Vállalati rendszerbe integrálható természetesnyelv-feldolgozó alkalmazás készítése digital-twin-distiller platformmal

Orosz Tamás[1], Csányi Gergely Márk[1,4], Gadó Krisztián[1], Üveges István[1], Vági Renátó[1,2], Vadász János Pál[1,3], Nagy Dániel[1]

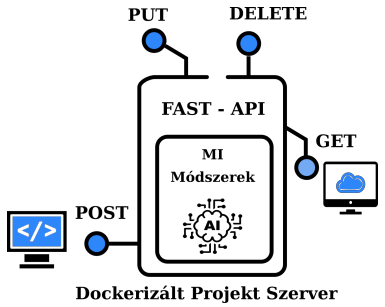
[1]MONTANA Tudásmenedzsment Kft.
[2]Eötvös Lóránd Tudományegyetem
[3]Nemzeti Közszolgálati Egyetem
[4]Budapesti Műszaki és Gazdaságtudományi Egyetem
{orosz.tamas,csanyi.gergely,gado.krisztian,uvegesi,vagi.renato,vadasz.pal,nagy.daniel}-@montana.hu

Kivonat Nagy méretű, mesterséges intelligenciát vagy egyéb numerikus módszert használó projektek esetén a modellek megalkotásán és finomhangolásán kívül számos kihívást jelentő feladat van, amelyeknek a megoldása egyre nehezebbé válik a projekt méretének és a projekten dolgozó mérnökök, kutatók számának a növekedésével. A cikkben egy olyan számítási platformot mutatunk be, amelynek az segítségével a benne készített numerikus modellek egy parancs kiadásával automatikusan átalakíthatóak egy vállalati rendszerbe integrálható, szabványos REST-API-t tartalmazó alkalmazássá. A cikk bemutatja az eszköz fontosabb részeit, és néhány rövid nyelvtechnológiai példán keresztül szemlélteti a kapcsolódó eszközöket.

1. Bevezetés

Ipari környezetben alkalmazott mesterséges intelligencia megoldásokat alkalmazó projektek fejlesztése során fontos kérdés, hogy az ügyfél igényeinek megfelelő, a célrendszerbe könnyen integrálható alkalmazást hozzunk létre. Az időnyomás és az infrastruktúra kialakítása fontos kérdés, amelyeknek az architekturális megalapozása és elkészítése komolyabb tervezést és erőforrásokat igényelhet, mint magának a szövegbányászati modellnek a betanítása. A természetesnyelv-feldolgozást támogató eszközözök közül a `spaCy` az, amelyik ipari alkalmazások létrehozását is támogató, template-szerű megoldásokat is tartalmaz `spaCy` (Schmitt és mtsai, 2019). Azonban, ahogy az a keretrendszer honlapján is olvasható (Honnibal és Montani, 2017), egy keretrendszer használata esetén előfordulhat, hogy saját platformszerű környezetet kell kialakítani, a különböző igényeket kielégítő, különböző eszközök kombinálásával. Ez a feladat megoldása során nagyobb szabadsági fokot biztosít a programozóknak, azonban lassabb, és kevésbé támogatja a fejlesztést, mint egy számítási platformszerű megoldás, ahol ezek

az igények egyetlen pontból, szorosan integrált módon elégíthetők ki (smartface, 2021). A `digital-twin-distiller` egy szövegbányászatot és numerikus modellezési feladatok fejlesztési folyamatát támogató számítási platform (Orosz és mtsai, 2021b,a). A Netflix által fejlesztett Metaflow-hoz hasonlóan (Ahmad (2021)), csoportmunkát és felhőben végzett számításokat automatikusan támogató, nyílt forráskódú Python-csomag `digital-twin distiller`[1]. Mindkét projekt azt használja ki, hogy a mesterséges intelligencia alapú, szövegbányászati, képfelismerési vagy egyéb komplex mérnöki problémák megoldása során sok az ismétlődő (rész)feladat (Orosz és mtsai, 2021a). A Metaflow ezeket a megoldási lépéseket egy gráfként reprezentálja, amelynek az egyes lépései újrahasznosíthatóak. A `digital-twin-distiller`-ben a feladatok egy fastruktúrába képezhetőek le, majd a tanítás után automatikusan deployolhatóak is a beépített REST-API-n keresztül (1. ábra). Egyes felmérések szerint, ennek a feladatnak a megoldása a projekt teljes idejének körülbelül a 10%-át veszi igénybe (Hecht, 2019).



1. ábra. API megvalósítása `digital-twin distiller` framework segítségével.

A `digital-twin distiller`-ben készített projektek automatikusan szállíthatók egy dockerizált (Merkel és mtsai, 2014) környezetbe csomagolva, ahol a végfelhasználó egy szerverként tudja futtatni a megvalósított digitiális ikerként funkcionáló alkalmazást.

A digitális iker fogalmát eredetileg a gyártási folyamatok fejlesztése során alkották meg, ez azonban az utóbbi években átalakult, kitágult, immár többféle definíciója létezik. Az élő és élettelen entitások olyan digitális replikációját jelentik, ahol a termék teljes életciklusa alatt keletkezett adatokat, numerikus szimulációkat, vagy akár a mesterséges intelligencián alapuló nyelvi modelleket tartalmazza (Vogel-Heuser és mtsai, 2021; Rassolkin és mtsai, 2021).

A dockerizálásnak köszönhetően, az így előállított digitális ikrek később, akár évek múlva is futtathatóak lesznek, a rendszer mélyebb ismerete nélkül. A vállalati kompetencia fluktuációja, a vendor lock-in és a szoftverkörnyezet változása

[1] https://github.com/montana-knowledge-management/digital-twin-distiller (2021.11.09.)

miatt van előnye a fent vázolt alkalmazásnak az ipari környezetben történő alkalmazás esetén.

A vállalati kompetencia nem egy állandó érték, hanem időről időre változik, ahogyan a munkavállalók feladatkört váltanak, vagy elhagyják a munkáltatójukat. Minél gyorsabban elsajátíthatók a vállalati rendszer egyes eszközei, annál könnyebben biztosítható a folytonos üzletmenet. Hiszen, egy termelési láncba állított kód esetén nem engedhető meg, hogy azt kizárólag abban az esetben lehessen megbízhatóan működtetni, ha a teljes kifejlesztéshez szükséges kompetencia rendelkezésre áll.

A vendor lock-in az előzőhöz hasonló problémakör, csak ez esetben nem valaminek a hiánya, hanem az egyes szoftverek, szoftver komponensek adott szolgáltatóhoz vagy gyártóhoz való kötöttsége miatt állhat elő az a helyzet, hogy a vállalat nem ura a saját kódjának, ezért kiszolgáltatottá válik az üzletmenet terén.

A népszerű természetesnyelv-feldolgozásra alkalmas keretrendszerek, mint a Gensim, a spaCy vagy a Keras elérhetőek a distillerben történő fejlesztés során is, azonban ezen könyvtárak mellett egyéb pluginként használható, természetesnyelv-feldolgozást támogató könyvtárak is használhatók. A következő fejezetekben bemutatjuk a `digital-twin distiller` keretrendszerben jelenleg elérhető főbb modulokat, plugineket.

2. I/O funkciók támogatása

Egy projekt megvalósítása során az egyik legalapvetőbb, és egyben leggyakrabban ismétlődő feladat a projekt során használt adatok fájlból beolvasása és fájlba írása. A `digital-twin distiller` keretrendszer mind beolvasás, mind fájlba írás esetében többféle formátumot is támogat, melyeket az 1. táblázat mutat be.

Támogatott formátum	Modulok		
	Reader	Writer	Data Snapshot
json	O	O	O
txt	O	O	X
pdf	O	X	X
zip	X	X	O

1. táblázat. A `digital-twin distiller` által támogatott formátumok. Jelölés: O: jelenleg támogatott, X: jelenleg még nem támogatott

A framework támaszkodik az Apache Tika[2] szoftverre, amely segítségével lényegében bármely szöveges formátum könnyedén beolvasható. Alapértelmezettként a pdf, txt és json fájlok beolvasására alkalmas osztályok érthetők el. A projektek életciklusa során gyakori feladat a különféle előfeldolgozási lépések

[2] https://tika.apache.org/ (Elérés: 2021.11.12.)

(pl. lemmatizálás, tokenizálás, szótövezés, POS-taggelés, egyéb adatok kinyerése stb.) elvégzése, amelyek igen időigényesek is lehetnek. A hatékonyabb fejlesztés célját szem előtt tartva a `digital-twin distillerben` implementáltunk egy `DataSnapshot()` névre hallgató osztályt, amely egy zip fájlba képes json kiterjesztésű adatok listáját kimenteni, illetve onnan betölteni. Ezzel egyszerűen megvalósítható egy hosszadalmas előfeldolgozás utáni adat gyors betöltése és további használata, amellyel jelentős idő takarítható meg.

3. Modellek mentése

Bizonyos gépi tanulási módszereket implementáló Python könyvtárak, mint a `scikit-learn` (Pedregosa és mtsai, 2011) nem biztosít modellmentési lehetőséget a keretrendszeren belül, így legtöbbször ezeket a modelleket `pickle` (Van Rossum, 2020), vagy `joblib` (Team, 2020) formátumban szokták a felhasználók elmenteni. A Python `pickle` modulja azonban kockázatot jelent a gépi tanulási modellek üzemeltetésével kapcsolatban, hiszen az adott modellt az összes osztályváltozójával együtt menti el. Ha az osztály egyik adattagja egy refaktorálás során megváltozik, akkor az pl. egy frissített Python könyvtárral már nem használható újra, így a modell tanításával töltött idő elvész.

A `digital-twin distiller`, az AGPL v3 licenszes `sklearn2json` modult[3] használja a `scikit-learn` modellek mentésére. Ez egy különálló, önmagában is használható modul, amely egy json formátumú fájlba menti az adatokat, így pl. egy frissítés során az osztályváltozók nevének változása esetén ez az egy helyen karbantartott csomag képes ezeket a változásokat lekezelni, így újra felhasználhatóvá tenni őket. Az `sklearn2json` emellett számos osztály mentésére kínál lehetőséget, legyen az klasszifikáló-, regressziós problémára alkalmazható-, klaszterező- vagy vektorizáló osztály.

A probléma kezelésére léteznek más csomagok is, például az `sklearn2pmml`[4], amelyet egy kisebb észt startup fejleszt, ez a megoldás azonban egy Java kódból generált Python API-val használható. Egyre nagyobb nemzetközi figyelmet és támogatottságot tudhat maga mögött az ONNX projekt[5], amely a különböző keretrendszerekben tanított modellek portabilitásáért felel, így nagyvállalatok által fejlesztett ipari standarddá is válhat. Az onnx, mint közös fájlformátum lehetővé teszi a gépi tanulási fejlesztők számára, hogy a modelleket különböző keretrendszerekkel, és eszközökkel használhassák.

4. Augmentálás

Habár napjainkban rengeteg digitális adat keletkezik, mégis a gépi tanulást is alkalmazó projektek esetében nem ritka, hogy az adott (rész)feladat megoldásához nincsen elegendő mennyiségű tanítóadat. Ilyen esetekben gyakran alkal-

[3] https://github.com/montana-knowledge-management/sklearn2json
[4] https://github.com/jpmml/sklearn2pmml (2021.11.12.)
[5] https://github.com/onnx/onnx

mazott megoldás valamilyen adatnövelő, más szóval augmentáló eljárás alkalmazása, amely során a meglevő adatok mellé különböző eljárásokkal szintetikusan előállított tanítóadatot lehet képezni. Az így kapott kibővített adathalmaz már jellemzően alkalmas pontosabb modellek tanítására. Fontos megjegyezni, hogy a szintetikusan előállított adatok minősége jellemzően elmarad a valós adatokétól, sok esetben mégis jelentős pozitív hatással vannak a modellek teljesítményére.

Szöveges adatok esetében az augmentálási módszereket két nagyobb csoportra oszthatjuk aszerint, hogy magát a szöveget módosítjuk, augmentáljuk (pl. szavak törlése, szinonimák hozzáadása, cseréje stb.), vagy pedig annak vektorreprezentációját (pl. SMOTE (Chawla és mtsai, 2002)). Ezek közül a `digital-twin distiller` jelenleg csak az előbbi típust támogatja, erre azonban több különböző megoldást is kínál.

Fontos megjegyezni, hogy az augmentálás során előfordulhat olyan eset, hogy bizonyos kifejezéseket nem szeretnénk, hogy módosítson az augmentáló algoritmus. Ezért elláttuk az összes jelenleg elérhető augmentálási módszerünket egy **védett szólista** megadásának lehetőségével.

A következő alfejezetekben sorra vesszük a `digital-twin distiller` által jelenleg támogatott augmentálási módszereket.

4.1. Easy Data Augmentation (EDA) (Wei és Zou, 2019)

Az ún. Easy Data Augmentation (EDA) négy egyszerű eljárást takar. Ezek a szinonima csere (Synonym replacement, SR), véletlenszerű törlés (Random deletion, RD), véletlenszerű szó szinonimájának beszúrása a szövegbe (Random insertion, RI), és a szövegben előforduló szavak véletlenszerű cserélgetése (Random swap, RS). A szinonimákat az eredeti megoldásban angol nyelvű WordNet segítségével keresik meg (Wei és Zou, 2019), a mi eszközünk pedig alapértelmezettként a magyar nyelvet támogatja a magyar WordNetre építve (Miháltz és mtsai, 2008).

Valamennyi megoldás esetén beállítható a művelet elvégzésének valószínűsége tokenenként, vagy a művelet elvégzésének száma.

Wei és Zou (2019) rövid szövegeken bizonyította az eljárások hatékonyságát, de természetesen ez nagyban függ a konkrét alkalmazási területtől is.

4.2. Szóbeágyazás alapú augmentálás

Egy tipikus augmentási megoldás a szemantikai hasonlóság alapján történő augmentálás (Csányi és Orosz, 2021). Ennek során véletlenszerűen kiválasztott szavakat egy betanított nyelvi modell segítségével lecserélünk a hozzájuk leghasonlóbb szavakra, ezzel az eredeti mondathoz lényegileg hasonló, mégis új adatot hozva létre.

Ilyen típusú augmentálás megvalósítására alkalmasak pl. a különböző szóbeágyazási modellek, amilyen a Word2Vec (Mikolov és mtsai, 2013), GloVe (Pennington és mtsai, 2014) vagy a FastText (Bojanowski és mtsai, 2017).

A keretrendszerben lehetővé tettük az ilyen eljárással történő augmentálást, melyet kutatási célra már alkalmaztak is (Üveges és mtsai, 2022). Az augmentálás során itt is beállítható, hogy milyen eséllyel legyen kiválasztva augmentálásra egy adott szó, valamint, hogy egy-egy szó cseréje során hány darab, hozzá leginkább hasonló más szóból történjen a kiválasztás. Utóbbi esetén szintén megadható, hogy ezekből a mintavételezés súlyozottan vagy teljesen egyenletes eloszlást követve történjen-e, esetleg az algoritmus minden esetben a leghasonlóbb szót válassza ki helyettesítésre.

Mivel ezek a szóbeágyazási modellek nem alkalmasak a kontextus figyelembe vételére, így egy adott szóhoz a leghasonlóbb szavakból építhető egy szótár (Python dictionary), amely az adott korpusz összes különböző szavához tartalmazza a legközelebbi vektorreprezentációval bíró szavakat. Ezen szótár fájlba mentésével jelentős időt lehet megtakarítani, hiszen nem szükséges minden alkalommal a szóbeágyazási modelltől lekérdezni a leghasonlóbb szavakat, illetve ha egy szó többször is előfordul a szövegben, úgy a lekérdezést szintén elég egy alkalommal elvégezni.

Jelenleg a keretrendszer által a `fasttext` és a `gensim` library-k által támogatott modellek használhatóak, az előbbiből 157 különböző nyelven érhetőek el modellek, így a módszer könnyen alkalmazható több különböző nyelvre is[6].

4.3. Kontextusfüggő beágyazás alapú augmentálás

Egy másik, szofisztikáltabb megoldás a kontextusfüggő, maszkolt nyelvi modellek használata augmentálásra, mint például a BERT (Devlin és mtsai, 2018). Mivel a nyelvi modellek tanítása során jellemzően az egyik célfeladat a kimaszkolt szövegrészek predikciója, így az eredeti szövegből maszkolással új, az eredetihez nagyon hasonló jelentésű szövegek hozhatók létre, amelyek nyelvtanilag lényegesen nagyobb százalékban eredményeznek helyes mondatot, mint az előzőekben bemutatott mondatok. Az előzőekhez hasonlóan az augmentálás aránya itt is megadható. Az augmentáláshoz használt modellek a `huggingface` oldaláról [7] érhetőek el, igen széles választékban.

5. Egyéb modulok

5.1. A `hungarian-stemmer` szótövező

Az utóbbi években nagy népszerűségnek örvendenek a hunspell-alapú szótövező eszközök (Halácsy és mtsai, 2004) mind a kutatók, mind az ipari felhasználók körében. Gyakori probléma volt azonban a magyar nyelvű verzióban, hogy a szótövező "túltövez", gyakran olyan töveket is visszaadva, amelyek jelentésben igen messze vannak a kiindulási szótól. Emellett tapasztalható volt, hogy az igekötős igék esetén a korábbi verzió igekötő nélküli szótövet adott vissza, illetve nem történt szótövezés néhány olyan esetben, amikor szükséges lett volna (lásd 2.

[6] https://fasttext.cc/docs/en/crawl-vectors.html
[7] https://huggingface.co/models

táblázat). Erre válaszul készült el a `hungarian-stemmer`, amely ezt a problémás működést kiküszöbölve orvosolja a fent említett problémákat. A különbségekről a 2. táblázat ismertet néhány példát.

2. táblázat. A hunspell és hungarian-stemmer közti különbség

Szó	**hunspell**	**hungarian-stemmer**
mással	más, ma, mi	más
megtörtént	történik	megtörtént, megtörténik
köztestületi	köztestületi	köztestület

5.2. Szöveg vektorizálás

Egy ismétlődő, gyakori feladat lehet annak vizsgálata, hogy az adott szövegekre egyszerűbb vektorreprezentációs formák használatával betanított modellek hogyan teljesítenek. A keretrendszer ezért tartalmaz egy dokumentum vektorizáló modult, amely jelenleg háromféle módon képes dokumentumokhoz vektorokat rendelni:

- dokumentumban szereplő szavak szóbeágyazásainak átlaga
- dokumentumban szereplő szavak szóbeágyazásainak a szavak idf értékeivel súlyozott átlaga
- Doc2Vec vektor készítése (Le és Mikolov, 2014).

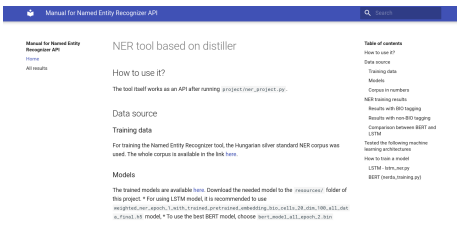
5.3. Pozitív-címkézetlen tanulás

A gyakorlatban sok esetben nem csak a gépi tanulási modell tanításához szükséges adatok elégtelen száma jelent problémát, hanem az adatok elégtelen címkézése is. Ilyen esetben például egy bináris klasszifikálási feladatnál csak az ismert, hogy a pozitív minták valóban pozitívak, azonban a többi adatról nem rendelkezünk egyéb információval; azok egyaránt lehetnek negatív és pozitív címkéjű adatok is. Ez a problémakör a pozitív-címkézetlen tanulás (Positive Unlabeled learning, PU learning) (li és Liu, 1970; Bekker és Davis, 2020). A többféle megközelítés közül a `digital-twin distiller` keretrendszerben jelenleg az Elkan és Noto (2008) által közölt megoldás érhető el.

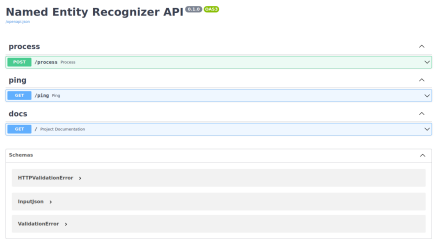
A plugin minden olyan gépi tanulási algoritmust támogat, amely képes valószínűség predikciójára, tetszőleges vektorizálási formán, tehát egyaránt alkalmas pl. tf-idf vagy szóbeágyazás alapú vektorokon is működni. A plugin rendelkezik egy olyan funkcióval is, hogy a pozitív entitások számát becsülje az Elkan és Noto (2008) által közöltek szerint, valamint használható a kézi annotálás hatékonyabbá tételére is.

6. Dokumentáció

Egy projekt megvalósításának az egyik legfontosabb sarokköve (különösen a későbbi üzemeltetés szempontjából) a megfelelő dokumentálás. Ennek a megvalósításában is segít a `digital-twin distiller` keretrendszer, amely a népszerű `mkdocs`[8] dokumentáció készítő szoftverre épít. A keretrendszerbe integráltuk továbbá az Open API[9] (korábban Swagger) által biztosított API dokumentációt bemutató eszközt is, amellyel nem csak az elkészített API végpontjainak dokumentációját lehet elolvasni, hanem ki is lehet próbálni ezeket. Ennek segítségével a kódolásban járatlanok is könnyedén tesztelhetik az elkészített API működését. Erre mutat egy példát a 2. ábra.



(a) Mkdocs-alapú dokumentáció kezelése



(b) Egy elkészült névelem-felismerő rendszer Open API dokumentációja

2. ábra. Dokumentáció a `digital-twin distiller` keretrendszeren keresztül

7. Alkalmazási példák

A keretrendszer működését, alkalmazását néhány rövid, gyakorlati példán is szemléltetjük.

7.1. Augmentálás

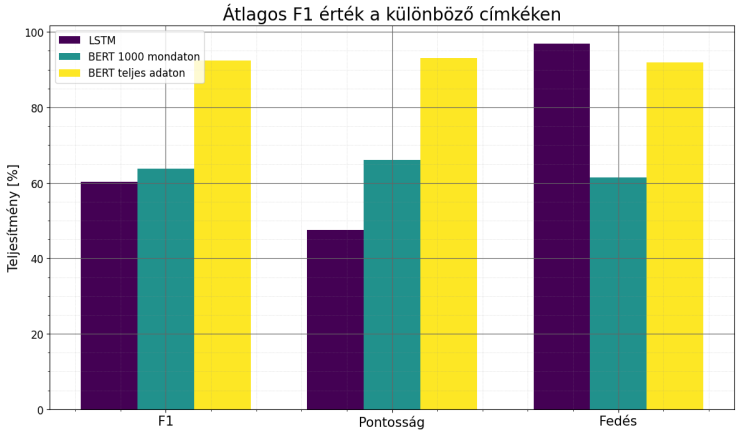
Az 4. fejezetben bemutatott augmentálási módszerek eredményeit egy példamondaton a 3. táblázat mutatja be. Mindegyik módszer esetében $\alpha = 0,5$-ös beállítást alkalmaztunk, amely esetén az adott algoritmus a mintapélda tokenjein átlagosan 50% eséllyel alkalmazta az adott augmentálási módszert. A módszerek időigényét egy eredeti mondatot 100 példányra augmentálva tüntettük fel, és a modellek betöltésének idejét az értékekbe nem számítottuk bele.

Látható, hogy maguk a módszerek különböző előnyökkel és hátrányokkal rendelkeznek. Az EDA algoritmusok jelentik a minőségben leggyengébb megoldást, azonban futási idejük viszonylag alacsony.

[8] https://www.mkdocs.org/
[9] https://www.openapis.org/

Módszer	**Futási idő [s]**	**Augmentálás eredménye**
Eredeti mondat	-	A horvát élvonalban játszó Kleinheisler László biztosan nem fog felépülni a magyar válogatott következő két vb selejtezőjére.
EDA/RD	0,61	A horvát Kleinheisler László
EDA/RS	0,52	A felépülni magyar játszó Kleinheisler selejtezőjére. következő biztosan fog vb a élvonalban nem László két horvát válogatott
EDA/RI	0,50	A horvát élvonalban makulátlanul játszó Kleinheisler László biztosan nem ereszt eredő fog felépülni a magyar válogatott következő két vb selejtezőjére.
EDA/SI	0,46	A horvát élvonalban játszó Kleinheisler László ügyesen nem ereszt felépülni a magyar válogatott rákövetkező két vb selejtezőjére.
FastText	24,92/0,12*	A horvát élvonalban játszik Kleinheislerről Tamás biztosan nem fogok felépülhet a külföldi válogatott fenti Két vb selejtezőjére.
BERT	44,67	A magyar színekben játszó Tóth már biztosan nehezen fog felépülni a magyar válogatott következő két vb selejtezőjére.

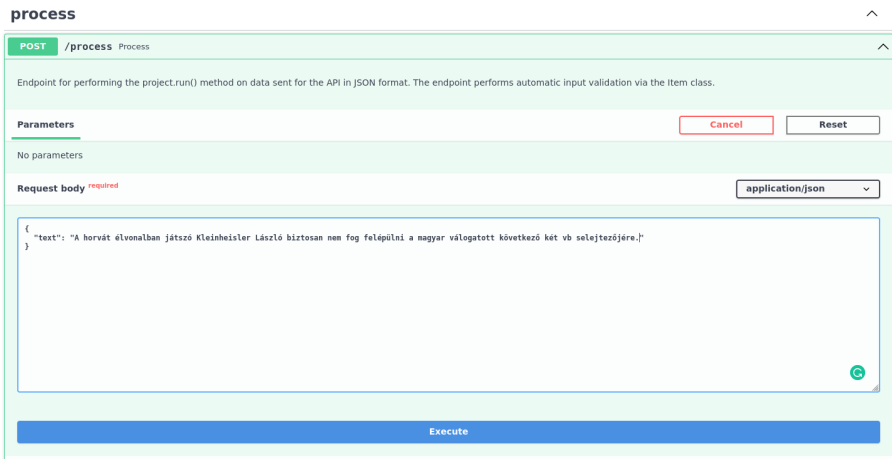
3. táblázat. Különböző augmentálási módszerek működése, $\alpha = 0,5$ beállítással. *: előre elkészített szótár használatával



3. ábra. Névelem-felismerés eredménye

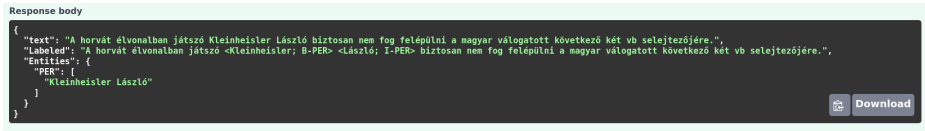
A minőségben jelentősebb előrelépést jelentő `FastText`-et használó augmentálás lényegesen lassabb amíg a szöveg szavaihoz leghasonlóbb szavakat tartal-

mazó szótár elkészül, azonban ha ezzel már rendelkezünk, akkor jelentősen javul az augmentálás sebessége is. A példa alapján is látszik, hogy a kontextusfüggő BERT módszer adja a legjobb minőségű augmentált adatot, nem ritkán nyelvtanilag tökéletes mondatokat alkotva, azonban ehhez is szükséges a legtöbb idő. Megfigyelhető tehát a gyakran megjelenő performancia-idő tradeoff (Baeza-Yates és Liaghat, 2017), másképpen a „jó munkához idő kell” népi bölcsesség.



(a) Névelem-felismerő tesztelése

(b) LSTM modell által talált entitás



(c) BERT modell által talált entitás

4. ábra. Különféle NER modellek egyszerű cserélhetősége

7.2. Névelem-felismerő rendszer

Ebben a fejezetben egy elkészült névelem-felismerő rendszer működését mutatjuk be. A keretrendszer képes bármilyen fájlba menthető illetve onnan beolvasható gépi tanulási modellt kezelni egy projekten belül, tehát egyaránt alkalmas pl. `sklearn`, `keras`, `huggingface` stb. alapú modellek hatékony használatára API-ként, ezáltal lehetőséget biztosít, hogy minimális módosítások elvégzésével a különböző módokon elkészült modellek egyaránt használhatóak legyenek.

Példaként egy egyszerű névelem-felismerőt tanítottunk fel a hunNERwiki korpuszon (Simon és Nemeskey, 2012), amely IOB2 taggelési formátumot használ.

Kétféle különböző modellt tanítottunk fel; az egyik egy `keras` segítségével feltanított LSTM (Hochreiter és Schmidhuber, 1997) alapú megoldás, a másik pedig a kisméretű többnyelvű BERT nagybetűket is kezelő változatának (bert-base-multilingual-cased) finomhangolása volt. Tanításhoz az adatok 80%-át használtuk, 10% validálási és 10% teszthalmaz felosztásban.

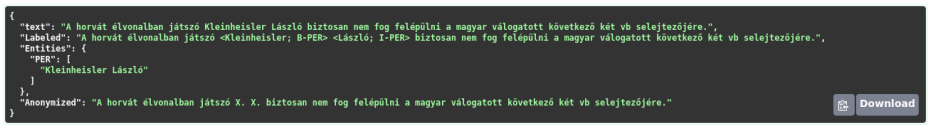
A teszthalmazon elért eredményeket a 3. ábra mutatja be, ahol az egyes entitások F1 értékeinek nem súlyozott átlagait tüntettük fel.

Látható, hogy az LSTM modell nagyobb fedést volt képes elérni, mint a többnyelvű BERT bármelyik verziója, azonban az F1-mérték átlagát tekintve jelentősen jobbnak bizonyult az utóbbi modell, amely már ezer(!) nem csak O entitást tartalmazó mondat esetén jobban teljesített, mint a több epochon keresztül tanított LSTM modell.

Az elkészült API tesztelését a különböző modellek segítségével a 4. ábra mutatja be. Látható, hogy mindkét modell megtalálta Kleinheisler László nevét, azonban az LSTM modell hozzátartozónak ismerte föl a „biztosan” szót is.

7.3. Anonimizálás

Az előző fejezetben feltanított névelem-felismerőnket ezt követően a `digital-twin distiller` keretrendszer segítségével egy kezdetleges anonimizáló eszközzé is formálhatjuk. Elkerülve a monogramok használatát, a neveket „X” karakterre cseréltük kizárólag az első karaktert meghagyva és a rövidítést ponttal jelölve. Ennek eredménye a 5. ábrán látható.



5. ábra. Anonimizálás eredménye

A példamondat alapján jól látszik, hogy ez a művelet miért nem tekinthető a GDPR-nak megfelelő anonimizálásnak (Csányi és mtsai, 2021). Ehhez egy statisztikai analízisre is szükség lenne, amelynek a segítségével bizonyítható, hogy a szövegben található személy személyazonossága nem ismerhető fel a mondatban található mikroadatokból.

Ismertek ugyanis a következő információk:

- sportolóról van szó,
- aki a horvát élvonalban játszik,
- a magyar válogatott tagja,
- és vb selejtezőn nem tud részt venni, tehát olyan sportágban, ahol ilyen egyáltalán létezik.

Ezek alapján feltételezve az egyik legnépszerűbb sportágat, a labdarúgást, valamint ha ismert, hogy mikor keletkezett a szöveg (2021), könnyen megtalálhatjuk, hogy kiről is van szó, hiszen jelenleg egyetlen labdarúgó játékos van, akire illik ez a leírás (Lovrencsics Gergő ugyanis visszamondta a válogatottságot). Ha ezen pszeudo-azonosítók (Dalenius, 1986) – amik önmagukban nem, de együttesen már alkalmasak lehetnek az egyén azonosítására – nincsenek megfelelően kezelve, akkor a fent bemutatott anonimizálási megoldás lényegében haszontalan. Fontos ezért információtartalom szerint is figyelembe venni a pszeudoinformációkat anonimizáláskor, ami az egyes információkhoz tartozó ekvivalenciaosztályok számosságának a becslésével lehetséges (Csányi és mtsai, 2021). Ha az ekvivalenciaosztály elegendően nagy, azaz a társadalom egy elegendően nagy részére illik az adott mikroadat, azzal biztosítható, hogy nehezen lehessen felismerni az adott szövegben szereplő egyén személyazonosságát. Ha a példamondatból kivesszük a két leginkább szűkítő információt, tehát, hogy horvát élvonalban játszik, és hogy magyar válogatott, illetve ezekből csak a nemzetiségneveket, akkor az már meggátolhatja a konkrét személy beazonosítását.

8. Összegzés

A cikkben egy Python alapú eszközt, a `digital-twin-distillert` mutattuk be, amelynek a segítségével készített szövegfeldolgozási modellek egy parancs segítéségével konténerizálhatóak egy digitális ikerként. Az ilyen módon előállt, kapszulázott mesterséges intelligencia modell könnyen és szabványos módon integrálható bármilyen más rendszerbe a projekthez csomagolt REST-API segítségével, emellett akár önállóan is használható a beépített webes alkalmazás segítségével. A számítási platform a modellalkotáshoz és a kutatási projektek támogatásához számos plugint tartalmaz, melyek tetszőlegesen bővíthetők. A cikkben három gyakorlati példán - egy augmentált példamondaton, több módszerrel feltanított névelem-felismerőn mutattuk be hogy hogyan lehet kutatási és ipari projektek támogatására, közzétételére használni a `digital-twin-distillert`. A számítási platformhoz számos plugin tartozik, amelyek segítségével a modellek tanítása és a szövegek preprocesszálása egyszerűsíthető, ezáltal jelentős idő megtakarítást elérve. Jó példa erre, hogy a hosszú ideig preprocesszált adatainkat kimenthetjük és beolvashatjuk egy snapshotba (`DataSnapshot`), szintetikusan létrehozott mintákkal javíthatjuk a gépi tanulási modelljeink teljesítményét (augmentáció), nem teljesen címkézett adatokon taníthatunk hatékonyabb gépi tanulási modelleket (PU-learning), dokumentumokat vektorizálhatunk egyszerűen, vagy szótövezhetünk. A `digital-twin distiller` keretrendszer a fentiekben bemutatott pluginokkal és modulokkal együtt szabadon elérhető[10].

[10] https://bitbucket.org/montanatudasmenedzsmentkft/distiller/src/master/ (2021.11.09.)

Köszönetnyilvánítás

A tanulmány a 2020-1.1.2-PIACI-KFI-2020-00049 számú projekt a Nemzeti Kutatási, Fejlesztési és Innovációs Hivatal támogatásával valósult meg, a 2020-1.1.2-PIACI KFI finanszírozási rendszer keretében.

Hivatkozások

Ahmad, H.: How netflix metaflow helped us build real-world machine learning services (2021), (2021.11.12.)

Baeza-Yates, R., Liaghat, Z.: Quality-efficiency trade-offs in machine learning for text processing. In: 2017 IEEE International Conference on Big Data (Big Data). pp. 897–904 (2017)

Bekker, J., Davis, J.: Learning from positive and unlabeled data: A survey. Machine Learning 109(4), 719–760 (2020)

Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics 5, 135–146 (2017)

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research 16, 321–357 (2002)

Csányi, G., Orosz, T.: Comparison of data augmentation methods for legal document classification. Acta Technica Jaurinensis (2021)

Csányi, G.M., Nagy, D., Vági, R., Vadász, J.P., Orosz, T.: Challenges and open problems of legal document anonymization. Symmetry 13(8), 1490 (2021)

Dalenius, T.: Finding a needle in a haystack or identifying anonymous census records. Journal of official statistics 2(3), 329 (1986)

Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

Elkan, C., Noto, K.: Learning classifiers from only positive and unlabeled data. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 213–220 (2008)

Halácsy, P., Kornai, A., László, N., András, R., Szakadát, I., Viktor, T.: Creating open language resources for hungarian (2004)

Hecht, L.E.: Add it up: How long does a machine learning deployment take? (2019), https://thenewstack.io/add-it-up-how-long-does-a-machine-learning-deployment-take/, (2021.11.12.)

Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9(8), 1735–1780 (1997)

Honnibal, M., Montani, I.: spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing (2017), to appear

Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International conference on machine learning. pp. 1188–1196. PMLR (2014)

li, X., Liu, B.: Learning from positive and unlabeled examples with different data distributions. pp. 218–229 (01 1970)

Merkel, D., és mtsai: Docker: lightweight linux containers for consistent development and deployment. Linux journal 2014(239), 2 (2014)

Miháltz, M., Hatvani, C., Kuti, J., Szarvas, G., Csirik, J., Prószéky, G., Váradi, T.: Methods and results of the hungarian wordnet project. In: Proceedings of The Fourth Global WordNet Conference. pp. 311–321 (2008)

Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)

Orosz, T., CSányi, G., Nagy, D.: Mesterséges intelligenciát alkalmazó szövegbányászati eszközök készítése a distiller keretrendszer segítségével–jogi szövegek automatikus feldolgozása: Development of artificial intelligence-based text mining tools with the distiller-framework–in case of legal documents. Energetika-Elektrotechnika–Számítástechnika és Oktatás Multi-konferencia pp. 62–69 (2021a)

Orosz, T., Gadó, K., Katona, M., Rassõlkin, A.: Automatic tolerance analysis of permanent magnet machines with encapsuled fem models using digital-twin-distiller. Processes 9(11) (2021b), https://www.mdpi.com/2227-9717/9/11/2077

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., és mtsai: Scikit-learn: Machine learning in python. the Journal of machine Learning research 12, 2825–2830 (2011)

Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)

Rassolkin, A., Orosz, T., Demidova, G.L., Kuts, V., Rjabtšikov, V., Vaimann, T., Kallaste, A.: Implementation of digital twins for electrical energy conversion systems in selected case studies. In: Proc. Est. Acad. Sci. vol. 70 (2021)

Schmitt, X., Kubler, S., Robert, J., Papadakis, M., LeTraon, Y.: A replicable comparison study of ner software: Stanfordnlp, nltk, opennlp, spacy, gate. In: 2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS). pp. 338–343. IEEE (2019)

Simon, E., Nemeskey, D.M.: Automatically generated ne tagged corpora for english and hungarian. In: Proceedings of the 4th Named Entity Workshop (NEWS) 2012. pp. 38–46. Association for Computational Linguistics, Jeju, Korea (July 2012), http://www.aclweb.org/anthology/W12-4405

smartface: Add it up: How long does a machine learning deployment take? (2021), What is the Difference Between a Platform and a Framework?, (2021.11.12.)

Team, J.D.: Joblib: running python functions as pipeline jobs (2020), https://joblib.readthedocs.io/, (2021.11.12.)

Van Rossum, G.: The python library reference, release 3.8. 2. Python Software Foundation 16 (2020)

Vogel-Heuser, B., Ocker, F., Weiß, I., Mieth, R., Mann, F.: Potential for combining semantics and data analysis in the context of digital twins. Philosophical Transactions of the Royal Society A 379(2207), 20200368 (2021)

Wei, J., Zou, K.: Eda: Easy data augmentation techniques for boosting performance on text classification tasks. arXiv preprint arXiv:1901.11196 (2019)
Üveges, I., Orosz, T., Csányi, G., Orsolya, R.: Szövegaugmentálási módszerek összehasonlítása politikai szövegek szentimentanalízise során. XVIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2022) (2022)