Università degli Studi di Torino

Dipartimento di Informatica

Relazione progetto parte I: Sistemi Cognitivi

Gianpiero Sportelli mat. 795469

Anno Accademico 2015/2016

Introduzione

L'esercitazione d'esame consiste nel costruire un **PoS Tagger** Stocastico basato su *HMM* che utilizzi per la decodifica **l'algoritmo di Viterbi**. L'esercitazione è stata svolta il linguaggio Java per la sua familiarità e per utilizzi futuri in altre esercitazioni con stesso linguaggio.

Sono state realizzate un insieme di classi che coprono completamente la pipeline di disambiguazione del PoS tag di tutte le parole di una frase. Durante la realizzazione sono state fatte numerose scelte di progetto che complessivamente hanno portano a risultati interessanti (95% sul test set considerato). Il confronto tra le varie implementazioni è stato fatto riferendosi a un algoritmo Baseline per il problema del PoS tagging che ha ottenuto performance decisamente inferiori (72%).

Nel tentativo di potenziare l'algoritmo base di Viterbi si è sviluppata una versione del algoritmo estesa. La versione estesa utilizza un modello HMM basato su i trigammi e una tecnica di smoothing basata su *deleted interpolation*. Per estendere le conoscenze morfologiche si è utilizzato morph-it. L'utilizzo di morph-it ha reso possibile attenuare il problema delle parole sconosciute e potenziare leggermente l'algoritmo Baseline.

La struttura dati principe di questa esercitazione è la *HashMap* che permette di accedere in tempi contenuti ai dati. La maggior parte delle classi sono state definite statiche poichè non riusulta necessario manipolare istanze diverse (es. LearnCorpus.java).

L'esercitazione è scomponibile in due task fondamentali :

- Apprendimento del modello (HMM)
- Decodifica: Algoritmo di Viterbi

per l'**Apprendimento** sono state realizzate un insieme di classi che sono contenute in due package (hmm e morph_it). Il primo package contiene la classe responsabile del apprendimento del modello HMM da un corpus e le classi che realizzano un astrazione sul modello HMM appreso (HMM.java, Antecedent.java). Il secondo package contiene le classi che sintetizzano morph-it (un semplice file di testo) rendedolo uno strumento più performante e accessibile (Morph_IT.java, PairTermPos.java).

La **Decodifica** è realizzata da 3 classi che implementano l'algorimo baseline, l'algoritmo di viterbi (bigrammi) e l'algoritmo di viterbi Special (trigrammi). Le classi per la decodifica fanno parte del package *algorithm* insieme a classi di supporto(Cell.java, AlfaParamter.java). L'algoritmo ViterbiSpecial utilizza la classe AlfaParameter.java per lo smoothing *deleted interpolation*.

Il package *test* contiene casi di test sulle classi realizzate. Le performance dei 3 algoritmi di decodifica sono state misurate mediante l'accuratezza e l'accuratezza media delle frasi.

$$accuratezza = \frac{TP}{POS}$$

$$accuratezzaMedia = \frac{\sum_{x \in Frasi} acc(f)}{\#Frasi}; acc(f) = \frac{TP(f)}{\#PoS\ in\ f}$$

Apprendimento del Modello (HMM)

L'apprendimento del modello HMM è effettuato dalla classe statica *LearnCorpus.java* che per l'apprendimento richiede in input un file .txt contenente un insieme di frasi corredate dalla sequenza di *Part of Speech*. Sono state utilizzate diverse strutture per mantenere i *count* delle transizioni e delle produzioni estratte dal corpus:

Listing 1: Strutture dati LearnCorpus.java

```
1 HashMap<String, HashMap<String, Double>> pos2_pos = new HashMap<>();
2 HashMap<String, HashMap<String, Double>> pos_pos = new HashMap<>();
3 HashMap<String, HashMap<String, Double>> term_pos = new HashMap<>();
4 HashMap<String, Integer> freq = new HashMap<>();
5 HashMap<String, Integer> freq2pos = new HashMap<>();
6 int n_pos = 0;
7 int N = 0;
8 HashSet<String> posTag = new HashSet<>();
```

- *pos_pos*: un *HashMap* utilizzata come struttura di supporto per apprendere le probabilità $P(t_i|t_{i-1})$ (transizione)
- *pos2_pos*: un *HashMap** utilizzata come struttura di supporto per apprendere le probabilità $P(t_i|t_{i-1}, t_{i-2})$(transizione).* La stringa utilizzata come prima chiave è una rappresentazione di una istanza della classe Antecedent.java.
- *term_pos*: un *HashMap* utilizzata come struttura di supporto per apprendere le probabilità $P(w_i|t_i)$(produzione/emissione).
- *freq*: un *HashMap* utilizzata per memorizzare i *count* di ogni PoS Tag. Le frequenze dei PoS sono utilizzate nella stima delle probabilità.
- *freq2pos*: un *HashMap** utilizzata per memorizzare i *count* di ogni coppia di PoS Tag.
- *posTag*: un *HashSet* contenente il Tag Set appreso dal corpus.
- *n_pos*: numero di PoS tag di cui è composto il Tag Set appreso.
- *N*: numero di coppie (parola,tag) utilizzate per l'apprendimento del modello.

I PoS tag estratti dal corpus vengono generalizzati per ridurne il numero (es. $DET\_A \rightarrow DET$).

L'algoritmo d'apprendimento prevede di leggere il file corpus per riempire le strutture dati contando le occorrenze (t_i, t_{i-1}), (t_i, t_{i-1}, t_{i-2}), (w_i, t_i) e (t_i). Finito di riempire le strutture dati, utilizza $freq$ e $freq2pos$ per convertire i count contenuti in *pos_pos*, *pos2_pos*, *term_pos* in probabilità. La conversione in probabilità è effettuata in maniera differete nelle tre strutture poichè ognuna conterrà probabilità di natura differente.

- *pos_pos* contiene $C(t_{i-1}, t_i)$ che è il numero di occorenze del bigramma nel corpus. Per ottenere una stima della probabilità $P(t_i|t_{i-1})$ si divide per $C(t_{i-1})$ valore contenuto in $freq$.
- *pos2_pos* contiene $C(t_{i-2}, t_{i-1}, t_i)$ che è il numero di occorenze del trigramma nel corpus. Per ottenere una stima della probabilità $P(t_i|t_{i-1}, t_{i-2})$ si divide per $C(t_{i-2}, t_{i-1})$ valore contenuto in $freq2pos$.
- *term_pos* contiene $C(t_i, w_i)$ che è il numero di occorenze della coppia (parola,PoS) nel corpus. Per ottenere una stima di $P(w_i|t_i)$ si divide il *count* per $C(t_i)$.

Dopo l'apprendimeto nel strutture saranno contenute le probabilità di transazione ed emissione del HMM.

La sintesi di morph-it è effettuata dalla classe statica *Morph_IT.java* che si occupa di leggere il file .txt contenente le coppie parola, annotazione morfologica. La classe Morph_IT richiede in input il file morph-it e un file contenente il mapping tra i PoS di morph-it e quelli utilizzati per annotare le frasi nel algoritmo di decodifica.

La classe HMM.java sintetizza morph-it e incapsula i risultati prodotti da LearnCorpus.java per offrire istanze di HMM utilizzabili a livello di decodifica. Le strutture dati di HMM sono quelle prodotte dalla fase di apprendimento (*pos_pos*, *pos2_pos*, *term_pos*) utilizzate per esporre semplici metodi di interrogazione.

Listing 2: Metodi HMM.java

```
1  public Double pos_to_pos(String prev, String pos)
2        return P(pos|prev)
3  public Double term_to_pos(String term, String pos)
4        return P(term|pos)
5  public Double antecedent_to_pos(Antecedent x, String pos)
6        return P(pos|x=pos1,pos2)
7  public String bestPoS(String term);
```

I metodi esposti da HMM permetto di ottenere le probabilità di transizione e emissione oltre al miglior PoS di un termine. In questi metodi vengono utilizzati criteri di smoothing per evitare le probabilità nulle. Viene effettuato uno smoothing per le

probabilità di transizione assegnando un valore molto basso a quelle transizioni con probabilità 0. Il valore in questione è $\frac{1}{N+1}$ dove N è il numero di casi usati per l'apprendimento del modello.

Decodifica

Per la decodifica sono state realizzate 3 classi statiche *BaseLine.java*, *Viterbi.java*, *Viterbi-Special.java*. Le 3 classi richiedono di essere inizializzate con un istanza di HMM.

BeseLine.java è l'implementazione di un algoritmo baseline del problema del PoS tagging. L'algoritmo prevede di assegnare ad ogni parola del contesto il PoS tag più probabile. Utilizza il metodo *bestPos* della classe HMM.java che dato un termine ritorna il pos più probabile. Se il termine passato a *bestPoS* è sconosciuto assegna un pos valido cercando in Morph_IT.java, altrimenti il pos NOUN.

L'algoritmo baseline legge il contesto e per ogni termine richiama il metodo bestPos e assegna il Pos risultato.

L'algoritmo baseline ottiene sul test set accuratezza del 72.75% e accuratezza media pesata di 71.77%.

Viterbi.java è l'implementazione del algoritmo di decodifica di viterbi. L'algoritmo di viterbi sfrutta l'assunzione Markoviana di grado 1 (dipendenza solo dallo stato precedente) per trovare mediante un algoritmo di programmazione dinamica la sequenza più probabile di PoS data la frase. Viterbi.java usa un'istanza di HMM.java che rappresenta il modello alla base del algoritmo. L'algoritmo implementato segue la specifica date a lezione con l'unica eccezione di aver invertito le righe con le colonne nella matrice usata per trovare la soluzione. Le celle della matrice sono istanze della classe *Cell.java*. Le istanze della classe Cell.java sono caratterizzate da 2 attributi

- $Cell[i][j].p$: probabilità del miglior assegnamento di pos tag della sotto frase (0,i) che termina con pos Pos(j).(utilizzato per il calcolo)
- $Cell[i][j].father$: indica l'indice del PoS che precede PoS(j) nella sequenza di assegnamento migliore della sotto frase (0,i) che termina con PoS(j).(backpointer per la ricostruzione della soluzione migliore)

Vengono utilizzati due PoS speciali per lo stato iniziale($0 * 0$) e lo stato finale($1 * 1$) e le relative probabilità di transizione.

L'algoritmo di viterbi si basa sull'induzione:

$$\begin{cases} v_1(j) = a_{\mathrm{j},0} \cdot b_{\mathrm{j}}(1) \; \forall 1 \leq j \geq N \\ v_{\mathrm{i}}(j) = \max\limits_{1 \leq k \leq N} v_{\mathrm{i}-1}(k) \cdot a_{\mathrm{j,k}} \cdot b_{\mathrm{j}}(i) \; \forall 1 \leq j \geq N; \; \forall 2 \leq i \geq T \end{cases}$$

- $v_{\mathrm{i}}(j)(Cell[i][j].p)$ è la probabilità del percorso più verosimile che termina nella parola word(i) con tag PoS(j).
- $a_{\mathrm{j,k}}(HMM.pos\_to\_pos(PoS(j), PoS(k)))$ è la probabilità di transizione dal PoS(k) al PoS(j); $(P(PoS(j)|PoS(k)))$.
- $b_{\mathrm{j}}(i)(HMM.term\_to\_pos(word(i), PoS(i)))$ è la probabilità di emissione di word(i) dato il PoS(j); $(P(word(i)|PoS(j)))$.
- N è la cardinalità del insieme dei PoS Tag.
- T è la lunghezza della sequenza delle parole.

$HMM.term\_to\_pos(word(i), PoS(i))$ restituisce la probabilità di emissione di una parola dato il pos tag. Se il termine in esame è sconosciuto si cercano i possibili PoS tramite morph-it, se il pos di interesse è uno dei possibili si restituisce come probabilità $\frac{1}{|possibili\_Pos|}$. Se il termine in esame non è presente in morph-it si restituisce $\frac{1}{|PoS|}$.

L'algoritmo inizializza la prima riga della matrice con la probabilità di iniziare la frase con un particolare Pos dato un termine; probabilità di transitare dallo stato iniziale in quello con il pos in esame per la probabiltà di emissione del termine dato il pos in esame. Si continua a riempire la matrice seguendo l'induzione fino a trovare il massimo cercando quella sequenza che ha probabilità massima considerando anche la transazione nello stato finale.

L'algoritmo di viterbi ottiene sul test set accuratezza del 95.24% e accuratezza pesata per le lunghezze delle frasi di 94.59%.

ViterbiSpecial.java è un implementazione estesa del algoritmo di viterbi. La differenza dal algoritmo precedente sta nella stima di $P(t_i|t_{i-1})$:

$$P(t_i|t_{i-1}) = \alpha_3 P(t_i|t_{i-1}, t_{i-2}) + \alpha_2 P(t_i|t_{i-1}) + \alpha_1 P(t_i)$$

I parametri α sono rappresentati dalla classe AlfaParameter.java che si occupa anche del apprendimento di quest'ultimi. L'apprendimento dei parametri è fatto mediante l'algoritmo di *deleted interpolation*. Il metodo *run* della classe ViterbiSpecial richiede un istanza di HMM e un istanza di AlfaParameter.

Una volta ottenuti gli α l'algoritmo procede in modo analogo al algoritmo standard utilizzando la formula per la stima delle probabilità di transizione.

Scelta importante è stata la classe Antecedent.java che rappresenta la coppia t_{i-1}, t_{i-2} usata formalmente come chiave nella struttura che memorizza le probabilità. L'utilizzo di Antecedent è solo formale poichè la chiave è una stringa costruita concatenando i due posTag.

L'algoritmo di *deleted interpolation* anche se segue le specifiche date a lezione non migliora di molto i risultati. Sicuramente sarà presente un errore nel calcolo degli α.

L'algorimo ViterbiSpecial ottiene sul test set accuratezza del 95.29% e accuratezza pesata per le lunghezze delle frasi di 94.82% con i parametri appresi automaticamente. Un semplice assegnamento a mano degli α come $\alpha_3 = 0.4, \alpha_2 = 0.6, \alpha_1 = 0.0$ ottiene risultati migliori (accurattezza=95.53% e accuratezza pesata=95.05%) dimostrando qualche problema nella stima degli α.