Formális és program nyelvek elemzése gépi tanulási modellekkel

Sisák Sándor

Bevezetés

Az önálló projekt célja gépi tanuláson alapuló módszerek vizsgálata formális, illetve programozási nyelvek automatikus értelmezésére. A gépi tanulás hatékony eszköznek bizonyult természetes nyelvi feladatok megoldására, ezért hamar felmerült, hogy programkódokon is alkalmazzák. A féléves munkámban a Stack Overflow-ról gyűjtött, felhasználók által írt válaszokban található Python kódokon végeztem bináris klasszifikációt különböző szempontok szerint.

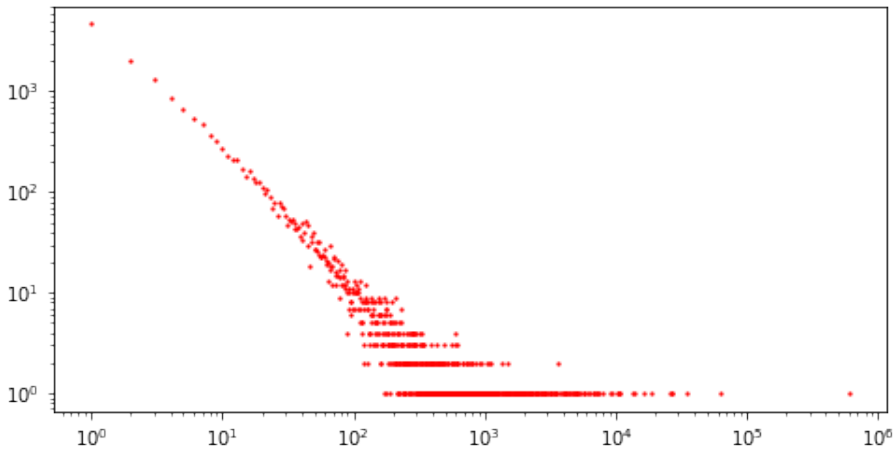
Az adathalmaz, alapfeladat

Az adatok a Kaggle *Python Questions from Stack Overflow* adathalmazából lettek előállítva [5]. A Stack Overflow fórumon programozással kapcsolatos kérdéseket tesznek föl a felhasználók, más felhasználók pedig válaszokat küldhetnek a feltett kérdésre. Az adathalmaz 607000 Python programozással kapcsolatos kérdést és ezekre adott 987000 választ tartalmaz. Minden kérdéshez tartoznak a felhasználók által meghatározott címkék (pl. *python-2.7*, *python-3.x*) amelyek a keresést segítik.

A projekt keretein belül előállítottam egy módosított adathalmazt, amelyben a válaszokból kinyert Python kód és a válaszhoz tartozó kérdés címkéi szerepeltek. A módosított adathalmazon neurális hálót tanítottam arra, hogy a kódokat egy előre rögzített címke szerint klasszifikálja.

Az adatok előkészítése

Csak azokat a válaszokat vettem figyelembe, amelyeknek legalább egy gyakori címkéje volt a *python* címkén kívül. Utóbbi címke minden kérdés címkéi között szerepelt, ezért nem hordoz információt, így a továbbiakban figyelmen kívül hagytam. Egy címkét akkor tekintettem gyakorinak, ha legalább 1000 különböző kérdés címkéi között szerepelt. A közel 17000 címkéből mindössze 207 volt gyakori (1. ábra).



1. ábra. A címkék gyakorisága. (x, y) pont azt jelöli, hogy y különböző címke van az adathalmazban, amely pontosan x különböző kérdésnél szerepel.

A válaszok szövegéből a *regex* könyvtár segítségével kinyerhetőek voltak a kódcellaként formázott részek. Ezeket a kódrészleteket tovább szűrtem az alapján, hogy szintaktikailag helyesek voltak-e. Ha az *ast* könyvtár *ast.parse* parancsa nem hibaüzenettel tért vissza a kódrészleten, akkor a kódrészletet szintaktikailag helyesnek tekintettem. Természetesen azok a válaszok nem kerültek a végső adathalmazba, amelyekben nem volt legalább egy szintaktikailag helyes kódrészlet. A fenti adatfeldolgozás eredményeként kapott adathalmazban már csak 492000 adatpont szerepelt. Minden adatpont két nemüres listából állt, az egyik kódrészleteket, a másik pedig címkéket tartalmazott. A mérések során a kódrészlet-listáknak csak az első elemét vette figyelembe a modell. Ezzel sok információt veszítek, de megkönnyítem az implementációt.

A modell felépítése

A projektben használt mélytanulási modell egy reprezentációt előállító részből és egy fejből áll. A reprezentációhoz a modell először tokenizálja, vagyis diszkrét (ebben az esetben egészértékű) vektorrá alakítja a kódrészleteket, majd átalakítja egy valós vektor reprezentációvá. Ezt a reprezentációt nevezzük beágyazásnak. A beágyazás a bemenet lényeges jellemzőit írja le, egy megfelelő beágyazás sokkal hatékonyabbá teszi az adatok további feldolgozását. A fej ezekből a jellemzőkből állítja elő a kimenetet, ami bináris klasszifikáció esetében egy kételemű valószínűség vektor.

A reprezentációs modellek betanítása nagyon számításigényes, ezért az előtanított *RoBERTa* modellt használtam [4]. Ez a modell a *BERT* [1] egy változata ami egy transzformer-alapú modell [6]. A transzformer újítása a korábbi modell-struktúrákhoz képest a self-attention modul. Egy ilyen modul $x_1, \dots x_n \in \mathbb{R}^k$ bemenetet vár és

$y_1, \dots y_n \in \mathbb{R}^l$ kimenetet ad. A modul az inputokat összehasonlítja, és súlyozott összegeket állít elő az alapján, hogy melyik inputra érdemes több figyelmet fordítani.

A self-attention modul a bemeneti vektorokból először lineáris transzformációval három-három belső vektor-reprezentációt állít elő: a kulcs, a query és az érték vektort. Rendre: $x_{K,i}$, $x_{Q,i}$, $x_{V,i}$. $x_{K,i}$ és $x_{Q,i}$ hossza megegyezik. Ezután minden indexre kiszámol egy $a_j = x_{Q,i} \cdot x_{K,j}$ értéket. Egyes transzformerek a_j-t másként számolják, de ugyanezekből a vektorokból. A végső *attention score:* $a'_1, \dots a'_n = \text{softmax}(a_1, \dots a_n)$, amelyeket az $y_i = \sum_{j=1}^{n} a'_j x_{K,j}$ kimenet súlyaiként használunk. A modell paraméterei a belső reprezentációt előállító lineáris transzformációk paraméterei.

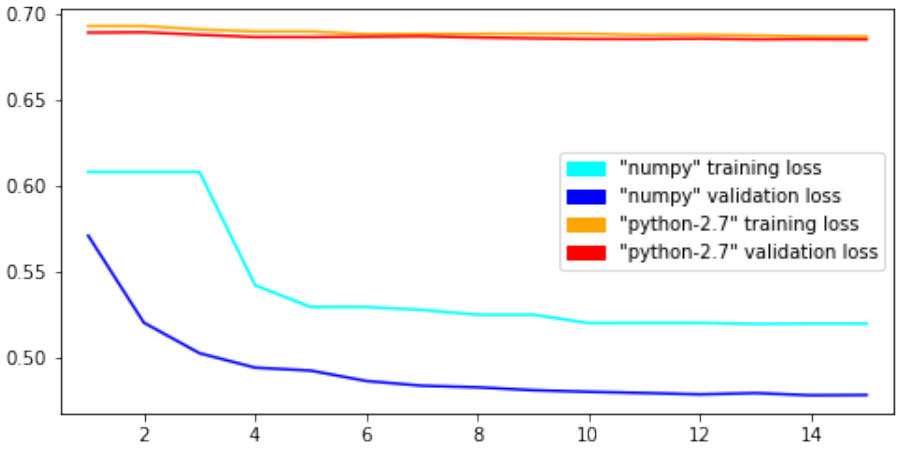
A RoBERTa természetes nyelvi korpuszon volt tanítva, ezért az általa létrehozott kód-beágyazás nem feladat specifikus, de bármilyen stringet elfogad bemenetként. Másrészt nagyon erős modell, ezért jó viszonyítási alapot fog jelenteni későbbi mérések értékeléséhez. Kísérleteimben a RoBERTa belső súlyai rögzítve voltak és csak a fej tanult. A fej egy lineáris sűrű réteg volt, amelynek a kimenete két valós szám, ami a pozitív (a címke illik a vizsgált kódra) és negatív (a címke nem illik a vizsgált kódra) osztály becsült valószínűsége. A végső kimenet azt írja le, hogy melyik valószínűséget ítélte a modell nagyobbnak. A veszteségfüggvény keresztentrópia-veszteség, az optimalizációt Adam-mal végeztem [3].

A modell értékelése

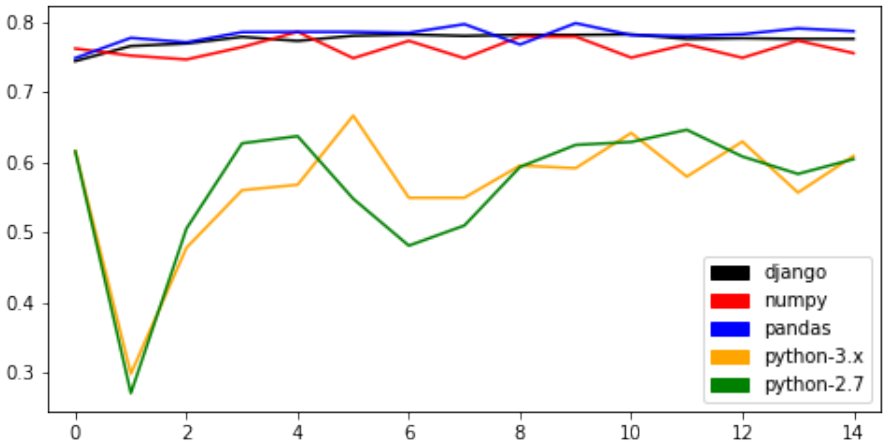
A beágyazás előállítása (előtanított modellel is) lassú, számításigényes feladat, ezért a féléves munkámban a modellt csak az öt leggyakoribb címke szerinti klasszifikációra tanítottam. Egy adott címke az adathalmaznak csak kis részében fordul elő, tehát bináris klasszifikáció szempontjából ferde az adathalmaz. A ferde adathalmazon tanított modellek teljesítménye gyengébb, így a modell tanítása előtt minden vizsgált címkére speciális tanítóadatot állítottam elő: a modell a teljes adathalmazból minden olyan adatpontot felhasznál, amely rendelkezik a címkével, és véletlenszerűen választ a maradékból még egyszer ugyanennyi adatpontot. Az így kapott kiegyensúlyozott adathalmaz 80%-a tanítóadat, 10%-a tesztadat, 10%-a validációs adat, véletlenszerűen elosztva. A vizsgált címkéket, előfordulásuk számát és arányát az adathalmazban, illetve a fenti módszerrel kapott címke-specifikus tanítóadat méretét az alábbi táblázat összesíti.

Címke	Össz. előfordulás	Előfordulás arány	Tanítóadat mérete
django	55105	11,19%	88168
python-2.7	37313	7,58%	59701
python-3.x	33387	6,78%	53419
numpy	30759	6,25%	49214
pandas	28227	5,73%	45163

A teljesítményt a validációs halmazon F1-score mérte. A modell minden címke tanítóadatán 15 epochon keresztül tanult. A *django*, *numpy* és *pandas* címkék esetében



2. ábra. A tanítási- és validációs veszteség alakulása egy könnyen és egy nehezen tanulható címkén 15 epoch alatt.



3. ábra. Az F1-score alakulása 15 epoch alatt.

a modell viszonylag jól teljesített: a veszteség csökkenéséből látszik, hogy bár lassan, de megtanulja az adatot, ugyanakkor az F1-score ingadozó volt. A *python-3.x* és *python-2.7* címkéken azonban gyenge volt a teljesítmény, mind a veszteségfüggvény, mind F1-score tekintetében, minden bizonnyal azért, mert a Python verziók közötti különbségek nehezebben azonosíthatóak, mint a Python könyvtárak használata (2. és 3. ábra).

Kitekintés

A tárgy folytatásában célul tűztem ki a feladat kiterjesztését többcímkés klasszifikációra, hogy tízes, vagy akár százas nagyságrendű címkével tudjon párhuzamosan címkézni a modell. Ez a technikai megvalósításon túl számos problémát vet föl. Egyrészt többcímkés klasszifikáció során az egyes címkékre külön-külön nagyon ferde adathalmazon kell tanulnia a modellnek. Másrészt néhány címkétől eltekintve a címkékre az adatban kevesebb mint 10000 példakód áll rendelkezésre, így vagy több adatra van szükség, vagy olyan modellt kell konstruálni, ami kevés adatból is jól tud általánosítani. Utóbbi a feladat komplexitása miatt nehezen megvalósítható.

A további adatgyűjtés azért is előnyös lehet, mert utat nyit a meglévő adatok szigorúbb szűrésének. A Stack Overflow válaszokban a felhasználók sokszor egyetlen függvény nevét is kódcellaként formázzák, ezekből a kódrészletekből a modell nagyon kevés információhoz jut, viszont hossz alapján könnyen szűrhetőek. Az eredeti adathalmazon rögzítve van a válaszok pontszáma is (más felhasználók értékelhetik a válaszokat), amely alapján tovább szűrhető az adat, hiszen feltehető, hogy a pontszám és a válaszhoz tartozó kódrészletek minősége között van korreláció.

A modellt jelenleg az adatpontok kódrészlet-listáinak csak egyetlen elemén tanítom, ezzel sok információt veszítve. Erősebb lenne egy olyan modell, ami az adatpontokhoz tartozó összes kódrészletet fel tudja dolgozni. A modell most előtanított RoBERTa tokenizációt és beágyazást használ. Mint azt a beszámoló korábban említi, a RoBERTa természetes nyelvi modell, így feltehetőleg a tokenizáció és a beágyazás nem a legmegfelelőbb programozási nyelvi feladatokhoz. A beágyazás tanításához sokkal nagyobb erőforrásokra van szükség, így a reprezentációt továbbra is előtanított modellekkel fogom előállítani, de itt is van lehetőség a teljesítmény javítására: fejleszthetek saját tokenizációt, vagy kiegészíthetem a jelenlegit, hogy több releváns információt hordozzanak a tokenek. Továbbá összehasonlíthatom a RoBERTát más előtanított modellekkel, mint a CodeBERT és a GraphCodeBERT [2]. A jobb beágyazások okot adnak összetettebb fej tanítására, tovább javítva a modellt.

Hivatkozások

[1] Jacob Devlin és tsai. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *arXiv preprint arXiv:1810.04805* (2018).

[2] Daya Guo és tsai. "GraphCodeBERT: Pre-training Code Representations with Data Flow". *arXiv preprint arXiv:2009.08366* (2020).

[3] Diederik P Kingma és Jimmy Ba. "Adam: A Method for Stochastic Optimization". *arXiv preprint arXiv:1412.6980* (2014).

[4] Yinhan Liu és tsai. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". *arXiv preprint arXiv:1907.11692* (2019).

[5] *Python Questions from Stack Overflow*. 2019. URL: https://www.kaggle.com/datasets/stackoverflow/pythonquestions.

[6] Ashish Vaswani és tsai. "Attention is all you need". *Advances in neural information processing systems* 30 (2017).