Disambiguace významu slov
Závěrečná zpráva

Petra Barančíková

2012/13

1 Popis úlohy

Cílem projektu je automaticky přiřadit víceznačnému slovu jeho význam na základě věty, ve které bylo použito. Konkrétně se jednalo o tři anglická slova - podstatné jméno "line", sloveso "serve"a adjektivum "hard". K tomu máme k dispozici data - soubory s větami a konkrétními významy v nich.
Správné určení významu slova má velký význam v lingvistice a při zpracování přírozeného jazyka. Lze ho aplikovat při ve strojovém učení, například právě slovo "line"může být podle svého významu překládáno jako například "čára", "linka", "řada", "fronta", "řádek", "poznámka", "šňůra", "kabel", "spojení", "trasa", "obor", "rozdělení"a mnohá další. Význam má ale i v mnoha jiných odvětvích jako například při dobývání informací, odvozování z kontextu, diskurzní analýze, odpovídání na otázky a dalších...
Prvním krokem při automatické disambiguaci je extrahovat z těchto vět takové *atributy*, které mají vliv na význam daného slova. To může být třeba spojení s konkrétní předložkou, předmětem, pozice ve větě, nebo tagy okolních slov. Na základě těchto atributů je potom možné natrénovat klasifikátor, který by určoval co nejspolehlivěji správný význam slova.

2 Data

K dispozici jsou tři soubory s daty *hard.devel/serve.devel/line.devel*. Každý z nich obsahuje příklady pro jedno slovo. Pro každý z příkladů jsou uvedeny tyto informace:

- **ID** - identifikační číslo dané věty
- **SENSE** - význam slova v této větě, tj. třída, kterou chceme určit
- **SENTENCE** - upravená věta s vyznačeným cílovým slovem, úprava spočívá v převedení všech písmen na malé a vložení mezer mezi všechny prvky věty, tj. slova i diakritiku
- **MORPH** - automaticky získaná morfologická analýza věty
- **WLT** - automaticky získaná morfologická analýza věty v alternativním formátu slovo/lemma/morfologický tag
- **PARSE** - seznam syntaktických závislostí, také získaný automaticky pomocí Stanford dependency parseru

2.1 Rozdělení dat

Následující tabulka obsahuje počty příkladů pro každé slovo. Množství použitých vět neodpovídá celkovému počtu vět, protože některé věty v datech neobsahovaly požadované slovo a byly proto vynechány.

slovo	hard	line	serve
příkladů v datech	3833	3646	3878
použitých příkladů	3831	3591	3847

Tabulka 1: Počet příkladů

Data jsem nerozdělovala na pevně dané trénovací, vývojová a testovací data. Místo toho jsem raději používala *křížovou validaci*. Při ní byla data náhodně rozdělena na deset stejně velkých podmnožin. Na každé z těchto podmnožin jsem poté vyhodnocovala model natrénovaný na komplementu této množiny, tj. zbylých $\frac{9}{10}$ dat. Obdobně při ladění modelu, kdy ale bylo voleno rozdělení pouze na 8 podmnožin, protože ladění modelů bylo časově náročné.

2.2 Určované třídy

Rozdělení jednotlivých tříd v datech nebylo rovnoměrné, většinou jedna třída výrazně dominovala, viz. tabulky 2, 3 a 4.

význam	HARD1	HARD2	HARD3
počet příkladů	3056	447	328
podíl ze všech	79.8%	11.7%	8.6%

Tabulka 2: Rozdělení tříd u *hard*

HARD1 označuje náročné, složité, vyžadující značné úsilí, HARD2 znamená tvrdé metaforicky, těžké (např. ostrý pohled, tvrdá práce) a HARD3 naopak fyzicky tvrdé, pevné, špatně zlomitelné či ohnutelné.
S ohledem na běžnou mluvu se i zdá docela přirozené, že se HARD1 vyskytuje v datech výrazně častěji než ostatní třídy. I pro *serve* a *line* je to velmi podobné - dominující třída je ta v jazyce nejčastěji používaná.

význam	cord	division	formation	phone	product	text
počet příkladů	332	321	295	378	1913	352
podíl	9.2%	8.9%	8.2%	10.5%	53.3%	9.8%

Tabulka 3: Rozdělení tříd u *line*

SERVE10 představuje podávat někomu jídlo či pití, SERVE12 vykonávat určité zaměstnání, povinnost či službu. SERVE2 znamená hrát určitou roli, mít specifické

význam	SERVE10	SERVE12	SERVE2	SERVE6
počet příkladů	1586	1119	753	389
podíl ze všech	41.2%	29.1%	19.6%	10.1%

Tabulka 4: Rozdělení tříd u *serve*

využití a významem SERVE6 je poskytovat (oblasti či skupině lidí) službu či produkt.

2.3 Výběr atributů

Dalším krokem je výběr atributů. Ty je potřeba zvolit tak, aby byly relevantní a aby jejich kombinace dokázala odhadnout správnou třídu i pro slovo s dosud neznámým významem. Jedná se proto o velmi podstatnou část celého úkolu.

hard	line	serve
27	60	45

Tabulka 5: Počet atributů

Význam slova lze určit pouze na základě kontextu.[2] Zaměřila jsem se proto převážně na slova, které se v kontextu jednoho významu objevovaly výrazně častěji než u jiného. Konkrétně jsem si udělala frekvenční analýzu slov v bezprostředním okolí určovaného slova a vybrala z nich takové, které se pro jednu třídu vyskytovaly výrazně častěji než pro ostatní. Na jejím základě jsem si pro tuto třídu spočítala pravděpodobnost P_{data}(třída|množina slov).
Tu jsem potom posuzovala individuálně - pokud se mi zdála dostatečně vysoká, vytvořila jsem atribut, který testuje přítomnost slov z množiny na dané pozici. V opačném případě jsem ji nevyužila, nebo zkusila vyřadit z množiny ty slova, které hodnotu P_{data}(třída|množina slov) snižovaly. Obecně platí, že jsem pro méně časté třídy ponechávala atributy s nižší hodnotou, než pro tu nejfrekventovanější resp. i takové atributy, které dobře oddělovaly více méně častých tříd od té nejčastější. Minimální velikost množiny slov, aby mohla tvořit atribut, jsem stanovila na 5 výskytů v datech.
Kromě výskytu konkrétního slova v nejbližším okolí jsem stejným způsobem vybírala atributy mezi slovními tvary určovaného slova, morfologickými tagy v jeho okolí, slovy, se kterými se spojovalo v určité syntaktické závislosti, určitými syntaktickými funkcemi obecně, předložkami, se kterými se často spojovalo a dalšími.
Protože každé slovo je jiného slovního druhu, jsou i relevantní atributy pro ně velmi odlišné a vybírala jsem je proto zvlášť. Jejich celkový počet je v tabulce 5 a kompletní seznam je v souborech hard.pl, line.pl a serve.pl, které slouží také k jejich extrakci z dat. Všechny atributy jsou binární a vybrané z hladin SENTENCE, MORPH a PARSE. Zde je příklad konkrétního atributu pro slovo *hard* pro každou hladinu:

- **SENTENCE** - po *hard* následuje abstraktní podstatné jméno jako *look, feeling, evidence, truth, edge, work*; P_{data}(HARD2|*hard_*{*look, feeling,...*}) = 0.93
- **MORPH** - po *hard* následuje tag TO (tj. slovo *to*); P_{data}(HARD1|*hard_to*) = 0.99

- **PARSE** - *hard* se pojí s předložkou *for*; P_{data}(HARD1|prep_for(hard,*)) = 0.99

Po získání množiny atributů jsem ji ještě zkoušela redukovat. K tomu existuje několik důvodů. První je, že velké množství atributů zpomaluje algoritmy a vyžaduje mnoho místa. Tento v mém případě nebyl tolik důležitý, protože množství extrahovaných atributů nebylo nijak závratné. Podstatnější důvod je, že pokud jsou mezi nimi nevhodně zvolené rysy, mohou vytvářet šum a výsledky klasifikátorů jsou potom náchylnější k chybám. Ideální je proto menší optimální množina atributů.[3] Příliš mnoho atributů také může způsobit přetrénování modelu a ten potom není schopen správně předpovídat pro dosud neviděná data.
Z mnoha možných postupů jsem zvolila dvě metody pro redukci počtu atributů Borutu a zpětné vyřazování.

2.3.1 Boruta

Jedná se o algoritmus pro výběr důležitých atributů. Je založen na klasifikačním algoritmu náhodného lesa.[1] Důležitost atributu je počítáná jako pokles přesnosti klasifikace způsobený náhodnou permutací hodnot atributů mezi příklady. Ze všech stromů v lese, které používají daný atribut pro klasifikaci, se vypočítá průměr a standardní deviaci ze ztráty přesnosti. Z-skóre je potom průměr podělený standardní deviací.
Boruta konkrétně funguje tak, že pro každý atribut se vytvoří jeho kopii, tzv. stínový atribut, jehož se hodnoty náhodně promíchají. Na všech atributech se pustí algoritmus náhodného lesa a pro každý z nich spočítá Z-skóre. Nalezne se maximální hodnota Z-skóre mezi stínovými atributy a ta se porovná s každým dosud nerozhodnutým atributem. Ty, jejichž důležitost je významně nižší, jsou označeny jako nedůležité a odstraněny, zatímco ty které mají důležitost významně vyšší jsou označeny jako důležité. Poté se odstraní stínové atributy a celý proces se opakuje, dokud nemají všechny atributy přiřazenou důležitost.[1]

hard	line	serve
2	15	8

Tabulka 6: Počet atributů vyřazených Borutou

Tento algoritmus je součástí balíčku Boruta pro jazyk R. Konkrétně jsem ho spouštěla příkazem `Boruta(V1 ~ ., data = a, confidence = 0.95)` pro 95% jistotu toho, že je daný atribut lepší než náhodná permutace hodnot. Základní nastavení je 99.9%, ale to je velmi časově náročné. Atributy, o kterých se algoritmus nedokázal rozhodnout, jsem ponechala.
Konkrétní vyřazené atributy a naopak atributy, které Boruta označila jako nejdůležitější jsou součástí přílohy, stejně jako grafy důležitosti atributů pro jednotlivá slova.

[1] viz kapitola 3.1.

2.3.2 Zpětné vyřazování

Tento algoritmus není založen nutně na konkrétním klasifikátoru, zde byl konkrétně používán na k-NN[2] a SVM[3] při základním nastavení. Model je nejprve natrénován na všech atributech. Potom se postupně odebírá po každém atributu a spočítá se přesnost modelu. Atribut, jehož odstraněním se nejvýše zvýšila přesnost klasifikátoru, je odebrán a celý proces se opakuje do té chvíle, kdy se odstraněním atributu přesnost klasifikátoru už nezvýší. Pokusy se zpětným výběrem atributů jsem několikrát opakovala a nakonec odebrala ty atributy, které byly vyřazeny ve většině případů.
Alternativní možností je také obrácený algoritmus, kdy se postupně přidává po jednom atributu, který nejvíce zvyšuje přesnost modelu, dokud se přidáním nového atributu přesnost zvýší. Každý z těchto postupů má své výhody a nevýhody. Přidávání po jedné může opomíjet ty atributy, které fungují dobře ve společné kombinaci a naopak odebírání může být velmi výpočtově náročné, pokud stačí použít pár atributů z velké množiny. Zde jsem zvolila zpětný postup, protože množství extrahovaných atributů není příliš vysoké. V tabulce 7 je počet vyřazených atributů pomocí zpětného vyřazování pro SVM a k-NN. Pro náhodný les se mi zdála dostatečná Boruta. Konkrétní vyřazené atributy jsou součástí přílohy.

	hard	line	serve
k-NN	3	6	4
SVM	3	4	3

Tabulka 7: počet vyřazených atributů

Je zajímavé, že shoda mezi algoritmy nebyla většinou příliš vysoká. Například pro *serve* se shodly pouze Boruta a zpětné vyřazování pomocí k-NN a to na jediném atributu. Naopak u *line* tvořily atributy vyřazené zpětně pro k-NN podmnožinu vyřazených Borutou.
Tímto způsobem jsem pro každé slovo získala čtyři různé množiny atributů, které jsem dále používala k trénování modelů. Nicméně nejlepších výsledků bylo většinou dosaženo stejně na původní, neredukované množině atributů.

3 Teoretický popis modelů

3.1 Náhodný les

3.1.1 Rozhodovací stromy

Rozhodovací strom je zakořeněný strom, tj. graf bez kružnic. Každý z jeho vnitřních uzlů reprezentuje právě jeden atribut a hrany, které z něj vedou odpovídají hodnotám, které tento atribut nabývá. Při klasifikaci příkladu se tedy postupuje shora dolů podle hodnot atributů a na závěr je mu přiřazena hodnota v listu. Rozhodovací strom můžeme také chápat jako disjunkci konjunkcí, kde vnitřní uzly jsou částečně ohodnocené formule a listy úplně ohodnocené, mají hodnotou.
Strom se staví na základě rozdělování trénovacích dat na co nejjednoznačněji určené

[2] viz kapitola 3.2
[3] viz kapitola 3.3

množiny. V každém kroku se zvolí takový atribut, který jako rozhodující kritérium způsobí největší pokles nečistoty dat, tj. co podaří se oddělit třídy co nejjednoznačněji. Způsobů měření nečistoty dat je více, program R jako základní kriterium bere Gini Index = $\sum_j p_j$, kde p_j je pravděpodobnost třídy j.[8] Při stavbě stromu je třeba dát si pozor na přetrénování - aby strom dokonale neokopíroval rozdělení trénovacích dat, místo požadované generalizace. Kvalitu stromu je možné zlepšit prořezáváním. Při něm se větve, které pracují s příliš malou podmnožinou trénovacích dat nahradí listem s hodnotou nejčastější třídy v této podmnožině.

3.1.2 Náhodný les

Náhodný les je tvořen velkým množstvím rozhodovacích stromů. Stromy v náhodném lese jsou tvořeny speciálním způsobem. Pokud trénovací data D_{train} obsahují n příkladů, každý ze stromů roste na množině n náhodně vybraných příkladů z D_{train} s opakováním. Pravděpodobnost výskytu konkrétního příkladu v takto vybrané množině je $1 - (1 - \frac{1}{n})^n \approx 1 - e^{-1} \approx 0.63$.
Pokud je m počet atributů, je zvoleno číslo m_d ostře menší než m. Při stavbě stromu se potom v každém uzlu vybírá mezi m_d náhodně vybranými atributy. Hodnota m_d je stejná pro všechny stromy v lese. Stromy se neprořezávají, každý je pěstován do maximálního rozsahu.
Při klasifikaci se testovacího příkladu nejprve vyhodnotí na každém ze stromu v lese. Jako výsledná třída je pak zvolena ta, pro kterou hlasuje nejvíce stromů.
Tento algoritmus má řadu výhod, patří mezi nejpřesnější mezi současnými klasifikátory, dokáže efektivně zpracovávat velká data a nevadí mu ani velké množství atributů.[1]

3.2 kNN - k-Nearest Neighbour

Algoritmus k-Nearest Neighbour patří mezi tzv. *líné metody*, to znamená, že si nevytváří žádný model, ale klasifikuje na základě podobnosti s trénovacími daty, které si uchovávají v paměti. Konkrétně při metodě k-NN se trénovací příklady uloží podle hodnot svých atributů v m-dimenzionálním prostoru, kde m je počet atributů. Při klasifikaci nového příkladu algoritmus v tomto prostoru hledá k jeho nejbližších sousedů. Poté je mu přiřazena třída, kterou má většina z jeho k nejbližších sousedů.
Vzdáleností mezi sousedy je obvykle chápána euklidovská vzdálenost, tj. pro dva příklady x_i a x_j je vzdálenost definována $d(x_i, x_j) = \sqrt{\sum_{r=1}^{m}(x_i^{(r)} - x_j^{(r)})^2}$, kde $x_i^{(r)}$ je hodnota x_i v r-té dimenzi.
Formálně, označíme-li D_{train} jako trénovací množinu, můžeme klasifikaci nového příkladu x_n popsat následovně. Pokud $x_n \in D_{train}$, je mu přiřazena třída z trénovacích dat, v opačném případě algoritmus hledá $A \subseteq D$ takové, že $\mid A \mid = k$ a $\sum_{x_i,x_j \in A} d(x_i, x_j)$ je minimální. x_n je přiřazena nejčastější třída v A.

3.3 SVM - Support vector machines

Trénovací příklady je možné si představit umístěné v prostoru, zkonstruovaném nad množinou všech hodnot atributů. SVM hledá separátor - nadrovinu, která by prostor s příklady rozdělila na dva podprostory obsahující příklady jedné třídy, tak aby příklady měly co největší vzdálenost k separátoru. Vzdálenost je chápána jako délka k nejbližšímu bodu separátoru.
Protože dvě třídy nebývají často dokonale oddělitelné a data mohou obsahovat šum, byl tento model dále vylepšen zavedením speciálních proměnných ξ. Ty penalizují příklady

na nesprávné straně separátoru. Další výrazné zlepšení SVM získalo zavedením kernelových funkcí, které umožňují i jiné než lineární rozdělení prostoru.
Hledání separátoru lze formálně popsat jako hledání takového α, které maximalizuje následující výraz: $L(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i^T x_j)$, tak že $\alpha_i \geq 0$ a $\sum_i \alpha_i y_i = 0$.

K označuje kernelovou funkci, mezi nejběžnější patří:

- lineární $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$
- polynomiální $K(\mathbf{x}, \mathbf{z}) = (\gamma \mathbf{x}^T \mathbf{z} + c)^d$
- radiální $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma(\| \mathbf{x} - \mathbf{z} \|))^2$
- sigmoida $K(\mathbf{x}, \mathbf{z}) = \tanh(\gamma \mathbf{x}^T \mathbf{z} + c)$

Algoritmus SVM je primárně určen pro klasifikaci dvou tříd. Je možné ho ale rozšířit i pro klasifikaci více tříd. Nejčastější způsob je pomocí hlasovacích schémat jako je *jeden-proti-jednomu* a *jeden-proti-všem*. Máme-li m tříd, je při jednom-proti-všem nacvičeno m binárních klasifikátorů. Každý z nich je natrénován na rozpoznání jedné třídy proti nové třídě, kterou tvoří příklady všech ostatních tříd. V programu R se používá jeden-proti-jednomu. Při tom natrénováno $\binom{k}{2}$ klasifikátorů na datech právě ze dvou tříd. Neznámému příkladu je přidělena vždy ta třída, pro kterou hlasovala většina klasifikátorů.

4 Praktický popis modelů

Všechny experimenty byly prováděny v jazyku R [7]. V příloze jsou dokumenty hard.r, line.r a serve.r, ktere obsahují jak skripty využité k trénování, tak výsledná nastavení modelů.
Automatické vyhodnocení modelu je možné spustit skriptem run.sh, který potřebuje jako první argument určované slovo, jako druhý trénovací množinu a jako třetí testovací množinu.
Výsledky experimentů jsou hodnoceny pomocí *přesnosti* - tj. míry správně klasifikovaných příkladů vůči všem příkladům použitým při klasifikaci. Aby tato míra byla co nejspolehlivější, je vždy počítána pomocí křížové validace.
V tabulce 8 je uvedena baseline - přesnost naivního klasifikátoru, který klasifikuje všechny přiklady nejčastější třídou v trénovacích datech.

hard	line	serve
79,9	53.27	41.23

Tabulka 8: baseline

4.1 Náhodný les

Pro trénovaní modelu náhodného lesa jsem použila knihovnu randomForest. Konkrétní příkaz pro natrénování základního modelu je randomForest(V1 ∼ ., data = train, method = "class"), kde V1 je atribut třídy a data je množina dat s příslušnými atributy.

atributy	hard	line	serve
všechny	**91.19**	**75.93**	**84.83**
Boruta	91.12	75.55	83.69

Tabulka 9: Náhodný les - základní model

Vzhledem k tomu, že pro každé slovo se nejlépe osvědčily všechny atributy, trénovala jsem parametry modelu na nich. Jednalo se konkrétně o tyto parametry:

1. ntree - počet stromů, které v lese roste, základní nastavení je 500. Zkoušela jsem natrénovat les i s hodnotami 400 a 600, ale rozdíly nebyly příliš významné a proto jsem poté nechávala původních 500 stromů.
2. mtry - počet atributů náhodně vybraných pro každý uzel stromu. Základní nastavení je odmocnina z počtu atributů. Proto jsem jí nastavovala u každého slova individuálně v okolí základního nastavení ± 2,3.
3. nodesize - Minimální velikost pro list. Základní nastavení je 1, pokud se toto číslo zvýší, stromy jsou menší a klasifikace je rychlejší. Zkoušela jsem interval od 1 do 5.

Kvůli zrychlení jsem také omezila křížovou validaci na pouze 8 rozdělení. Konkrétní příkaz potom vypadal například: tune.randomForest(V1 ~ ., data = data, mtry = (4:10), nodesize = (1:5), tunecontrol = tune.control(sampling = "cross", cross = 8)).

slovo	mtry	nodesize	presnost
hard	7	1	91.27
line	8	2	75.89
serve	11	5	85.91

Tabulka 10: Náhodný les - modely na všech atributech s vyladěnými parametry

4.2 k-NN

Pro trénovaní modelu náhodného lesa jsem použila knihovnu class. Přesnost při základním nastavení (tj. pomocí příkazu knn(trénovací-data, testovací-atributy, testovací-třídy) je v tabulce 11.

Pro k-NN není mnoho parametrů, které lze upravovat. Jeden z nich je k = počet nejbližších sousedů, které algoritmus bere v úvahu při vyhodnocování. Při použitém základním nastavení platí $k = 1$, zkusila jsem proto počet sousedů postupně navyšovat. Pro každou množinu dat jsem vždy jedenáctkrát nacvičila model, kde $k = 1..30$. Grafy přesnosti na základě k jsou v obrázcích 1,2 a 3, které jsou součástí přílohy.

Výsledky jsou zajímavé. Pro *hard* se přesnost prakticky neměnila a stále kolísala okolo 90-91%. Nejlepšího výsledku dosahovala pro zpětně odebírané, kde dosáhla maxima u $k = 10$. S rostoucím k potom přesnost klesala, zatímco pro Borutu a všechny atributy se při vysokém k naopak přesnost zvyšovala a předstihla zpětně odebíráné. Naopak pro

atributy	hard	line	serve
všechny	90.02	73.40	83.49
Boruta	89.95	74.72	82.57
zpětně odebrané	**90.82**	**75.60**	**84.39**

Tabulka 11: základní k-NN

slovo	atributy	k	přesnost
hard	zpětně odebírané	10	90.97
line	zpětně odebírané	1	75.61
serve	zpětně odebírané	1	84.39

Tabulka 12: finální nejlepší k-NN modely

line se přesnost od 1 s rostoucím k prudce snižovala.
U *serve* šla přesnost také od 1 dolů, ale zdaleka ne tak radikálně jako u *line*. Navíc se u *serve* prohodily množiny atributů. Jako v jediném modelu zde vychází hůř množina atributů vybraných Borutou než množina všech atributů. Nejlepší výsledky k-NN pro jednotlivá slova jsou v tabulce 12.

4.3 SVM

Pro trénovaní SVM modelu jsem použila knihovnu e1071. Základní příkaz potom může vypadat takto: model = svm(V1 ∼., data = train, type = 'C'), kde V1 ∼. znamená, že model bude určovat třídu V1 na základě všech ostatních v dále zadaných datech a type = 'C' určuje, že se bude jednat o klasifikaci. Dále je možné měnit řadu parametrů v nastavení funkce svm. Jde především o kernelovou funkci, kde základní nastavení je radiální, ale je možné ji nastavit na polynomiální, lineární, nebo sigmoidu změnou parametru kernel.

4.3.1 Hard

Pro orientaci jsem si nejprve spočítala průměrnou přesnost různých kernelových funkcí na všech množinách parametrů. Výsledky jsou v tabulce 13.
Nejlepších výsledků dosahuje pro množinu zpětně vybíraných atributů a pro radiální a polynomiální kernelovou funkci. Tyto dvě jsem proto zvolila k dalšímu ladění atributů. To jsem dělala pomocí funkce tune.svm. U SVM jsem ladila konkrétně tyto parametry:

- degree - určuje se pro polynomiální kernelovou funkci - stupeň polynomu. V formulích je značen d.
- gamma - určuje se pro všechny kernelové funkce mimo lineární. Základní nastavení je 1/počet příkladů. V formulích je značen γ.
- cost - postih za příklad na nesprávné straně hranice.

Konkrétně potom ladící příkaz vypadal a = tune.svm(V1 ∼ ., data = data, kernel = "polynomial", degree = (1:5), gamma = $2^{-8..1}$, cost = $2^{-3..4}$, tunecontrol = tune.control(sampling = "cross", cross = 8)). Laděním ale nedošlo k žádnému výraznému zlepšení (viz tabulka 14).

kernel	všechny atributy	Boruta	zpětně vybrané
radiální	91.20	91.14	**91.24**
polynomiální	90.97	91.12	**91.25**
sigmoida	88.54	88.36	**89.16**
lineární	90.84	**90.99**	90.86

Tabulka 13: hard - průměrná přesnost základního modelu SVM podle kernelových funkcí

kernel	degree	gamma	cost	přesnost
radiální	-	0.03125	1	**91.28**
polynomiální	2	0.125	0.25	91.25

Tabulka 14: *hard* - průměrná přesnost vyladěného SVM modelu

4.3.2 Line

atributy	všechny atributy	Boruta	zpětně vybrané
radiální	**75.88**	75.29	75.82
polynomiální	**75.93**	75.57	75.85
sigmoida	**75.85**	75.34	75.82
lineární	**75.96**	75.46	75.85

Tabulka 15: *line* - průměrná přesnost základního modelu SVM podle kernelových funkcí

Pro *line* se na základních modelech nejvíce osvědčila původní množina atributů a tak jsem dále pokračovala s ní. Vzhledem k tomu, že rozdíly mezi výsledky různých kernelové funkcí byly minimální, zkusila jsem vyladit všechny. Ladění *line* a *serve* probíhalo stejně jako u *hard*.
U *line* ladění pomohlo u radiální kernelové funkce, pro ostatní se přesnost příliš nezměnila, respektive v některých případech dokonce poklesla. (tabulka 16)

4.3.3 Serve

Pro *serve* jsou si výsledky velmi podobné (viz tabulka 17) stejně jako to bylo u *line*, tedy jsem zvolila i stejný postup - vyladit všechny kernelové funkce.
Občas jsem zkoušela vyladit určitý model několikrát. Je zajímavé, že pro lineární kernelovou funkci mi vyšly velmi rozdílné hodnoty a obě byly hodnocené hůře než při základním nastavení. Naopak polynomiální a radiální kernelové funkce si polepšily (viz tabulka 18).

5 Výsledek

Získala jsem tedy řadu modelů s do značné míry podobnou přesností. Pro každé slovo jsem vybrala dva, které dávaly ty nejlepší výsledky. Pro ně jsem zkusila dále zkoumat, jakou by měli úspěšnost na odlišných datech a jestli se jejich úspěšnost na různých datech bude lišit výrazněji, tj. zda-li jsou rozdíly mezi modely statisticky významné.
To jsem řešila pomocí *bootstrappingu*. Tato metoda dovoluje získat velké množství různých

kernel	degree	gamma	cost	přesnost
radial	-	0.0078125	4	**76.23**
polynomial	2	0.0625	0.125	76.03
sigmoid	-	0.00390625	8	75.24
linear	-	-	0.5	75.60

Tabulka 16: *line* - průměrná přesnost vyladěného SVM modelu

atributy	všechny	baruta	zpětně odebírané
radiální	**84.69**	83.61	84.48
polynomiální	**84.69**	83.83	84.61
sigmoida	**84.90**	83.88	84.74
lineární	**85.05**	83.78	84.48

Tabulka 17: *serve* - průměrná přesnost podle kernelových funkcí

množin dat tím, že se opakovaně z původních dat vybírá náhodný vzorek stejné velikosti, ve kterém se ale příklady mohou opakovat. Na každé takto vybrané množině je natrénován model a ten se ohodnotí na množině příkladů, které nebyly vybrány. V tomto případě jsem trénovala každý model na 1000 takto získaných množinách.
Poté jsem tyto data využila k tomu, abych získala průměrnou přesnost, konfidenční interval pro průměr na hladině 95% a 95% percentil. Konfidenční interval pro průměr na hladině 95% je spočítán jako $\bar{Y} \mp t(0.975, n-1) * \frac{s}{\sqrt{n}}$, kde $\bar{Y}$ je průměrná hodnota, s je standardní odchylka, n je velikost dat a $t(0.975, n-1)$ je 97.5 percentil t distribuce s n-1 stupni volnosti.[6]
95% percentil jsem získala tak, že jsem z celého vzorku vzala limitní body bez prvních 2,5% a posledních 2,5%, tedy konkrétně bez prvních a posledních 25 hodnot uspořádané množiny. Na tom je dobře vidět rozptyl hodnot jednotlivých modelů. Výsledky experimentů s bootstrappingem jsou uvedeny v tabulce 19.
Posledním způsobem, jak jsem využila dat získaných bootstrappingem, bylo porovnání úspěšnosti dvou nejlepších modelů pro dané slovo. To jsem dělala pomocí Welchova t-testu jako konfidenční interval hodnot na vektorech výsledků obou modelů na stejných datech získaných bootstrappingem. Pokud výsledný interval obsahuje 0, znamená to, rozdíl mezi úspěšností modelů není statisticky významný. V opačném případě je rozdíl statisticky významný a můžeme s jistotou 95% tvrdit, že je jeden model lepší. Výsledky tohoto testu jsou v tabulce 20.

kernel	degree	gamma	cost	přesnost
radiální	-	0.0078125	4	**85.45**
polynomiální	3	0.0625	0.25	85.16
sigmoida	-	0.0078125	4	83.95
lineární	-	-	32	84.93
lineární	-	-	0.5	84.76

Tabulka 18: *serve* - průměrná přesnost vyladěného SVM modelu

slovo	model	křížová val.	průměr boots.	95% k.i. průměr	95% percentil
hard	radiální SVM	91.28	91.09	(91.05,91.13)	(89.84,92.29)
	náhodný les	91.27	91.07	(91.03,91.11)	(89.99,92.25)
line	polynom. SVM	76.03	74.93	(74.70,75.16)	(54.82,77.45)
	radiální SVM	76.23	75.70	(75.61,75.78)	(71.28,77.84)
serve	radiální SVM	85.45	84.86	(84.81,84.91)	(83.18,86.31)
	náhodný les	85.91	85.27	(85.22,85.32)	(83.72,86.80)

Tabulka 19: dva nejlepší modely pro každé slovo, výsledky získané křížovou validací a bootstrappingem. Finálně zvolené modely pro každé slovo jsou označeny tučně.

slovo	metody	95% konfidenční interval
hard	radiální SVM x náhodný les	(-0.00073,0.00034)
line	polynom. SVM x radiální SVM	(-0.010,-0.0053)
serve	radiální SVM x náhodný les	(-0.0048,-0.0033)

Tabulka 20: konfidenční interval rozdílů výsledků různých modelů pro stejné slovo

Jedině pro *hard* nelze prohlásit o jednom modelu, že by byl lepší. I co se týče průměrné přesnosti jsou si až neuvěřitelně podobní. Přesto jsem vybrala jako nejlepší model náhodné stromy kvůli lehce vyšší dolní hranici percentilu. Pro klasifikaci *line* vychází lépe SVM s radiální kernelovou funkcí, a to nejen kvůli výsledku testu, ale především kvůli výrazně nižší dolní hranici percentilu u SVM s polynomiálním kernelem (viz tabulka 19). Pro *serve* se ukázaly jako nejvhodnější model náhodné stromy.
Pokusy ukázaly velmi podobné výsledky pro všechny klasifikátory. Nejlepších hodnot dosahovaly náhodné stromy a SVM, naopak nejhorší výsledky vycházely pro algoritmus k-NN. To lze připisovat především nerovnoměrnému zastoupení tříd v datech, kdy jedna třída nabízela výrazně výše sousedů. Tato skutečnost má na k-NN větší vliv než například na náhodné stromy. Pro ilustraci uvádím tabulku 21, která ukazuje zaokrouhlené průměrné ohodnocování k-NN algoritmu (s $k = 1$) pro slovo *line* v rámci křížové validace.

	cord	division	formation	phone	product	text
cord	12	0	0	3	16	0
division	0	23	0	0	8	0
formation	0	0	16	0	11	0
phone	1	0	0	24	11	0
product	2	0	1	1	185	1
text	1	0	0	0	23	8

Tabulka 21: matice chyb pro klasifikaci *line* pomocí k-NN, průměr z deseti pokusů

Tato tabulka byla velmi podobná i pro všechny ostatní modely. Nejčastější špatná klasifikace byla přiřazení nejčastější třídy - *product*, ostatní třídy se mezi sebou prakticky "nepraly". Pokud ale data odpovídají skutečnému výskytu daného slova v běžném jazyce, nemusí být nutně na škodu přiřadit špatně zařaditelný výskyt slova té nejčastější třídě.

Také by bylo vhodné udělat ještě druhotnou analýzu atributů, ačkoli jsem se snažila hledat takové atributy, které oddělovaly málo časté třídy od té nejčastější.

6 Závěr, shrnutí

Cílem této práce bylo automatické určování významu slov. Za tím účelem byly z dat vybrány atributy a nich nacvičené modely pro tři různé klasifikátory - náhodné stromy, k-NN a SVM. Ukázalo se, že modely dosahovaly na datech velmi podobné přesnosti, ale i tak byly výrazně lepší než baseline. Nejvíce se osvědčily náhodné stromy a SVM. Konečné výsledky jsou uvedené v tabulce 22 včetně přesnosti, které modely dosáhly při testování na 500 předem neviděných větách.

	hard	line	serve
baseline	79,8	53.3	41.2
nejlepší model	náhodný les	radiální SVM	náhodný les
přesnost při křížové validaci	91.1	75.7	85.3
přesnost na neznámých datech	90.0	74.5	85.0

Tabulka 22: baseline vs. nejlepší model

Literatura

1 Breiman Leo, Cutler Adele, *Random Forests*.
URL: http://www.stat.berkeley.edu/∼breiman/RandomForests-cc-home.htm

2 Hanks Patrick, Oxford English Dictionaries, *Do word meanings exist?*, Computers and the Humanities (2000), s. 171-177.

3 Kohavi Ron, John George H., *Wrappers for feature subset selection* (1996).
URL: ai.stanford.edu/ ronnyk/wrappersPrint.pdf

4 Kursa Miron B., Rudnicki Witold R., *Feature Selection with the Boruta Package*.
URL: http://www.jstatsoft.org/v36/i11/paper

5 NIST/SEMATECH *e-Handbook of Statistical Methods*, (2013).
URL: http://www.itl.nist.gov/div898/handbook/

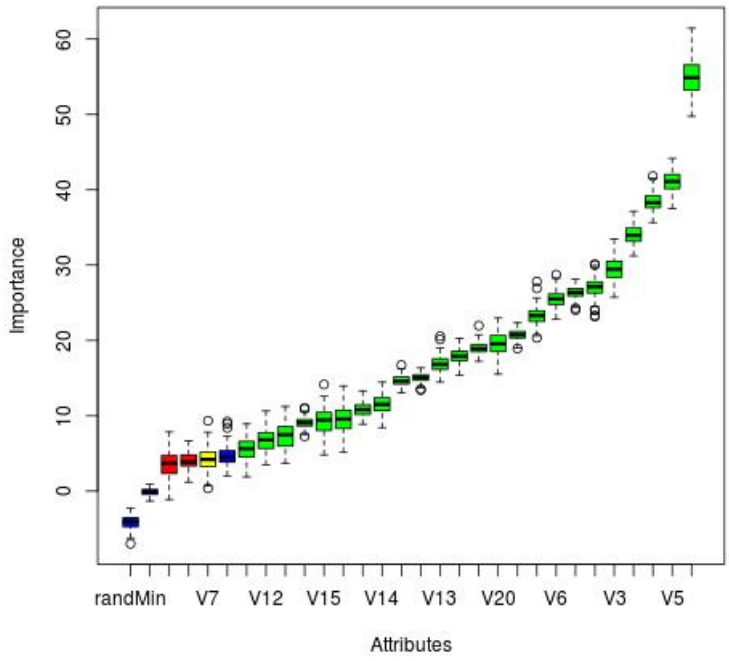
6 Snedecor, George W. and Cochran, William G. (1989), *Statistical Methods*, Eighth Edition, Iowa State University Press

7 The R Foundation. *The R project for statistical computing*.
URL: http://www.r-project.org/index.html.

8 Vidová-Hladká, B.*Decision Trees*.
URL: http://ufal.mff.cuni.cz/∼hladka/ML/LECTURES/PFL054-decision-trees.pdf

9 Vidová-Hladká, B. *Support Vector Machines*.
URL: http://ufal.mff.cuni.cz/∼hladka/ML/LECTURES/PFL054-SVM.pdf

10 Wikipedia. *Bootstrapping*, (2013).
URL: http://en.wikipedia.org/wiki/Bootstrapping_statistics



Obrázek 1: *hard* - důležitost atributů pro slovo *hard*

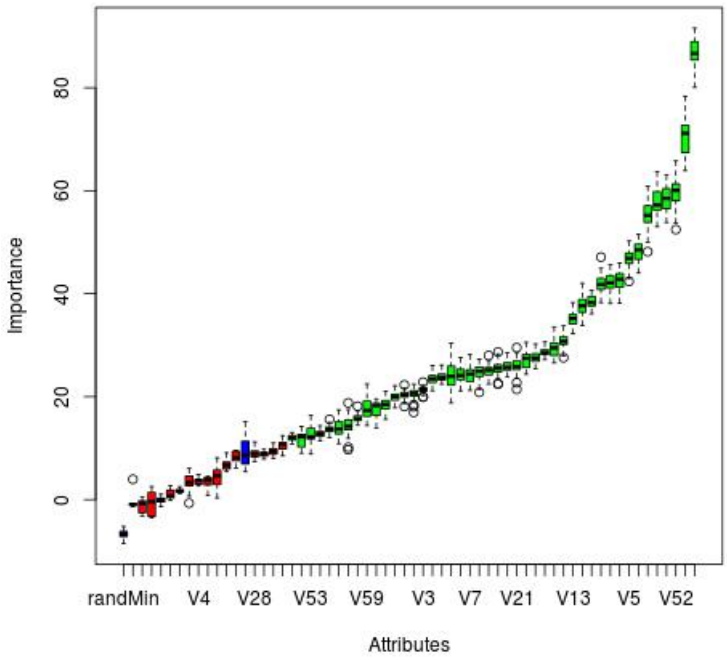
Příloha

Boruta

- **hard** - pro hard byly vyřazeny 2 atributy. Jednalo se o:
  V18 - tag před *hard* je JJ
  V26 - *hard* funguje jako otvřený větný doplněk - *xcomp*
  Naopak jako výrazně nejdůležitější atribut se ukázalo:
  V23 - *hard* je adjektivní modifikátor slova z množiny abstraktních podstatných jmen jako *time, question, decision...*

- **line** - bylo vyřazeno 15 atributů V4, V10, V24, V25, V28, V32, V34, V35, V36, V37, V44, V48, V54, V57, V58.
  Naopak jako výrazně nejdůležitější atributy se ukázaly postupně:
  V14 - po *line* následuje slovo z množiny slov spojených s telefonováním *telephone,direct,customer,..*
  V56 - *line* se pojí s předložkou *between*
  V52 - *line* je navázáno pomocí předložky *on*

- **serve** - pro slovo *serve* Boruta vyřadila osm atributů - V15 V23 V25 V26 V28 V29 V32 V41. Jako ty nejdůležitější pro určení správnou klasifikaci označila:
  V5 - věta obsahuje slovo *purpose*
  V34 -slovo *serve* je použito s některým pomocného slovesem
  V3 - věta obsahuje jedno z množiny slov označujících vedoucí funkce např. *chair-*



Obrázek 2: *hard* - důležitost atributů pro slovo *line*

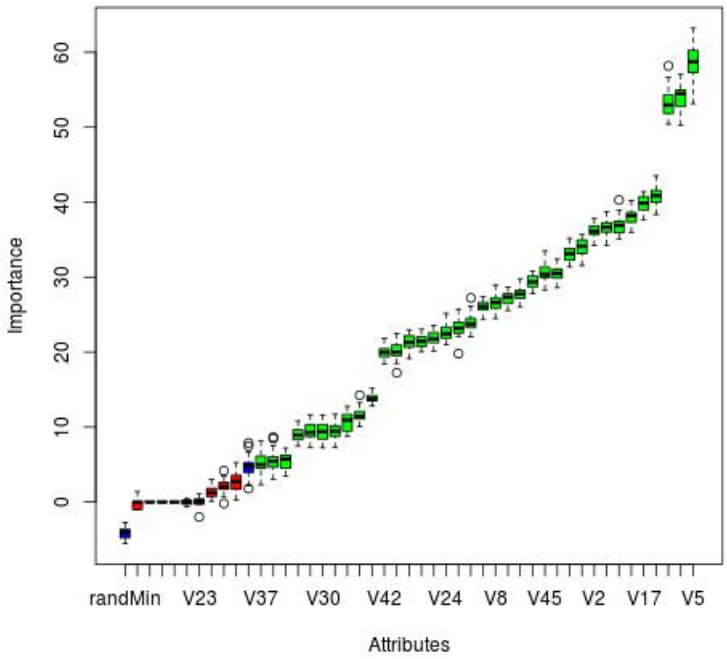
man, president, chief..

Zpětné vyřazování

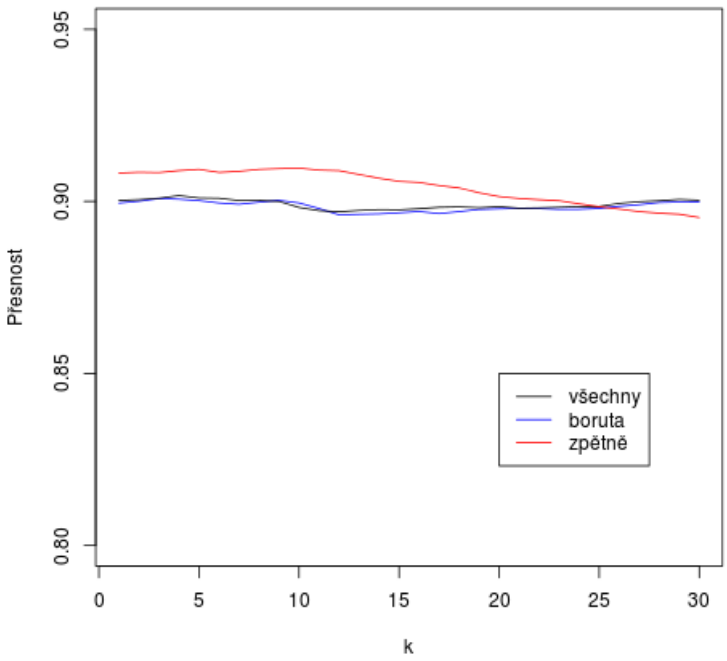
vyřazené atributy	hard	line	serve
SVM	V2, V7, V18	V10, V34, V35, V48	V6, V14, V24;
k-NN	V7, V12, V14	V8, V23, V24, V35, V51, V58	V7, V15, V19, V20

Tabulka 23: Zpětné vyřazování

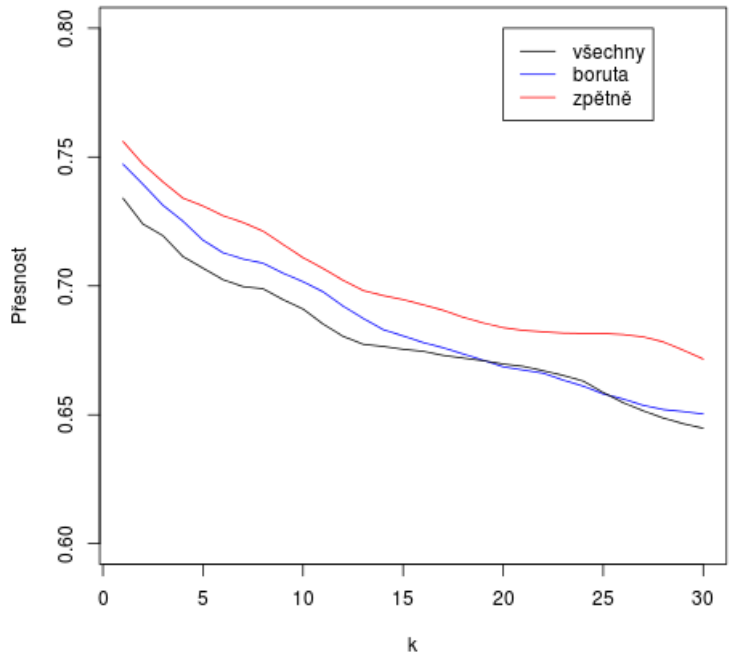
k-NN



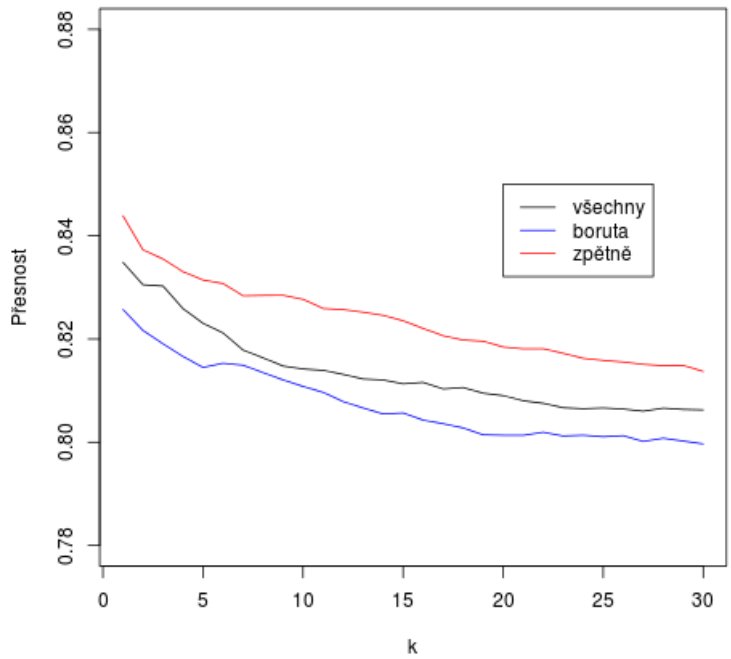
Obrázek 3: *hard* - důležitost atributů pro slovo *serve*



Obrázek 4: *hard* - přesnost k-NN na základě k



Obrázek 5: *line* - přesnost k-NN na základě k



Obrázek 6: *serve* - přesnost k-NN na základě k