feldarabolása

A vektorizáláshoz fel kell darabolni a dokumentumot, ezeket a darabokat vektorizáljuk, az dokumentumot így egy vektor listával reprezentáljuk, a vektor listák összehasonlításán keresztül vonjuk le a következtetést két dokumentum hasonlóságáról.

3.b a dokumentum darabok vektorizálása

A dokumentum darabokban szereplő szavakat összehasonlítjuk az egész dokumentum abc sorrendbe rendezett szókincsével, ahol egyezést találunk ott növeljük a vektor elemét 1-el, lásd lentebb (**Példa a dokumentum vektorokká alakítására**)

4. vektorműveletek elvégzése, hasonlósági metrika meghatározása
5. Az egyes dokumentumokra meghatározott metrikák összehasonlítása, egyezés mértékének meghatározása

A algoritmusok bemutatása érdekében angol nyelvű példákat is bemutatok, de az egyes módszerekből összeválagott python implementációmat magyar dokumentumok összehasonlítására alkalmasnak tervezem. Az angol nyelvú példák pusztán szemléltetés céljából lettek felhasználva, a magyar szövegek feldolgozhatóságával összeegyeztethetők a példák.

Az alkalmazási területek:

Weboldalak szerző szerinti csoportosítása
Plágiumkeresés

1.2. A Weboldalak plain text-re alakítása

A weboldalak plain text-re alakítására két módszer áll rendelkezésre, az egyik hogy saját regexekkel dolgozunk, a másik pedig hogy kész erre kihegyezett könyvtárakat használunk. A saját regexek írásánál az alapgondolat, hogy az olyan tag-eket amiken belül nincs további, azoknak a tartalmát eltároljuk, majd kitöröljük a belső taget és folytatjuk ezt a folyamatot ameddig az összes tag-en belüli tag eltűnik. A belső tageket a

```
.*?(<.*?>)([^<^>]*[\w\s]{3,}[^<^>]*)<\/.*>
```

regex kifejezéssel mutatnak egyezőséget. A szimbólumok greedy azaz kapzsi viselkedését a ? szimbólummal megtudjuk változtatni, ez hasznos mivel alap esetben a * és + karakterek a legtöbb karakteres egyezést használják,így átsiklanánk a belső tageken. Olyan tagekkel mutat egyezőséget amely `<></>` ei között (azaz `<>itt</>`) legalább 3 egymást követő `[\s\w]` van, azaz legalább három szavas a tartalma (kiszűrve az 1,2 szavas tagek tartalmát pl.: menüpontok, oldalon belüli navigáció elemei). A zárójelekkel a csoportokat el tudjuk kapni azaz a tag határolóit és a tag tartalmát külön tudjuk kezelni, a csoportokra a `\1 (nyitó tag), \2 (tartalom), \3(zárótag)` karaktersorozattal tudunk hivatkozni. A legtöbb esetben a script tag-ek problémások, mivel bennuk a `< és >` karakterek amik a tag-eket határolják, itt algebrai kifejezéseket is takarnak. Ezeket a javascript blokkokat érdemes a feldolgozás előtt eltávolítani, az alábbi kifejezések alkalmasak erre:

egy soros `<script...</script>` esetén:

```
.*?<script.*?<\/script>
```

több soros `<script...\n..\n..<\script>` esetén:

```
.*?<script.*$(\n.*?$)*?\n.*?<\/script
```

A legnagyobb probléma a belső tagekből újraalkotni a szöveget, ezt nem is sikerült saját magam megvalósítanom. Az erre fejlesztett könyvtárak egy fabejárási algoritmust használnak erre a célra, hogy a szöveget újra össze lehessen rakni a tageken belüli tagek eltárolása után. A felhasznált könyvtár a BeautfiulSoup könyvtár, a plain text-re alakítás után a whitespace karaktereket egy rekurzív reguláris kifejezés segítségével távolítottam el. A `tag.decompose()` eltávolítja a `script` és `style` tag-eket.

```
import urllib.request
import bs4
import html5lib
def get_text_bs(html):
    tree = bs4.BeautifulSoup(html, 'html5lib')
    body = tree.body
    if body is None:
        return None
    for tag in body.select('script'):
        tag.decompose()
```

```
    for tag in body.select('style'):
        tag.decompose()
    text = body.get_text(separator='\n')
    return text
tidy = lambda c: re.sub(
    r'(^\s*[\r\n]+|^\s*\Z)|(\s*\Z|\s*[\r\n]+)',
    lambda m: '\n' if m.lastindex == 2 else '',
    c)
html = urllib.request.urlopen('https://totalcar.hu/magazin/hirek/')
raw = get_text_bs(html)
```

A raw változó tartalmazza a weboldal plain text tartalmát.

Az html fájl és a kapott eredmény a mellékletben látható [1].

A html fájl és az abból kapott plain text összehasonlítása helyességét a fejlesztő oldalán található unit testekkel bizonyítja:

https://github.com/akalongman/python-beautifulsoup/blob/master/bs4/testing.py

a dokumentációban a fejlesztő kitér rá, hogy bug észlelése esetén jelentsék azt a felhasználók. Az könyvtár újabb és újabb verziói kerülnek a jövőben kiadásra ezért, folyamatosan tesztelik a könyvtárat.

1.3. A dokumentum határok, különböző darabolási eljárások

A dokumentumok összehasonlításához fontos,hogy megtudjunk határozni azokat a határokat amik szerint a szövegeket összehasonlítjuk, például szavanként, mondatonként, bekezdésenként stb.
Ha túl kicsi határokat választunk akkor a nagyobb az esély a hamis pozitív hasonlóságra, azaz olyan dokumentumokat is hasonlóknak vélhetünk amik nem hasonlóak. Ha viszont túl tág határokat választunk, akkor az egyezőségek felett átsiklunk.

Az alábbi tanulmány darabolási eljárásokat mutat be és ismerteti a fázis problémát:

Algoritmusok egynyelvű és különböző nyelvek közötti fordítások és plágiumok megtalálására doktori (Ph.D.) disszertáció - Pataki Máté DOI:10.15774/PPKE.ITK.2013.004

Egy dokumentumot feldarabolva vektorizálunk, a darabokat vektorizáljuk, a darabolásnál fontos , hogy a határokat úgy válasszuk meg, hogy két különböző dokumentumnál az egyezést mutató szövegeknél, a darabok határai megegyezzenek

Vegyünk két dokumentumot amik csak annyiban térnek el, hogy a második dokumentum címe egy szóval hosszabb. Egy szövegrészlet az első bekezdésből legyen pl.:

“felkelek, majd reggelizek, délelőtt még elmegyek futni és bevásárolni”

Ha a dokumemtum pl. 3 db szavanként kerül elhatárolásra, akkor a darabok legyenek az első dokumentum esetében:

[felkelek, majd, reggelizek]
[délelőtt, még, elmegyek]
[futni, és, bevásárolni]

Ekkor a második dokumentum esetében a darabok így kerülnek elhatárolásra:

[…, …, felkelek]
[majd, reggelizek, délelőtt]
[még, elmegyek, futni]
[és, bevásárolni, …]

Az egyezést az előtte lévő tartalom miatt nem ugyanott kezdtük el darabolni, ezért nem fogunk egyező darabokat találni. Ezt „**fázis-problémának**” nevezzük.

Az előző példával a **szavas darabolást (word chunking)** mutattam be n=3 esetén.

Az átlapolódó szavas darabolási eljárás (overlapping word chunking) hasonlít a szavas daraboláshoz, azzal a különbséggel, hogy itt minden szónál kezdődik egy új darab, amely szintén n db szóból áll. Ezzel az eljárással ki tudjuk szűrni a szöveg esetleges eltolódását. Ebből következik, hogy itt mindig annyi töredék keletkezik ahány szó van a dokumentumban, a szavas darabolással ellentétben, ahol n db szavanként keletkezenek a darabok. Ha n=1 akkor minden szó új darabba kerül.

A mondatonkénti darabolás (sentence chunking) mondathatárolonként darabolunk, kísérletezhetünk, azzal, hogy a vesszőt is mondathatárlóként használjuk, de sokszor a vessző más szerepet tölt be pl. felsorolás esetén.

A hash-kódon alapuló darabolási eljárás (hashed breakpoint chunking) egy egyszerű és
gyors függvényt (hash-függvény) használ annak megállapítására, hogy mely szavak
legyenek határok. Ehhez minden szóra kiszámítunk egy számértéket, esetünkben pl. a szó betűinek ASCII kódjait összeadjuk. Amennyiben ez a szám maradék
nélkül osztható n-nel akkor ez a szó a határoló, a következő egységet az ezt követő szó alkotja és az egységet lezárja a következő szó aminek számértéke n-el osztható.

Az alábbi módszerek esetén:

- szavas darabolás(word chunking),
- átlapolódó szavas darabolási eljárás (overlapping word chunking),
- hash-kódon alapuló darabolási eljárás (hashed breakpoint chunking)

a paraméterekkel való kísérletezés szükség a fázis-probléma kiküszöbölése érdekében. A mondatonkénti darabolás pedig nagyban támaszkodik a helyes központozásra, ami a weboldalak tartalmára kevésbé jellemző.

Egy további eljárás a **Noun Phrase Chunking**. Ez figyelembe veszi a nyelvtani szabályokat a határok megállapítása során, így küszöböli ki a fázisproblémát, paraméterezése ezért nem szükséges. Az angol nyelvben a szintaktikai elemzés során szócsoportokat (Phrase) különítünk el, majd ezeket további összetevőkre bontjuk. A mondat alapját az igei és a névszói csoportok (Verb Phrase, Noun Phrase) adják. Az igei és névszói csoportokra való bontás (szintaktikai elemzés) az angolban szokásos elemzési mód, a magyarban azonban szokatlan lehet a „hagyományos” elemzést gyakorlóknak.

A Noun Phrase Chunking implementációja az e-magyar rendszer része.

forrás: https://e-magyar.hu/

“Az **e-magyar.hu** rendszer a Magyar Tudományos Akadémia támogatásával készült a 2015-ben kiírt infrastruktúrafejlesztési pályázat keretében. A munkálatok a pályázat kedvezményezettje, a Nyelvtudományi Intézet koordinálásával széleskörű együttműködés keretében folytak, melyben részt vett a hazai nyelvtechnológia számos vezető kutatás-fejlesztő műhelye.
Közreműködő partner intézetek: MTA Nyelvtudományi Intézet, AITIA International Zrt.
Szegedi Tudományegyetem, MTA SZTAKI, Pázmány Péter Katolikus Egyetem, Morphologic Kft.”

“Mit csinál egy szövegfeldolgozó eszközlánc? Mit csinál konkrétan aze-magyar digitális nyelvfeldolgozó rendszer szövegfeldolgozó eszközlánca? Magyar nyelvű írott szöveget elemez, és lát el különféle kiegészítő információkkal az elemzés eredményeképpen. Egymásra épülő eszközökből áll: az egyes eszközök működésük során felhasználják a korábbiak eredményét.”

A rendszer először megállapítja a szavak mondatok határát, külön kezeli az írásjeleket, kivéve a rövidítéseknél, ahol a záró pont a rövidítés részét képezi,A morfológiai elemzés megadja az egyes szavakról az alaktani információkat, képzőket, igeragokat, stb. így a szótő visszafejthető. A magyar szóalakok jelentős részének, több alaktani elemzése van. A rendszer a szövegkönyezetalapján automatikusan dönt ilyen esetekben, kiválasztja a helyes elemzést. pl.: nő, fűz, nyír, birka, hat, hold, szülő, vár, ár, nyár, láb, eső. A függőségi elemzés következik ezután, ahol az ige és igekötő kapcsolatai jelenik meg pl. hogy a **barátságban** az **állt** igéhez kapcsolódó határozó. Végül a lánc utolsó tagja megjelöli a tulajdonneveket, a személy-, hely- és intézményneveket.

A szótövek meghatározására alkalmas, így a főnévi csoportok vektorizálása előtt a szavakat tövesíteni tudjuk, így még jobb egyezést kimutatva más főnévi csoportokkal ahol ugyanazon igék például múlt idejű formáját használjuk.

1.4. Az e-magyar rendszer szövegfeldolgozó eszközlánc

Az e-magyar rendszer elérhető docker file-ként, ami egy virtuális gépnek tekinthető, így nincs dolgunk az egymásra épülő modulok telepítésével, egy inputot megadva, megkapjuk a szöveg teljes elemzését.

A kódrészlet egy fájlba írja az e-magyar eszközlánc outputját.

```
def docker_to_file(in_file, out_file):
    task = subprocess.Popen("cat %s | docker run -i mtaril/emtsv
tok,spell,morph,pos,conv-morph,dep,chunk,ner" % in_file, shell=True,
stdout=subprocess.PIPE)
    with open('%s' % out_file, 'w') as f:
        for item in task.stdout.readlines():
            f.write("%s\n" % item.decode('utf-8'))
```

A szótövesítés és a Noun Phrase chunking funkciót az outputban a mellékletben található képen [2] látható formátumban találjuk meg. Az első két sor outputot lerövidítettem, hogy lényegretörő legyen.

A “stem” illetve “lemma” kifejezések után szerepel a szótő. A “stem” utáni szótő a szövegfeldolgozás egyértelműsítési szakaszában kerül kiértékeléshez, tehát olyan többértelmű szavak esetében ahol a szövegkörnyezet alapján a rendszer nem tud egyértelműsíteni, csak a “lemma” kifejezés után találunk felsorolt lehetséges szótöveket.

Az utolsó előtti kifejezések pl.: B-NP, I-NP, E-NP, 1-NP, O megmutatják a szavak főnévi csoportbeli viszonyaikat. Ezek jelentését az alábbi e-magyar.hu-n szereplő példával szemléltetem.

https://e-magyar.hu/hu/textmodules/emchunk

1.5. emChunk - főnévi csoport (frázis) felismerő

Mire jó? Mit csinál?

Az emChunk modul a szövegben a maximális NP-ket azonosítja, vagyis olyan NP-ket, melyek nem részei egy magasabb szintű NP-nek sem.

Mi a bemenet?

Az elemzőlánc előző szintjein feldolgozott magyar nyelvű szövegekkel dolgozik, amelyek már i) szavakra és mondatokra vannak bontva, ii) a szavakhoz

hozzá van rendelve a teljes morfológiai elemzésük. A chunker modul hatékony működéséhez szükségesek ezek az információk.

Mi a kimenet?

A modul a szavakra és mondatokra bontott szövegben minden tokenhez hozzárendel egy címkét, amely azt jelöli, hogy az adott szó i) eleme-e egy maximális főnévi frázisnak, ha igen, akkor ii) egy- vagy többelemű-e, ha ez utóbbi, akkor iii) a frázis kezdő, közbülső vagy záró eleme-e. A kimenetben az előző szintek elemzése is megmarad, és a chunker modul is hozzáteszi a saját címkéit.

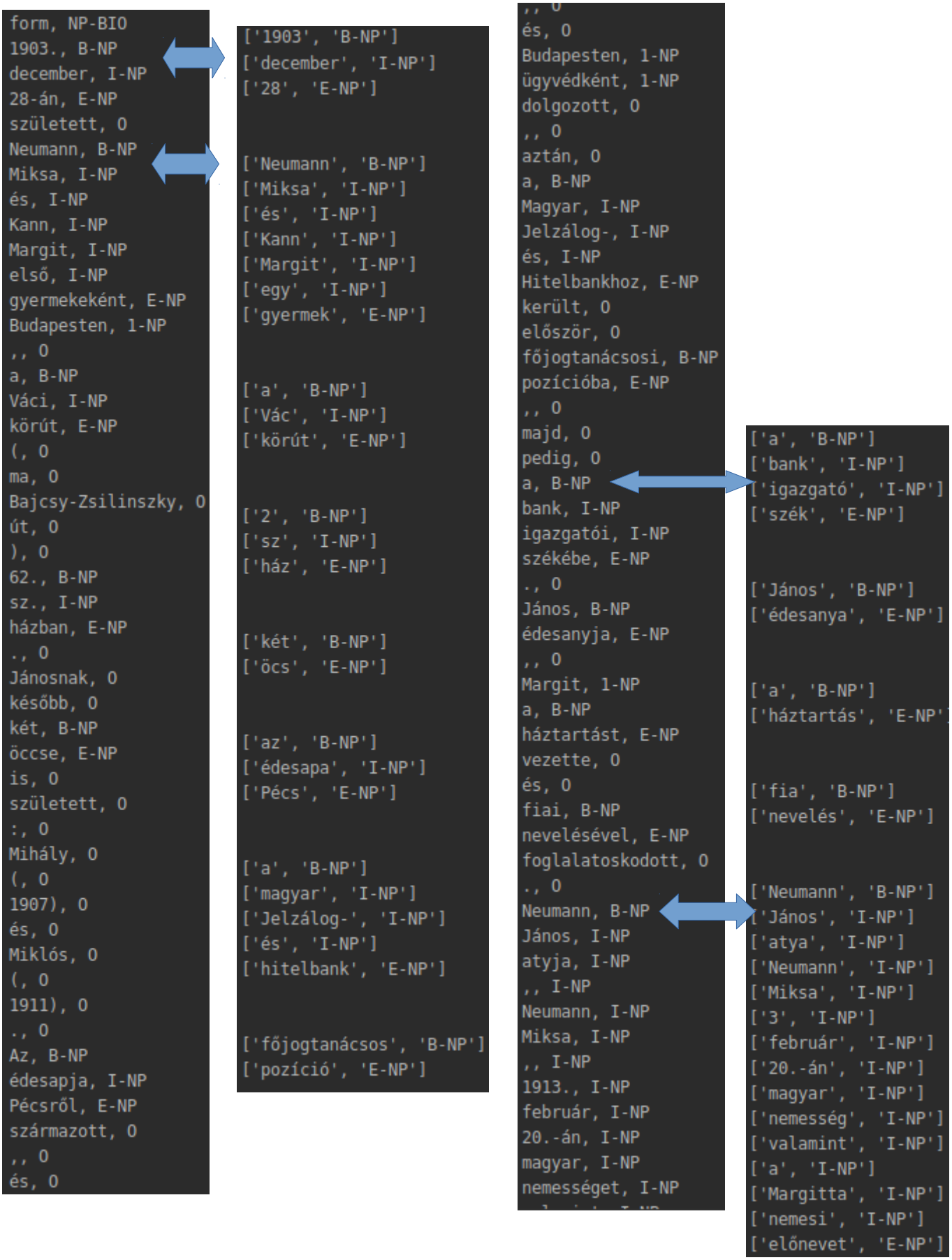
Egy példa a működésre.

A példamondatban két maximális NP-t és két O-val jelölt elemet találunk, ez utóbbiak nem NP-k. A 'B' jelöli a frázisok kezdetét, az 'E' a frázisok végét, az 'I' pedig azt, hogy az adott token a frázis közbülső eleme.

A szállásunk egy Balaton melletti kis üdülőfaluban, Zamárdiban volt.

A	B-NP
szállásunk	E-NP
egy	B-NP
Balaton	I-NP
melletti	I-NP
kis	I-NP
üdülőfaluban	I-NP
,	I-NP
Zamárdiban	E-NP
volt	O
.	O

Reguláris kifejezésekkel a főnévi csoportbeli viszonyok kigyűjtése:



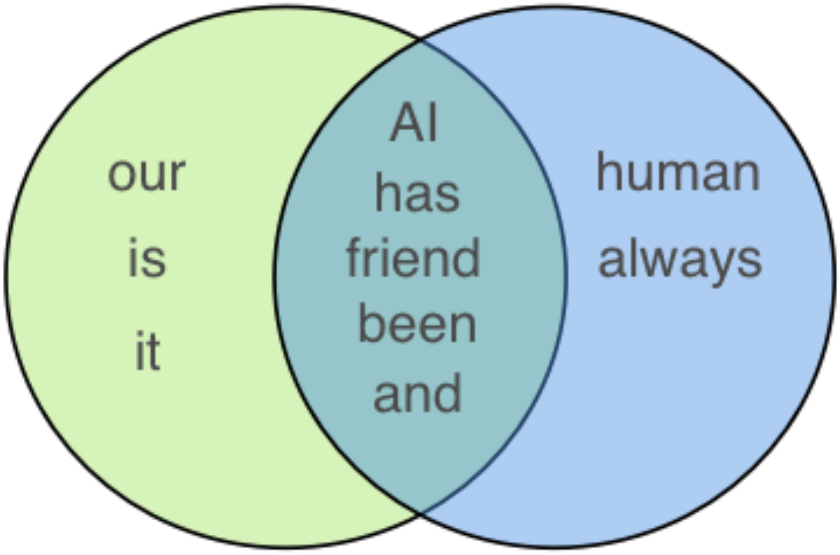
1.6. Jaccard hasonlóság:

A Jaccard hasonlóságot úgy határozzuk meg, hogy a metszet méretét elosztjuk a két halmaz uniójának méretével. Vegyük például az alábbi két mondatot:

AI is our friend and it has been friendly
AI and humans have always been friendly
forrás: https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50

Annak érdekében, hogy a hasonlóság a Jaccard-hasonlóság alapján kiszámítható legyen, először lemmatizálást végezzünk, hogy a szavakat ugyanazon gyökérszóra redukáljuk. Esetünkben a „friend” és a „friendly” egyaránt „friend”.

Rajzoljunk Venn diagramot a kapott két mondatról:



A példamondatok szavai Venn diagrammon [3]

A fenti két mondathoz ezt a Jaccard hasonlóságot kapjuk: 5 / (5 + 3 + 2) = 0,5, amely a halmaz metszetének mérete osztva a két halmaz uniójának méretével.

Hátránya: Ha például az első mondatban a “friend” szó 10-szer többször szerepelne, a metrika változatlan maradna, ez a tulajdonsága miatt a Jaccard hasonlóságot használva nagyban figyelmen kívűl hagynánk a szerző fogalmazásmódját, a szóismétlési szokásait.

A dokumentumok szerző alapján történő kategorizálásra kevésbé alkalmas, viszont alkalmas például termékleírások összehasonlítására, mert abban az esetben a ismétlődő szavak gyakoriságának figyelmen kívűl hagyása nem vonnak le a szövegek hatékony összehasonlításából, ahol csupán a szöveg tartalmán van a lényeg.

Egy további módszer a hasonlóság kimutatására a dokumentumokban , hogy megszámoljuk a szavak előfordulásainak számát, azokat a szavakét amelyek mindkét dokumentumban megtalálhatók, így ha például a ‘számítógép’ szó 10-szer fordul elő az 1-es dokumentumban, 13-szor a 2-es dokumentumban és 4-szer a 3-as dokumentumban, akkor azt mondhatjuk, hogy a kettes dokumentum több hasonlóságot mutat az egyes dokumentummal mint a hármas.

A hátránya hogy ahogy a dokumentumok hossza nő, úgy a közös szavak száma is, még ha a szövegek témája el is tér, már csak a kötőszavak(pl.: akár, azért hogy, és, illetve, vagy….) és névelők(a,az,egy...) miatt is.

1.7. Koszinus hasonlóság

A koszinusz hasonlóság alkalmazásánál az előbbi bekezdésben említett hátrány nem áll fenn.

Ez a metrika a következő képpen számolható ki:

$$Cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\,\|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2}\,\sqrt{\sum_1^n b_i^2}} \qquad \vec{a}\cdot\vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

forrás: https://en.wikipedia.org/wiki/Cosine_similarity

Azaz **a két vektor által bezárt szög koszinusza.** Ezt megkapjuk, hogy ha a skaláris szorzatukat elosztjuk a két vektor hosszainak szorzatával. vektorok koordinátáit a hosszukkal leosztva kapjuk meg ezt az alakot.

Háromdimenziós vektorok esetén szemléletesen lehet ábrázolni a metrikát

A példa során során, fontos, hogy olyan szövegeket tekintsük amelyek fő témája eltér, de hasonlóságot mutat szóhasználatban. A Wikipediáról kiragadott sorfolytonos részleteket tekintsük.

Az “A” szöveg a Kovács Győző informatikusról szóló Wikipedia cikk. *https://hu.wikipedia.org/wiki/Kovács_Győző_(informatikus)*

A “C” szöveg pedig a Neumann Jánosról szóló Wikipedia cikk. https://hu.wikipedia.org/wiki/Neumann_János

A “B” szöveg pedig egy kisebb részlet a Neumann Jánosról szóló Wikipedia cikkből.

A Kovács Győzőről szóló cikk tematikailag nagyon hasonló a Neumann Jánosról szóló cikkhez, az alábbi Kovács Győzőt bemutató idézet is alátámasztja ezt:

> *"Kovács Győző magyar villamosmérnök, számítástechnikus, informatikus, az informatikai kultúra jeles terjesztője"*
>
> *"Neumann János életének és munkásságának legalaposabb ismerője és emlékének ápolója. A Neumann centenáriumi év (2003) fő szervezője, a jubileumra létrehozott kiállítás (100 éve született Neumann János, Természettudományi Múzeum, 2003) szakmai vezetője."*

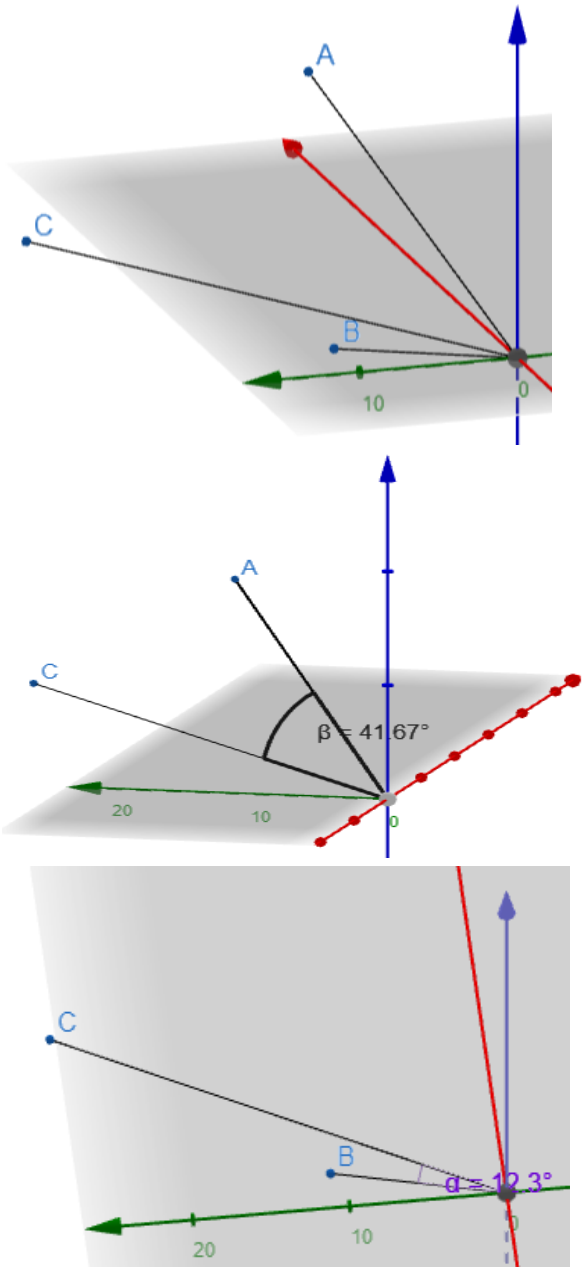
(forrás: Wikipedia)

A metrika tesztelésére alkalmasok a szövegek mivel, a közös szavak megszámolásával arra jutunk, hogy a Neumann János-ről szóló "B" szöveg nagyobb hasonlóságot mutat a Kovács Győzőről szóló "A" cikkel mint a saját magából kiragadott részlettel ,a "C" szöveggel, mivel a "B" és "A" cikk terjedelemben jobban közelítenek egymáshoz így már csak a kötőszavak(és, illetve, vagy, aztán pedig, és, is….) és névelők(a,az,egy...) miatt is.

Kovács	876 szó	(A)
Neumann_Kicsi	988 szó	(B)
Neumann	3487 szó	(C)

Cikk	szavak száma	„magyar"	„Neumann"	„Kovács"
Kovács (A)	876	16	13	21
Neumann_Kicsi (B)	988	7	24	0
Neumann (C)	3487	21	82	7

A koordináta rendszer X,Y,Z tengelye a „magyar", „Neumann", „Kovács" szavak számát reprezentálja. A szövegeket így e három szó gyakoriságával reprezentával, vektorokként írhatjuk le.



$BC \sphericalangle = \alpha = 12.3°$ $AC \sphericalangle = \beta = 41.67°$

$\cos(\alpha) > \cos(\beta)$

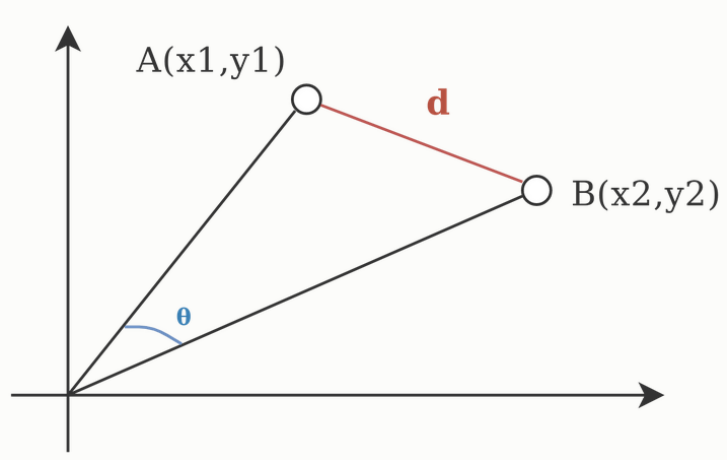
Az α szög kisebb mint a β, ezzel a metrika azt mutatja, hogy C és B szöveg között több a hasonlóság mint C és A között.

A koszinusz hasonlóság alkalmas a szövegek közötti hasonlóságra egy metrikát adni, ami nem függ a szövegek hosszától.

1.8. Euklideszi távolság

Az Euklideszi távolság a szöveget hasonlóan a koszinusz hasonlósághoz, vektorokként reprezántalja.

Az ábrán a “d” távolság az euklideszi távolság:



A metrika ebben az esetben a vektorok végpontjainak távolsága ami így függ a szöveg hosszától, ugyanis számításban veszi a vektorok hosszát is nem csak az egymással bezárt szögeket.

Az szöveg összehasonlítási módszerek közül, a koszinusz hasonlóság implementációja tűnik a leginkább célravezetőbbnek összehasonlítva a többi metrikával.

Példa a dokumentum vektorokká alakítására

forrás: [4]

A koszinusz hasonlóság alkalmazhatóság feltétele ,hogy a szövegeket vektorizálni tudjuk.

A “**szöveg”, “dokumentum”, “szókincs”** szavak a példa kontextusában értendő kifejezések.

Nézzük az alábbi két szöveget.

```
1. "John likes to watch movies. Mary likes movies too."
```

```
2. "John also likes to watch football games."
```

A szövegek szavak listájaként is reprezentálhatók.

```
1. ['John', 'likes', 'to', 'watch', 'movies.', 'Mary', 'likes', 'movies', 'too.']
```

```
2. ['John', 'also', 'likes', 'to', 'watch', 'football', 'games']
```

Távolítsuk el az ismétlödő szavakat és használjuk ezen szavak gyakoriságát az ismétlődések kifejezésére.

```
1. {"John":1,"likes":2,"to":1,"watch":1,"movies":2,"Mary":1,"too":1}
```

```
2. {"John":1,"also":1,"likes":1,"to":1,"watch":1,"football":1,   "games":1}
```

Feltéve, hogy a két szöveg egy dokumentumban van, alább látható a szavak gyakorisága az egész dokumentumra. Mindkét szöveget számításba véve. Program a látható:

```
 {"John":2,"likes":3,"to":2,"watch":2,"movies":2,"Mary":1,"too":1,  "also":1,"football":1,"games":1}
```

A kettő szövegből álló dokumentumban 10 egyedi szó van. Ezt a gyűjteményt nevezzük a dokumentum **szókincsének**.

Mondathatárokat használva, a mondatokat külön külön vektorizáljuk. A vektor dimenziója egyenlő lesz a **szókincs** elemeinek számával Ez esetben 10. A vektorokat [0,0,0,0,0,0,0,0,0,0] ként inicializáljuk. Ezután végig iterálunk a mondat szavain és ha egyezik a szó a szókincs elemével a megfelelő indexű elemét a vektornak növeljük 1-el. Így a két szöveg vektorizálva:

```
John likes to watch movies. Mary likes movies too.[1, 2, 1, 1, 2, 1, 1, 0, 0, 0]
```

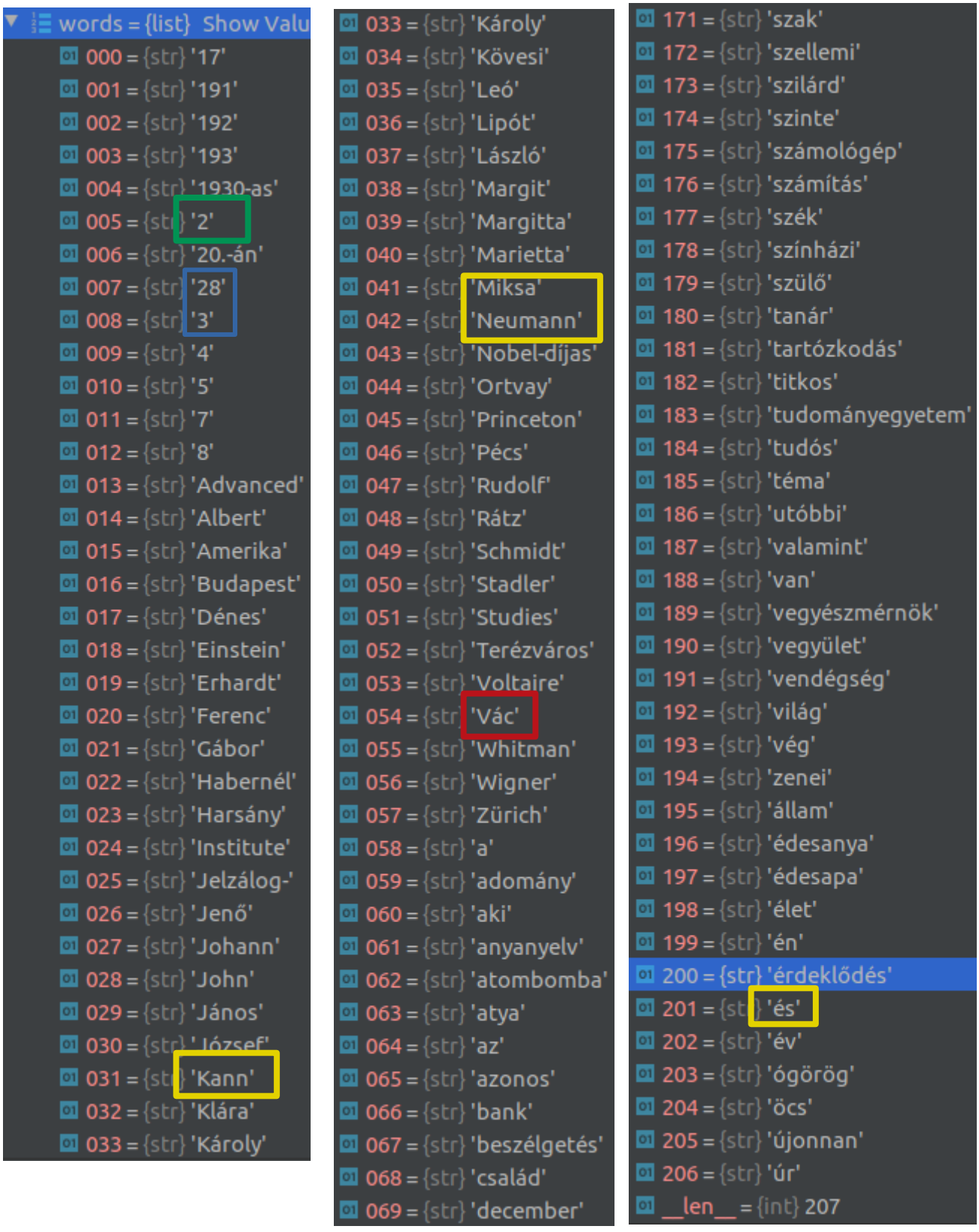
```
John also likes to watch football games.[1, 1, 1, 1, 0, 0, 0, 1, 1, 1]
```

Az outputból hasznosítva a szótő felismerést és a központozás felismerést, kigyűjtöttem reguláris kifejezésekkel a főnévi csoportokat, utólag a központozást eltávolítottam, az alábbi algoritmussal:

```
def generate_3d_array(out_file):
    arrayNPext = [] #Belső listái tartalmazzak a főnévi csoportokat
    with open('%s' % out_file) as f:
        sor = f.readline()
        arrayNPint = [] #Egy darab főnévi csoportot tartalmaz
        #A dokumentum e-magyar eszközlánc általi outputján soronként
iterálunk végig
        while sor:
            patternStem = re.compile('^(.*?)\s.*"stem": \["([1-9a-zA-
ZÁÉÍÓÖŐÚÜŰáéíóöőúü
            #a reguláris kifejezés egyezőségét tartalmazó boolean
            resultStem = patternStem.match(sor)
            ……
            #további reguláris kifejezések
            ……
            NP = ""
            sorArray = []
            #megvizsgáljuk, melyik reguláris kifejezés illik az adott
sorra és a kívánt          #egyezőségi csoportot
            # azaz a szótövet és a főnévi csoportban elfoglalt helyet
eltároljuk
            if resultStem:
                sorArray.extend([resultStem.group(2),
resultStem.group(3)])
                NP = resultStem.group(3)
            elif resultLemma:
                ……
            elif resultRaw:
                ……
            elif resultPunct:
                ……
            else:
                sor = f.readline()
                continue
            if NP == "B-NP":
                if arrayNPint.__len__() > 0:
                    arrayNPext.append(arrayNPint)
                    arrayNPint = []
                arrayNPint.append(sorArray)
            elif NP == "I-NP" or NP == "E-NP":
                i = len(arrayNPint)-1
                while i >= 0:
                    if arrayNPint[i][1] == "I-NP":
                        i = i-1
                        continue
                    elif arrayNPint[i][1] == "B-NP":
                        arrayNPint.append(sorArray)
                        break
            sor = f.readline()
    #központozás kiszűrése
    arrayNPextNonPunct = []
    patternNonPunct = re.compile('[1-9a-zA-ZÁÉÍÓÖŐÚÜŰáéíóöőúüű\-]+')
    for i in arrayNPext:
        arrayNPintNonPunct = []
        for j in i:
            if patternNonPunct.match(j[0]):
                arrayNPintNonPunct.append(j[0])
                print(j)
```

```
        print('\n')
        arrayNPextNonPunct.append(arrayNPintNonPunct)
    return arrayNPextNonPunct
```

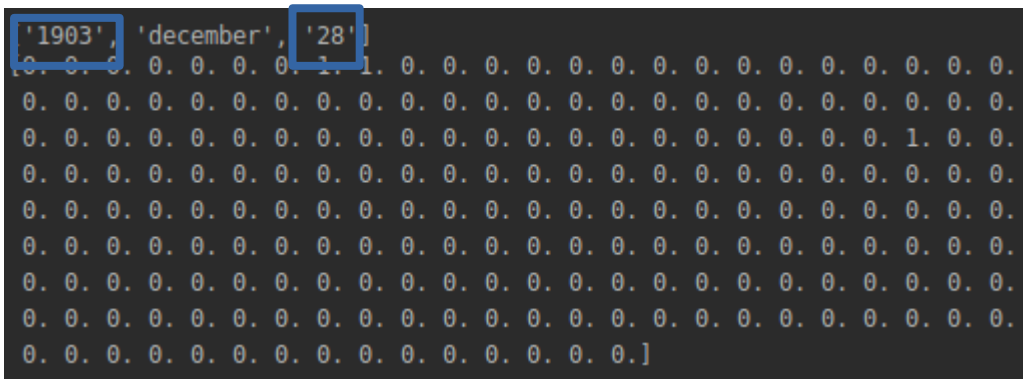
A következő lépésben vektorizálom ezeket a főnévi csoportokat, ehhez szükséges, hogy kigyűjtsük a dokumentum szókincsét. A szókincsből a duplikátumokat kiszűrtem és ábécé sorrendbe rendeztem.

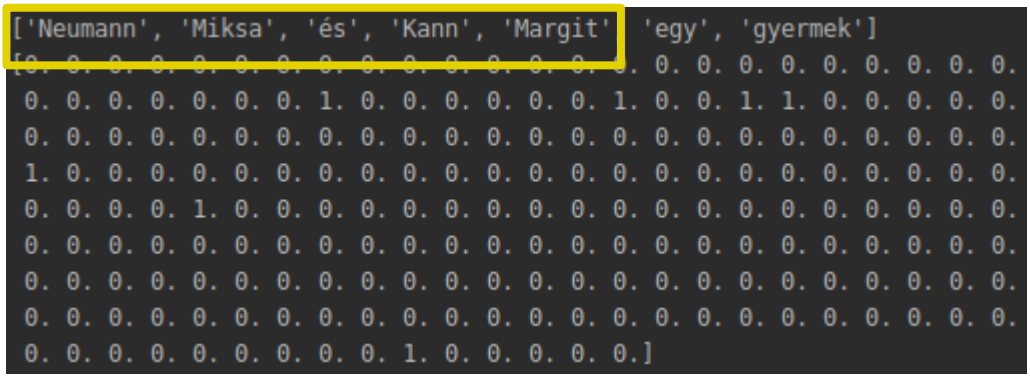


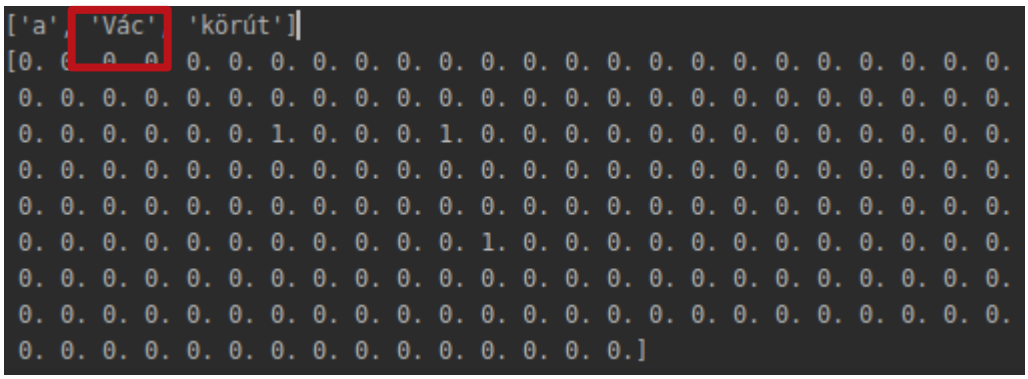
A dokumentum szókincse 207 elemű, ezért minden főnévi csoportot egy 207 elemű vektorként fogunk ábrázolni. A dokumentumot reprezentálni lehet annyi vektorral amennyi főnévi csoport van, a 207 elemű vektorok gyűjteménye reprezentálja a dokumentumot.

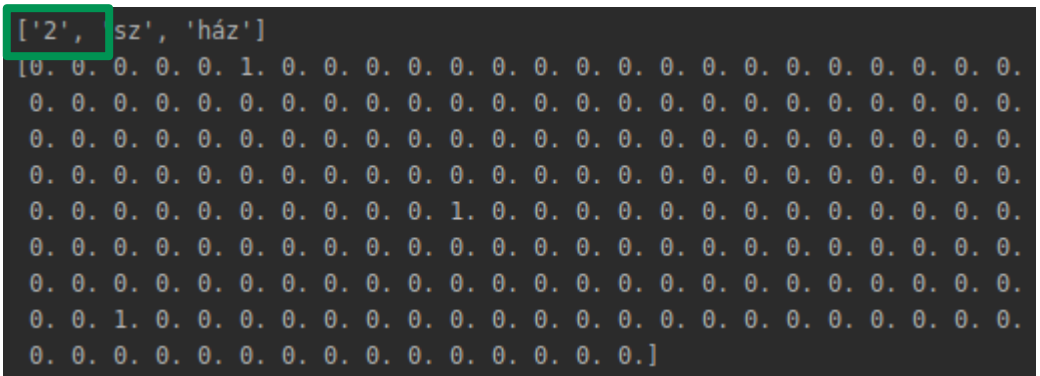
A emagyar.hu rendszer “1903.” esetén “3” ra vágja le, ez az emagyar.hu rendszer hibája, főleg 0-t tartalmazó évszámoknál és kötőjeles és toldalékos évszámoknál sokszor hibás eredményt kapunk.

A vektorizálás eredménye:









1.9. Koszinusz hasonlóság tesztelése

A dokumentum összehasonlítása más dokumentumokból abból áll, hogy az első dokumentum szókincsét használjuk fel a második dokumentum vektorizálásához, majd az első dokumentum vektorain végig iterálva, minden egyes vektorral a második dokumentumból elvégezzük a koszinus hasonlóság metrika meghatározását. Majd vesszük a maximumot, majd a következő iterációs lépés következik az első dokumentum vektorainak listáján, ezután ugyanúgy eltároljuk a maximumot. A maximumokat összeadva kapom meg a két dokumentum hasonlóságának értékét. Ez az érték egy abszolút hasonlóságot ad meg, azaz csak más összehasonlítások végeredményeivel vethetők össze.

```
bow =
generate_bow_matrix(generate_bow(generate_3d_array('emChunker.txt')),
generate_3d_array('emChunker.txt'))

bow1 =
generate_bow_matrix(generate_bow(generate_3d_array('emChunker.txt')),
generate_3d_array('emChunker1.txt'))

bow2 =
generate_bow_matrix(generate_bow(generate_3d_array('emChunker.txt')),
generate_3d_array('emChunker2.txt'))

from numpy import dot
from numpy.linalg import norm
#compare bag_of_words
def compare_bows(bow_1, bow_2):
    sum_of_max = 0
    for vector_b1 in bow_1:
        max_cos_sim = 0
        for vector_b2 in bow_2:
            if numpy.count_nonzero(vector_b1) == 0 or
numpy.count_nonzero(vector_b2) == 0:
                continue
            cos_sim = dot(vector_b1, vector_b2) / (norm(vector_b1) *
norm(vector_b2))
            if max_cos_sim < cos_sim:
                max_cos_sim = cos_sim
        sum_of_max += max_cos_sim
    return sum_of_max
#Teszt
print(compare_bows(bow, bow1)) 52.6676332569895
print(compare_bows(bow, bow2)) 27.56308450766862
```

Nézzük az eredményeket a dokumentumokra a vektorizálási példából:

Az “A” szöveg a Kovács Győző informatikusról szóló Wikipedia cikk. *https://hu.wikipedia.org/wiki/Kovács_Győző_(informatikus)*
A “C” szöveg pedig a Neumann Jánosról szóló Wikipedia cikk. https://hu.wikipedia.org/wiki/Neumann_János

A “B” szöveg pedig egy kisebb részlet a Neumann Jánosról szóló Wikipedia cikkből.

C dokumentummal összevetve az A: 27.563084507668627

C dokumentummal összevetve a B: 52.6676332569895

Mivel a második esetben a metrika nagyobb (52>27) ez azt jelenti, hogy összességében a koszinusz hasonlóságok nagyobbak voltak, azaz a vektorok egymással bezárt szöge kisebbek, így a B dokumentum jobban hasonlít a C dokumentumra mint az A dokumentum a C dokumentumra.

1.10. Jaccard hasonlóság tesztelése

Az e-magyar rendszer szótövesítését felhasználva, a Jaccard hasonlóság eredménye:

C és A Jaccard hasonlósága: 0.8

C és B Jaccard hasonlósága: 0.8888888888888888

A Jaccard hasonlóság implementációja Python-ban:

```
def intersection(lst1, lst2):
    return list(set(lst1) & set(lst2))

def union(lst1, lst2):
    return list(set(lst1) | set(lst2))

def generate_array(out_file):
    with open('%s' % out_file) as f:
        sor = f.readline()
        arrayStem = []
        while sor: #A dokumentum e-magyar eszközlánc általi outputján
soronként                       #iterálunk végig

            patternStem = re.compile('^(.*?)\s.*"stem": \["([1-9a-zA-
ZÁÉÍÓÖŐÚÜŰáéíóöőúüű]+?)".*\s([^\s]+?)\s[^\s]+$')

            resultStem = patternStem.match(sor) #a reguláris kifejezés
egyezőségét                                                 #tartalmazó
boolean

            patternLemma = re.compile('^(.*?)\s.*"lemma": "([1-9a-zA-
ZÁÉÍÓÖŐÚÜŰáéíóöőúüű]+?)".*\s([^\s]+?)\s[^\s]+$')

            resultLemma = patternLemma.match(sor)
            patternRaw = re.compile('^([1-9a-zA-ZÁÉÍÓÖŐÚÜŰáéíóöőúüű\-]
+?)\s.*\s([^\s]+?)\s[^\s]+$')

            resultRaw = patternRaw.match(sor)
            if resultStem:
                arrayStem.extend(resultStem.group(2))
            elif resultLemma:
                arrayStem.extend(resultLemma.group(2))
            elif resultRaw:
                arrayStem.extend(resultRaw.group(1))
            else:
```

```
                sor = f.readline()
                continue
            sor = f.readline()
    return arrayStem

def jaccard_sim(out_file1, out_file2):
    i = intersection(generate_array(out_file1),
generate_array(out_file2)).__len__()
    u = union(generate_array(out_file1),
generate_array(out_file2)).__len__()
    return i/u

print(jaccard_sim('emChunker.txt', 'emChunker1.txt')) 0.8888888888888888
print(jaccard_sim('emChunker1.txt', 'emChunker2.txt')) 0.8
```

Látszik, hogy a metrika most is a C és B dokumentum hasonlóságát értékeli nagyobbra, viszont a számok között nincs akkora eltérés mint a koszinusz hasonlóság esetében, sőt igen csekély.

1.11. Euklideszi távolság tesztelése

A vektorok távolsága kiszámítható a pitagorasz tételből származtatva:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

```
from scipy.spatial import distance
distance.euclidean()
```

A képletet implementálja a distance.euclideon() függvény a scipy könyvtárból.

A koszinusz hasonlóság esetéhez hasonlóan programoztam:

A dokumentum összehasonlítása más dokumentumokból abból áll, hogy az első dokumentum szókincsét használjuk fel a második dokumentum vektorizálásához, majd az első dokumentum vektorain végig iterálva, minden egyes vektorral a második dokumentumból elvégezzük a **euklideszi távolság** metrika meghatározását. Majd vesszük a minimumot, majd a következő iterációs lépés következik az első dokumentum vektorainak listáján, ezután ugyanúgy eltároljuk a minimumot. A minimumokat összeadva kapom meg a két dokumentum hasonlóságának értékét. Ez az érték egy abszolút hasonlóságot ad meg, azaz csak más összehasonlítások végeredményeivel vethetők össze.

C dokumentummal összevetve az A: 57.72905056694958

C dokumentummal összevetve a B: 131.12723918223597

Mivel a második esetben a metrika nagyobb (131>57) ez azt jelenti, hogy összességében a euklideszi távolságok nagyobbak voltak, azaz a vektorok távolsága nagyobbak, így az A dokumentum jobban hasonlít a C dokumentumra mint a B dokumentum a C dokumentumra.

Az euklideszi hasonlóság Pythonban Noun Phrase darabolással:

```
def compare_bows_eucl(bow_1, bow_2):
    sum_of_min = 0
    for vector_b1 in bow_1:
        min_eucl_sim = 99999999
        for vector_b2 in bow_2:
            if numpy.count_nonzero(vector_b1) == 0 or
numpy.count_nonzero(vector_b2) == 0:
                continue
            eucl_sim = distance.euclidean(vector_b1, vector_b2)
            if min_eucl_sim > eucl_sim:
                min_eucl_sim = eucl_sim
        if min_eucl_sim != 99999999:
            sum_of_min += min_eucl_sim
    return sum_of_min
```

Az egyes hasonlósági metrikák eredményei összefoglalva:

	Koszinusz hasonlóság	Euklideszi távolság	Jaccard hasonlóság
C dokumentummal összevetve az A	27.563084507668627	57.729050566949578	0.8
C dokumentummal összevetve a B	52.6676332569895	131.12723918223597	0.8888888888888880

1.12. A módszerek alkamazása a weboldalak közötti hasonlóság mértékének megállapítására

A kielemzésre és összehasonlításra szánt weboldalak egy webcrawler általi mentés. A webcrawler egy weboldalról kiindulva bejárja annak aloldalait a linkeket követve egy bizonyos mélységig és ezt a fordított fastrukturát menti le.

Így több mint 5 ezer weboldalt kaptam kielemzésre.

A weboldalakat plaintext-é alakítottam majd az e-magyar eszközlánccal a főnévi csoportok kigíűjtéséhez szükséges nyelvtani elemzést futattam le, az eredményeket weboldalanként külön fájlba írtam ki a meghajtóra. A html fájlok neve 1-5257-ig terjedő számok. Ugyanígy neveztem el a plaintext-é alakított fájlokat és az e-magyar rendszer eredményeit és más más mappába írtam ki ezeket a szöveges fájlokat. Ez az eljárás körülbelül öt napot vett igénybe mire minden weboldal nyelvtani kielemzése megtörtént. Az SSD meghajtóm elhasználódása miatt, a folyamat többször megakadt. Lehetővé kellett tenni, hogy ott folytassa a kód a kielemzést ahol abbahagyta. Azon fájlok amik a folyamat megakadásakor kerültek kielemzésre újboli kielemzésre szorultak.

```
import os
directory_in_strcucc = "cucc"
directorycucc = os.fsencode(directory_in_strcucc)
directory_in_strout = "output"
directoryout = os.fsencode(directory_in_strout)
redo =[]
#[5295,5286,5336,3522,5307,5309,2502,5308,3712,5344,5317,4939,5292,3517,55
49,5339,5335,2365,5332,5323,2722,2546,2545,5349,5557,5296,3758,3677,5348,2
354,5315,2527,5331,5338,5009,5329,5347,351,3829,2361,3681,5326,5346,5322,5
320,5293,5285,5300,5297,2195,5321,5305,3664,3686,5310,3711,5535,2921,5301,
2406,2922,4039,5306,570,5289,5327,5334,598,5294,4396]
output_files = [os.fsdecode(h) for h in os.listdir(directoryout)]
for file in os.listdir(directorycucc):
    filename = os.fsdecode(file)
    #print("%s" % filename)
    if filename in redo:
        f = codecs.open("cucc/%s" % filename, 'r')
        raw = get_text_bs(f)
        out_file = "plaintext/%s" % filename
        with open('%s' % out_file, 'w') as g:
            g.write(raw)
        docker_to_file("plaintext/%s" % filename, "output/%s" % filename)
        print(filename)
    elif filename not in output_files:
        f = codecs.open("cucc/%s" % filename, 'r')
        raw = get_text_bs(f)
        out_file = "plaintext/%s" % filename
        with open('%s' % out_file, 'w') as g:
                g.write(raw)
        docker_to_file("plaintext/%s" % filename, "output/%s" % filename)
```

A lementett weboldalak plaintext-é alakítását és ebből az e-magyar rendszer általi nyelvtani elemzést végző függvény

5257 darab html fájl, összesen 373 MB került kielemzésre a fenti függvény által. A plaintext -é konvertálás után 95,1 MB -ot foglal el az eredmény, ennek a nyelvtani elemzése pedig 1,4 GB -ot.

A három hasonlósági metrika kiszámítására használt algoritmus lefut 0,05-0,3 másodperc alatt két dokumentum összehasonlítása esetén. Viszont 5257 darab dokumentum esetén $\frac{5257 \cdot 5256}{2!} = \binom{5257}{2} = 13\,815396 \approx 1{,}36 \cdot 10^7$ darab összehasonlítás szükséges.

Az időigényesség felmérése után a metrikák kiszámításához a programot egy Amazon EC2 szolgáltatáson keresztül bérelhető z1d [5] típusú virtuális gépen futtattam meg. A z1d típus magas órajelű és nagy egymagos teljesítményű valamint SSD-vel szerelt.

A metrikakat egy szöveges fájlba írattam ki, minden sor egy összehasonlítás eredménye. Az ismétlődő összehasonlításokat elkerültem egy lista feltöltésével. Az összehasonlítások eredményét a következő formátumban írattam ki soronként:

```
[fájl1],[fájl2],[koszinusz hasonlóság],[jaccard hasonlóság],[euklideszi távolság]
```

Azon olyan fájlokat amik 0 Byte és 89 Byte méret között voltak kihagytam az összehasonlításból, ezeket a fájlokat szúrópróba szerűen átnézve azt találtam, hogy nagyon kisméretű weboldalak feldolgozásának az eredményei amikben nincs emberi szerzőtől származó szöveg. Ezeket a fájlokat egy shell paranccsal listáztam ki olyan formátumba, hogy pythonba lista definiálásra alkalmas legyen. A shell parancs a következő:

```
find ~/Pycharmm/emts/output/ -type f -size +0b -size -89b -exec ls -l {} + | sed  -r 's@^.*/([0-9]{1,4})$@\1@' | awk -vORS=, '{print $1}'
```

Az eredmény több mint 900 kihagyható fájl. Így is egy iteráció, azaz egy fájl összehasonlítása fennmaradó összes többivel 5 percet vett igénybe, így több mint 18 nap lenne a teljes lefutás ideje, ez 0,2 dollár per óra bérlési díjnál több mint 86 dollárt jelent. 142 fájl kielemzése után állítottam le a program futását.

```
voltak = []
tulkicsik =
[154,1744,2163,2169,2175,2180,2195,2207,2329,2338,2349,2354,2361,2365,2384
,2398,2401,2406,2411,2414,2418,2424,2430,2452,2453,2454,2462,2469,2470,247
1,2490,2500,2502,2527,253,2545,2546,2590,2591,2659,2661,2685,2693,2701,270
7,2709,271,2711,2712,2713,2714,2715,2716,2717,2718,2719,272,2720,2721,2722
,2734,2796,2813,282,2820,287,291,2921,2922,2923,2927,2955,29
#find ~/Pycharmm/emts/output/ -type f -size +0b -size -89b -exec ls -l {}
+ | sed  -r 's@^.*/([0-9]{1,4})$@\1@' | awk -vORS=, '{print $1}'
metrikafile = "metrikak/metrikak"
with open('%s' % metrikafile, 'w') as m:
    for file in os.listdir(directoryout):
        filename = os.fsdecode(file)
        f = codecs.open("output/%s" % filename, 'r')
        if filename in tulkicsik:
            continue
        if generate_array(f.name).__len__() < 1:
            continue
        print(f.name)
        bow =
generate_bow_matrix(generate_bow(generate_3d_array(f.name)),
generate_3d_array(f.name))
        for bfile in os.listdir(directoryout):
            bfilename = os.fsdecode(bfile)
            bf = codecs.open("output/%s" % bfilename, 'r')
            if bf.name not in voltak:
                bbow =
generate_bow_matrix(generate_bow(generate_3d_array(f.name)),
generate_3d_array(bf.name))

                m.write("%s,%s,%f,%f,%f" %
(filename,bfilename,compare_bows(bow, bbow),jaccard_sim(f.name,
bf.name),compare_bows_eucl(bow, bbow)))
                m.write("\r\n")
        voltak.append(f.name)
```

Az összehasonlítások eredményét tartalmazó szöveges fájlt egy MariaDB adatbázis táblába importáltam a statisztikák egyszerű lekérdezéséhez. Létrehoztam egy adatbázist, majd egy táblát és importáltam bele a vesszővel elválasztott sorokat a szöveges fájlból.

```
CREATE DATABASE DB1;
USE DB1;
CREATE TABLE kiertekeles(int fajl1,int fajl2,float
cos,float jaccard,float eucl);
load data local infile 'file.csv' into table table
fields terminated by ','  enclosed by '"'  lines
terminated by '\n'  lines terminated by '\n';
```

A futási eredmény példájaként nézzük meg a 4490-es weboldal statisztikáit. Kilistáztam azokat a rekordokat ahol a 4490-es fájlt összehasonlítjuk egy másikkal és ezt Jaccard hasonlóság szerint csökkenő sorrendbe rendezzük, majd másodlagosan koszinusz hasonlóság szerint csökkenő sorrendbe. Ebből meg tudjuk, hogy a Jaccard hasonlóságot figyelembe véve, melyik weboldalak hasonlóak a 4490-es weboldalhoz.

```
SELECT fajl1,fajl2,cos,eucl,jaccard FROM kiertekeles
WHEREjaccard <= (SELECT MAX(k1.jaccard) FROM
kiertekeles k1 WHERE k1.fajl1 <> k1.fajl2 AND
fajl1='4490') AND fajl1 = '4490' ORDER BY jaccard
DESC,cos DESC LIMIT 28;
```

fajl1	fajl2	cos	eucl	jaccard
4490	4490	509	0	1
4490	4408	508.897	2	1
4490	3432	508.791	1.73205	1
4490	4008	508.791	1.73205	1
4490	3494	508.699	3	1
4490	2571	508.699	3	1
4490	3536	508.699	3	1
4490	4075	508.333	3.73205	1
4490	4459	506.125	10.0246	1
4490	3356	506.072	11.9428	1
4490	4443	505.416	13.1708	1
4490	4101	505.3	12.9282	1
4490	2686	504.535	15.121	1
4490	2484	504.535	15.121	1
4490	3030	503.925	14.8631	1
4490	2599	503.801	15.2818	1
4490	2703	503.583	19.6062	1
4490	4568	503.583	19.6062	1
4490	3381	503.116	20.5099	1
4490	3154	503.073	20.5099	1
4490	3349	502.911	21.2772	1
4490	289	502.894	20.9594	1
4490	2806	502.817	20.9594	1
4490	275	502.817	20.9594	1
4490	2669	502.719	21.4236	1
4490	2506	501.817	22.8277	1
4490	3642	501.685	22.4967	1
4490	4473	509	0	0.983333

Nézzük meg ugyanezt a Jaccard hasonlóság helyett koszinusz hasonlóságra. Ebből meg tudhatjuk, hogy melyik webolaldalak hasonlóak a 4490-es weboldalhoz a koszinusz hasonlóság alapján.

```
SELECT fajl1,fajl2,cos,eucl,jaccard FROM kiertekeles
WHERE cos <= (SELECT MAX(k1.cos) FROM kiertekeles k1
WHERE k1.fajl1 <> k1.fajl2 AND fajl1='4490') AND fajl1
= '4490' ORDER BY cos DESC LIMIT 28;
```

fajl1	fajl2	cos	eucl	jaccard
4490	4490	509	0	1
4490	4473	509	0	0.983333
4490	4408	508.897	2	1
4490	3432	508.791	1.73205	1
4490	4008	508.791	1.73205	1
4490	2571	508.699	3	1
4490	3494	508.699	3	1
4490	3536	508.699	3	1
4490	4075	508.333	3.73205	1
4490	3093	507.699	4.73205	0.967213
4490	2497	507.59	6.14626	0.983333
4490	2845	507.59	6.14626	0.983333
4490	2738	507.132	5.91359	0.983333
4490	3785	507.132	5.91359	0.983333
4490	2757	506.607	11.2925	0.983333
4490	2529	506.2	12.2925	0.983333
4490	2525	506.2	12.2925	0.967213
4490	3263	506.2	12.2925	0.983333
4490	3173	506.2	12.2925	0.967213
4490	4459	506.125	10.0246	1
4490	3356	506.072	11.9428	1
4490	3878	506.041	12.4741	0.983333
4490	4443	505.416	13.1708	1
4490	2873	505.4	13.4388	0.983333
4490	3405	505.324	14.4388	0.967213
4490	4101	505.3	12.9282	1
4490	3038	504.795	16.8211	0.967213
4490	2518	504.795	16.8211	0.967213

A két lekérdezés eredményének metszetét tekintve, két metrikát is figyelembe tudunk venni a hasonlóság mértékének megállapításához. Nézzük meg, hogy az előző két lekérdezés metszete mi lesz. A két lekérdezést egyenként 28 rekord lekérdezésére írtam meg, ezen két lekérdezés metszete 13 elemű.

```
SELECT jac.* FROM (SELECT k4.fajl1,k4.fajl2
k4f2,k4.cos,k4.eucl,k4.jaccard FROM kiertekeles k4
WHERE k4.cos <= (SELECT MAX(k3.cos) FROM kiertekeles k3
WHERE k3.fajl1 <> k3.fajl2 AND k3.fajl1='4490') AND
k4.fajl1 = '4490' ORDER BY k4.cos DESC LIMIT 28) jac
INNER JOIN (SELECT k2.fajl1 ,k2.fajl2
k2f2,k2.cos,k2.eucl,k2.jaccard FROM kiertekeles k2
WHERE k2.jaccard <= (SELECT MAX(k1.jaccard) FROM
kiertekeles k1 WHERE k1.fajl1 <> k1.fajl2 AND
fajl1='4490') AND k2.fajl1 = '4490' ORDER BY k2.jaccard
DESC,k2.cos DESC LIMIT 28) cosi ON jac.k4f2=cosi.k2f2 ;
```

fajl1	k4f2	cos	eucl	jaccard
4490	4490	509	0	1
4490	4473	509	0	0.983333
4490	4408	508.897	2	1
4490	3432	508.791	1.73205	1
4490	4008	508.791	1.73205	1
4490	2571	508.699	3	1
4490	3494	508.699	3	1
4490	3536	508.699	3	1
4490	4075	508.333	3.73205	1
4490	4459	506.125	10.0246	1
4490	3356	506.072	11.9428	1
4490	4443	505.416	13.1708	1
4490	4101	505.3	12.9282	1

1.13. Összegzés

A metrikák használhatók a dokumentumok közötti hasonlóságok kimutatására, a lekérdezésekkel több metrikát is figyelembe tudunk venni a hasonlóság megállapításához. A módszer nagyban megkönnyíti a dokumentumok össszehasonlítását nagy mennyiségű dokumentumok esetén, de az eljárásnak így is nagy időigénye van. Az eredmények felhasználásáról a későbbiekben a témavezetőmmel fogok együttműködni.

1.14. Melléklétek

1 -

```
<!DOCTYPE html>
<html class="no-js" lang="hu" prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb#">
<head>
    <title>Totalcar - Magazin - Hírek</title>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE11">
    <!-- [if IE]>
    <meta http-equiv="imagetoolbar" content="no" />
    <meta name="MSSmartTagsPreventParsing" content="true" />
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <![endif]-->
        <meta name="manis:breakpoint">
    <meta name="manis:breakpoints">
            <meta name="robots" content="max-snippet:-1, max-image-preview:standard,
            max-video-preview:15" />
        <meta name="copyright" content="https://totalcar.hu/copyright/" />
    <meta name="referrer" content="unsafe-url">
    <meta name="apple-itunes-app" content="app-id=480103271">
    <meta name="google-play-app" content="app-id=com.aff.index.main">
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta property="fb:app_id" content="659056284133606" /><meta property="fb:pages" content=
"117738001600393" /><meta name="description" content="Főtitkára lesz a
Renault-Nissan-Mitsubishinek Így néz ki most az 1900 lóerős Rimac Közös gyárat épít a BMW és a
Great Wall " />    <meta property="og:image" content=
"https://totalcar.hu/assets/images/facebook_logo.png?v3" /><meta property="article:publisher"
content="https://www.facebook.com/Totalcarhu" /><meta name="twitter:card" content=
"summary_large_image"><meta name="twitter:image:src" content=
"https://totalcar.hu/assets/images/facebook_logo.png">        <link rel="search" title="Totalcar" type=
"application/opensearchdescription+xml" href="/assets/static/opensearch.xml" />
    <link rel="copyright" title="Szerzői jogok" href="/copyright/" />
    <link rel="author" title="Impresszum" href="/impresszum/" />
    <link rel="home" title="" href="/" id="homelink" />
    <link rel="apple-touch-icon" sizes="180x180" href=
    "/assets/images/favicons/apple-touch-icon.png">
<link rel="icon" type="image/png" href="/assets/images/favicons/favicon-32x32.png" sizes="32x32">
<link rel="icon" type="image/png" href="/assets/images/favicons/favicon-16x16.png" sizes="16x16">
<link rel="manifest" href="/assets/images/favicons/manifest.json">
<link rel="shortcut icon" href="/assets/images/favicons/favicon.ico">
    <link rel="mask-icon" href="/assets/images/favicons/safari-pinned-tab.svg" color="#1f2e60">
    <meta name="apple-mobile-web-app-title" content="Totalcar">
    <meta name="application-name" content="Totalcar">
    <meta name="theme-color" content="#1f2e60">
<script type='text/javascript'>var _sf_startpt=(new Date()).getTime()</script>
<link href="https://totalcar.hu/assets/static/indexnew_css/public/global.css?v=1575048240" rel=
"stylesheet" type="text/css" /><link href=
"https://totalcar.hu/assets/static/indexnew_css/public/tc-global.css?v=1575048240" rel="stylesheet"
type="text/css" />            <link rel="stylesheet" type="text/css" href=
"https://totalcar.hu/assets/static/indexnew_css/public/tc-blog.css?v=1575048240"  />
        <link rel="stylesheet" type="text/css" href=
        "https://totalcar.hu/assets/static/indexnew_css/public/tc-fixed_header.css?v=1575048240"  />
        <link rel="alternate" type="application/rss+xml" title="Hírek posztok" href="/rss/" />
    <meta name="logcustomtarget" content="1"/>
    <!-- tc3 -->
    <script src="https://totalcar.hu/ident/getall/?c=dGMz" type="text/javascript"></script>
    <script src="https://totalcar.hu/assets/static/js/public/index_vendor.ugly.js?v=1575048241" type
```

```
TotalCar
Magazin
Hírek
        Főtitkára lesz a Renault-Nissan-Mitsubishinek
Andróczi Balázs
            7 órája
Módosítva: 2019.11.30 08:30
 14
A Renault-Nissan-Mitsubishi korábban Carlos Ghosn irányításával jött létre, de az ő zajos bukása óta nem nagyon megy a bolt. A Nissan eredményei látványosan beleálltak a földbe, és a közös projektek mintha szintén elakadtak volna. A megoldást az igazgatók és ügyvezetők ülésén egy új poszt bevezetésében látják, aki mintha épp Ghosn korábbi szervező szerepét töltené be.
Távolról sincs vége, olvasson még
        Legalább tízezer dolgozójától szabadul a Mercedes
Andróczi Balázs
            tegnap, 16:21
Módosítva: 2019.11.29 16:27
 104
A Daimler durván 3%-os létszámcsökkentést hajt végre a következő három év során, erről nyilatkozott a cég HR igazgatója.
Távolról sincs vége, olvasson még
        Így néz ki most az 1900 lóerős Rimac
Andróczi Balázs
            tegnap, 14:30
Módosítva: 2019.11.29 14:32
 56
A Rimac első, nagyobb mennyiségű gyártásra tervezett modellje nagyjából egy év múlva válhat elérhetővé, de most megmutatták, milyen a prototípus, amelyen épp a futómű és a kormánymű beállításait finomhangolják.
3
Galéria: Rimac C_Two prototípus képek
Távolról sincs vége, olvasson még
        Közös gyárat épít a BMW és a Great Wall
Andróczi Balázs
            tegnap, 12:24
Módosítva: 2019.11.29 12:25
 0
A BMW új partnerével elektromos autók gyártására alkalmas üzemet alapít, ahol BMW, Mini és Great Wall villanyautókat gyártanak majd. Nem is keveset, a tervek szerint évi 160 ezer autó előállítására rendezkednek be.
26
Galéria: Mini Rocketman
Távolról sincs vége, olvasson még
        Hibrid Porsche 911-es készül
Andróczi Balázs
            tegnap, 06:17
Módosítva: 2019.11.30 08:30
 139
Oliver Blume, a Porsche márka vezetője egy interjúban beszélt a 911-es sorozattal kapcsolatos tervekről. Kiderült, hogy a villanyosítást ez sem kerülheti el, de egyelőre nem tisztán elektromos hajtásban gondolkodnak, sőt, még csak nem is konnektoros hibridben.
17
Galéria: Sajtógaléria: Porsche 911 – 2019.
Távolról sincs vége, olvasson még
        Javítanak az Audi E-tron hatótávján
```

2 -


1903. december 28-án született Neumann Miksa és Kann Margit első gyermekeként Budapesten,
a Váci körút (ma Bajcsy-Zsilinszky út) 62. sz. házban.
Jánosnak később két öccse is született: Mihály (1907) és Miklós (1911).
Az édesapja Pécsről származott, és Budapesten ügyvédként dolgozott,
1903 ..... [{"lemma": "1903.", "tag":Digit]=3+[_O.....TT 2 B-NP O
december true {"stem": ["december"], "anas": [[["st", "decem..... I-NP O
28-án true {"stem": ["28"], "anas": [[["st", "28"], ["po", "adj_num"], ["is", "dikA_DATE_noun"], ["t
született true {"stem": ["született", "születik"], "anas": [[["st", "született"], ["po", "adj"], ["ts", "N
Neumann true {"stem": ["Neumann"], "anas": [[["st", "Neumann"], ["po", "noun_prs"], ["ts", "NOM
Miksa true {"stem": ["Miksa"], "anas": [[["st", "Miksa"], ["po", "noun_prs"], ["ts", "NOM"]]]} [{"le
és true {"stem": ["és"], "anas": [[["st", "és"], ["po", "con"]]]} [{"lemma": "és", "tag": "[/Cnj]", "mo
Kann false {"stem": [], "anas": []} [] kan [/N][Nom] NOUN Case=Nom|Number=Sing 8 COC
Margit true {"stem": ["Margit"], "anas": [[["st", "Margit"], ["po", "noun_prs"], ["ts", "NOM"]]]} [{"le
első true {"stem": ["egy", "első"], "anas": [[["st", "egy"], ["po", "adj_num"], ["is", "dik_ORDINAL_a
gyermekeként true {"stem": ["gyermek"], "anas": [[["st", "gyermek"], ["po", "noun"], ["ts", "NOM"
Budapesten true {"stem": ["Budapest"], "anas": [[["st", "Budapest"], ["po", "noun_prs"], ["ts", "N
, true {"stem": [","], "anas": [[["st", ","], ["po", "punct"]]]} [{"lemma": ",", "tag": "[Punct]", "morp
a true {"stem": ["a"], "anas": [[["st", "a"], ["po", "noun"], ["ts", "NOM"], ["al", "a-vá"], ["al", "a-val"]
Váci true {"stem": ["Vác"], "anas": [[["fl", "\""], ["st", "Vác"], ["po", "noun_prs"], ["ts", "NOM"], ["al
körút true {"stem": ["körút"], "anas": [[["st", "körút"], ["po", "noun"], ["ts", "NOM"], ["al", "körutak"]
(false {"stem": [], "anas": []} [{"lemma": "(", "tag": "[Punct]", "morphana": "", "readable": "", "twole
ma true {"stem": ["ma"], "anas": [[["st", "ma"], ["po", "noun"], ["ts", "NOM"]], [["st", "ma"], ["po",
Bajcsy-Zsilinszky true {"stem": ["Bajcsy-Zsilinszky"], "anas": [[["st", "Bajcsy-Zsilinszky"], ["po",
út true {"stem": ["út"], "anas": [[["st", "út"], ["po", "noun"], ["ts", "NOM"], ["al", "utak"]]]} [{"lemm
) false {"stem": [], "anas": []} [{"lemma": ")", "tag": "[Punct]", "morphana": "", "readable": "", "twole
62. true {"stem": ["2"], "anas": [[["6"]], [["st", "2"], ["po", "adj_num"], ["ts", "NOM"], ["al", "2-vé"],
sz. true {"stem": ["sz", "sz."], "anas": [[["st", "sz"], ["po", "noun"], ["ts", "NOM"], ["al", "sz-szé"]
házban true {"stem": ["ház"], "anas": [[["st", "ház"], ["po", "noun"], ["ts", "NOM"], ["al", "házak"],
. true {"stem": ["."], "anas": [[["st", "."], ["po", "punct"]]]} [{"lemma": ".", "tag": "[Punct]", "morp
Jánosnak true {"stem": ["János"], "anas": [[["st", "János"], ["po", "noun_prs"], ["ts", "NOM"], ["is"
később true {"stem": ["később", "késő"], "anas": [[["st", "később"], ["po", "adv"], ["al", "későbbre"
két true {"stem": ["két"], "anas": [[["st", "két"], ["po", "adj_num"], ["ts", "NOM"]]]} [{"lemma": "k
öccse true {"stem": ["öcs"], "anas": [[["st", "öcs"], ["po", "noun"], ["is", "POSS_SG_1"], ["ts", "N
is true {"stem": ["is"], "anas": [[["st", "is"], ["po", "con"]]]} [{"lemma": "is", "tag": "[/Adv]", "morpl
született true {"stem": ["született", "születik"], "anas": [[["st", "született"], ["po", "adj"], ["ts", "N
: true {"stem": [":"], "anas": [[["st", "", ""], ["po", "punct"]]]} [{"lemma": ":", "tag": "[Punct]", "mo
Mihály true {"stem": ["Mihály"], "anas": [[["st", "Mihály"], ["po", "noun_prs"], ["ts", "NOM"]]]} [{
(false {"stem": [], "anas": []} [{"lemma": "(", "tag": "[Punct]", "morphana": "", "readable": "", "twole
1907) true {"stem": [], "anas": []} [{"lemma": "1907", "tag": "[/Num|Digit][Nom][Punct]", "morphar
és true {"stem": ["és"], "anas": [[["st", "és"], ["po", "con"]]]} [{"lemma": "és", "tag": "[/Cnj]", "mo
Miklós true {"stem": ["Miklós"], "anas": [[["st", "Miklós"], ["po", "noun_prs"], ["ts", "NOM"]]]} [{
(false {"stem": [], "anas": []} [{"lemma": "(", "tag": "[Punct]", "morphana": "", "readable": "", "twole
1911) true {"stem": [], "anas": []} [{"lemma": "1911", "tag": "[/Num|Digit][Nom][Punct]", "morphar
. true {"stem": ["."], "anas": [[["st", "."], ["po", "punct"]]]} [{"lemma": ".", "tag": "[Punct]", "morp
Az true {"stem": ["az"], "anas": [[["st", "az"], ["po", "noun_pron"], ["ts", "NOM"], ["al", "azé"], ["a
édesapja true {"stem": ["édesapa"], "anas": [[["st", "édesapa"], ["po", "noun"], ["is", "POSS_SG_
Pécsről true {"stem": ["Pécs"], "anas": [[["st", "Pécs"], ["po", "noun_prs"], ["ts", "NOM"], ["al", "P
származott true {"stem": ["származik", "származott"], "anas": [[["st", "származik"], ["po", "vrb"]
, true {"stem": [","], "anas": [[["st", ","], ["po", "punct"]]]} [{"lemma": ",", "tag": "[Punct]", "morp
és true {"stem": ["és"], "anas": [[["st", "és"], ["po", "con"]]]} [{"lemma": "és", "tag": "[/Cnj]", "mo
Budapesten true {"stem": ["Budapest"], "anas": [[["st", "Budapest"], ["po", "noun_prs"], ["ts", "N
ügyvédként true {"stem": ["ügyvéd"], "anas": [[["st", "ügyvéd"], ["po", "noun"], ["ts", "NOM"], ["is
dolgozott true {"stem": ["dolgoz", "dolgozott", "dolgozik"], "anas": [[["st", "dolgoz"], ["po", "vrb"],
, true {"stem": [","], "anas": [[["st", ","], ["po", "punct"]]]} [{"lemma": ",", "tag": "[Punct]", "morp
aztán true {"stem": ["aztán"], "anas": [[["st", "aztán"], ["po", "con"]]]} [{"lemma": "aztán", "tag":
a true {"stem": ["a"], "anas": [[["st", "a"], ["po", "noun"], ["ts", "NOM"], ["al", "a-vá"], ["al", "a-val"]
Magyar true {"stem": ["magyar"], "anas": [[["st", "magyar"], ["po", "noun"], ["ts", "NOM"]], [["st",
Jelzálog- true {"stem": [], "anas": []} [] jelzálog [/N][Nom][Hyph:Hyph] NOUN Case=Nom|Nun
és true {"stem": ["és"], "anas": [[["st", "és"], ["po", "con"]]]} [{"lemma": "és", "tag": "[/Cnj]", "mo
Hitelbankhoz true {"stem": ["hitelbank"], "anas": [[["pa", "hitel"], ["st", "hitel"], ["po", "noun"], ["ts",
került true {"stem": ["kerül", "került"], "anas": [[["st", "kerül"], ["po", "vrb"], ["ts", "PRES INDIC

3 - https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50
4 - https://www.freecodecamp.org/news/an-introduction-to-bag-of-words-and-how-to-code-it-in-python-for-nlp-282e87a9da04/
5 - https://aws.amazon.com/ec2/instance-types/z1d/