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, reprezentációk építésére. A gépi tanulás hatékony eszköznek bizonyult természetes nyelvi (NL) feladatok megoldására, ezért hamar felmerült, hogy programkódokon (PL) is alkalmazzák. A gyakorlatban a legelső feladat a forráskód olyan reprezentációja, amit mélytanulással vizsgálni lehet. A féléves munkámban főleg az ezzel kapcsolatos nehézségeket jártam körbe.

Feladatok

Elsősorban olyan feladatok vetődnek fel, melyek segítséget nyújthatnak a szoftverfejlesztésben. Ilyen például hibás kódok automatikus javítása, kódok automatikus kommentálása, címkézése, klasszifikációja. Felmerül kódok szemantikai egyenértékűségének felismerése (*clone detection*), ahol a bemenet egy kódpár és 1 a kimenet ha ezek egyenértékűek, egyébként 0. Megfogalmazhatunk olyan feladatokat is, melyekben az NL és PL felismerés egyaránt szerepet kap: kódkeresés egy adatbázisból NL leírás alapján, illetve kód beszélt nyelvre fordítása.

A témám leírásában szerepelt a feladatok között a kódok futtathatóságának eldöntése. A futtathatóságnak nincs egzakt matematikai definíciója – jelentheti akár a megállási problémát is – de gyakorlati szempontból érdekes kérdés lehet.

A programkód-reprezentálás sajátosságai

A programkódokban jellemzően sokkal sűrűbb az információ, mint a természetes nyelvekben, ráadásul ennek az információnak jelentős része strukturális információ. Például két parancs megcserélésével jelentősen eltérő értelmű kódot kaphatunk, vagy egy ciklus lezárása után a következő parancssor kontextusa, szövegkörnyezete nem az azt megelőző sor, hanem a teljes ciklus előtti sor. Akár hosszabb programokban is gyakran előfordul, hogy egy függvényt csak egyetlen alkalommal hívunk meg. Az ilyen függvény reprezentálása megnehezíti a gépi tanulást, hiszen ilyenkor egyetlen kontextusra hagyatkozva kell megsejteni a jelentést. Az is megnehezíti a kódok feldolgozását, hogy nincs egy előre rögzíthető szótár: az elemi parancsok és a legnépszerűbb könyvtárak persze minden programnyelven ismertek, de a vizsgált kód megalkotója is deklarál változókat és függvényeket, és nem zárható ki hogy olyan könyvtárakat használ, amit a kódfeldolgozó program előre nem ismer.

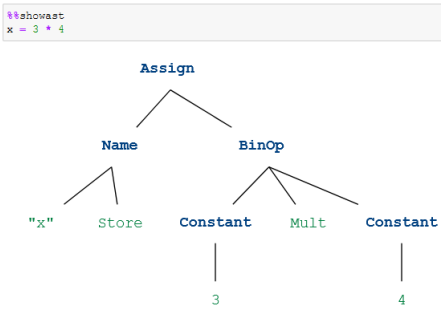
A fentiek miatt a nyers szövegből dolgozó nyelvfelismerő módszerek (pl. CBOW) alkalmatlannak bizonyultak formális nyelveken történő alkalmazásra.

Absztrakt szintaxis fák

Hasonlóan a beszélt nyelv-feldolgozáshoz (NLP), a terület egyik alapfeladata a programok vektor-reprezentációja, beágyazása. NLP-re már léteznek jó beágyazások, melyekkel például a clone detection-höz hasonló feladatok kevés tévesztéssel megoldhatóak. [1] A nyers forráskód szavankénti beágyazása viszont nem effektív,

a jelenlegi legjobb modellek más módszereket alkalmaznak. [3] Ezért feltételezzük hogy pontosabb megoldásokat kapunk ha megfelelően alakítjuk a bemenetet.

Az absztrakt szintaxis fák (*abstract syntax tree*, a továbbiakban AST) a PL-t leíró gyökeres fák, melyek csúcsai a kódban szereplő szimbólumok. (változók, operátorok, stb.) Mivel az AST lényegében azt írja le ahogyan a számítógép olvassa és végrehajtja a kódot, segít a programkód-felismerésben, ugyanis jól elkülönül a strukturális információ a másodlagos információktól, mint a változók nevei.



1. ábra. Példa AST-re

Egy vektor-reprezentáció akkor effektív, ha hasonló értelmű szimbólumokat kis különbségű vektorok írnak le. Erre ad eljárást [5, 4], ahol a szimbólumokhoz rendelt vektorokat az AST-ben szereplő gyerekeinek vektorával definiáljuk. (Ez mondatok ágrajzán már kipróbált módszer [6]) Legyen p az AST egy csúcsa, melynek közvetlen gyerekei $c_1, c_2, \dots c_k$. Jelölje $\text{vec}(v)$ a v csúcshoz rendelt vektort. Ekkor

$$\text{vec}(p) \approx \tanh(\sum_{i=1}^{k} l_i W_i \cdot \text{vec}(c_i) + \mathbf{b})$$

lesz, ahol $\mathbf{b} \in \mathbb{R}^N$ a *bias*, $l_i = \frac{c_i \text{ alatti levelek száma}}{p \text{ alatti levelek száma}} \in \mathbb{R}$ súlyok, és
$W_i = \frac{n-i}{n-1} W_{bal} + \frac{i-1}{n-1} W_{jobb} \in \mathbb{R}^{N \times N}$. A $\tanh(x)$ függvényt itt elemenként alkalmazzuk.
A $\text{vec}(p)$ nem lesz pontosan a fenti képletben meghatározott vektor, mivel *negative sampling* módszert alkalmazunk, hogy elkerüljük a triviális (azonosan nulla) reprezentációkat.

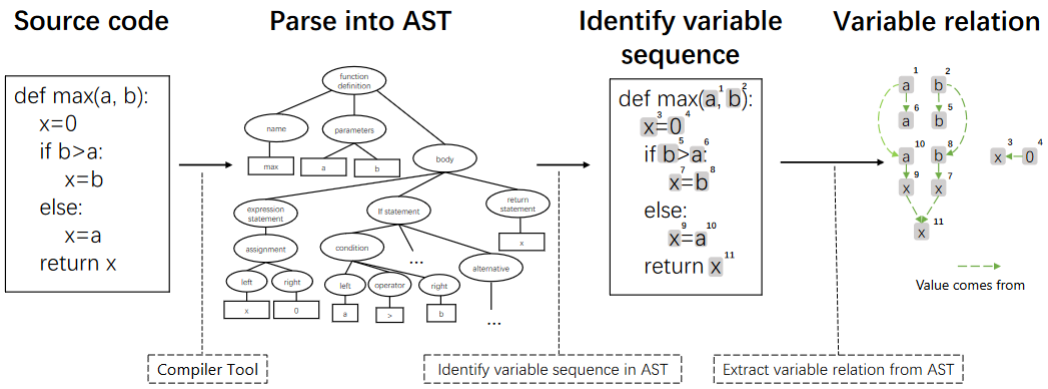
Data flow

Egyes programnyelveken látszólag egyszerű kódok AST-je is nagy lehet, ekkor nehezen kezelhető. Bizonyos programkód feldolgozó algoritmusok [2] data flow-t használnak AST helyett a beágyazáshoz.

A data flow olyan gráf, melyen a programban használt változók közötti összefüggéseket ábrázoljuk. Ha hivatkozunk egy változóra, akkor a hivatkozásnak megfelelő csúcsból fut egy él a legutóbbi értékadásnak megfelelő csúcsba. Amint a 2. ábra mutatja, a data flow az AST-ből kinyerhető, tehát kevesebb információt tartalmaz. Ennek ellenére akár az AST helyett, akár az AST-t kiegészítve alkalmazható a kód átfogóbb struktúrájának tömör reprezentálására.

Kitekintés

További önálló munkámban szeretnék a reprezentáción túl a feldolgozásra használt modellekkel (például transzformer, RNN) is mélyebben megismerkedni, valamint nagyobb hangsúlyt fektetni a gyakorlatra saját beágyazások tesztelésével különböző modelleken.



2. ábra. A data flow konstrukciója [2]

Hivatkozások

[1] Dasha Bogdanova és tsai. "Detecting semantically equivalent questions in online user forums". *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. 2015, 123–131. old.

[2] Daya Guo és tsai. "Graphcodebert: Pre-training code representations with data flow". *arXiv preprint arXiv:2009.08366* (2020).

[3] Shuai Lu és tsai. "CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation". *arXiv preprint arXiv:2102.04664* (2021).

[4] Lili Mou és tsai. "Convolutional neural networks over tree structures for programming language processing". *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[5] Hao Peng és tsai. "Building program vector representations for deep learning". *International conference on knowledge science, engineering and management*. Springer. 2015, 547–553. old.

[6] Richard Socher és tsai. "Grounded compositional semantics for finding and describing images with sentences". *Transactions of the Association for Computational Linguistics* 2 (2014), 207–218. old.