DOI:10.15774/PPKE.ITK.2016.003

NYELVTECHNOLÓGIAI ALGORITMUSOK KORPUSZOK AUTOMATIKUS ÉPÍTÉSÉHEZ ÉS PONTOSABB FELDOLGOZÁSUKHOZ

DOKTORI (PH.D.) DISSZERTÁCIÓ

Endrédy István

Témavezető:
Dr. Prószéky Gábor,
az MTA doktora

PÁZMÁNY PÉTER KATOLIKUS EGYETEM

INFORMÁCIÓS TECHNOLÓGIAI ÉS BIONIKAI KAR

ROSKA TAMÁS MŰSZAKI ÉS TERMÉSZETTUDOMÁNYI DOKTORI ISKOLA

Budapest, 2016

DOI:10.15774/PPKE.ITK.2016.003

DOI:10.15774/PPKE.ITK.2016.003

„…márpedig a beszéd titka az összes titok közül a legnagyobb”
Prágai regék, A gólem

DOI:10.15774/PPKE.ITK.2016.003

DOI:10.15774/PPKE.ITK.2016.003

Köszönetnyilvánítás

Szeretnék köszönetet mondani azoknak, akik nélkül ez nem sikerült volna.

Köszönöm felsős magyartanáromnak, Bukovits Mártának, aki a nyelvtant világosan tudta tanítani, és azt a benyomást keltette, hogy a nyelvtanban matematikai pontosság létezhet. Köszönöm Naszódi Mátyás egyetemi tanáromnak, hogy bevezetett a számítógépes nyelvészetbe és a MorphoLogicba. Az itt leírt munkáim és eredményeim szorosan kapcsolódnak a MorphoLogicban eltöltött 13 évemhez. Ezért a disszertációm a MorphoLogic előtti tisztelgésnek is tekinthető. Hálás vagyok a cégnél töltött évekért, sokat tanultam a munkatársaktól: Kis Balázs, a hatékony; Tihanyi Laci, a gyors; Pál Miki, az alapos; Kundráth Peti, a „kóder”; Novák Attila, a tudós; Sebestyén Zsolt, akire mindig számíthattam; Hubay Kati, a problémamegoldó; Aggod Andi, a kitartó tesztelő; Kincse Szabi, a kommunikátor, és mindenki más, akikkel jó volt együtt dolgozni.

Köszönöm témavezetőmnek, Dr. Prószéky Gábornak az atyai bátorítást és a jó légkörű beszélgetéseket. Bármikor örömmel fogadott, még ha nagyon elfoglalt volt is. Pótolhatatlan volt a szerepe ebben a dolgozatban.

Köszönöm a Bírálóknak az értékes észrevételeiket, javaslataikat.

Köszönöm családomnak, feleségemnek, Orsinak, gyermekeimnek, Balázsnak, Katának, Dorkának és Bencének, hogy támogattak ebben a munkában, és türelmesen elfogadták, hogy emiatt nem tudok velük annyi időt tölteni. (9 éves Dorka lányomnak ígértem, ha elkészülök ezzel a munkával, építünk együtt egy műhelyt a pincében. Kétnaponta megkérdezte, hogy hogyan állok, mikor kezdhetjük.)

Nem tudom hogyan megköszönni szüleimnek az életemet, és ahogy figyelemmel kísérték ezt az ügyemet is. Testvéreimnek ahogy szeretnek.

Köszönöm a doktorandusztársaknak is a közös projekteket, az utazásokat. Elévülhetetlen érdeme van Indig Balázsnak, aki – a tudta nélkül – átsegített a holtponton, ahol fel akartam adni, és Dr. Wenszky Nórinak, hogy mindig készségesen lektorálta az írásaimat.

Köszönöm a Doktori Iskola korábbi és jelenlegi vezetőinek, Dr. Roska Tamás, Nyékyné Dr. Gaizler Judit és Dr. Szolgay Péter dékánoknak, hogy lehetővé tették és több módon is támogatták kutatói munkámat. Dr. Vida Katinkának a kiemelt figyelmet, az egyetem többi munkatársának is a háttérmunkát.

Köszönöm a szerda déli miséket az egyetem kápolnájában.

DOI:10.15774/PPKE.ITK.2016.003

Abstract

This thesis is interdisciplinary: it is built on the common points of linguistics and information technology, as it focuses on corpus building and corpus management. Several linguistic studies are based on big text corpora. On the one hand, building and processing such a large amount of text require the methods and tools of information technology; on the other hand, these corpora serve the purposes of linguistics. Below, I present a method of building corpora automatically, then researches in this corpus are shown.

The study touches the themes of stemming, lemmatization, diacritics restoration, noun phrase detection and sentence analysis as well. The workflow was corpus driven: ideas were inspired and verified by the corpus.

The first part of this study describes the tasks, problems and results of automatic corpus building through the description of several corpora created during this research. The size of the corpus is one of its most important attributes. However, it is rather expensive to build a large corpus. Besides, if a corpus is built based on texts from the internet, its size can constantly increase. Furthermore, an up-to-date corpus makes it possible for linguists to investigate the frequent structures of the given language, and the changes in usage over time (section 2, 9).

Next, the investigations carried out in these corpora are described. The steps were driven by the corpus: Hungarian comments on the internet are usually written without diacritics. This can be fixed by a diacritics restoration application. The more accurate processing of texts needs a more accurate stemmer or lemmatizer. The lemmatizer based on the analysis of Humor and an application for its evaluation were also created (section 3, 4, 5).

Finally, this study traces the detection of noun phrases in English and Hungarian. A rule based system and a statistical learning algorithm were also created and evaluated. The word frequency and N-gram collections resulted in interesting consequence: these measurements themselves are suitable for the investigation of typical structures of noun phrases and sentences (section 6).

Therefore, an online tool was developed. The present state-of-the-art NP chunking system for Hungarian was produced by the error analysis of the HunTag chunker. Most of the errors were caused by neighboring NPs, the false unification of possessors and their possesses, and because the POS tag for participles and adjectives did not differ in the MSD formalism. Explicit features were defined for them, resulting in a significant improvement (section 7, 8).

DOI:10.15774/PPKE.ITK.2016.003

Kivonat

Jelen dolgozat témája interdiszciplináris: a nyelvészet és az informatika közös pontjaira épül. A nyelvtechnológia maga is ezen a határsávon halad, de jelen feladatnál ez különösen is igaz. Számos kutatás és nyelvészeti munka alapvető alapanyaga a nagyméretű szövegkorpusz. Egyrészről ezen nagy mennyiségű szöveg automatikus építése és feldolgozása során szükség van az informatika módszereire és eszközeire, másrészről a korpuszon nyelvészeti kutatásokat szeretnénk végezni. Arra vállalkozom, hogy bemutassam hogyan lehet automatizáltan, webes forrásból nagy korpuszokat építeni, majd pedig az ezekben végzett kutatásokat mutatom be.

A dolgozat érinti a szótövesítés, az ékezetesítés, a főnévi csoport felismerése és a mondatvázak témáját is. A munkafolyamat korpuszvezérelt volt: az ötleteket a korpuszból merítettem, és azon is ellenőriztem.

Először az automatikus korpuszépítés feladatait, problémáit és eredményeit mutatom be több általam épített korpusz példáján. A webkorpuszok előnye, hogy az alapanyagul szolgáló web tartalma folyamatosan növekszik és ingyenes. Hátránya, hogy ez a tartalom nagy részben zajt tartalmaz (menüsorok, ismétlődések, reklámok, HTML formázás stb.), amitől meg kell tisztítani. A kézzel összeállított korpuszokat könnyen megelőzik méretükben a webkorpuszok, azonban ahhoz, hogy a minőségben is felvegyék a versenyt, az kizárólag a gépi feldolgozás, összeállítás lépésein múlik. A webkorpusz előnye feltétlenül az is, hogy az internetes közösség által folyamatosan létrehozott szövegekben megvizsgálható az adott nyelv gyakori struktúrái, akár időbeli változásai is (2. és 9. fejezetek).

A dolgozat második felében a korpuszban történt vizsgálatokkal foglalkozom. Az egyes lépéseket a korpusz vezérelte. Például az internetes hozzászólások, kommentek sokszor ékezet nélküliek. Ezt a kifejlesztett, elemzőalapú ékezetesítő orvosolhatja. A szövegek pontosabb feldolgozásához nélkülözhetetlen egy minél pontosabb tövesítő. A Humor elemzéseire épülő tövesítő, és ennek minőségét kiértékelő eszköz is készült (3., 4., és 5. fejezetek).

A dolgozat végezetül a magyar és angol főnévi csoportok felismerésével foglalkozik, mind szabályalapú rendszerrel, mind statisztikai tanulóalgoritmussal is. A korpuszban történt többféle N-gram-keresés érdekes eredményt hozott: ezek önmagukban is alkalmasak a tipikus főnévi csoportok és a mondatok szerkezetének vizsgálatára. Ezért egy online eszköz is készült, amit más kutatók fel tudtak használni a saját kutatásaikban (6. fejezet).

A magyar főnévi csoport felismerésben elért eddigi legjobb eredményt a HunTag hibaelemzésével sikerült elérni. A legtöbb hiba a szomszédos NP-k felismerése (hol tévesen egybe-, hol tévesen különvette), illetve a birtok és a birtokos téves szétválasztása volt. Az ezekben történt javítások eredményezték a legjelentősebb javulást (7. és 8. fejezetek).

DOI:10.15774/PPKE.ITK.2016.003

Tartalomjegyzék

DOI:10.15774/PPKE.ITK.2016.003

DOI:10.15774/PPKE.ITK.2016.003

DOI:10.15774/PPKE.ITK.2016.003

Táblázatjegyzék

DOI:10.15774/PPKE.ITK.2016.003

DOI:10.15774/PPKE.ITK.2016.003

Ábrajegyzék

DOI:10.15774/PPKE.ITK.2016.003

Glosszárium

boilerplate – A szó eredetileg ’változtatás nélkül újrahasználható rész’-t jelent az iparban (acélfeldolgozás, nyomdaipar). Egy weblap esetén azt a szöveget jelenti, ami nem tartozik a fő tartalomhoz (fejléc, lábléc, menük, hirdetések, designelemek stb.). *Boilerplate removal*-nak hívja a szakirodalom azt a műveletet, amikor ezeket a járulékos részeket egy honlap tartalmából eltávolítjuk, így megkapjuk a lap értékes tartalmát.

crawler – keresőrobot (más néven *spider* vagy *bot*)
Olyan alkalmazás, amely képes letölteni és tárolni nagy mennyiségű webes tartalmat, a linkek automatikus bejárásával.

F-mérték – kiértékelésnél használt metrika, a pontosság *tp / (tp + fp) és* a fedés *tp / (tp + fn)* harmonikus közepe, ahol *tp: true positive, fp: false positive, fn: false negative* értékek

HTML – HyperText Markup Language
Weblapok formátuma, leíró nyelve. Ez tartalmazza egy weblap megjelenítéséhez, viselkedéséhez szükséges információkat. Ez internetes szabvány is.

IR – *information retrieval* ’keresés, információ-visszakeresés’
Az a művelet, amikor egy információs igényhez visszaadjuk a releváns dokumentumokat, illetve az ehhez kapcsoló algoritmusok, modellek.

korpusz – Nyelvészeti céllal összegyűjtött nagy mennyiségű szöveg vagy szövegek összessége

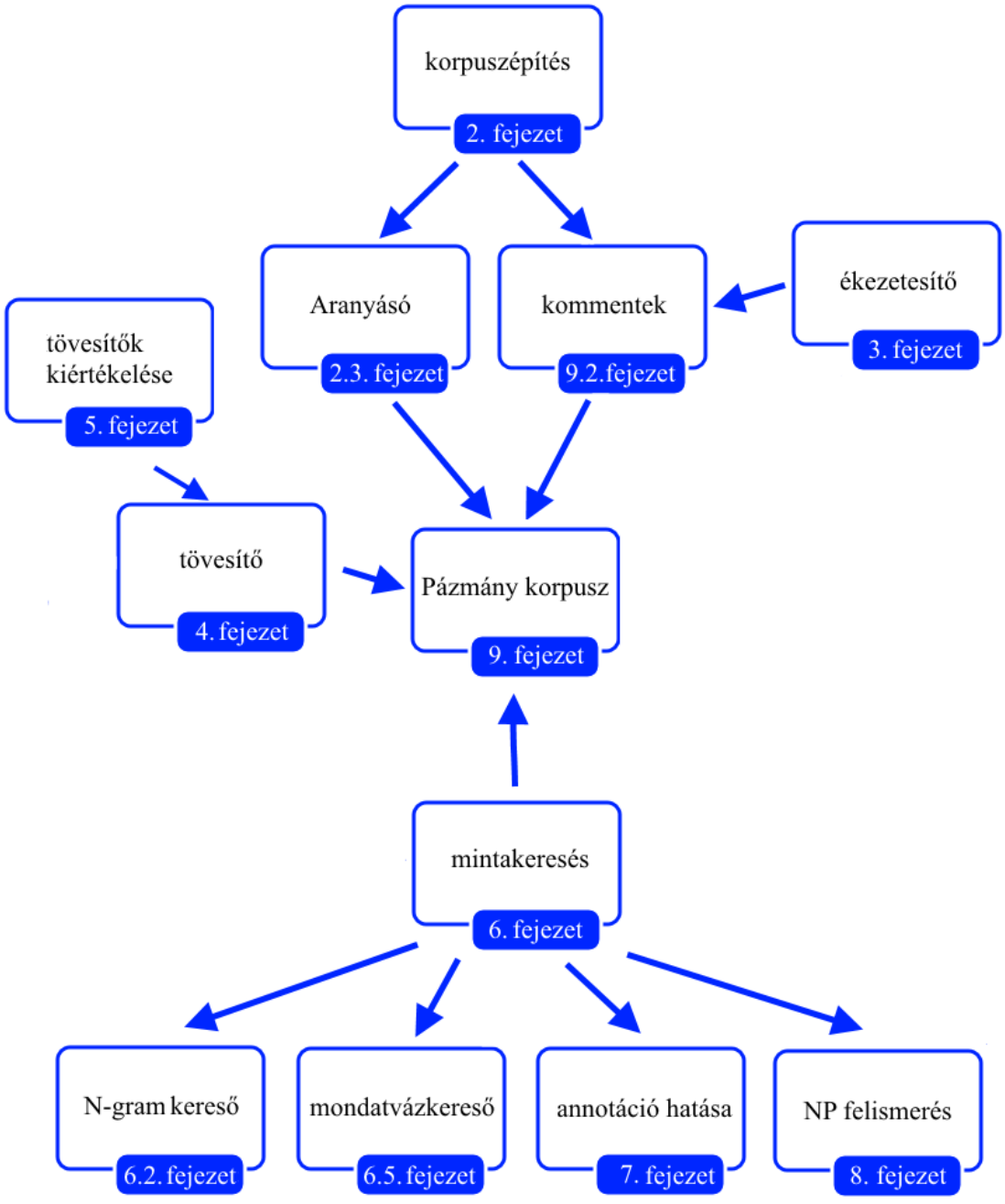
NP – *noun phrase* ’főnévi csoport’

OOV – *out of vocabulary*, egy program számára ismeretlen szó.

overfitting – amikor egy modell túlzottan "rátanul" a tanítóadatra, és ismeretlen adatra rosszabbul teljesít (a konkrét adatot tanulja meg, nem vele reprezentált feladatot)

DOI:10.15774/PPKE.ITK.2016.003

Áttekintés – a részfeladatok kapcsolódásai



1. ábra. A disszertáció részfeladatai és kapcsolódásaik

DOI:10.15774/PPKE.ITK.2016.003

1. Bevezetés

Az emberi nyelvtechnológia számos területén szükségesek a nagy korpuszok. A korpusz egyik fontos tulajdonsága a mérete, azonban a használhatóságát nagyban befolyásolja a kiegyensúlyozottsága: a szövegei mennyire reprezentatívak. A többféle forrásból és változatos műfajból származó szövegekből felépülő korpusz jobban reprezentálja a nyelvet, általános nyelvészeti kutatásokat jobban ki tud szolgálni. Azonban meglehetősen költséges munka egy nagy korpusz építése. Másrészt, a web ingyenes, és a tartalma másodpercenként növekszik. Ha jó minőségű korpuszt építenénk az internet alapján, ez lehetővé tenné a korpusz méretének állandó növekedését, és lehetőséget adna arra, hogy megvizsgáljuk az adott nyelv gyakori struktúráit és ezek időbeli változásait. A dolgozat érinti a szótövesítés, az ékezetesítés, a főnévicsoport- felismerés és a mondatvázak témáját is. A munkafolyamat korpuszvezérelt volt: az ötleteket a korpuszból merítettük, és vele is ellenőriztük.

A korpuszépítés és vizsgálat volumenét jól illusztrálhatja a Biblia, az egyik legrégebbi korpusz: 600.000 szóból áll, és könyv formában 4-5 cm vastag. Azonban már egy 500 millió tokenes korpusz könyv formájában nagyjából 40 métert tenne ki, elolvasása pedig legalább 9 évig tartana (ha napi 8 órát olvasnánk az év minden napján). Ekkora méretű korpusznak nem csak az emberi feldolgozása, hanem már csupán az elolvasása is nehézkes lenne. Kezelésükhöz mindenképpen gépi támogatás szükséges.

1.1. Háttér

Nagy változás lenne az emberiség történetében, ha a számítógép értené a természetes nyelvet. Ez megváltoztatná a számítógép-használatot, a gépi fordítás minőségét, és a keresési módszereket is. Ez az elképzelés talán még távoli, de még a 8000 mérföldes út is kis lépésekből áll. Jelen kutatás egy kis lépés ezen az úton.

Egy nagy korpusz rengeteg információt tartalmaz a világról, az emberekről, és magáról a nyelvről is. Újságcikkek leggyakoribb főnév kollokációi például (többek között) az adott időszak híres és népszerű személyeit mutatják meg. Napjainkban a legnagyobb és emellett ingyenes korpusz lehet a web. A kutatás arról szól, hogyan hozhatunk létre nagyméretű korpuszt az internetről, miként ismerhetjük fel benne a főnévi csoportok és mondatok leggyakoribb mintáit.

Kutatócsoportunk, az MTA-PPKE Magyar Nyelvtechnológiai Kutatócsoportja egy szövegelemzőn dolgozik, teljesen új megközelítésekkel. A projekt egyik újítása szorosan kapcsolódik a gyakori mintákhoz: ha egy ismert szerkezet érkezik a szövegben, akkor az

DOI:10.15774/PPKE.ITK.2016.003

elemző a szerkezet előre kiszámított elemzését veszi elő: épp úgy, mint az emberek. A szövegfeldolgozás egy felnőtt és egy hároméves gyermek esetén leginkább az ismert minták számában különbözik. A felnőtt többet ismer, ezért gyorsabban fel tudja dolgozni a szöveget. Ez az alapgondolata a leendő elemzőnknek, hasonlóan kell majd működnie, mint az emberi szövegfeldolgozás. Ezt a célt szolgálja majd a szoftverünk által az internet anyagaiból épített korpusz. A felismert gyakori minták hozzáadódnak az elemző „világismeretéhez", amelyet később ismertetünk (6.4. és 7.1.4. fejezetek).

Ez a dolgozat segíthet egy jobb elemző létrehozásában, különös tekintettel a korpuszra (9. fejezet), és az ennek során szerzett tapasztalatok javíthatják más nyelvi eszközök minőségét: Humor morfológiai elemzőt (Prószéky és Kis 1999), PurePos szófaji egyértelműsítőt (Orosz és Novák 2013), web crawlert és a GoldMiner szövegkinyerő algoritmust (Endrédy és Novák 2013).

1.2. Motiváció

A projekt részfeladataihoz – a teszteléshez, elemzéshez, gyakori szóalakok, mondatszerkezetek feltérképezéséhez – elengedhetetlenül fontos, hogy nagy méretű, valós, ember által írt szövegek álljanak rendelkezésre.

Az elérhető magyar korpuszok száma örvendetes módon egyre növekszik. Az egyik legnagyobb a 2004-ben készült a BME MOKK webkorpusza (Halácsy és mtsai. 2004), melynek mérete 600 millió szó. A Magyar Nemzeti Szövegtár a maga 187,5 millió szavas méretével ugyan ennél kisebb, de a tudatosan válogatott tartalom teljes mértékben szófaj-egyértelműsített (Váradi 2002). Ennek a korpusznak a felújított változata, az MNSZ2 folyamatosan növekszik (Oravecz, Váradi, és Sass 2014), jelenleg 784 millió tokenes.

A 1,2 millió szavas Szeged Korpusz (Alexin és mtsai. 2003) egyedülálló abban, hogy morfológiai annotációja emberi ellenőrzéssel készült. Ezt egyébként a mai gigakorpuszokon gyakorlatilag nem lehet megvalósítani éppen a hatalmas méretből adódó jelentős időigény miatt. A Szeged TreeBank (Csendes és mtsai. 2005) a Szeged Korpusz mondatelemzés-annotációval kiegészített változata.

Azonban azzal kellett szembesülnünk, hogy a magyar korpuszok világában eddig nem volt elérhető olyan igazán nagy méretű (milliárd tokenes) korpusz, amelyik mind méretével, mind többféle annotációjával performancia alapú elemzést megcélzó kutatási projektünk igényeit (Prószéky és Indig 2015) megfelelően ki tudja szolgálni.

Másrészről a nyelv folyamatosan változik, él: új kifejezések jelennek meg, mások eltűnnek. Az internet a folyamatosan létrehozott tartalmaival jól tükrözi az éppen használt nyelvet, az adott időszak nyelvhasználatának lenyomata. Az internet még nincs 30 éves, de már most

DOI:10.15774/PPKE.ITK.2016.003

bányásszák egy-egy időszak érdeklődési területeit, kereséseit[1], megnyilatkozásait (Battelle 2005; Lampos és mtsai. 2014; Miháltz és mtsai. 2015), ami már nem csak nyelvészeti (hanem például gazdasági) szempontból is értékesnek bizonyul. Az, hogy a világháló őrzi az egyes évtizedek netes közösség által létrehozott szövegeit, önmagában is értékes tulajdonság, ez is a webkorpuszok mellett szóló érv volt. A web szövegeinek előnye a méretén túl az is, hogy a segítségükkel időbeli változásokat (trendek, gyorsan elterjedő új kifejezések) is ki lehet mutatni. Úgy éreztük, hogy szükség van egy nagy, átfogó, annotált és folyamatosan frissített adatbázisra.

1.3. Feladatok

Kutatási témám a magyar web szövegeiből való korpusz építése, ezen szövegek elemzése, az előforduló szavak, szókapcsolatok és hibák feltérképezése. Ehhez szükséges volt egy crawler, azaz egy webet letöltő robot elkészítése. A crawler önmagában is rengeteg alfeladatot jelent: a letöltött lapokon található urleket ki kell gyűjteni, hogy a crawler azokat is bejárhassa. Az urleket tárolni kell, a letöltési dátummal és a tiszított tartalom md5 hashével együtt: ez lehetővé teszi, hogy többedszeri letöltéskor összehasonlítható legyen a tartalom. A letöltés bizonyult a folyamat leglassabb részének: ugyanahhoz a szerverhez kis késleltetés után "illik" csak újabb kérést küldeni, különben terheléses támadásnak (DDoS) tűnne a sok szálon indított nagy számú letöltés. (Még így is feltűnt a crawler nagy étvágya a külvilágnak: egy alkalommal érdeklődtek a hatóságok, hogy mi ez a nagy forgalom, pontosan mi is történik a szerverünkön. Ezután még "udvariasabb" beállításokat kellett eszközölni.)

A következő feladat a letöltött HTML tartalomból való értékes rész kinyerése volt. A korpusz a megtisztított szöveghalmazból épült fel. Az épülő korpuszban pedig többféle keresést, elemzést, tisztítást, szétválasztást kellett végezni azért, hogy minél többféle célra alkalmassá tegyük az adatbázist. Ahogy a kivonatban jeleztem, a kutatásom korpuszvezérelt (*corpus driven*) módon történt: a korpusz maga - vagy a vele kapcsolatos teendők - jelölték ki a következő feladatokat. Például a 6.2. fejezetbeli N-gram keresés rámutatott a korpusz duplikátumhibáira, így egy jobb szövegkinyerő algoritmust kellett fejlesztenem (2.3.2. fejezet), vagy az NP-felismerés kutatásomat is a korpusz motiválta az NP annotáció kapcsán (7. és 8. fejezetek).

[1] `https://www.google.com/trends/topcharts`

DOI:10.15774/PPKE.ITK.2016.003

2. Korpusz építése webről

Nagyméretű szövegkorpusz építéséhez a folyamatosan növekvő és mindenki számára elérhető web nagyon jó alapanyag. Az elmúlt 20 évben az interneten található tartalom mennyisége nagymértékben nőtt, nem csak a domének száma (1. táblázat) növekedett ugrásszerűen, hanem az internetes közösség is egyre aktívabban növeli a tartalmat (2. táblázat). Azonban a weblapokon található szöveg nyelvészeti szempontból (is) eltérő minőségű, jellegű, illetve sok, szöveggyűjtés szempontjából használhatatlan tartalom van benne (menüsorok, fejlécek, hirdetések), és az anyag sokféle szerveren található. Emellett a web tulajdonsága az is, hogy az oldalak időről időre változnak, frissülnek. Fel kell készülni arra, hogy egy adott oldal designja megváltozik, így a szöveges tartalom kinyerésének az ilyen típusú változásokat is követnie kell.

év	domének száma	növekedés aránya
1993	134	100%
1996	257 000	192 000%
2012	634 000 000	520 000 000%
2014	968 000 000	722 000 000%

1. táblázat. A domének számának növekedése (http://www.internetlivestats.com/total-number-of-websites/)

A legtöbb korpuszépítő eszköz azt az utat járja, hogy nagy mennyiségben letölt HTML formátumú lapokat, ezekből kinyeri a szöveget, majd egy utófeldolgozással megtisztítja a duplikátumoktól az adatbázist. Ez utóbbi két lépés kritikus a végeredményként előálló korpusz minősége szempontjából. A szövegkinyeréskor fellépő problémák egyike, hogy a zajos szöveg a későbbi feldolgozást nehezíti (például egy menüsor szöveggé alakítása minden mondatszegmentálót nehéz feladat elé állít). A másik probléma a duplikátumok jelenléte: a weben - akár egyetlen hírportálon belül is - teljesen azonos mondatok, szövegdarabok több lapon is szerepelnek (például top hírek vagy egyéb ajánlók), és ezek az egyforma részek többször is bekerülhetnek a korpuszba (bővebben a 2.3.1. és 9.1. fejezetekben). Az utólagos duplikátummentesítés a folyó szövegből kitöröl bizonyos részeket (amit duplikátumnak minősít). Így az eredmény nem összefüggő, koherens szöveg lesz, és a korpusz mondatai nem mindenütt kapcsolódnak egymáshoz. Ebben a fejezetben ezekről a problémákról lesz szó részletesen.

DOI:10.15774/PPKE.ITK.2016.003

év	tweetek száma/nap	növekedés aránya
2007	5000	100%
2010	50 000 000	10 000%
2013	500 000 000	100 000%
2015	870 000 000	174 000%

2. táblázat. A naponta küldött tweetek száma (http://www.internetlivestats.com/twitter-statistics/)

A fenti elvárások alapján olyan hangolható, robosztus letöltő robotra volt szükség, amely (1) képes nagy mennyiségben weblapokat letölteni, (2) kinyeri belőlük az értékes tartalmat, lehetőleg emberi beavatkozás nélkül, automatikusan.

2.1. Webes korpuszok minősége

A korpuszoknál - az automatikusan építetteknél különösen - fontos, hogy a tokenszámon túl mérni lehessen a korpusz minőségét. Elvárás a szöveg kohéziója, bekezdések és azokon belül a mondatok összefüggése, a duplikátum-mentesség, a minél tisztább, zaj- és reklámmentes, homogén nyelvű szöveg. Számos olyan statisztikai ismérvet (Biemann és mtsai. 2013) mutatok be a következőkben (*Tipikus statisztikák* és *indikátorok* felsorolások a 19. oldalon), melyeknek segítségével a korpuszok összehasonlíthatóak (Quasthoff, Goldhahn, és Heyer 2013), és amelyek rávilágíthatnak a korpusz minőségi problémáira. Amennyiben hasonló korpuszokat hasonlítunk össze (azonos nyelvű, méretű, műfajú), akkor ha a korpuszok ezen paraméterekben (*statisztikák* és *indikátorok*) eltérnek, akkor ez minőségbeli hibára utalhat. Másrészt, ha azonos nyelvű korpuszokat hasonlítunk össze, de eltérő műfajból (domén) származnak a szövegek, akkor a műfajjal korreláló paraméterek azonosíthatóak. Harmadrészt, eltérő nyelvű korpuszok esetén a paraméterek közötti korrelációk azonosíthatóak (tipológiai osztályok vagy nyelvcsaládokba való osztályozás). Ezen paraméterek átfogó képet adhatnak a korpusz állapotáról: tipikus korpuszhibákat, crawlerhibákat, kódolási- vagy akár mondatszegmentálási hibákat is képesek jelezni. Hogyan lehet egy korpusz ezen hibáit illetve minőségét kimutatni?

A minőségbiztosítás nagy kihívás, különösen ha korpuszok százairól beszélünk, mondatok millióiról, amelynek a kézi kiértékelése nehézkes lenne. Ennek egyik módja, ha extrém értékek szerepelnek bizonyos statisztikákban, ez jelezheti a zajos, problémás adatokat, amelyeknek további, mélyebb vizsgálata szükséges (például ha a korpusz 10 leghosszabb szava 1000 betű hosszúságú: akkor ezek nem szavak lesznek, hanem inkább adathiba). Szavak, mondatok, vagy akár egész dokumentumok lehetnek kétes minőségűek, és ezek eltávolítása nagyban javítja a korpusz minőségét. Más esetben egy adott paraméter nem egyenletes eloszlása vagy kiugró értéke lehet jó indikátora a korpusz egy problémájának. Például szélső esetben a 30-as ASCII kód alatti karakterek nagy száma utalhat arra, hogy bináris adat (mp3, avi) került a korpuszba; enyhébb formában a betűk eltérő eloszlása

DOI:10.15774/PPKE.ITK.2016.003

jelezheti, hogy más nyelvű szöveg is jelen van; ha a bekezdések átlagos hossza túl rövid, ez jelezheti címkefelhő vagy egyéb felsorolások tömegesen jelenlétét.

Tipikus statisztikák:

- szavak, mondatok, dokumentumok hosszának eloszlása
- karakterek vagy N-gramok eloszlása
- a nyelv bizonyos tapasztalati szabályaival való egyezés (Zipf-szabály)

Ha anomáliák fordulnak elő ezekben a statisztikákban, akkor meg kell keresni ezek okát, és további tisztításokat kell végezni az adatbázison. A következő jegyek jó indikátornak bizonyultak a lehetséges hibák felderítéséhez:

- C1: a korpuszban szereplő legnagyobb domén és mérete
- C2: a források száma adott idő alatt (folyamatos crawling esetén)
- W1: szavak hosszának eloszlása
- W2: a korpusz *n* leggyakoribb szava
- W3: a leghosszabb szavak listája az *n* leggyakoribb és leghosszabb szavak közül
- W4: nagybetűs stopword-ben végződő szavak
- A1: karakterek frekvencialistája
- A2: lehetséges hiányos rövidítések
- S1: legrövidebb és leghosszabb mondatok
- S2: mondatok hosszának eloszlása (szavakban és karakterekben egyaránt mérve)

Ezen statisztikák vizsgálata a következő típusú problémákat jelezheti:

- crawling problémák (C1, C2)
- heterogén nyelvű szöveg (W3, A1)
- rossz karakterkódolás (A1, W3, S1)
- közel azonos mondatok (S2)
- mondatszegmentálási problémák (W4, S1)
- egy adott terület túlreprezentálása (C1, W2)
- sok hibás mondat (W4, A2, S1, S2)

DOI:10.15774/PPKE.ITK.2016.003

A web alapján épített korpuszok rengeteg duplikátumot tartalmaznak, amely abból fakad, hogy sok portálon ugyanaz a tartalom más url alatt is elérhető (például az archívum illetve más rovat alatt is), és a hozzászólások idővel gyarapodnak: sem az url, sem a tartalom alapján nem tudja felismerni a duplikátumot egy algoritmus (Benko 2013). Ezért általában utólagos duplikátum-mentesítést (deduplicate) szoktak végezni: dokumentum-, bekezdés- illetve mondatszinten (Pomikálek 2011).

2.2. Letöltő robot – crawler

A projekt első részében egy crawlert készítettem C++ nyelven. A program egy paraméterben megadott weblapról indítva lapokat tölt le, az egyes domének HTML tartalmait külön kimeneti fájlokban gyűjti. A bejárt url-eket adatbázisban tárolja, és a weblapokról szerzi az újabb bejárható url-eket. A web egyik szépsége, hogy bármely pontján kezdve elméletileg bejárható az egész pókháló. Gyakorlatilag sok időbe telne, mire az egész hálózatot bejárná a program, ezért még a Google robotjához is hozzá lehet adni kézzel újabb oldalakat (*www.google.com/addurl*), hogy ne kelljen sokat várni, míg a robot véletlenül felfedezi. A crawlert elegendő egy ponton elindítani, és eddig még nem állt le a futása azzal, hogy elfogytak a lapok. Viszont az előfordult, hogy egy adott doménre szükség volt, így azt megcélozva újra el kellett indítani: ezáltal nem kellett kivárni, míg oda jut magától.

A lapok letöltése azonban csak az egyik feladata a programnak. Ezen túl még az is szükséges volt, hogy a crawler kinyerje a fő szöveges tartalmat a weblapokról. A tartalomkinyerés, melyet *boilerplate removal*-nak hív a szakirodalom, eltávolítja a fejlécet, láblécet, menüket, hirdetéseket a weboldalról, csak az értékes cikket menti a korpuszba. Ez a lépés kritikus az végeredmény korpusz minősége szempontjából: minél több ismétlődés, illetve boilerplate található benne, annál kevésbe használható.

Ezért a második lépés az eddigieknél jobb *boilerplate removal* algoritmus kifejlesztése volt, ami képes tiszta, duplikátummentes, ingyenes korpuszt készíteni, reklámok és más nem kívánt tartalom nélkül. A Mexikóban bemutatott új algoritmus neve **Aranyásó** (GoldMiner) lett (Endrédy és Novák 2013), amit a 2.3.2 fejezetben mutatok be.

A crawler 2013-ban indult, és fut azóta is. Korpuszának mérete jelen dolgozat írásakor 1 200 millió szó. A crawler mögötti adatbázis SQLite3 volt. Az SQLite3 kis méretű és egyszerűsített adatbázismotor erőssége, hogy könnyű vele fejleszteni és tesztelni, az adatbázist egyetlen, hordozható fájlban tárolja. Azonban az ugrásszerűen megnőtt adatméret miatt a crawler adatbázisát migrálni kellett MySQL adatbázis motor alá, amely már nagyobb adatmennyiség kezelésére képes. Bár a két adatbázis motor eltérő SQL szintaxist használ, a crawler képes bármelyikkel együttműködni: az adatbázis típusa (MySQL vagy SQLite3) paraméter a crawler számára.

DOI:10.15774/PPKE.ITK.2016.003

A crawler funkciói:

- HTML lementése
- szövegkinyerés (3 féle módon: Aranyásó, JusText, BTE)
- MySQL vagy SQLite3 adatbázisban tárolja az url-ek, lapok md5 hash-ét
- offline html lapokból is ki tudja nyerni a szöveget
- 4 szintű loggolás
- megadható kiindulási pont (url), ahonnan a crawler elindul
- beállítható, hogy más doménekre is átmenjen, vagy csak a kiinduló domént járja be
- megadható doménszűrő
- megadható tiltólista, a kihagyandó domének nevével
- figyelembe veszi a szerverek robots.txt-jét, amiben leírják, hogy mit szabad tőlük letölteni és mit nem
- megadható egy késleltetés, amennyi idő múltán kérdezhet a robot újra ugyanattól a szervertől (domén delay, hogy ne okozzon DoS támadást senkinek)
- hány szálon fusson a letöltés és feldolgozás
- állítható letöltési stratégia: új lapokat töltsön le vagy inkább változásokat keressen már letöltött lapokon
- az Aranyásó algoritmus (2.3.2 fejezet) tanulási egysége állítható: felsőszintű domén (valami.hu), aldomén (aloldal.valami.hu)
- az Aranyásó algoritmus bizonyos letöltött oldalszám után tanulja meg egy adott domén oldalainak szerkezetét. Ez az oldalszám állítható: ennyi letöltött oldal alapján tanulja meg az adott domént.

2.3. Sablonszűrés weblapokról

A webről nyert korpuszok építésekor az általában dinamikusan generált HTML-tartalomból történő szövegkinyerés nem triviális feladat a különböző oldalakon ismétlődő rengeteg irreleváns sablonos tartalom miatt. Ezt a feladatot az angol irodalomban *boilerplate removal*-nek, sablonszűrésnek nevezik. A művelet lényege, hogy a HTML-tartalomból csak az értékes részeket igyekszik kiszűrni, a menük, fej- és láblécek, reklámok, a minden oldalon ismétlődő struktúra kiszűrésével.

A fenti feladatra számos algoritmus létezik. A következőkben bemutatom milyen boilerplate megoldásokat használtam a crawlerben, és a 2.4. fejezetben pedig az elérhető megoldásokat veszem sorra. Ez utóbbiak a méréseim alapján nem adtak megfelelő minőségű eredményt, de összehasonlításnak jó szolgálatot tettek.

DOI:10.15774/PPKE.ITK.2016.003

2.3.1. Kiindulási alap: JusText algoritmus

A JusText algoritmus (Pomikálek 2011) a HTML-tartalmat bekezdésekre bontja a szöveget tartalmazó címkék (HTML tag) mentén. Minden egységben megszámolja a benne szereplő linkek, stopword-ök és szavak számát. Ezek alapján osztályozza őket: bizonyos küszöb mentén *jó*, *majdnem jó* illetve *rossz* kategóriákba. Majd a *jókkal* körülvett *majdnem jók* szintén bekerülnek a *jók* közé, és így születik meg az értékes szöveg: a jó bekezdések. Az algoritmus kiváló minőséget ad még extrém oldalakon is.

A JusText Pythonban készült, de újraírtam az algoritmust C++ nyelven, mert a crawlert abban programoztam. Ezt a portolást megosztottam a fejlesztői közösséggel[2].

Az algoritmus használatával magyar hírportálokról kinyert tartalmat vizsgálva azonban feltűnt az a probléma, hogy a kapott korpusz még mindig a várthoz képest nagyon erősen túlreprezentált kifejezéseket tartalmaz. Nem volt világos, hogy egy hírportálon miért szerepel a világ egyik legismertebb politikusa kevesebbszer, mint egy helyi híresség, vagy akár egy kattintásvadász cím. Az várnánk, hogy egy cikk címe csak egyszer szerepeljen a korpuszban, a híresség pedig ismertségük arányában. A várakozással ellentétes furcsa jelenséget az alábbi 4-gram példák (1–2) mutatják:

(1)
Utasi Árpi-szerű mesemondó . (10587 db)
cumisüveg potenciális veszélyforrás . (1578 db)
Obama amerikai elnök , (292 db)

(2)
etióp atléta: cseh jobbhátvéd (39328 db)
Barack Obama amerikai elnök (2372 db)
Matolcsy György nemzetgazdasági miniszter (1633 db)
George Bush amerikai elnök (1626 db)

Azt találtam, hogy a problémát elsősorban a cikkek alján található (kapcsolódó) cikkajánlók nem megfelelő kiszűrése, másrészt a kizárólag a cikkajánlókat tartalmazó oldalak okozzák. Ezért szükségessé vált az algoritmus módosítása, ezen problémák kiküszöbölésére.

[2] `https://github.com/endredy/jusText`

DOI:10.15774/PPKE.ITK.2016.003

2.3.2. Továbbfejlesztett változat: az Aranyásó algoritmus

A problémát az adott egyedi weboldalaknál magasabb szintre lépve sikerült megoldani a korábbinál hatékonyabban. Eleinte abba az irányba indultam, hogy az egyes weblapok nem kívánt részeit távolítsam el. Mikor nem sikerült átütő eredményt elérni, akkor jutott eszembe, hogy miért a rosszat keressük, miért nem az értékes részeket? Ahogy az életben is érdemes a jót keresni és nem a rosszat, így tettem a weboldalak esetén is: így született meg az Aranyásó algoritmus[3]. Egy aranyásó több köbméter salakot is átszitál azért, hogy egy pici aranyrögöt megtaláljon, és ha megvan, örül neki. Hasonlót tesz ez az algoritmus is: sok lapot átnéz egy minta után kutatva, amit aztán megtanulva az adott weboldalra alkalmazhat.

A megoldás azon alapul, hogy egy adott webdoménen/aldoménen belül a dinamikusan generált weboldalak, illetve url-ek nem szöveges tartalma (pl. a HTML-kód) általában tartalmaz közös mintázatokat, hasonló designt, amelyeket azonosítva megtalálható az értékes tartalom. Az algoritmus az adott domén oldalaiból mintát vesz, és megpróbálja megkeresni azt a HTML-címkét, amelyen belül (az oldalak zömében) a cikk található, különös tekintettel azokra a mintákra, amelyek az oldalakon ismétlődnek. Például gyakori a hírportálokon, hogy a cikk alján feltüntetik a legnépszerűbb 5 cikk ajánlóját. Ezeket a szövegeket a korábbi sablonszűrő algoritmusok nem szűrik ki (pl. a *JusText* a cikk részének veszi őket), mert önmagukban nem rossz szövegek, viszont duplikátumot okoznak majd a korpuszban, erősen felülreprezentálva az ezekben található szöveget.

Az Aranyásó algoritmus egyfajta előtétként működik a *JusText* előtt: csak azt a HTML kódot adja át neki, ami nagy eséllyel az értékes részt tartalmazza, a többit már eleve eldobja, így a sablonszűrőnek jelentősen leegyszerűsíti a feladatát. Más szavakkal: minden doménre megtanuljuk azt a HTML szülőcímkét, amely csak a cikket tartalmazza, majd csak ezen HTML DOM csomópont tartalmát adjuk oda a *JusText* sablonszűrő algoritmusnak. Előnye ennek a módszernek az, hogy azon oldalak, ahol nincs cikk (tematikus nyitólapok, címkefelhők, keresőlap eredmények stb.), ott az algoritmus nem ad semmit, hiszen a doménre jellemző cikk címke nincs jelen. Így az algoritmus automatikusan kiszűri ezeknek a lapoknak a tartalmát, ami örvendetes. Azokon lapokon pedig, ahol valóban van cikk, a sablonszűrő algoritmus már csak a lényegi tartalmat kapja, erősen megkönnyítve a dolgát.

Első lépésként megtanuljuk a doménre jellemző HTML-mintázatot, ez a tanulófázis. Az algoritmus letölt pár 100 oldalt, és lefuttatja rajtuk a *JusText* sablonszűrő algoritmust, amely minden oldal tartalmát bekezdésekre bontja és értékeli (értékes szöveg vagy boilerplate). Az Aranyásó algoritmus az egyes kinyert bekezdéseket átnézi: ismétlődnek-e más oldalakon az összes letöltött mintában. Más szavakkal, a letöltött lapok összes, *JusText* által értékesnek

[3] `https://github.com/endredy/GoldMiner`

DOI:10.15774/PPKE.ITK.2016.003

minősített bekezdését végignézi, és jelöli benne a duplikátumokat. A duplikátumok olyan bekezdések, amelyek nem a fő tartalomban helyezkednek el, hanem általában körülveszik a fő tartalmat (kommentek főcíme, "Anikó névnapja van", "Mindeközben" rovat néhány friss eleme, stb.) Ezután az algoritmus a duplikátumokat rossz bekezdésnek minősíti, majd megkeresi a jó bekezdések legközelebbi, közös szülő címkéjét a DOM hierarchiában. Ez úgy történik, hogy általában egy oldalon a jó bekezdéseket kölülveszik a duplikátumok. Emiatt a jó és a rossz bekezdések közötti sávban lehet megtalálni a jó vágómintát, ami elkülöníti a cikket, fő tartalmat. Így az algoritmus azt csinálja, hogy rossz bekezdések után és az első jó bekezdés fölötti HTML kódban kigyűjt többféle (1-5) hosszúságú mintát. Azok a minták nem alkalmasak, amelyek ezen pozíció előtt is megtalálhatóak a lapon (például `<div>`). Ugyanezt megtesszük a cikk végén is: a jó bekezdések után (és a rossz bekezdések előtt) kigyűjt mintákat, itt is 1-5 hosszúakat (például az egy hosszú minta: `</div>`; kettő hosszú: `</ul></div>`; három hosszú: `<!-- vege --></ul></div>`, stb.) A tanulási folyamat végén az összes kezdő és záró mintát összesítjük: melyik hányszor szerepelt az egyes lapokon. A leggyakoribb minta annál jobb eséllyel lesz az optimális (doménra jellemző) vágási pont, minél több cikk, fő tartalommal rendelkező lap volt a tanulási mintában. A HTML minták keresését reguláris kifejezéssel valósítottam meg: ötféle hosszúságban. Nem csak a HTML címkék, hanem a közöttük lévő kód is lehet a minta része.

A tanulófázis végén a leggyakoribb, pozitív minősítésű szülő címke lesz a győztes: ezen nyitó és záró címke (HTML minta) között található az adott doménen a cikk. Ezért ezeket a címkéket eltároljuk, és minden, erről a doménről származó lapon ezen címkék között keressük a cikket.

Nem kapunk optimális eredményt, ha teljes HTML DOM csomópontokat tanulunk meg, nem pedig egyszerű HTML mintákat. Előfordul ugyanis, hogy a teljes cikket tartalmazó szülő címke nem kívánt bekezdéseket is tartalmaz. Ilyen esetben az algoritmus nem találná meg az optimális vágási pontokat. Ezért nem a DOM hierarchiában, hanem egyszerűen a HTML kódban keressük az optimális pontokat, amely több címkéből álló sorozat is lehet. (Például a DOM hierarchiában a `<div class="article">` csomópont tartalmazza ugyan a teljes cikket, de nem kívánt részeket is. Ezért a HTML kódban próbálunk közelebbi pontot keresni.) Miután ezt kiválasztottuk, az algoritmus (természetesen a tanulófázis oldalaiból is) csak az így kivágott rész tartalmát adja át a sablonszűrőnek.

Az Aranyásó csak azokból az oldalakból tanul, amelyben a kinyert bekezdések hossza elér egy minimumot: pár doménen, ahol rendkívül gyakoriak a tényleges tartalmat nem adó gyűjtőlapok, e nélkül nem sikerült a megtanulni a legjobb vágási pontot.

DOI:10.15774/PPKE.ITK.2016.003

A 2. ábra segítségével egy példán keresztül szeretném bemutatni az algoritmus egyes lépéseit:

(1) Egy doménből letöltünk pár 100 oldalt.

(2) A *JusText* algoritmussal kinyerjük belőlük a bekezdéseket. Kapunk tehát olyan bekezdéseket, amelyeket jó szövegnek minősített a JusText, de természetesen ezekben lesz nem kívánt tartalom (olyan, ami tévesen kerül bele, cikknek véli a JusText, de csak kapcsolódó cikk, komment, stb.) Az ábrán piros és zöld színnel jelöltem ezeket a bekezdéseket.

(3) Megkeressük az ismétlődő bekezdéseket. Ha egy bekezdés többször is előfordul (duplikátum), az azt jelenti, hogy több lapon is előfordul, azaz jó eséllyel nem a fő tartalomhoz tartozik. Ezeket az 2. ábrán piros színnel és szaggatott nyilakkal jelöltem: jól láthatóan körülveszik a cikket (a cikk bekezdései zöld kiemeléssel rendelkeznek). Például "*Kapcsolódó cikkek*" vagy "*Gyerekek sérültek Tiszabercelnél*"

(4) Megkeressük ezen a lapon a lehetséges jó vágómintákat. A rossz (piros) bekezdések és az egyedi (zöld) bekezdések között van valahol a cikk kezdetét jelző minta. Ezért az első egyedi (zöld) bekezdés előtt többféle hosszúságú HTML mintát kigyűjtünk: `<h1>`; `<div id="article-head"...;` `<a id="temacimke"`; stb. Ugyanezt megtesszük a cikk végén is: az egyedi (zöld) bekezdések után és a duplikátumok (piros) előtti szintén kigyűjtünk mintákat, ezek fogják jelezni a cikk végét. Esetünkben ebből kettő van: `<div id="article-related">` és `<h3>`.

(5) Ha az összes, tanulásra letöltött lapon ezt megtettük, akkor összesítjük a nyitó és záró mintákat: melyik hányszor szerepelt. A leggyakoribb lesz a győztes, ezt a mintát megjegyezzük az adott doménhez. A 2. ábrán ez a minta folytonos nyilakkal van jelölve.

Erről a doménról érkező lapoknál a megtanult HTML-minta *közötti* tartalmat adjuk csak oda a JusText algoritmusnak. Így csak a lényeges, cikket tartalmazó HTML-kódot kell feldolgoznia a boilerplate-nek, a hatásfoka nagyban javul.

DOI:10.15774/PPKE.ITK.2016.003

2. ábra. Példa az Aranyásó algoritmus egyes lépéseire a HTML-kódban

2.3.3. Eredmények

Az Aranyásó algoritmussal jelentősen sikerült csökkentenünk a legyűjtött korpuszban szereplő ismétlődéseket. Három doménen (origo.hu, index.hu, nol.hu) való futtatással 2000 oldal letöltése után a bejárást leállítva kapott eredményeinket a 3. táblázatban foglaljuk össze.

Jelen mérésben az egyedi mondatok arányával mérem a sablonszűrés hatékonyságát. Felmerülhet a kérdés, hogy ez jó metrika-e? Nehéz automatizált, jó metrikát definiálni, mert a legjobb az emberi kiértékelés, ahol kézzel ellenőrizzük a szövegkinyerés pontosságát. Az Aranyásó algoritmus kiértékeléséhez létre is hoztam egy ilyen gold standardet, ahol kézzel jelöltem 4 domén 70 weblapját[4], angol nyelvre (Endrédy és Novák 2013). Jelen kiértékelés azonban sokkal több lapon történt, és a tapasztalataim alapján az egyedi mondatok aránya hatékonyan jelzi a sabonszűrés hibáit. Azaz ha ismétlődő mondat van a kinyert szövegekben, az jó eséllyel abból fakad, hogy a sabonszűrés hibázott: olyat tartalmat is bevett, amit nem kellett volna. Többnyire ez a jelenség több lapon is megtörténik, így ez a külső (nem kívánt) szöveg más cikkhez is hozzá fog csapódni. Természetesen elképzelhető olyan zaj, ami egyedi.

[4] https://github.com/ppke-nlpg/CleanPortalEval

DOI:10.15774/PPKE.ITK.2016.003

Algoritmus		Kinyert mondatok száma	Egyedi mondatok száma	Összes karakterszám	Egyedi mondatok karakterszáma	Egyedi mondatok aránya	Egyedi mondatok aránya karakterszámban
origo.hu	összes szöveg	264 423	63 594	16 218 753	7 048 011	24%	43%
	BTE	60 682	33 269	12 016 560	7 499 307	54%	62%
	JusText	58 670	30 168	8 425 059	4 901 528	51%	58%
	Aranyásó	22 475	21 242	3 076 288	3 051 376	**94%**	**99%**
nol.hu	összes szöveg	509 408	144 003	25 358 477	12 570 527	28%	49%
	BTE	154 547	107 573	24 292 755	13 544 130	69%	55%
	JusText	186 727	128 782	14 167 718	11 665 284	68%	82%
	Aranyásó	162 674	123 716	12 326 113	11 078 914	**76%**	**89%**
index.hu	összes szöveg	232 132	55 466	9 115 415	4 542 925	23%	49%
	BTE	51 713	26 176	5 756 176	4 061 697	50%	70%
	JusText	40 970	29 223	4 371 693	3 441 337	71%	78%
	Aranyásó	13 062	11 887	1 533 957	1 489 131	**91%**	**97%**

3. táblázat: Sablonszűrő algoritmusok összehasonlítása az egyes doméneken

A 3. táblázatban látható, hogy az Aranyásó algoritmus mindhárom hírportál esetén jobban teljesített. Különösen az origo.hu esetén tudott sokat javítani (+41%), aminek az az oka, hogy ezen az oldalon a fő tartalom alatti cikkajánlók a többi algoritmust megtévesztik, és ezeket - tévesen - a cikkhez csatolják. Ez a 2.3.1 fejezetben leírt a korpuszhibához vezet: *Obama elnök*nél gyakrabban fordul elő a *cumisüveg*. Az Aranyásó algoritmus ezt sikeresen javítja.

2.4. Kapcsolódó kutatások

Ebben a fejezetben néhány releváns boilerplate removal algoritmust tekintek át, amelyek szabadon hozzáférhető implementációval rendelkeznek és emiatt kiértékelésnél jól használhatóak (baseline). Számos jó ötletet is tartalmaznak, és ezek alapján egy folyamat rajzolódik ki az áttekintésben: a módszerek gyakran az előző eredményeire épülnek. Néhány algoritmust ezek közül újraimplementáltam C++ nyelvben, így könnyen és gyorsan kiértékelhetőek voltak.

Body Text Extraction (BTE) algoritmus

A BTE (Finn, Kushmerick, és Smyth 2001) a weblap azon általában jellemző tulajdonságát használja ki, hogy a boilerplate tartalomban sűrűbben szerepelnek a HTML címkék. Ezért megkeresi azt a leghosszabb részt, ahol a címkék száma minimális. Az ötlet egyszerű, azonban sokszor téved, és nem képes szöveget kinyerni olyan helyzetekben, ahol az értékes részben a

DOI:10.15774/PPKE.ITK.2016.003

feltételezéssel ellentétben mégis sűrűbben fordulnak elő HTML címkék, például ha táblázat szerepel benne, vagy egy reklámokkal szabdalt cikk esetében. Ilyenkor az értékes tartalom jelentős része (akár az egész) elveszhet, esetleg helyette teljes egészében irreleváns tartalom jelenik meg.

A boilerpipe algoritmus

A boilerpipe algoritmusnak (Kohlschütter, Fankhauser, és Nejdl 2010) az az érdeme, hogy a szerzők kísérleti úton bebizonyították, hogy a boilerplate tartalmat hatékonyan lehet azonosítani az egyszerű szöveg tulajdonságainak jó kombinációival.

500 annotált dokumentumot tartalmazó tanítókorpuszt használtak (elsősorban a Google News-ból) arra, hogy megtalálják a leghatékonyabb tulajdonság-kombinációkat. Megpróbálták a cikkeket szövegjegyekkel (*shallow text feature*) kinyerni, 8-10 különböző kombinációt kipróbáltak, majd kiértékelték az eredményeket.

Kísérleteikben a szó- és a linksűrűség tulajdonságok adták a legjobb eredményt (92% F-érték). Továbbá, a módszer nagyon gyors és nem igényel előfeldolgozást. Mind a tanítóanyag, mind az eszköz letölthető.

JusText algoritmus

A JusText algoritmus (Pomikálek 2011) a HTML-tartalmat bekezdésekre bontja a szöveget tartalmazó címkék mentén. Minden egységben megszámolja a benne szereplő szavak, linkek és stopword-ök számát. (A stopword azokat a szavakat jelenti, amelyeket egy rendszer figyelmen kívül hagy a feldolgozás során. Általában ezek egy nyelv leggyakoribb szavai (*a, az, és, hogy* stb.) és feladatonként eltérő lehet a listájuk.) Ezek alapján osztályozza őket: bizonyos küszöb mentén *jó*, *majdnem jó* illetve *rossz* kategóriákba. Majd a jókkal körülvett majdnem jók szintén bekerülnek a jók közé, és így születik meg az értékes szöveg: a jó bekezdések. Az algoritmus elfogadható minőséget ad még extrém oldalakon is.

DOI:10.15774/PPKE.ITK.2016.003

2.5. Összefoglalás

Készítettem egy crawlert, amellyel nagyméretű szövegkorpuszok készíthetők automatikusan. Az egyes lapok HTML-tartalmán túl az oldalak szövegét is külön kinyeri, eltávolítva belőlük az ismétlődő vagy nem kívánt tartalmakat (fejléc, lábléc, menü, reklámok stb.). Ehhez létrehoztam az Aranyásó algoritmust.

Kapcsolódó tézisek:

1. tézis: Megalkottam a GoldMiner algoritmust, ami az eddigieknél hatékonyabban nyeri ki a weblapokról a cikkeket.

1.a tézis: Létrehoztam egy új crawlert, amely korpuszt tud építeni.

A tézishez kapcsolódó publikációk: [1], [9]

DOI:10.15774/PPKE.ITK.2016.003

3. Ékezetesítés

A kommentkorpusz sok helyen ékezet nélküli szöveget tartalmazott, ezért a későbbi automatikus elemzés elősegítésére szükségessé vált egy ékezetesítő szoftver elkészítése. A mobileszközök elterjedésével az utóbbi időben ismét megnőtt az ékezet nélkül írt szövegek aránya, amelyek nyilvánosan elérhető felületeken is megjelennek. Ennek oka egyrészről a beviteli módban keresendő: a felhasználó billentyűzete például nem magyar vagy a felhasználó olyan mobil eszközt használ, amelyen az ékezetes betűk begépelése nehézkes lenne. (Például külföldön van a felhasználó, és ott nem magyar billentyűzetet használ.) Másik tipikus oka felhasználói szokás: így egyszerűbb. Ez főleg SMS üzeneteken, chatszobákban (vagy az email hőskorában) szocializálódott felhasználók esetén fordul elő: ott még nem volt ékezet, és ezt a felhasználó megszokta. Időnként még ma is előfordul, hogy kódolási hibás emailt kapunk, amiben olvashatatlanok lettek az ékezetes betűk. Vannak olyan haladó felhasználók, akik szemében az angol billentyűzet a biztos, és nem csak fájlnevekben nem használ ékezeteket, hanem emailekben sem. Még ma (2015) is találkozhatunk ékezet nélküli szövegekkel:

```
Az "addig nyujtozkodj, ameddig a takarod er" mondas
eletszemlelette valasa megovhat sokunkat a nagyobb gondoktol.
Ezt kell megertenie mindenkinek, a csaladoktol a kormanyokig.
```

Az ember számára általában nem jelent problémát az ékezetek nélkül írt szövegek értelmezése, de az automatikus nyelvfeldolgozó eszközök nincsenek felkészülve az ilyen szövegek feldolgozására. Ahhoz, hogy a kommentkorpusz igazán használható legyen, szükséges volt egységes, ékezetes szöveggé alakítani. Ezért kísérletet tettünk arra, hogy ezen szövegeket automatikus módszerrel, minél jobb minőségben ékezetes szöveggé alakítsuk. Ez magyar szövegek esetén a következő ékezetes betűk visszaállítását jelenti: *í, ö, ü, ó, ő, ú, é, á, ű.*

3.1. A Humor szóelemző

A magyarhoz hasonló agglutináló nyelvek esetében nehézkes lenne a lehetséges szóalakok milliárdjait felsorolni, és így eltárolni (a magyarban például igék esetén egy szó lehetséges releváns alakjainak száma a képzők produktivitása miatt akár az ezres nagyságrendet is elérheti). Bár egy korpuszban nem fordul elő minden lehetséges alakja egy szónak, mégis minden alakot tudni kell kezelni. Ezért a lehetséges szóalakok szabályrendszeren (nyelvtanon) alapuló definiálása nyújthat hatékony megoldást a szóelemzés problémájára.

DOI:10.15774/PPKE.ITK.2016.003

Jelen ékezetesítő megoldás a Humor (High-speed Unification MORphology) szóelemzőre (Prószéky és Kis 1999), illetve az ahhoz készült magyar szóalaktani adatbázisra épül (Novák 2003). A Humor elemző első változatait Tihanyi László (MorphoLogic) készítette el, majd Pál Miklós (MorphoLogic) újraírta, C++ nyelven. Az eszköz zárt forráskódú.

A Humor elemző a szavakat morfokra (minimális szóelemek, morfémák konkrét lehetséges alakjaira) bontja, miközben ellenőrzi a szomszédos morfémák kapcsolatait, illetve a teljes szószerkezet helyességét. Az egyes morfémáknak tulajdonságai, jegyei vannak, és ezek segítségével megszorítások fogalmazhatók meg a szomszédos morfémák között, amelyek leírják, hogy mely elemek kapcsolódhatnak egymással. Elemzés közben az elemző folyamatosan ellenőrzi, hogy a felismert szóelemek tulajdonságai egymással összeférnek-e, illetve, hogy az adott elemzés a lehetséges morfémaszerkezeteket megadó és a távoli morfémák közötti megszorításokat is leíró grammatikai leírásnak is megfelel-e.

A Humor egyes elemzései morfok sorozatából állnak. Minden egyes morfnak van egy lexikai (szótári) és egy felszíni alakja (amilyen alakban a morféma az adott szóalakban megjelenik), és mindegyikhez tartozik egy kategóriacímke. A lexikai és a felszíni alak különbözhet. Csak akkor szerepel az elemzésben mindkét alak, ha különböznek. A 4. táblázatban látható egy elemzés és az egyes részeinek értelmezése. A morfémák alapján kiszámolható a szó töve is, de erről a 4. fejezetben lesz szó.

Elsőként röviden bemutatom a Humor-elemzés értelmezését, majd pedig azt, miképpen lehet ezt ékezetesítésre használni. A Humor elemzése a következő formátumú:

<morféma 1><címke> + <morféma 2><címke> + ... + <morféma n><címke>

Ha a lexikai alak eltér a felszíni alaktól, akkor a *<morféma><címke>* helyett ez áll:

<lexikai alak><címke>=<felszíni alak>

Az elemző gyakran több elemzést ad ugyanahhoz a szóhoz. Nemcsak a többértelmű szavak esetében fordul ez elő, hanem gyakran ugyanazon lexéma paradigmájának különböző tagjai is egybeesnek (pl. *keresnék* 'ők azt', ill. 'én valamit'). A program gyakran váratlan elemzésekkel is megörvendeztet, amire nem is gondolunk, például a *fejetlenség* szó utolsó vagy *szerelem* második elemzése a 5. táblázatban.

DOI:10.15774/PPKE.ITK.2016.003

***elmentek* szó elemzése**	
el[IK]+megy[IGE]=men+tek[t2]	
morf	**jelentése**
el[IK]	*el* morféma, igekötő (IK)
megy[IGE]=men	*megy* lexikai alak itt eltérő, *men* felszíni alakkal, ige
tek[t2]	jelen idő többes szám 2. személy kijelentő mód

4. táblázat. Példa egy Humor-elemzés értelmezésére

szó	**elemzés**	**értelmezés**
várnak	vár[FN]+nak[DAT]	*vár* mint főnév, ‘annak’
	vár[IGE]+nak[t3]	*vár* mint ige, ‘ők …’
alma	alma[FN]+[NOM]	*alma*
	alom[FN]=alm+a[PSe3]+[NOM]	*alom* (az állat *alma*)
fejetlenség	fejetlenség[FN]+[NOM]	‘káosz’
	fejetlen[MN]+ség[_PROP]+[NOM]	‘kaotikus mivolt’
	fej[FN]+etlen[_FFOSZT]+ség[_PROP]+[NOM]	‘hogy nincs feje’
	fej[IGE]+etlen[_IFOSZT]+ség[_PROP]+[NOM]	‘hogy nincs megfejve’
szerelem	szerelem[FN]+[NOM]	*szerelem* főnév
	szerel[IGE]+em[Te1]	*szerel* ige (*szerelem* a biciklimet)
	szer[FN]+elem[FN]+[NOM]	*szer* + *elem* összetett szó

5. táblázat. Példa a Humor többértelmű elemzéseire

3.2. Ékezetesítés morfológiai elemzés alapján

Jelen feladatnál ékezeteket szeretnénk tenni a bemeneti szóra, az elemző segítségével. A feladat megoldásához úgy módosítottuk a morfológiai elemző lexikonját, hogy a felszíni alakok helyére az ékezet nélküli felszíni alakok kerültek, a lexikai alakokat pedig az ékezetes felszíni alakokkal helyettesítettük. Ez a lexikonmódosítás gyakorlatilag azt jelentette, hogy a Humor lexikonforrásban a "felszíni alak" oszlopot bemásoltuk a "lexikai alak" oszlop helyére, ezután a "felszíni alak" oszlopból töröltük az ékezeteket. A lexikonmódosítást Novák Attila végezte. Ékezet nélküli szavak elemzésekor így a morfémák lexikai alakjai hozzák az ékezetes alakot (6. táblázat). Ettől a módosítástól természetesen az elemző csak ékezet nélküli szavakat tud majd elemezni, de jelen feladatnál éppen ez a cél. A 4. fejezetben leírt viszonylag bonyolult tövesítési algoritmus helyett egyszerűen a lexikai alakokat kell konkatenálni, és megkapjuk az ékezetes alakot. Ezt az átalakítást mutatja be a 6. táblázat, ahol az ékezet nélkül

DOI:10.15774/PPKE.ITK.2016.003

érkező szavakat megelemzi a (módosított lexikonú) Humor elemző, és az elemzés lexikai alakjai tartalmazzák az ékezetes alakot.

Az algoritmus ötlete egyszerű, nem érinti a Humor motort, csupán az adott nyelvet leíró tárolójában (lexikon fájl) kell módosítani. Ezt a funkciót a 4. fejezetben bemutatásra kerülő tövesítő eszközben valósítottam meg: a modul rendelkezik egy ékezetesítő üzemmóddal, amely ékezet nélküli bemenetre az elemzés egyes lexikai alakjait összegyűjti, az így kapott ékezetes alternatívákat pedig visszaadja.

	bemenet	**Humor elemzés**	**kimenet**
elemző lexikonnal	kutyának	kutya[FN]=kutyá+nak[DAT]	
	eltávolításuk	el[IK]+távolít[IGE]+ás[_IF]+uk[PSt3]+[NOM]	
ékezetesítő lexikonnal	kutyanak	kutyá[FN]=kutya+nak[DAT]	kutyának
	eltavolitasuk	el[IK]+távolít[IGE]=tavolit+ás[IF]=as+uk[PSt3]	eltávolításuk

6. táblázat. Példa a Humor-elemzésen alapuló ékezetesítésre

3.3. A lehetséges alternatívák kezelése

Az ékezetesítés szempontjából az számít többértelműségnek, ha egy adott (ékezet nélküli) szónak többféle ékezetes alakja lehetséges. A 7. táblázatban látható, hogy az emberek számára is nyilvánvaló többértelműségek mellett olyan alakok is előállhatnak, amit elméletileg valóban le lehet írni, de soha nem (vagy nagyon ritkán) használjuk: *ősszé, villamosmegállóbán, címkéjé* stb. Ez utóbbiak abból is adódnak, hogy a Humor produktívan megengedi a szóösszetételeket, és így olyan szavakat is összeilleszt, amelyek számunkra meglepőek (például *címkéje=cím+kéje*, amelyhez felkínálja a *címkéjé* alakot). A 7. táblázatban látható ékezetes alternatívák gyakoriságát a webkorpuszból vettük (Halácsy és mtsai. 2004; Kornai és mtsai. 2006). Jól látható, hogy bizonyos alakok nem (vagy nagyon ritkán) fordulnak elő a valóságban.

Az alternatívák kezelésénél annyit szeretnénk elérni, hogy a jó, legnagyobb eséllyel helyes alak legyen az első. A kevésbé jó ötleteket nem akarjuk letiltani, elegendő, ha hátrébb soroljuk őket. Ennélfogva a nagyon ritka ékezetes alakokat egyszerű szógyakoriság alapján hátrébb sorolhatjuk.

DOI:10.15774/PPKE.ITK.2016.003

Ékezet nélküli szó	Ékezetes alternatívák	Előfordulás
vereb	veréb	743
	véreb	111
fokabel	főkábel	7
	fókabél	3
erintettel	érintettel	264
	érintettél	12
toroljuk	toroljuk	7
	töröljük	1 321
reszletet	részletet	2 722
	részletét	1 757
ugy	úgy	504 105
	ügy	24 404
úr	úr	87 525
	űr	1 466
ossze	össze	124 872
	ősszé	5
testuket	testüket	1 022
	testükét	0
	testűket	0
	testűkét	0
cimkeje	címkéje	114
	címkéjé (összetett szó: cím+kéjé)	0

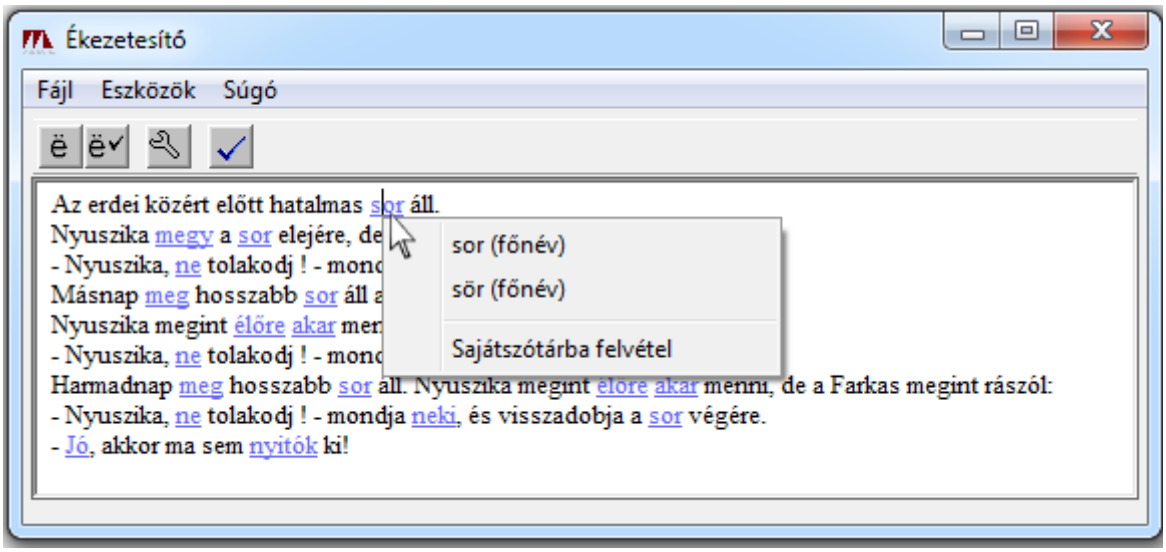
7. táblázat. Többértelműségek az ékezetesítésben, a webkorpuszbeli előfordulásokkal

Három módszert próbáltam ki a többértelmű ékezetes alternatívák kezelésére. A legegyszerűbb, amikor az első elemzést használjuk. A második módszer szerint gyakoriságalapú döntést hozunk. Pusztán a szóalakok korpuszbeli gyakorisága alapján jól kiszűrhetőek a téves ékezetes alakok: pl. a *villamosmegállóbán* típusú álösszetételek (*villamos-megálló-bán*), ami csak a gép számára alternatíva, emberek nem használják. A gyakoriságot korpuszmintából vettem: az előforduló ékezetes szavakhoz letároltam a töveik gyakoriságát. A tő tárolása nem csak kisebb tárhelyet igényel (az összes, többféleképpen ékezetesíthető magyar szó tárolása költséges lenne), hanem kereséskor is hatékonyabb megoldás. Nem támaszkodhatunk azonban kizárólag a szóalakok korpuszbeli gyakoriságára, hiszen – többek között a nyelv ragozó mivolta miatt – teljesen értelmes szóalakok sem szerepelnek még igen nagy korpuszokban sem. Ezért a lemmák és a toldaléksorozatok gyakoriságát is figyelembe vettem a rangsorolásnál – a felszíni szóalakénál kisebb súllyal, lényegében csak abban az esetben, ha több helyes ékezetes alaknak is ugyanaz a lemmája. Például *bizalmasan* és *bizalmasán* esetén a töveik azonosak, így a tő tárolása nem tud segíteni. Így abban az esetben, ha a toldalék többféleképpen ékezetesíthető (még ha - jelen esetben - a tő ékezet nélküli is), akkor a tanulási fázisban nem csak a tő-, hanem a teljes alak gyakoriságát tároltam le. Teszteléskor pedig nem csak a tőgyakoriságot, hanem a teljes ékezetes alak gyakoriságát is figyelembe vettem az ékezetes alakok sorbarendezésénél. Ezért a *bizalmasán* alak hátrébb került (mert ritkábban fordult elő), így az első javasolt alak a *bizalmasan* lett.

DOI:10.15774/PPKE.ITK.2016.003

Végezetül egy betűhármasokon (trigramokon) alapuló alternatívaválasztási módszert is kipróbáltam: tanításkor az egyes ékezetes szavakat (illetve ha egyértelmű, akkor tövüket) a szomszédos szavaik trigramjaival együtt tároltam le. Használatkor pedig ugyanígy a szomszédos szavak trigramjaival kerestem meg a legtöbb trigrammal egyezőt. Így számolja ki a kérdéses szó környezete alapján a legvalószínűbb ékezetes alternatívát. Például a tanulási fáziban a "*három veréb csivitel*" esetén a *veréb* ékezetes alakhoz letárolom a *három* és az *csivitel* trigramjait is (*hár, áro, rom*, illetve *csi, siv, ivi, vit, ite, tel*). Teszteléskor pedig egy adott szó ékezetes alternatíváit aszerint sorrendezi, hogy melyiknek van több egyezése a jelenlegi szomszédos szavak trigramjai és a megtanult ékezetes alak szomszédos szavainak trigramjaival. Tehát a "*három véreb ugat*" eldöntésekor a *három* és az *ugat* trigramjait veti össze a *veréb* és a *véreb* megtanult kontextustrigramjaival, és amelyiknél több egyező trigram van, az lesz az első javaslat.

Az ékezetesítő modul fölé készítettem egy grafikus programot is (3. ábra), amelyiknek a segítségével plain text és rtf formátumú ékezet nélküli szövegek ékezetesíthetők. Amelyik szónál egy ékezetes alak van, ott automatikusan megtörténik az átalakítás. Ahol több ékezetes alak is elérhető, azt más színnel jelzi a program. Ezeket szükséges átnézie a felhasználónak.



3. ábra Az. ékezetesítő alkalmazás egy többértelmű szónál

Az ékezetesítés kiértékelésénél az alap (baseline) az ékezet nélküli szöveg volt, azaz amikor a program nem csinál semmit. Egy 67 ezer szavas korpuszon értékeltük ki a 2,8 millió szavas korpuszon betanított modelleket (8. táblázat). Legjobban a gyakoriság-alapú modell teljesített, mert a szövegkörnyezetet is figyelembe vevő trigram modell nem számol a gyakoriságokkal.

DOI:10.15774/PPKE.ITK.2016.003

	szópontosság	**magánhangzó-pontosság**
nincs ékezetesítés	53,0	71,7
első alternatíva	90,2	94,9
szógyakoriság	**94,3**	**97,2**
trigram	92,3	96,0

8. táblázat. Az ékezetesítés többértelműségét kezelő megoldások összehasonlítása

3.4. Az algoritmus alkalmazása egy másik feladatra

Az ékezetesítés algoritmusa - egy másik lexikonfájl használatával - változtatás nélkül alkalmazható egy másik feladatra: egy tájszólás, a zárt *ë* hang írásbeli jelölésére.

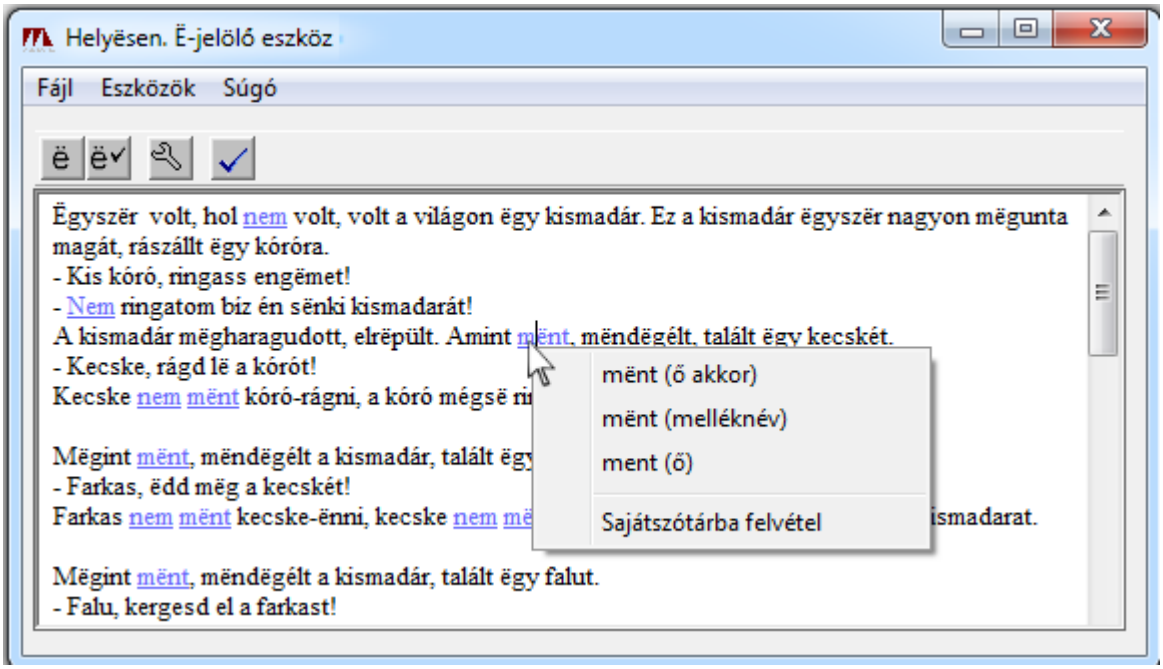
A Bárczi Géza Kiejtési Alapítvány felkérésére készítettünk 2005-ben az ékezetesítőhöz hasonló alkalmazásként egy, a zárt *ë* hang írásbeli jelölését segítő automatikus eszközt (Novák és Endrédy 2005). A zárt *ë* hang a régi magyar nyelvben megtalálható magánhangzó. A helyesírásban nem jelöljük, de dialektálisan él egyes mai nyelvjárásokban. A különböző *e* hangok között a budapestiek 80%-a már nem tud különbséget tenni (Kontra és mtsai. 2011; Hattyár, Kontra, és Vargha 2009). A zárt-ë hang írásbeli jelölését még Kodály Zoltán is szívügyének tekintette, és az alapítvány – a mai kor kívánalmainak megfelelően – egy alkalmazást szeretett volna erre a célra. Az eszközhöz készült lexikonban az *ë* fonémák jelölésével kiegészített felszíni alakok kerültek a lexikai alak helyére, így a tövesítő eszköz ékezetesítő üzemmódban használva éppen a kívánt feladatot végzi el, ahogy az a 9. táblázatban látható. A toldalékmodellt magunk adaptáltuk a feladathoz, a tőtárban az *ë* fonémák jelölését és az elöl képzett nyitótövek azonosítását az alapítvány munkatársai, Buvári Márta és Mészáros András végezték el.

	bemenet	**elemzés**	**kimenet**
elemző	elmentem	el[IK]+megy[IGE]=men+t[MIB]+ek[PL]+[NOM]	elmentem
zárt ë átalakító	elmentem	el[IK]+mën[IGE]=men+t[MIB]+ek[PL]+[NOM]	elmëntem

9. táblázat. Zárt ë átalakító algoritmusa azonos az ékezetesítőével

DOI:10.15774/PPKE.ITK.2016.003

A nyelvi modult egy rich text szövegszerkesztőbe építettem be, ahol az egyes többértelmű szavak aláhúzással jelennek meg, majd jobb egérgomb hatására láthatóak az alternatívák egy pop-up menüben (4. ábra). Az alkalmazásból MorphoLogic-termék is készült.



4. ábra. A zárt-ë átalakító alkalmazás (ë-jelölő), egy többértelmű szónál

DOI:10.15774/PPKE.ITK.2016.003

3.5. Kapcsolódó kutatások

Az egyik legkorábbi cikk (Kornai és Tóth 1997) szótáralapú. Csak a leggyakoribb szavak ékezetesítésével (*es, ket,* stb) 78%-ot értek el, további műveletekkel (rövid szavak kezelése, nagyszótár használata, utófeldolgozás) szópontosságban 4,16%-os hibaarányt tudtak elérni, azaz 95,8%-ot, magánhangzó-pontosságban pedig 98,72%-ot (egy 2000 szavas szövegben), amely eredmények jobbak az általam elértnél (8. táblázat). Megerősítik, hogy a hibák felét csak a kontextus ismeretében lehetne javítani. A cikk érdekessége, hogy felveti azt a megoldási lehetőséget, hogy a „*korszerűbb számítógépek és az ISO-8859-2 kódlap elterjedésével az emberek előbb-utóbb felhagynak az ékezetek nélküli gépelés rossz szokásával”*. (1997.)

Egy másik ékezetesítő megoldás (Zainkó és Németh 2010; Németh és mtsai. 2000; Tarján és mtsai. 2013) TTS alkalmazások számára készült egy komplex szövegfeldolgozó rendszer része volt, morfológiai elemzéssel. A szerzők 95%-os eredményről számoltak be, de az eszköz nem elérhető.

Az egyik legfrissebb magyar vonatkozású eredmény (Novák és Siklósi 2015) alapötlete az, hogy gépi fordítás segítségével ékezetesít: ékezet nélküli szövegről fordít ékezetesre. A webkorpusszal (Kornai és mtsai. 2006) betanított eszköz ismeretlen szavaknál a Humor elemzőt használja a fent leírt, ékezetesítésre módosított lexikonnal, ez a hibrid megoldás rendelkezik az eddigi legjobb eredménnyel (99,23%).

3.6. Összefoglalás

A komment-korpusz szövegeiben sok ékezet nélküli szöveg volt. Ezért szükséges volt egy ékezetesítő eszköz, amely képes az ékezet nélküli szövegeken visszaállítani a ékezeteket (96% szópontossággal). Ezzel sikerült a kommentkorpusz minőségét javítani, és lehetővé tenni az anyag további nyelvészeti feldolgozását. Az elkészült ékezetesítő elve egy másik hasonló feladatra is alkalmazható, így fejlesztettem egy eszközt, amely a magyar zárt ë-t képes jelölni a szövegben.

Kapcsolódó tézis:

2. tézis: Létrehoztam egy ékezetesítő modult, tervezésében egy szerzőtárssal együttműködve, ami egy módosított Humor-lexikonnal 94.3% pontossággal képes ékezet nélküli szövegek ékezetesítésére.

A tézishez kapcsolódó publikáció: [2], [11]

DOI:10.15774/PPKE.ITK.2016.003

4. Szövegek pontosabb feldolgozása: lemmatizáló

A korpusz feldolgozása, illetve általánosan véve az információkereső és lekérdezőrendszerek működése, valamint számos más szövegfeldolgozási feladat megoldása általában feltételez egy olyan algoritmust, amely képes a szövegekben szereplő szavak tövének vagy lehetséges töveinek megállapítására. Különösen érvényes ez a magyarhoz hasonlóan bonyolult morfológiájú nyelvek esetében, hiszen egy-egy lexémának a szövegekben előforduló rengeteg különböző toldalékolt alakja csak így képezhető le egy közös formára, amelynek segítségével a ragozott alakok mind megtalálhatóak. Az adott szóelőfordulás értelmét nem helyesen tükröző tő megadása hibás vagy hiányzó találatokhoz vezet. Így a lemmatizálás minősége befolyásolja az egész rendszer minőségét, annak ellenére, hogy az ilyen rendszerek általában többé-kevésbé kifinomult algoritmusokat alkalmaznak a találatként kapott dokumentumok relevancia szerinti sorba rendezésére, ami a lemmatizálás gyengeségeit részben elfedheti. A lemmatizálás kevésbé kritikus kérdés kevéssé ragozó nyelveknél, például az angolnál. A magyar esetében azonban a sok toldalékolt alak miatt egy gyenge minőségű tövesítő sokat ronthat a keresőrendszer hatékonyságán.

A különböző szótő-megállapító algoritmusokat több szempontból csoportosíthatjuk. Az egyik lényeges szempont az, hogy az adott algoritmus által visszaadott tő vagy tövek mennyire felelnek meg az adott nyelv lexikográfiai hagyományai szerint szótőnek tekinthető alakoknak. Lemmatizálónak nevezzük azokat az eszközöket, amelyek mindig a szótárírói hagyománynak megfelelő reprezentáns alakot, a lemmát adják vissza. A magyar esetében ez névszóknál általában az egyes szám alanyesetű alakot, igéknél a jelen idő kijelentő mód egyes szám harmadik személyű alakot jelenti. Más nyelvek esetében az utóbbi helyett gyakran inkább az infinitívusz használatos. A lemmával szemben ráadásul általában elvárás, hogy létező szóalak legyen. Ezért ha egy adott defektív paradigmájú lexéma paradigmájából éppen a szokásos lemmának megfelelő alak hiányzik (pl. a *megsínyli* ige esetében), akkor másik alakot választanak.

Ezzel szemben stemmernek nevezünk minden olyan eszközt, amely valamilyen tőalakot létrehoz az adott szóalakhoz, bármiféle megszorítás nélkül arra nézve, hogy az adott alak valóban az adott lexéma paradigmájának valamelyik tagja legyen, vagy akár csak létező szóalak legyen az adott nyelven. Az általános, szótárt nem tartalmazó gyors algoritmikus tövesítő algoritmusok a szóalak csonkolásával gyakran ilyen tőalakokat hoznak létre, amely bizonyos feladatokra elegendő is lehet (10. táblázat).

DOI:10.15774/PPKE.ITK.2016.003

	stemmer	**lemmatizáló**
példaszó: *kelyhek*	*kelyh*	*kehely*
rendhagyó szavak kezelése	nem	igen
ismeretlen szavak előfordulhatnak	nem	igen
komplexitás	egyszerű algoritmus: végződések levágása	morfológiai műveletek
erőforrásigény	kevés erőforrás, nincs lexikon	lexikon fájl(ok), több CPU/memória
Melyik a jobb?	?	

10. táblázat. A lemmatizáló és a stemmer összehasonlítása

Most egy általunk készített magyar nyelvű lemmatizálót mutatunk be, amelynek teljesítményét összevetjük más, a magyar nyelvre alkalmazható lemmatizáló és általános tövesítőeszközök teljesítményével. Az elkészített modul számos cég alkalmazásaiba beépült: erre épül a Microsoft Indexing Service, az Országos Atomenergetikai Hivatalban tárolt dokumentumok tárolására és keresésére szolgáló rendszer, az MTI szerkesztőségi rendszere és a PolyMeta kereső. Az MTA Nyelvtudományi Intézete által készített Magyar Nemzeti Szövegtár második bővített kiadásának (MNSZ2) morfológiai annotálása ugyancsak ezzel az eszközzel készült.

4.1. Humor elemzőre épülő lemmatizálás algoritmusa

A kifejlesztett lemmatizáló modul a Humor elemzéseire épül: az elemzésben szereplő morfokból azok címkéje által meghatározott szerepük figyelembevételével építi fel a szó tövét. (A Humor elemzésekről bővebben a 3.1 fejezetben írtam.)

Az algoritmus tervezésében nagy segítségemre volt Novák Attila, aki nem csak Humor elemzéseit és belső működését ismerte, hanem jó érzékkel vette észre a készülő lemmatizáló algoritmus és paramétereinek továbbfejlesztési lehetőségeit. Nagy szerepe van abban, hogy minden ismert esetet tud kezelni a lemmatizáló: Attila segített észrevenni ezeket az eseteket, és hasznos meglátásai voltak, hogyan lehet ezeket megoldani.

A lemmatizáló algoritmus lényege az, hogy a tő építésekor az abban szereplő morfémák felszíni alakját használjuk fel, kivéve az utolsó tőalkotó morfémát: ennek a szótári alakja szerepel a tőben. Természetesen kérdés, hogy mely morfémák számítanak tőalkotónak, és melyek nem. A képzők esetén ez különösen fontos kérdés. Például az *adósság* szó esetén a *-ság* képzőt levágva *adós* lesz a tő. De ha az *-s* illetve az *-ó* képzőt is levágjuk, akkor már *adó* illetve *ad* lesz a szó töve. Látható, hogy meghatározó a végeredmény tő szempontjából, hogy egy adott képzőt tőalkotónak tekintünk-e vagy sem. Általában érdemes figyelembe venni, hogy az adott alkalmazás szempontjából mi lehet a legkedvezőbb megoldás. Minél több képző levágása növelheti a tövesítés fedését (kevesebb potenciálisan releváns találatot veszítünk el),

DOI:10.15774/PPKE.ITK.2016.003

de ronthatja a pontosságát (több nem releváns találat áll elő). Néhány példa látható a 11. táblázatban a Humor elemzéseire és a belőlük előállított tövekre.

input: szó	elemzés	output: a szó töve
várnak	vár[FN] +nak[DAT]	vár[FN]
	vár[IGE]+nak[t3]	vár[IGE]
alma	alma[FN]+[NOM]	alma[FN]
	alom[FN]=alm+a[PSe3]+[NOM]	alom[FN]
mentem	ment[IGE]+em[Te1]	ment[IGE]
	megy[IGE]=men+tem[Me1]	megy[IGE]
	megy[IGE]=men+t[_MIB]+em[PSe1]+[NOM]	ment[MN]
fejetlenség	fejetlenség[FN]+[NOM]	fejetlenség[FN]
	fejetlen[MN]+ség[_PROP]+[NOM]	fejetlenség[FN]
	fej[FN]+etlen[_FFOSZT]+ség[_PROP]+[NOM]	fejetlenség[FN]
	fej[IGE]+etlen[_IFOSZT]+ség[_PROP]+[NOM]	fejetlenség[FN]
adósság	adósság[FN]+[NOM]	adósság[FN]
	adós[FN]+ság[_COL]+[NOM]	adósság[FN]
	adós[MN]+ság[_PROP]+[NOM]	adósság[FN]
	adó[FN]+s[_SKEP]+ság[_PROP]+[NOM]	adósság[FN]
	ad[IGE]+ós[_SZOK]+ság[_PROP]+[NOM]	adósság[FN]

11. táblázat. Humor elemzések és a belőlük kiszámolt tövek, adott képzőbeállítások mellett

Amint a 11. táblázatban látható, egy szónak több is töve lehet, és ritka esetektől (például szóviccektől) eltekintve ezek közül csak egy a valódi töve egy adott környezetben. Azonban a tövesítő modulok bemenete általában csak egy szó (szövegkörnyezet nélkül), és az IR rendszerek architektúrája is szó alapon hívja a tövesítőket. Így a modulnak – ezen a szinten – nincs lehetősége különbséget tenni a tövek között, csupán heurisztikus sorrendbe teheti a tőalternatívákat.

Más nyelveknél is előfordul ez a fenti jelenség, ezért a vezető IR rendszerek képesek arra, hogy egy szóhoz több tövet is letároljanak (Lucene, Elastic Search, MS Indexing Service, MSSQL Full-text search stb.), de már találkoztam olyan keresőrendszerrel, ami csak egyet fogadott el (Tovek Tools[5]).

Azt, hogy a tőmeghatározásnál egy adott a morféma tőalkotónak számít-e, a morféma címkéje dönti el. Az adott morfológiai lexikonban használt címkék halmaza és az, hogy pontosan mely képzőket milyen esetben érdemes tőalkotónak tekinteni, illetve milyen egyéb beállításokra van szükség, a nyelvtől és az adott alkalmazástól is függ. A tövesítés szempontjából releváns címkekészletek leírására ezért a lemmatizáló egy külön konfigurációs

[5] http://en.tovek.cz/products-tovek-tovek-tools

DOI:10.15774/PPKE.ITK.2016.003

fájlt használ. Így a lemmatizáló modul forráskódjába nem kell tenni nyelvspecifikus konfigurációs adatokat, másrészt a korábban bemutatott képzők problémáját ilyen módon hangolni lehet az adott feladathoz. Ez nagy szabadságot és hangolhatóságot biztosít a modulnak.

Létezett már korábban is Humor elemzéseire épülő lemmatizáló: a **HelyesLem** (Prószéky és Tihanyi 1992; Prószéky, Pál, és Tihanyi 1994). A HelyesLem lemmatizálót Tihanyi László (MorphoLogic) implementálta C nyelven. Főként egyszálú használatra készült, és bár az alapvető lemmatizálási feladat megoldására alkalmas, számos bonyolultabb szókonstrukció esetében nem helyes lemmát ad vissza. Az itt bemutatott implementáció C++ alapú, többszálú (multithread safe), és a lemmatizálási algoritmust, illetve a morfémák osztályozását kiegészítettem mindazokkal a finomságokkal, amelyek a HelyesLem által nem helyesen kezelt szerkezetek esetében is helyes eredményt adnak (ikerszavak kezelése; szóösszetételek jelölése; *-ó, -ás* vagy *-s* képző helyes kezelése, *nagybefektető* töve ne *nagybefektet* legyen, egyéb szűrők, minderről bővebben a 4.3. fejezetben). Ez a modul a funkcionalitás szempontjából a HelyesLem utódjának tekinthető, bár a forráskódot tekintve nincs egyetlen közös soruk sem. A modul első változata 2005-ben készült, *stem2005* projektnéven. Később számos finomítás és refaktort végeztem a kódjában. Legkésőbb 2012-ben az ikerszavak kezeléséhez teljesen újra kellett írnom az algoritmus implementációját.

DOI:10.15774/PPKE.ITK.2016.003

4.2. Konfigurációs fájl

A modulhoz definiált konfigurációsfájl-formátum felülről kompatibilis a HelyesLem lemmatizáló hasonló konfigurációs fájljának formátumával. A konfigurációs fájl formátumát a kompatibilitás és a László iránti tisztelet miatt megtartottam. A konfigurációt mindig az adott feladathoz kell hangolni (szótárazás, keresés, stb.) Az egyes szekciók jelentéseit az 4.3. fejezet taglalja, a működésére példa a 4.6. fejezetben található.

Részlet a konfigurációs fájlból:

```
[stem]
DET
ELO
FN
...

[conversion]
_OKEP MN
_COL FN
_DATUM FN
...

[compound must have]
# one of these labels must be present in the analysis for
being a compound word
FN
MN
```

4.3. A lemmatizáló hangolható paraméterei

Bár a lemmatizáló algoritmusának és konfigurációs lehetőségeinek fejlesztésekor a magyar alaktani szerkezetek teljes és helyes lefedése volt az elsődleges célunk, az eszköz általános, és nyelvfüggetlenül használható. A Humor morfológiai elemzőhöz számos nyelvre készült morfológiai leírás, ezek mindegyikére jól használható az itt ismertetett lemmatizáló is, beleértve például azokat az újlatin nyelveket is, amelyeknél – hasonlóan a magyar ikerszavakhoz – a szóalakok belsejében is előfordulnak ragok.

A lemmatizáló konfigurációs fájljában (amelynek formátuma a Windows ini fájljainak formátumára hasonlít) különböző szekciók szolgálnak az egyes morféma-osztályokba tartozó morfémák címkéinek megadására, illetve különböző konverziók és speciális szűrők definiálására. A konfigurációs fájlban található paraméterezhető tulajdonságok egy-egy nyelvi jelenség kezelésére születtek, majd többször finomítottuk a leírásokat és a lemmatizáló algoritmusát is. A jelenlegi változat képes az összes eddig felmerült eset kezelésére.

DOI:10.15774/PPKE.ITK.2016.003

A két legalapvetőbb szekció a tőalkotó morfémák címkéinek *(stem)* és a képzők eredő szófajának *(conversion)* megadására szolgál (pl. hogy az -i képző melléknevet hoz létre). Emellett lehetőség van olyan címkekonverziók megadására is, amely nem jár azzal, hogy képzőnek is tekintse az algoritmus az adott morfémát *(tag replace).* Ez lehetőséget ad többek között a morfológiai elemző által visszaadott címkekészlet egyszerűsítésére vagy egyszerűen a címkék más alkalmazás által várt formára való hozására.

A tő szófaját az utolsó tőalkotó morféma szófaja határozza meg. Ha ez képző, akkor a fent definiált módon a képzett szófaj szerepel az eredményben. Ez alól kivételt jelent például az elliptikus szerkezetekben szereplő szóösszetételi tagok végén álló, a hibás központozásból adódó szóvégi kötőjel vagy gondolatjel/hosszú kötőjel. Ezek esetében az eredő szófaj az írásjel szófaja lenne (mert ezek az elemek tőalkotók, amikor valóban a szó belsejében állnak), azonban szó belseji írásjelként *(internal punctuation)* való megadásukkal ez elkerülhető. Az egyébként is csak a szó szélein megjelenő írásjelek nem okoznak hasonló problémát, ezeket a program egyszerűen levágja.

Eddig az volt a tapasztalat, hogy a felhasználási feladat határozza meg, hogy a képzőket érdemes-e tőalkotónak jelölni. Keresési, indexelési feladatnál érdemes: itt csak a találati pontosságot rontaná, ha a képzőket levágva egy sokkal rövidebb tövet adnánk (pl. *tudatlanságot* szó töve *tud* lenne). De elképzelhető olyan feladat, ahol bizonyos képzők nélküli tövekre van szükség. Például egy kis szótárban való keresésnél jobb eséllyel kapok találatot, ha a ság/ség képzőt eltávolítom: a *nagyszerűség* nem biztos, hogy szerepel a szótárban, a *nagyszerű* már inkább, vagy egy keresésnél ha nincs találat például az *esőzés* szóra, akkor a hasznos lehet az *eső* találatait megmutatni.

A tő kezdetét megelőző címkék alapesetben nem kerülnek be az eredménybe. A prefixumként *(prefix)* megadott morfémák ez alól kivételt képeznek. A magyarban ilyen például a felsőfok jele. A tövet követő címkéket inflexiónak tekinti az algoritmus, és – amennyiben az eszközt használó alkalmazásnak (például egy morfológiai egyértelműsítő programnak) szüksége van erre az információra – a szófajcímkét követően visszaadja.

A lemmatizáló tudja jelezni a szóösszetételi határ(oka)t. Ehhez két konfigurációs szekció beállítása szükséges: az egyik az összetételi tagként szereplő morfémák címkéinek megadására szolgál *(compound member)*, a másik azon morfémák megadására, amelyek egyikének mindenképpen szerepelniük kell egy összetételben *(compound must have).*

Korábban említettük, hogy az alkalmazástól függhet, hogy bizonyos képzőket tőalkotónak érdemes-e tekinteni, vagy sem. Sok esetben érdemes például az igenévképzőket nem tőalkotónak tekinteni, így a *mosó, mosott, mosandó* alakokat, és ezek továbbragozott alakjait a *mos* tőre vezethetjük vissza. (Például ha nagyobb fedéssel szeretnénk keresni, vagy ha egy

DOI:10.15774/PPKE.ITK.2016.003

szótárprogramnál a képző nélküli alakot is kikeressük, így jobb eséllyel lesz rá találat.) Hasonlóképpen a melléknevek esetében a fokozást a legtöbb feladatban érdemes inflexiónak tekinteni, és a közép- vagy felsőfokú alakokat az alapalakra visszavezetni. Azokban az esetekben azonban, amikor egy ilyen elemet olyan másik képző követ, amelyet tőalkotónak akarunk tekinteni (pl. *megnagyobbít),* az egyébként kváziinflexióként kezelt elemet is a tő részének kell tekinteni. Ennek az a módja, hogy az adott morfémát felsoroljuk a képzők között, de nem szerepeltetjük a tőalkotó morfémák között. Ezek feltételesen tőalkotóvá válnak abban az esetben, ha tőalkotó morféma (képző vagy tő) követi őket.

Az *-ó, -ás* vagy *-s* képző esetleges kváziinflexióként való kezelése hibás eredményhez vezetne az olyan szerkezetekben, mint a *kőtörő*, *nagybefektető* vagy *háromemeletes,* hiszen ezek lemmájaként hibásan a *kőtör, nagybefektet, háromemelet* alakok állnának elő. Itt nem követi más tőalkotó elem ezeket a morfémákat, mégsem hagyhatók ki a tőből. Ezt a hibát úgy küszöböltük ki, hogy ezeket a képzőket összetételekben tőalkotónak tekintendő elemekként definiáljuk *(stem if compound),* és az algoritmust is ennek megfelelően módosítottuk.

A tő meghatározásának akár az algoritmusa is paraméterezhető: a *reg* szekcióban megadható, hogy az egyes címkékre illeszkedő morfémáknak a felszíni vagy a szótári alakját számítsa-e bele a tőbe. Például a `((?:(?:FN|NOM|KJ) )+)(FN|KJ) => {surf}\1 {lex}\2` bejegyzés jelentése: a bal oldali reguláris kifejezés ha illeszkedik, akkor a nyíl jobb oldalán megadott módon az első zárójelezett csoportnak a felszíni alakja kerül a tőbe *(surf)*, a második csoport pedig a lexikális alakjával szerepel. A működését egy szón bemutatva: *adatbázis-indító-paranccsal* szó esetén az elemzés: `adatbázis[FN]+[NOM]+-[KJ]+indító[FN]+[NOM]+-[KJ]+parancs[FN]=paranccs+al[INS]`, amire a fenti reguláris kifejezést és a definiált szabályt alkalmazva ez jön ki: *adatbázis-indító-parancs*, mert a FN KJ FN KJ FN sorozat elemeiből az első illeszkedő zárójeles csoport (*group*) esetén a felszíni alak kell ({surf}\1), ez jelen esetben az "*adatbázis-indító-*", a második illeszkedő csoportból pedig a lexikai alak kell, ami a "*parancs*".

A magyarra implementált alapalgoritmus eddig minden általunk kezelt nyelvre (angol, német, francia stb.) alkalmazhatónak bizonyult. A reguláris kifejezéseken alapuló kiegészítés olyan szerkezetek esetében használható, amelyekre a magyarra kifejlesztett algoritmus esetleg nem ad kielégítő eredményt.

Végezetül lehetőséget biztosít a lemmatizáló arra, hogy a morfológiai elemző bizonyos elemzéseit egyszerűen kihagyja a *pattern to delete* szekcióban megadott reguláris kifejezésekre illeszkedő elemzések kiszűrésével. Ily módon lehet megszabadulni az esetleges téves összetételektől (pl. *anyó+som, szak+adás)* vagy az adott alkalmazásban nem kívánatos elemzésektől (pl. légy→van, román→roma). Hogy mi a nem kívánatos, az a felhasználás

DOI:10.15774/PPKE.ITK.2016.003

módjától függ. Például egy országos szerkesztőségi rendszer jelezte azt a lemmatizáló-hibát, hogy kínos, ha a *román* szó miatt a tematizáló szoftverük "kisebbségi" címkét ad egy cikknek (a *roma* tő miatt), vagy *iránt* szóra jön az "Irán", az *államtitkára* esetén a "szigorúan bizalmas" címke, az *államtitok* miatt. Azt kérték, hogy a lemmatizáló ezekre a szavakra soha ne adja ki ezeket a töveket.

Az alábbi minta például a téves *adás/adó* végű összetételek kiszűrésére szolgál:

```
(ár|borz|fog|hal|láz|mar|rag|szak|tag)\[FN\]\+ad(\[IGE|(ó|ás)\[FN)\]
```

Jelentése: ezen feltételezett előtagok után *ad* ige vagy *adó/adás* főnevek nem elfogadható elemzést jelentenek, így nem lesznek benne a kimenetben. Például az *ár+adás, borz+ad, tag+adó* elemzések létrejönnek, a lemmatizáló ezeket kiszűri.

4.3.1. Ikerszavak kezelése

Az ikerszavak *(jövök-megyek, ágával-bogával, okosat-jót)* kezelése különös körültekintést igényel. Névszók és igék is alkothatnak ikerszavakat, és ezekben a két tőnek azonos szófajúnak kell lennie, és azonos toldalékokat kell viselniük (amelyek adott esetben képzők is lehetnek). Maga az elemző nem ellenőrzi a toldaléksorozatok azonosságát, és a szófaji megszorításokat sem teljes körűen. Ezek az ellenőrzések a lemmatizáló szintjén történik. Az ilyen elemzések csak akkor jelennek meg a lemmatizáló kimenetén, ha helyes elemzés nincs, és ilyenkor is hibás szóként jelöli meg őket.

```
>kastélynak-várnak
kastély-vár[FN][DAT]
>kastélyt-várnak
kastély-vár<incorrect word>[FN][DAT]
kastély-vár<incorrect word>[IGE][t3]
```

4.3.2. Cache

A morfológiai elemzés kiszámítása időbe telik. Nagyobb szöveg indexelésénél jelentős gyorsulás érhető el, ha cache-t használunk, azaz ha az elemzéseket és a lemmatizáló által előállított töveket csak egyszer számoljuk ki, később a gyorsítótárból vesszük elő. Így a lemmatizáló/indexelő alkalmazás futásimemória-igényének bizonyos fokú növekedése árán jelentős, akár 8–10-szeres gyorsulást érhetünk el. A memóriaigény/elért gyorsítás arány optimalizálása érdekében a cache-nek két üzemmódja van. Az egyik üzemmód a cache-építési szakasz, ekkor a bemeneti szavakhoz letárolja az eredményül kapott töveket, és kilépéskor ezeket fájlba menti. (Természetesen ilyenkor nemcsak építi a cache-t, hanem használja is.)

Mivel minden szövegkorpuszban a különböző szóalakok nagyobb része csak egyszer fordul elő, sok szöveg tövesítésekor nem érdemes minden szóalakot eltárolni a cache-ben, mert ez

DOI:10.15774/PPKE.ITK.2016.003

komoly memóriaigény-növekedést jelenthet. Ehelyett nagyméretű korpuszból szóalak-gyakorisági listát készítve csak a gyakori szóalakokat érdemes a cache-ben eltárolni. Így tudjuk az alkalmazás sebességét korlátozott memória-többletráfordítással a leghatékonyabban növelni. A cache másik, használati üzemmódjában a lemmatizáló a fájlba mentett cache-t csak olvasható módon nyitja meg, és használat közben nem ad hozzá újabb szavakat.

4.3.3. Kivételszótár és ragozó sajátszótár

Előfordul, hogy egy-egy szó ismeretlen a Humor elemző számára. Folyamatosan új szavak kerülnek a nyelvbe, a tulajdonnevek halmaza sem felsorolható, a legnagyobb körültekintés ellenére is életszerű, hogy előfordul a morfológiai elemző számára ismeretlen szó. (Például jelen sorok írásakor a *drón* ismeretlen szó a lexikonban.) Erre fel kell készülnie a modult használó alkalmazásnak is.

Ha egy szó ismeretlen a morfológiai elemző számára, akkor természetesen a lemmatizáló számára is az. Új szavakat a morfológiai lexikon újrakompilálásával lehet felvenni. Ám ez időigényes és szakértelmet igénylő feladat. Emiatt felmerült az igény, hogy a lemmatizáló modul támogassa az új szavak felvételét. Erre kétféle megoldást implementáltam. Az egyik egy kivételszótár, amiben az ott felsorolt adott input szóalakokhoz megadhatjuk a hozzájuk tartozó töv(ek)et. Ez egyben másik lehetőséget ad a korábban említett reguláriskifejezés-alapú megoldás (4.3. fejezet) mellett a nem kívánt tövek kiszűrésére is. Ha egy szóalakhoz a sajátszótárban csak az adott alkalmazásban elvárt töveket adjuk meg, akkor a rendszer ezektől különböző tövet nem ad vissza. Pl. a *román* szóalak töve nem lesz *roma,* csak *román,* az *iránt* szóé pedig nem lesz *Irán* (nem nagybetűérzékeny tövesítés esetén), ha a kivételszótárban csak ezeket a töveket adjuk meg. Emellett a cache fájl szerkesztése és betöltése is lehetőséget ad az elemzések szűrésére, illetve bővítésére.

A másik megoldás egy ragozó sajátszótár, ahol az új szavak alaktani viselkedését egy a morfológiai elemző által már ismert másik azonos módon ragozott szó megadásával lehet a rendszer tudomására hozni. A ragozó sajátszótár formátuma az alábbi:

<új szó> <ismert, hasonló végződésű és ragozású szó> <kívánt szófaj, opcionális>

Például:

ódalog andalog
ódalg andalg
Intel lepel FN
vmi valami
fészbuk mameluk

DOI:10.15774/PPKE.ITK.2016.003

Elemzés előtt a modul az ismeretlen szó helyére az itt kiválasztott ismert szót helyettesíti be (összetételekben is), így hívja meg a morfológiai elemzőt, majd visszahelyettesíti az eredeti szót az elemzésbe. Így a lemmatizáló algoritmus már jó elemzést kap, amiből ki tudja számolni a tövet. Ha egy összetett szó első tagját nem ismeri a lemmatizáló, akkor elegendő azt felvenni, ettől az összetételekben is javulni fog. Például ha ismeretlen szó a *google-keresés*, akkor elegendő a *google*-t felvenni.

4.4. A lemmatizáló kimenetének beállításai

A lemmatizáló kimenetének részletessége sokféleképpen beállítható. Állítható, hogy csak a tövet adja-e vissza vagy a szófajt is. Jelezze-e a szóösszetételi határokat, visszaadja-e az összes kategóriacímkét, illetve az eredeti elemzést is. A következő beállítások bármelyike egymástól függetlenül bekapcsolható:

- a kimenet csak a töveket tartalmazza (*"alma,alom"*)
- a kimenetben a tő+szófaj szerepel (*"alma[FN],alom[FN]"*),
- a szóösszetételi határokat jelzi a kimeneten (*"ablak+kilincs"*),
- minden morfológiai kategóriát tartalmaz a kimenet, (*képviselőházban* → *képviselőház[FN][INE]*)
- a kimenet tartalmazza az eredeti Humor elemzéseket is
- a bemenet kis/nagybetű állapotát másolja a kimenetre is

DOI:10.15774/PPKE.ITK.2016.003

4.5. A lemmatizáló szűrői

A lemmatizáló bizonyos szavakra több tövet is visszaadhat. Az elemzésekben akár ugyanaz a tő is ismétlődhet (11. táblázat), azonban általában nincs szükség arra, hogy azonos eredmények ismétlődjenek a kimeneten. Emellett egyéb szűrésekre is szükség lehet az adott alkalmazás igényeinek megfelelően. A következő szűrők ki tudták szolgálni az eddig felmerült igényeket:

- ugyanaz a tő csak egyszer szerepeljen (akár homonima, akár más okból többször is szerepelne egy tő a kimenetben, akkor abból csak egyet engedjen)
- ugyanaz a tő+szófaj csak egyszer szerepeljen
- az elemző által produktívan előállított összetett szavakat nem adja vissza, ha van más tő is (pl. a *szer+elem* tövet kihagyja, mert a *szerelem* tövet egyben megtalálta)
- képzők levágásával nyert töveket ne adja vissza, ha talált a lexikonban más tövet (az *adós* töve nem lesz *ad*, ha az *adós* egyben is szerepelt a lexikonban)
- azon töveket ne adja vissza, amelyek az inputtal egyeznek. Erre tipikusan keresési/indexelési feladatnál lehet szükség, ha az indexelő az eredeti szóalakot eleve automatikusan felveszi az indexbe.

4.6. Futási példák a lemmatizálás működésére

A fenti beállításokat, paramétereket, konfigurációs fájl használatát és az algoritmus lépéseit (4.2.-4.5.fejezetek) néhány konkrét szó lemmatizálásán mutatom be. Az elemző az *igazságot* szóra ezeket az elemzéseket adja:

```
igazság[FN]+ot[ACC]
igaz[MN]+ság[_PROP]+ot[ACC]
```

A lemmatizáló balról jobbra végigolvassa az elemzéseket, és az első esetében talál egy `FN` morfémát, mivel ez tőalkotó a konfigurációban (a *stem* szekcióban szerepel az `FN`), így az `igazság` bekerül a lemmába, szófaja `FN`. Az *ot* morféma címkéje `ACC`, ez nem tőalkotó, hanem inflexió, ez kimarad a tőből. Így az első elemzésből az `igazság[FN]` jön ki. A második esetben az `MN` tőalkotó, az *igaz* bekerül a lemmába. A *ság* morféma címkéje `_PROP` (általában az _ jellel kezdődő címkék képzők), ez szerepel a konfiguráció *stem* szekciójában: ezért ez *ság* morféma bekerül a lemmába. A `_PROP` címke szerepel a conversion szekcióban is (`_PROP FN`) ezért a lemma szófaja `_PROP` helyett `FN` lesz. A következő morféma - ahogy az

DOI:10.15774/PPKE.ITK.2016.003

első esetben is - inflexió, így kimarad a tőből. A második elemzés végeredménye: `igazság[FN]` lesz. Ha be van kapcsolva a lemmatizáló szűrője (például lemma+szófaj szűrő, bővebben az 4.5. fejezetben), akkor ez a két egyforma lemma helyett (`igazság[FN]` és `igazság[FN]`) csak egy fog megjelenni a kimeneten.

Másik futási példa a *nagybefektetője* bemeneti szóra kapott elemzések feldolgozása:

```
nagy[ELO]+befektető[FN]+je[PSe3]+[NOM]
nagy[ELO]+be[IK]+fekszik[IGE]=fek+tet[_MUV]+ő[_OKEP]+je[PSe3]+[NOM]
```

Az első elemzés hasonló az előzőkre: az `ELO` és `FN` morfémák tőalkotók (*stem* szekcióban szerepelnek), a többi nem, így a kimenet a `nagybefektető[FN]` lesz. Itt szerencsés eset áll fenn: a *befektető* szó egyben is megvan a lexikonban, nem csak produktívan áll elő.

A második elemzés új elemeket tartalmaz: az `ELO`, `IK`, `IGE` és `_MUV` címkék tőalkotók. Új elem, hogy az ige morfémája két alakot is tartalmaz: `fekszik[IGE]=fek` (a *fekszik* lexikai alak ill *fek* felszíni alak). Lemmaépítésnél minden tőalkotónak a felszíni alakját tesszük a lemmába, kivéve az utolsóét: annak a lexikai alakját. Így ezen a ponton (`_MUV` morfémánál) a lemma tartalma: `nagybefektet`, szófaja `IGE` (mert a `_MUV` igére képzi a szófajt a konfiguráció szerint). A szó már összetett szónak számít, mert az eddig morfémák szerepelnek a *compound member* és *compound must have* szekciókban. Ezért figyelembe kell venni a *stem if compound* szekciót is, ahol az `_OKEP` szerepel, aminek az a jelentése, hogy összetett szó esetén ezen címkéjű morfémák tőalkotónak számítanak. Ezért az `_OKEP` morféma ebben az esetben tőalkotó lesz. A szófajt az `_OKEP` igéről melléknévre módosítja (*conversion* szekció). Az ezutáni morfémák azonban nem, így az elemzés végső kimenete: `nagybefektető` lemma és `MN` szófaj. Összegezve a két elemzést: a két kiszámolt tő a `nagybefektető[FN]` és a `nagybefektető[MN]` lesz.

4.7. Kapcsolódó kutatások

Az egyik leghíresebb tövesítő algoritmus a Porter Stemmer (Porter 1980). Ezen algoritmus lényege, hogy a bemeneti szó végéről gyakori végződéseket (suffixeket) levág, a szó magánhangzóinak-, mássalhangzóinak száma, illetve végződése alapján. A Snowball modul kapcsolódik ehhez: a Snowball egy olyan programozási nyelv, amelyben Porter tövesítő algoritmust lehet írni tetszőleges nyelvre.

Előnye, hogy algoritmikus, azaz minden szóra tud mondani tövet, állítható benne a levágandó toldalékminták mennyisége (hangolható a tövesítés mértéke). Hátránya, hogy attól,

DOI:10.15774/PPKE.ITK.2016.003

hogy egy szó vége toldaléknak látszik, nem biztos, hogy az is. Sok esetben valóban közelebb kerül a tőhöz, de gyakori, hogy hibás szavakat ad tőnek (*Péterék → Pétere*, *banán → ban* stb.)

Magyar nyelv esetén rendszer nagyjából 150 szabállyal, 9 lépésben eljut a szó tövéig, rekurzió nélkül. A 9 lépés rövid leírása a következő:

1. a *val/vel* rag eltávolítása: *-al -el* eltávolítása, ha kettős mássalhangzó előzi meg. A kettős mássalhangzóból csak egyet töröljünk.
2. keressük meg és töröljük ezek közül a leghosszabb toldalékot: *ban ben ba be ra re nak nek onként enként en on stb.* A terminális elemet javítsuk á-ról a-ra illetve é-ről e-re.
3. az előző lépés ezekre: *án ánként én*.
4. az előző lépés ezekre: *astul estül stul stül*
5. az előző lépés ezekre: *án ánként én*
6. az előző lépés ezekre: *oké öké aké eké ké éi é áké áéi stb.*
7. az előző lépés ezekre: *ünk unk nk juk jük uk ük em om am m od ed ad öd d ja je a e o ánk ájuk ám ád á stb.*
8. az előző lépés ezekre: *jaim jeim aim eim im jaid jeid aid eid id jai jei stb.*
9. az előző lépés ezekre: *ák ék ök ok ek*

Mivel az ilyen módon eredményül kapott tő gyakran nem egyezik a morfológiai tővel, ezért a tövesítő minőségét szokták úgy is mérni, hogy egy keresőrendszer minőségét mennyiben tudja javítani. Magyar nyelvre a Snowball stemmert négyféle suffix vágási súllyal értékelték ki (Tordai és De Rijke 2006), így akár 20%-os javulást is tudtak elérni. A Porter és Snowball tövesítők kiértékelését, ennek módszereit a 5. fejezetben mutatom be.

DOI:10.15774/PPKE.ITK.2016.003

4.8. Összefoglalás

A korpuszok pontosabb feldolgozásához szükség volt egy jobb, a magyar nyelv sajátosságait jól kiszolgáló lemmatizálóra. Fejlesztettem egy lemmatizáló modult, ami a Humor morfológiai elemzőre épül, és a bemeneti szavakra adott elemzésekből számolja ki a lemmá(ka)t. A működése egy konfigurációs fájlon keresztül hangolható, az adott feladat és nyelv igényei alapján. Az elkészült lemmatizáló kiértékeléséhez többféle metrikát definiáltam, a részletes kiértékelés a *Tövesítők kiértékelése* fejezet 5.5. és 5.6. alfejezeteiben található. A lemmatizálóm a mérések alapján jobb eredményeket produkált, mint a pillanatnyilag elérhető más megoldások, és számos cég alkalmazásaiban is felhasználásra került (Microsoft Indexing Service, Országos Atomenergetikai Hivatal, MTI, PolyMeta kereső).

Kapcsolódó tézis:

3. tézis: Létrehoztam egy lemmatizáló motort, tervezésében egy szerzőtárssal együttműködve, ami a Humor elemzéseiből kiszámolja a szó tövét, és ez a kiértékelések alapján a legjobb eredményeket adja a magyar nyelvre.

Kapcsolódó publikációk: [2], [4]

DOI:10.15774/PPKE.ITK.2016.003

5. Tövesítők kiértékelése

A tövesítőket - akár lemmatizáló, akár stemmer - az 5.1.1. fejezetben bemutatott kiértékelések a közvetlen kimeneten mérték. Ez hátrányos a stemmerekre nézve, hiszen náluk egyáltalán nem elvárás, hogy lemmákat (pontos nyelvtani töveket) állítsanak elő. Lehetséges, hogy egy stemmer jobban teljesít például egy keresési feladatnál, vagy egy adott nyelvre felesleges lemmatizálót írni (elegendő egy stemmer is), de ezt nem tudjuk kideríteni, ha a stemmereket lemmapontossággal mérjük. Ezért készítettem egy tövesítő-kiértékelő rendszert, ami nem csak a korábban bemutatott metrikákat, hanem IR alapú kiértékeléseket is végez.

Egy adott alkalmazásnál az a kérdés szokott felmerülni, hogy az elérhető tövesítők közül itt most melyiket érdemes választani. Másrészről, az előbbi fejezetben bemutatott lemmatizálót is könnyebb kiértékelni, ha a szélesebb mezőnyben mérjük meg a képességeit. Ezért létrehoztam egy kiértékelési módszert, amivel tetszőleges tövesítő kiértékelhető. Az elkészült eszköz nyílt forráskódú[6].

A módszer alapötlete, hogy bármely lemmával annotált korpuszból automatikusan készíthető keresőrendszer (*information retrieval*, továbbiakban IR) számára használható kiértékelő teszthalmaz. Majd ezen kétféle kiértékelést végzünk: a tövesítő közvetlen kimenetét, illetve IR rendszerbeli hasznosságát mérjük meg. Az elkészült kiértékelés naprakész áttekintést ad 10 tövesítőről 3 különböző ragozási bonyolultságú nyelven (angol, lengyel és magyar). A létrehozott eszköz további tövesítőkre és más nyelvekre is használható.

Háromféle kiértékelést végeztem:

- közvetlen kimeneten
- IR alapú
- egy kísérleti IR nélküli, ami korrelál az IR alapúval

5.1. A tövesítő közvetlen kimenetén mért kiértékelések

A tövesítőknél alapvetően kétféle kiértékelés létezik: az egyik a tövesítők közvetlen kimenetét méri, a másik közvetetten a hatásukat, például egy IR rendszer hatékonyságát méri. Az előbbi módszert ebben a fejezetben, az utóbbit a következőben mutatom be.

A tövesítők kimenetét 5.1.1. fejezetben definiált 5 metrika szerint értékeltem ki. Ezen felül a Paice-metrika (Paice 1994) is rendelkezésre áll, az understemming index (UI), overstemming index (OI), és az ezek alapján definiált hibaarányt (ERRT) is kiszámítja az

[6] `https://github.com/endredy/stemmerEval`

DOI:10.15774/PPKE.ITK.2016.003

NLTK segítségével (Bird 2006). Ezen mérőszámok bővebb bemutatása a 5.7 fejezetben található. Most csak utalni szeretnék arra, hogy ezen indexek a szavak elvárt és kiszámolt töveire épülve számolja ki a tövesítő ideálistól való eltérését.

Összegezve, 6 metrikát alkalmaztam a tövesítők közvetlen kimenetére, amely méri a lemmapontosságot, a lemmasorrendet, a hibás lemmákat és a Paice-metrikát is. Kétségtelen, hogy a stemmerek hátrányban vannak ezekben a mérésekben, mert ők eleve nem lemmákat adnak vissza. Ezzel együtt a fenti metrikák adhatnak egy képet az adott stemmerről és nyelvről.

5.1.1. Tövesítő kiértékelési metrikák

Ebben a fejezetben bemutatom azokat a metrikákat, amelyeket a tövesítők kimenetének kiértékeléséhez használtam. A későbbi fejezetekben ezeket használom: a 5.5. fejezetben részletesen a magyar tövesítő megoldások kiértékeléséhez, a 5.6. fejezetben pedig az angol és lengyel nyelveken végzett kiértékelés kapcsán. (Humor elemző használható sok idegen nyelvre is.)

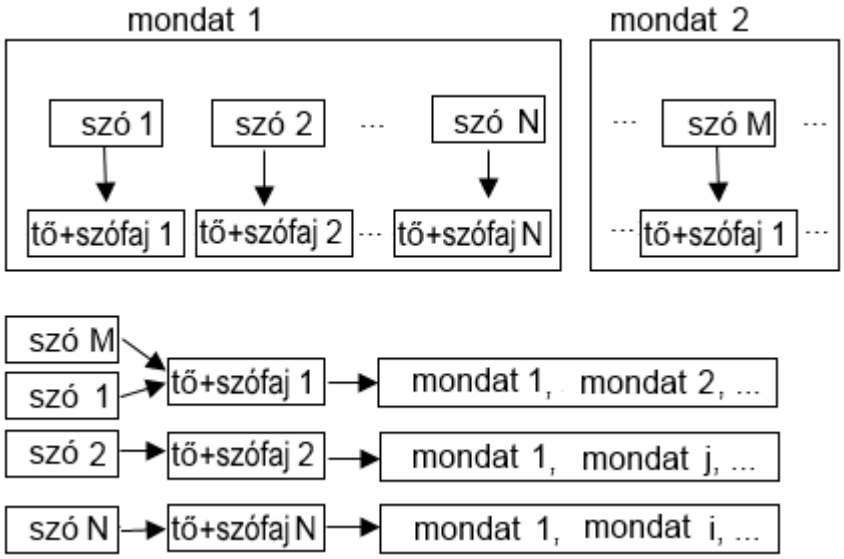
A tövesítők kiértékeléséhez a közvetlen kimenetükre 5 féle kiértékelési metrikát definiáltam. Elsőként tőalternatívák első elemének a lemmapontosságát (*accuracy*) mértem meg. A következő három metrika azt pontozza, hogy az egyes tőalternatívák önmagukban mennyire jók, illetve ezek sorrendezése mennyire helyes. (3.3 fejezetben található bővebb leírás a problémáról.) Az egyik metrika méri a tövek sorrendjét (*ranking*): a helyes tő előtti hibás tövek +1 FP (*false positive*) találatot jelentenek, ha az *n.* tő a helyes, akkor *n-1* FP és 1 TP hibapontot kap. Ha hiányzik a helyes tő, akkor +1 FN. Ezen metrika neve *average precision at maximum recall* (Lindén 2009; Novák 2015). A másik metrika azt feltételezi, hogy minden egyes tövet letárolunk, így az szigorúbb: minden hibás alternatíva +1 FP, helyes tő +1TP, hiányzó helyes tő +1FN. A harmadik, tőalternatívákkal foglalkozó metrika az első tőre koncentrál: ha ez helyes TP, ha helytelen FN, ha az alternatívák között szerepel a helyes akkor +1FP.

A következő metrika az összes, korpuszban előforduló helyes tövet figyelembe veszi. Így először minden egy szóhoz megkeressük a korpuszbeli összes helyes tövét (a gold standard szerint), majd ezeket a halmazokat hasonlítjuk össze a tövesítő kimenetével. A metszetük TP, csak a korpuszban szereplő lemmák FN, csak a tövesítőben szereplő lemmák pedig FP találatot jelentenek.

DOI:10.15774/PPKE.ITK.2016.003

5.2. Tövesítők IR-alapú kiértékelése

Megközelítésem arra az ötletre épül, hogy minden, tövekkel annotált korpusz átalakítható IR kiértékelő gold standarddé. A módszer alapja, hogy a korpusz mondatai (egy vagy több mondata) lesznek a IR találati egységek (dokumentumok), az egyes mondatok szavai pedig az ezekhez a dokumentumokhoz (mondatokhoz) tartozó lekérdezések.



5. ábra. Tövesítők IR-alapú kiértékelése tővel annotált korpusszal: mondatok=dokumentumok és szavak=lekérdezések, az eredmény mondathalmazok a gold standarddal kiértékelhetők

Az IR gold standard úgy generálható ki egy tövekkel annotált korpuszból, hogy a benne szereplő összes szóhoz tartozó (*mondatazonosító, eredeti szóalak, szótő, szófaj*) sorozat alapján elkészítjük azt a listát, ami tartalmazza az egyes szóalakok tő+szófaj értékeit, és azt, hogy ezek milyen mondatokban fordulnak elő. Más szavakkal, kilistázzuk a korpuszból a literális szóalakokat és a hozzájuk (tő és szófaj információn keresztül) kapcsolódó mondathalmazokat (lásd 5. ábra). Így bármelyik szóalakhoz meg tudjuk mondani, hogy mely mondatokban fordul elő.

Ezek a szó–mondat összerendelések jelentik a gold standardet, amihez képest a tövesítők kiértékelhetők. A pontosság és fedés az alapján számítható, ha összevetjük egy adott tövesítő egyes szavakra adott találatait és a gold standardet.

Itt kiemelem annak a részletnek a fontosságát, hogy a gold standard a szófaj információ figyelemebevételével is készül. Ez a kapcsolódó cikkemben (Endrédy 2015a) még nem így szerepel, ott a tő önmagában kapcsolja össze a szóalakot és a mondatokat. Ez fontos részlet, ami tisztítja a szóalak-mondat összerendeléseket (12. táblázat).

DOI:10.15774/PPKE.ITK.2016.003

Szóalak	Hozzárendelt mondatok	
	tő alapján	tő + szófaj alapján
várunk	"igazi nagy vár falakkal" "várjatok mondta csendesen" "az várja a bolond uralkodót"	"várjatok mondta csendesen" "az várja a bolond uralkodót"
zsarolók	"zsaroló stratégiájának célja" "meg tudta fizetni a zsarolók váltságdíját"	"meg tudta fizetni a zsarolók váltságdíját"
rokonuk	"rokon sorsú családok" "rokonaik látogatnak néha "	"rokonaik látogatnak néha "

12. táblázat. Néhány példa a szóalak-mondat összerendelésekre csupán tő- illetve tő+szófaj alapján

A tő + szófaj alapján gyártott gold standard alapján is elvégeztem az IR kiértékeléseket, ezért a cikk és a disszertáció ebben érintett táblázatai különböznek (mindegyik tövesítő modul eredményére hatással volt, mindhárom nyelven).

Mivel angol nyelv esetén gyakori, hogy egy szóalak több szófajjal is elemezhető (például sok ige főnévként is használható), a kiértékelő program paraméterezhető, hogy a szófajt is figyelembe vegye-e a gold standard készítésekor. Ugyanis ez a szófaji többértelműség minden tövesítőt nagyon kedvezőtlenül értékel. Egy olyan egy nyelvben, mint az angol, ez a jelenség gyakori, ráadásul a kevés ragozott alak miatt még többször egybeesik az eltérő szófajú alak (például *dogs, sinks, parks* főnév és ige is lehet), ezért úgy láttam jónak, ha paraméterként a szófaj információ kikapcsolható. Az angol szófajok és szóalakok egybeeséséről, ezek kiértékelésre gyakorolt hatásáról a 5.6.2. fejezetben lesz szó bővebben.

A kiértékelő eszköz tartalmaz egy Java-alapú Lucene keresőmotor segítségével létrehozott kiértékelőt. Minden egyes tövesítővel a korpusz mondatait (külön dokumentumként) leindexeli, majd a korpusz szavaival (egyedi, eredeti alakban) lekérdezi. A kapott eredményhalmazokat összeveti a gold standarddel. Ez egy tesztgyűjtemény-alapú kiértékelés (*test collection*), amit az IR tesztekben használnak. De azokban a dokumentumokat és a lekérdezéseket kézzel állítják össze (Hull 1996), emiatt pár ezer lekérdezésből állnak. Jelen esetben a lekérdezések nagyságrendje akár milliós is lehet (British National Corpus esetén 2 millió), másrészről ezek a lekérdezések automatikusan generálhatóak korpuszból.

A Java-kód két kiértékelést is elvégez: egyrészt kiértékeli az összes találatot, másrészt külön kiértékeli az első *n* találatot is. Ez utóbbi lehetőség azt a célt szolgálja, hogy szimulálja az emberi kiértékelést: kereséskor az első találatoknak nagyobb a súlya, ritkán szoktuk a sokadik találati oldalt végignézni. Így az ott lévő esetleges hibás találatok nem annyira zavaróak. A tesztek során derült ki, hogy – bár nem ezzel a céllal készült – az első *n* találat kiértékelése arra is alkalmas, hogy magának az IR-rendszernek a sorrendező algoritmusát (*ranking*) kiértékelje. Ha nem ebben a találati tartományban vannak a TP találatok, akkor nem jó a *ranking*. Az *n* érték paramétere a kiértékelésnek, méréseimben a 300 illetve 1000 értékeket használtam.

DOI:10.15774/PPKE.ITK.2016.003

5.3. Natív IR nélküli kiértékelés

Kiértékeltem a tövesítőket egy másik megközelítéssel is, ami az IR kiértékeléshez hasonló, de natív IR nélkül működik. Ennek az a lényege, hogy nem közvetlenül a tövesítő kimenetét méri, hanem a hatását, viszont nem kell hozzá igazi IR rendszer. Más szavakkal: ez a kiértékelés elrejti az egyes tövesítők (stemmerek vagy lemmatizálók) kimeneteit, nem kell szabályos tövet adniuk, hanem elegendő, ha az egyes ragozott alakokat közös tőre vezetik vissza.

A kiértékeléshez használt korpusz minden szavához - a tövesítés után - tartozik egy-egy mondat-halmaz, amit az egyes tövesítők számítanak ki. A gold standard szó-mondat összerendelések pedig a kézi lemma-annotációból származnak (5.2. fejezet). A következő módszer lehetővé teszi, hogy kiértékeljük a tövesítő hasznosságát anélkül, hogy akár lemmapontosság-, akár IR kiértékelést végeznénk. A vizsgált tövesítők végigmennek a korpusz szavain, és eltároljuk az eredeti szót, a töveit és a mondat azonosítót (SID). A stopwordöket kihagyjuk. A folyamat végén az egyes szavak SID halmazait összevetjük a gold standard szavainak SID halmazával. Az egyezés TP, az eltérés FP találatot jelent, FN pedig az, ha egy SID létezik a gold standardben, de hiányzik a tövesítőnél.

Ennek a rendszernek a F-mértéke korrelált a valódi Lucene alapú IR kiértékelés eredményeivel (5.2. fejezet) abban az esetben, ha az összes találatot kiértékeltem, nem csak az első *n* találatot. Ezt a összefüggést a Pearson korrelációval ellenőriztem (Benesty és mtsai. 2009).

5.4. A kiértékeléshez használt korpuszok

A kiértékelést három olyan nyelven végeztem el, amelyeknek különböző szintű ragozási morfológiával rendelkeznek: angol, lengyel és magyar. Először a **British National Corpus** (BNC) anyagát használtam (Clear 1993), amely 100 millió szavas lemmával is annotált korpusz. Több mint 2 000 000 mondatot tartalmaz gondosan válogatott doménekből (újságcikk, szépirodalom, rádióműsorok, párbeszédek, egyéb írott és beszélt források). A kiértékelést a 3. XML kiadás alapján végeztem.

A **Polish National Corpus** 1 millió szavas alkorpusza (Degórski és Przepiórkowski 2012) volt a lengyel nyelvű mérés alapja. Ez a korpusz 64.000 mondatból áll, és szintén kiegyensúlyozott tartalmakkal rendelkezik többféle doménből (irodalom, napi újságcikkek, folyóiratok, párbeszédek leiratai, különböző forrásokból származó rövid szövegek).

A **Szeged TreeBank** (Csendes és mtsai. 2005) volt a magyar kiértékelés alapja, amely a legnagyobb, kézzel annotált korpusz 80.000 mondattal, 1.200.000 szóval. A tövek is jelölve

DOI:10.15774/PPKE.ITK.2016.003

vannak, a lehetséges tőalternatívákkal együtt, és ez a korpusz is több doménnel rendelkezik (újságcikk, szépirodalom, tanulók fogalmazásai, jogi, üzleti és informatikai szövegek).

5.5. Kiértékelés magyar nyelvre

A 4. fejezetben bemutatott lemmatizáló teljesítményét néhány szabadon hozzáférhető tövesítőével összevetve a legnagyobb elérhető tövekkel is annotált kézzel ellenőrzött korpuszon, a Szeged Korpusz 2.0-s változatán (Csendes és mtsai. 2005) értékeltük ki. Ebben a korpuszban nagyjából 80 000 mondat szerepel, többféle forrásból (szépirodalom, 14-16 éves tanulók fogalmazásai, újságcikkek többféle napi-, illetve hetilapból, számítástechnikai, jogi és üzleti szövegek). A korpuszban jelölve van minden szó adott kontextusban érvényes töve (lemmája), illetve az egyéb lehetséges lemmák. Sajnos nem minden tő helyes a korpuszban (illetve mint később visszatérünk rá (5.5.2. fejezet), a korpuszban a lemmatizálás elvei több ponton különböznek attól, ahogy az alább kiértékelt eszközök működnek), de mivel ez a legnagyobb elérhető annotált korpusz magyar nyelvre, és számos nyelvtechnológiai mérés használja, ezt választottuk a mérés alapjául. A kiértékelést számos különböző metrika szerint elvégeztük. Megmértük azoknak a szavaknak az arányát, amelyre egy-egy tövesítő nem adott eredményt (ismeretlen (OOV=out-of-vocabulary) szavak), illetve az egyes tövesítő modulok sebességét.

A különböző lemmatizáló modulok különböző morfológiai címkekészleteket használnak (KR, Humor kód stb.), ezek mindegyike különbözik a Szeged Korpuszban használt MSD kódoktól is, ezért a kiértékelésnél csak a tő helyességét vizsgáltuk, a szófaj- és egyéb morfológiai címkékét nem.

5.5.1. Vizsgált tövesítő modulok

Az egyik szabadon hozzáférhető tövesítő a **Hunspell**, amelyet széles körben használnak, főként open source projektekben. (Jelen sorok írásakor több mint 30 alkalmazás használja, köztük a LibreOffice, OpenOffice, Firefox, Thunderbird, Google Chrome.) Alapvetően helyesírás-ellenőrzésre használják, de tövesíteni is tud. Számos nyelvre készült hozzá lexikon.

A következő, részben a Hunspell és az ahhoz készített magyar morfológiai leírás fejlesztésekor szerzett tapasztalatok felhasználásával készült eszköz az OCaml nyelven implementált **Hunmorph (Ocamorph)** (Trón és mtsai. 2005), amely a morphdb.hu adatbázisra épül (Trón és mtsai. 2006). A lexikonfájlok a Hunspellhez hasonló formátumúak (aff/dic), azonban nem teljesen kompatibilisek vele. A Hunmorph nemcsak tövesíteni tud, hanem teljes morfológiai elemzést ad. Az Ocamorph elemzőn alapul az **Ocastem** lemmatizálómodul, amely kifejezetten információ-visszakereső alkalmazások igényeinek kiszolgálására készült. Az Ocastem alkalmazás egyik legnagyobb előnye, hogy pusztán a

DOI:10.15774/PPKE.ITK.2016.003

lehetséges toldalékok levágásával olyan szavak lemmatizálására is képes, amelyek a szótáraul szolgáló morphdb.hu adatbázisban nem szerepelnek, így minden szóra ad vissza eredményt. Az Ocastem alapbeállítása az, hogy a produktívan összetett szavakat tagjaira bontja, és ezeket külön tőként adja vissza. A Szeged Korpuszban szereplő lemmák nem így vannak megadva, és ez a tövesítő mért eredményeit hátrányosan befolyásolná, ezért az Ocastemet olyan beállítással futtattuk, amely ezt az összetettszó-daraboló működést kikapcsolja. A számos egyéb beállítási lehetőséget is kipróbálva alább a kiértékelésnél a legjobb eredményeket adó fent említett, mindig pontosan egy tövet visszaadó beállítást használtuk.

A magyar **Snowball** tövesítőnek (Tordai és De Rijke 2006) az NLTK fejlesztői csomagban (Bird 2006) szereplő változatát használtuk. Mivel ez a végződéseket egy szótárt nem használó algoritmus alapján vágja le az angolra készült Porter Stemmer (Porter 1980) mintájára, sokszor nem igazi szótövet ad, hanem egy szócsonkot. Előnye viszont, hogy ebből fakadóan az Ocastemhez hasonlóan számára nincs ismeretlen szó, mint az a 13. táblázatban látható.

A **Hunmorph-foma** egy véges állapotú fordítóautomatákon alapuló eszköz, amely a Foma morfológiai elemzőhöz (Hulden 2009) készített, a Hunmorph adatbázisából konvertált magyar morfológiai leíráson alapul. A véges állapotú elemzőimplementáció rendkívül gyors: bár az automata bejárása során a többértelműségek miatt nem elkerülhető, hogy az elemző visszalépjen, és újabb bejárási útvonalakat is kipróbáljon, tehát a bejárás nem determinisztikus, mégis ez a tövesítő a leggyorsabb.

5.5.2. Eredmények

A magyar tövesítőmodulok eredményeit a 5.1.1. fejezetben bemutatott metrikák alapján kiértékeltük. (Az angol és lengyel modulok kiértékelése a 5.6. fejezetben található.) A valódi lemmatizáló modulok által adott elemzések esetében, amelyek a tő mellett morfoszintaktikai annotációt is tartalmaznak, elvileg lehetőség lett volna a szó környezetét figyelembe vevő statisztikai egyértelműsítő (pl. (Halácsy és mtsai. 2006; Orosz és Novák 2013)) alkalmazására is az adott kontextusban helyes tő kiválasztására. Ettől a méréstől itt eltekintettünk, egyrészt mert szükség lett volna hozzá a Szeged Korpuszban használt MSD kódrendszer és az összes többi eszköz egyedi címkézési rendszere közötti konverzióra, másrészt mert az információ-visszakereső rendszerekben ilyen egyértelműsítő eszközt általában nem használnak. Ehelyett az indexelőrendszerrel és magukkal a tövesítőeszközökkel kapcsolatos különböző feltételezésekkel élve az alábbi méréseket végeztük el.

Hogy az egyes modulok a korpuszban szereplő szóalakok mekkora részére nem adnak vissza tövet, vagyis mekkora az ismeretlen (OOV) szavak aránya, a 13. táblázat tartalmazza.

DOI:10.15774/PPKE.ITK.2016.003

alkorpusz	méret	***Hunspell***	***Hunmorph -foma***	***Hunmorph compound***	***Hunmorph***	***Ocastem***	***Snowball***	***Humor***
szépirodalom	185 436	3,5	2,2	**1,1**	2,4	**0**	**0**	1,5
tanulók	278 497	1,4	1,0	0,7	1,4	**0**	**0**	**0,4**
újságcikk	182 172	4,8	3,8	2,6	5,5	**0**	**0**	**1,1**
IT	175 991	8,2	5,5	4,1	7,9	**0**	**0**	**2,7**
jogi	220 069	7,0	6,8	5,7	7,5	**0**	**0**	**2,0**
üzleti	186 030	8,0	7,8	5,7	9,0	**0**	**0**	**1,4**
összesen	1 228 195	5,4	5,2	3,7	5,7	**0**	**0**	**1,5**

13. táblázat. A tövesítő modulok számára ismeretlen szavak aránya a Szeged Korpuszon

Az algoritmikus Snowball és az ismeretlenszó-elemzést alkalmazó Ocastem minden szóra ad vissza tövet. A szigorúan szótáralapú eszközök közül az itt ismertetett Humor-alapú lemmatizáló lexikona adta a legjobb fedést a korpuszon (egy részkorpusz kivételével). A Hunmorph elemző a produktív szóösszetétel bekapcsolásával (Hunmorph compound oszlop) közelítette meg ezt leginkább. Bár a Hunspell helyesírás-ellenőrzőként képes produktív összetételek létrehozására, a tövesítetlenül maradt szavak átnézésekor azt tapasztaltuk, hogy valamilyen implementációs hibából kifolyólag ez a tövesítő üzemmódban nem működik jól. Ennél a kiértékelésnél azt is észrevettük, hogy a Hunmorph-foma elemző az igekötős igékre, összetett számnevekre, illetve az önálló szóként nem előforduló összetételekre olyan elemzést ad vissza, amelyben az első összetételi tag (pl. az igekötő) elvész. Ezt kijavítottuk az elemző forráslexikonának módosításával, és az alább leírt méréseket már ezen a javított lexikonon végeztük.

Következő mérésünknél (14. táblázat) azzal az alapfeltételezéssel éltünk, hogy az indexelőrendszer egyetlen optimális elemzést, illetve tövet vár a tövesítőalkalmazástól, amelynek ezt a kontextus ismerete nélkül kell meghatároznia. A Snowball és az Ocastem esetében ez a feltétel eleve teljesül, a többi eszköz esetén egyenként kimértük, hogy a teljes korpuszon a tövesítő által visszaadott első tő vagy pedig a leghosszabb tő használata adta-e a jobb eredményt, és azt használtuk a kiértékelésnél. Másként fogalmazva: az egyes tövesítők egyéni rekordja került a végeredménybe: mindegyiket a teljes korpuszon lemértük, hogy a leghosszabb- vagy az első tő ad-e jobb eredményt, és a jobbikat vettük figyelembe. Két részkorpusz kivételével megint a Humor-alapú lemmatizáló adta a legjobb eredményt. A Hunmorph-foma esetében az eredeti nem javított adatbázissal csak 71,5% pontosságot kaptunk.

DOI:10.15774/PPKE.ITK.2016.003

alkorpusz	***stemmer nélkül***	***Hunspell*** *(első tő)*	***Hunmorph-foma*** *(leghosszabb tő)*	***Hunmorph-compound*** *(első tő)*	***Hunmorph*** *(első tő)*	***Ocastem***	***Snowball***	***Humor*** *(leghosszabb tő)*
szépirodalom	52,4	86,6	76,2	86,6	86,4	**88,7**	58,3	88,4
tanulók	52,9	**88,6**	78,1	88,2	88,1	88,0	57,0	88,3
újságcikk	57,3	84,5	75,5	83,1	81,8	88,6	64,7	**92,8**
IT	57,9	81,9	75,8	81,7	79,3	87,9	68,6	**92,5**
jogi	62,0	81,8	77,4	82,4	80,8	86,7	72,4	**93,8**
üzleti	55,5	78,1	68,9	80,2	78,9	87,6	65,2	**91,4**
összesen	56,2	83,9	75,6	84,0	83,0	87,9	64,0	**91,0**

14. táblázat. Tövesítő modulok lemmapontossága az első/leghosszabb javaslat figyelembevételével a Szeged Korpuszon

A tövesítő-kiértékelést továbbfejlesztettem, és egy méréssorozatban nem csak a tövesítő kimenetét vizsgáltam, hanem IR rendszerbeli teljesítményét is megmértem (5.2. fejezet). Az IR alapú tövesítő-kiértékelés magyar vonatkozású eredményét itt közlöm (15. táblázat), hogy könnyebben összehasonlítható legyen a tövesítő kimenetén mért metrikák (5.1.1. fejezet) eredményeivel, különös tekintettel a lemmapontosság kiértékelésre (14. táblázat).

domén	**tövesítő nélkül**	**Hu-light**	**Snowball**	**Hunspell**	**Humor**
szépirodalom	25,2	61,9	66,7	67,8	**78,3**
tanuló	14,0	55,5	56,3	69,0	**75,4**
újság	16,5	79,6	81,1	77,1	**85,9**
IT	19,7	71,7	73,8	73,4	**81,8**
jogi	18,3	52,6	53,4	70,1	**75,3**
üzleti	23,2	73,1	73,5	44,4	**87,9**
összesen	18,4	65,6	67,3	66,3	**80,1**

15. táblázat. Tövesítők IR-kiértékelése magyar nyelvre a Szeged TreeBank mondatai alapján (Lucene-motorral)

Méréseim alapján magyar nyelv esetén a tövesítő kimenetén és az IR alapon mért eredmények karakterisztikája hasonló, azaz minél pontosabb a tövesítés, annál jobb az IR minősége is..

A következő mérésben, melynek az eredménye a 16. táblázat harmadik oszlopában szerepel, arra voltunk kíváncsiak, hogy egy ideális orákulum használata esetén, amely ki tudná választani a helyes elemzést az összes közül, milyen pontosságot kapnánk. Ez a mérés azt adja meg, hogy az esetek mekkora részében szerepel az elemzések között a Szeged Korpuszban megadott lemma. Látható, hogy bár ebben a mérésben is a Humor-alapú lemmatizáló érte el a legjobb eredményt (96%), a 4%-nyi eltérésre nem ad magyarázatot a mindössze 1,5%-nyi ismeretlen szó. Az eltérések oka az, hogy a lemmatizálás a Szeged Korpuszban részben más

DOI:10.15774/PPKE.ITK.2016.003

elveken nyugodott, mint ahogy akár a mi lemmatizálónk, akár a többi eszköz működik. A legfőbb eltérések a következők:

A *-hat* toldalékos igék (pl. *futhatott*) lemmája a Szeged Korpusz adott verziójában tartalmazza a *-hat* végződést (*fut* helyett *futhat*)

A ragozott személyes névmások (pl. *rajtam, velünk*) lemmája a Szeged Korpuszban nem a személyes névmás (*én, mi*), hanem a *rajta*, *vele* alakok.

A melléknévi igenevek nem az igére vannak visszavezetve, hanem egyszerűen melléknévként vannak annotálva.

	Első tő	**Leghosszabb tő**	**optimális tőalternatíva-választással elérhető**
Hunspell	**83,9**%	83,2%	87,6%
Hunmorph-foma	73,8%	**75,6**%	91,0%
Hunmorph – compound	**84,0**%	78,4%	90,0%
Hunmorph	**83,0%**	82,2%	87,4%
Ocastem	87,9%	87,9%	91,4%[7]
Snowball	64,0%	64,0%	64,0%
Humor	89,6**%**	**91,0%**	**96,0%**

16. táblázat. A tövesítő modulok pontossága több tőalternatíva esetén más-más kiválasztási módszer mellett

A következő kiértékelésben azt feltételeztük, hogy az egyes tövesítők minden tőjavaslatát az indexbe helyezzük, és minden egyes a korpuszban megadott lemmától eltérő indexbe került tétel hibás találatot jelent (*false positive*, FP), illetve plusz hibapontot jelent, ha a korpuszban szereplő lemma nem szerepel a javaslatok között (*false negative*, FN). A helyes lemmák jelentenek jó találatot (*true positive*, TP). Így pontosságot (P=TP/(TP+FP)) és fedést (R= TP/(TP+FN)) számolva, és ezekből a pontosságot és a fedést egyforma súllyal figyelembe vevő F-pontszámot kiszámolva kaptuk a 17. táblázatban szereplő eredményeket.

alkorpusz	***stemmer nélkül***	***Hunspell***	***Hunmorph -foma***	***Hunmorph compound***	***Hunmorph***	***Ocastem***	***Snowball***	***Humor***
szépirodalom	68,8	87,5	62,0	37,5	56,1	**94,0**	73,6	89,8
tanulók	69,2	86,9	60,1	40,1	55,9	**93,6**	72,6	88,5
újságcikk	72,8	88,9	62,4	29,5	53,4	**93,9**	78,6	92,5
IT	73,3	90,3	63,3	30,9	53,3	**93,5**	81,3	92,9
jogi	76,5	90,2	64,3	29,0	48,7	**92,9**	84,0	92,3
üzleti	71,3	86,5	60,6	27,0	51,3	**93,4**	78,9	92,3
összesen	72,0	88,3	62,0	32,3	53,1	**93,5**	78,0	91,1

17. táblázat. Tövesítő modulok hibás alternatíváit szigorúan lepontozó F-pontszáma Szeged Korpuszon

[7] Az Ocastemet olyan beállítással lefuttatva kaptuk ezt az eredményt, hogy az összes lehetséges tövet adja vissza. A többi táblázatban szereplő beállítással itt is 87,9% állna.

DOI:10.15774/PPKE.ITK.2016.003

Ebben a mérésben a mindig csak egyetlen tövet visszaadó Ocastem lemmatizáló bizonyult a legjobbnak, bár az alapjául szolgáló rengeteg „vicces” elemzést generáló Hunmorph teljesítménye e szerint a szigorú metrika szerint nagyon messze elmaradt még attól a megoldástól is, ha egyáltalán nem végeztünk tövesítést. A Humor lemmatizáló ebben a megmérettetésben a második legjobb eredményt adta annak ellenére, hogy nem végeztünk szűrést a javaslatain, így az alternatív tövek sok hamis pozitív találatot adtak. A Humor elemző ugyanis sok produktív elemzést is képes adni (5. táblázat), ami jelen esetben jelentős büntetőpontokat jelentett, bár szűrhettük volna őket (konfigurációs beállításokkal). Ezt a mérést gyakran úgy végzik el, hogy a mérés az ajánlatok sorrendezését is minősítse. Ilyenkor a pontosságot nem az egész listán számolják, hanem azt mérik meg, hogy a találati listán végigmenve a maximális fedés elérésekor milyen pontosságot érünk el (*precision at maximum recall*) (Lindén 2009; Novák 2015). Ilyenkor a listákon a helyes tövet követő javaslatokat nem tekintjük téves pozitívnak. Így a 18. táblázatban látható eredményt kapjuk. Ebben a mérésben ismét a Humor a legjobb.

alkorpusz	méret	***stemmer nélkül***	***Hunspell***	***Hunmorph -foma***	***Hunmorph - compound***	***Hunmorph***	***Ocastem***	***Snowball***	***Humor***
szépirodalom	185 436	68,8	93	89,1	89,6	92,1	94,0	73,6	**94,7**
tanulók	278 497	69,2	94,1	88,4	91,3	93,0	93,6	72,6	**94,1**
újságcikk	182 172	72,8	91,8	90,0	85,9	89,5	93,9	78,6	**95,4**
IT	175 991	73,3	90,3	88,8	86,2	88,1	93,5	81,3	**94,9**
jogi	220 069	76,5	90,2	87,4	87,3	88,7	92,9	84,0	**93,7**
üzleti	186 030	71,3	88,2	89,1	83,3	87,7	93,4	78,9	**95,9**
összesen	1 228 195	72,0	91,5	88,6	87,6	90,1	93,5	78,0	**94,7**

18. táblázat. Tövesítő modulok alternatívákat pontozó kiértékelése a Szeged Korpuszon

Kiértékeltük a tövesítőket abból a szempontból is, hogy a korpuszban szereplő szóalakoknak a korpuszban ténylegesen szereplő lemmáit milyen jól fedik le. Minden szóalakhoz felvettük a korpuszannotációban szereplő összes lemmát, és ezt a halmazt hasonlítottuk össze az egyes tövesítők által visszaadott tövek halmazával. A metszet TP, a csak a korpuszban szereplő lemmák FN, a csak a tövesítő által javasolt tövek pedig FP jelölést kaptak. Így kaptuk a 19. táblázatban látható eredményeket. Ez az kiértékelés azt méri, hogy a tőalternatívák közül melyeknek van létjogosultsága az adott korpuszon, más szavakkal: egy tövesítőnek mennyire van esélye az adott korpuszon jó tövet adni. Ebben a kiértékelésben ismét a Humor-alapú lemmatizáló végzett az élen.

DOI:10.15774/PPKE.ITK.2016.003

alkorpusz	***stemmer nélkül***	***Hunspell***	***Hunmorph -foma***	***Hunmorph compound***	***Hunmorph***	***Ocastem***	***Snowball***	***Humor***
szépirodalom	57,6	88,2	80,7	63,3	74,1	83,1	57,9	**91,0**
tanulók	60,6	86,4	79,5	64,5	74,1	82,1	56,1	**88,2**
újságcikk	61,1	86,2	82,7	58,6	70,7	84,1	64,4	**93,1**
IT	62,0	84,4	82,4	58,9	68,4	83,1	68,3	**91,7**
jogi	64,5	84,1	83,0	59,0	67,6	83,1	72,4	**91,2**
üzleti	61,3	82,3	81,0	53,6	65,9	82,0	64,4	**93,2**
összesen	61,2	85,4	81,4	59,9	70,3	82,9	63,4	**91,1**

19. táblázat. F-pontszám az összes lehetséges helyes tő figyelembevételével a Szeged Korpuszon

Természetesen egy alkalmazásnál fontos a tövesítés sebessége, egy újraindexelés futási idejét ez nagyban meghatározza. A 20. táblázatban láthatók a Szeged Korpusz elemzésekor mért futási idők, illetve az egyes modulok sebessége. Itt a cache bekapcsolásával 3-szoros gyorsulást értünk el a Humor-alapú lemmatizáló esetében.

	Futási idő (1.2M token)	**token/s**
Hunspell	847,0 s	1 450
Hunmorph-foma	**44,1 s**	27 850
Snowball	70,2 s	17 495
Humor	163,2 s (cache-sel: 59,9 s)	7 525 (20 503)
Hunmorph	4464,8 s	275
Ocastem	1285,0 s	955

20. táblázat. A tövesítő modulok sebessége
(teszteléshez használt gép: 8-core 1.1GHz CPU, 74GB memória, 64 bit ubuntu)

5.6. Kiértékelés angol és lengyel nyelvekre

5.6.1. Vizsgált tövesítő modulok

A 5.5.1 fejezetben bemutatott tövesítőkön túl az következő tövesítőket értékeltem ki. Az Apache Lucene (McCandless, Hatcher, és Gospodnetic 2010) népszerű, nagy teljesítményű Java-alapú keresőmotor, amely számos beépített tövesítővel rendelkezik. Ezen tövesítőit teszteltem: **KStem**, **Porter**, **EnglishMinimal**, Hunspell, Humor angol nyelvre, **Stempfel**, **Morphologik**, Hunspell, Humor lengyel nyelvre. A félköverrel jelölt modulok elérhetők a Lucene-ben. A Lucene Java-ban írták, a Hunspellt viszont C++ nyelven. A Lucene rendelkezik ugyan egy saját Java Hunspell implementációval is, de annak még a Lucene fórumokban is rossz híre van: az nem egyenértékű az eredeti, C++ verzióval, minősége messze

DOI:10.15774/PPKE.ITK.2016.003

elmarad attól. Így a Hunspell esetén az eredeti implementációt használtam[8], a tövesítési funkcióhoz újra kellett fordítani, és illeszteni friss Lucene API-hoz. A Humor modul C++ kódját - hasonlóan a Hunspellhez - java natív csomagolással illesztettem a Lucene API-jához.

A kiértékelést nyelvenként 6+2 metrika szerint végeztem el: a közvetlen kimenetre definiált 6 metrika kiértékelését (5.1.1. fejezet) és az IR kiértékelés pedig 2 metrika szerint: összes találat illetve az első *n* találat kiértékelését (5.2. fejezet). Ezek közül két metrika eredményeit mutatom be a következő fejezetben: a lemmapontosság - és az IR összes találat kiértékelését.

5.6.2. Eredmények

Érdekes módon az angol nyelv eredményei a magyartól és a lengyeltől eltérő tendenciát mutattak. A Kstem, Porter, En-minimal és Snowball fedése magas volt, de a pontossága nagyon alacsony. Ezt FP találatok milliói okozták. (Ezzel ellentétesen, a tövesítő nélküli mérésben a pontosság magas volt és a fedés alacsony.) Ennek egyik oka az volt, hogy az előbbi stemmerek a kötőjelek mentén részekre vágják a szavakat (például a *low-print-run* töve *low, print* és *run*; vagy a *man-in-the-street* szóból *man, in, the, street* lesz), ami alacsony pontosságot okoz. A másik hibatípus az overstemming volt: sok olyan - csupán közös prefixszel rendelkező - teljesen eltérő szót vezetett vissza ugyanarra a tőre. Ez drasztikusan rontotta a pontosságot. Ráadásul, a kötőjelek mentén való darabolás illetve az overstemming nem csak alacsony pontosságot, hanem nagyobb indexet is okoz (21. táblázat és 22. táblázat). Az IR kiértékelésben a vizsgált modulok szignifikánsan különböztek (*t-test* alapján), kivéve a Porter és Kstem modulokat. Angol esetén a két kiértékelési módszer, a közvetlen kimeneten - illetve az IR alapon mért eredmények karakterisztikája eltérő volt, a lemmapontosságban a Kstem modul -, az IR kiértékelésben pedig az En-minimal modul volt a legjobb.

domén	tövesítő nélkül		Kstem		Porter		En-minmal		Snowball		Hunspell		Humor	
	F	oov	F	oov	F	oov	F	oov	F	oov	F	oov	F	oov
ACPROSE	78,4	0	92,6	0	67,1	0	91,6	0	71,4	0	87,1	9,8	**93,3**	4,8
CONVRSN	73,6	0	**93,6**	0	82,6	0	91,6	0	87,0	0	90,5	5,3	88,5	4,9
FICTION	72,8	0	90,3	0	75,8	0	86,4	0	79,5	0	86,6	6,1	**91,4**	3,9
NEWS	69,8	0	**92,1**	0	74,7	0	90,0	0	79,4	0	86,1	11,9	91,4	7,4
NONAC	74,4	0	92,1	0	70,1	0	90,6	0	74,5	0	86,7	9,9	**92,6**	5,3
OTHERPUB	74,2	0	**92,3**	0	73,4	0	91,3	0	77,6	0	87,5	9,8	92,3	6,0
OTHERSP	78,6	0	**94,3**	0	77,9	0	92,8	0	82,0	0	90,6	5,1	90,6	5,6
UNPUB	72,2	0	**92,7**	0	72,8	0	91,3	0	76,6	0	87,7	9,1	92,6	5,5
összesen	74,4	0	**92,2**	0	72,7	0	90,4	0	77,0	0	87,3	9,0	92,1	5,4

21. táblázat. A lemmapontosság kiértékelése angol nyelvre a BNC alapján

[8] https://github.com/dren-dk/HunspellJNA

DOI:10.15774/PPKE.ITK.2016.003

domén	token	tövesítő nélkül	Kstem	Porter	En-minimal	Snowball	Hunspell	Humor
tudományos	15,7M	59,9	60,1	47,9	**72,6**	42,8	53,5	48,0
társalgás	4,2M	48,8	55,9	57,1	55,5	57,2	51,7	**64,0**
regény és vers	16,1M	50,7	57,8	53,3	**60,4**	46,3	48,7	47,8
újságok	9,4M	53,5	61,4	54,2	**65,7**	53,4	54,1	58,7
nem tudományos próza	24,1M	55,1	56,1	46,6	**66,3**	41,9	51,1	46,3
más publikált anyagok	17,9M	54,4	55,9	49,6	**66,9**	46,1	50,9	53,2
más beszélt szöveg	6,1M	53,2	60,1	52,6	61,9	53,8	51,1	**65,1**
nem megjelent anyagok	4,4M	54,0	62,3	55,3	**67,1**	54,6	55,3	66,5
összesen	98,3M	54,4	57,9	50,1	**65,7**	45,9	51,5	51,3

22. táblázat. Tövesítők IR-kiértékelése angol nyelvre a BNC mondatai alapján (Lucene-motorral)

Angol nyelv esetén nagyon gyakori azonban, hogy egy szóalak egyszerre ige, főnév vagy akár melléknév is egyszerre. A kevés ragozott alak (egy szónak maximum 6-8 alakja lehet) miatt tömegesen fordul elő, hogy egy szóalak egybeesik egy másik szófajú szóval (23. táblázat). Ez azt eredményezi, hogy a legjobb tövesítők is csak alacsony F-értéket tudnak elérni ebben az IR kiértékelésben: kiszámolt töveik alapján sokkal több mondat jön eredményül, mint a gold standardben.

Szótő	Szófaj	Korpuszban előforduló alakjai
sink	SUBST	sink, sinks
sink	VERB	sunk, sink, sinking, sank, Sink, sinked
spread	VERB	spread, spreads, spreading, Spread
spread	SUBST	spread, Spread
spread	ADJ	spread
boring	ADJ	Boring, boring
bore	VERB	boring, Bore, bores, bored, bore
boring	SUBST	boring
park	SUBST	Park, park, parks, Parks
park	VERB	park, parking, parked, parks, Park, Parked
past	PREP	past, Past
past	SUBST	past
past	ADJ	past
past	ADV	past

23. táblázat. Angol nyelv esetén különösen gyakori, hogy egy szóalak többféle szófajú (BNC / társalgási alkorpusz)

Például a *sinks* szóalak esetén ha egy tövesítő a *sink* tövet adja, akkor egy IR rendszer a *sinks*-re keresve az összes olyan mondatot visszaadja, ahol a *sink* akár igeként, akár főnévként szerepelt. Azonban a gold standardben ennek csak egy töredéke lesz: azok a mondatok, ahol főnévként fordult elő (mert a BNC társalgási alkorpuszában a *sinks* csak főnévként fordul elő,

DOI:10.15774/PPKE.ITK.2016.003

lásd 23. táblázat). Az így keletkezett sok *false positive* találat jelentősen lerontja a tövesítő pontosságát.

A lengyel eredmények azt mutatták, hogy a tövesítés minősége nagy hatással van az IR kiértékelés F-mértékére: az IR minőségét kétszeresen is javíthatja a tövesítő (25. táblázat). A lengyel nyelv esetén a Morfologic stemmer volt a legjobb a lemmapontosságban (24. táblázat), az IR kiértékelésben pedig a Hunspell érte el a legjobb eredményt (25. táblázat).

A magyar IR eredmények a 5.5.2. fejezetben találhatók (14. táblázat, 15. táblázat), ahol a magyar tövesítők kiértékelését tárgyalom. Itt csak röviden összefoglalva: a magyar, mint gazdag morfológiájú nyelvnél a két kiértékelési módszer - a közvetlen kimeneten illetve az IR alapon mért - karakterisztikája hasonló. Minél pontosabb volt a lemmatizálás, annál jobb eredményt adott az IR is.

domén	**token**	**tövesítő nélkül**		**Stempfel**		**Morfologik**		**Hunspell**		**Humor**	
		F	**oov**	**F**	**oov**	**F**	**oov**	**F**	**oov**	**F**	**oov**
szépirodalom	54.205	53,80	0	78,36	2,7	**89,34**	2,7	84,45	5,6	88,71	7,8
tájékoztató	56.779	54,08	0	77,31	3,9	**89,09**	3,9	84,38	7,7	87,87	5,8
párbeszédek	59.024	68,26	0	72,60	9,5	**83,62**	9,5	78,58	11,3	79,63	12,8
regény	169.270	55,81	0	77,48	4,1	**88,98**	4,1	84,38	5,9	87,84	4,2
beszélt rádió	23.303	64,52	0	73,94	6,4	**86,11**	6,4	82,12	8,1	83,43	10,2
kutatás oktatás	20.229	50,23	0	79,56	3,6	**89,64**	3,6	85,90	7,1	89,38	5,8
internet	72.273	55,27	0	77,15	3,2	**88,78**	3,2	83,48	8,0	86,02	7,8
újság	506.214	51,78	0	79,23	2,8	**90,03**	2,8	86,10	6,1	89,16	4,6
írott parlamenti	66.315	51,24	0	78,77	4,3	89,85	4,3	85,48	7,4	**89,86**	2,7
eszközök	30.998	52,22	0	80,82	1,2	90,65	1,2	87,48	7,5	**91,38**	2,3
nem osztályozott	10.140	54,38	0	78,72	2,0	**90,11**	2,0	85,55	3,9	88,52	3,9
összesen	1.028.671	54,21	0	78,2	3,6	**89,25**	3,6	85,02	6,7	88,08	5,4

24. táblázat A lemmapontosság kiértékelése lengyel nyelvre a PNC alapján

DOI:10.15774/PPKE.ITK.2016.003

domén	tövesítő nélkül	Stempel	Morfologik	Hunspell	Humor
szépirodalom	47,5	64,3	67,3	**71,6**	70,2
tájékoztató	46,3	67,1	70,6	**73,9**	73,8
párbeszédek	56,8	60,3	59,8	**64,4**	59,4
regény	41,0	57,5	61,5	**65,4**	64,1
beszélt rádió	60,2	68,0	65,3	**71,6**	67,1
kutatás oktatás	55,0	76,9	80,1	**82,5**	81,5
internet	50,0	63,4	65,3	**70,3**	67,3
újság	30,7	58,7	64,6	**67,5**	67,4
írott parlamenti	42,9	66,7	79,9	82,6	**83,1**
eszközök	47,8	77,2	81,2	83,3	**85,0**
nem osztályozott	64,7	73,3	72,3	**77,9**	76,5
összesen	36,9	60,4	65,3	**68,7**	68,0

25. táblázat. Tövesítők IR-kiértékelése lengyel nyelvre a PNC mondatai alapján (Lucene-motorral)

5.7. Kapcsolódó kutatások

A tövesítők számos módon kiértékelhetők. Elsősorban teszt lekérdezésekkel (*test collection*) mérhető az IR teljesítményére gyakorolt hatásuk. A korábbi tövesítő kiértékelések (Hull 1996; Tordai és De Rijke 2006; Halácsy és Trón 2007) előre definiált lekérdezésekből és az ezekre elvárt találati listákból állt. Ezeket *experimental collection*-nek (kísérleti gyűjteménynek) nevezik. A pár száz lekérdezést és a hozzájuk tartozó találati listát kézzel állítják össze, majd az egyes tövesítőkkel kapott találati listát ezzel értékelik ki (dokumentumok száma 1.000-120.000 közötti).

A lekérdezéseket nagyon gondosan kell összeállítani: ennek döntő hatása van a kiértékelés eredményére. A sorrendezés (*ranking*) szintén része a kiértékelésnek. Ezek a kiértékelések megmutathatják egy-egy tövesítő hasznosságát, de a tövesítő készítője nem kap közvetlen, konkrét visszajelzést a modul tipikus hibáiról. Ezért olyan hibametrikákat definiáltak (Paice 1994), amelyek betekintést adnak nem csak a tövesítők összehasonlíthatóságába, hanem részletes információt is szolgáltatnak a hibákról.

Alapvetően kétféle hibát különböztethetünk meg: *overstemming* (túl-tövesítés) és *understemming* (alul-tövesítés). Magyarra alkalmazva ez a felosztás azt feltételezi, hogy egy szó végéről betűket levágva eljuthatunk az ideális tőhöz. Azonban ez nyelvünk esetében nem igaz: vannak lemmák, amik így sosem állnak elő: *lehettünk → van*, *bokrokat → bokor* stb. Ezzel együtt is a Paice-mérőszámok magyar esetén is jelzik, hogy egy adott szót a lemma/stem milyen más szóalakokkal rendeli össze. Ilyen értelemben beszélhetünk *over-* ill *understemming*ről.

DOI:10.15774/PPKE.ITK.2016.003

Az *overstemming index* (OI) azt mutatja, hogy a tövesítő hány alkalommal vágott le túl sok betűt a szavakról. Minél magasabb az OI értéke, annál több nem releváns szót rendel hozzá az IR ugyanahhoz a tőhöz, így a pontosság csökken. Az *understemming index* (UI) azt számolja, hogy egy tövesítő hány alkalommal nem távolított el suffixet. Ez a hiba azt eredményezi, hogy egy szó ragozott alakjait nem rendeli egymáshoz. Egy IR-ben ennek az a következménye, hogy alacsony lesz a fedés. Az ideális eset az, amikor az OI és UI értéke is nulla. Az OI/UI arány a tövesítő súlyát vagy erejét jelenti: egy gyenge tövesítő csak keveset vág le a szavak végéről, egy erősebb ugyanezt nagyobb étvággyal teszi. Ez a súly nem a tövesítő minőségét jelenti, csupán jellemzi a végződés-eltávolítási hajlandóságát. Egy másik arányt definiáltak arra, hogy a tövesítők összehasonlíthatóak legyenek: *error rate relative to truncation* (ERRT, a csonkolás relatív hibaaránya), amely az (UI, OI) sorozatok ideálistól való eltérését mutatják.

Több elérhető, nyílt forráskódú IR rendszer létezik, mint például Lucene (McCandless, Hatcher, és Gospodnetic 2010) és az erre épülő rendszerek: Apache Solr (Smiley és mtsai. 2015), Elasticsearch (Gormley és Tong 2015). Ezekben több tövesítő is található. Azonban nem könnyű eldönteni, melyiket érdemes választani egy adott feladathoz. A tövesítő kiértékelések régebbiek, mint ezek a keresőrendszerek és tövesítőmoduljaik, és jelen sorok írásakor nem találtam a moduljaikról elérhető kiértékelést.

DOI:10.15774/PPKE.ITK.2016.003

5.8. Összefoglalás

A tövesítő modulok kiértékeléséhez új módszereket hoztam létre, majd ennek segítségével 10 tövesítőt 3 nyelven kiértékeltem. Egy hiánypótló, naprakész kiértékelést szerettem volna adni a tövesítőkről, egyben biztosítani egy módszert és egy nyelvfüggetlen eszközt is (Python+Java), amely további nyelvek és tövesítők kiértékeléséhez használható.

A kiértékeléshez többféle metrikát definiáltam, amelyek a lemmapontosságot, az egyes tőalternatívák helyességét és sorrendjét, illetve amely egy szó esetén az összes lehetséges helyes tövet is figyelembe veszi. Ezek a metrikák abban segítettek, hogy minél több szempontból meg lehessen mérni egy tövesítő teljesítményét: mekkora a lemmapontossága, több lemma alternatíva esetén mennyire jól sorrendezi őket, IR rendszerbe építve felveszi-e a versenyt a nála sokkal egyszerűbb és gyorsabb stemmerekkel.

Lemmával annotált korpuszokból hoztam létre gold standard-et, amivel a tövesítők lemmapontosságát és keresésnél mérhető hatását külön mértem meg. Az elkészült eszköz nyílt forráskódú.

Kapcsolódó tézisek:

4. tézis: Létrehoztam egy kiértékelési módszert, amellyel tetszőleges, lemmával annotált korpusz alapján lemérhető egy tövesítő (i) pontossága, (ii) IR minősége, (iii) UI, OI, ERRT értéke, és (iv) egyéb metrikák szerinti kiértékelése. Mindezt angol, lengyel és magyar nyelvre 10 tövesítőre kimértem.

4.a tézis: Létrehoztam egy módszert, amivel korpuszból automatikusan létrehozható olyan gold standard, amely tövesítők közvetlen illetve IR-rendszerbeli kiértékeléséhez használható.

4.b tézis: A kiértékelést angol, lengyel és magyar nyelvekre elkészítettem.

4.c tézis: Kimutattam, hogy az erősen ragozó nyelveknél (lengyel, magyar) a lemmapontosság és az IR-minőség korrelál.

4.d tézis: Kidolgoztam egy IR-tövesítőkiértékelést, ami natív IR rendszer nélkül képes IR-rel korreláló eredményt mérni.

4.e tézis: Definiáltam a tövesítő közvetlen kimenetére olyan kiértékeléseket, amelyek alkalmasak a tövesítők összehasonlítására, és visszajelzést is adhatnak a tövesítő hibáinak feltárására és azok kijavítására.

4.f tézis: Kimutattam és megmértem, hogy az IR-tövesítőkiértékelés első *n* találatának kiértékelése alkalmas az IR-ranking algoritmus kiértékelésére is.

A tézishez kapcsolódó publikációk: [2], [4]

DOI:10.15774/PPKE.ITK.2016.003

6. Mintázatok keresése korpuszban

A következő fejezetekben áttekintem a korpuszvezérelt munkafolyamatot: az ötleteket és feladatokat a korpuszból merítettük, és azon is ellenőriztük. Elsőként a gyakori minták: szó- valamint szófajsorozatok kigyűjtését végeztem el (6.2. fejezet). Ez önmagában is érdekes eredményt hozott: a gyakori minták nem csak a korpuszról képesek jellemző adatokat adni, hanem a korpuszba kódolt világismeret is bizonyos fokig kinyerhető velük.

A második nagy terület a főnévi csoportok felismerése volt (6.3. fejezet), amely a mondatok hatékonyabb elemzéséhez szükséges. A főnévi csoport azonosítására szabályalapú- és statisztikai módszert is kipróbáltam. A felismert főnévi csoportokat egy kereshető online rendszerbe töltöttem be (6.5. fejezet), ahol mind a szófaji minták, mind a mondatvázak kereshetőek.

6.1. Bevezetés az NP-felismeréshez

A mondatok kombinatorikus változatossága elsősorban a főnévi csoportok (*noun phrase*, továbbiakban NP) számosságától is függ. Egy mondatszerkezet drasztikusan egyszerűsödik, ha a benne szereplő NP-ket egy-egy összevont egységnek kezelünk, ahogy azt ebben a fejezetben látni fogjuk. Az összevont, egyszerűsített formában könnyebb a mondatokat további elemzésnek alávetni.

Az MTA-PPKE Magyar Nyelvtechnológiai Kutatócsoport által fejlesztett pszicholingvisztikai megközelítésű elemző működéséhez szükséges a magyar nyelv mondatainak felépítését ismerni. Két ponton is kapcsolódik ehhez a jelen munka: a mondatszerkezetek közül azok az érdekesek számunkra, amelyek előfordulnak. Sok nyelvészeti probléma azon a ponton válik egyre bonyolultabbá, minél inkább minden esetre fel szeretnénk készíteni. Ez az ipar más területein is igaz: ha egy 95%-os rendszer elkészítése egységnyi erőforrást igényel, akkor egy 99%-os rendszer többnyire ennek sokszorosát. Ezért arra teszünk kísérletet a kutatócsoport ezen projektjében, hogy a valóban leírt (és szövegekben ténylegesen előforduló) esetekre és szerkezetekre koncentrálunk. Azzal a feltételezéssel tesszük ezt, hogy ha egy szerkezet nem fordul elő a valóságban (azaz nem írták le a weben és egyéb korpuszokban), akkor az most nem fontos. (Természetesen számít a korpusz nagy mérete, ez garantálja az *overfitting* elkerülését.) Ehhez a projekthez a 2. fejezetben leírt korpuszépítés az egyik segítség. A másik az NP-felismerés javítása, amivel a mondatelemzés könnyebbé válik. Ezért szükségessé vált áttekinteni az NP-felismerés jelen módszereit, minőségét, és lehetőség szerint javítani rajta.

DOI:10.15774/PPKE.ITK.2016.003

Azt a feladatot, amikor címkéket rendelünk egy mondat szavaihoz, szekvenciális címkézésnek (*sequential tagging*) nevezzük. Ismertebb esetei ennek a feladatnak a szófaji címkézés (*Part-of-Speech tagging*): ekkor minden tokenhez egy szófajt rendelünk. Rokon feladat a *chunking*, amikor csoportokat ismerünk fel, például főnévi vagy igei csoportot. Speciális esete ennek a névelem felismerés (*Named-Entity Recognition*), ahol először névelem csoportokat azonosítunk, majd egy típust is rendelünk hozzá (személy, hely stb.) A *chunking* a csoporton kívüli elemeket is jelöli, így speciális címkézésnek tekinthető: a mondat minden tokenjéhez úgynevezett IOB-címkét rendel.

Minden egyes tokenhez hozzárendelünk egy címkét, amely azt jelzi, hogy a token egy elem elején (B, *begin*), belsejében (I, *inside*), vagy azon kívül (O, *outside*) áll. Ezeket IOB-címkéknek nevezzük, és számos reprezentációjuk létezik (Tjong Kim Sang és Veenstra 1999). Van, amelyik külön jelöli a végét is (*end*, E), vagy az egy hosszúságú elemeket (S vagy 1). A 26. táblázat a többféle IOB reprezentációt mutatja be. Ezen kívül típusa is lehet az egyes sorozatoknak, ami az adott címkézési feladatnál fontos (például nem csak NP-t jelöl a tagger, hanem VP, PP stb. típusokat is). A címkézési feladatoknál az a formátum terjedt el, hogy minden egyes szó új sorban szerepel, mellettük a tulajdonságaik, hagyományosan az utolsó oszlopban pedig az IOB-címke.

szó	IOB1	IOB2	IOE1	IOE2	OC
These	I	B	I	E	[]
include	O	O	O	O	O
,	O	O	O	O	O
among	O	O	O	O	O
other	I	B	I	I	[
parts	I	I	I	E	]
,	O	O	O	O	O
each	I	B	I	I	[
jetliner	I	I	E	E	]
's	B	B	I	I	[
two	I	I	I	I	O
major	I	I	I	I	O
bulkheads	I	I	I	E	]
,	O	O	O	O	O

26. táblázat. Több IOB reprezentáció: egy mondatrész öt különféle IOB-címkekészlettel

Az NP- és mondatmintákat egy nagy korpuszban figyeltük meg. Ha egy bizonyos minta gyakorisága egy előre meghatározott küszöbérték fölött volt, akkor úgy tekintettük, mint egy ismert mintát. Ezután, minden ismert NP-t kicseréltünk "*NP*"-re a mondatban, így a mondatszerkezet komplexitása jelentősen lecsökkent (ez látható a 27. táblázatban).

DOI:10.15774/PPKE.ITK.2016.003

eredeti mondat	Egyedi elbírálást kér a kormánytól a károk enyhítésénél az árvízsújtotta Felsőzsolca önkormányzata.
NP-k zárójelekkel jelölve	(*Egyedi elbírálást*) kér (*a kormánytól*) (*a károk enyhítésénél*) (*az árvízsújtotta Felsőzsolca önkormányzata*).
NP-k helyettesítve	*NP*-t kér *NP*-től *NP*-nél *NP*.

27. táblázat. Példa az NP-felismerésre: a mondatszerkezet egyszerűbbé válik, ha a főnévi csoportokat jelöljük

Az egyszerűsített mondatszerkezet nagyban megkönnyíti a mondat elemzését. Az elemzőnek nem kell foglalkoznia a főnévi csoportok belső szerkezeteinek gazdagságával, ennélfogva jóval kevesebb esetet kell csak kezelnie. Az alany, az állítmány attribútumai, a határozók egy-egy szónak felelnek meg a mondatban, ami az elemzőnek ideális.

Létezik eszköz, ami kifejezetten az állítmány attribútumait kutatja: a Mazsola rendszer (Sass 2008; Sass 2011). A Mazsola online felületén kereshetők gyakoriság szerint egy adott ige attribútumai, illetve fordított irányból, egy attribútumhoz kereshetők a vele leggyakrabban használt igék. Az eredményt címkefelhőben mutatja, illetve a konkrét előfordulásokat konkordancialistában. Jelen megközelítés különbözik ettől: amíg a Mazsola az igékre és vonzataikra fókuszál, addig az egyszerűsített mondatszerkezet (27. táblázat) az NP-felismerésre teszi a hangsúlyt. Nem vizsgálom az NP-k egymással való viszonyát vagy az NP-felismerés után megmaradt mondatrészek további vizsgálatát, a Mazsola rendszer viszont pont ennek a specialistája.

6.2. Szóstatisztika, N-gram keresés

A 2.1 fejezetben bemutatott webkorpusz minőségi mutatók két elemét (*karakterek vagy N-gramok eloszlása*) is implementáltam: egy általános N-gram számlálót készítettem, ami a szövegben előforduló szó *n*-eseket, illetve szófaj *n*-eseket számolja meg. Ez megfelel az N-gram és POS N-gram minőségellenőrzéseknek (Biemann és mtsai. 2013), amelyek rávilágíthatnak egy korpusz esetleges hibáira.

A szószámlálás leprogramozása során egy érdekes jelenséget tapasztaltam. Azok a szósorozatok, amelyek egy gyakori mintázat belsejében találhatók, többször is megszámolásra kerültek. Ez nem kívánt jelenség, hiszen ezáltal egy ritka szósorozat gyakorinak tűnhet.

Azt tapasztaltam, hogy egyszerű szó *n*-es statisztika esetén egy gyakori (általában rövid) minta gyakorinak mutatja az őt tartalmazó mintákat. Más szavakkal, ha egy szó *n*-es gyakori, akkor ennek a rövidebb (*n-1, n-2* stb.) változatai is legalább olyan gyakoriak lesznek, pedig ezeket már beleszámoltuk a szó *n*-esbe. Ez torzítja a statisztikát, mert nem látszódik a rövidebb változatok önálló gyakorisága. Például tegyük fel, hogy egy korpuszban a következő gyakorisággal rendelkeznek ezen szó *n*-esek: "*az üveghegyen túl*": 100, "*az üveghegyen*": 150, "*az*": 1000. Ekkor "*az üveghegyen*" 150-100=50-szer fordul elő a önállóan (a "*az üveghegyen*

DOI:10.15774/PPKE.ITK.2016.003

túl" kifejezésen kívül), az "*az*" pedig 1000-150=850-szer. A helyzetet bonyolítja, hogy lehetnek egymásba ágyazott minták is, ahol nem szabad egyszerűen kivonni, mert az előfordulásra negatív érték jönne ki. (Például: "*ABABA*": 100, "*AB*": 150, "*A*": 200 esetén ha az "*ABABA*" miatt levonnék 3×100 "*A*" előfordulást, akkor negatív érték jönne ki "*A*"-ra.) Általánosan fogalmazva: egy N-gram pontos gyakorisága úgy számolható ki, ha a gyakoriság értékét csökkenjük a benne megtalálható rövidebb N-gramok gyakoriságával.

Mi a valódi gyakoriságra vagyunk kíváncsiak, ezért a fenti jelenség kiküszöbölésére egy algoritmust találtunk ki. Az algoritmusnak két fontos bemeneti paramétere van a keresendő szó *n*-esekre vonatkozóan: egy *limit*: a szó *n*-esek minimális gyakorisága, a másik *n*: a szó *n*-esek maximális hossza. A program először végigmegy a fájlon, és minden mondatban az összes lehetséges szó *n*-est megszámolja.

Ekkor adott az összes szó *n*-es és a gyakoriságaik. Ezután a szövegen újra végigmegy *n* hosszú mintákat keresve. Ha egy minta az adott limit fölött van, akkor a benne szereplő kisebbek gyakoriságát csökkenti, és megjegyzi a pozíciókat: ezen a mondaton belül többet ezeket ne vonja majd le, illetve ezt a mintát kiveszi (később se akarja kivonni senki). Majd az utolsó lépést *n-1* hosszú mintákkal megismétli, *n* lépésben csökkentve a keresendő hosszt, egészen 1-ig. Azért van szükség *n* menetre, mert egy minta gyakorisága lehet, hogy lecsökken az algoritmus közben a *limit* alá, így eldobhatóvá válik. Az pedig, hogy egy minta *limit* fölötti-e, kulcsfontosságú a levonás eldöntésekor.

Ezt az N-gram kereső algoritmust lefuttattam a 9. fejezetben leírt korpuszra. Egyrészt a módszer hatékonyan ki tudta mutatni a korpusz kezdeti problémáit (például a felülreprezentált szósorozatok jelezték, hogy a korpusz sok duplikátumot tartalmaz: javítani kellett a cikk-kinyerő algoritmuson, bővebben a 2.3. fejezetben). Másrészt a leggyakoribb minták egyfajta képet adtak a korpuszban leírt világról (28. táblázat). Ez a keresési megközelítés megmutatta, hogy a korpusz sok információt tartalmaz a világról: híres emberek és helyek, egy adott szerep jellemző tevékenységei mind kódolva vannak a korpuszban. Ez a tény arra inspirált minket, hogy ezt az erőforrást használjuk ki valahogy, hogy jobb elemzőt készíthessünk.

Az N-gram keresés átfogó képet adott a korpuszról, a hibáiról, sok értékes NP-t is meg tudott adni, a leggyakoribbakat is. Azonban ez természetesen nem NP-felismerés. A következő fejezetben megmutatom, hogy mennyiben lehet ezt NP-felismerésre használni, és mintákra épülő szabályokkal milyen minőséget tudtam elérni.

DOI:10.15774/PPKE.ITK.2016.003

minta: [főnév] [főnév] [melléknév] [főnév]	
Matolcsy György nemzetgazdasági miniszter	694
Barack Obama amerikai elnök	664
Sólyom László köztársasági elnök	367
Angela Merkel német kancellár	345
Nicolas Sarkozy francia elnök	256
Schmitt Pál köztársasági elnök	229
Vlagyimir Putyin orosz elnök	186
minta: [névelő] [főnév] [ige]	
A szóvivő elmondta	660
A szakember elmondta	480
A miniszter közölte	320
A bíróság megállapította	29
minta: [főnév] [főnév]	
Orbán Viktor	8181
New York	4085
Wall Street	1358
Harry Potter	691
Johnny Depp	645
Angelina Jolie	627
Puskás Ferenc	380

28. táblázat. N-gram példák a korpuszból

6.3. Szabályalapú NP-felismerés

Az előző fejezetben bemutatott N-gramok szerkezete azt sugallta, hogy érdemes lenne egy szabályalapú NP-felismerőt készíteni, hiszen a leggyakoribb minták között – akár szófaj címke sorozatoknál, akár tövek vagy felszíni alakoknál – magas számban NP-k szerepeltek. Így készítettem egy NP-nyelvtant leíró nyelvet, amiben a népszerű reguláris kifejezések szintaxisával lehet szófajokra, tövekre, felszíni alakokra illetve ezek sorozatára szabályokat definiálni. Néhány szabályt mutat be a 29. táblázat. A legegyszerűbb szabályok egyfajta makrók: egy adott típus(oka)t egy másikkal helyettesít. A makrók három célt szolgálnak: egyrészt az eltérően viselkedő elemekre lehet külön hivatkozni, mintha egy új szófajt definiálnék, azonban nem kell sem a morfológiát módosítani, sem a szöveget újra elemezni. (Például a *mert* kötőszó egy új szófaji címkét kap: *KOT* helyett *BECAUSE*.) Másrészt több, hasonlóan viselkedő elemre lehet közös névvel hivatkozni (például *CAS=ACC,ABL,ADE,SUP,...*). Harmadrészt itt van mód arra, hogy a szótőre, felszíni alakra megkötést definiáljunk, és később erre ezzel a típussal hivatkozzunk. Mindhárom eset

DOI:10.15774/PPKE.ITK.2016.003

rövidebb és olvashatóbb szabályokat eredményez. A szabályok pedig az előbb definiált elemek sorozatát engedi vagy tiltja.

A Humor elemző szófaji címkéiből illetve Ligeti-Nagy Noémi által definiált pontosabb címkék (Ligeti-Nagy 2015) felhasználásával hoztam létre a szabályokat. A Szeged TreeBank (Csendes és mtsai. 2005) mondataiból a maximális NP-ket kigyűjtöttem, és átalakítottam a NP keresés formátumára (lásd 6.1. fejezet). A kiértékelés chunk alapon történt, helyes NP chunk *tp*, helytelen chunk *fp*, hiányzó chunk pedig *fn* találatnak számított. Ez alapján számoltam az F értéket. Végeredményben 44 szabály, 120 makró segítségével el tudtam érni olyan F értéket (30. táblázat), ami megközelít korábbi méréseket (Miháltz 2011).

Összefoglalva szabályalapú maximális NP kereső algoritmus készült, amely szófajilag annotált mondatokon futott. Ez képes felismerni nagy, akár 20 szavas (!) NP-t is. A felismerés minősége a Szeged TreeBank korpuszon 80,5%. Ezzel a módszerrel nem sikerült átütő eredményt elérnem (30. táblázat).

Makró definíciók	**jelentése**
`THAT:cat="KOT" surface="hogy"`	*hogy* legyen *THAT*
`NAND:cat="KOT" surface="!és"`	*NAND* legyen minden olyan kötőszó, ami nem *és*
`MN:cat="HA" surface="elég" nextRegEx="(((MN|SZN)\+(NOM|PL|DAT|ACC))|IGE\+_OKEP)"`	*elég* csak akkor legyen MN, ha MN van mellette (pl. *elég jó*)
Szabályok	**magyarázat**
`((FN[\+\w\d]*) COMMA \2)`	felsorolás
`(FN\+TER)(?!.*(IGE\+_OKEP|FN\+TER|FN\+INE))`	*ig* lezárja az NP-t, kivéve: "*x-ig terjedő …*", "*talpig vasban*", "*térdig sáros*"
`(NN_OK([\w+]*) (?:COMMA )?DET (?:MN[\w+]* )*FN(?:\+PSe3)?\2)`	Pl. "*azzal, a kellemetlenséggel*"

29. táblázat. Reguláris kifejezéseken alapuló szabályok NP-felismeréshez

	Memóriaigény	**Token/s**	**Pontosság**	**Fedés**	**F**
HunTag – 2011	4,2GB	650	78,6%	84,9%	**81,7%**
szabályalapú NP-felismerés	32KB	4480	79,6%	81,4%	80,5%

30. táblázat. Szabályalapú NP-felismerés kiértékelése (8-core 1.1GHz CPU, 74GB memory, 64 bit ubuntu server)

DOI:10.15774/PPKE.ITK.2016.003

6.4. Szemantikai támogatás az NP-felismeréshez

A szabályalapú NP-felismerés készítése során olyan hibába ütköztem, ami a szabályok, vagy a szintaxis szintjén nem megoldható. Az algoritmus pontosságát rontották az olyan NP-k, amelyek (hibásan) két kisebb NP-re szakadtak. Ezt főként az *"és"* okozta, ahogy a 31. táblázat mutatja.

Aláírják *(a finanszírozási szerződést)* *(a Budapesti Közlekedési Központ igazgatósága)* és *(a Fővárosi Közgyűlés jóváhagyásával)*	Az eredeti mondat
Aláírják *NP-t* NP és *NP-vel.*	Hibás, egy NP-t két részre bontva
Aláírják *NP-t NP-vel.*	Helyes, két minimális NP-t összevonva

31. táblázat. Példa egy szintaktikailag nehezen felismerhető NP-re

Ez a probléma nem oldható meg csupán szintaktikai módszerekkel. Az algoritmusnak „tudnia" kellene, hogy a két NP kapcsolódik egymáshoz, és ezek együtt alkotnak egy nagyobb NP-t. Hogyan tudjuk implementálni és használni szemantikai információt NP-felismerés érdekében?

A fejezetben bemutatott módszer javít az NP-felismerés minőségén, korpuszból tanult szemantikai összefüggések alapján. Olyan eseteket is tud kezelni, amelyeket pusztán szintaktikai módszerekkel nem lehetne eldönteni. A módszer egyszerű: a korpuszokban gyakran együtt előforduló minták szemantikai kapcsolatát használja fel az NP-felismerésben. A 6.3 fejezetben bemutatott szabályalapú felismerést támogatja a korpuszból kinyert világismeret is. A módszer nyelvfüggetlen, korpuszalapú és korpuszvezérelt, és automatikusan, emberi beavatkozás nélkül működik.

A megoldás arra alapszik, hogy a szemantikai adatot vegyünk a korpuszból. Az algoritmusnak meg kellene „értenie" a mondatot, hogy képes legyen felismerni, hogy utolsó két NP valójában egy NP. Azt kellene tudnia, hogy a két NP szorosan összefügg (31. táblázat). Az ötlet az volt, hogy a szemantikai információt magából a korpuszból nyerjük ki. Ezért a 9. fejezetben szereplő korpusz segítségével, a felismert NP-k konjunkcióiból, illetve kollokációiból gyűjtöttem ki néhány típust (32. táblázat).

Korpuszból kigyűjtött minták
NP1 **és** NP2
NP1 **valamint** NP2
NP1 blabla1, NP2 **pedig** blabla2

32. táblázat. Szemantikai kapcsolatok kigyűjtése korpuszból konjunkció alapján

DOI:10.15774/PPKE.ITK.2016.003

Azzal a feltételezéssel éltem, hogy ezen NP-k fejei ugyanabban a doménben vannak, más szóval kapcsolatban állnak. Egy kis szövegminta (75M) már 5500 szópárt adott, amelyek egy kis világismeret alapját (is) képezik. Ezek a párok a következő módon használhatók az NP-felismerő algoritmusban: ha két NP *"és"*-sel kapcsolódik, és a fejeik párok ebben az adatbázisban, akkor feltételezhetjük, hogy egymással kapcsolatosak (33. táblázat), mi több, ők egy NP-t alkotnak.

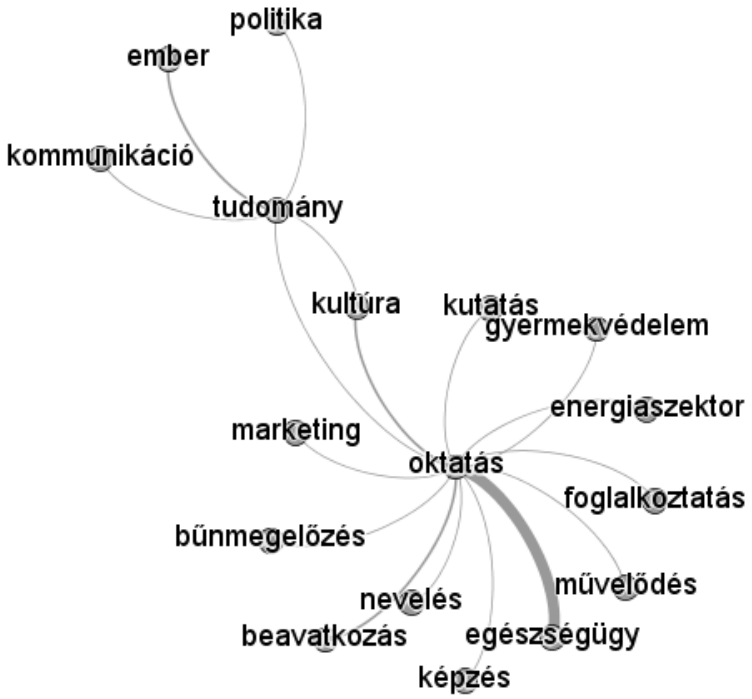
Más szóval a korpusz segít a pontosabb NP-felismerésben. Minél nagyobb a korpusz, annál jobb lesz az NP-felismerés minősége.

IMF	EU
fagylalt	jégkrém
bér	nyugdíj
munka	kenyér
délután	reggel
növekedés	foglalkoztatás
...	...

33. táblázat. Korpuszból kinyert szemantikus kapcsolatok

A korpuszból kinyert kapcsolódó szavak gráfban is megjeleníthetők, a kapcsolat gyakoriságát az él vastagságával jelezve. A gráf egy kis darabját mutatja be a 6. ábra.

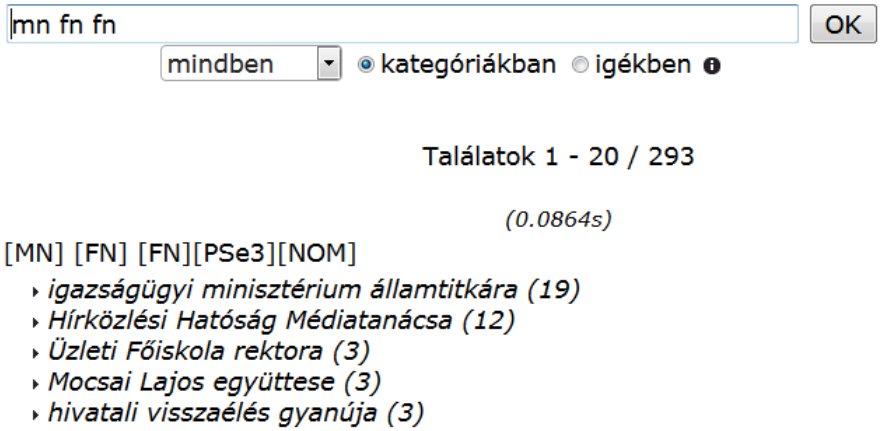
DOI:10.15774/PPKE.ITK.2016.003



6. ábra. Példa a korpuszban található kollokációkra – gráfban

6.5. Mondatvázkereső alkalmazás

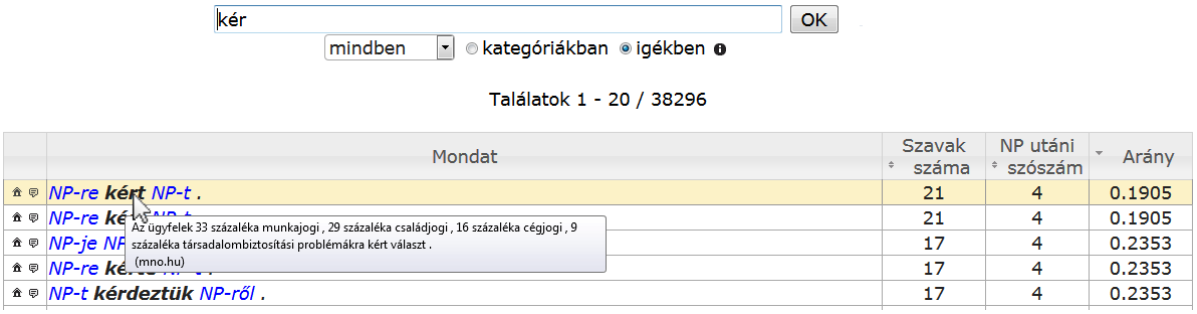
Három korpuszon (inforadio.hu, mno.hu és egy könyv) futott le az NP-kinyerő algoritmus, és egy online webes alkalmazás készült, amely képes ezeket keresni és listázni (7. ábra).



7. ábra. Szófaji címkék alapján NP-struktúra keresés az online felületen

DOI:10.15774/PPKE.ITK.2016.003

Az alkalmazásnak két fő funkciója van. Egyrészt az NP mintákat szófajsorozatok megadásával lehet kilistázni. Például *mn főnév főnév* kilistázza az összes olyan NP-t, amelyben ezen szófajú szavak ebben a sorrendben alkotnak egy NP-t. Másrészt mondatok is listázhatók ige alapján: ez megadja az egyszerűsített mondat mintákat, minden NP helyett egy egyszerűsített "NP" felirattal (8. ábra).



8. ábra. Mondatstuktúrák böngészése az online felületen

Más szóval, lehetséges az is, hogy adott szerkezetű NP-ket keressünk vagy mondatokat listázzunk ki egy adott igével kapcsolatban, NP-mentes formában (8. ábra). Ez utóbbi funkció nagyon hasonlít a Mazsola-rendszerre (Sass 2008; Sass 2011), de ez az eszköz egész mondatot ad, a Mazsola viszont igei keretet mutat.

Az első funkció lehetőséget adott arra, hogy új kategóriákat definiáljunk a Humor elemzőben: az NP-felismerési hibák utaltak a hiányzó vagy hibás szófaji kategóriákra. Például a nevek vagy szakmák szintén főnevek, de eltérő szerepet játszanak a főnévi csoportban. Ez arra inspirált minket, hogy új kategóriákat hozzunk létre (Ligeti-Nagy 2015). Az eszközt más kutatók is fel tudták használni a saját kutatásaikhoz.

DOI:10.15774/PPKE.ITK.2016.003

7. Az annotáció hatása az NP-felismerés teljesítményére

A 6.1. fejezetben bemutatott adatformátum lehetővé teszi, hogy bővítsük vagy akár módosítsuk az egyes szavakhoz rendelt jegyeket, annotációkat. Ebben a fejezetben azt mutatom be, hogy az annotáció módosításával milyen jelentős javulást lehet elérni az NP-felismerés minőségében. Bármilyen külső információ, ami "korrelál" a kívánt kimeneti címkékkel, jól mérhetően segít bármely tanulóalgoritmusnak.

Másként fogalmazva, a kutatásom egyik motivációja az volt, hogy ha sikerül olyan jegyeket találni, amelyekkel "félig" előre annotáljuk az NP-ket, akkor könnyebb lesz a gépi felismerésük. Az adatbányászat terminológiájával élve: a doménismeret megszerzése a kulcs (*acquiring background domain knowledge*). Ebben a fejezetben bemutatok egy módszert a kedvezőbb annotáció módosítására, és egy eszközt, ami ebben segít.

Másik motiváció az annotáló eszköz elkészítéséhez az volt, hogy legyen egy olyan online eszköz, amivel NP szabályokat lehet definiálni és tesztelni. Nekem nem sikerült átütő eredményeket elérni a szabályalapú NP felismerésben (6.3. fejezet), de valaki másnak sikerülhet. Ezért készítettem egy eszközt, ami támogatja a NP szabályok létrehozását, bízva abban, hogy valakinek ez a segítségére lesz.

7.1. A nyelvész intuíciója és a statisztika ereje – az NP-előfeldolgozó eszköz

Az NP-felismerés érdekében végzett jegy (*feature*) -hangolás és -javítás nehéz feladat. A módosítások hatása sokszor nem megjósolható, hanem gyakran csak egy tanítás és tesztelés lefuttatása ad visszajelzést. Így a jegyhangolás sokszor *trial-and-error* stílusban történik.

Ezért egy online eszközt készítettem, amely három célt szolgál: (1) elemzi az NP-felismerés tanítóhalmazát, (2) jegy javaslatokat tesz, amit a felhasználó áttekinthet és elfogadhat, majd végezetül (3) a új, módosított adat exportálható (tanító és teszt adat). Az eszköz egy becsült F-értéket számol a módosítások közben, ezzel egy gyors visszajelzést ad a nyelvésznek.

Az eszközt angol és magyar nyelven tesztelem. Alkalmas arra, hogy előkészítse a tanító és tesztadatokat hatékonyabb NP-felismeréshez. Ezenkívül olyan hasznos szófajokra vonatkozó javaslatokat adhat WordNet alapján, ami jobb F-értéket eredményez.

DOI:10.15774/PPKE.ITK.2016.003

Az eszköznek csak egy böngészőre van szüksége (nincs függősége, és nem kell külön szoftvert sem telepíteni), és könnyen használható nem haladó felhasználóknak is. A fejezetben leírt műveletek (és eredmények) természetesen megvalósíthatóak egy linux konzolban is. De pont az volt az eszköz egyik célja, hogy haladó számítógépes ismeretek nélkül is használható legyen, a jegyek fejlesztése felhasználóbarát módon történjen. Az eszköz maga nem akar okos lenni, csak segíteni azoknak, aki nem ismernek hosszú linux parancsokat, de vannak jó megérzéseik és egy böngészőjük. Az NP preprocess eszköz egyfajta nyelvész játék, amivel ki lehet próbálni dolgokat - olyanok számára is, akik nem konzol guruk, de egy böngészőben szívesen hangolnák az adatot. Az eszköz felhasználóbarát módon szeretné egyesíteni a nyelvész absztrakciós képességét és egy statisztikai motor erejét.

Az NP-felismerés szempontjából kulcsfontosságú a jegykészlet (*feature set*), bármilyen címkézési algoritmust is használunk Ezen az absztrakciós szinten, a jegykészlet független a felismerési módszertől. Ezért a jegykiválasztás a felismerési módszertől függetlenül elvégezhető. A hatása közvetlenül monitorozható az eszközben, majd exportálás után a tényleges felismerési algoritmusban pedig ellenőrizhető.

Címkézési feladatnál az a tipikus kérdés szokott felmerülni, hogy mely jegyek segítenek és melyek nem. Ezen problémák megoldásánál a nyelvész vagy a megérzéseire hagyatkozik (és bizonyosakat módosít), vagy rábízza ezt a műveletet egy statisztikai tanulóalgoritmusra. Számos módszer létezik jegyek kiválasztásra, amelyek különféle típusú jegyek számát hatékonyan tudják csökkenteni. Lényegében ezek a módszerek azt jelzik, hogy a szavak mellé generált sokféle jegy (oszlop) közül melyek segítenek, és melyek nem. A kettő között nem volt átmenet. Olyan jegy-hangolás vagy ennek támogatása tűnt hasznosnak, ami egyrészről nem egy teljes oszlop automatikus eldobását/megtartását jelenti, másrészről a nyelvész irányíthatja a hangolási folyamatot. Harmadrészt nem is teljesen kézzel kell az NP-felismerés szempontjából pontosabb jegy-készletet definiálni. Ezt a rést igyekszik ez az eszköz bepótolni.

Az elérhető NP-felismerőknek (Koeling 2000; Osborne 2000; Brants 2000) van egy közös pontja ebből a szempontból. Mindegyik fekete dobozként működik, belső folyamataik nem transzparensek, illetve nem irányíthatóak közvetlenül. Azonban ezek a módszerek könnyen adaptálhatóak bármilyen új adatra vagy nyelvre. Ezzel szemben, a szabályalapú rendszerek (Déjean 2000; Johansson 2000) teljes mértékben a nyelvész irányítása alatt állnak, de ezek a rendszereket kézzel építik, és az adott nyelvre és kódkészletre szabottak (pl. szófaj).

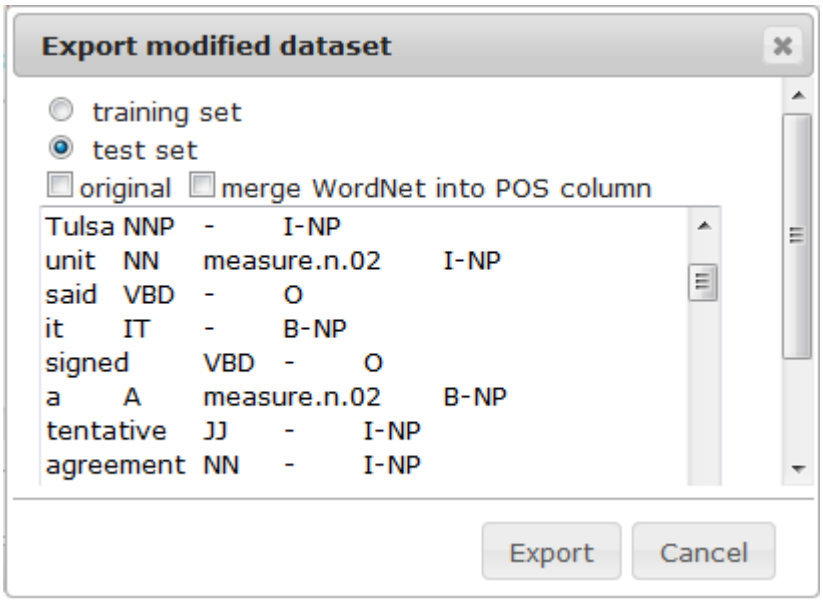
7.1.1. Ötlet

A módszer alapötlete az, hogy kössük össze a nyelvész irányítását és intuícióját a statisztikai motor erejével. A tanítóhalmaz hasznos információkat tartalmaz az egyes jegyek és a kimeneti címkék között, amely kapcsolatok kinyerhetőek statisztikai módszerekkel. Például egy ember

DOI:10.15774/PPKE.ITK.2016.003

nehezen találna olyan új jegyet, ami eltérően viselkedik az IOB-címkék szempontjából. Pedig az ilyen címkéket érdemes alosztályokra bontani.

A célunk az, hogy felderítsük és megmutassuk a IOB-címkézés szempontjából legjobb jegyeket, és megkeressük azokat, amelyek hangolásra szorulnak. A folyamat végén a tanító és teszthalmaz exportálható (9. ábra) és bármely NP-felismerőben használható. A módszert a szófajjegy hangolásával mutatom be, azonban más jegyek esetén is használható.



9. ábra. A módosított tanító és teszthalmaz exportáló dialógusa

Statisztikai vagy szabályalapú felismerők is jobban teljesítenek olyan jegyekkel, amelyek gyakrabban fordulnak elő ugyanazzal az IOB-címkével. Például a névelő leggyakrabban NP elején szerepel (IOB-címkéje legtöbbször: *B*). Ebből kiindulva minél több ilyen jegyet szeretnénk.

Olyan szófajokat (feature) keresünk, amelyeknek egy alosztálya rendre ugyanazt az IOB-címkét kapja, azaz ha saját alszófajt kapna, akkor az sokkal jobban korrelálna egy IOB-címkével. Ezt a felosztást olyan jegy esetén lehet elvégezni, amely (1) kevés értéket vehet fel, és (2) egy alosztálya jobban korrelálna egy IOB-címkével. Más szavakkal, ha egy jegyhez rendelt összes előforduló IOB-címke típusa csökkenthető (például 4 féle címke helyett csak 3) egy aljegy definiálásával, akkor érdemes létrehozni azt az aljegyet. Például, tegyük fel, hogy a főnévnek azonos valószínűséggel 3 IOB-címkéje lehet: *B, I, E*, de egy alosztálya a legtöbbször *B* címkét szokott kapni ("*Mr.*" szó), akkor a MISTER alszófaj létrehozása javítana a rendszer teljesítményén. Ezek a feltételek teljesülnek a szófajoknál, nem csak angol, de magyar nyelven is. A létrehozott új alszófajok a régi szófajjegy helyett állnak.

DOI:10.15774/PPKE.ITK.2016.003

A fent leírt módszert az eszköz úgy valósítja meg, hogy megvizsgálja a tanítóhalmazt, és javaslatokat tesz azokra a szófajokra, amelyekhez többféle IOB-címke tartozik. Természetesen ez lehet normális (egy főnév állhat az NP többféle pozícióján), de ez jelezheti azt is, hogy ezt az adott szófajt több alosztályra osztva jobb IOB-címkézés érhető el. Például, egy szófaj egy adott szava eltérően viselkedik, mint a többi (a címkézés szempontjából), következésképpen a szónak definiálni kellene egy új alosztályt. Ezek a javaslatok fontosak, és az eszköz fő funkciója, hogy ezekre vadászik.

7.1.2. Az eszköz

Egy online eszközt[9] fejlesztettem, amely NP-felismerés tanító- és tesztadatát tartalmazza. Pillanatnyilag a CoNLL2000 adat (Tjong Kim Sang és Buchholz 2000) van importálva, amelynek mérete 259 000 token. A CoNLL2000 adat minden sorban a következő értékhármasokat tartalmazza: szó, szófaj, IOB-címke. Bár az IOB-címkét önmagában is lehet sikeresen tovább hangolni (Shen és Sarkar 2005; Tjong Kim Sang és Veenstra 1999), a következőkben a szófajcímkék hangolásával mutatom be az eszköz működést. Összegezve, ez az eszköz egyszerre csak egy jegyet optimalizál, azonban bármely jegyre alkalmazható, akár más jegykiválasztó módszerek kimenetére is.

Az eszköz funkciói:

- a legjobb jegy minták keresése
- a többféle kimeneti címkét kapó (többértelmű) minták listázása
- jegyek tallózhatóak az IOB-címkék szempontjából
- új jegyek definiálhatóak, amelyek alkalmaz a tanító- és teszthalmazra
- gyorsíthatja a jegyhangolást
- egy reguláriskifejezés-alapú nyelvtan készül automatikusan az elfogadott jó mintákból, a minőség ellenőrzésére
- nyelvtani szabályok kézzel is hozzáadhatóak
- menet közben hozzávetőleges F-értéket számol (tanítóhalmaz alapján)
- minden javaslat a tanítóhalmazból jön, a teszthalmazt külön kezeli. Azt csak exportnál használja, hogy a végső módosításokat oda is átvezesse
- a módosított tanító- és teszthalmaz exportálható
- az eszköz nyílt forráskódú

[9] a forráskód elérhető: `https://github.com/endredy/onlineChunkerToolkit`

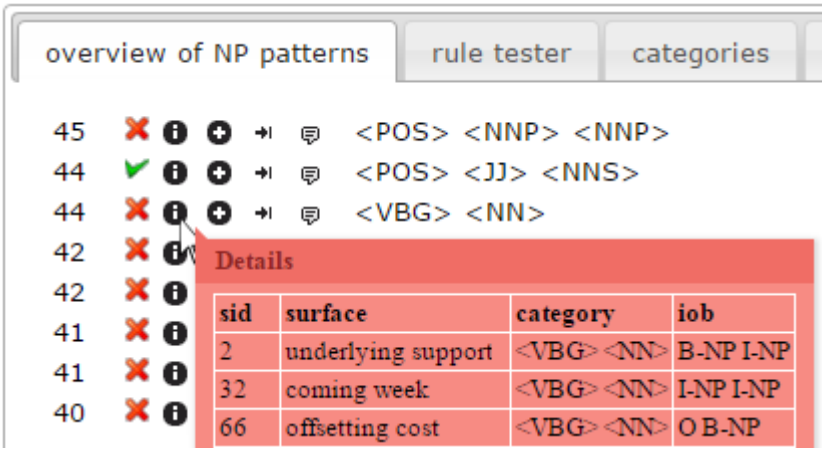
DOI:10.15774/PPKE.ITK.2016.003

7.1.3. Az NP előfeldolgozó eszköz működése

Először, az eszköz átvizsgálja a tanítóhalmazt, a szófajcímkék és az egyes kimeneti címkék kapcsolatát, nem csak szó szinten, hanem szófajcímke-sorozatokat is kiértékel kimeneti címke szempontjából. Másodszor, az NP előfeldolgozó eszköz többféle funkcióval segíti, hogy a felhasználó át tudja tekinteni és módosítani tudja az adatot, hogy az minél inkább a kimeneti címkéknek kedvezzen. Ezt célt a következő áttekintő nézet-funkciókkal támogatja: a szófajcímkék tallózása (rendezhető a gyakoriság/hozzárendelt IOB-címkék száma/hasznosság a IOB-címkézés szempontjából), minden érvényes NP-sorozat listázása, és azon szófajjavaslatok listázása, amelyek jobban korrelálnának az IOBcímkékkel (WordNet-ből generálva, bővebben a 7.1.4. fejezetben). Ha a felhasználó elfogad egy javaslatot (vagy újat hoz létre), akkor az eszköz az adathalmazt automatikusan újra átvizsgálja a módosított szófajjal együtt, és frissíti a nézeteket: a folyamat kezdődik elölről.

Ha egy jegysorozat (esetünkben szófajsorozat) mindig ugyanazt a IOB-címkesorozatot kapja a tanítóhalmazból, akkor ezt a mintát zöld színnel jelöli a program, és a minta bekerül a szabályok közé. A többi minta piros színű, ezekhez többféle IOB-sorozat is tartozik. (10. ábra)

Azonban fontos megjegyezni, hogy minden zöld minta elfogadása *overfitting*-et és alacsony fedést okozna. Kipróbáltam a 6.3 fejezetben bemutatott szabályalapú rendszernél azt, hogy nagy korpuszokon futtatva minden zölddel jelölt címkesorozatból (zöld NP-mintából) automatikusan szabály született. A próbálkozás negatív eredményt hozott: az így generált 107.000 szabály nem volt hatékonyabb, mint a kézzel írt 40. Ezért fontos, hogy ezeket a döntéseket a nyelvész hozza, a program csak előkészít és javasol. Ha a felhasználó csak általános eseteket fogad el, az *overfitting* elkerülhető.

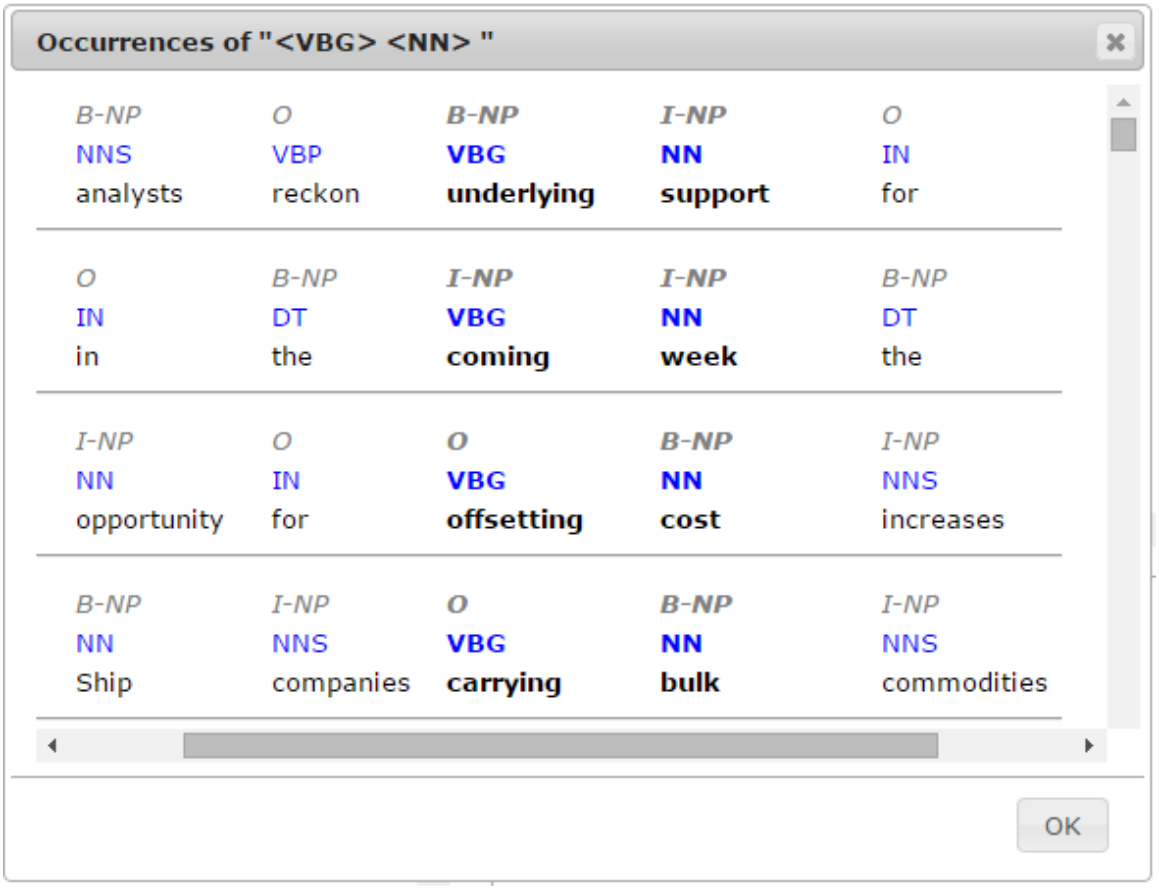


10. ábra. Az IOB-címkék szempontjából értékelt NP-minták – zöld és piros színnel jelölve

DOI:10.15774/PPKE.ITK.2016.003

Pillanatnyilag a szabályokat csak arra használja a program, hogy egy becsült F-értéket lehessen számolni, a szabályok nem játszanak szerepet a végül kiexportált adatban. A szabályok reguláris kifejezésekből állnak, hasonlóan a Natural Language Toolkit (NLTK) grammar szabályaihoz (Bird 2006). A különbség annyi, hogy a mi szabályaink egy böngészőben Javascriptben futnak, nem egy önálló Python programban, és az eredmények azonnal láthatóak. A szintaxis hasonló, de a grammarmotort újraírtam, két célból is. Elsősorban azért, hogy gyors visszajelzést tudjon adni a legutóbbi módosításokról: jobb állapotba vitték-e az adatkészletet vagy sem. Másodszor, hogy a szabályok kézzel is hangolhatóak legyenek. A piros mintákat érdemes áttekinteni az IOB-címkék szempontjából, és ha a minta elfogadhatónak tűnik, akkor egy kattintással betehető a szabályok közé.

Ezen túl, a jegyek áttekinthetőek az IOB-címkék szempontjából: minden egyes jegy mutatja a hozzá rendelt IOB-címkéket (a tanítóhalmazból). Ha egy jegynek csak egy címkéje van, az a legjobb eset. (Statisztikai NP-felismerők ezt könnyen megtanulják.) Ha több, akkor az összes tanítómintabeli előfordulása áttekinthető (11. ábra).



11. ábra. Egy POS-minta összes előfordulásának áttekintése (kontextussal) az IOB-címkéik szempontjából

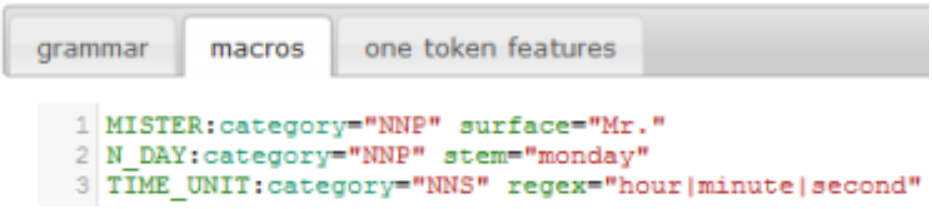
Ezen a ponton általában szokott lenni néhány olyan szó, amelyik másképp viselkedik, mint a szófajának többi szava. Például a CoNLL2000 adatkészletben a *Mr*. szófaja NPP, de eltérően viselkedik, mint a többi tulajdonnév: *Mr*. szeret az NP elején állni. (Más szavakkal:

DOI:10.15774/PPKE.ITK.2016.003

legtöbbször a *B* címkét kapja.) A program segít abban, hogy az ilyen típusú szavakat megtaláljuk, amelyek többnyire (például 80%-ban) ugyanazt a címkét kapják. Másik példa lehet a RB szófajú szavak (*now, ago, also,* stb.) között az *only*, mert ez az esetek nagy részében *B-NP* címkét kap, míg a többi kizárólag *O* címkét. Ehhez hasonló a *that* (szófaja IN), amelyik - szemben a szófajába tartozó szavak nagy részével - nem *O*, hanem többnyire *B-NP* címkét kap. Ezeket az eltérően viselkedő szavakat az eszköz megmutatja a felhasználónak, aki egy kattintással alosztályokra oszthatja az adott szó szófaját.

Az eszköz kulcsa egy úgynevezett makrószolgáltatás. Ez teszi lehetővé, hogy új alosztályokat definiáljuk a létezők fölé. Ennek segítségével új jegyet adhatunk a régi helyett egy adott szófaj, tő, felszíni alak vagy reguláris kifejezés segítségével. (Ezen makró elve egyezik a 6.3 fejezetben bemutatott makróval.) Például a *Mr*. szó esetében az NNP helyett MISTER nevű jegyet (szófajt) kap, ezáltal külön kezelhető a többi NNP-től. Ezen a módon a speciális (eltérő viselkedésű) szó adott jegyét egy új, most létrehozottra cseréljük (néhány példa és működése látható a 34. táblázatban).

Amikor a jegyeket áttekintjük, egyetlen kattintással tudunk új makrót definiálni (12. ábra). A program automatikusan elkészíti a makrót, és lehetőséget biztosít, hogy módosítsuk, ha szükséges. A makrók lehetővé teszik nagyobb kifejezőerejű nyelvtan definiálását, nem csak jegyek (például szófajok), de felszínialak- vagy tőalapú makrók is használhatók a nyelvtanban.



12. ábra. Példa új makró definiálására: szófaj, tő, felszíni alak vagy reguláris kifejezés segítségével

makró	jelentése
MISTER:category="NNP" surface="Mr."	ha egy szónak a felszíni alakja *Mr*. és a szófaja *NNP*, akkor az új szófajcímke legyen *MISTER*
N_DAY:category="NNP" stem="monday"	ha egy szónak a töve monday és a szófaja *NNP*, akkor az új szófajcímke legyen *N_DAY*
TIME_UNIT:category="NNS" regex="hour|minute|second"	ha egy szónak a töve illeszkedik a *hour|minute|second* reguláris kifejezésre és a szófaja *NNS*, akkor az új szófajcímke legyen *TIME_UNIT*

34. táblázat. Példák a makrókra és jelentésükre

Amikor új makrót alkalmazunk (egy gomb megnyomásával), akkor a program átkonvertálja a tanítóhalmazt az új makrókkal, és a legjobb jegy minták kilistázása ezen új

DOI:10.15774/PPKE.ITK.2016.003

makrók felhasználásával történik. Ha ekkor az F-érték nem nő, akkor a hozzáadott makrók nem hasznosak. El kell őket vetni, és helyettük másokat kell hozzáadni. A kiértékelés csak pár másodperc, sokkal gyorsabb, mint egy tanulóalgoritmus, például CRFsuite (Lafferty, McCallum, és Pereira 2001; Okazaki 2007) A CRFsuite a Conditional Random Fields (CRFs) gyors implementációja, egy címkéző (*sequential tagging*) tanulóalgoritmus.

Az eszköz által megjelenített F-érték egy hozzávetőleges érték, mert a tanítóhalmazon számolja ki, egyszerű NP-nyelvtan alapján. Azért nem a teszthalmazon méri, mert így a hibaelemzés és jegyhangolás a teszthalmazon történne, ami nem kívánatos (*development set* lenne a teszthalmaz). Másrészt a NP-nyelvtan alapján mért F-érték sokkal alacsonyabb (az én méréseimben), mint egy tanulóalgoritmusé, így inkább csak alsó becslésnek tekintettem.

A szabályokat automatikusan építi a program a felhasználó makrói és egyéb döntései alapján. A szabályokat az egyes mondatokra alkalmazva kijelölik az NP-ket, amit a gold standardhez mér. A helyes NP chunk *true positive* találtot jelent, hibás *false positive*, hiányzó pedig *false negative*. Az F-mérték szokás szerint azt mutatja, hogy a szabályok mennyire találják el az NP-ket. Emellett jelen esetben azt is kifejezi, hogy mennyire sikerült IOB-barát annotációt kialakítani, ami megkönnyíti az NP IOB címkék kiosztását. Másként fogalmazva, az egyes jegyek mennyire korrelálnak az IOB-címkékkel. A makrók és az egyéb műveletek abba az irányba mozdítják az adatot, hogy az egyes jegyek minél inkább egyértelműen kötődjenek egy adott IOB-címkéhez (amennyiben lehetséges). Más szavakkal, az F azt fejezi ki jelen esetben, hogy mennyire „IOB-barát” az adat.

7.1.4. WordNet segítségével felfedezett új jegyek

Az előzőekben létező jegyek finomabb felosztásával kaptunk új alosztályokat, amelyek jobban korrelálnak az IOB-címkékkel. A most bemutatásra kerülő módszer új jegyeket javasol gépi támogatással, a WordNet synset-ek segítségével (Miller 1995).

Az alapötlet az volt, hogy biztosan létezik olyan – szófajok és egyéb jegyek felett álló – világismeretre épülő tulajdonság, ami alapján az ember könnyedén eligazodik az NP-szerkezetekben. A következőképpen gondolkodtam: ha számomra ismeretlen nyelvű szöveget kellene elemeznem, akkor milyen információ segítene? Az nem lenne elég, ha az egyes szavak szófaját megmutatnák. Szükség lenne az egyes szavak jelentésére is, vagy minimum arra, hogy az egyik szó egy keresztnév, másik egy betegség stb.

Ha ezeket a tulajdonságokat meghatározhatnánk, és a szavakhoz rendelnénk, akkor ezt egy NP-chunker is meg tudná tanulni. Például ha egy szóról nem csak azt tudnánk, hogy *főnév*, hanem azt is, hogy *ember*, *állat*, *tárgy* vagy egy *társadalmi szokás*, akkor a

DOI:10.15774/PPKE.ITK.2016.003

tanulóalgoritmusnak jobb esélyei lennének az NP-felismeréséhez. Le kell szögezni, hogy az emberi elemzés pontosságát és módját nem tudja utánozni a most leírt módszer, de új szempontokat tud adni. A keresendő tulajdonságok egy része megtalálható a WordNet adatbázisban, így erre esett a választásom.

Elsőként a tanítóhalmaz minden szavához kigyűjtöttem a synseteket és a hipernímákat. Majd ezeket összevetettem a szavaikhoz rendelt IOB-címkékkel. Végül az IOB-címkék száma és gyakorisága alapján listáztam ki a synseteket. Azok a synsetek a leghasznosabbak jelen esetben, amelyeknek a kimeneten csak egy IOB-címkéje van, vagy kettő, de azokból az egyik a dominánsabb. Ha ezeket a synseteket, mint új tulajdonságokat (jegyeket) hozzáadjuk az adathoz, akkor ez javítja az NP-felismerés minőségét.

A fejezetben bemutatott eszköz elvégzi a fenti műveletet, csupán a synseteket kell külön oszlopban megadni a tanító- és teszthalmazban. (Az egyes szavakhoz kigenerált több synset / jellel van elválasztva.) Ez alapján a program elvégzi az IOB-címkékkel való összevetést, és az összes synset kilistázható IOB-címkegyakoriság szerint (13. ábra).

Ha a felhasználó bármelyik synsetet alkalmasnak találja, akkor egy kattintással hozzáadhatja a szófajhoz: így ezek – ebben az esetben – az eredeti szófaj helyére kerülnek be, a tanító- és teszthalmazban egyaránt. Például a *period* szó (és szinonimái) esetén a *period.n.07* synset használható az eredeti szófaj (főnév) helyett, mert az előbbi jobban korrelál az IOB-címkékkel, mint az utóbbi. Néhány hasonló javaslatot mutat a 13. ábra.

category	sum	label	freq	%
period.n.07	207	I-NP	197	95.17%
		B-NP	10	4.83%
side.n.10	84	I-NP	80	95.24%
		B-NP	4	4.76%
large_indefinite_quantity.n.01	1654	I-NP	1577	95.34%
		B-NP	77	4.66%
about.s.01	258	B-NP	246	95.35%
		I-NP	12	4.65%

13. ábra. WordNetből generált lehetséges új tulajdonságok áttekintése

DOI:10.15774/PPKE.ITK.2016.003

7.2. Eredmények

A CoNLL2000 adatot importáltam az eszközbe. Mivel az NP-felismerésre fókuszáltam, az egyéb típusú címkéket (VP, PP stb.) "O"-ra cseréltem. A kiértékelés során először a szófajokat néztem át az IOB-címkék szempontjából. Majd makrókat hoztam létre azokra a szavakra, amelyeknek azonos a szófajuk, de eltérően viselkednek (34. táblázat). A legjobb WordNet-javaslatokat is hozzáadtam, végezetül a tanító- és teszthalmaz exportáltam, és több chunkerben kipróbáltam.

Mindezen lépéseket az eszközben végeztem el, és csupán a szófaj-információt módosítottam. Az eredményeket az eredeti és a módosított adatot mértem le. A baseline mérést az NLTK unigram és bigram chunkere jelentette. Ezen felül a HunTag3 (Endrédy és Indig 2015) chunkerrel is végeztem mérést.

Módszer	**F-érték**
NLTK – unigram chunker	
eredeti adaton	83,2%
A program által módosított adaton	**83,8%**
NLTK – bigram chunker	
eredeti adaton	84,5%
A program által módosított adaton	**86,1%**
HunTag3	
eredeti adaton	92,68%
A program által módosított adaton	**92,74%**
Szavazás több IOB reprezentáció között	
eredeti adaton	92,74%
A program által módosított adaton	**94,12%**
Szavazás több IOB reprezentáció között + HunTag3	
eredeti adaton	93,13%
A program által módosított adaton	**94,59%**

35. táblázat. Az NP-előfeldolgozóval módosított adat több chunkerrel mért eredményei

Az eredmények azt mutatják, hogy az előfeldolgozó eszköz minden chunker eredményén tudott javítani (35. táblázat). A jelenlegi state-of-the-art NP felismerő, későbbiekben SS05 (Shen és Sarkar 2005) 95,23%-on teljesít a CoNLL2000 adaton. Ez az eredmény több IOB-reprezentációval (IOB1, IOB2, IOE1, IOE2, O+C) tanított architektúrán alapszik, amely taggeléskor az egyes reprezentációk eredményeit szavaztatja. Az IOB-címkéket a szófajjal is kiegészíti. Ez a megoldás hasonlít a fejezetben bemutatott megközelítéshez, de az nem a szófajokat, hanem az IOB-címkéket módosítja. Az SS05 megoldását újraimplementáltam Pythonban, de az ő eredményüket nem tudtam reprodukálni. Kapcsolatba léptem a szerzőkkel, és Anoop Sarkar professzor úr rendelkezésemre bocsátotta az eredeti Perl kódjukat. Ez sokat segített az algoritmus jobb megértésében, ám ezzel sem sikerült reprodukálni az eredményüket. Azonban az alapötlet (több IOB reprezentáció szavaztatása) már önmagában

DOI:10.15774/PPKE.ITK.2016.003

sokat segített (35. táblázat), az eredeti kód pedig új lendületet adott a kutatásnak (Indig és Endrédy 2016).

Az újraimplementálás magában foglalta az IOB-címkereprezentációk közötti konverziót, az IOB-címkék módosítását szófaj információval. Majd ennek tanítását és taggelését az NLTK csomag T'n'T taggerérel valósítottam meg (Bird, Klein, és Loper 2009). Az eredményeket közös IOB-címkekészletre konvertáltam, majd szavaztattam. A HunTag3 eredményét is hozzátettem, mint újabb szavazásban résztvevő 6. modul (36. táblázat). Ez további +1,4%-os javítást jelentett.

Még ha az eredmények alacsonyabbak is, mint az SS05 esetében, az NP-előfeldolgozó hatékonyságát alátámasztják a mérések.

formátum	Eredeti szófajjal	Módosított szófajokkal
IOB1	92,01%	93,57%
IOB2	90,71%	92,04%
IOE1	90,64%	92,18%
IOE2	88,67%	89,96%
O+C	90,52%	91,71%
szavazás után	92,74%	94,12%
szavazás után, HunTag3-mal	93,13%	**94,59%**

36. táblázat. Az NP-előfeldolgozó többféle IOB-reprezentáció közötti szavazással elért eredményei

7.3. Kapcsolódó kutatások

A chunkerek teljesítményét alapvetően kétféleképpen lehet javítani: szoftver- vagy adatfejlesztésekkel. Számos kutatás készít, javít vagy kombinál chunkereket, amelyek jobban teljesítenek ugyanazon az adaton (Koeling 2000; Osborne 2000; Sun és mtsai. 2008). A másik megközelítés olyan jegyek és/vagy címkék megtalálása, definiálása vagy kombinálása, amelyek jobb F-értéket eredményeznek az adott felismerési feladatnál (Shen és Sarkar 2005; Simon 2013). A F-értéket tipikusan egy gold standardhez viszonyítva szokták meghatározni: a helyesen felismert egységek jelentenek *true positive* (*tp*) találatot, a helytelenek *false positive* (*fp*) és a hiányzóak *false negative* (*fn*). Ez alapján számolják az F-értéket (képlete a Glosszáriumban található).

A specializáció szintén hatékony módszer. Ez azt jelenti, hogy az IOB-címkéket részletesebb osztályokra bontják (Molina és Pla 2002; Shen és Sarkar 2005). Ebben a megközelítésben bizonyos küszöbérték feletti gyakorisággal rendelkező szavak IOB-címkéit szófaj-információval egészítik ki. Például, a következő (szó, szófaj, címke) értékhármas *(te,*

DOI:10.15774/PPKE.ITK.2016.003

NM, B-NP) részletesebb IOB-címkét kap: *(te, NM, NM-B-NP)*. Jelen megoldás hasonló, azonban én a szófajt bontottam részletesebb alosztályokra, nem az IOB-címkéket.

A jegy hangolásra tipikus megoldás a *trial-and-error* módszer, ahol a nyelvész új jegyet ad az adathoz (új oszlopot ad a szavakhoz), és kipróbálja ezeket illetve kombinációikat. Ez sok tanító/tesztelő fázist, és több hét iterációs időt jelenthet. A fejezetben bemutatott módszer az adatjavításhoz kapcsolódik, és csupán a taggelés előtti adat-előfeldolgozásban szeretne segítséget nyújtani.

DOI:10.15774/PPKE.ITK.2016.003

7.4. Összefoglalás

Készítettem egy online eszközt, amivel kereshetőek a korpuszok szerkezete. Főnévi csoportok szófaji címke alapján, mondatvázak ige alapján listázhatók ki. Az eszköz az általam letöltött korpuszokban keres, és mindenki számára elérhető.

A főnévi csoportok hatékonyabb felismeréséhez egy olyan előfeldolgozó online eszközt készítettem, amely a tanító- és tesztadat módosításával 0,8-2% javulás érhető el. Az eszköz működését a szófajokban végzett pontosításokkal demonstráltam. A WordNet alapján olyan jegy-javaslatokat is felkínál a program, amelyek az emberi elemzéshez hasonlóan a világismeretet használják fel a főnévi csoportok meghatározásához. Méréseim alapján a jobb annotáció fontosabb lehet, mint maga a tanulóalgoritmus. Ha olyan annotációt használunk, ami "korrelál" a kívánt kimeneti címkékkel, az minden vizsgált tanulóalgoritmusnak segített.

Kapcsolódó tézisek:

5. tézis: Kimutattam, hogy az NP-felismeréshez használt jegykészlet tovább javítható, ha (i) a jegyek kevés értéket vehetnek fel, és (ii) ha létezik olyan finomabb osztályozás, amely mellett a kimeneti címkékkel jobban korreláló jegyeket kapunk.

5.a tézis: Az angol nyelv esetében a szófaj kategóriát olyan alkategóriákra bontottam, amelyek jobban korrelálnak az IOB-címkékkel (+2% javulás).

5.b tézis: Létrehoztam egy módszert, amivel az 5.a tézisben leírt művelet a WordNet synsetjeinek segítségével gépi támogatással elvégezhető.

5.c tézis: Becslést adtam egy jegyhalmaz hasznosságára a címke- és a jegykorreláció kiszámolásával, méghozzá egy tanulóalgoritmus (NLTK unigram, bigram chunker; HunTag; SS05) futási idejénél gyorsabban.

5.d tézis: Kimutattam, hogy nagyon fontos az input annotáció szerepe, ami akár fontosabb lehet, mint maga a gépi tanuló algoritmus. Bármilyen külső információ segít, ami „korrelál” a leendő kimeneti annotációval.

5.e tézis: Kimutattam, hogy kevés olyan szószintű jegy van, amely a kimeneti címkével 100%-osan korrelál, viszont ezek mindegyike fontos.

Kapcsolódó publikációk: [5], [6]

DOI:10.15774/PPKE.ITK.2016.003

8. Statisztikai alapú NP-felismerés – HunTag3

A HunTag eszköz (Recski 2014; Recski és Varga 2012) egy rejtett Markov-modell-alapú, liblinear osztályozóval működő általános nyílt forráskódú tagger, amely NER, POS-tagging, illetve NP chunking feladatokra is használható. (A liblinear egy lineáris osztályozásra készített könyvtár, amely jegyek millióit képes kezelni.) A magyar NP-felismerés területén ez az eszköz rendelkezett a legjobb eredménnyel, az itt leírt módszerekkel ezen sikerült tovább javítani.

Elkértem azokat a tanító- és tesztadatokat, amelyeken az eddig megjelent HunTag cikkek mérései alapultak. Magyar nyelvre alapvetően három szófaji kódrendszert használnak. Mindegyik a Szeged TreeBank (Csendes és mtsai. 2005) mondatait használja, amely magyar nyelven a legnagyobb, NP-t jelölő, kézzel annotált korpusz. Az eddigi mérések alapvetően a tanító/teszt halmazok felosztásában illetve a használt szófaji kódokban különböztek. A Szeged TreeBank az úgynevezett MSD (*morphosyntactic descriptions*) kódot használja (Erjavec 2010), a cikkek pedig többnyire KR-kódokat (Kornai és mtsai. 2004).

Az adatok formátuma megegyezik a nemzetközileg is használt korpuszokéval: a mondat minden szava új sorban van, tabulátorral felsorolt tulajdonságokkal, jegyekkel (ún. feature-ök), majd a legvégén a gold standard NP annotáció (14. ábra). Ehhez az adathoz az NP-felismerő hozzárendel egy címkét (innen a *sequential tagger*, sorozatcímkéző elnevezés). A címkék többfélék lehetnek. Az alapváltozatban 3 címke létezik: "B" (a NP eleje), "I" (NP-n belül) és "O" (NP-n kívül). Ezt IOB-címkének nevezik. A címkékről részletesebb leírás a 6.1. fejezetben található. A HunTag megtanulja (*training*) az egyes jegyekhez, jegysorozatokhoz rendelt legvalószínűbb kimeneti címkét, és ezt alkalmazza (*tagging*) új adatokra is.

```
Tulsa      NNP  -              0   5   5   I-NP
unit       NN   measure.n.02   0   4   6   I-NP
said       VBD  interact.v.01  0   4   7   B-VP
it         PRP  content.n.05   1   2   8   B-NP
signed     VBD  contract.v.01  0   6   9   B-VP
a          DT   -              1   1   10  B-NP
tentative  JJ   -              0   9   11  I-NP
agreement  NN   statement.n.01 0   9   12  I-NP
```

14. ábra: Példa tanítómintára (word, pos, wordnet synset, stopword bit, hossz, mondatbeli pozíció, IOB-címke)

DOI:10.15774/PPKE.ITK.2016.003

A felismerés minőségét nagyban befolyásolják a mondat szavai mellett felsorolt jegyek. A HunTag erőssége, hogy új jegyek Python függvény formájában is definiálhatóak, a szó tetszőleges környezetének figyelembevételével. Ez nagy kifejezőerőt ad, és adott hibák kiküszöbölhetők vele.

8.1.1. Új jegyek definiálása

A jegyek meghatározása kulcsfontosságú a felismerés minőségében. Egy jegy alapvetően két esetben segíti az NP-felismerési feladatot. A legjobb eset, ha mindig ugyanazt a IOB-címkét vonja maga után. Ilyen jegy kevés van, de ezeket tanulja meg legkönnyebben a statisztikai motor, ezek a felismerés fix pontjai. Klasszikus példa rá az ige szófaj vagy bizonyos írásjelek: ezek biztosan "O" címkét kapnak.

Az (IOB-címke szempontjából) egyértelmű szószintű jegyekből kevés van, de minél több ilyet fel kell deríteni, és belevenni a folyamatba. Ehhez készítettem egy eszközt, ami segít hatékonyan megkeresni ezeket (7. fejezet). Többnyire azonban egy szószintű jegy önmagában nem tudja meghatározni a kimeneti címkét. Szükség van a környezetére is. Elég arra gondolni, hogy még egy ember sem tudja eldönteni egyetlen kiragadott főnévről, hogy az egy NP elején, belsejében vagy végén található. Mindenképpen látni kell hozzá a kontextust.

Az ötlet lényege az, hogy az egyes jegyeket aszerint osztályozzuk, hogy hányféle kimeneti címkét rendel hozzá a tanítóhalmaz. Ezt megteszi a NP preprocess eszköz (Endrédy 2015b) vagy a HunTag3 (Endrédy és Indig 2015) *most informative feature* opciója. A *most informative features* a HunTag3 olyan értékelő opciója, ami a tanuláskor megkapott jegyeket képes kilistázni egyfajta hasznossági sorrendben: feltünteti, hogy az adott jegy hányféle kimeneti címkét vesz fel, és milyen arányban. (Itt is hangsúlyozom, hogy a HunTag3-at - és ezt a funkcióját is - Indig Balázs készítette.) Felmerülhet a kérdés, hogy mi a különbség a HunTag3 ezen funckiója és az NP preprocess eszköz WordNetes funkciója (7.1.4. fejezet) között? A két funkció hasonló: mindkét esetben előáll egy lista, amiből válogathatunk. A különbség felhasználói szempontból van: a HunTag3 esetében egy linux konzolban vagyunk, és a listát itt tudjuk átnézni (kézzel vagy esetemben - a 8.1.2. fejezetben tárgyalt módon - egy küszöb felett elfogadva az összeset). Ennek megvan az előnye, és egy haladó felhasználónak nem is kell ennél több. Ezzel szemben az NP preprocess eszköz esetében végig egy böngészőben vagyunk, grafikus felületen tekinthető át egy-egy jegy előfordulása, egy kattintással definiálható új jegy, és a döntés hatását is pár másodperc múltán láthatjuk: a legutóbbi ötlet milyen hatással volt az F-mértékre. Vannak konzol-pártiak, akik jelezték: minden, az NP preprocess eszközben megvalósított funkció elvégezhető konzolban is. Igazuk van. A különbség két funkció között a felhasználói felületben van.

DOI:10.15774/PPKE.ITK.2016.003

Számunkra most azok a jegyek az érdekesek, amelyek bizonyos arány (például 95%) felett egyetlen IOB-címkét kapnak. Ezek lesznek az erős jegyek, a felismerés fix pontjai, amelyek nem csak a saját kimeneti címkéjük meghatározásában, hanem a szerkezetek felismerésében is segítséget jelenthetnek. Például a NP preprocess eszköz (7. fejezet) megmutatta, hogy az "is" szófaja (határozószó) túl általános az NP-felismerés szempontjából, és ez a szó jobban kezelhető lenne, ha egy saját jegyet kapna.

A HunTag hibaelemzésekor látható volt, hogy az egyik eset, amikor javítani lehet rajta az, amikor a birtokos és birtok gyakran külön NP-be kerül (pl. *Az életem talán egyik legérdekesebb napja* NP-t tévesen két NP-re bontotta). Ezzel rokon hiba, amikor a birtokos szerkezetet tévesen összevonja egy másik NP-vel: *Az aznapi program London város látogatása* esetén egy NP-be vonta, kettő helyett. Másik tipikus tévesztés, hogy két szomszédos, más esetben lévő főnév egy NP-be került. (pl. *osztály karácsonykor*). Az igenevek és a melléknevek kódja azonos az MSD-kódban, viszont ezek eltérően viselkednek a magyar főnévi csoportokban. Ezért az előbbi három hiba kiküszöbölésére külön explicit jegyeket hoztam létre, hogy a HunTag3 jobban megtanulhassa ezen esetek eltérő kezelését. Az ilyen esetekben létrejövő külön jegy erősebben jelzi a HunTag3 számára, hogy hol az NP határa. Konkrétan a következők lesznek ezek a jegyek[10]: ha szomszédos főnevek más esetben vannak, akkor készítek hozzájuk egy *casDiff* jegyet, a birtokos és a birtok jegyeihez hozzáteszek egy-egy *possession* prefixet, az igenevek pedig kaptak egy explicit *MIGE* címkét is. Ezeken túl még az is hasznos tanulási támpontnak bizonyult, ha egy főnév megkapja a megelőző névelőtől (ha van) számított összes szófaj címkét, egy önálló jegyként.

8.1.2. WordNet-javaslatok

A 7.1.4 fejezetben leírt WordNet javaslatokat felhasználó módszert megvalósítottam a HunTag3 esetében is. Létrehoztam egy új oszlopot az adatban, amelybe synseteket tettem a következő módon. Minden egyes szóhoz kigeneráltam a WordNet összes szülő synset-jét (például *kutya* szó esetén: *négylábú, emlős, állat, élőlény, entitás*). Majd megszámoltam, hogy az egyes synset-ekhez hányféle IOB-címke tartozik (példánkban tegyük fel az *állat* és az *élőlény* 90%-ban "E" címkét kap, a többi synsetnél nincs egyértelmű győztes). Ezután a synset-ek IOB-címkeszám és előfordulás szerinti listázása megadja a legjobban használható tulajdonságokat. Ez látható a 37. táblázatban. Ennek értelmezése a következő: a *mister.n.01* synset 785-ször fordult elő a tanítómintában, ebből 767 alkalommal B-NP kimeneti címkét kapott, 18-szor I-NP címkét. Ez százalékban kifejezve: 98% B-NP és 2% I-NP.

[10] casDiff() és possConnect() függvények, bővebben:
`https://github.com/ppke-nlpg/HunTag3/blob/master/features.py`

DOI:10.15774/PPKE.ITK.2016.003

Ha a WordNet synset-eket tartalmazó adatra lefuttatjuk a HunTag3 *most informative features* opciót (8.1.1. fejezet), akkor éppen azt a listát adja ki. A lista azon legjobb elemeit kiválasztottam, amelyek rendre ugyanazt a IOB-címkét kapják 95% fölötti gyakorisággal. Ezeket a jegyeket megtartottam, a többit töröltem. Ezzel csökkent a statisztikai keresési tér (gyorsabb lett a tanulási folyamat), ugyanakkor új jegyekkel bővült a tanulóminta.

Fontos megjegyeznem, hogy itt egy limit feletti összes synsetet megtartottam, míg a 7.1.4. fejezetben szereplő NP preprocess eszközben a felhasználó egyedi döntése alapján kerül be jegyként egy-egy synset. Ez súlyos különbségnek bizonyult: míg az NP preprocess eszközzel módosított adat minden tanulóalgoritmusnak tudott segíteni, addig a jelen megoldás nem: egy ellenpróba (synset jegy oszlopának ki-bekapcsolása) megmutatta, hogy ezek a tömegesen átvett jegyek nem javítják a NP felismerés minőségét.

WordNet synset	gyakoriság	hozzárendelt IOB-címkék	arány
mister.n.01	785	B-NP:767/I-NP:18	B-NP:0.98/I-NP:0.02
nation.n.03	63	I-NP:63	I-NP:1.0
number.n.11	90	I-NP:89/B-NP:1	B-NP:0.01/I-NP:0.98
country.n.04	67	I-NP:66/B-NP:1	B-NP:0.01/I-NP:0.99
period.n.05	84	I-NP:82/B-NP:2	B-NP:0.02/I-NP:0.98
day.n.10	186	I-NP:179/B-NP:7	B-NP:0.04/I-NP:0.96
month.n.02	273	I-NP:264/B-NP:9	B-NP:0.03/I-NP:0.97

37. táblázat. Példák a WordNet-ből generált jegy javaslatokra

8.1.3. Algoritmikus fejlesztések

Elkészült a HunTag3, amely nemcsak a HunTag újraírását jelentette, hanem többféle matematikai modul kipróbálását is magával hozta. Az újraírást Indig Balázs végezte, Python3 alapon, a liblinear osztályozást a *Scikit-learn* (*LinearRegressionClassifier*) váltotta ki, *SciPy sparse* mátrixokkal és *NumPy* tömbökkel. A *LinearRegresionClassifier* egy standard könyvtár, ami a liblinear-ra épít, azonban SVM-eket is használ, lineáris szeparátort csinál *n*-dimenzióban, és ebből jönnek ki a jegyek súlyai. A HunTag3 rendelkezik *CRF suite* támogatással is, ami azt jelenti, hogy a bemeneti adatból, a konfiguráció (kért oszlopok, egyedi python jegy függvények, stb.) beállításaival olyan formátumú kimenetet képes

DOI:10.15774/PPKE.ITK.2016.003

készíteni, ami megfelel a CRFsuite bemenetének. Másként fogalmazva, a HunTag3 képes összeállítani a CRFsuite bemenetét.

Az eddigi megoldások a Viterbi algoritmust (Finkel, Grenager, és Manning 2005; Baldridge 2005) használtak arra, hogy az egyes mondatok legvalószínűbb kimeneti címkéit előállítsák. Ezt (jelenleg) mind bigramalapon tették. Az én részem a HunTag újraírásban az volt, hogy a bigram helyett trigraminformációt használjon a Viterbi algoritmus. Azt tapasztaltam, hogy háromelemű IOB-címkekészlet esetén nem tudott segíteni a trigrammegoldás. Egy adott címke feltételes valószínűségénél az előző címke számít, az azt megelőző nincs ráhatással. (Például ha az előző címke "O" volt, akkor a jelenlegi "B", "O" vagy "I" lehet, függetlenül a kettővel megelőzőtől.) Azonban ha finomabb IOB-címkéket használunk (pl. IOBES), akkor a trigram javítani tudott a Viterbi algoritmuson.

8.1.4. Angol eredmények

Az angol eredményeink összehasonlíthatósága kedvéért a CoNLL2000 adatot (Tjong Kim Sang és Buchholz 2000) használtuk, amin a state-of-the-art NP-felismerőt is mérték (SS05). A HunTag3 jobban teljesített, mint a HunTag, azonban a CRFsuite mindkettőnél jobb eredményt ért el. A cross validáció során mindhárom eredmény szignifikánsan különbözött a t-test szerint. A CRFsuite sem tudta az SS05 eredményét legyőzni (38. táblázat). Az egyes módszerek sorrendje eltér: más angolban, mint magyarban.

módszer	F-érték
T'n'T (nltk)	86,25
HunTag	91,38
Huntag3, CRF	**93,42**
Huntag3, bigram	92,79
Huntag3, trigram	93,41

38. táblázat: NP-felismerési eredmények angol nyelvre (CoNLL-2000 adatra)

A legjobb eredményt angolra a CRFsuite hozta. Ebben a HunTag3 modul is részt vett: létezik egy kapcsolója, ami a CRFsuite bemeneti formátumára konvertálja az adatot. Előtte a HunTag3 az adott oszlop- és jegy-konfigurációval feldolgozta és előkészítette az adatot.

8.1.5. Magyar eredmények

Magyar nyelvre a state-of-the-art NP-felismerő a HunTag (Recski és Varga 2012; Recski 2014), amely a KR-címkekészletet (Kornai és mtsai. 2004) használja, és 90,28% F-értéket ért el. Ez volt a baseline a tesztjeinkben. A magyar tanító- és tesztmintának a Szeged TreeBanket

DOI:10.15774/PPKE.ITK.2016.003

(Csendes és mtsai. 2005) használtuk, amely 326 000 kézzel annotált maximális NP-t tartalmaz.

A felismerők minősége akkor hasonlítható össze igazán, ha ugyanazon a korpuszon, ugyanazzal tanító- és teszthalmaz felosztással és kódkészlettel (KR, MSD, Humor) dolgoznak. Ezért egy korábbi MSD-kód-alapú, másik tanító-tesztadat felosztással (Miháltz 2011) is leteszteltük az összehasonlítás kedvéért (39. táblázat 1. oszlopa).

Minden tesztben az IOBES-címkekészletet és az eredeti chunktípusokat használtuk a hivatkozott mérésekkel összhangban. A tanító halmaz 10%-át fenntartottuk development halmazként, és ezen optimalizáltuk a jegyeket. Trigramokat és simítást használva sikerült a korábbi publikált eredményeknél jobb F-értéket elérni. A legjobb eredményünk 93,59% F-érték volt egy korábban használt tanító/teszt felosztással (Miháltz 2011). A hibaelemzés és a mérések azt mutatták, hogy a legjelentősebb javulást a szomszédos NP-k pontosabb felismerése eredményezte.

A 8.1.1 fejezetben leírt fejlesztések önmagukban 1,7%-ot javítottak. A HunTag3 újraírása önmagában átlagosan 0,19% javulást jelentett (a 39. táblázat 4. sora). Az eredeti T'n'T taggerrel is megmértük, hogy a megoldásunkkal összemérhető legyen. Az angol eredményekkel megegyezően magyarban is a T'n'T-nél jobb volt a HunTag3 eredménye. Mindhárom mérésben – magyar esetén – a legjobb eredményt a HunTag3 trigram használatával és IOBES címkékkel mértük. Az angollal ellentétben magyar esetén a CRFsuite nem hozott jobb eredményt.

	MSD	**KR**	**KR + jobb jegyek**
T'n'T	68,52	70,95	-
baseline	81,71	88,72 *(Recski mérése: 90,28)*	-
HunTag	93,20	88,96	90,78
HunTag3 – bigram	93,43	89.10	90,72
HunTag3 – trigram	93,59	89,83	91,50
CRF	92,27	89,12	89,77

39. táblázat. NP-felismerési eredmények magyar nyelvre, F-érték különféle kódkészletekkel (Szeged Treebank)

DOI:10.15774/PPKE.ITK.2016.003

Tízszeres cross validációt végeztük az eredmények ellenőrzésére. Ez a következő F-értékeket mutatta: HunTag: 86,9±4,5%, HunTag3 CRFsuite-tal: 89,2±3,8%, HunTag3 bigrammal: 90,1±3,7% és HunTag3 trigrammal: 90,3±3,8% (40. táblázat). Az eredményeknek hasonló a trendje, azonban szignifikáns különbség csak a CRFsuite és a HunTag3 bigram között van (*t-test*).

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	**átlag**	**szórás**
HunTag	91,5	88,5	84,7	82,5	87,9	89,0	87,0	90,1	85,1	82,8	86,9	4,5
Huntag3, CRFsuite	92,9	90,9	87,4	85,3	89,4	90,6	90,1	92,0	88,0	86,0	89,3	3,8
Huntag3, bigram	93,8	91,5	88,2	86,4	91,1	91,5	91,3	92,5	88,6	86,5	90,1	3,7
Huntag3, trigram	94,0	91,8	88,4	86,7	90,5	91,7	91,6	92,8	88,8	86,3	90,3	3,8

40. táblázat. Crossvalidáció a magyar eredmények ellenőrzésére

DOI:10.15774/PPKE.ITK.2016.003

8.2. Összefoglalás

A főnévi csoportok felismerése fontos lépés a mondatelemzés során. A legjobb magyar eredményt produkáló HunTag eszközt Indig Balázs kutatótársam továbbfejlesztette és modulárissá tette. Így lehetőségem nyílt több irányt is kipróbálni. Viterbi, bigram- és trigramátmenet, illetve CRFsuite vizsgálata, háromféle kódkészlet (MSD, KR, Humor) kipróbálása során magyar nyelvre a legjobb modellnek a trigramátmenet bizonyult. Majd számos futtatás során kikísérletezett új jegyek hozzáadásával sikerült több, mint 3% javulást elérni. A tesztadaton nem történt hibaelemzés, az eredmények szignifikanciáját pedig crossvalidációval ellenőriztem. A hibaelemzés és a mérések azt mutatták, hogy a legtöbb hiba a szomszédos NP-k felismerésében (hol tévesen egybe, hol tévesen külön), illetve a birtok és a birtokos téves szétválasztásában vagy egy másik NP-vel való téves összevonásában volt. Ezeken történő javítások eredményezték a legjelentősebb javulást.

Angol nyelv esetén a CRFsuite hozta a legjobb eredményt, de ez nem lépte át az eddigi legjobb angol publikált eredményeket.

Kapcsolódó tézisek:

6. tézis: Sikerült javítanom a főnévi csoport felismerésének minőségét új jegyek definiálásával, egy társszerzővel létrehozott trigramátmeneti modellel és a HunTag3 alkalmazásával, amelyek együttesen az eddigi legjobb magyar NP-felismerést eredményezték. (93.59%)

6.a tézis: Kimutattam a főnévicsoport-felismerésnél, hogy a trigram-átmenetvalószínűségi modell csak részletesebb, háromnál több elemű (típusos, vagy finomabb felosztású) IOB címkék esetén tud többet adni, mint a bigram-modell.

Kapcsolódó publikációk: [6]

DOI:10.15774/PPKE.ITK.2016.003

9. Pázmány korpusz

Jelen munka során nagy méretű webkorpusz is készült. A 2. fejezetben bemutatott crawlert 2012-ben kezdtem el építeni, ami évek alatt összegyűjtötte a korpusz alapanyagát. A nagy mennyiségű magyar nyelvű webes szövegen többféle tisztítást, szűrést végeztem. Ebben a fejezetben ezeket a munkálatokat mutatom be, és a végeredményképpen elkészült korpuszt.

9.1. A korpusz építése

A weblapok letöltött HTML tartalomnak csak töredéke, 15-20%-a a használható nyers szöveg. Ezért nagy adatletöltéssel jár a korpuszépítés. Tapasztalataim alapján a gyűjtésnek leglassabb eleme a letöltés. Egyrészt ugyanattól a szervertől kis késleltetéssel illik újabb lapot lekérni, másrészt 15 szálnál többet nem futtattunk. A HTML-tartalmakat is tároltam, így a későbbi procedurális módosításoknál, az utómunkák során az újrafuttatás nagyságrenddel gyorsabban zajlott, hiszen nem igényelt újabb letöltéseket.

A crawler az Aranyásó algoritmus (2.3.2. fejezet) segítségével tisztította meg a szöveget. A letisztított webtartalmakat további tisztításnak és utómunkáknak vetettem alá. A magyar lapok többféle kódlapot használnak: iso-8859-2, utf-8, iso-8859-1, valamint az is gyakori, hogy a magyar ékezetes betűk HTML-entityvel szerepelnek. A szövegeket ezért egységesen utf-8 kódlapra konvertáltam a többféle HTML kódolásból.

Bár a duplikátummentességre már a Aranyásó algoritmus kialakításánál is törekedtem, ám hiába unikus minden letöltött url és a kinyert tartalom md5-hashkódja is, még így is előfordult, hogy ugyanaz a tartalom többször is letöltésre került. Ez egyrészt annak köszönthető, hogy a portálokon a cikkek egy idő után archívumba kerülnek, azaz kapnak egy másik url-t; vagy egy cikk egyszerűen több url-en is elérhető (pl. `belfold.domain.hu/cikk123` és `hirek.domain.hu/cikk123`) másrészt előfordul, hogy a tartalmon valami egészen apró módosítás történik, ami ilyenkor értelemszerűen újnak fog számítani. Ezért duplikátummentesítést végeztem nemcsak url- és bekezdés-, hanem mondatszinten is, utólag, az Aranyásó algoritmuson utáni szövegen is. Minden mondatnak elkészítettem az md5 hashkódját, és ha egy hashkód már szerepelt (adatbázis alapján), akkor azt a mondatot kidobtam, hiszen az már egyszer volt.

DOI:10.15774/PPKE.ITK.2016.003

9.2. Tartalom alapú válogatások - algoritmikusan

9.2.1. Összefüggő szöveg ↔ felsorolások

A korpusz szövegei nagyon eltérő jellegűek voltak. Egyrészről cikkek, blogok, elbeszélések, másrészről termékfelsorolások, címkefelhők, vagy a leggyakrabban keresett kifejezések stb. szerepeltek az anyagban. Más szavakkal: összefüggő szövegek és felsorolások vegyesen fordultak elő. Emiatt szükségesnek tartottam, hogy osztályozzam őket, és lehetőség szerint elkülönítsem ezeket. Az egész korpuszt átszűrtem a következő módon: ha egy weblap bekezdéseinek átlagos hossza és a stopword-arány egy küszöbérték alatt volt, akkor az adott oldal az „egyéb" kategóriába került (41. táblázat). Megtartottam ezt a halmazt is későbbi feldolgozás céljából, bár ezek nem olyan szövegek, amiket „olvasni szoktunk". Ez a felosztás lehetővé tette, hogy a fő korpusz tisztított, jobb minőségű szövegeket tartalmazzon.

alkorpusz	tokenszám	mondatszám	NP-szám
fő korpusz	954 298 454	48 536 849	223 347 534
egyéb tartalmak	228 806 919	15 802 499	52 865 889
kommentek	58 985 126	3 505 818	13 867 066
összesen	**1 242 090 499**	67 845 166	290 080 489

41. táblázat. A Pázmány korpusz összetétele

9.2.2. Cikkek ↔ kommentek

A webes korpusz építése során felmerült az ötlet, hogy a cikkek hozzászólásait érdemes volna külön is megvizsgálni. Az Aranyásó algoritmus sok weblap esetén képes jól megtanulni a fő tartalom helyét, azonban nem mindig képes különbséget tenni a hozzászólások és a cikk között. A kommentek sokszor eltérő jellegűek, mint maga a cikk: ezeket nem újságírók és szerkesztők készítik, hanem szabadon bárki, olykor keresetlen szavakkal, sok érzelemmel. Mivel a két tartalomtípus jellege jelentősen eltér, ezért erőfeszítéseket tettem annak érdekében, hogy ezeket elkülönítsem, azzal a céllal, hogy később külön is vizsgálhatóak legyenek. Ezért döntöttem úgy, hogy külön alkorpuszba válogatom a kommenteket: a korpusznak hasznos a minél több műfajú szöveg, viszont meghagytam a lehetőségét, hogy egy adott kutatásban ki- illetve bekapcsolható legyen. Így a nagyobb hírportálok hozzászólásait külön legyűjtöttem egy 58 millió szavas alkorpuszba. Ehhez arra volt szükség, hogy egy szövegben meghatározható legyen: hol ér véget egy cikk és hol kezdődnek a kommentek.

DOI:10.15774/PPKE.ITK.2016.003

A cikkek alatt található kommenteket bizonyos mintázatok (keresztnév egy dátummal vagy számmal; dátum #123; <szám> hozzászólás; „Hozzászólások 123”; „nickname 2014.02.02.”, stb.), illetve emotikon sűrűség (ha "`[;:X][()D]`" regex illeszkedése az adott szakaszon egy küszöbérték fölötti) alapján azonosítottam.

A kommentekben használt karakterek eloszlása eltér a cikkekétől. Ennek okát nem csak abban kereshetjük, hogy a kommentek gyakran ékezet nélküliek, hanem a hangulatjelek sűrűsége miatt is. A kommentezők legaktívabb rétegét a fiatalok teszik ki. Érdekes vizsgálat lehetne, hogy a fiatalok chat- és hozzászólás-megnyilvánulásai mennyiben térnek el korosztály szerint, illetve a kifejezési módot mennyiben befolyásolja a beviteli mód (más nyelvű billentyűzet, mobileszköz használata). Sokszor már nem is írnak teljes mondatokat, hanem rövid szavakat és sok hangulatjelet, smiley-kat.

Az internet hőskorában az emailekben megjelenő smiley-k létrejöttét kifejezetten a beviteli mód határozta meg: valamiképpen hangulati előjelet, érzelmet szerettek volna a szövegbe tenni, hiszen mindez élőszóban egyértelmű, de leírva nem. Sokszor egy mondat helyes értelmezéséhez ez elengedhetetlen, például hogy pejoratív vagy pozitív értelemben kell-e venni az írottakat.

	jel	**Előfordulás / összes tokenszám**			**Az eltérés aránya (komment/fő)**
		komment alkorpuszban *(58M token)*	**fő alkorpuszban** *(954M token)*	**egyéb alkorpuszban** *(228M token)*	
hangulatjel	:)	0,07%	0,02%	0,03%	3,6
	;)	0,004%	0,001%	0,002%	3,0
	:D	0,03%	0,004%	0,009%	5,8
	:(	0,008%	0,003%	0,01%	2,5
írásjel	?!	0,01%	0,001%	0,002%	12,0
	!!!	0,08%	0,004%	0,02%	17,3
	!	0,7%	0,2%	0,5%	3,0
	?	0,6%	0,35%	0,62%	1,7
	,	7,8%	7,7%	6,8%	0,9
	.	3,9%	4,4%	2,6%	1,0

42. táblázat. Néhány hangulatjel gyakorisága cikkekben és hozzászólásokban

DOI:10.15774/PPKE.ITK.2016.003

A számok azt mutatják, hogy a hangulatjelek már nemcsak az üzenet értelmezésében segítenek, hanem sokszor már ők maguk lettek az üzenet. Jelen dolgozat nem terjed ki ezek vizsgálatára, de a chat és kommentelési felületek (facebook, twitter, skype, viber, gmail chat) mutatják például az írott ifjúsági nyelvre irányuló kutatások fő forrásait, hiszen a fiatalok főként ezeken az eszközökön keresztül kommunikálnak egymással.

Jól látható a komment- és a fő korpusz alapján, hogy a hangulatjeleket egyre gyakrabban használjuk, de egyelőre csak az informális szövegekben. Azon írásjeleket is választottam az 42. táblázatba, amelyek érzéseket, hangulati töltetet tudnak adni egy mondatnak (?, !, ?!), illetve kontrollként hozzávettem a leggyakoribb írásjeleket is (, .). Jól látható, hogy egy cikkben inkább az egy hosszúságú írásjeleket (,.?!) használjuk. A kettő- illetve három hosszú írásjelek sokkal többször szerepelnek a komment korpuszban. Ennek oka valószínűleg az, hogy ezekben sokkal több az érzelmi töltet. Ezt sugallja az is, hogy a , és . írásjelek előfordulása közel azonos, azonban minden más írásjel és hangulatjel sokkal sűrűbben fordul elő kommentekben.

9.3. Szövegfeldolgozás

A **Huntoken** modullal (Halácsy és mtsai. 2004) történt a mondatszegmentálás. Ez a modul volt az első a feldolgozás során, ami a nyers szöveget elemezni kezdte. Gyakran előfordult, hogy bizonyos szövegdarabon elszállt. Főként az "egyéb" alkorpuszban található rendkívül hosszú felsorolások (terméklista, leggyakrabban keresett szavak, címkefelhők, stb.) nehezítették meg - érthető módon - a Huntoken dolgát. Például ha egy 3000 elemű termékfelsorolást (minden eleme külön sorban) megpróbált mondattá fűzni, akkor egy ponton hibaüzenettel kilépett. Nyilván ha az input nem mondat, elfogadható, ha a modul nem kezeli. (Az hiba, hogy nem kihagyja ezt a szövegdarabot, hanem elszáll.) Azonban a Huntoken ezen tulajdonsága jelen esetben kifejezetten hasznos volt: a modul szűrőként is működött. Ha megszakította a futást, akkor ez önmagában jelezte, hogy ezen a szakaszon nagy valószínűséggel nem mondat van. (Érdekességképpen jelzem, hogy volt 5000 szavas helyes mondat is!) Azaz a tokenizáló jelezte azt az adatot, ami vagy teljesen kihagyható, vagy esetleg az "egyéb" alkorpuszba való (például egy címkefelhő vagy felsorolás).

Ezért a korpuszt 7MB-os adagokban tokenizáltam, és ha tokenizáló megakadt egy ponton, akkor csak az adott fájl hátralévő része maradt ki a korpuszból. Második körben a félbehagyott fájlokat dolgoztam fel egy programmal: ha bizonyos hibajavító megoldások (pár sor kihagyása, újsorok beszúrása, stb.) után a tokenizáló tovább tudott dolgozni, akkor ez a szöveg is bekerült a korpuszba (de így már csak az "egyéb" alkorpuszba).

DOI:10.15774/PPKE.ITK.2016.003

A lemmákat a 4. fejezetben bemutatott **Humor** alapú lemmatizálóval (Endrédy és Novák 2015), a szófajokat az erre a lemmatizálóra is épülő **PurePos** (Orosz és Novák 2013) alkalmazásával jelöltem. Egy új, **pontosabb szófaji kódolás**sal (Ligeti-Nagy 2015) is annotáltam a szavakat. A korpuszban a főnévicsoport-felismerést a 8. fejezetben szereplő **HunTag3** (Endrédy és Indig 2015) segítségével végeztem. Így nem csak a szófaj és lemma alapján lehet keresni a korpuszban, hanem a főnévi csoportok is kereshetővé váltak a magyar weben előforduló szövegekben.

Nehéz annak megbecsülése, hogy ez a korpusz a teljes magyar web hányad része lehet. A Google sem lát minden oldalt, de jelenleg 460 millió magyar domént tart nyilván (*site:.hu* keresés eredménye). A web tulajdonsága, hogy folyamatosan változik. Nem csak új oldalak születnek, hanem egész domének is. Azon idő alatt, amíg ezt a bekezdést leírtam, 8297 új domén jött létre az *http://www.internetlivestats.com* szerint. Ez azt jelenti, hogy a crawlerek mindig fognak majd új oldalakat találni, ami a mi szempontunkból is pozitív: a korpuszépítés folyamatos lehet.

9.4. Kapcsolódó kutatások

Általános- illetve speciális webkorpusz építése régóta kutatási téma (Baroni és Ueyama 2006). Webes kommenteket már gyűjtöttek véleménybányászat (opinion mining) céljából, például The Birmingham Blog Corpus vagy a CROW (Neunerdt és mtsai. 2011). Magyar nyelvre is készült kommentkorpusz (Miháltz és mtsai. 2015), ami 1,9 millió Facebook-kommentet vizsgált (226 000 felhasználótól) politikai szempontból. A kommenteket az alábbi módon dolgozták fel: tokenizálás, lemmatizálás, szófajok jelölése, névelem-felismerés (politikusok, pártok) és érzelemelemzés (sentiment analysis). A pozitív, negatív megnyilvánulásokat elemezték, összevetve a választási eredményekkel (A cikkben vizsgált időszak eseménydús volt politikai téren: három választás is történt Magyarországon: Parlamenti-, Európai parlamenti- és Önkormányzati választás.)

DOI:10.15774/PPKE.ITK.2016.003

9.5. Összefoglalás

A crawler (2.2. fejezet) nagy adatmennyiséget gyűjtött össze 2012 óta, amelyet az *Aranyásó* algoritmussal (2.3.2. fejezet) megtisztítottam, többféle módon szűrtem, tokenizáltam, duplikátummentesítést végeztem mondatszinten, majd a korpuszt annotáltam saját lemmatizálóval (4. fejezet), többféle szófajjal és a HunTag3 segítségével NP címkékkel is (8. fejezet).

Az elkészült korpusz 1 242 090 499 tokent tartalmaz 67 845 166 mondatban, 33 453 doménről. A korpusz szigorú értelemben véve nem kiegyensúlyozott: nem nyelvészek válogatták a tartalmát, hanem a robot bejárási döntései, azaz a kigyűjtött linkek alapján épült. Heterogén, sokféle jellegű szöveget tartalmaz. (újságcikkek, blogok, videóportálok, bulvároldalak, játékportálok stb.). Szófajkóddal, lemmával és főnévi csoportok kezdő- és végcímkéivel is annotáltam a korpuszt. Az 1,2 milliárd tokenes méretével jelenleg a legnagyobb magyar annotált korpusz. (A szintén annotált MNSZ2 jelenlegi mérete 785 millió token.) A korpuszt kereshető formában közzétettem a népszerű Sketch Engine (Kilgarriff és mtsai. 2014) korpuszkeresőben.

Kapcsolódó tézis:

7. tézis: A crawler segítségével létrehoztam az 1,2 milliárd tokenes Pázmány korpuszt.

DOI:10.15774/PPKE.ITK.2016.003

10. Záró fejezetek

Kutatásomban először egy eszközt fejlesztettem, amely korpuszt képes építeni a webről, automatikus módszerrel. Ezzel az eszközzel korpuszokat hoztam létre. A letöltött szöveg jobb feldolgozásához egy tövesítőt, annak kiértékeléséhez egy tövesítő-kiértékelőt készítettem, minőségének normalizálásához pedig az ékezet nélküli szövegek ékezetesítésére alkalmas modult fejlesztettem ki. Az így nyert szövegben előforduló főnévi csoportokat (NP) és ezek kiemelése után a mondatban fennmaradó struktúrákat azonosítottam. A magyar NP-felismerésben sikerült az eddig legjobb eredményt elérnem. A részeredményeket kereshetővé, illetve megjeleníthetővé tettem egy webes felületen, amivel mások számára is használható kutatási eszköz jött létre.

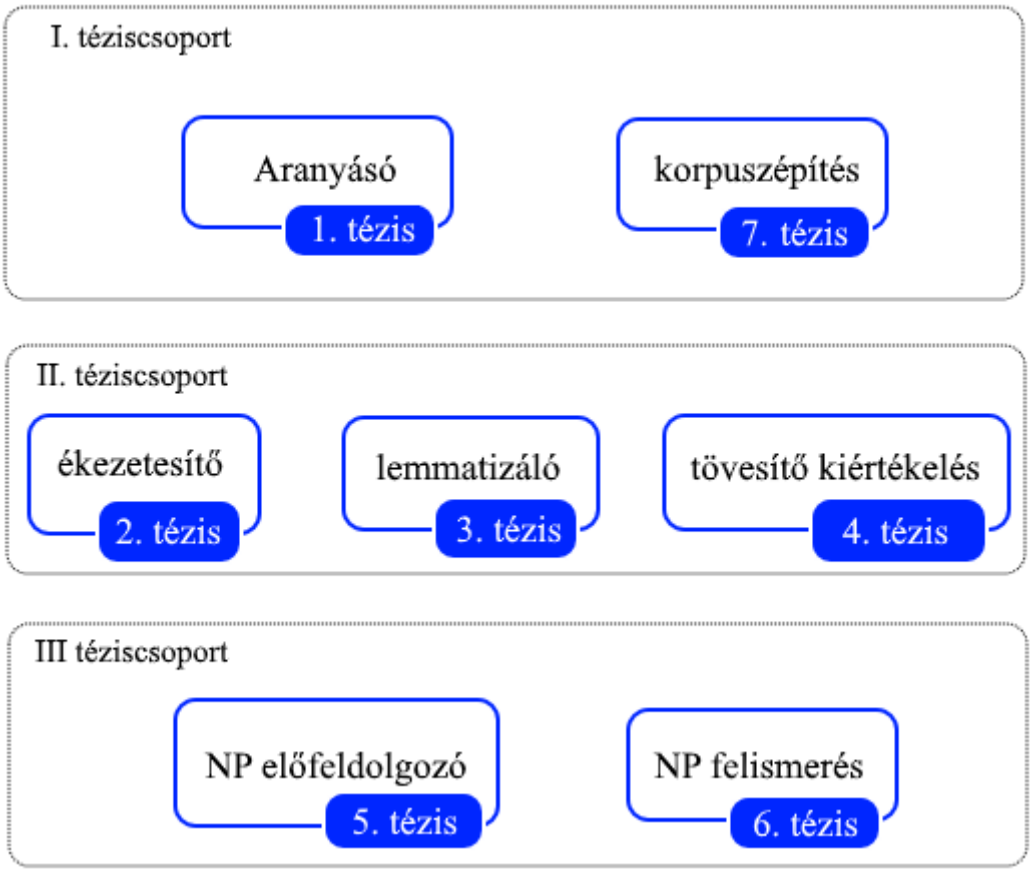
A **fő eredményem**nek tartom az **Aranyásó algoritmus**t, amely weblapokról öntanuló módon nyeri ki a értékes tartalmat; a **lemmatizáló**t, amely számos termékben alkalmaznak és amelyet a **tövesítő kiértékelés**ek legjobbnak mértek magyar nyelvre; a **magyar főnévi csoport felismerés**ben elért eredményeket, és az elkészült **Pázmány Korpusz**t.

A következőkben a dolgozat egyes fejezeteiben található téziseket összegyűjtve, egy helyen közlöm, rövid összefoglalással. Ez a jelen kutatás alapján megfogalmazható tézisek rövid foglalata.

DOI:10.15774/PPKE.ITK.2016.003

10.1. Új tudományos eredmények összefoglalása

A dolgozatban bemutatott eredményeket a három téziscsoportra lehet osztani (15. ábra). Az első téziscsoport a korpuszépítésre, illetve az ennek során használt, weboldalak értékes szövegének kinyerését végző algoritmusra vonatkozik. A második téziscsoportban létrehoztam a - Humor elemzőre épülő - lemmatizálót, ékezetesítőt és ezek kiértékelését végző módszert. Végül a harmadik téziscsoport a főnévicsoport-felismerés terén végzett újításokat foglalja magában.



15. ábra Tézisek áttekintése

I. TÉZISCSOPORT

Az első téziscsoportban azzal a problémával foglalkoztam, miként lehet algoritmikusan kinyerni egy weblapról az értékes szöveges tartalmat. Egy weboldal számos ismétlődő és irreleváns sablonos tartalmai megnehezítik a fő szöveges tartalom kiszűrését. A menük, fej- és láblécek, reklámok, a minden oldalon ismétlődő struktúra nem csak doménenként, olykor aldoménenként, hanem időben is változhat. Így olyan algoritmust kellett létrehozni, ami ezt öntanuló módon követni tudja, emberi beavatkozás nélkül. A jelenlegi sablonel távolító

DOI:10.15774/PPKE.ITK.2016.003

algoritmusoknál sikerült jobb eredményt elérni az Aranyásó (GoldMiner) algoritmussal.

Az Aranyásó alapján működő szövegkinyerő (2.3.2. fejezet) az egyes domének egyedi sajátosságait egy mintavétel alapján tanulja meg: megjegyzi azt a leggyakoribb HTML cimkesorozatot, amely közrefogja a legtöbb értékes tartalmat.

Egy crawler is készült, ami az Aranyásó segítségével gyűjtötte a magyar nyelvű szövegeket a webről. A letöltött szövegekből épült a 1,2 milliárd tokenes Pázmány korpusz (9. fejezet), amely a több mint 30 000 domén szövegeit tartalmazza. A további felhasználás érdekében a korpuszból különválogattam a kommenteket, mert ezek eltérő jellegűek és szerkezetűek.

1. tézis: Megalkottam a GoldMiner algoritmust, ami az eddigieknél hatékonyabban nyeri ki a weblapokról a cikkeket.

1.a tézis: Létrehoztam egy új crawlert, amely korpuszt tud építeni.

Kapcsolódó publikációk: [1], [9]

7. tézis: A crawler segítségével létrehoztam a 1,2 milliárd tokenes Pázmány Korpuszt.

II. TÉZISCSOPORT

A második téziscsoportban egy lemmatizálót illetve egy ékezetesítőt készítettem (3. és 4. fejezetek), mely előbbi kiértékeléséhez egy módszert és alkalmazást is megalkottam (5. fejezet).

A lemmatizáló kiértékelését két módon is elvégeztem: a közvetlen kimenetét illetve egy keresőrendszerbeli teljesítményét is megmértem. A közvetlen kiértékelés lehetővé tette, hogy gold standard lemmákhoz képest mérjem az egyes megoldások minőségét. Az IR kiértékelés ezzel szemben nem várta el a tövesítőktől, hogy a szótári tövet meghatározzák (stemmerektől ez nem is várható el), hanem itt elegendő, ha - a lemmapontosságtól eltekintve - jól vezeti vissza a ragozott alakokat egy közös alakra. Ez egyrészt képes megmutatni, ha egy adott nyelvre elegendő csak stemmert készíteni, lemmatizálóra nincs szükség, másrészt ez egy valós feladat közben méri a tövesítőket, nem laboratóriumi körülmények között. Mindkét kiértékelés gold standardjét lemmával annotált korpuszokból automatizáltan állítottam elő. Az eredmények összehasonlíthatóságához tíz tövesítőn (lemmatizálók és stemmerek vegyesen) végeztem el a kiértékelést, három nyelven (angol, lengyel, magyar).

Magyar nyelv esetén az elkészült lemmatizáló bizonyult a legjobbnak mindkét kiértékelésben. IR kiértékelésnél 85%, lemmapontosságnál 91,0%, tőalternatívák

DOI:10.15774/PPKE.ITK.2016.003

kiértékelésénél 91-94% F értékekkel. Az erősen ragozó magyar nyelv tulajdonsága, hogy tövesítő nélkül 17% eredményt lehet elérni.

2. tézis: Létrehoztam egy ékezetesítő modult, tervezésében egy szerzőtárssal együttműködve, ami egy módosított Humor-lexikonnal 94.3% pontossággal képes ékezet nélküli szövegek ékezetesítésére.

Kapcsolódó publikációk: [2], [11]

3. tézis: Létrehoztam egy lemmatizáló motort, tervezésében egy szerzőtárssal együttműködve, ami a Humor elemzéseiből kiszámolja a szó tövét, és ez a kiértékelések alapján a legjobb eredményeket adja a magyar nyelvre.

Kapcsolódó publikációk: [2], [4]

4. tézis: Létrehoztam egy kiértékelési módszert, amellyel tetszőleges, lemmával annotált korpusz alapján lemérhető egy tövesítő (i) pontossága, (ii) IR minősége, (iii) UI, OI, ERRT értéke, és (iv) egyéb metrikák szerinti kiértékelése. Mindezt angol, lengyel és magyar nyelvre 10 tövesítőre kimértem.

4.a tézis: Létrehoztam egy módszert, amivel korpuszból automatikusan létrehozható olyan gold standard, amely tövesítők közvetlen illetve IR-rendszerbeli kiértékeléséhez használható.

4.b tézis: A kiértékelést angol, lengyel és magyar nyelvekre elkészítettem

4.c tézis: Kimutattam, hogy az erősen ragozó nyelveknél (lengyel, magyar) a lemmapontosság és az IR-minőség korrelál.

4.d tézis: Kidolgoztam egy IR-tövesítőkiértékelést, ami natív IR rendszer nélkül képes IR-rel korreláló eredményt mérni.

4.e tézis: Definiáltam a tövesítő közvetlen kimenetére olyan kiértékeléseket, amelyek alkalmasak a tövesítők összehasonlítására, és visszajelzést is adhatnak a tövesítő hibáinak feltárására és azok kijavítására

4.f tézis: Kimutattam és megmértem, hogy az IR eredmény első *n* találatának kiértékelése alkalmas az IR-ranking algoritmus kiértékelésére is.

Kapcsolódó publikációk: [2], [4]

DOI:10.15774/PPKE.ITK.2016.003

III. TÉZISCSOPORT

A dolgozat harmadik téziscsoportjában a főnévi csoport felismerésével foglalkozom. Egyrészről a felismerés pontosságát javító jegyek definiálását és finomítását kutattam. Létrehoztam egy eszközt (7. fejezet), ami egy tanítóminta jegyeinek hasznosságát képes mérni és segíti a hangolásukat azzal, hogy áttekintést ad a jegyek és a kimeneti (IOB) címkék közötti kapcsolatra. Nem képes önállóan javítani a jegyeket, csupán egy segédeszköz szeretne lenni a nyelvész számára. A statisztika erejét és a nyelvész intuicióját kapcsolja össze.

Másrészről a HunTag hibaelemzésével, új jegyek definiálásával és a HunTag3 alkalmazásával sikerült az eddigi legjobb magyar NP felismerési eredményt elérni (8. fejezet).

5. tézis: Kimutattam, hogy az NP-felismeréshez használt jegykészlet tovább javítható, ha (i) a jegyek kevés értéket vehetnek fel, és (ii) ha létezik olyan finomabb osztályozás, amely mellett a kimeneti címkékkel jobban korreláló jegyeket kapunk.

5.a tézis: Az angol nyelv esetében a szófaj kategóriát olyan alkategóriákra bontottam, amelyek jobban korrelálnak az IOB-címkékkel (+2% javulás).

5.b tézis: Létrehoztam egy módszert, amivel a 5.a tézisben leírt művelet a WordNet synsetjeinek segítségével gépi támogatással elvégezhető.

5.c tézis: Becslést adtam egy jegyhalmaz hasznosságára a címke- és a jegykorreláció kiszámolásával, méghozzá egy tanulóalgoritmus (NLTK unigram, bigram chunker; HunTag; SS05) futási idejénél gyorsabban.

5.d tézis: Kimutattam, hogy nagyon fontos az input annotáció szerepe, ami akár fontosabb lehet, mint maga a gépi tanuló algoritmus. Bármilyen külső információ segít, ami „korrelál” a leendő kimeneti annotációval.

5.e tézis: Kimutattam, hogy kevés olyan szószintű jegy van, amely a kimeneti címkével 100%-osan korrelál, viszont ezek mindegyike fontos.

Kapcsolódó publikációk: [5], [6]

6. tézis: Sikerült javítanom a főnévi csoport felismerésének minőségét új jegyek definiálásával, egy társszerzővel létrehozott trigramátmeneti modellel és a HunTag3 alkalmazásával, amelyek együttesen az eddigi legjobb magyar NP-felismerést eredményezték. (93.59%)

6.a tézis: Kimutattam a főnévicsoport-felismerésnél, hogy a trigram-átmenetvalószínűségi modell csak részletesebb, háromnál több elemű (típusos, vagy finomabb felosztású) IOB címkék esetén tud többet adni, mint a bigram-modell.

Kapcsolódó publikációk: [6]

DOI:10.15774/PPKE.ITK.2016.003

10.2. Az eredmények alkalmazási területei

Az elkészített tövesítő modul számos cég alkalmazásaiba beépült: erre épül a Microsoft Indexing Service, az Országos Atomenergetikai Hivatalban tárolt dokumentumok tárolására és keresésére szolgáló rendszer, az MTI szerkesztőségi rendszere és a PolyMeta kereső. Az MTA Nyelvtudományi Intézete által készített Magyar Nemzeti Szövegtár második bővített kiadásának (MNSZ2) morfológiai annotálása ugyancsak ezzel az eszközzel készült.

A második téziscsoportban bemutatott tövesítő kiértékelési módszer angol, lengyel és magyar nyelveken túl más nyelvekre és további tövesítő modulok alapos vizsgálatára is használható. A kiértékeléshez készített alkalmazás felhasználásával más nyelvekre is automatikusan elkészíthető lemmapontosság- és IR minőség mérésére alkalmas gold standard. A kiértékelő script pedig további tövesítőkre is lefuttatható.

A dolgozat során készült 1,2 milliárd tokenes, lemmával, szófajilag és főnévi csoportokkal annotált Pázmány korpusz hasznos alapanyaga lehet további nyelvtechnológiai kutatásoknak.

DOI:10.15774/PPKE.ITK.2016.003

11. A szerző publikációi

Folyóiratcikk

[1] **Endrédy, István**, Attila Novák. 2013. “More Effective Boilerplate Removal—The GoldMiner Algorithm” *Polibits Journal* 48: pp. 79–83.

[2] **Endrédy István**, Novák Attila. 2015. “Szótövesítők összehasonlítása és alkalmazásaik” In: Navracsics Judit (szerk.) *Alkalmazott Nyelvtudomány,* XV. évfolyam, 1-2. szám, pp. 7-27, Veszprém

Könyvfejezet

[3] Indig Balázs, **Endrédy István**. 2016. “Gut, Besser, Chunker - Selecting the best models for text chunking with voting” In: A. Gelbukh (Ed.) *Lecture Notes in Computer Science*: Computational Linguistics and Intelligent Text Processing, Springer International Publishing *(megjelenés folyamatban)*

Külföldi konferenciakötet

[4] **Endrédy István**. 2015. “Corpus based evaluation of stemmers”, *7th Language & Technology Conference, Human Language Technologies as a Challenge for Computer Science and Linguistics*, pp. 234-239, Poznań

[5] **Endrédy István**. 2015. “Improving chunker performance using a web-based semi-automatic training data analysis tool”, *7th Language & Technology Conference, Human Language Technologies as a Challenge for Computer Science and Linguistics, ,* pp. 80-84, Poznań

[6] **Endrédy István**, Indig Balázs. 2015. “HunTag3, a general-purpose, modular sequential tagger – chunking phrases in English and maximal NPs and NER for Hungarian”, *7th Language & Technology Conference, Human Language Technologies as a Challenge for Computer Science and Linguistics*, pp. 213-218, Poznań

[7] **Endrédy, István**. 2014. “Hungarian-Somali-English Online Dictionary and Taxonomy.” In *Proceedings on “Collaboration and Computing for Under-Resourced Languages in the Linked Open Data Era,”* 38–43. Reykjavik, Iceland

[8] **Endrédy, István**, László Fejes, Attila Novák, Beatrix Oszkó, Gábor Prószéky, Sándor Szeverényi, Zsuzsa Várnai, and Beáta Wagner-Nagy. 2010. “Nganasan–Computational Resources of a Language on the Verge of Extinction.” In *7th SaLTMiL Workshop on Creation and Use of Basic Lexical Resources for Less-Resourced Languages (LREC 2010),* pp. 41-44 Valetta, Malta

DOI:10.15774/PPKE.ITK.2016.003

Hazai konferenciakötet

[9] **István Endrédy,** Novák Attila. 2012. “Egy hatékonyabb webes sablonszűrő algoritmus – avagy miként lehet a cumisüveg potenciális veszélyforrás Obamára nézve.” In: *IX. Magyar Számítógépes Nyelvészeti Konferencia*, pp 297–301. SZTE, Szeged

[10] Bakró-Nagy Marianne, **Endrédy István**, Fejes László, Novák Attila, Oszkó Beatrix, Prószéky Gábor, Szeverényi Sándor, Várnai Zsuzsa, Wagner-Nagy Beáta. 2010. “Online morfológiai elemzők és szóalakgenerátorok kisebb uráli nyelvekhez”. In: *VII. Magyar Számítógépes Nyelvészeti Konferencia*, pp. 345–348, SZTE, Szeged

[11] Novák, Attila, **István Endrédy**. 2005. “Automatikus Ë-jelölő program.” In: *III. Magyar Számítógépes Nyelvészeti Konferencia*, pp 453–54. SZTE, Szeged

DOI:10.15774/PPKE.ITK.2016.003

12. Irodalomjegyzék

Alexin, Zoltán, Tibor Gyimóthy, Csaba Hatvani, László Tihanyi, János Csirik, Károly Bibok, és Gábor Prószéky. 2003. „Manually annotated Hungarian corpus.” In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2*, 53–56. Association for Computational Linguistics.

Baldridge, Jason. 2005. „The OpenNLP project.” *URL: http://opennlp.apache.org/index.html , (accessed 30 April 2015)*.

Baroni, Marco, és Motoko Ueyama. 2006. „Building general-and special-purpose corpora by web crawling.” In *Proceedings of the 13th NIJL international symposium, language corpora: Their compilation and application*, 31–40.

Battelle, John. 2005. „The search: How Google and its rivals rewrote the rules of business and transformed our culture.”

Benesty, Jacob, Jingdong Chen, Yiteng Huang, és Israel Cohen. 2009. „Pearson correlation coefficient.” In *Noise reduction in speech processing*, 1–4. Springer.

Benko, Vladimír. 2013. „Data Deduplication in Slovak Corpora.” *Slovko 2013: Natural Language Processing, Corpus Linguistics, E-learning*, 27–39.

Biemann, Chris, Felix Bildhauer, Stefan Evert, Dirk Goldhahn, Uwe Quasthoff, Roland Schäfer, Johannes Simon, Leonard Swiezinski, és Torsten Zesch. 2013. „Scalable construction of high-quality web corpora.” *Journal for Language Technology and Computational Linguistics* 28 (2): 23–60.

Bird, Steven. 2006. „NLTK: the natural language toolkit.” In *Proceedings of the COLING/ACL on Interactive presentation sessions*, 69–72. Association for Computational Linguistics.

Bird, Steven, Ewan Klein, és Edward Loper. 2009. *Natural language processing with Python*. O’Reilly Media, Inc.

Brants, Thorsten. 2000. „TnT: a statistical part-of-speech tagger.” In *Proceedings of the sixth conference on Applied natural language processing*, 224–31. Association for Computational Linguistics.

Clear, Jeremy H. 1993. „The Digital Word.” In , szerkesztette George P. Landow és Paul Delany, 163–87. Cambridge, MA, USA: MIT Press.

Csendes, Dóra, János Csirik, Tibor Gyimóthy, és András Kocsor. 2005. „The Szeged Treebank.” In *Lecture Notes in Computer Science: Text, Speech and Dialogue*, 123–31. Springer.

Degórski, Łukasz, és Adam Przepiórkowski. 2012. „Recznie znakowany milionowy podkorpus NKJP.” In *Narodowy Korpus Języka Polskiego*, szerkesztette Adam Przepiórkowski, Mirosław Tomasz Bańko, Rafała L. Górski, és BarbaraEditors Lewandowska-Tomaszczyk, 51–58. Wydawnictwo Naukowe PWN.

Déjean, Hervé. 2000. „Learning Syntactic Structures with XML.” In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th CoNLL - Volume 7*, 133–35. ConLL ’00. Stroudsburg, PA, USA: ACL. http://dx.doi.org/10.3115/1117601.1117632.

Endrédy, István. 2015a. „Corpus based evaluation of stemmers.” In *7th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, szerkesztette Zygmunt Vetulani; Joseph Mariani. Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu.

———. 2015b. „Improving chunker performance using a web-based semi-automatic training data analysis tool.” In *7th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, szerkesztette Zygmunt Vetulani; Joseph Mariani. Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu.

DOI:10.15774/PPKE.ITK.2016.003

Endrédy, István, és Balázs Indig. 2015. „HunTag3: a general-purpose, modular sequential tagger – chunking phrases in English and maximal NPs and NER for Hungarian.” In *7th Language & Technology Conference, Human Language Technologies as a Challenge for Computer Science and Linguistics (LTC ’15)*, 213–18. Poznań, Poland: Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu.

Endrédy, István, és Attila Novák. 2013. „More Effective Boilerplate Removal—the GoldMiner Algorithm.” *Polibits - Research Journal on Computer Science and Computer Engineering with Applications*, sz. 48: 79–83.

———. 2015. „Szótövesítők összehasonlítása és alkalmazásaik.” Szerkesztette Navracsics Judit. *Alkalmazott Nyelvtudomány* 15 (1-2): 7–27.

Erjavec, Tomaž. 2010. „MULTEXT-East Version 4: Multilingual Morphosyntactic Specifications, Lexicons and Corpora.” In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC’10)*, szerkesztette Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, és Daniel Tapias. Valletta, Malta: European Language Resources Association (ELRA).

Finkel, Jenny Rose, Trond Grenager, és Christopher Manning. 2005. „Incorporating non-local information into information extraction systems by gibbs sampling.” In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, 363–70. Association for Computational Linguistics.

Finn, Aidan, Nicholas Kushmerick, és Barry Smyth. 2001. „Fact or Fiction: Content Classification for Digital Libraries.” In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*.

Gormley, Clinton, és Zachary Tong. 2015. *Elasticsearch: The Definitive Guide*. O’Reilly Media, Inc.

Halácsy, Péter, András Kornai, László Németh, András Rung, István Szakadát, és Viktor Trón. 2004. „Creating open language resources for Hungarian.” *Proceedings of 4th Conference on Language Resources and Evaluation (LREC)*, 203–10.

Halácsy, Péter, András Kornai, Csaba Oravecz, Trón Viktor, és Dániel Varga. 2006. „Using a morphological analyzer in high precision POS tagging of Hungarian.” *Proceedings of 5th Conference on Language Resources and Evaluation (LREC)*, 2245–48.

Halácsy, Péter, és Viktor Trón. 2007. „Benefits of resource-based stemming in Hungarian information retrieval.” In *Evaluation of Multilingual and Multi-modal Information Retrieval*, 99–106. Springer.

Hattyár, Helga, Miklós Kontra, és Fruzsina Sára Vargha. 2009. „Van-e Budapesten zárt ë?” *Magyar Nyelv* 105: 453–68.

Hulden, Mans. 2009. „Foma: a finite-state compiler and library.” In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*, 29–32. Association for Computational Linguistics.

Hull, David A. 1996. „Stemming algorithms: A case study for detailed evaluation.” *JASIS* 47 (1): 70–84.

Indig, Balázs, és István Endrédy. 2016. „Gut, Besser, Chunker – Selecting the best models for text chunking with voting.” In *Computational Linguistics and Intelligent Text Processing - 17th International Conference, CICLing 2016*. Konya, Turkey: Springer.

Johansson, Christer. 2000. „A Context Sensitive Maximum Likelihood Approach to Chunking.” In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th CoNLL - Volume 7*, 136–38. CoNLL ’00. Stroudsburg, PA, USA: ACL. doi:10.3115/1117601.1117633.

Kilgarriff, Adam, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlỳ, és Vít Suchomel. 2014. „The Sketch Engine: ten years on.” *Lexicography* 1 (1): 7–36.

Koeling, Rob. 2000. „Chunking with Maximum Entropy Models.” In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th CoNLL - Volume 7*, 139–41. CoNLL ’00. Stroudsburg, PA, USA: ACL. doi:10.3115/1117601.1117634.

DOI:10.15774/PPKE.ITK.2016.003

Kohlschütter, Christian, Peter Fankhauser, és Wolfgang Nejdl. 2010. „Boilerplate detection using shallow text features." In *Proceedings of the third ACM international conference on Web search and data mining*, 441–50. WSDM '10. New York, NY, USA: ACM. doi:10.1145/1718487.1718542.

Kontra, Miklós, Csilla Bartha, Anna Borbély, Helga Hattyár, és Tamás Váradi. 2011. „Budapesti beszélt nyelvi vizsgálatok= The study of spoken language in Budapest." *OTKA Kutatási Jelentések| OTKA Research Reports*.

Kornai, András, Péter Halácsy, Viktor Nagy, Csaba Oravecz, Viktor Trón, és Dániel Varga. 2006. „Web-based frequency dictionaries for medium density languages." In *Proceedings of the 2nd International Workshop on Web as Corpus*, 1–8. Association for Computational Linguistics.

Kornai, András, Péter Rebrus, Péter Vajda, Péter Halácsy, András Rung, és Viktor Trón. 2004. „Általános célú morfológiai elemző kimeneti formalizmusa (The output formalism of a general-purpose morphological analyzer)." In *Proceedings of the 2nd Hungarian Computational Linguistics Conference*, 172–76. Szeged, Hungary.

Kornai, András, és Géza Tóth. 1997. „Gépi ékezés." *MAGYAR TUDOMÁNY* 42 (4): 400–410.

Lafferty, John, Andrew McCallum, és Fernando CN Pereira. 2001. „Conditional random fields: Probabilistic models for segmenting and labeling sequence data."

Lampos, Vasileios, Daniel Preotiuc-Pietro, Sina Samangooei, Douwe Gelling, és Trevor Cohn. 2014. „Extracting socioeconomic patterns from the news: Modelling text and outlet importance jointly." *ACL 2014*, 13.

Ligeti-Nagy, N. 2015. „Szövegkorpuszok pontosabb annotációja gépi elemzéshez." In *Többnyelvűség és kommunikáció Kelet-Közép-Európában*, szerkesztette A. Benő, E. Fazekas, és E. Zsemlyei, 421–29.

Lindén, Krister. 2009. „Entry generation by analogy—encoding new words for morphological lexicons." *Northern European Journal of Language Technology* 1 (1): 1–25.

McCandless, Michael, Erik Hatcher, és Otis Gospodnetic. 2010. *Lucene in Action: Covers Apache Lucene 3.0*. Manning Publications Co.

Miháltz, Márton. 2011. „Magyar NP-felismerők összehasonlítása (Comparing Hungarian NP-chunkers)." In *Proceedings of the 8th Hungarian Computational Linguistics Conference*, 333–35. Szeged, Hungary.

Miháltz, Márton, Tamás Váradi, István Csertő, Éva Fülöp, Tibor Pólya, és Pál Kővágó. 2015. „Beyond Sentiment: Social Psychological Analysis of Political Facebook Comments in Hungary." In *6TH WORKSHOP ON COMPUTATIONAL APPROACHES TO SUBJECTIVITY, SENTIMENT AND SOCIAL MEDIA ANALYSIS WASSA 2015*, 127.

Miller, George A. 1995. „WordNet: a lexical database for English." *Communications of the ACM* 38 (11): 39–41.

Molina, Antonio, és Ferran Pla. 2002. „Shallow parsing using specialized hmms." *The Journal of Machine Learning Research* 2: 595–613.

Németh, Géza, Csaba Zainkó, László Fekete, Gábor Olaszy, Gábor Endrédi, Péter Olaszi, Géza Kiss, és Péter Kis. 2000. „The design, implementation, and operation of a Hungarian e-mail reader." *International Journal of Speech Technology* 3 (3-4): 217–36.

Neunerdt, Melanie, Bianka Trevisan, Tomas Cury Teixeira, Rudolf Mathar, és Eva-Maria Jakobs. 2011. „Ontology-based Corpus Generation for Web Comment Analysis." In *ACM conference on Hypertext and hypermedia (HT 2011)*. Eindhoven.

Novák, Attila. 2003. „Milyen a jó humor." *Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2003)*, 138–45.

———. 2015. „Making Morphologies the "Easy" Way." In *Computational Linguistics and Intelligent Text Processing*, 127–38. Springer.

Novák, Attila, és István Endrédy. 2005. „Automatikus zárt ë-jelölő program." In *A 3. Magyar Számítógépes Nyelvészeti Konferencia előadásai*, 453–54. Szeged, Hungary.

Novák, Attila, és Borbála Siklósi. 2015. „Automatic Diacritics Restoration for Hungarian." *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2286–91.

DOI:10.15774/PPKE.ITK.2016.003

Okazaki, Naoaki. 2007. *CRFsuite: a fast implementation of Conditional Random Fields (CRFs)*. http://www.chokkan.org/software/crfsuite/.

Oravecz, Csaba, Tamás Váradi, és Bálint Sass. 2014. „The Hungarian Gigaword Corpus.” In *Proceedings of LREC*. Reykjavik.

Orosz, György, és Attila Novák. 2013. „PurePos 2.0: a hybrid tool for morphological disambiguation.” In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2013)*, 539–45. Hissar, Bulgaria: INCOMA Ltd. Shoumen, BULGARIA. http://aclweb.org/anthology//R/R13/R13-1071.pdf.

Osborne, Miles. 2000. „Shallow Parsing As Part-of-speech Tagging.” In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th CoNLL - Volume 7*, 145–47. ConLL ’00. Stroudsburg, PA, USA: ACL. doi:10.3115/1117601.1117636.

Paice, Chris D. 1994. „An evaluation method for stemming algorithms.” In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 42–50. Springer-Verlag New York, Inc.

Pomikálek, Jan. 2011. „Removing Boilerplate and Duplicate Content from Web Corpora.” PhD dissertation, Masaryk University, Faculty of Informatics.

Porter, Martin F. 1980. „An algorithm for suffix stripping.” *Program* 14 (3): 130–37.

Prószéky, Gábor, és Balázs Indig. 2015. „Magyar szövegek pszicholingvisztikai indíttatású elemzése számítógéppel.” *Alkalmazott Nyelvtudomány* 15 (1-2): 29–44.

Prószéky, Gábor, és Balázs Kis. 1999. „A Unification-based Approach to Morpho-syntactic Parsing of Agglutinative and Other (Highly) Inflectional Languages.” In *ACL*, szerkesztette Robert Dale és Kenneth Ward Church. ACL. http://dblp.uni-trier.de/db/conf/acl/acl1999.html#ProszekyK99.

Prószéky, Gábor, Miklós Pál, és László Tihanyi. 1994. „Humor-based applications.” In *Proceedings of the 15th conference on Computational linguistics-Volume 2*, 1270–73. Association for Computational Linguistics.

Prószéky, Gábor, és László Tihanyi. 1992. „A Fast Morphological Analyzer for Lemmatizing Corpora of Agglutinative Languages.” Szerkesztette G\’abor Kiss Ferenc Kiefer és J\’ulia Pajzs. *Papers in Computational Lexicography*, 265–78.

Quasthoff, Uwe, Dirk Goldhahn, és Gerhard Heyer. 2013. „Technical Report Series on Corpus Building.” *Abteilung Automatische Sprachverarbeitun g, Institut für Informatik, Universität Leipzig*, sz. 5: 1–238.

Recski, Gábor. 2014. „Hungarian Noun Phrase Extraction Using Rule-based and Hybrid Methods.” *Acta Cybernetica* 21 (3): 461–79.

Recski, Gábor, és Dániel Varga. 2012. „Magyar főnévi csoportok azonosítása (Identifying Hungarian noun phrases).” *Általános Nyelvészeti Tanulmányok*, 81–95.

Sass, Bálint. 2008. „The verb argument browser.” In *Text, Speech and Dialogue*, 187–92. Springer.

———. 2011. *Igei szerkezetek gyakorisági szótára - Egy automatikus lexikai kinyerő eljárás és alkalmazás*. PhD disszertáció. Pázmány Péter Katolikus Egyetem.

Shen, Hong, és Anoop Sarkar. 2005. „Voting Between Multiple Data Representations for Text Chunking.” In *Advances in Artificial Intelligence, 18th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2005, Victoria, Canada, May 9-11, 2005, Proceedings*, szerkesztette Balázs Kégl és Guy Lapalme, 3501:389–400. Lecture Notes in Computer Science. Springer.

Simon, Eszter. 2013. „Approaches to Hungarian Named Entity Recognition.” Budapest University of Technology and Economics Budapest.

Smiley, David, Eric Pugh, Kranti Parisa, és Matt Mitchell. 2015. *Apache Solr Enterprise Search Server*. Packt Publishing Ltd.

Sun, Xu, Louis-Philippe Morency, Daisuke Okanohara, és Jun’ichi Tsujii. 2008. „Modeling latent-dynamic in shallow parsing: a latent conditional model with improved inference.” In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, 841–48. Association for Computational Linguistics.

Tarján, Balázs, Tímea Nagy, Péter Mihajlik, és Tibor Fegyó. 2013. „Magyar nyelvű, kísérleti e-mail diktáló rendszer.” *IX. Magyar Számítógépes Nyelvészeti Konferencia*, 13–20.

DOI:10.15774/PPKE.ITK.2016.003

Tjong Kim Sang, Erik F., és Sabine Buchholz. 2000. „Introduction to the CoNLL-2000 Shared Task: Chunking.” In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, 127–32. ConLL ’00. Stroudsburg, PA, USA: Association for Computational Linguistics.

Tjong Kim Sang, Erik F, és Jorn Veenstra. 1999. „Representing text chunks.” In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, 173–79. Association for Computational Linguistics.

Tordai, Anna, és Maarten De Rijke. 2006. *Four stemmers and a funeral: Stemming in hungarian at clef 2005*. Springer.

Trón, Viktor, Péter Halácsy, Péter Rebrus, András Rung, Péter Vajda, és Eszter Simon. 2006. „Morphdb. hu: Hungarian lexical database and morphological grammar.” In *Proceedings of 5th International Conference on Language Resources and Evaluation*, 1670–73.

Trón, Viktor, András Kornai, György Gyepesi, László Németh, Péter Halácsy, és Dániel Varga. 2005. „Hunmorph: open source word analysis.” In *Proceedings of the Workshop on Software*, 77–85. Association for Computational Linguistics.

Váradi, Tamás. 2002. „The Hungarian National Corpus.” In *Proceedings of the Third International Conference on Language Resources and Evaluation*, 385–89. Las Palmas.

Zainkó, Csaba, és Géza Németh. 2010. „Ékezetek gépi helyreállítása.” In *A MAGYAR BESZÉD; Beszédkutatás, beszédtechnológia, beszédinformációs rendszerek*, 485–88. Budapest: Akadémiai Kiadó.