

TDK-dolgozat

Skrabák Boglárka

Alkalmazás a HuSpaCy és e-magyar nyelvi elemzők összehasonlításához

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

HUN-REN NYELVTUDOMÁNYI KUTATÓKÖZPONT



Szerzők:

Skrabák Boglárka

programtervező informatikus BSc

3. évfolyam

Témavezetők:

Dr. Ligeti-Nagy Noémi

HUN-REN Nyelvtudományi

Kutatóközpont, Nyelvtechnológiai

Kutatócsoport, tudományos munkatárs

Budapest, 2024. 12. 17.

Tartalomjegyzék

1. Bevezetés	3
2. Elemzés és tervezés	4
2.1. Az alkalmazás bemutatása	5
2.1.1. Az alkalmazás céljai és azok megvalósulása	5
2.1.2. A feladat nyelvtechnológiai kutatási vonatkozásának körvonalazódása	6
3. Megvalósítás és alkalmazás	8
3.1. Elvi gyengeségek és logikai egységesítés az elemzők között	8
3.1.1. Tokenizálás	8
3.1.2. Morfológia	9
3.1.3. Lemmatizálás	9
3.1.4. Szófaji címkézés	9
3.1.5. Függőségi elemzés	10
3.1.6. Névelem-felismerés	12
3.2. Különbségek kezelése	12
3.3. Teljesítményjavítás	14
3.3.1. Kivétel megelőzése	16
3.3.2. Maximumkeresés javítása	17
3.3.3. A szófaji ellenőrzés mellőzése	19
3.3.4. A 0 hasonlósági arány és az értelmetlen elemzések elválasztása . .	19
4. Összefoglalás és eredmények	23
Köszönetnyilvánítás	24
A. Szemelvények a tesztszövegből	25
A.1. Szövegrészlet, amelyben előfordul időbélyeg	25

A.2. Példa hibás helyesírású alakokra	25
A.3. Jegyzőkönyvi szövegelemek és előbeszédbe bekialálások	26
Irodalomjegyzék	28
Ábrajegyzék	31
Táblázatjegyzék	32
Forráskódjegyzék	33

1. fejezet

Bevezetés

Napjainkban a nyelvtechnológiai kutatások és gyakorlati alkalmazások alapvető része az, hogy szövegeket tudjunk értelmezni és nyelvészeti szempontból elemezni. Ehhez szükségünk van különböző elemzőrendszerekre, amelyek minél nagyobb pontossággal tudják leképezni egy-egy szövegrész nyelvi sajátosságait. A magyar nyelv szempontjából ezen a területen kiemelkedő jelentőségű a HuSpaCy [1, 2] és az e-magyar [3, 4, 5, 6, 7, 8, 9, 10, 11, 12] elemző.

A két elemző meglehetősen más megközelítéssel dolgozik, vannak különböző gyengeségeik és hibáik, amelyekről fontos tájékozódnia annak a kutatónak, aki a saját specifikus célfeladatán szeretné alkalmazni ezeket az eszközöket. Ezt az igényt próbálja kielégíteni az általam fejlesztett Launcher_huspace_emagyar alkalmazás, amely a klasszikus szoftverfejlesztés keretein túlmutató munkát is igényelt. Az alkalmazás elkészítése során fény derült az elemzők néhány, nyelvtechnológiai szempontból kifejezetten fontos tulajdonságára, sőt javítható hibájára is. A dolgozat célja ezek felderítése, tárgyalása, a megoldási javaslatok prezentálása és az eredmények összevetése az eredeti állapottal. Mivel a problémák szorosan összefonódnak az alkalmazás fejlesztési folyamatával, annak tükrében és a fejlesztési folyamattal párhuzamot vonva tárgyalom az egyes témákat.

2. fejezet

Elemzés és tervezés

A számítógépes szövegfeldolgozás a nyelvtchnológiai gyakorlati feladatok alapköve. A magyar nyelv feldolgozásában kiemelkedő teljesítményt nyújt például a HuSpaCy és az e-magyar rendszer. Az alábbiakban tömören összefoglalom az elemzők főbb tulajdonságait, leginkább arra a modularitásra fókuszálva, amelyre az általam fejlesztett Launcher_huspace_emagyar alkalmazás épül és összpontosít.

A HuSpaCy egy ipari feladatokra is kiválóan alkalmas, gyors és hatékony elemző, SpaCy-könyvtár formájában elérhető alkalmazás, amely SpaCy-modellek felhasználásával nyújt szövegfeldolgozási eszközöket. A SpaCy által nyújtott alapvető eszközök mellett a magyar nyelvre specializált elemzőlánc tartalmaz tokenizáló, mondattagoló, lemmatizáló, morfológiai elemző, szófaji címkéző, függőségi elemző és névelem-felismerő modult. [13]

Az e-magyar inkább kutatási célú, sokféle elemzési komponenst magába foglaló elemzőlánc. Tartalmaz tokenizálót (emToken), morfológiai elemzőt (emMorph), lemmatizálót (emLem), szófaji címkézőt (emTag), függőségi elemzőt (emDep), összetevős elemzőt (emCons), részleges mondatelemzőt (emChunk), valamint névelem-felismerőt (emNer). Az e-magyar fő erőssége, hogy nagy mértékben alkalmazkodik a magyar nyelv adottságaihoz, erről a későbbiekben a morfológiai elemzés kapcsán lesz szó részletesebben.

A Launcher_huspace_emagyar (továbbiakban: Launcher) alkalmazás célja, hogy a két elemző nyelvészeti teljesítményének összehasonlításához megfelelő, kényelmesen használható környezetet nyújtson. Ez számos olyan egységesítést követelt meg, amely során legalább az egyik elemzőt (illetve annak kimenetét) informatikailag át kellett alakítani, hogy értelmezhető legyen az összehasonlítás egy közös szempont mentén. Ennek során derültek ki olyan, többnyire informatikai megvalósítási problémában gyökerező jelensé-

gek és esetenként hibák, amelyek jelentős mértékben hatással lehetnek az elemzési eredményekre, és ezzel a nyelvészeti teljesítményre is.

2.1. Az alkalmazás bemutatása

A Launcher egy konzolos Python-alkalmazás formájában működik, a launcher.py fájl futtatásával és a megfelelő paraméterek, kapcsolók kiválasztásával indítható el. Ez az egyszerű állítás már önmagában is az alkalmazás egyik fő célkitűzése és életre hívó követelménye.

2.1.1. Az alkalmazás céljai és azok megvalósulása

A HuSpaCy Python-könyvtár, az e-magyar pedig Docker-alkalmazás formájában üzemel jelenleg, és ez a két különböző megközelítés az eddigiek során jelentősen megnehezítette az elemzőkkel dolgozó felhasználó munkáját. A felhasználónak indítania kellett egy Python-scriptet, amiben importálta a HuSpaCy-t, elindította az elemzőláncot, beállította, mely szempontokra lesz szüksége az elemzésből, majd kinyerte a kimenetet. Az e-magyar esetében pedig indítania kellett egy Docker-konténert, lefuttatni benne az elemzőt, majd a kimeneti fájlt opcionálisan átemelni a Docker konténerből a lokális tárhelyre, hogy ott fel tudja dolgozni a megfelelő módon. A fent leírt műveletek önmagukban nem túl bonyolultak, de annak a kutatónak, akit csak az a kérdés érdekel, hogy egy adott szövegen melyik elemző teljesített jobban, egyértelműen kényelmetlen és hosszadalmas folyamat a fenti lépéseket elvégezni. A Launcher a Docker SDK for Python - docker Python-könyvtár segítségével a fent leírt műveletsort automatikusan elvégzi az e-magyar futtatása során, az elemzési fájlt is átemeli a lokális tárhelyre, majd a konténert törli az elemzés befejeztével. Erről a felhasználót tájékoztatja, így ha a működés valamilyen oknál fogva megszakadna, és a konténer nem törlődne megfelelően, a felhasználó értesül a hibáról.

A program futtatásához meg kell adni egy tetszőleges txt kiterjesztésű bemeneti fájl elérési útvonalát, amely lehet relatív is. Ezen kívül meg lehet adni, melyik elemző fusson le (lehet mindkettő, vagy akár csak az egyik is), erre szolgál a -huspacy és a -emagyar kapcsoló, valamint az -outh és -oute kapcsolókkal megadható az is, hogy melyik elemző nyers kimenetét szeretnénk a képernyőn látni. További opcionális kapcsolókkal az egyes nyelvi szintenkénti eredményeket kérhetjük le.

A két kapott elemzési fájl a program eltárolja, a felhasználó bármikor hozzáfér az eredeti, nyers kimenetekhez. Ezek *whitespace*-ekkel tagolt, táblázatos fájlok, az oszlopokban az elemzési szempontokat látjuk, a sorokban pedig a tokeneket (illetve az e-magyar esetében mondatonként egy-egy *newline* karaktert is, amely leginkább a tagolást szolgálhatja). Ez a két kimenet önmagában elemezhető és némi átalakítással, táblázatkezelő programok segítségével viszonylag jól használható, azonban az összehasonlításra közvetlenül nem alkalmas. A Launcher ezért kinyeri ezekből a kimenetekből a megfelelő információkat, majd nyelvi szintenként dolgozza fel és adja vissza őket egy-egy összehasonlítási táblázat és esetenként néhány egyéb hasznos adat formájában. Ilyen táblázat elérhető a tokenizálás (-tok kapcsoló), a morfológia (-morph kapcsoló), a lemmatizálás (-lem), a szófaji címkézés (-pos kapcsoló), a függőségi elemzés (-dep kapcsoló) és a névelemfelismerés (-ner) szintjén. A táblázatok első oszlopa mindig az adott szempontok szerinti egyenlőségvizsgálat eredményét adja vissza (True/False), majd a többi oszlopban rendre a HuSpaCy eredményeit és az e-magyar eredményeit közli, az utolsó két oszlopban pedig az eredeti tokeneket adja vissza.

2.1.2. A feladat nyelvtechnológiai kutatási vonatkozásának körvonalazódása

A fentiekben leírt specifikáció önmagában még csak egy szoftverfejlesztési folyamatot takar. Azonban a fejlesztés közben kiderült, hogy a fent leírt követelmények implementálásához meg kell ismerni, sőt, át is kell alakítani az elemzők működési logikáját, egyes esetekben némi nyelvészeti rálátás segítségével. Így a Launcher 5 fő feladatot lát el valójában: az elemzők futtatását, egyszerű teljesítménymetrika implementálását (egyenlőségvizsgálat az összehasonlítható kimenetek között), logikai egységesítést, különbségkezelést, illetve teljesítményjavítást. A dolgozat célja az utóbbi három szempont részletes tárgyalása, a futtatás menetéről már esett szó a 2.1.1. fejezetben, a teljesítménymetrika eredményei pedig nyelvészeti vizsgálatok tárgykörébe tartoznak.

A vizsgálatokhoz a 2022. 06. 20-ai országgyűlési jegyzőkönyv szövegét használtam fel, ugyanis ez egy 2498 sorból, 290 606 karakterből álló szöveg, amelyen nagyon jól tesztelhető számos olyan eset, ahol az elemzők más eredményt adnak. Az írott szöveg és az előbeszéd jellegzetességei is előfordulnak ebben a szövegben. Találhatunk például retorikus, nyelvtanilag igényes és alaposan megfogalmazott szövegre, beszélt nyelvben előforduló reakciókra, helyesírási hibákra, illetve a jegyzőkönyvi formából adódó szövegrész-

letekre is (pl. leíró megjegyzések, megszólalók nevei, tagolókarakterek, sőt e-mail-címek, postafiók száma stb.).

3. fejezet

Megvalósítás és alkalmazás

3.1. Elvi gyengeségek és logikai egységesítés az elemzők között

3.1.1. Tokenizálás

A tokenizálás fogalmát a legtöbb forrás a szöveg szavakra bontásaként definiálja. [14, 15, 16, 17]

A naiv megközelítés szerint elegendő lenne ehhez a *whitespace* karakterek és a központosítási karakterek mentén feldarabolni a szöveget, azonban a valóságban ez az irányelv nem elég, például a szóközzel tagolt számok egy tokennek kell számítsanak, illetve a sorszámoknál is célszerű a pontot a sorszám részének tekinteni. Ezek többnyire elemzési hibákhoz vezetnek, erről a későbbiekben lesz szó.

A fenti irányelv implementálása során a tokenként nem értelmezendő *whitespace* karaktereket érdemes megtartani, hogy a tokenizált szövegből visszaállítható legyen az eredeti. Az e-magyar ezt úgy oldja meg, hogy minden tokenhez megadja, milyen *whitespace* követi őt a szövegben (ez a nyers elemzésben pl. külön oszlop). Ezen kívül minden mondatot egy plusz *newline* karakterrel választ el. A HuSpaCy ezzel szemben tokenként felsorolja a *whitespace*-csoportokat is, így az összes token konkatenációjával állítható vissza az eredeti szöveg. Önmagában nem rossz ez a megközelítés, azonban a HuSpaCy a *whitespace*-eket tokenként is értelmezi, és megpróbál hozzájuk valamilyen értelmes elemzési címkét társítani (pl. szófajilag felcímkézi őket főnévnek, igének, stb.). Ez a megjelenítés szempontjából is zavaró: pl. a „\n \n” csoport két sortörés formájában jelenik meg. Az értelmetlen elemzéseknek a kiszűrésére van beépített metódus (`token.is_space`), azonban

a *pipeline*-on ettől függetlenül még végighalad a token, és ez fog még problémát okozni.

Ezen a szinten tehát az egységesítés azt jelenti, hogy az e-magyar kimenetéből leszűrjük a mondattagoló *newline* karaktereket, a HuSpaCy kimenetére pedig alkalmazzuk a beépített *whitespace*-szűrőt.

3.1.2. Morfológia

A morfológia szintjén az alkalmazás egységesítési feladata az elemzések egymásnak történő megfeleltetése. Az e-magyarban a morfológiai elemzést az EmMorph komponens valósítja meg, amely kimenetként képes kétféle elemzést biztosítani: egyrészt egy Universal Dependencies formájú kimenetet, másrészt pedig a magyar nyelv szempontjából sokkal jobban értelmezhető xpostag elemzést. Ezzel szemben a HuSpaCy tartalmazza a SpaCy-ből származó Universal Dependencies elemzést, illetve integrált formában az EmMorph komponenst, amelyből előállít Universal Dependencies és xpostag-formájú elemzést. Így összesen 5 formában, 3-féle forrásból elérhető valamilyen típusú elemzés. Az alkalmazás az összehasonlítást az xpostag-elemzéseken végzi el.

3.1.3. Lemmatizálás

A lemmatizálás gyakorlatilag a morfológiai elemzés részeredménye, ugyanis mindegyik elemzési forma megtalál egy lemmát a toldalékok leválasztása során. A morfológiai elemzések 3 forrásának megfelelően 3-féle lemmával dolgozhatunk: az e-magyar lemmájával, a HuSpaCy saját, SpaCy-jellegű lemmájával, illetve az integrált EmMorph morfológiai elemzéséből kinyert lemmával. Az alkalmazás az összehasonlítási egyenlőségvizsgálatot a három formán együttesen végzi el.

3.1.4. Szófaji címkézés

Szófaji címkét 4 forrásból nyerhetünk. Egyrészt az e-magyarból kapunk egy címkét, másrészt pedig a HuSpaCy-ből hármat: a pos (egyszerű Universal Dependencies címke), a tag (részletes SpaCy-címke) elemzéseket, illetve az integrált EmMorph morfológiai elemzéséből kinyerhető szófaji címkét.

A szófaji címkék halmaza érdekes módon különbözik az e-magyarban és a HuSpaCy-ben. Elvileg mindkét esetben Universal Dependencies szófaji címkéket látunk, gyakorlatilag azonban megfigyelhető, hogy az e-magyarban a koordináló konjunkció (melléren-

delő kötőszó, pl. és) CONJ címkét kap, míg a többi három elemzőeszközben CCONJ címkét (ez a helyes Universal Dependencies szófaji címke). Ehhez szükséges egy egyszerű konverzió: amikor a nyers e-magyar elemzésből legyűjti a Launcher külön listákba az eredményeket, minden CONJ helyett CCONJ legyűjtésével a probléma megoldható. A konverzió nélkül, az elemzők eredményeinek naiv összehasonlításával ez jelentősen nagy számú „False” összehasonlítást eredményezne: a tesztszövegen az e-magyart futtatva 1642 db CONJ címkét kapunk.

3.1.5. Függőségi elemzés

A függőségi elemzésben (dependenciasyntaxban) komoly problémákat okoz a két elemző különböző megközelítése. A következőkben tömören összefoglalom a függőségi elemzés alapelveit.

A függőségi elemzés működése

A függőségi elemzés a mondatok szerkezetének egyfajta leképezési módja, amelynek alapjait Lucien Tesnière fejlesztette ki 1959-ben megjelent művében. [18, 19] A napjainkban használt forma szerint a mondat szavai között felépíthető egy olyan hierarchikus kapcsolat (fa), amelyben minden szónak pontosan egy szülője – úgynevezett feje – van, és a szó és a feje közötti kapcsolat megadja a szó szintaktikai szerepét. [20] Minden mondat rendelkezik pontosan egy gyökérellemmel, amelyhez pontosan egy szó kapcsolódik ROOT címkével, így minden más elem szintaktikailag ehhez a szóhoz vagy valamely leszármazottjához csatlakozik az adott kapcsolattal. A dependenciafa egy csúcsának tetszőlegesen sok gyereke lehet. [21]

Megvalósulás az elemzőkben

Implementációs szempontból tehát egy függőségi elemzés létrejöttéhez szükséges egy címkehalmaz, amely a dependenciarelációt leírja, a ROOT címkéjű elem egyértelmű megállapítása, az elemek bekötése a megfelelő fejekhez (tehát a fa kialakítása), valamint az elem és feje közötti reláció helyes felcímkézése (lényegében a fa élcímkézése). Az elemzők kimenetét megvizsgálva kiderült, hogy a felsorolt feladatok egyikében sincs teljes elvi egyetértés a két elemző között, így viszonylag komoly átalakítások kellenek ahhoz, hogy egyáltalán értelmezhető legyen az összehasonlítás.

A címkehalmaz a két elemző esetében teljesen más. Az e-magyar által használt címkehalmaz [22] sokkal szűkebb, mint a HuSpaCy által használt, magyar nyelvre specializált Universal Dependencies címkehalmaz [23], azonban néhány esetben részletesebb annál. Emiatt a konverzió nem lehet egyértelmű egyik irányban sem. Valamilyen szintű konverzió viszont szükséges az összehasonlításhoz, ezért a Launcher tartalmaz egy általam készített konvertert a HuSpaCy címkehalmazáról az e-magyaréra, amelyben egy json szótár segítségével minden HuSpaCy-címkét egy e-magyar-címkére fordít.

A ROOT címkéjű elem megállapítása a HuSpaCy-nél a tesztek során nem mutatott különösebb anomáliát, az e-magyarnál viszont előfordult olyan mondat, amelyben nem volt ROOT címkéjű elem („(Szórványos taps a Momentum soraiban.)”). A mondat szerkezetében vagy környezetében nem volt olyan speciális jelenség, ami indokolta volna ezt a hibát, hasonló típusú mondatokat az elemző kivétel nélkül jól elemzett a tesztszövegben. Így, bár nem konzekvens a hiba, mégis azt indikálja, hogy a függőségi elemzés invariánsa meg tud sérülni az e-magyarban, amely így elvi hiba.

A dependenciafejek kiosztásánál mindkét elemző esetén megfigyelhetők további elvi hibák. A HuSpaCy-ben a tokenként értelmezett *whitespace*-csoportok szolgálhatnak fejként egyes tokenek számára, ugyanis attól még, hogy a kimenet szűrhető a fentiekben említett módon, a *pipeline*-ban benne maradnak ezek a bemenetek, és bemenetként továbbra is átlagos, normális tokenként próbálja őket értelmezni az elemző. Azon kívül, hogy ilyen *whitespace*-fej nyilvánvalóan elemzési hiba esetén fordulhat csak elő, elvi szinten kezelni kell a problémát, hiszen a megjelenítésben továbbra is zavart okoznak a sortörésként megjelenő *newline* karakterek. Ezért a Launcher ezekben az esetekben „HEAD IS WHITESPACE!” hibaiüzenettel jelzi a felhasználó számára a jelenséget a tényleges sortörések megjelenítése helyett, hogy az összehasonlítási táblázat használható maradjon.

A HuSpaCy-vel ellentétben az e-magyarban a fejek kiosztásánál nem a fejként szolgáló tokenet kapjuk vissza. Az e-magyar minden mondatban 1-től a mondat hosszáig beszámozza a tokeneket, majd minden tokenhez hozzárendeli annak a tokennek a számát, amely annak a feje. A ROOT címkéjű elem 0 címkét kap. A probléma ezzel az, hogy sajnos sok esetben nem csak a ROOT címke kap 0 címkét, hanem egyéb olyan tokenek is, amelyeket az e-magyar nem tudott értelmezni, pl. zárójelek, központoszási karakterek (amelyeket az esetek többségében viszont képes bekötni). Ezzel kapunk olyan elemeket a mondatban, amelyek nem kapcsolódnak a dependenciafához, amely szintén inkonzisztencia. Az e-magyar még egy fontos helyen eltér a Universal Dependencies szabályrendszerétől, amelyet elvileg követ: a mellérendelő kötőszavakat nem az első koordinált elemhez

köti be, hanem a másodikhoz.

3.1.6. Névelem-felismerés

A névelem-felismerés fontos és összetett feladat, ez az elemzők eredményein is látható. A Launcher kétféle megközelítéssel is támogatja a probléma megközelítését.

Az egyik lehetséges megközelítés az, hogy minden tokenről megállapítjuk, névelem-e. Erre szolgál az IOB-címkézés. [24] A HuSpaCy a nem névelemeket O címkével jelöli, a tulajdonnév első elemét B-[típus] címkével (függetlenül attól, hogy az az első elem egyben az egyetlen-e), és a tulajdonnév további névelemeit I-[típus] címkével jelöli (függetlenül attól, hogy a belső elem egyben az utolsó-e). Ehhez képest az e-magyar az összes esetet elkülöníti: O címkét pontosan az a token kaphat, amely nem névelem, B-[típus] címkét pontosan az kaphat, amely többbelemű tulajdonnév első eleme, I-[típus] címkét az, amely többbelemű tulajdonnévnek nem első és nem utolsó, de valamelyik belső eleme, E-[típus] címkét, amely többbelemű tulajdonnév utolsó eleme, végül pedig 1-[típus] címkét az, amely önmagában egy egyelemű tulajdonnév. Ez egy minimális egységesítést von maga után: a Launcher az 1-[típus] és E-[típus] címkéket átfordítja a HuSpaCy-beli megfelelőikre, így lehetővé válik az egyenlőségvizsgálat a kapott címkék között.

A másik megközelítés nem a tokenek felől, hanem a névelemek felől indul. Eszerint megkeressük az elemzővel a névelemnek ítélt tokeneket, majd ezek együttesét elemezzük. A Launcher így minden névelemnek ítélt tokenhez készít egy halmazt, amelybe gyűjti a típuscímkéket, majd ezekre vizsgál egyenlőséget. Ebben a megközelítésben elveszítjük azt az információt, hogy melyik token melyik előfordulása eredményezte az adott címkét, viszont kompakt szemléltetést kapunk egy-egy token együttes eredményéről, illetve elkülönül az eltérő címkék esetétől az a jelenség, amikor egy névelemet valamelyik elemző nem talált meg (A HuSpaCy egyébként jelentősen több névelemet talál meg, mint az e-magyar, a HuSpaCy 46635 tokenből 2536 db tokenet ítélt névelemnek a tesztfájlra, az e-magyar pedig 1946 db-ot).

3.2. Különbségek kezelése

Ahogy az eddigiekben is láthattuk, a tokenizálás minden nyelvi szinten az elemzés, illetve az összehasonlítás alapját képezi. Ezért ha az egyik elemző hibázik a tokenizálás

során, az eredmények elcsúsznak, nehezen értelmezhetővé válnak. Ezért a tokenizálásból adódó különbségek kezelése a Launcher egyik elsődleges feladata.

A tokenizálás esetén a különböző teljesítmény azt jelenti, hogy az egyik elemző egy karaktercsoportot egy tokennek ítél, míg a másik többnek. A tesztfájlból például az alábbi esetek fordultak elő:

Eltérés	HuSpaCy tokenek	e-magyar tokenek	Előfordulás (db)
Zárójelben lévő mondat utolsó szava + pont	A pont külön token	A pont nem külön token	153
„(1)”	„(”, „1”, „)”.	„(”, „1)”	7
3 szóból álló kötőjeles összetétel	Az egész kifejezés egy token	A kötőjelek és az összetételi szavak is külön tokenek (5 db token)	5
szóközzel tagolt számok	Szóközönként külön tokenek	Az egész szám egy token	5

3.1. táblázat. A 2022. 06. 20-ai országgyűlés jegyzőkönyvében leggyakrabban előforduló tokenizálási különbségek

Az összehasonlításra alkalmas táblázatok elkészítéséhez meg kell találni azt a pontot, ahol ez a problémás tokencsoport elkezdődik és véget ér, és itt néhány sor erejéig meg kell bontani a táblázatot, hogy a későbbi sorok újra összhangban lehessenek. A Launcherben ezt a következő heurisztika vezérli: ha valahol az i . és a j . token nem egyezik, akkor az egyik elemző $i + n$, a másik pedig $i + m$ tokenet kell ugorjon a másiktól függetlenül a következő szinkronizációs pontig, azaz ahol újra egyezik a két megtalált token. A fenti vizsgálatokból látszik, hogy n és m maximuma tipikusan kicsi szám, ugyanis leginkább egymást követő numerikus és központosítási karakterek, valamint kötőjelek okozzák az eltéréseket. A Launcherben jelenleg ez a maximum az 5, amely eddig – nemtipikus szövegrészleteken, pl. e-mail-címek, honlapcímek esetén is – minden esetben elegendőnek bizonyult, azonban elméletben tetszőlegesen nagy számot választhatunk ennek a konstansnak, hogy a hibázás veszélye minimális legyen.

A tokenizálási összehasonlítás explicit meg is jeleníti a két elemző által megtalált tokenek számát, a többi szinten történő összehasonlítás pedig egy rövid figyelmeztetéssel jelzi a felhasználó számára, hogy – bár a heurisztikus logika kiegyeli az elcsúszást a tesztelt esetek 100%-ában – számítani kell az elcsúszásból adódó anomáliákra. Az ilyen tokencsoportoknál az összehasonlítás lefut a nyilvánvalóan nem egyező első-első tokenre,

majd egészen addig szünetel, amíg be nem fejeződött a hosszabbik csoport. Ekkor a táblázatban az összehasonlítási eredmény („True”/„False”) helyett kijelzi, hogy melyik elemző tokenjét látjuk, majd felsorolja a többletet jelentő tokeneket az elemzéseikkel együtt. Így információ nem vész el, maunálisan lehet tovább dolgozni ezekkel az eredményekkel is. A korrekció minden elemzési szinten jelen van.

3.3. Teljesítményjavítás

A fentiekben esett már néhány szó arról, hogy a HuSpaCy integrált formában tartalmazza az e-magyarban is jelen lévő EmMorph koomponenst. Az integrációt a következő kódrészlet valósítja meg a jelenlegi hivatalos HuSpaCy-verzióban: [25]

```

1 from spacy.tokens import Doc, Token, MorphAnalysis
2 from spacy.language import Language
3 from spacy.lang.hu import Hungarian
4 from spacy.morphology import Morphology
5 from typing import List, Tuple, Dict
6 from emmorph2ud2 import EmMorph2UD2
7 from emmorphpy import EmMorphPy
8
9
10 @Hungarian.factory(
11     "emmorph",
12     requires=["token.text"],
13 )
14 def create_emmorph(nlp: Language, name: str) -> "EmMorphPyComponent":
15     return EmMorphPyComponent(nlp, name)
16
17 class EmMorphPyComponent:
18     def __init__(self, nlp: Language, name: str):
19         self._nlp = nlp
20         self._name = name
21         self._em = EmMorphPy()
22         self._converter = EmMorph2UD2()
23         Token.set_extension("em_anas", default=[], force=True)
24         Token.set_extension("em_tag", default=None, force=True)
25         Token.set_extension("em_lemma", default=None, force=True)
26         Token.set_extension("ud_tag", default=[], force=True)

```

```
27     Token.set_extension("ud_anas", default=[], force=True)
28     Token.set_extension("ud_morph", default=None, force=True)
29
30     def __call__(self, doc: Doc) -> Doc:
31         tok: Token
32         for tok in doc:
33             analyses: List[Tuple[str, str]] = self._em.stem(tok)
34             ud_analyses: List[Tuple[str, str]] = [self._converter.
35                 parse(tok, ana[0], ana[1]) for ana in analyses]
36
37             tok._.em_anas = analyses
38             tok._.ud_anas = ud_analyses
39
40             morph_dict: Dict[str, str] = tok.morph.to_dict()
41             best_sim_score = 0
42             for (em_lemma, em_tag), (ud_tag, ud_morph) in zip(
43                 analyses, ud_analyses):
44                 if ud_tag == tok.pos_:
45                     emmorph_dict: Dict[str, str] = Morphology.
46                         feats_to_dict(ud_morph)
47                     sim_score: float = sum(
48                         emmorph_dict.get(key) == value
49                         for key, value in morph_dict.items()
50                     ) / (len(
51                         set(morph_dict.keys()) | set(emmorph_dict.
52                             keys())
53                     ) or 1 )
54                     if sim_score > best_sim_score or str(tok.morph)
55                         == "" and ud_morph == "_":
56                         tok._.em_tag = em_tag
57                         tok._.ud_tag = ud_tag
58                         tok._.em_lemma = em_lemma
59                         tok._.ud_morph = MorphAnalysis(self._nlp.
60                             vocab, ud_morph)
61
62         return doc
```

3.1. forráskód. EmMorph integráció a HuSpaCy hivatalos változatában

3.3.1. Kivétel megelőzése

A futtatások során hamar kiderült, hogy az integráció nem tökéletes. A felsőfokú és túlzófokú mellékneveket tartalmazó szövegrészleteken kivétel keletkezett, ami az egész *pipeline*-t megakasztotta, és a HuSpaCy programja összeomlott. Némi vizsgálat után kiderült, hogy ezt egy konverziós hiba okozza a következő kódrészletben:

```

1  def __call__(self, doc: Doc) -> Doc:
2      tok: Token
3      for tok in doc:
4          analyses: List[Tuple[str, str]] = self._em.stem(tok)
5          ud_analyses: List[Tuple[str, str]] = [self._converter.
              parse(tok, ana[0], ana[1]) for ana in analyses]
```

3.2. forráskód. Konverziós hiba az EmMorph integrációban

A tok változó Token típusú. Ebben a kódban ez alapvetően nem problémás, azonban az 5. sorban a meghívott *parse* függvény tartalmaz egy metódust, amely Token típusra nincsen értelmezve. A függvény az *emmorph2ud2* konverter része, amely az EmMorph formátumát Universal Dependencies stílusú morfológiai elemzéssé alakítja. [26] Informatikailag azért nehéz megoldani a felső- és túlzófokú melléknevek megfelelő kezelését, mert a magyarban a felső- és túlzófok jele circumfixum típusú toldalék (azaz a szótő a toldalék belsejébe kerül, ld. „leg...bb”), ráadásul inflexiók jegy (tehát nem hoz létre új lexikai egységet, nem keletkezhet új szófaj). [27] Az EmMorph alapvető formátuma a következő: a morfológiai kódok szögletes zárójelek között, egymás után felsorolva találhatók, a szófaji kód a kapcsos zárójelen belül „/” jellel kezdődik, a derivatív jegyek „_” jellel kezdődnek, az inflexiók jegyek pedig közvetlenül a nyitó szögletes zárójel után állnak. A felsőfokú melléknevek esetében azonban az elemzés így néz ki: [/Supl]/[Adj][_Comp/Adj][Nom]. Ezért a konverter a felsőfok jelét (Supl) külön kezeli, hogy ne szófajként értelmezze a rendszer:

```

1  # a felsőfok jele szófaj volt, atkerul az inflexios jegyek koze
2  for feat in pos_feats:
3      if feat in mp.POS_TO_INFL_FEAT:
4          pos_feats.remove(feat)
5          infl_feats.append(feat)
```

3.3. forráskód. A felsőfok leválasztása az emmorph2ud2 konverterben

Majd az inflexiók jegyek kezelésénél sor kerül a következő kódrészletre:

```

1     if univpos in ('NOUN', 'ADJ', 'NUM', 'PRON', 'PROPN', 'ADV'):
2         # jegyek a mappingsbol
3         for infl_feat in infl_feats:
4             if infl_feat in mp.NOMINAL_INFL_RULES:
5                 univfeature.extend(mp.NOMINAL_INFL_RULES[
6                     infl_feat])
7
8         # ha felsőfok, akkor a középfok jegye már nem kell
9         if 'Degree=Sup' in univfeature:
10             while 'Degree=Cmp' in univfeature:
11                 univfeature.remove('Degree=Cmp')
12             if token.startswith('legesleg'):
13                 univfeature.remove('Degree=Sup')
14                 univfeature.append('Degree=Abs')

```

3.4. forráskód. A felsőfok vizsgálata string-művelettel az emmorph2ud2 konverterben

Itt pedig egy Token típusú elemre hívódik meg a `startswith` metódus, ami kivételt okoz. A probléma javítható: az EmMorph-integrációban *stringgé* konvertálva a tokent az elemzés megfelelően lefut a Token típusnak és a Python rugalmas típuskonverzióinak köszönhetően. A Launcher ezt a módosítást tartalmazza. Emellett érdemes a bemeneti tokenekből kiszűrni a *whitespace*-karaktereket is, így összességében két javítással együtt használja fel a Launcher az integrációs kódot: a sztringkonverzióval és a `tok.is_space` vizsgálattal. A továbbiakban ezt a változatot eredeti verzióknak hívom.

3.3.2. Maximumkeresés javítása

Az integrációs kódrészlet működési logikája a következő:

1. Lefut az EmMorph elemzése, visszakapjuk a fentiekben említett formátumú lehetséges elemzések listáját. Egy szóalak általában többféleképpen is elemezhető (pl. *almát* – *alma* tárgyesetben, vagy *alom* birtokos jelzővel és tárgyesetben), ekkor az összes lehetséges elemzést megkapjuk.
2. Átkonvertáljuk a kapott lista elemeit Universal Dependencies formátumra az emmorph2ud2 konverterrel.
3. Lefuttatjuk a HuSpaCy SpaCy-jellegű morfológiai elemzését, illetve készítünk belőle egy szótárat (`morph_dict`).

4. Végigiterálunk az EmMorph elemzésein, az eredeti és a konvertált változaton egyaránt.
5. Amennyiben egy elemzésnél már a szófaji címke sem egyezik a HuSpaCy által adott szófaji címkével, az elemzést eldobjuk, ugyanis ez nyilvánvalóan nem lehet a leghasonlóbb elemzés.
6. A konvertált EmMorph-elemzésből készítünk egy szótárat (`emmorph_dict`), majd összehasonlítjuk a `morph_dict` szótárral. A `morph_dict` kulcsaival keresünk az `emmorph_dict`-ben értékeket, és összehasonlítjuk, egyezik-e ugyanazon kulcs alatt a két szótárban található érték. Az EmMorph-elemzés helyességének vizsgálatához használt metrika: az egyező kulcs-érték párok száma osztva a kulcsok uniójának számosságával. Ez az arány a *similarity score* (`sim_score`), a továbbiakban hasonlósági arány.
7. Maximumkeresést végzünk a hasonlósági arányokon. A legjobb hasonlósági arányú elemzést, amennyiben nem üres, beállítjuk végeredményként.

A maximumkeresés azonban hiányos, ugyanis ha találtunk egy jobb hasonlósági arányt, nem cseréljük le a jelenlegi maximumot, hanem az megmarad az inicializált 0 értékkel. Így mindig a lista legutolsó elemzése kerül be végeredményként az elemzésbe, nem pedig a valódi legjobb.

A tesztfájlon futtatva az eredeti integrációs változatot a következő eredményeket kapjuk: 46635 tokenből 18796 esetében nem egyezik meg a HuSpaCy-EmMorph és az emagyar-EmMorph által kiadott morfológiai eredmény, a speciális karaktereket (alfanumerikus karaktereket nem tartalmazó karaktercsoportokat) leszűrve¹ 11091 esetben térnek el, amelyből 4987 esetért felel None kimenetű elemzés a HuSpaCy-EmMorphban.

A maximumkeresés javításával futtatott eredmények az alábbiak szerint javulnak: 13883 eltérés a két elemző között, speciális karakterek nélkül 6178 db (ezzel a hibák száma majdnem a felére csökkent), amelyből None kimenetű eredmény továbbra is 4987 db.

¹Megfigyelhető, hogy a None kimenetű elemzések jelentős részét ezek a speciális karakterek okozzák. Ezeket leszűrve értékesebb információt kapunk arról, hogy valójában milyen értelmes szóalakokra nem kapunk értelmes elemzést a HuSpaCy-EmMorphban.

3.3.3. A szófaji ellenőrzés mellőzése

Az értelmes (nem speciális karakterekből álló) szavakat vizsgálva felmerül a kérdés, hogy morfológiailag viszonylag egyszerű szavakra miért kapunk None eredményt. A HuSpaCy-EmMorph nyers elemzési listáiban sok ilyen szóra látunk értelmes elemzést, tehát nem az EmMorph hibája az értelmetlen eredmény. Némi vizsgálat megmutatta, hogy a gond a következő állítással van (ld. Maximumkeresés javítása, felsorolás 5. pontja): „Amennyiben egy elemzésnél már a szófaji címke sem egyezik a HuSpaCy által adott szófaji címkével, az elemzést eldobjuk, ugyanis ez nyilvánvalóan nem lehet a leg-hasonlóbb elemzés”. Ez önmagában egy igaz állítás, de feltételezi, hogy a HuSpaCy által adott szófaji címke helyes. Előfordulhat azonban, hogy a szófaji címke nem helyes, az EmMorph megtalálja a helyes morfológiai elemzést a lista valamely pontján, amelynek a hasonlósági aránya tényleg a maximális lesz, mégsem lehet végeredmény ez az elemzés, mert nem jut el a maximumkeresési logikáig a szófaji elemzés miatt. Így a HuSpaCy egy másik komponensének esetleges hibája torzíthatja ennek a komponensnek az eredményét.

Ezt a szófaji ellenőrzést eliminálva a következő javulásokat tapasztalhatjuk az elemzési eredményekben: 11 242 db eltérés az elemzők eredményei között, speciális karaktereket nem számolva 3537 db, a None kimenetű elemzések száma pedig 1202 db-ra csökken (kb. az eddigi negyedére).

A Launcher ezt a javítást is tartalmazza.

3.3.4. A 0 hasonlósági arány és az értelmetlen elemzések elválasztása

A hasonlósági arány számításakor előfordulhat, hogy az EmMorph kínálta lehetőségek közül egyik sem egyezik egyetlen kulcs-érték párjában sem a SpaCy-morph elemzésével. Ez azonban nem jelenti azt, hogy az adott szóalaknak nincsen értelmes elemzése, csak azt, hogy minden lehetséges elemzése egyformán rossz a SpaCy-morph szemszögből nézve. A Launcher a két eset között különbséget tesz: a None eseteket fenntartja az értelmetlen elemzéseknek, az egyformán rossz elemzésekből pedig „sorsol” egyet: a listában utolsóként szereplő eredményt állítja be végeredménynek. Ezzel már nullánál nagyobb a valószínűsége annak, hogy rossz SpaCy-morph elemzés esetén mégis jó EmMorph-elemzés kerül a végeredménybe.

A tesztbemeneten futtatva ezt a javított verziót azt látjuk, hogy a HuSpaCy és az emagyar kevesebb esetben különbözik, 10028 eltérést látunk, speciális karaktereket mellőzve 3129 db-ot, amelyből 321 db None kimenet. Ez természetesen - ahogyan az eddigi-

ekben sem - nem zárja ki annak a lehetőségét, hogy mindkét elemző egyszerre hibázzon és ugyanazt a rossz megoldást válassza, ennek ellenőrzése azonban már nyelvészeti feladat, kézzel annotált, gold standard tesztadatbázist igényel, amelyet a dolgozatnak nem célja elkészíteni. Az mindenesetre biztos, hogy a 321 db None eredmény most már pontosan azokat az eseteket jelöli, amikor az EmMorph által adott lehetséges elemzések listája üres lista. Így az integrációs kód változtatása ebből a szempontból nem tudja tovább csökkenteni az értelmetlen elemzések számát, ugyanis az üres listákat valamely, az integráció által felhasznált másik program hibája okozza.

Összefoglalva tehát az eddigi összes változtatást, a Launcher jelenlegi legjobb változatában a következő módosított integráció üzemel:

```

1 from spacy.tokens import Doc, Token, MorphAnalysis
2 from spacy.language import Language
3 from spacy.lang.hu import Hungarian
4 from spacy.morphology import Morphology
5 from typing import List, Tuple, Dict
6 from emmorph2ud2 import EmMorph2UD2
7 from emmorphpy import EmMorphPy
8
9 @Hungarian.factory(
10     "emmorph",
11     requires=["token.text"],
12 )
13 def create_emmorph(nlp: Language, name: str) -> "EmMorphPyComponent":
14     return EmMorphPyComponent(nlp, name)
15
16 class EmMorphPyComponent:
17     def __init__(self, nlp: Language, name: str):
18         self._nlp = nlp
19         self._name = name
20         self._em = EmMorphPy()
21         self._converter = EmMorph2UD2()
22         Token.set_extension("em_anas", default=[], force=True)
23         Token.set_extension("em_tag", default=None, force=True)
24         Token.set_extension("em_lemma", default=None, force=True)
25         Token.set_extension("ud_tag", default=[], force=True)
26         Token.set_extension("ud_anas", default=[], force=True)
27         Token.set_extension("ud_morph", default=None, force=True)
28

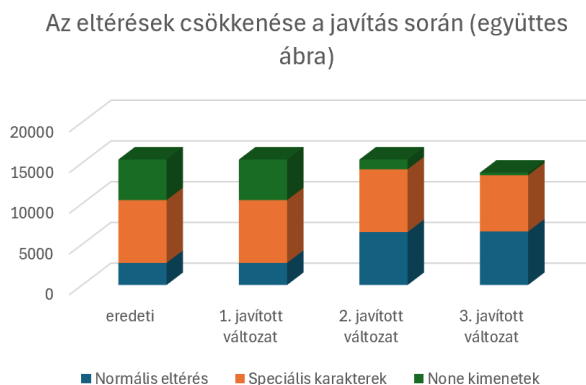
```

```

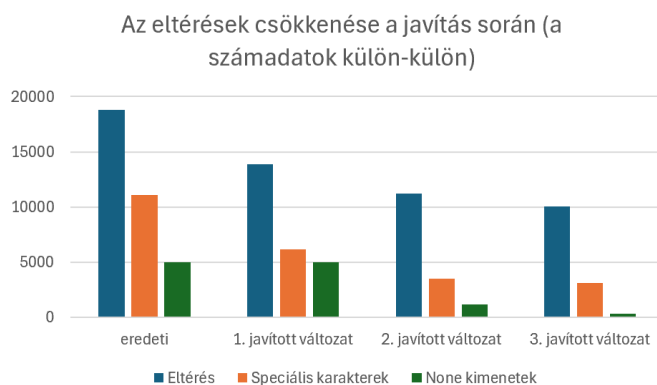
29     def __call__(self, doc: Doc) -> Doc:
30         tok: Token
31         for tok in doc:
32             if not tok.is_space:
33                 analyses: List[Tuple[str, str]] = self._em.stem(tok
34                     )
35                 ud_analyses: List[Tuple[str, str]] = [self.
36                     _converter.parse(str(tok), ana[0], ana[1]) for
37                     ana in analyses]
38
39                 tok._.em_anas = analyses
40                 tok._.ud_anas = ud_analyses
41
42                 morph_dict: Dict[str, str] = tok.morph.to_dict()
43                 best_sim_score = 0
44                 for (em_lemma, em_tag), (ud_tag, ud_morph) in zip(
45                     analyses, ud_analyses):
46                     #if ud_tag == tok.pos_:
47                     emmorph_dict: Dict[str, str] = Morphology.
48                         feats_to_dict(ud_morph)
49                     sim_score: float = sum(
50                         emmorph_dict.get(key) == value
51                         for key, value in morph_dict.items()
52                     ) / (len(
53                         set(morph_dict.keys()) | set(emmorph_dict.
54                             keys())
55                     ) or 1 )
56                     if sim_score > best_sim_score or str(tok.morph)
57                         == "" and ud_morph == "_" or best_sim_score
58                         == 0:
59                         tok._.em_tag = em_tag
60                         tok._.ud_tag = ud_tag
61                         tok._.em_lemma = em_lemma
62                         tok._.ud_morph = MorphAnalysis(self._nlp.
63                             vocab, ud_morph)
64                         best_sim_score = sim_score
65
66         return doc

```

3.5. forráskód. Az általam módosított HuSpaCy-EmMorph integráció végleges változata



3.1. ábra. Együttes oszlopdiagram az eltérés csökkenésének folyamatáról



3.2. ábra. Oszlopdiagram az eltérés csökkenésének folyamatáról, az adatok szerint külön oszlopokkal

4. fejezet

Összefoglalás és eredmények

Bár a `Launcher_huspacy_emagyar` szoftver egy nyelvészeti kutatásokat segítő szoftverfejlesztési feladatban gyökerezik, a munka során számos olyan információra fény derült a `HuSpaCy` és `e-magyar` elemzőkről, amelynek komoly nyelvészeti vonzata van, azonban az eddigiekben olyan informatikai támogatás, automatikus korrekció vagy explicit figyelemfelhívás nem történt, amely célzottan a két elemző egymáshoz viszonyított szempontjából közelítette volna meg a problémát. A `Launcher_huspacy_emagyar` mellett, hogy ezt a hiányosságot tudja pótolni, a `HuSpaCy` esetében a teljesítmény javítására is kísérletet tesz a morfológiai elemző integrációjának módosításával. Az integráció konverzióhiba-javítása elengedhetetlenek bizonyult ahhoz, hogy a `HuSpaCy` integrált `EmMorph`-komponense immár összeomlás nélkül lefuthasson tetszőleges bemeneti szövegre, az egyéb javítások pedig drasztikusan le tudták csökkenteni az értelmetlen `None` kimenetek számát. A nyelvtchnológus közösség számára a javítás ténye és az összehasonlítási lehetőség megteremtése elengedhetetlen, de a felhasználóbarát megközelítésnek köszönhetően más területeken dolgozó nyelvészeknek is hasznára válhatnak az alkalmazáson keresztül elérhető összehasonlítási funkciók.

Köszönetnyilvánítás

Köszönöm a rengeteg támogatást és segítséget témavezetőmnek, Dr. Ligeti-Nagy Noéminek, akitől életem egyik legnagyobb szerűbb lehetőségét kaptam azáltal, hogy ebbe a kutatómunkába becsatlakozhattam. Szeretnék köszönetet nyilvánítani a HUN-REN Nyelvtudományi Kutatóközpontnak, amiért befogadtak szakmai gyakorlatra és ez a projekt elindulhatott, hálás vagyok Héja Enikőnek és Ligeti-Nagy Noéminek az ehhez nyújtott segítségükért. Köszönöm továbbá a folyamatos szakmai és szakmán kívüli támogatást az Eötvös József Szakkollégiumnak, az Informatikai Műhely tagjainak, és különösen mentoromnak, Szabó Barbarának, aki ebben a félévben végigkísért ezen az úton.

A. függelék

Szemelvények a tesztszövegből

A.1. Szövegrészlet, amelyben előfordul időbélyeg

Nagyon jelentősnek tartom azt, és ez a mai helyzetben napjainknak is az egyik alapkérdése, hogy mindazokat a stratégiai elemeket, legyenek azok az energiatermelésre, energiatárolásra, energiaelosztásra alkalmas létesítmények és cégek, a magyar állam visszavásárolta.

(13.10)

Visszavásároltuk vagy magyar kézben tartottuk azokat a cégeket is, amelyek a magyar gazdaság számára létfontosságúak.

A.2. Példa hibás helyesírású alakokra

Végiggondoltáke önök, hogy hogyan befolyásolná a hazánkba érkező befektetéseket, ha duplájára nőne a vállalati adószint Magyarországon? Kérdezem, hogy végeztek-e ezzel kapcsolatban számításokat.

Végiggondoltáke önök azt, hogy az EU bevezetné ezt az adót, a világ más gazdaságilag erős térségei pedig nem? Végiggondoltáke, hogy ez a javaslat kedvező lehet a magas adószintű országoknak, ami Magyarország versenyelőnyét csökkentené?

A.3. Jegyzőkönyvi szövegelemek és előbeszédbeli bekiabálások

ELNÖK: Most Tordai Bencének adom meg a szót, parancsoljon!

TORDAI BENCE (Párbeszéd): (A mikrofonja nem szól.) Köszönöm a szót. Tisztelt Országgyűlés! Kedves Polgártársak! Engedjék meg, hogy külön köszöntsem török vendégeinket Finike városából.

Tegnap volt apák napja, és úgy tűnik, hogy ez az ünnep pártokon és politikai oldalakon átívelő támogatást élvez, hiszen nemcsak... (Közbeszólások az ellenzéki pártok padsoraiból, köztük: Nincs hang! Közbeszólások a kormánypártok padsoraiból, köztük: Nem baj! Jó így! Jó lesz így is!)

ELNÖK: A sajátját tette föl, valami baj van. (Tordai Bence elé hordozható mikrofont helyeznek.)

TORDAI BENCE (Párbeszéd): Akkor kérném az óra visszaállítását.

ELNÖK: Legyenek szívesek, kezdjük előlről!

Irodalomjegyzék

- [1] György Orosz és tsai. „HuSpaCy: an industrial-strength Hungarian natural language processing toolkit”. *XVIII. Magyar Számítógépes Nyelvészeti Konferencia*. Szeged, 2022, 59–73. old.
- [2] György "Orosz és tsai. „Advancing Hungarian Text Processing with HuSpaCy: Efficient and Accurate NLP Pipelines””. *Text, Speech, and Dialogue*. Szerk. Kamil "Ekštejn, František Pártl és Miloslav" Konopík. "Cham": "Springer Nature Switzerland", 2023, 58–69. old. ISBN: 978-3-031-40498-6.
- [3] Eszter Simon és tsai. „Újabb fejlemények az e-magyar háza táján”. *XVI. Magyar Számítógépes Nyelvészeti Konferencia*. Szerk. Gábor Berend, Gábor Gosztolya és Veronika Vincze. Szeged: Szegedi Tudományegyetem Informatikai Tanszékcsoport, 2020, 29–42. old.
- [4] Balázs Indig és tsai. „One format to rule them all – The emtsv pipeline for Hungarian”. *Proceedings of the 13th Linguistic Annotation Workshop*. Florence, Italy: Association for Computational Linguistics, 2019. aug., 155–165. old. DOI: 10.18653/v1/W19-4018. URL: <https://www.aclweb.org/anthology/W19-4018>.
- [5] Balázs Indig és tsai. „emtsv — Egy formátum mind felett”. *XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2019)*. Szerk. Gábor Berend, Gábor Gosztolya és Veronika Vincze. Szeged: Szegedi Tudományegyetem Informatikai Tanszékcsoport, 2019, 235–247. old.
- [6] Tamás Váradi és tsai. „E-magyar – A Digital Language Processing System”. Angol. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Szerk. Nicoletta Calzolari (Conference chair) és tsai. Miyazaki, Japan: European Language Resources Association (ELRA), 2018. máj. ISBN: 979-10-95546-00-9.

- [7] Tamás Váradi és tsai. „Az e-magyar digitális nyelvfeldolgozó rendszer”. *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2017)*. Szerk. Gábor Berend, Gábor Gosztolya és Veronika Vincze. Szeged: Szegedi Tudományegyetem Informatikai Tanszékcsoport, 2017, 49–60. old.
- [8] Attila Novák, Borbála Siklósi és Csaba Oravecz. „A New Integrated Open-source Morphological Analyzer for Hungarian”. Angol. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Szerk. Nicoletta Calzolari és tsai. Portorož, Slovenia: European Language Resources Association (ELRA), 2016. máj. ISBN: 978-2-9517408-9-1.
- [9] Attila Novák. „A New Form of Humor – Mapping Constraint-Based Computational Morphologies to a Finite-State Representation”. Angol. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*. Szerk. Nicoletta Calzolari és tsai. Reykjavik, Iceland: European Language Resources Association (ELRA), 2014. máj. ISBN: 978-2-9517408-8-4.
- [10] György Orosz és Attila Novák. „PurePos 2.0: a hybrid tool for morphological disambiguation”. *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2013)*. Hissar, Bulgaria: INCOMA Ltd. Shoumen, 2013, 539–545.
- [11] Dávid Márk Nemeskey. „Egy emBERT próbáló feladat”. *XVI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2020)*. Szeged, 2020, 409–418. old.
- [12] Gergely Pethő és tsai. „Igekötő-kapcsolás”. *XVIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2022)*. Szerk. Gábor Berend, Gábor Gosztolya és Veronika Vincze. Szeged, 2022, 77–91. old.
- [13] URL: <https://github.com/huspace/huspace> (elérés dátuma 2024. 12. 01.).
- [14] URL: https://polc.ttk.pte.hu/tamop-4.1.2.b.2-13/1-2013-0014/2/37%5C_a%5C_jelents%5C_a%5C_szmgtgpek%5C_szmra.html (elérés dátuma 2024. 12. 01.).
- [15] URL: <https://spacy.io/usage/linguistic-features%5C#how-tokenizer-works>, (elérés dátuma 2024. 12. 01.).
- [16] URL: <https://www.inf.u-szeged.hu/~rfarkas/ML20/NLP.html> (elérés dátuma 2024. 12. 01.).

- [17] URL: http://omagyarkorpusz.nytud.hu/documents/papers/chapters/simonsass-chapter%5C_%5C_nyelvtech%5C_es%5C_kult%5C_orokseg.pdf (elérés dátuma 2024. 12. 01.).
- [18] URL: https://en.wikipedia.org/wiki/Lucien%5C_Tesni%5C%C3%5C%A8re (elérés dátuma 2024. 12. 01.).
- [19] Prószték Gábor. *Számítógépes nyelvészet*. Számítástechnika-alkalmazási Vállalat, 1989.
- [20] URL: <http://sandbox.hlt.bme.hu/%5C~makrai/oktatas/nytud/nlp22/7-domotor-synt.pdf> (elérés dátuma 2024. 12. 01.).
- [21] URL: <https://universaldependencies.org/u/overview/syntax.html> (elérés dátuma 2024. 12. 01.).
- [22] Veronika Vincze és tsai. „Hungarian Dependency Treebank”. *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*. Szerk. Nicoletta Calzolari és tsai. European Language Resources Association, 2010. URL: <http://www.lrec-conf.org/proceedings/lrec2010/summaries/465.html>.
- [23] URL: <https://universaldependencies.org/hu/index.html> (elérés dátuma 2024. 12. 01.).
- [24] URL: [https://en.wikipedia.org/wiki/Inside%5C%E2%5C%80%5C%93outside%5C%E2%5C%80%5C%93beginning%5C_\(tagging\)](https://en.wikipedia.org/wiki/Inside%5C%E2%5C%80%5C%93outside%5C%E2%5C%80%5C%93beginning%5C_(tagging)) (elérés dátuma 2024. 12. 01.).
- [25] URL: <https://github.com/huspace/huspace/blob/master/docs/recipes/emmorph.md> (elérés dátuma 2024. 12. 01.).
- [26] Noémi Vadász és Eszter Simon. „Konverterek magyar morfológiai címke-készletek között”. *XV. Magyar Számítógépes Nyelvészeti Konferencia*. Szeged: Szegedi Tudományegyetem TTIK, Informatikai Intézet, 2019, 99–111. old. ISBN: 9789633153932.
- [27] URL: https://polc.ttk.pte.hu/tamop-4.1.2.b.2-13/1-2013-0014/2/10%5C_feladat3.html (elérés dátuma 2024. 12. 01.).

Ábrák jegyzéke

3.1. Együttes oszlopdiagram az eltérés csökkenésének folyamatáról	22
3.2. Oszlopdiagram az eltérés csökkenésének folyamatáról, az adatok szerint külön oszlopokkal	22

Táblázatok jegyzéke

3.1. A 2022. 06. 20-ai országgyűlés jegyzőkönyvében leggyakrabban előforduló tokenizálási különbségek	13
---	----

Forráskódjegyzék

3.1. EmMorph integráció a HuSpaCy hivatalos változatában	14
3.2. Konverziós hiba az EmMorph integrációban	16
3.3. A felsőfok leválasztása az emmorph2ud2 konverterben	16
3.4. A felsőfok vizsgálata string-művelettel az emmorph2ud2 konverterben . .	17
3.5. Az általam módosított HuSpaCy-EmMorph integráció végleges változata	20