Újabb fejlemények az e-magyar háza táján

Simon Eszter, Indig Balázs, Kalivoda Ágnes, Mittelholcz Iván, Sass Bálint, Vadász Noémi

MTA Nyelvtudományi Intézet
Budapest, Benczúr u. 33.
{VEZETÉKNÉV.KERESZTNÉV}@nytud.mta.hu

Kivonat A cikkben az e-magyar nyelvfeldolgozó eszközlánc új verziójában, az emtsv-n végrehajtott fejlesztéseket mutatjuk be. Az emtsv fő tulajdonságai közé tartozik a teljes modularitás, amit az egységes formátum és keretrendszer tesz lehetővé. Ebből következik, hogy az emtsv-be könnyen lehet új modulokat integrálni, valamint az egyes elemzési lépéseknél be- és kiszállni. Ezt illusztrálandó egyrészt már létező eszközöket integráltunk (UDPipe, Hunspell), másrészt új modulokat fejlesztettünk (emTerm, emDiff, emZero), harmadrészt a már meglévő modulokat fejlesztettük tovább (detokenizálási funkció az emToken-ben). A cikkben ezeket mutatjuk be, továbbá az emtsv-t teljesítmény és gyorsaság szempontjából összehasonlítjuk hasonló funkcionalitásokkal bíró magyar nyelvfeldolgozó eszközláncokkal, mint a UDPipe, a huspaCy és a Magyarlánc. Az emtsv LGPL 3.0 licenc alatt elérhető a https://github.com/dlt-rilmta/emtsv GitHub repozitóriumból.
Kulcsszavak: e-magyar, emtsv, eszközlánc, erőforrás, tsv, modularitás

1. Bevezetés

Az e-magyar nyelvfeldolgozó rendszer (Váradi és mtsai, 2017) a fejlesztésekor elérhető state-of-the-art (SOTA) eszközöket integrálta egy egységes, könnyen kezelhető, fenntartható és fenntartott eszközláncba. Fontos célja volt, hogy az elkészült eszközlánc a magyar nyelv kutatás- és alkalmazásközpontú felhasználását egyaránt elősegítse, továbbá hogy moduláris és nyílt legyen. Az e-magyar első változatának keretrendszerét a GATE (Cunningham és mtsai, 2011) szolgáltatta, így a modulok közötti kötőszövetet is a GATE belső XML formátuma teremtette meg (Sass és mtsai, 2017). Ennek számos hátránya derült ki a felhasználók visszajelzései nyomán, ezért az e-magyar-nak egy új verziója lett kifejlesztve emtsv néven (Indig és mtsai, 2019b). Az emtsv fontos új tulajdonságait Indig és mtsai (2019a) és Indig és mtsai (2019b) mutatják be – ezek közül itt csak a legfontosabbakat emeljük ki: egységes formátum egy egységes keretrendszerben az xtsv segítségével, teljeskörű modularitás, az elemzőláncba való beszállás és az abból való kiszállás lehetősége a lánc bármely pontján, új modulok egyszerű integrálhatósága, valamint felhőalapú technológiák alkalmazása.

Jelen cikkben az emtsv legújabb fejlesztéseit mutatjuk be. A 2. fejezetben az xtsv-t írjuk le, ami nem csak egy egységes adatformátum, hanem az egész

elemzőlánc keretrendszerét szolgáltatja. A 3. fejezetben az új modulokat, illetve a már meglévő modulokon eszközölt új fejlesztéseket ismertetjük. A 4. fejezetben az `emtsv` könnyebb felhasználhatóságát lehetővé tevő felhőalapú technológiákról szólunk. Az 5. fejezet az `emtsv` egy konkrét projektbeli alkalmazását mutatja be. A 6. fejezetben az `emtsv`-t hasonló szövegfeldolgozó láncokkal vetjük össze teljesítmény és gyorsaság szempontjából. A cikket a 7. fejezetben összegzés és a jövőbeli tervek ismertetése zárja.

2. xtsv

Az `xtsv` keretrendszert az eredetileg az `emtsv`-hez kialakított, de nagyon általános, kiterjeszthető formátum és a formátumot kezelő API együttesen alkotja. Az `emtsv` számára Indig és mtsai (2019b) specifikálták a szükséges API-t és formátumot. Ezt úgy általánosítottuk tovább, hogy az `emtsv` nem nyelvi elemzést végző, általánosítható részeit külön csomagba szerveztük és egységesítettük. Így egy általános keretrendszer jött létre, ami minden, az API-t támogató modullal használható. Ez az `xtsv`, ami elérhető pip csomagként[1] és LGPL 3.0 licenc alatt a GitHubon is[2].

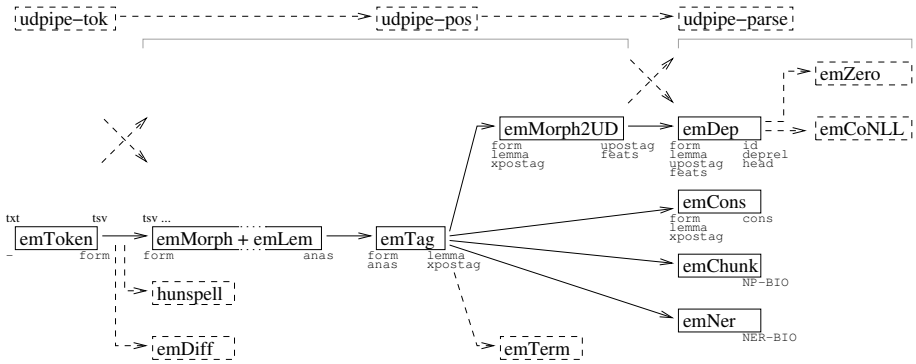
Az `xtsv`-ben használt formátum szembetűnő hasonlóságokat mutat a CoNLL-U Plus formátummal[3], mely a CoNLL-U formátumot[4] kívánja kiterjeszthetővé tenni a kompatibilitás megtartásával. A CoNLL-U Plus viszont egyelőre híján van implementált keretrendszernek.

Az új keretrendszer jellemzői közé tartozik (1) a Javát használó modulok egyszerű konfigurálhatósága, (2) a modulok lusta betöltése (csak azok a modulok töltődnek be, amikre az aktuális futtatás során tényleg szükség van), illetve (3) modulok tetszőleges sorozatának (ún. *preset*eknek) könnyű definálhatósága.

3. Új fejlesztések és modulok

Az `xtsv` lehetővé teszi tetszőleges számú újabb modul beépítését a rendszerbe, amely lehetőséggel éltünk is. Az 1. ábrán láthatjuk a rendszer felépítését; szaggatott vonal jelöli azokat a kiegészítő modulokat, amelyek nem voltak részei az `e-magyar` eszközlánc első verziójának. Az első verzió moduljai: `emToken` tokenizáló, `emMorph` + `emLem` morfológiai elemző és lemmatizáló, `emTag` POS tagger, `emMorph2UD` a morfológiai elemző kimeneti formátumáról a Universal Dependencies (UD) formátumára való átalakítást végző modul, `emDep` dependenciaelemző, `emCons` konstituenselemző, `emChunk` főnévicsoport-felismerő és `emNer` tulajdonnévfelismerő. Ebben a fejezetben az új fejlesztéseket mutatjuk be.

[1] `https://pypi.org/project/xtsv/`
[2] `https://github.com/dlt-rilmta/xtsv`
[3] `https://universaldependencies.org/ext-format.html`
[4] `https://universaldependencies.org/format.html`



1. ábra: Az `emtsv` feldolgozó lánca, a bemeneti és kimeneti mezőkkel. A nyilak értelmezése: minden modul előfeltételezi azt a modulsort, ahonnan hozzá nyíl vezet, de bármely későbbi ponton is lefuttatható.

3.1. UDPipe

A UDPipe (Straka és Straková, 2017) egy olyan eszközlánc, amely CoNLL-U formátumú fájlok elemzését végzi bármilyen nyelvű bemeneti szövegen, amelyen létezik CoNLL-U formátumú tanítóanyag. Az elemzési szintek, amiket megvalósít: tokenizálás, POS taggelés[5] és dependenciaelemzés. A tokenizálásnak természetesen folyó szöveg is lehet a bemenete, de azután az egységes átmeneti formátumot a CoNLL-U adja. Magyarra az egyetlen CoNLL-U formátumú annotált korpusz a Universal Dependencies oldalán található korpusz[6], amely a Szeged Dependency Treebanknek (Vincze és mtsai, 2010) egy kb. 42.000 tokenes része. Ez van train-devel-test alkorpuszokra felosztva, 50-25-25 százalékos arányban. A UDPipe magyar modelljei ezen a tanítókorpuszon lettek tanítva.

Fontosnak tartottuk a széles körben használt szövegfeldolgozó eszközláncok integrálását is az `emtsv`-be, ezért beépítettük a UDPipe-ot, amit annak nyílt forráskódja is támogatott. A UDPipe által megkülönböztetett három szint három önálló modulként jelenik meg az `emtsv`-ben, így ezek bármelyikén be és ki lehet lépni a láncból, és kombinálni is lehet az azonos célú modulokat. (Ezt jelzik az 1. ábrán a keresztező nyilak.) A kényelmes használat kedvéért megoldottuk, hogy a UDPipe által végzett három feladatból kettőt (udpipe-tok-pos, udpipe-pos-parse) vagy az összeset (udpipe-tok-parse) egy lépésben is lehessen futtatni. A kombinálhatóság tekintetében azonban figyelembe kell venni, hogy a magyar nyelvű UD treebank morfológiai címkekészlete a UD 2-es verzióját követi, míg az `emDep` még a UD 1-es verzióját használja. A két verzió között morfológiai

[5] A cikkben egységesen POS taggelésnek hívjuk azt az elemzési szintet, amelynek a kimenete a tokenhez rendelt egyértelmű és teljes morfológiai elemzés (lemma, szófaji címke és inflexiós jegyek).

[6] `https://github.com/UniversalDependencies/UD_Hungarian-Szeged`

szinten alig van különbség[7], de azért számolni kell vele. A két dependenciaelemző kimenete viszont eltér egymástól, mivel az `emDep` a teljes Szeged Dependency Treebanken lett tanítva, ami nem lett átkonvertálva a UD címkekészletére.

3.2. Detokenizálás

Az `e-magyar` első verziójában használt `emToken` tokenizáló eredetileg XML és JSON kimenetet produkált, és ezekben megőrizte a szóköz jellegű karaktereket is. Ezáltal biztosította azt az információt, amivel a kimenet egyértelműen detokenizálható volt.

A `emtsv` egységesen minden modultól az `xtsv` kimenetet várja el, ami egy `tsv`-n alapuló formátum, minden tokent külön sorban ábrázol, és a mondathatárokat üres sorokkal jelzi. Ezért a detokenizálhatóságot ebben az új formátumban másképp kellett implementálni. A kimenet első oszlopa (`form`) tartalmazza a tokeneket, míg a szóköz jellegű szekvenciák megőrzésére új oszlopot vezettünk be (`wsafter`). Amellett döntöttünk, hogy a teljes szekvenciát megőrizzük idézőjelek között, így a kimenet szabad szemmel történő áttekintése során is egyértelmű, hogy hol van és hol nincs szóköz a tokenek után. A szóköz jellegű karaktereket a C nyelvben megszokott escape sorozatokkal ábrázoljuk (`\n`, `\r`, `\t`, `\v` és `\f`).

3.3. Hunspell

A UDPipe-hoz hasonlóan egy másik széles körben ismert eszközt, a Hunspellt[8] is integráltuk az `emtsv`-be. A Hunspell egy nyílt forráskódú helyesírás-ellenőrző, lemmatizáló és morfológiai elemző, amely elsősorban a magyarra és hasonló gazdag morfológiájú nyelvekre lett kifejlesztve. A Hunspell a LibreOffice, OpenOffice.org, a Mozilla Firefox és a Google Chrome beépített helyesírás-ellenőrzője, de zárt programcsomagok is használják. Egy önálló modulként lett az `emtsv`-be integrálva, és – ahogy az 1. ábrán is látható – elkülönül a többi modultól abban az értelemben, hogy a kimenete nem adható át bemenetként egy másik modulnak. Ennek az az oka, hogy a Hunspell egy egyedi morfológiai címkekészletet használ[9], amelyre alkalmazható konverter jelenleg nem elérhető.

3.4. `emDiff`: összehasonlító és kiértékelő modul

A modul célja, hogy két `xtsv` formátumú fájl az `emtsv` egyes moduljai által biztosított elemzési rétegek mentén összehasonlítható legyen. A modul háromféle feladat megoldására alkalmas:

- egyszerű összevetésre,
- az egyik szöveget gold standardként tekintve kiértékelési feladatokra, és
- annotátorok közötti egyetértés számolására.

[7] `https://universaldependencies.org/v2/summary.html`
[8] `http://hunspell.github.io/`
[9] `https://github.com/laszlonemeth/magyarispell/`

Az egyszerű összevetés esetében az `emDiff` csak a szóalakok szintjén végez összevetést, így a tokenizálók kimenetei közötti különbségeket ragadhatjuk meg (pl. az `emToken` és a `udpipe-tok` modul összevetésekor); ehhez a Python `difflib` csomagját[10] használja. A kiértékelési feladatok esetében a két fájl közül az egyiket gold standardnak tekintjük, és ahhoz hasonlítjuk a másik fájl tartalmát. Az `emDiff` a különböző elemzési rétegeket eltérő módszerekkel értékeli ki, így például a tulajdonnév-felismeréshez pontosságot, fedést és F-mértéket számol, míg a függőségi elemzéshez Labeled Attachment Score (LAS) és Unlabeled Attachment Score (UAS) értékeket. Annotátorok közötti egyetértést a címkézési feladatok esetében az `nltk.metrics` csomag[11] által biztosított metrikák (S (Bennett és mtsai, 1954), π (Scott, 1955), κ (Cohen, 1960) és α (Krippendorff, 1980)) alapján számolhatunk. Az `emDiff` minden modul kimenetére használható, vagyis az elemzőláncnak minden ízületére illeszthető (lásd az 1. ábrán).

3.5. `emTerm`: kifejezésannotáló modul

Az `emTerm` tetszőleges forrásból származó, azonosítóval bíró, egy- vagy többszavas kifejezéseket ismer fel és annotál tokenizált, mondatokra bontott és POS taggelt bemeneten. A modulnak kétféle bemenetre van szüksége: az elemzendő szövegre és egy `tsv` fájlra, amely a szövegben jelölendő egy- vagy többszavas kifejezéseket tartalmazza az azonosítójukkal együtt. Egy sorban egy kifejezés szerepelhet, a hozzá tartozó azonosítóval. A többszavas kifejezések szavait @ szimbólum választja el egymástól. A bemenetre egy példa látható az 1. táblázatban.

azonosító	**kifejezés**
`1116804-04,2821`	`etnikai@kisebbség`
`1116832-24,3206`	`képzési@alap`
`47639-52`	`katonai@légi@forgalom`
`47674-1236,28,2821`	`kisebbség`

1. táblázat. Példa az `emTerm` bemenetére. A példában IATE kifejezések és azonosítóik szerepelnek (lásd az 5. fejezetet).

A modul tervezésekor abból indultunk ki, hogy a kifejezések nem nyúlhatnak át mondathatáron, ezért az annotálás mondatnyi egységeken történik. Az annotálást egy egyszerű algoritmus végzi, amely

1. kinyeri a mondatban található összes olyan tokenszekvenciát, amelynek a hossza nem haladja meg a leghosszabb keresendő kifejezés hosszát (pl. a *süt a nap* a következő szekvenciákra tagolódik: *süt*, *süt a*, *süt a nap*, *a*, *a nap*, *nap*);

[10] `https://docs.python.org/3/library/difflib.html`
[11] `https://www.nltk.org/api/nltk.metrics.html`

2. minden tokenszekvenciát olyan formájúra alakít, hogy kereshető legyen a kifejezések között: a szekvencia utolsó tokenjének a szótövét őrzi meg, a többi tokennek pedig a felszíni alakját;
3. ha egy átalakított szekvencia megtalálható a jelölendő kifejezések között, elvégzi a találat annotálását.

A megtalált kifejezések annotációja egy új oszlopba kerül. Az annotáció formátuma `találat_sorszáma:azonosító`. A találatok számozása a mondat elején 1-gyel kezdődik. A többszavas kifejezések esetében az első szónál adjuk meg a teljes annotációt, a kifejezés későbbi szavainál csak a találat sorszámát ismételjük. Ha egy token több kifejezésnek is a része, a találatokat pontosvessző választja el egymástól. Amennyiben az adott token nem része egy keresett kifejezésnek sem, az annotáció cellájába alulvonás kerül. Az annotációs formátumot a 2. táblázat szemlélteti.

ID	token	lemma	szófaj	kifejezés
10	az	az	DET	_
11	etnikai	etnikai	ADJ	`1:1116804-04,2821`
12	kisebbségekre	kisebbség	NOUN	`1;2:47674-1236,28,2821`
13	vonatkozó	vonatkozó	ADJ	_

2. táblázat. Példa az `emTerm` kimenetére. `1`-es sorszámmal szerepel az *etnikai kisebbség* kifejezés, `2`-es sorszámmal pedig a külön azonosítóval rendelkező *kisebbség*. Az azonosítók IATE kifejezések azonosítói (lásd az 5. fejezetet).

Mivel az `emTerm` modul POS taggelt bemenetet igényel, az `emTag` után illeszthető be az elemzőláncba (lásd az 1. ábrán). Ez azt jelenti, hogy a többszavas kifejezések annotálása után bármelyik modul irányába továbbléphetünk újabb nyelvi annotációk hozzáadása céljából.

3.6. `emZero`: zérónévmás-beszúró modul

A modul bemenete a tokenizált, POS taggelt és függőségi elemzéssel ellátott szöveg. Az `emZero` egy szabályalapú rendszer, ami a Szeged Dependency Treebank morfológiai és függőségi annotációján alapul, vagyis az eszközláncnak egy pontjára illeszkedik, az `emDep` mögé (lásd az 1. ábrát).

A program a következő elemeknek a helyére illeszt be zérónévmást:

– finit ige alanyának, ha annak nem volt testes alanya;
– határozott ragozású finit ige tárgyának, ha annak nem volt testes tárgya;
– birtok birtokosának, ha annak nem volt testes birtokosa;
– ragozott és ragozatlan infinitívusz alanyának.

A program egyszerű szabályok mentén végzi az elemek beillesztését, melyek alkalmazása során különböző elemzési rétegek tartalmára támaszkodik (lemma, szófaji címke, függőségi elemzés).

A zérónévmások beillesztése után a mondatfában plusz ágak jelennek meg. A zéró elemek anélkül kapnak saját ID-t, hogy a függőségi elemző által kiosztott ID-ket megváltoztatnák. A kimenetben az alany az ige után, a tárgy az ige és a zéró alany után, a birtokos pedig a birtok után jelenik meg, és egy kombinált ID-t kap, ami az őt megelőző elem ID-jéből és a zéróelem szintaktikai szerepének rövidítéséből (SUBJ, OBJ, POSS) áll. Szófajuk névmás (PRON) lesz, a morfológiai jegyeik között pedig az ige vagy a birtok alapján kiszámolható szám és személy jegyek jelennek meg. A zéró elemek alanyként, tárgyként vagy birtokosként elfoglalják helyüket a mondatfában – erre láthatunk egy példát a 2. ábrán.

```
1       Főként          ADV    3  MODE
2       ötvözőelemként  NOUN   3  OBL
3       használják      VERB   0  ROOT
3.SUBJ  ZERO            PRON   3  SUBJ
3.OBJ   ZERO            PRON   3  OBJ
4       .               PUNCT  0  PUNCT
```

2. ábra: Egy testetlen alany és egy testetlen tárgy beillesztése a függőségi elemzésbe.

4. Felhasználási módok

Komoly elvárás az újonnan megjelenő rendszerek esetében a könnyű futtathatóság és a szolgáltatásként történő könnyű telepítés és skálázódás. Az `emtsv`-ben ezt úgy kívánjuk elérni, hogy az `xtsv`-be belefejlesztettünk egy úgynevezett futtatható Docker módot, amivel az egy paranccsal letölthető Docker image parancssori alkalmazásként használhatóvá válik, ahogy egy `.jar` fájl esetében történne. Ezenfelül lehetőség van a Docker image használatával (vagy nélküle) a REST API-n kereszül elérhető szolgáltatásként elindítani az `emtsv`-t (vagy bármilyen `xtsv`-re alapuló programot), ami a Docker ökoszisztéma elterjedtségének köszönhetően jól skálázható és beépíthető különféle felhős munkamenetekbe.

Szükségesnek tartottuk, hogy a felhasználók a lehető legkevesebb technikai képzettséggel is képesek legyenek használni az `xtsv`-re épülő szolgáltatást. Ezért a keretrendszer része egy főleg demózás és prototipizálás céljára használható webes felhasználói felület, mely segítségével sztenderd HTML interfészen keresztül lehet iterakcióba lépni a szerelőszalaggal. Mivel a háttérben ugyanazt a REST API-t használja a felület, ugyanúgy alkalmas nagy adatok feldolgozására is, mint sok felhasználó által beadott sok kis adat egyidejű kezelésére.

5. Alkalmazás

Az `e-magyar` nyelvelemző rendszert közvetlenül hasznosítjuk az EU által támogatott, hét ország részvételével zajló MARCELL projektben[12]. A projekt célja, hogy elemzett korpuszokat állítson elő a bolgár, a horvát, a lengyel, a magyar, a szlovák, a szlovén és a román nemzeti joganyagból. Terveink szerint ezek a korpuszok *in domain* tanítóadatként fognak szolgálni az EU gépi fordító rendszere számára, melynek fő feladata éppen a különféle jogi szövegek automatikus fordítása.

A jogi szövegek feldolgozása bizonyos esetekben nehezebb a köznapi magyar szövegekhez képest. A feldolgozólánc – elsősorban a függőségi elemző – számára kihívást jelentenek a jogi szövegekben előforduló nagyon hosszú (akár 5.000 szavas!) mondatok. Ezek legtöbbször hosszú felsorolásokat tartalmaznak például kinevezésekről, kitüntetésekről vagy egy másik országgal megkötött vámmegállapodás tételeiről.

A projekt keretében készült el a tokenizáló modul detokenizálási funkciója (lásd a 3.2. fejezetet). Ennek segítségével tudjuk szolgáltatni a megkívánt `SpaceAfter=No` annotációt, mely azt jelzi, hogy az adott token után nem volt szóköz az eredeti szövegben.

Szintén a projekt elvárása volt, hogy megjelöljük a szövegben két jogi terminológiai adatbázis entitásait: az egyik a IATE (InterActive Terminology for Europe)[13], egy többnyelvű terminológiai adatbázis, amelyet az európai intézmények a fordításokhoz használnak; a másik az EUROVOC[14], amely az EU-ban kifejlesztett és rendszeresített, elsősorban jogi fogalmakat tartalmazó tezaurusz. Ezekben az adatbázisokban minden egyes kifejezésnek saját azonosítója van – ezek jelennek meg azonosítóként az `emTerm` bemeneti listájában és persze a kimentében is, ahogy a 2. táblázatban is láthatjuk. A IATE és EUROVOC kifejezések annotálását az `emTerm` modul segítségével, a 3.5. fejezetben leírtak alapján végezzük el.

6. Kitekintés és összehasonlítás

Az `emtsv` szövegfeldolgozó eszközláncnak természetesen léteznek alternatívái, amelyek ki tudják váltani bizonyos funkcionalitásait. Ebben a fejezetben ezeket mutatjuk be és hasonlítjuk össze az `emtsv`-vel az alábbi paraméterek mentén: teljesítmény, gyorsaság, nyelvfüggetlenség, ki- és beszállási lehetőség az egyes lépéseknél, új modulok integrálhatósága és könnyű használhatóság. Ki- és beszállás alatt azt értjük, amikor a felhasználó az elemzőláncot csak egy adott lépésig, például a POS taggelésig használja, az automatikus kimenetet kézzel ellenőrzi, majd a megfelelő formátum betartásával az elemzőláncba vissza tud szállni az immár javított annotációval. Ennek elsősorban az olyan felhasználói

[12] `https://marcell-project.eu/`

[13] `https://iate.europa.eu/home`

[14] `https://op.europa.eu/en/web/eu-vocabularies/th-dataset/-/resource/dataset/eurovoc`

forgatókönyvekben lehet jelentősége, ahol fontos a nyelvi annotáció pontossága, hogy elkerüljük a halmozódó hibákat a felsőbb elemzési szinteken.

Az `emtsv`-nek egy alternatívája a már ismertetett UDPipe (lásd a 3.1. fejezetet), ami nyelvfüggetlen és könnyen használható. Az teszi könnyen használhatóvá, hogy csak le kell emelni a polcról a kész tanítóanyagot vagy a már előre tanított modellt. Viszont ugyanez a hátránya is: nagy mértékben függ a tanítóanyagtól. A UDPipe, annak ellenére, hogy a forráskódja szabadon elérhető, nem ad lehetőséget új, saját modulok, például szabályalapú elemzők beépítésére. A modularitás egy másik feltételét, mégpedig azt, hogy az egyes lépéseknél ki- és be lehessen lépni az elemzőláncba, viszont teljesíti, amit az egységes CoNLL-U formátum tesz lehetővé.

Hasonló eszköz a spaCy[15], ami kifejezetten ipari alkalmazásra ajánlott, ezért a könnyű használat és a gyorsaság kritikus fontosságú. Modulárisnak mondható abban az értelemben, hogy támogatja új moduloknak a láncba való integrálását, és az is megoldható, hogy az egyes lépéseknél ki- és be lehessen lépni a láncba. Nyelvfüggő abban az értelemben, hogy ha nincsenek előállítva a megfelelő modellek és erőforrások egy adott nyelvre, akkor arra nem használható. A hivatalos repozitóriumban a magyar még nem szerepel, de számos modell letölthető Orosz György GitHub repozitóriumából[16].

A Magyarlánc (Zsibrita és mtsai, 2013) áll legközelebb az `emtsv`-hez abban az értelemben, hogy mindkettő kifejezetten a magyar nyelvre lett kifejlesztve. A Magyarlánc egy Java-alapú, zártan integrált lánc, amely jelenleg a 3.0 verziójánál tart. A rendszer zártságából fakad, hogy az egyes lépéseknél nem lehet ki- vagy beszállni, és nem lehet új modulokat hozzáadni. Ebből (is) következik, hogy teljesen nyelvfüggő, csak a magyarra működik. Zártságának előnye viszont, hogy könnyen használható és viszonylag gyors. A rendszer a legfrissebb SOTA eszközöket tartalmazza, amelyek részben átfednek az `emtsv` egyes moduljaival.

6.1. Teljesítmény

A teljesítmény kiértékeléséhez a 3. táblázat nyújt segítséget. A morfológiai egyértelműsítés és a lemmatizálás esetében accuracy, a dependenciaelemzés esetében Unlabeled Attachment Score (UAS) és Labeled Attachment Score (LAS), a tulajdonnnév-felismerés (NER) esetében pedig F-mérték a táblázatban megadott mérőszám.

Az utolsó oszlop tartalmazza a SOTA eszközök teljesítményét. Ez az oszlop vonatkozik a Magyarlánc és az `emtsv` teljesítményére is, mivel mindkettőre igaz az, hogy a magyarra elérhető SOTA eszközök lettek beépítve, és ezek az eszközök megegyeznek: a PurePos (Orosz és Novák, 2012) és a dependenciaelemzést végző Bohnet parser (Bohnet és Nivre, 2012) a Szeged Dependency Treebanken tanítva. A morfológiai egyértelműsítés és a lemmatizálás számai a Magyarláncba beépített PurePos teljesítményét mutatják, és magukban foglalják a teljes morfológiai címkézést a lemmatizálással együtt. A dependenciaelemzés számai is a Ma-

[15] `https://spacy.io/`

[16] `https://github.com/oroszgy/spacy-hungarian-models/`

gyarláncba beépített elemző teljesítményét mutatják. A tulajdonnév-felismerés (NER) cellája üres a UDPipe esetében, mert a UD treebankekben nem szerepel tulajdonnévi annotáció. A SOTA szám a NER esetében Simon (2013) eredményein alapul, és a Szeged NER korpusz (Szarvas és mtsai, 2006) 90%-án lett tanítva és 10%-án tesztelve.

A UDPipe fejlesztői kiértékelték az egyes modulok teljesítményét[17] – ezek láthatók az első oszlopban. A UD 2.4-es verziójú treebankjein tanított modellek teljesítményét vettük figyelembe, ezek közül is azokat, amelyeknek nyers szöveg volt a bemenete, mivel a többi elemzőlánc esetében is ez a kezdeti bemenet. A morfológiai egyértelműsítés sorába azt a számot másoltuk át, amely a minden morfológiai címkét tartalmazó (AllTags) elemzés mérőszáma. A UDPipe a UD annotációs sémáját és formátumát követő fájlokkal tud csak dolgozni, ezért a 3. táblázatban látható nem túl magas számok valószínűleg annak köszönhetőek, hogy a magyar nyelvre elérhető egyetlen tanítóanyag nem túl nagy (lásd a 3.1. fejezetet).

feladat	UDPipe	huspaCy	SOTA
morf. egyértelműsítés (ACC)	86,40	94,91	96,33
lemmatizálás (ACC)	88,50	95,49	96,33
dependenciaelemzés (UAS)	72,70	76,18	93,22
dependenciaelemzés (LAS)	67,10	66,58	91,42
NER (F1)	-	93,95	96,10

3. táblázat. Az összehasonlításban részt vevő rendszerek teljesítménye.

A második oszlop a magyar spaCy eredményeit tartalmazza, Orosz György mérései alapján[18]. Azt meg kell jegyeznünk, hogy a morfológiai egyértelműsítésnél szereplő szám itt csak a szófaji címkékre vonatkozik, nem a teljes morfológiai elemzésre. A tulajdonnév-felismerő modul a Szeged NER és a Szeged Criminal NE korpuszok[19] 90%-án, valamint a magyar Hunnerwiki korpuszon (Nemeskey és Simon, 2012) lett tanítva, és a Szeged NER és a Szeged Criminal NE korpuszok 10%-án lett tesztelve. A spaCy weboldalán erőteljesen hangsúlyozzák, hogy a gyorsaság mellett a teljesítmény is megmarad, de a magyar spaCy eredményei – különösen a dependenciaelemzésben – a SOTA eszközök teljesítménye alatt maradnak.

6.2. Gyorsaság

A fenti rendszereket összehasonlítottuk sebesség szempontjából is. Sebesség alatt azt értjük, hogy az adott rendszer egy másodperc alatt hány tokenhez bocsátja

[17] `http://ufal.mff.cuni.cz/udpipe/models`
[18] `https://tinyurl.com/y4ole3ul`
[19] `https://rgai.sed.hu/node/130`

ki a megfelelő szintű nyelvi annotációt. Minden elemzővel kétfajta mérést végeztünk: folyó szöveg feldolgozása POS taggelésig, illetve folyó szöveg feldolgozása dependenciaelemzésig. A méréseket ugyanazon a 100.000 tokenes fájlon végeztük el[20] minden esetben ötször, és az eredményeket átlagoltuk. Az olvasás és írás műveletekhez *RAM disk*-et használtunk, a programok inicializálási idejét figyelmen kívül hagytuk. A mérésekhez a rendszereknek a cikk írásának idejében elérhető legfrissebb verzióját használtuk, és egy erősebb teljesítményű asztali gépen futtattuk.[21] Az eredmények a 4. táblázatban láthatóak.

elemző	POS	dependencia
`emtsv` (CLI)	2.320	300
`emtsv` (REST)	2.600	310
Magyarlánc	5.550	450
UDPipe	9.280	3.300
huspaCy	33.980	15.000

4. táblázat. Az összehasonlításban részt vevő rendszerek sebessége (token/másodperc).

Jól látszik, hogy a huspaCy és a UDPipe mindkét versenyszámban jelentősen gyorsabb, mint a többi program. Sőt, a huspaCy-ról nyugodtan állíthatjuk, hogy nagyságrendekkel gyorsabb, mint a versenytársai. Az `emtsv`-nek a kliensszerver (REST) módban való futtatása nagyobb teljesítményt biztosít, mint a parancssoros (CLI) használata. Az `emtsv` Docker verziója körülbelül 300 token/másodperccel teljesít rosszabbul a táblázatban feltüntetett értéknél a POS taggelésben, és körülbelül 20 token/másodperccel a dependenciaelemzésben.

6.3. Összehasonlítás, diszkusszió

A paraméterek, amelyek mentén az egyes rendszerek összehasonlítását végeztük, az alábbiak: teljesítmény, gyorsaság, nyelvfüggetlenség, ki- és beszállási lehetőség az egyes lépéseknél, új modulok integrálhatósága és könnyű használhatóság. Mindezen paramétereket már részletesen kifejtettük a megelőző fejezetekben, itt egy összefoglaló táblázatban a teljes képet tekintjük át. Az 5. táblázatban nyújtjuk a fentebb leírtak mátrixos ábrázolását.

A könnyű használhatóság mindegyik rendszerre igaznak bizonyult, így annak megkülönböztető ereje nincsen. Az egyes elemzési szinteknél való be- és kilépési lehetőség mindegyik rendszerre áll, kivéve a Magyarláncot, ami teljesen zárt ebből a szempontból. A nyelvfüggetlenséget szigorúan véve igazából egyik rendszer sem teljesíti, de a UDPipe minden olyan nyelvre használható, amire létezik UD treebank, ami jelenleg 90 nyelvet jelent, ami messze felülmúlja az összes általunk

[20] Kivéve a huspaCy-t, ami túl gyors volt ahhoz, hogy megbízható eredményt adjon 100.000 tokenen, ezért itt 1.000.000 tokenen végeztük ezt a mérést.

[21] CPU: 4 mag, 4 GHz; RAM: 16 GB

	teljesítm.	gyors.	nyelvfügg.	ki-be	integrálh.	használh.
`emtsv`	O	X	X	O	O	O
Magyarlánc	O	X	X	X	X	O
UDPipe	X	O	O	O	X	O
huspaCy	X	O	X	O	O	O

5. táblázat. A rendszerek összehasonlító táblázata a vizsgált paraméterek mentén (teljesítm. = teljesítmény, gyors. = gyorsaság, nyelvfügg. = nyelvfüggetlenség, ki-be = ki és belépési lehetőség, integrálh. = új modulok integrálhatósága, használh. = könnyű használhatóság). O-val jelöljük azt, ha a rendszer az adott paraméter megvizsgálásából pozitívan jön ki, X-szel, ha nem.

ismert, magát nyelvfüggetlennek állító rendszer teljesítményét. A teljesítmény és a gyorsaság fordított arányosságban tűnik lenni, vagyis nincs olyan rendszer, ami egyszerre valósítja meg ezt a két kontradiktórius elvárást.

7. Összegzés és jövőbeli tervek

A cikkben az `e-magyar` újabb verzióján, az `emtsv`-n végrehajtott újabb fejlesztéseket mutattuk be. Az újonnan fejlesztett modulok, reményeink szerint, még több jövőbeli kutatáshoz tudnak segítséget nyújtani. A fejlesztések során kifejezetten szem előtt tartottuk a bölcsészet- és társadalomtudományok művelőit is, hogy számukra is praktikus alkalmazásokat hozzunk létre.

A teljes elemzőlánc elérhető LGPL 3.0 licenc alatt a rendszer GitHub repozitóriumában[22]. Mivel az `e-magyar` folyamatos fejlesztés alatt áll, cikkünkben a fejlesztés folyamatának csak egy pillanatképét tudjuk adni. Ebből az következik, hogy a fenti repozitóriumot érdemes újra és újra felkeresni.

Az `xtsv` keretrendszer megalkotásakor szükségünk volt egy olyan egységes formátumra, ami a kötőszövetet tudja alkotni az egyes modulok között, és nem kizárólag belső szabvány, hanem igazodik a nemzetközileg használt, elterjedt formátumokhoz is. Az egyik ilyen a cikkben többször is említett CoNLL-U, aminek több hátránya is mutatkozott, ezért dolgoztuk ki az `xtsv` formátumot. Időközben a Universal Dependencies közösség fejlesztői kidolgozták a CoNLL-U-nak egy kiterjesztett verzióját, a CoNLL-U Plust, ami számos tulajdonságában megegyezik az `xtsv`-vel, így ha minden paraméterében megfelel, lehet, hogy az `e-magyar`-t átállítjuk a CoNLL-U Plus formátum használatára.

Hosszabb távú terveink között szerepel egy olyan nagyméretű tanítókoprusz létrehozása, amely a UD v2-es formátumát és címkekészletét követi. Ezzel képesek lennénk teljesen UD-kompatibilis dependenciaelemzés kibocsátására. Hozzátesszük, hogy ez a teljes magyar nyelvtechnológia számára nagyon fontos előrelépés lenne.

[22] `https://github.com/dlt-rilmta/emtsv`

Köszönetnyilvánítás

A kutatást az Európai Bizottság CEF Telecom programjában nyertes 2017-EU-IA-0136 számú MARCELL nevű projektje támogatta.

Hivatkozások

Bennett, E.M., Alpert, R., Goldstein, A.C.: Communications Through Limited-Response Questioning. Public Opinion Quarterly 18(3), 303–308 (1954)

Bohnet, B., Nivre, J.: A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. pp. 1455–1465. EMNLP-CoNLL '12, Association for Computational Linguistics, Stroudsburg, PA, USA (2012)

Cohen, J.: A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20(1), 37–46 (1960)

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M.A., Saggion, H., Petrak, J., Li, Y., Peters, W.: Text Processing with GATE (Version 6) (2011)

Indig, B., Sass, B., Simon, E., Mittelholcz, I., Vadász, N., Makrai, M.: One format to rule them all – the `emtsv` pipeline for Hungarian. In: Proceedings of the 13th Linguistic Annotation Workshop. pp. 155–165. Association for Computational Linguistics, Florence, Italy (2019a)

Indig, B., Sass, B., Simon, E., Mittelholcz, I., Kundráth, P., Vadász, N.: `emtsv` — egy formátum mind felett. In: Berend, G., Gosztolya, G., Vincze, V. (szerk.) XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2019). pp. 235–247. Szegedi Tudományegyetem Informatikai Tanszékcsoport, Szeged (2019b)

Krippendorff, K.: Content Analysis: An Introduction to its Methodology. Sage, Beverly Hills, CA (1980)

Nemeskey, D.M., Simon, E.: Automatikus korpuszépítés tulajdonnév-felismerés céljára. In: Tanács, A., Vincze, V. (szerk.) IX. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2013). pp. 106–117. Szeged (2012)

Orosz, Gy., Novák, A.: PurePos — an open source morphological disambiguator. In: Sharp, B., Zock, M. (szerk.) Proceedings of the 9th International Workshop on Natural Language Processing and Cognitive Science. p. 53–63. Wroclaw (2012)

Sass, B., Miháltz, M., Kundráth, P.: Az `e-magyar` rendszer GATE környezetbe integrált magyar szövegfeldolgozó eszközlánca. In: Vincze, V. (szerk.) XIII. Magyar Számítógépes Nyelvészeti Konferencia. pp. 79–90 (2017)

Scott, W.A.: Reliability of content analysis: The case of nominal scale coding. The Public Opinion Quarterly 19(3), 321–325 (1955)

Simon, E.: Approaches to Hungarian Named Entity Recognition. Ph.D.-értekezés, PhD School in Cognitive Sciences, Budapest University of Technology and Economics (2013)

Straka, M., Straková, J.: Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In: Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. pp. 88–99. Association for Computational Linguistics, Vancouver, Canada (2017)

Szarvas, Gy., Farkas, R., Felföldi, L., Kocsor, A., Csirik, J.: A highly accurate Named Entity corpus for Hungarian. In: Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06). pp. 1957–1960. ELRA (2006)

Váradi, T., Simon, E., Sass, B., Gerőcs, M., Mittelholcz, I., Novák, A., Indig, B., Prószéky, G., Farkas, R., Vincze, V.: Az `e-magyar` digitális nyelvfeldolgozó rendszer. In: Vincze, V. (szerk.) XIII. Magyar Számítógépes Nyelvészeti Konferencia. pp. 49–60 (2017)

Vincze, V., Szauter, D., Almási, A., Móra, Gy., Alexin, Z., Csirik, J.: Hungarian Dependency Treebank. In: Proceedings of LREC 2010. ELRA, Valletta, Malta (May 2010)

Zsibrita, J., Vincze, V., Farkas, R.: magyarlanc: A Toolkit for Morphological and Dependency Parsing of Hungarian. In: Proceedings of RANLP. pp. 763–771 (2013)