Relazione TLN

Carlo Alberto Barbano
Mat. 811588

1 - Traduttore diretto EN-IT

1.1 - TreeBank

Per la creazione dei modelli è stato utilizzato il treebank Universal Dependency nella versione inglese. Per la gestione del dataset è stata creata la classe TreeBank (treebank.py) che si occupa di parsificare i treebank e di offrire un'interfaccia agevole per il loro utilizzo.

La classe TreeBank precalcola inoltre le probabilità di emissione delle parole per ogni tag e il numero totale di occorrenze dei tag, memorizzando le informazioni in un dizionario (tabella di hash) in modo da ottenerle in un tempo O(1).

Es.

```
#probabilità di emissione della parola "make" per il tag VERB
treebank.tags['VERB']['emission']['make']
#numero di occorrenze del tag VERB nel treebank
treebank.tags['VERB']['count']
```

Per accedere ai dati del treebank, la classe TreeBank può essere utilizzata come un iteratore:

```
for sentence, tags in treebank:
    # sentence is a list of tokens
    # tags are the pos tag for each token in the sentence
    …
```

1.1 - PoS Tagging

Per la prima parte del progetto sono stati sviluppati due modelli di PoS Tagger: un modello di baseline (BaselinePoSTagger) e un modello HMM (MarkovPoSTagger).

Per addestrare un modello, è sufficiente effettuare una chiamata al metodo *train*, passando come argomento un treebank. La procedura di training si occupa di calcolare le probabilità necessarie per la fase di decoding.

Baseline Tagger

Il modello baseline assegna il tag più frequente per una certa parola all'interno del treebank, e il tag NOUN nel caso di parole sconosciute. L'accuracy è di 89% sul test set e 94% sul training set.

Durante il training, il modello baseline calcola la frequenza con cui ogni tag è associato a una certa parola, e in fase di decoding selezionerà per ogni parola il tag con frequenza maggiore.

HMM Tagger

Il secondo modello individua la sequenza di tag più probabile per una certa sequenza di parole, usando l'algoritmo di Viterbi. L'accuracy ottenuta è di 91% sul test set e 96% sul training set.

Nel training, il modello HMM calcola le probabilità di transizione da un tag *t* a un tag *t+1*, e in fase di decoding usa questo risultato, insieme alla probabilità di emissione dei tag (calcolata dal treebank) per produrre la sequenza di tag usando l'algoritmo di Viterbi:

$$v_t(j) = max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t)$$

dove:

- $v_t(i)$ è il valore di viterbi calcolato al passo *t* per lo stato *i*
- a_{ij} è la probabilità di transizione dallo stato *i* allo stato *j*
- $b_j(o_t)$ è la probabilità di emissione per il token o_t nello stato b_j

Nel caso in cui una transizione da uno stato *i* a uno stato *j* non fosse stata osservata nel treebank di training, viene effettuata un'ipotesi di smoothing iniziale

$$\alpha = P(t_j|t_i) = 1 / \#tags$$

Per quanto riguarda la probabilità di emissione $b_j(o_t)$, nel caso in cui la parola o_t fosse sconosciuta vengono effettuate le seguenti ipotesi iniziali di smoothing:

$$P(o_t|b_j) = \begin{cases} \beta = 1. & b_j = PNOUN \ and \ isCapitalized(o_t) \\ \gamma = 1./\#tags & otherwise \end{cases}$$

Per l'ottimizzazione degli iperparametri $\alpha\ \beta\ \gamma$ viene effettuata una grid search in un range $[1.0;\ 1e-10]$ usando il development set, con una chiamata al metodo *MarkovPoSTagger.tune_hyperparameters(dev_set)*. I valori ottimali degli iperparametri trovati in questo modo sono:

$$\alpha = 0.01;\ \beta = 1e-6;\ \gamma = 1e-7;$$

1.2 - Traduttore

2. Traduttore ITA-YODA