Vincent Kríž

Rozpoznávanie sémantických kolokácií

(Záverečná správa)

Popis zadania

V tejto správe popisujem svoje riešenie úlohy Rozpoznávanie sémantických kolokácií. Podstatou úlohy je vytvorenie binárneho klasifikátora, ktorý o zadanej inštancii určí, či sa jedná o sémantickú kolokáciu alebo nie.

Všeobecne termín kolokácia popisuje zmysluplné a gramaticky správne slovné spojenia, ktoré sa často vyskytujú v prirodzenom jazyku.

Sémantická kolokácia tvorí navyše ešte sémantickú jednotku. Táto jednotka navyše nemôže byť úplne určená podľa významov jednotlivých komponentov, z ktorých sa skladá. Preto sémantické kolokácie musia byť vymenované v slovníku.

Uvediem príklad. Kolokácia „čierny kôň“ je považovaná za „jednoduchú“ kolokáciu, pretože význam objektu, ktorý popisuje môžeme zložiť z významov jednotlivých komponentov - kôň, ktorého popisujeme má čiernu farbu srsti. Naopak, kolokácia „čierny trh“ nepopisuje trh, ktorý má čiernu farbu, ale slovo čierny v tomto prípade odkazuje na nelegálnosť trhu.

Úloha rozpoznávania sémantických kolokácií je tradičnou úlohou v korpusovej lingvistike. Cieľom úlohy je získať zoznam sémantických kolokácií z textového korpusu, z ktorého sa následne vytvorí lexikón kolokácií. Ten sa môže následne použiť v mnohých oblastiach počítačovej lingvistiky.

Popis vstupných dát

Pri riešení úlohy budem používať dodané údaje získané z tektogramatickej roviny Pražského závislostného korpusu 2.0. Dodané kolokácie sú reprezentované ako *feature vectors*, ktoré obsahujú niekoľko druhov informácií:

- Lemma, morfologické a syntaktické informácie
  - každé slovo v kolokácii má uvedenú svoju lemmu (stĺpce l1 a l2), morfologický tag (t1, t2) a analytickú funkciu (a1, a2)
- Štatistické údaje o počte výskytov kolokácie (f, b1 - b4)
- 82 rôznych štatistických mier vypočítaných na základe počtu výskytov kolokácií v korpuse (x1 - x82)
- Výsledné hodnotenie troch nezávislých anotátorov (a, b, z)
- Celkové hodnotenie (tp)

Pri riešení nebudem používať rysy a, b, z a l1 a l2. Celkové hodnotenie totiž závisí na hodnotení troch anotátorov, takže klasifikátor by okamžite získal úspešnosť 100%. Kolokácia je označená za sémantickú, ak sa na tom zhodnú všetci traja anotátori. Lemmy kolokácií som sa rozhodol nepoužívať z dôvodu väčšej všeobecnosti klasifikátora.

Rozdelenie vstupných dát

Dodané údaje obsahujú 9232 kolokácií. Pre potreby natrénovania a otestovania klasifikátorov som sa rozhodol rozdeliť dáta nasledujúcim spôsobom.

Najskôr údaje náhodne premiešam podľa nejakej permutácie. Následne oddelím 232 kolokácií ako testovacie dáta. Tie použijem pri záverečnom vyhodnocovaní úspešnosti klasifikátorov. Zvyšných 9000 inštancií budem používať pri trénovaní a tuningu parametrov pomocou cross-validation:

DEVELOPMENT		TEST
TRAIN	HELDOUT	
Dáta použijem pri trénovaní klasifikátora	*Dáta použijem pri tuningu parametrov klasifikátora*	*Dáta použijem pri záverečnom vyhodnotení*
8100 (91%)	**900 (7%)**	**232 (2%)**

Teoretický popis použitých metód

V práci postupne natrénujem a otestujem 3 druhy klasifikátorov:

1. Rozhodovacie stromy (DT)
2. Instance-based learning (kNN)
3. Support Vector Machines (SVM)

Ako baseline použijem naivný klasifikátor, ktorý najskôr zistí koľko je v trénovacom súbore sémantických kolokácií a potom na základe toho označí všetky testovacie dáta rovnako, podľa toho, či bolo v trénovacom súbore viac sémantických kolokácií alebo viac „obyčajných“ kolokácií.

Rozhodovacie stromy (DT)

Rozhodovací strom je zakorenený strom, ktorého vnútorné uzly obsahujú otázku, na základe ktorej sa vstupné dáta delia do jednotlivých synov. Koncové (terminálne) uzly, alebo listy obsahujú konečnú triedu, ktorú klasifikátor priradí inštanciám, ktoré sa do listu dostali.

Na rozhodovací strom môžeme nazerať ako na štruktúru, ktorá rozdeľuje vstupné dáta na disjunktné skupiny. Koreň stromu obsahuje všetky dáta. Každý syn koreňa obsahuje takú podmnožinu dát, ktorá zodpovedá jeho „odpovedi“ na otázku stanovenú koreňom.

Pri vytváraní rozhodovacích stromov je najdôležitejšou úlohou vybrať otázku, ktorá sa v spracovávanom uzle použije. Existuje niekoľko metód, ktoré sa snažia určiť, ktorý atribút najlepšie rozdeľuje vstupné dáta. Najznámejšie metódy sú

- misclassification error
- information gain
- gini index

Ak umožníme, aby bol rozhodovací strom príliš veľký, je možné, že nastane neželané pretrénovanie. Pri tomto jave má strom vysokú úspešnosť na trénovacích dátach, ale malú na testovacích. Je to spôsobené tým, že strom nevyjadruje všeobecný model pre zadanú úlohu, ale je len modelom zadaných vstupných dát.

Riešením tohto problému je ovplyvňovanie algoritmu na tvorbu stromu tým, že obmedzíme hĺbku stromu, počet inštancií, ktoré môžu byť minimálne v jednom uzle, alebo najskôr vytvoríme košatý strom, ktorý následne vhodným spôsobom „orežeme“, pomocou metód prunningu.

Instance-based learning (kNN)

Jedná sa o tzv. „lazy“ metódu, ktorá trénovacie údaje nijak nespracováva a po zadaní dotazu na

novú inštanciu použije pre vyhodnotenie istú podmnožinu trénovacích údajov, ktoré sú zadanej inštancii najviac „podobné“.

Podobnosť možno najlepšie ukázať na kvantitatívnych rysoch, kde nad vektorovým priestorom môžeme definovať napríklad euklidovskú metriku. Dva vektory potom budeme považovať za podobné, ak bude ich vzdialenosť malá. Pomocou definovanej metriky môžeme potom hovoriť o „najbližších susedoch“ zadanej inštancie.

Algoritmus kNN (k-Nearest Neighbor) pre zvolenú inštanciu nájde k najbližších susedov a ohodnotí ju na základe ohodnotenia susedov. V základnej verzii tohto algoritmu má každý sused rovnakú váhu pri určovaní ohodnotenia zadanej inštancie. Rozšírená verzia algoritmu považuje vzdialenejších susedov za menej podstatných ako susedov bližších.

Jadrom tejto metódy je vhodné zvolenie metriky, pomocou ktorej sa vypočítajú vzdialenosti medzi inštanciami. Pri úlohách s veľkým počtom rysov môže nastať jav, ktorý sa nazýva „prekliatie dimenzionality“. Menej podstatné rysy môžu výrazne ovplyvňovať výslednú vzdialenosť na úkor viac podstatných. Ďalším problémom *mnohodimenzionálnych* priestorov je, že vzdialenosti medzi jednotlivými inštanciami sú veľmi veľké a tak v podstate neexistujú „blízki susedia“, len „vzdialení“.

Support Vector Machines (SVM)

Hlavnou myšlienkou tejto metódy je hľadanie optimálnej nadroviny vo vektorom priestore inštancií, ktorá priestor rozdelí na 2 časti, ktoré zodpovedajú binárnemu ohodnoteniu inštancií, ktoré sa nachádzajú v tomto priestore.

Ak sú údaje separovateľné pomocou nejakej takejto nadroviny, potom existuje nekonečne veľa takýchto nadrovín. Metóda SVM pomocou riešenia optimalizačnej úlohy vyberie takú nadrovinu, aby maximalizovala vzdialenosť najbližších bodov z každej časti priestoru od tejto nadroviny.

Ak dáta nie sú separovateľné, možno zostrojiť nadrovinu, ktorá rozdelí inštancie, ale umožní i existenciu chybne ohodnotených inštancií (tzv. „slacks“). Ďalšou možnosťou je prevedenie vektorového priestoru do vyšších dimenzií použitím tzv. „kernelových funkcií“, ktoré vypočítajú nové súradnice pre každú inštanciu s nádejou, že vo vyšších dimenziách budú dáta separovateľné.

Spôsob vyhodnotenia výsledkov

Pri hodnotení výsledkov klasifikátorov sa používajú miery *accuracy, precision, recall* a *F-measure*. Všetky sa dajú vypočítať z tzv. *confusion matrix*, ktorá má nasledujúcu podobu:

		PREDICTED	
		YES	NO
TRUE CLASSIFICATION	YES	TP	FN
	NO	FP	TN

Jednotlivé bunky označujú:

- TP (*true predicion*) - počet správnych pozitívnych ohodnotení
- FN (*false negative*) - počet nesprávnych negatívnych ohodnotení
- FP (*false pozitive*) - počet nesprávnych pozitívnych ohodnotení
- TN (*true negatives*) - počet správnych negatívnych ohodnotení

Na základne týchto štyroch hodnôt sa jednotlivé miery vypočítajú pomocou týchto vzorcov:

- Accuracy - $\frac{TP+TN}{TP+FP+FN+TN}$
- Recall - $\frac{TP}{TP+FN}$
- Precision - $\frac{TP}{TP+FP}$
- F-measure - $\frac{1}{\alpha P^{-1}+(1-\alpha) R^{-1}}$

Pre záverečné rozhodovanie, ktorý z použitých klasifikátorov je najlepší, budem používať mieru Accuracy.

Výsledky použitých metód

V nasledujúcej časti uvádzam postup zostavenia, trénovania a testovania použitých klasifikátorov.

Naivný klasifikátor pre určenie baseline

Naivný klasifikátor najskôr zistí, či sa v trénovacích údajoch nachádza viac pozitívnych alebo negatívnych ohodnotení a podľa toho ohodnotí celý súbor testovacích údajov.

Výsledky tohto klasifikátora použijeme ako baseline pre ďalšie experimenty:

Accuracy (train)	0.789
Accuracy (test)	0.797

Tabulka 1: Výsledky funkcie Kriz.R::naive_classifier()

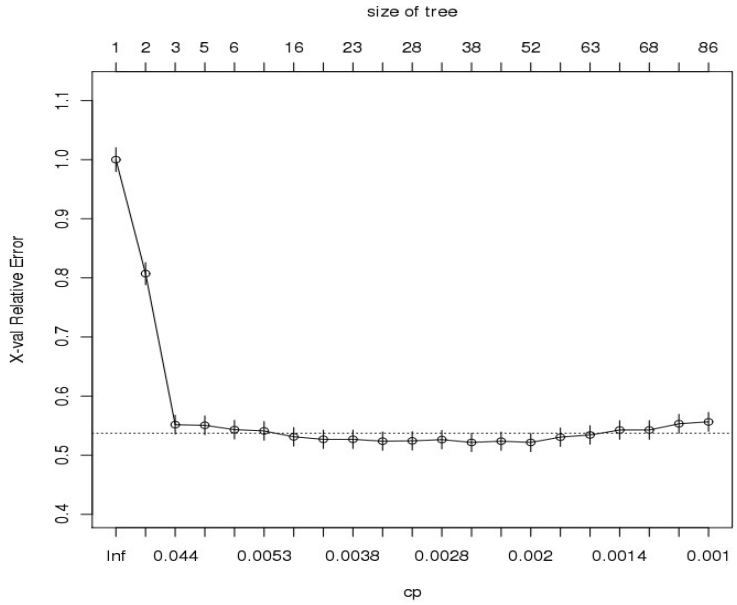
Rozhodovacie stromy

Model bez akéhokoľvek tuningu dosiahol úspešnosť

Accuracy (train)	0.884
Accuracy (test)	0.862

Tabulka 2: Výsledky funkcie Kriz.R:dt_base()

Ako prvý parameter som sa rozhodol upraviť CP. Pri hodnote cp=0.001 môžeme volaním funkcie plotcp() získať nasledujúci graf:



Z grafu môžeme vidieť, že relatívna chyba a počet vrcholov stromu je najnižší asi pri hodnote 0.0038. Následne sa udržiava približne na rovnakých hodnotách a rastie len počet vrcholov stromu, až okolo hodnoty 0.002 začína opať stúpať.

Zaujímavé je, že znižovaní hodnoty cp stúpa úspešnosť na trénovacích dátach a klesá na testovacích. Už pri tejto hodnote cp tak môžeme vidieť jav overfittingu, kedy klasifikátor modeluje presne trénovacie dáta ale testovacie už nedokáže modelovať dostatočne dobre.

Za hodnotu cp teda zvolím 0.0038. Pri tejto hodnote je strom ešte dostatočne malý, aby sme ho mohli považovať za obecný model.

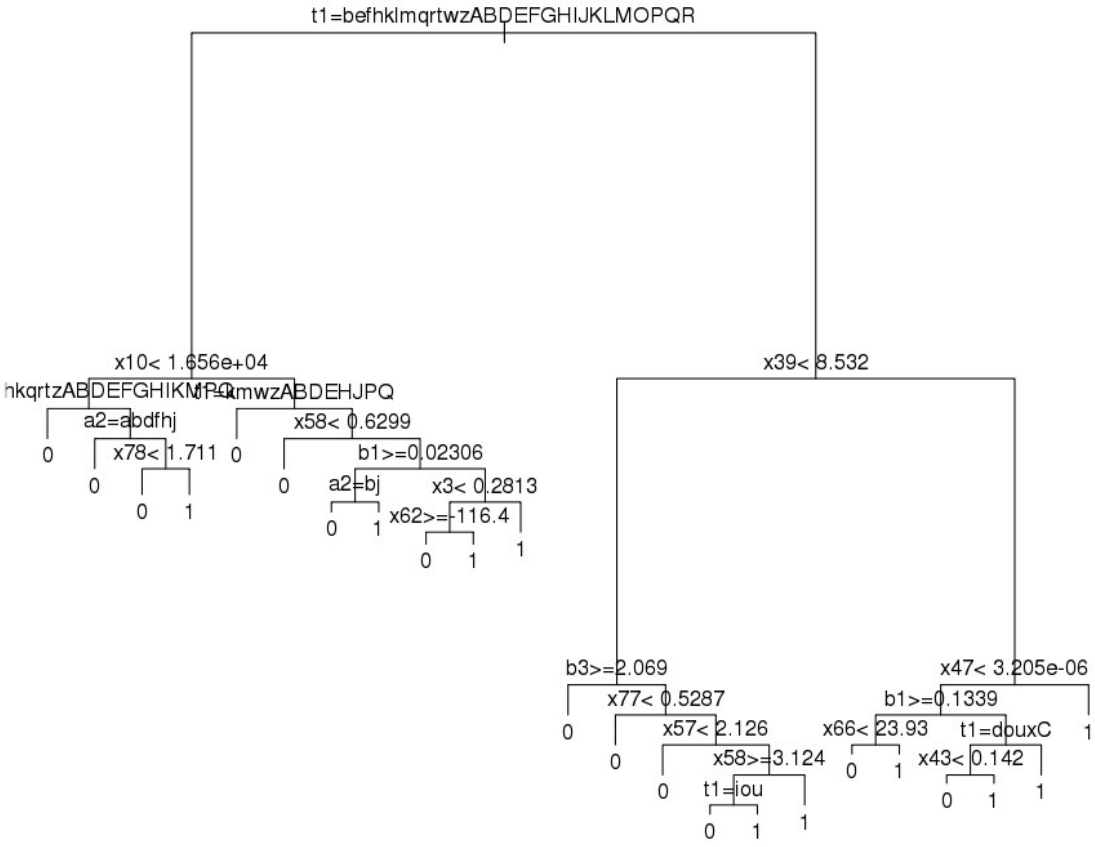
Ako ďalší som sa rozhodol optimalizovať parameter minsplit, ktorý určuje, koľko inštancií môže daný uzol reprezentovať minimálne. Vytvoril som funkciu **`dt_minsplit()`**, ktorá postupne vytvára model s rôznymi hodnotami minsplitu, v rozmedzí od 1 do 100 a každý vyhodnocuje. Získané úspešnosti sa s rôznymi minsplitmi líšia až na mieste tisícin, preto hodnotu minsplit ponechávam na defaultnej hodnote 20.

Posledným experimentom, ktorý som s modelom spravil, bolo nastavenie rôznych hodnôt parametra maxcompete, ktorý určuje koľko rôznych delení sa vyskúša, kým sa v danom uzle vyberie najlepšie možné delenie. Defaultná hodnota je 4, hodnotu som menil v rozsahu 1 až 100 a model nezaznamenal žiadne výrazne zmeny v úspešnosti oproti defaultnej hodnote.

Klasifikátor sémantických kolokácií pomocou metódy rozhodovacích stromov má teda výslednú podobu:

```
rpart(tp ~ ., data, cp=0.0038)
```

Výsledný rozhodovací strom má takúto podobu:



Aby som získal štatisticky relevantné údaje o úspešnosti klasifikátora, nad trénovacími údajmi spustím 10-fold cross-validation. Údaje rozdelím na 10 častí, v každom z desiatich pokusov natrénujem model na 9 častiach údajov a otestujem na zvyšnej časti údajov. Pomocou získaných vektorov vypočítam 95% konfidenčné intervaly pre úspešnosť klasifikátora. Výsledky uvediem na záver správy, kde porovnám všetky klasifikátory.

Instance-based learning (kNN)

Pred spustením algoritmu kNN som upravil vstupné údaje tak, aby obsahovali len číselné údaje. Použil som len stĺpce x1 - x82 a stĺpce f1, b1, b2, b3 a b4. Metóda bez tuningu mala nasledujúce úspešnosti:

Accuracy (train)	1
Accuracy (test)	0.836

Tabulka 3: Výsledky funkcie Kriz.R:knn_base()

Prvým experimentom, ktorý som s modelom skúsil bola normalizácia rysov vo feature vectoroch. Každý číselný rys v dodaných údajoch má rôzne rozpätie hodnôt. Keďže kNN používa euklidovskú metriku, rysy s vysokými hodnotami budú tvoriť väčšiu časť vo výslednej vzdialenosti inštancií ako rysy s malými hodnotami. Pomocou funkcie scale() som všetky rysy znormalizoval. Každý z rysov

bude mať rovnaké rozpätie, pričom stredná hodnota rysu bude 0 a variabilita 1. Model s normalizovanými rysmi bez ďalšieho tuningu si zlepšil úspešnosť nad testovacími údajmi o 2 percentá:

Accuracy (train)	1
Accuracy (test)	0.857

Tabulka 4: Výsledky funkcie Kriz.R:knn_scaling()

Ďalším experimentom bolo skúšanie rôznych hodnôt k. Defaultná hodnota parametra je 1, čo znamená, že pri ohodnotení zadanej inštancie sa uvažuje len jeden (najbližší) sused. V skripte postupne skúšam zvyšovať počet susedov až na hodnotu 50. Najlepšiu úspešnosť dosiahol model pri hodnotách 25 až 28 susedov. Úspešnosť modelov sa pre tieto hodnoty k líši až v desatinách percenta úspešnosti, preto som zvolil najnižšiu hodnotu k=25:

Accuracy (train)	0.886
Accuracy (test)	0.900

Tabulka 5: Výsledky funkcie Kriz.R::knn_tuning_k()

Podobne ako v prípade klasifikátora pomocou metódy rozhodovacích stromov som aj v prípade kNN spravil 10-fold cross-validation nad trénovacími údajmi. Výsledky validácie uvediem na konci správy.

Support vector machines (SVM)

Klasifikátor sémantických kolokácií pomocou metódy SVM s polynomiálnym kernelom bez tuningu parametrov má úspešnosti:

Accuracy (train)	0.871
Accuracy (test)	0.862

Tabulka 6: Výsledky funkcie Kriz.R::svm_base()

Tento základný model som sa snažil vylepšiť nastavením hodnôt cost, degree a gamma. Hodnotu cost som testoval pomocou skriptu **`svm_tuning_cost()`**, ktorý za parameter postupne dosadzuje hodnoty od 100 do 1000 v kroku po 100. Najlepší výsledok model dosiahne pri hodnote 400:

Accuracy (train)	0.941
Accuracy (test)	0.905

Tabulka 7: Výsledky funkcie Kriz.R::svm_tuning_cost()

Ďalším parametrom, ktorý som skúšal vyladiť bola hodnota degree. Pri zmene tejto hodnoty z defaultných 3 na inú (väčšiu i menšiu) model vykazoval horšiu úspešnosť. Hodnotu degree som preto zachoval na čísle 3.
Hodnota gamma je defaultne definovaná ako 1/počet dimenzií dodaných údajov. Vytvoril som skript, ktorý postupne vytvoril, natrénoval a otestoval niekoľko modelov s rôznymi hodnotami gamma. Ani jeden z nich však nepresiahol úspešnosť modelu s defaultnou hodnotou gamma. Výsledný klasifikátor pomocou metódy SVM má teda tvar

```
svm(tp ~ ., train, kernel="polynomial", type="C-classification",
                                        cost=400)
```

Klasifikátor dosiahol tieto úspešnosti:

Accuracy (train)	0.941
Accuracy (test)	0.905

Tabulka 8: Úspešnosť SVM po tuningu parametrov

Podobne ako v predchádzajúcich modeloch použijem 10-fold cross-validation na získanie štatisticky signifikantných výsledkov.

Porovnanie klasifikátorov

Nad každým z vytvorených klasifikátorov som spustil 10-fold cross-validation nad trénovacími údajmi. Tu sú výsledky:

		DT	**kNN**	**SVM**
Sady dát pri cross-validation	1	0.877	0.870	0.887
	2	0.882	0.863	0.882
	3	0.883	0.880	0.906
	4	0.890	0.875	0.907
	5	0.891	0.873	0.899
	6	0.892	0.888	0.901
	7	0.896	0.873	0.886
	8	0.897	0.892	0.908
	9	0.900	0.892	0.893
	10	0.902	0.864	0.883
AVG(AC)		0.891	0.877	0.895
Konf. interval		0.885 - 0.897	0.869 - 0.885	0.889 - 0.901
Accuracy(TEST)		0.870	0.900	0.905

Posledný riadok Accuracy(TEST) v tabuľke zobrazuje výsledky nad testovacími údajmi, ktoré som v cross-validácii nepoužil.

Z výsledkov môžeme okamžite prehlásiť model SVM za najúspešnejší. Modely DT a kNN majú prekrývajúce sa konfidenčné intervaly, preto použijem párový t-test na zistenie, ktorý z modelov je lepší.

Podľa párového t-testu zavolaného nad vektormi výsledkov cross-validácie s hypotézou, že stredná hodnota rozdielu je 0, je lepším modelom DT. Ako však môžeme vidieť, nad testovacími údajmi mal model kNN väčšiu úspešnosť o tri percentá.

Za najlepší klasifikátor teda prehlasujem model SVM.

Ako použiť najlepší klasifikátor

Súčasťou tejto správy je skript Best.R, ktorý obsahuje funkcie potrebné pre načítanie dát, natrénovanie klasifikátora a ohodnotenie zadaných testovacích údajov. Tu sú presné pokyny, ako dospieť až k faktoru s ohodnotením testovacích údajov:

1. Načítajte skript **`Best.R`** do prostredia progamu R.
2. Skript predpokladá, že v aktuálnom adresári sa nachádzajú súbory **`candidates.train.features.9232.csv`** a **`unseen.features.3000.csv`**. Oba súbory načíta a do globálnych premenných uloží procedúra **`preprocessing()`**
3. Ak sú údaje naparsované v globálnych premenných **`train`** a **`test`**, je možné spustiť procedúru **`train()`**, ktorá vytvorí a natrénuje model môjho najlepšieho klasifikátora. Model uloží do globálnej premennej **`model`**.
4. Záverečná procedúra **`evaluation()`** vráti faktor ohodnotení testovacích dát klasifikátorom uloženým v globálnej premennej **`model`**.