Automatická tvorba učebných plánov pre strojové učenie

Tomáš Kuzma, Igor Farkaš

Centrum pre kognitívnu vedu, KAI FMFI UK, Univerzita Komenského v Bratislave
Mlynská dolina, 842 48 Bratislava
Email: {tomas.kuzma,igor.farkas}@fmph.uniba.sk

Abstrakt

Pri tvorbe učebných plánov sa často používa jednoduchá zásada – na začiatku učenia sa študentovi predkladajú jednoduché príklady a s postupom času sa zaraďujú aj príklady ťažšie. Cieľom tohoto postupu je pomôcť správnemu zovšeobecňovaniu v danej úlohe a urýchliť a zjednodušiť proces učenia. S myšlienkou použiť túto zásadu aj na trénovanie umelých neurónových sietí prišiel Elman (1993), pričom zvýšil úspešnosť učenia na syntetickej jazykovej úlohe pomocou ručne vytvoreného plánu. Na túto prácu nadväzujú aj Bengio a spol. (2009), demonštrujúc efektivitu tejto metódy (pod názvom *curriculum learning*) na ručne vytvorených plánoch pre viacero modelov a syntetických úloh. V tejto práci navrhneme niekoľko metód, ako takýto učebný plán (*curriculum*) tvoriť automaticky, a otestujeme ich efektívnosť na syntetických, ale aj praktických problémoch.

1 Úvod

V oblasti vzdelávania sa pri tvorbe učebných plánov často využíva nasledujúci princíp: ako prvé sa v preberanej látke objavujú príklady, ktoré sú jednoduchšie, menšie, či majú menej súvislostí, a v priebehu výučby sa táto náročnosť zväčšuje. Úlohou tohoto postupu je najprv naučiť študentov základy, a potom postupne pridávať rôzne rozšírenia a špeciálne prípady, ktoré by inak mohli byť na začiatku mätúce.

Hypotézu, že takéto postupné učenie (*curriculum learning*) by mohlo byť prospešné aj pri učení (trénovaní) modelov v strojom učení, predstavil Elman (1993). Vo svojom prvom experimente učil jednoduchú rekurentnú neurónovú sieť s kontextovou vrstvou predikovať nasledujúce slovo vo vete, pričom tieto vety boli generované jednoduchou umelou gramatikou. Sieť sa podarilo úspešne natrénovať len vtedy, keď sa najprv trénovala na kratších a jednoduchších vetách. (Vo svojom druhom experimente tiež dosiahol podobne pozitívne výsledky inou metódou, kde postupný nárast zložitosti učenia nebol dosiahnutý zmenami distribúcie vstupných dát, ale postupným nárastom reprezentačných možností trénovaného modelu. Sieť počas trénovania zabúdala, čo bolo implementované pomocou mazania skrytej kontextovej vrstvy po istom počte krokov.)

V nedávnej dobe sa k tejto myšlienke pod názvom *curriculum learning* (*postupné učenie*) vrátili v rovnomennom článku Bengio a spol. (2009), kde k nej tiež poskytli formálnu definíciu – postupnosť distribúcií nad trénovacími príkladmi s narastajúcou entropiou – a demonštrovali jej efektivitu na zmesi syntetických úloh: hľadanie Bayesovského klasifikátora, klasifikácia lineárne oddeliteľných dát a rozpoznávanie geometrických útvarov; a tiež na predikcii reálneho jazyka, pri ktorej sa hľadá skóre pre možné pokračovania začatej vety. Vo všetkých týchto scenároch dosiahli mierne, no signifikantné zlepšenie schopnosti generalizácie.

V tomto článku sa ďalej budeme venovať skúmaniu metód, ktoré budú takéto učebné plány vytvárať bez ďalších znalostí či už o úlohe, alebo hlbšej štruktúre jej vstupných dát.

2 Tvorba plánu

Základom nášho prístupu je myšlienka, že lineárne oddeliteľné rozhodovacie problémy sú exaktne reprezentovateľné a ľahko naučiteľné pre skoro všetky modely používané v strojovom učení. Pre problémy, ktoré ale lineárne oddeliteľné nie sú, by v duchu postupného učenia dobrým začiatočným bodom mohla byť čo najväčšia lineárne oddeliteľná podmnožina vstupov.

Naším počiatočným problémom je teda rozhodovacia úloha, ktorej dve triedy nie sú (v priestore vstupov) lineárne oddeliteľné, a chceme z nich vybrať podmnožinu, ktorá by už lineárne oddeliteľná bola. Je zjavné, že táto formulácia úlohy je veľmi nejednoznačná. Keby sme uvažovali podmienku, že vybraná množina má byť najväčšia spomedzi všetkých vyhovujúcich, narazíme na problém – úloha síce má jednoznačné riešenie, no už nie je efektívne riešiteľná.

Približná verzia nášho problému sa ale rieši implicitne pri trénovaní akéhokoľvek lineárneho klasifikátora – po natrénovaní klasifikátora sú totiž dáta, ktoré klasifikuje správne, lineárne oddeliteľné. Pre dobre natrénovaný klasifikátor bude navyše táto množina *približne* najväčšia možná (no nie vždy exaktne najväčšia). Z dôvodov efektivity sa teda budeme musieť uspokojiť s takýmto rýchlym približným riešením.

2.1 Stroje s podpornými vektormi

S veľmi efektívnym riešením problému (nielen) lineárnej klasifikácie prišli Cortes a Vapnik (1995) vo svojom modeli *stroj s podpornými vektormi* (*Support Vector Machine*), kde sa hľadá oddeľujúca nadrovina, od ktorej sú všetky trénovacie body čo najďalej, a to v správnom smere (taká môže zjavne existovať len pre lineárne oddeliteľné problémy); vo variante s *mäkkými okrajmi* (*soft margin*) sa navyše umožňuje túto vlastnosť porušovať, ale takéto porušenie je penalizované.

Ak si označíme počet vstupov n, dimenziu vstupov d, vstupy $\mathbf{x}_i \in \mathbb{R}^d$ a triedy $y_i = \pm 1$, našou úlohou je nájsť optimálne hodnoty pre koeficienty deliacej roviny ($\mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R}$) a vo variante *s mäkkými okrajmi* navyše aj hodnoty pre vedľajšie[1] premenné ($\xi_i \in \mathbb{R}_0^+$). Za cieľ si volíme maximalizovať vzdialenosť medzi triedami a navyše vedľajšie premenné penalizujeme regularizačným parametrom $C \in \mathbb{R}^+$; dostávame teda tento kvadratický program:

$$\text{min:} \qquad \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s podmienkou:} \qquad y_i(\mathbf{w}\cdot\mathbf{x}_i - b) \geq 1 - \xi_i \quad (\forall\, 1 \leq i \leq n)$$

Nakoľko takto sformulovaný kvadratický problém je pozitívne definitný, je možné ho efektívne riešiť (všeobecné kvadratické programovanie je NP-ťažké) niekoľkými spôsobmi, a to v primálnej i duálnej forme.

2.2 Rozhodovacie úlohy

Náš postup teda spočíva v natrénovaní stroja s podpornými vektormi na vstupných dátach a použitím tohoto modelu na vybratie *jednoduchších* trénovacích vzorov z celkovej množiny. Pre každý vstupný vzor si vypočítame jeho orientovanú vzdialenosť od rozhodovacej nadroviny (ďalej len *okraj*), pričom vzory ležiace ďalej od tejto roviny v správnom smere budeme považovať za jednoduchšie. Do sady jednoduchších dát pripustíme iba vzory, kde tento *okraj* je väčší ako nejaká hodnota – *prah*. Pre kladné hodnoty tohoto *prahu* dostaneme jednoduchšiu množinu, ktorá je už lineárne oddeliteľná; záporné hodnoty tohto parametra vyberajú aj už lineárne neoddeliteľné vstupy, no stále zahadzujú viac *vytŕčajúce dáta* (*outliers*).

2.3 Klasifikačné úlohy

Pre úlohy, ktoré majú $k > 2$ tried, už takýto jednoduchý postup nie je možný – viac tried sa nedá oddeliť použitím jednej roviny. Dá sa ale uvažovať nad použitím súboru viacerých lineárnych klasifikátorov, ktorého jednotlivé binárne rozhodnutia budú určovať výslednú klasifikáciu.

[1] Tieto nám umožňujú, aby niektoré body boli bližšie k rovine, ako majú byť (geometrická vzdialenosť $\geq \frac{1}{\|\mathbf{w}\|}$). Vo formulácii *SVM* bez mäkkých okrajov sa tieto premenné nenachádzajú (sú rovné nule).

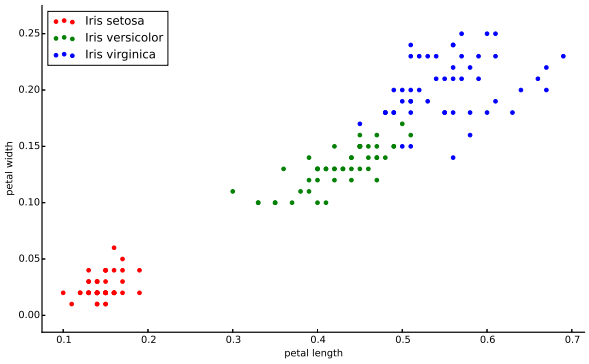
Ukázalo sa, že zmena vzájomnej početnosti jednotlivých tried má zásadný vplyv na úspešnosť trénovania; do našej sady jednoduchších dát preto budeme brať zástupcov všetkých tried rovnomerne.

Budeme skúmať dva takéto prístupy: *one–vs–rest*, ktorý používa k klasifikátorov, a *one–vs–one*, ktorý používa $\binom{k}{2}$ klasifikátorov.

2.3.1 One–vs–rest

V jednoduchšom variante sa pre k vstupných tried natrénuje k klasifikátorov, pričom i-ty oddeľuje triedu i od zvyšných. Výsledným *okrajom* pre vzor patriaci do i-tej triedy je v tomto prípade vzdialenosť daného bodu od i-tej rozhodovacej nadroviny. Výhodou tohoto postupu je jeho jednoduchosť a výpočtová nenáročnosť, no iba veľmi málo problémov sa dá úspešne riešiť takýmto súborom k klasifikátorov.

Obmedzenia tejto metódy si môžeme ilustrovať napríklad na známej dátovej sade *Iris flower data set* (Fisher, 1936; Anderson, 1935) (pozri obr. 1). Oddeliť červené body od zvyšných je možné exaktne, modré body od zvyšných s vysokou presnosťou. Zelené body sú ale obkolesené zvyšnými triedami z dvoch strán a ani približné oddelenie nie je možné. (Zobrazené sú iba posledné dva rozmery; oddelenie je ale podobne beznádejné aj s použitím všetkých štyroch rozmerov.)



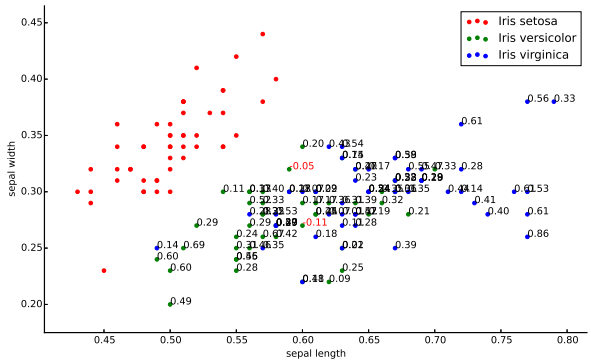
Obr. 1: Iris flower data set, zobrazené sú posledné dva rozmery: šírka a dĺžka okvetných lístkov.

2.3.2 One–vs–one

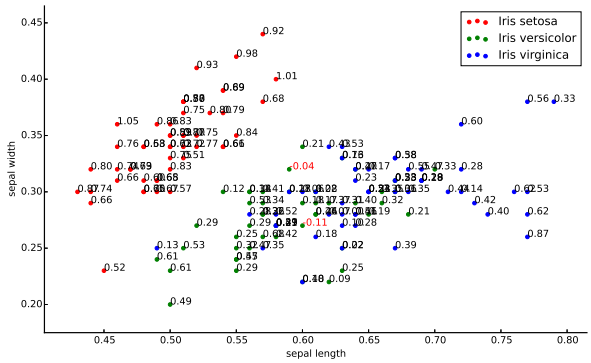
Zmysluplnejšie *okraje* sa dajú dosiahnuť trénovaním klasifikátorov, ktoré oddeľujú dvojicu tried i a j, takýchto klasifikátorov ale potrebujeme $\binom{k}{2} = O(k^2)$. Výhodou je, že každý klasifikátor počas svojho trénovania vidí len časť vstupných dát, a nie všetky, ako v prípade prístupu *one–vs–rest*, čo znižuje výpočtovú náročnosť trénovania veľkého množstva klasifikátorov.

Situáciu, kde mala metóda *one–vs–rest* ťažkosti, metóda *one–vs–one* rieši uspokojivo. Oddelenie dvojíc tried prebehne bez problémov, najťažšie je oddelenie *Iris versicolor* a *Iris virginica* (pozri obr. 2); výsledné skóre

ukazuje obr. 3. Je dobré si všimnúť, že konečné *okraje* sú celkom intuitívne: pre vzory na okrajoch tried sú vysoké, a na miestach, kde sa triedy prelínajú, sa pohybujú okolo nuly.



Obr. 2: Nedokonalé lineárne oddelenie kvetov *Iris versicolor* od kvetov *Iris virginica* – dva zo sto kvetov sa nachádzajú na nesprávnej strane oddeľovacej nadroviny. (Zobrazená je projekcia na prvé dva vstupné atribúty zo štyroch: šírka a dĺžka kališných listov; čísla nad bodmi znázorňujú vzdialenosť od oddeľujúcej nadroviny; záporné okraje sú kreslené červenou.)



Obr. 3: Konečné skóre pre všetky vstupné vzory. (Zobrazená je projekcia na prvé dva vstupné atribúty zo štyroch: šírka a dĺžka kališných listov; čísla nad bodmi znázorňujú vzdialenosť od oddeľujúcej nadroviny; záporné okraje sú kreslené červenou.)

3 Pokusy na syntetických úlohách

Úspešnosť nášho postupu sme skúmali na umelej úlohe klasifikácie geometrických tvarov, ktorú predstavili (Bengio a spol., 2009): cieľom je klasifikovať obrázky v odtieňoch šedej s veľkosťou 32×32 bodov do troch kategórií (elipsy, obdĺžniky a trojuholníky). Pre túto úlohu vytvorili aj učebný plán, čo nám umožňuje porovnať jeho efektivitu s naším učebným plánom.

V snahe priblížiť sa čo najviac pôvodným experimentom sme robili pokusy na umelej neurónovej sieti obsahujúcej tri skryté vrstvy (so *soft-sign* aktiváciou) po 400 neurónoch nasledované výstupnou *softmax* vrstvou (jednotlivé triedy boli reprezentované pomocou *one-hot* kódovania). Sieť sme trénovali *metódou poklesu gradientu* s využitím vysokého momenta po 256 epoch[2]. Výsledky pre 50 behov sú uvedené v tabuľke 1.

variant	estimácia	validácia	testovanie
bez plánu	13.43±0.98	29.52±0.94	29.76±1.00
one–vs–one	9.92±1.03	24.38±1.48	24.63±1.60
Bengiov plán	7.60±1.48	23.27±1.09	23.75±1.20

Tab. 1: Priemerné klasifikačné chyby (v %) dosiahnuté na pôvodnej verzii úlohy GeomShapes (odtiene šedej).

Ukázalo sa, že pre lineárny model je mimoriadne náročné mať v jednej triede aj svetlé útvary na tmavom pozadí, aj naopak. Vytvorili sme teda zjednodušenú monochromatickú verziu dát, kde je útvar vždy biely a pozadie vždy čierne. Táto úloha sa prekvapivo ukázala byť omnoho jednoduchšia aj pre nelineárnu neurónovú sieť; výsledky 50-tich behov sumarizuje tabuľka 2.

variant	estimácia	validácia	testovanie
bez plánu	0.03±0.06	8.84±0.41	9.57±0.48
one–vs–one	0.04±0.03	8.84±0.29	9.47±0.35
Bengiov plán	0.04±0.03	8.90±0.42	9.53±0.51

Tab. 2: Priemerné klasifikačné chyby (v %) dosiahnuté na monochromatickej verzii úlohy GeomShapes.

4 Pokusy na praktických úlohách

Ako praktickú úlohu sme si vybrali štandardnú sadu rukou písaných číslic MNIST (LeCun a Cortes, 2010). V tejto úlohe je potrebné klasifikovať číslice 0 až 9, ako vstup slúžia obrázky, ktoré majú majú 28×28 obrazových bodov v odtieňoch šedej. Na testovanie sme použili rovnaký postup a architektúru ako pri predchádzajúcej úlohe, nakoľko majú podobný charakter a typ vstupu. Jedinou zmenou je zmenšenie počtu vstupných neurónov z 32×32 na 28×28 a zníženie počtu epoch na 60. Výsledky pre 50 nezávislých behov uvádza tabuľka 3.

variant	estimácia	validácia	testovanie
bez plánu	0.03±0.03	1.63±0.10	1.66±0.08
one–vs–one	0.02±0.02	1.68±0.11	1.69±0.08

Tab. 3: Priemerné klasifikačné chyby (v %) dosiahnuté na úlohe MNIST; na validáciu sme vybrali náhodnú šestinu trénovacích dát.

[2] V prípade Bengiovho plánu sme trénovali sieť 128 epoch na ľahších a následne ďalších 128 epoch na ťažších dátach; pre náš plán, ktorý má v jednoduchej množine menej dát ako Bengio, sa ukázalo byť vhodné zvoliť dĺžku trénovania 16 + 240 epoch.

5 Implementačné detaily

Na efektívne trénovanie lineárnych SVM používame knižnicu `liblinear` (Fan a spol., 2008). Trénovanie na d-rozmernej dátovej množine obsahujúcej n vzorov má časovú zložitosť $O(d \cdot n)$, zatiaľ čo trénovanie všeobecnejších SVM má zložitosť $O(d \cdot n^2)$.

Pri použití metódy *one–vs–rest* sa trénuje k klasifikátorov – jeden pre každú triedu. Trénovanie jedného takéhoto klasifikátora má zložitosť $O(d \cdot n)$, pričom všetky trénovania sú nezávislé a dajú sa ľahko paralelizovať. Celková zložitosť je teda $O\left(k \cdot n \cdot d\right)$.

Pri použití metódy *one–vs–one* trénujeme dokonca $\binom{k}{2} = O(k^2)$ klasifikátorov, pričom každý oddeľuje nejakú triedu i od inej triedy j; tieto klasifikátory sa takisto dajú trénovať paralelne. Celková zložitosť je teda $O(k^2 \cdot d \cdot n)$, ale za predpokladu, že v každej triede je zhruba rovnaké množstvo dát, $O(n/k)$, čo platí napríklad pre všetky dátové množiny použité v tomto článku, je možné získať jemnejší odhad. Pre trénovanie klasifikátora rozhodujúceho sa medzi triedami i a j sú totiž potrebné len dáta patriace do týchto tried, ktorých nie je n, ale len $2 \cdot O(n/k)$ – získavame teda zložitosť:

$$O\left(\binom{k}{2} \cdot \tfrac{2}{k} n \cdot d\right) = O(k \cdot n \cdot d)$$

Na trénovanie dopredných neurónových sietí používame vlastnú implementáciu v jazyku Python, pričom na samotné výpočty používame knižnicu Theano (Bergstra a spol., 2010; Bastien a spol., 2012) – tá oproti naivnej implementácii prináša niekoľko výhod, vrátane optimalizácií na numerickú stabilitu a rýchlosť výpočtu a umožňuje *Just-in-Time* generovanie a kompiláciu zdrojového kódu pre výpočty. Výsledný strojový kód môže byť efektívne vykonávaný priamo na procesore (pričom využíva špecifickú inštrukčnú sadu konkrétneho procesora, dosahujúc zhruba 5-násobné zrýchlenie), alebo pomocou grafickej karty (ktorá dokáže mnohé operácie vykonávať paralelne na stovkách špecializovaných výpočtových jadier, s praktickým zrýchlením v ráde desiatok až stoviek).

6 Záver

Pokusy na syntetických dátach ukázali, že s použitím automaticky vytvoreného učebného plánu je možné dosiahnuť zlepšenie na skoro tak vysokej úrovni, ako s ručne vytvoreným plánom. Vytvorili sme tiež modifikovanú verziu tejto úlohy, ktorá je pre lineárny klasifikátor jednoduchšia. V tejto verzii získané okraje lepšie popisujú zložitosť jednotlivých vstupov a preto automaticky vytvorený plán dosahuje dokonca lepšie výsledky, ako plán vytvorený ručne.[3] Ďalšie skúmanie možno odhalí, prečo nám automaticky tvorený plán na praktických dátach nepriniesol žiadne zlepšenie.

[3]ktorý bol ale vytvorený pre nemodifikovanú úlohu

Pôvodný článok (Elman, 1993) podrobili ostrej kritike Rohde a Plaut (1999), pričom ukázali, že na ich variantoch pôvodnej úlohy a pôvodného modelu postupné učenie nielen nepomáha, ale dokonca výsledky zhoršuje. Dá sa teda predpokladať, že techniky postupného učenia nie sú úplne univerzálne a nedajú sa úspešne použiť na ľubovoľný typ úloh.

Poďakovanie

Tento príspevok vznikol za podpory grantovej agentúry VEGA v rámci grantovej úlohy 1/0898/14 a grantovej agentúry KEGA v rámci grantovej úlohy 076UK-4/2013.

Literatúra

Anderson, E. (1935). The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N. a Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y., Louradour, J., Collobert, R. a Weston, J. (2009). Curriculum learning. V *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, str. 41–48.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. a Bengio, Y. (2010). Theano: a CPU and GPU math compiler in python. V *Proceedings of the 9th Python in Science Conference (SciPy 2010)*, str. 3–10.

Cortes, C. a Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. a Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188.

LeCun, Y. a Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

Rohde, D. L. T. a Plaut, D. C. (1999). Language acquisition in the absence of explicit negative evidence: how important is starting small? *Cognition*, 72:67–109.