Beszámoló

a Pro Progressio Alapítvány által kiírt
Flextronics TMIT hallgatói és kutatói ösztöndíj pályázathoz

„x86 alapú rendszerek prediktív karbantartása és diagnosztikája visszaküldött hibás egységek logjait felhasználva” című témához

Az ösztöndíjas kutatási munkámban egy konkrét rendszer (telekommunikációban használatos router kártyák) által készített logfájlok kiértékelhetőségét vizsgáltam. Az első lépés az adathalmaz rendszerezése volt, majd a szöveg tisztítása közben a hagyományos nyelvfeldolgozáshoz képest jelentősen eltérő megoldások alkalmazása. A fő aspektusa ezeknek a megoldási lehetőségeknek, vagyis a logfájl csomagok megtisztításának a vizsgálata és bemutatása volt, valamint az odáig vezető út közben felmerülő háttértudás megszerzésének ismertetése. Az első feladat az volt, hogy hogyan jutunk el egy nagy loghalmazból egy rendszerezett struktúrán át ennek egy matematikai reprezentációjáig.

A fejlesztés végső célja, hogy a régi, felcímkézett, különböző hibás futások alatt keletkezett logfájlok alapján egy új eset jelentésekor automatizáltan meg tudjuk mondani, hogy milyen hiba történt a rendszerben. Ezt az összetett feladatot úgy tudjuk megfogni, ha egy olyan egyszerűsítéssel élünk, hogy kizárólag egy hibatípussal foglalkozunk. Ez azt takarja, hogy kétféle adatunk lesz: az egyik ennek a konkrét hibának a jelentkezése alatt keletkezett, a másik típus pedig hibátlan működés alatt (ez későbbi pontosítást igényel). A későbbiekben ez jó kiindulási alap több hibatípus felismeréséhez/megkülönböztetéséhez is.

Mindenféle tanulóalgoritmus egyfajta matematikai modell, ezért ezek számára olyan formátumba kell hozni az adatokat, melyekből vektorok és mátrixok képezhetők, hogy ez már bemenetként szolgálhasson bármelyik választott osztályozó algoritmus számára. A kutatási munka célkitűzése volt a szöveg megtisztítása, modellbe való leképzése is. Ez a szöveg tokenizálását, vagyis szavakra bontását jelentette, ami a logfájlok – folyószöveghez viszonyítva – különleges szerkezete miatt egyedi megoldásokat igényelt.

A munka a Flex budapesti fejlesztési központjában folyt, az általuk (fejlesztett és) karbantartott SSR (Smart Service Router) kártyák hibás működésekor keletkezett logfájl csomagok álltak rendelkezésre. Ezeket a partnerek az általuk észlelt hibák jelentésekor

csatolják, olykor akár eltérő ticketing rendszereken keresztül. Az így készülő kártyák (ticketek) a log csomagok címkézése szempontjából fontos információkkal szolgálnak, a tesztmérnökök itt írják le a probléma okát és megoldását.

Ezekből a halmazokból a megvalósítandó feladathoz 2 féle típusra van szükségünk, hibás futás esetén, illetve helyes működéskor keletkezettre. Ez pontosítást igényel, ugyanis egyrészt egy fajta hibából származó adatokra van csak szükségünk, ami megoldható. Másrészt viszont hibátlan működésről nem készítenek a partnerek kártyákat, így ilyenről logcsomagok sem állnak a rendelkezésünkre. Ezt úgy tudjuk megoldani, hogy másfajta hibából származó adatokkal dolgozunk, és ezeket tekintjük helyes működésnek. Ezt úgy is értelmezhetjük, hogy tulajdonképpen két féle hibát akarunk megkülönböztetni.

Egy ilyen logcsomagban többféle logtípust találunk, ez csomagonként nem teljesen egységes, valamint a típusok egymástól vagy részben vagy akár egészen eltérő formájúak is lehetnek. Ezek mind kihívások abban, hogy egységes alakra hozzuk az egész adathalmazt, valamint mivel összességében nagyon nagy adatmennyiségről van szó, a releváns információt hordozó részhalmazok kiválasztása elkerülhetetlen. Ez azt jelenti, hogy a használandó logfájlok csoportját leszűkítjük a mindegyik csomagban jelenlévő és hasonló szerkezetű típusokra.

A feladatban, habár nem folyószöveggel dolgozunk, mégis mivel szövegfeldolgozásról van szó a természetesnyelv-feldolgozásban (NLP – Natural Language Processing) használatos módszerek alkalmazása a legkézenfekvőbb. A fejlesztés – adatbányászatban és gépi tanulásban megszokott módon – Python nyelven történt, valamint több függvénykönyvtárt is felhasználásra került (pl.: keras, scikit learn, nltk). Ezek magas szintű függvényeket szolgáltatnak, ellentétben mondjuk a TensorFlow-val, ami alacsonyabb szintű és valamivel több nyelvhez áll rendelkezésre. Ez lényegesen megkönnyítik munkánkat mind az adatelőkészítés, mind későbbi fejlesztés közben a tanuló algoritmusok készítése és az eredmények kiértékelése alatt.

Biztonsági okok és céges előírások miatt a fejlesztőkörnyezet céges gépen lett felállítva. Első lépés a Python 3 telepítése volt, a legtöbb adatbányászatban és gépi tanulásban használatos függvénykönyvtárnak Python3-hoz áll rendelkezésre API-ja. Ezután több lehetőség is adódik, az egyik az Anaconda telepítése, mely magában foglalja az általunk is használni kívánt fejlesztőeszközök nagy részét. Ekkor az Anacondán keresztül érjük el mind az IDE-ket, mind a könyvtárakat. Egy másik módszer a Python csomagkezelőjének a használata a pip. Ezen keresztül minden szükséges szoftver installálható.

IDE választásakor hármat próbáltunk ki: Sublime, Spyder illetve Jupyter Notebook, utóbbi kettő az Anacondának is része. Végül a Jupyter Notebook mellett döntöttümk, ez böngészőben

futtatható, nagy előnye, hogy a kódot jól elkülönített tagokra lehet bontani és részekben lefuttatni, miközben a változók értékei megmaradnak. Így nem kell minden iterációnál a teljes szkript futását végig várni, ami nagy adathalmaznál (több gigabájtos logcsomagokról van szó, még akkor is, ha a teljes halmaznak csak egy részét használjuk) eléggé számottevő különbség a munkamenetben (kicsit más formában, de ez a Spyder-ben is megoldott).

Természetesnyelv-feldolgozás alatt általában kevéssé strukturált folyószöveg, vagy beszéd automatikus feldolgozását értjük. Általánosságban elmondható, hogy a szöveg rendezetlensége folyamatos kihívást jelent és feldolgozása nem egy egyértelműen megoldott kérdés. A tématerület magában foglalja a hagyományosabbnak tekinthető, statisztikai alapokon nyugvó megoldásokat, ám a tudomány mai állása szerint a neurális hálókkal jobb eredmények érhetők el. Ezeknek számos alkalmazási területük van, mint például hosszú szövegből kivonat képzése, szövegek osztályozása tartalmuk alapján, beszédfelismerés, vagy helyesírás-ellenőrzés.

A logfájlok ugyan strukturált szerkezetűek, mégis ez a tématerület áll a legközelebb, jó kiindulási alapnak szolgál. A munkamenet is hasonló, első feladat a *szöveg megtisztítása*, itt lényegesen más problémákkal szembesülünk, mint egy hagyományos szöveg esetében. Ugyanez érvényes a szöveg *tokenizálás*ára, majd az így kapott szövegből alkotunk egy *bag-of-words* (szózsák) modellt (ennek több fajtája is van). Ebből a modellből pedig már tudunk statisztikákat és elemzéseket csinálni, valamint ez már olyan formátumú, hogy egy *osztályozó algoritmus* is fel tudja dolgozni. Az osztályozás előtt még opcionálisan beiktatható a hasonló jelentésű szavak, szószerkezetek egységesítése, ez egy továbbfejlesztési lehetőség, érdekes lehet megvizsgálni, hogy egy determinisztikus rendszer által generált szövegből elérhető-e javulás ez által.

Logfájlok tisztításakor ki tudjuk használni a szöveg strukturáltságát, ami részben előnyt jelent a folyószöveggel szemben, viszont a reguláris kifejezések használatát nem lehet kikerülni. A szükség azt diktálja, hogy bizonyos ismétlődő karaktersorozatok – például dátum – mintaillesztéssel felismerhetők és kezelhetők legyenek, a reguláris kifejezéseknek pedig éppen ez a céljuk. Nagy segítséget jelentett egy webes alkalmazás, amivel – a szkript futtatása nélkül – a szabadon választott mintaszövegen tesztelhetők a megírt kifejezések.

A logfájlok természetéből adódik, hogy gyakran ugyanaz a sor ismétlődik egymás után sokszor, ez érdemi információt nem hordoz, így ezekből elegendő egynek a megtartása. Egy másik logfájl specifikus tulajdonság, hogy minden sor időbélyeggel kezdődik. Ezek lehetnek eltérő formátumúak, de mivel a későbbiekben a szavak sorrendje nem fog szerepet játszani, így a sorok elejéről ezek is törölhetők. Fontos, hogy ezt is reguláris kifejezésekkel oldjuk meg, hiszen előfordulhat, hogy egy-egy sor elején nincs dátum-idő, valamint a logtípusonként eltérő

formátumok is így kezelhetők. A zárójeles részek elvetése egy tervezési döntés volt, hiszen ezek csak kiegészítő információkat tartalmaznak, így ami bármilyen típusú zárójelben volt az törlésre került.

Folyószöveg kezelésétől eltérő módon a helyesírási hibák, elgépelések nem mérvadók jelen probléma megoldása közben. Ugyanígy az írásjelek kezelése sem tartozik a feladathoz, hiszen soronként tördelt szövegről van szó, viszont a nagybetűs- és a speciális karakterek, valamint a számok kezelése már előre mutat a tokenizálásra.

Tokenizálás alatt a nyers szöveg modellezhető formára hozását értjük, amikor végeredményben a megtisztított szöveg szavainak listáját kapjuk. Ezek a szavak a tokenek. A token típus az azonos alakú tokenek osztálya, a token típusok halmaza pedig a szótárt adja, azaz a szótári elemek a token típusok lesznek. Mivel a logfájlok egyedi szerkezetűek, ennek a feladatnak egy nagy része manuálisan történt, viszont nagyon sok függvénykönyvtárban van megoldás rá. Ezek az általánosnak mondható dolgokat, mint a nagybetűk kicsire cserélése, vagy a speciális írásjelek, esetleg más kódolású karakterek helyettesítése megoldják.

Ennek ellenére sok feladatspecifikus probléma adódott, ezek saját implementációval lett megoldva. Sokszor az állománynevek, változók vagy függvények '_' karakterekkel vannak tagolva (például egy ilyen változó: target_cma_init_card_rec), viszont ha ezeket a többi különleges karakterrel együtt töröljük, az azt eredményezi, hogy egy ilyen név 3-4 különálló szóként fog szerepelni a modellünkben, így ez előfeldolgozást igényelt. Ezután a speciális tokenek bevezetése következett, melyek jelzésére egy-egy nagybetűs tokent különítettünk el. Hogyha ezt a cserét a teljes szöveg kisbetűssé tétele után tesszük meg, akkor ezek összetéveszthetetlenül egyediek lesznek.

A *dátum*ok nem soreleji előfordulása (a sorelejivel ellentétben) nem elvetendő információ, viszont nem összekeverendők a sima számokkal, így ezek cseréje az egyéb számok kezelése előtt kell történjen. Ehhez hasonlóak az *IP cím*ek, az előzőek alapján ennek magától adódik a megoldása. A *file*-ok és elérési utak szintén széttagolódnának, így ezeket is a különleges karakterek előtt kell kezelni. A hexadecimális számok együttes kezelése különlegesen fontos, hiszen ha nem így teszünk, akkor az akár több mint 10 bites számok rengeteg fölösleges elemmel szaporítják majd a szótárunkat. (Csak érdekességképp: ennek az egy különleges tokennek a bevezetésével nagyjából felére csökkent a szótár mérete.) Ez utóbbi még olyan szempontból egy kicsit különleges, hogy a sima szám karaktereket és a több karakterből álló – akár hexadecimális – számokat külön esetként kell feldolgozzuk.

Összességében a feladat során eljutottunk a kiindulási alapként szolgáló logcsomagoktól egy tanításra kész modellig. Eközben megismertük a logfájl specifikus szövegtisztítást és tokenizálást, valamint a bag-of-words modellt és ennek variációit, továbbfejlesztési lehetőségeit. A munkafolyamatban a hangsúly egyértelműen a tokenizáláson volt, a legtöbb munkát a speciális tokenek megtervezése, valamint az ehhez szükséges reguláris kifejezések megírása jelentette.

Budapest, 2019, szeptember 1.

Dr. Szűcs Gábor