Taggery

Barbora Obluková

Tagger

- cíl: odstranění nejednoznačnosti přidělení značky určující druh slova (POS tag), v případě více variant vybrána jen jedna
- jednoduché vs. atribute vectors značky
- rozhodování podle předešlého kontextu
- hladový algoritmus – v případě hledání Max hodnoty jen aktuální hodnota
- jazyková závislost

TreeTagger

- Helmut Schmid, Stuttgart 1994
- původní vývoj a testování na angličtině, dále na němčině
- desambiguace normálních slov od vlastních jmen
- třídy ekvivalence – slova se stejnou množinou možných značek
- tag brán atomicky, není dělen na části
- Vitterbiho algoritmus a rozhodovací strom pro určení pravděpodobnosti
- Vitterbiho algoritmus:
  - najde nejpravděpodobnější sekvenci stavů, která vygenerovala pozorovaná data

TreeTagger

- rozhodovací strom:
  - binární
  - každý token jím projde
  - struktura a prořezávání podle informačního zisku, snaha co nejvíce rozlišit možnosti (70 % vs. 10 %)
  - případné rozlišení podle dodatečných podmínek

TreeTagger

„The big house“
– 70% pravděpodobost, že 3. slovo (house) je NN, 10%, že je to ADJ

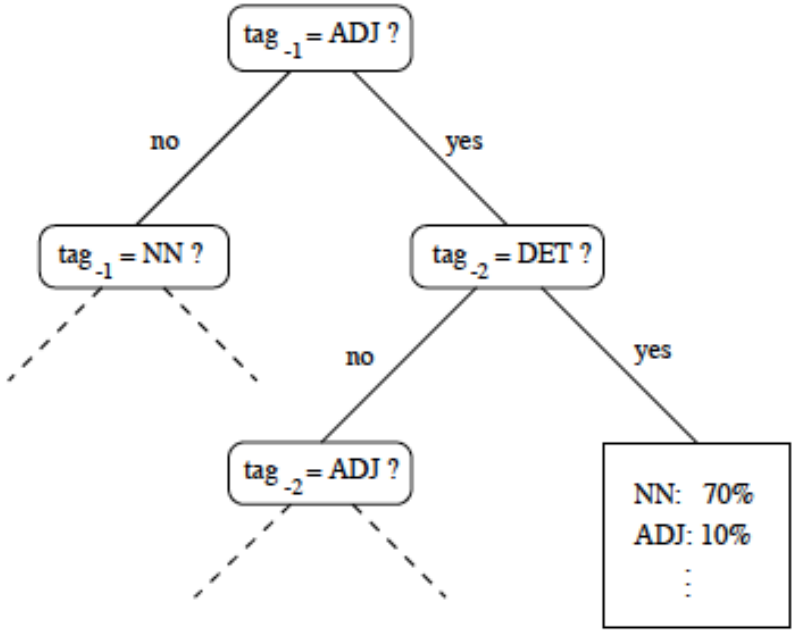


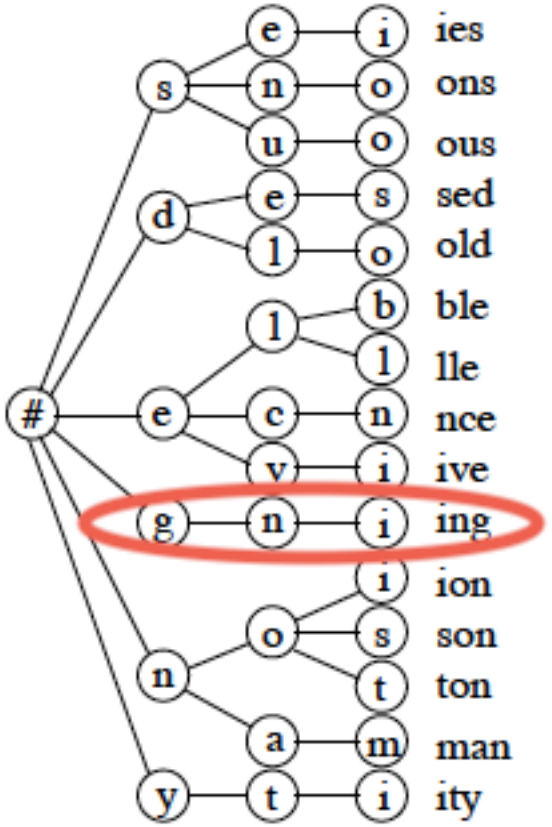
Figure 1: A sample decision tree

TreeTagger

- lexikon – dohledávání tagů pro dané slovo v již označkovaných datech (např. *Studying*)
    1. full form lexicon – hledání konkrétního slova (*Studying*), pokud nenalezeno, převedou se všechna písmena na malá a znovu se vyhledá. Pokud stále nic nenalezeno, krok 2
    2. suffix lexicon – vytvořen z trénovacího korpusu, organizován jako strom, ve kterém se hledá posloupnost 3 posledních písmen (nejprve *g*, poté *n* a *i*). Řetězec je ukončen znakem #. Pokud neúspěšně, krok 3
    3. default entry – vrácen výchozí vstup, který je vytvořen z relativních frekvencí v sufixovém stromě

TreeTagger

suffix lexicon – příklad Studying



TreeTagger

- nejlépe určí tag s využitím Vitterbiho algoritmu
- malá citlivost na velikost korpusu – menší trénovací data nezpůsobí menší přesnost
- nejúspěšnější verze – kvatrogramová
- nahrazování 0: vyměnění hodnoty 10^{-10}, což je užito namísto 0, za hodnotu 0,1 zlepšilo přesnost pouze o 0,02 % => tento parametr není stěžejní

method	context	accuracy
trigram tagger	trigram	96.06 %
TreeTagger	bigram	95.78 %
TreeTagger (0.1)	trigram	96.34 %
TreeTagger	quatrogram	96.36 %
TreeTagger (10^{-10})	trigram	96.32 %

RFTagger

- Helmut Schmid, Florian Laws, Stuttgart 2008
- označkování věty *Das zu versteuernde Einkommen sinkt.* („The to be taxed income decreases.“ *The taxable income decreases.*)

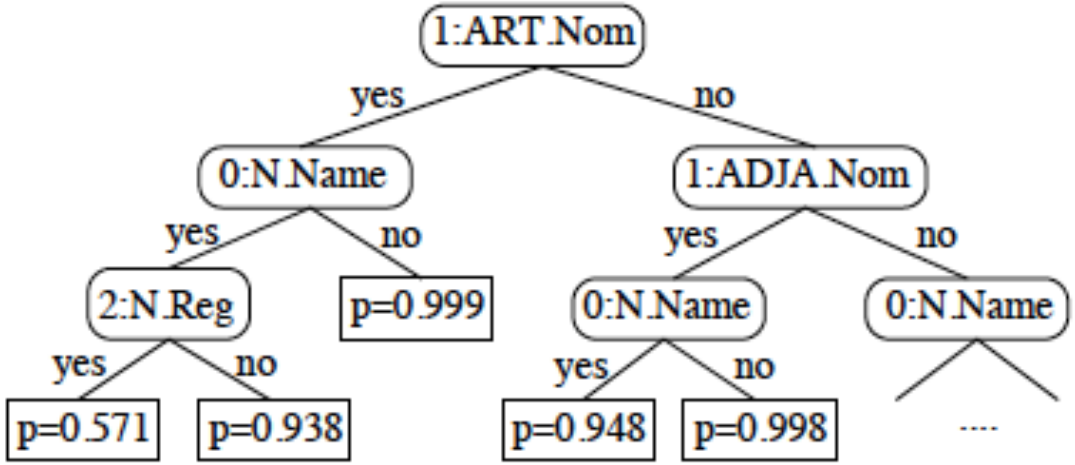
Das	ART.Def.Nom.Sg.Neut
zu	PART.Zu
versteuernde	ADJA.Pos.Nom.Sg.Neut
Einkommen	N.Reg.Nom.Sg.Neut
sinkt	VFIN.Full.3.Sg.Pres.Ind
.	SYM.Pun.Sent

RFTagger

- postup:
  1. trénování – vložen velký označkovaný korpus, atributy odděleny tečkou
  2. anotace textu – vložen čistý text a trénovací model, výstupem je anotovaný text, ve kterém každé slovo má přesně jeden tag
- rozumí struktuře tagu
- tagy formou vektorů atributů
- pevný počet atributů oddělených tečkou – první atribut představuje kategorii, druhý upřesňuje
- nejprve určení slovního druhu, poté sady morfologických atributů
- atributy morfologicky vázané – odlišuje se např. pád u podstatných a přídavných jmen

RFTagger

- rozhodovací stromy
  - odlišnost od TreeTagger
  - dekompozice na jednotlivé atributy
  - pro každou hodnotu atributu separátní rozhodovací strom (o každém atributu je rozhodováno samostatně)
  - získání pravděpodobnosti pro každý atribut



RFTagger

- užití Skrytých Markovových modelů – modul je pomocí korpusu naučen, která slova může dát dohromady na základě kontextu
- podporuje němčinu, češtinu, slovenštinu, slovinštinu, maďarštinu a ruštinu
- nedělá lemmatizaci, lemma lze získat z korpusu
- Lexikon, Wordclass automaton, Maximální kontextová velikost

Maximální kontextová velikost

- užita na vypočítání pravděpodobnosti
- jak daleko do kontextu se tagger může dívat
- kontext 1 = pouze vedlejší slovo

TreeTagger	RFTagger
Baseline – 70,54 %	Kontext 1 – 90,89 %
Kontext 1 – 86,22 %	Kontext 2 – 92,06 %
Kontext 2 – 87,31 %	Kontext 10 – 92,43 %
Kontext 5 – 87,47 %	
Kontext 10 – neuspělo	

Publikované výsledky

TreeTagger	RFTagger
Penn Treebank (2 mil.) • 96,81 %	German Tiger Treebank (886 k) • Baseline –69,4 % • 92,2 %
Stuttgart Zeitung (25 k) • 97,53 %	Czech Academic corpus (652 k) • 89,53 %

Ostatní taggery

- Desam – kombinovaný tagger, který využívá Vitterbiho algoritmus
- Czech Tagger
- Morce
- Morphodita

Tvorba POS taggeru v Pythonu

- POS tagging – strojové učení s učitelem
- Case-sensitive features
  - tagger si všímá rozdílu velkých a malých písmen
  - výhodné u gramaticky správného textu
  - robustní tagger
    - case-sensitive features nevhodné
    - vhodnější funkce, která porovná frekvenci slova s malým a s velkým písmenem v datech z webu
- jednoznačná slova
  - netřeba kontrolovat tag, stačí jen u víceznačných
- neužívat zbytečně složité taggery – např. Pattern, NLTK

Průměrovaný percepton

- sčítá váhové koeficienty rysů pro každou pozici v daném kontextu
- získání váhy slova:

1. váhové koeficienty všech rysů nastaveny na 0
2. nový pár – vstupní parametr (slovo) a POS tag
3. součet váhových koeficientů rysů v daném kontextu = váha vstupu
4. odhad hodnoty POS tagu pomocí aktuální váhy vstupu
5. pokud špatný odhad
   - přidání +1 k vahám spojeným se správnou třídou pro tento vstup
   - odečtení –1 od vah spojených se špatně zvolenou třídou
   - penalizována váha, která vedla ke špatnému závěru

Získání rysů

```
def _get_features(self, i, word, context, prev, prev2):
    '''Map tokens-in-contexts into a feature representation, implemented a
    set. If the features change, a new model must be trained.'''
    def add(name, *args):
        features.add('+'.join((name,) + tuple(args)))

    features = set()
    add('bias') # This acts sort of like a prior
    add('i suffix', word[-3:])
    add('i pref1', word[0])
    add('i-1 tag', prev)
    add('i-2 tag', prev2)
    add('i tag+i-2 tag', prev, prev2)
    add('i word', context[i])
    add('i-1 tag+i word', prev, context[i])
    add('i-1 word', context[i-1])
    add('i-1 suffix', context[i-1][-3:])
    add('i-2 word', context[i-2])
```

Váha vstupu

```
def train(self, nr_iter, examples):
    for i in range(nr_iter):
        for features, true_tag in examples:
            guess = self.predict(features)
            if guess != true_tag:
                for f in features:
                    self.weights[f][true_tag] += 1
                    self.weights[f][guess] -= 1
        random.shuffle(examples)
```

Průměrovaný perceptron

- v několika iteracích zpracována celá vstupní data
- výsledek předán Vitterbiho algoritmu jako pravděpodobnost přechodu
- nedostatek algoritmu perceptronu:
  - nutný průměr vah po několika iteracích – např. 5 000 příkladů, 10 iterací => průměr vah 50 000 hodnot
  - nezměněná hodnota – uložena do slovníku
  - změněná hodnota – změní se i v následujících iteracích
  - bez průměrování:
    - algoritmus se bude zabývat tím, co měl špatně a přizpůsobí tomu celý model
    - trénování na 2 trochu odlišných datech => úplně odlišné modely

Zdroje

- Helmut Schmidand Florian Laws: Estimation of Conditional Probabilities with Decision Trees and an Application to Fine-Grained POS Tagging, COLING 2008, Manchester, Great Britain.
- Schmid, Helmut. 1994. Probabilistic part-of-speechtagging using decision trees. In Proceedings of theInternational Conference on New Methods in LanguageProcessing, pages 44–49, Manchester, UK.
- Schmid, Helmut. 1995. ImprovementsIn Part-of-SpeechTaggingWithanApplicationTo German. In Proceedings ofEACL SIGDAT workshop, Dublin,Ireland.
- Spoustová, Drahomíra. 2009. Kombinované statisticko-pravidlové metody značkování češtiny. ÚFAL. Praha, Czech Republic.
- Honnibal, Matthew. 2013. A Good Part-of-Speech Tagger in about 200 Lines of Python. https://explosion.ai/blog/part-of-speech-pos-tagger-in-python