Luca Sorrentino

Tecnologie del Linguaggio Naturale

Relazione esercizio d'esame: Traduttore direct IT-EN

– Parte prima: Alessandro Mazzei –

https://github.com/DrLux/NLP-Exercises/tree/master/viterbi

Indice

1 Descrizione del software 1
1.1 Linguaggio scelto 1
1.2 Classi implementate 1
1.2.1 Translator 1
1.2.2 Dictionary 2
1.2.3 Viterbi 3

2 Risultati e ottimizzazioni 5
2.0.1 Processing del Data Set 5
2.0.2 Tuning e ottimizzazioni 5
2.0.3 Risultati finali 6

3 Pos Tagging e Traduzioni 9
3.0.1 Traduzione delle frasi 9

1

Descrizione del software

Tra gli esercizi proposti, ho scelto il primo, ovvero quello che richiede di implementare il PoS tagger statistico per litaliano basato su HMM ed usarlo in un sistema di traduzione direct.

1.1 Linguaggio scelto

Inizialmente la scelta del linguaggio di programmazione da usare e' ricaduta su Python per via della sua semplicita' e praticita' di utilizzo, anche se il prezzo da pagare e' la velocita' di esecuzione. Per questo motivo ho deciso di realizzato in poco tempo un' implementazione "prototipo" dell' tagger, e poi, una volta volta presa confidenza con l'algoritmo ho riscritto il tutto in Java andando a decimare i tempi di esecuzione, passando letteralmente da 10 secondi per una frase a meno di uno nell' implementazione finale.

1.2 Classi implementate

- Translator: classe che contiene il Main, si occupa di richiamare l' algoritmo di pos tagging e di tradurre andando ad applicare le regole di riordinamento delle parole ottenute;
- Viterbi: contiene il cariamento dei dati del data set, l' algoritmo di pos tagging e le routine per il calcolo della precisione del modello sul test set;
- Dictionary: e' un enorme dizionario che associa ogni parola italiana ad una corrispettiva inglese, in base al pos tag associato.

1.2.1 Translator

Per prima cosa instanzia l' oggetto Viterbi (andando a caricare in memoria tutti i dati necessari), dopodiche' calcola la baseline, la precisione del modello e traduce le frasi richieste nell' esercitazione.

Andiamo a vedere un po' piu' nel dettaglio cosa succede all' interno di questa classe. Per prima cosa, la frase subisce un preprocessing, in cui ogni preposizione articolata viene scissa in due parole, una contenente la preposizione di base e un'altra contenente l'articolo. Il tutto viene realizzato con uno switch case seguendo i casi riportati nella tabella qui di seguito.

	IL	LO	L'	LA	I	GLI	LE
DI	DEL	DELLO	DELL'	DELLA	DEI	DEGLI	DELLE
A	AL	ALLO	ALL'	ALLA	AI	AGLI	ALLE
DA	DAL	DALLO	DALL'	DALLA	DAI	DAGLI	DALLE
IN	NEL	NELLO	NELL'	NELLA	NEI	NEGLI	NELLE
SU	SUL	SULLO	SULL'	SULLA	SUI	SUGLI	SULLE

Figura 1.1. Tabella delle preposizioni articolate.

A questo punto viene richiamata la routine di pos tagging che associa ad ogni parola la relativa "parte del discorso" ed, a questo punto, il tutto viene inviato alla routine di traduzione.

La routine di traduzione associa ad ogni parola la rispettiva versione inglese, per mezzo di un dizionario, ma in alcuni casi deve effettuare delle operazioni piu' complesse, come andare a vedere i tag successivi a quello attuale e fornire anche questa informazione al dizionario per il recupero della traduzione piu' appropiata.

A questo punto avremo una lista di termini in inglese e il corrispettivo pos tag che ci serviranno per effetturare delle operazioni di riorganizzazione delle parole (nei nostri casi semplicemente uno scambio tra nome ed agettivo, ma e' facilmente possibile estendere il set di regole aggiungendo riordinamenti tra pronome e verbo in caso di domande o tra negazione e verbo in case di frasi negative...). In questa fase inoltre, andremo anche ad aggiungere i soggetti delle azioni, che nelle frasi in italiano possono essere impliciti. (Questa operazione non viene fatta nel preprocessing in quanto ci sono alcune parole come "it", non naturalmente presenti nel nostro vocabolario e che avrebbero distrorto troppo la frase di input).

1.2.2 Dictionary

Si tratta di una HashMap che collega una parola italiana ad una seconda sub-HashMap in cui ogni possibile traduzione di quella parola viene associata al rispettivo pos tag. Oltre a cio', ci sono anche un paio di accorgimenti in piu', ovvero alcuni casi in cui vengono presi in considerazione anche i tag successivi, come nell' esempio dell' immagine qui di seguito.

```
Map<String, String> stati = new LinkedHashMap<String, String>();
stati.put("NOUN", "states");
stati.put("AUX-VERB", "been");
stati.put("VERB", "were");
words.put("stati", stati);
```

Figura 1.2. Esempio di gestione di una parola del dizionario.

Il caso di parole che acquisiscono un certo senso solo se messe vicine, viene gestito nel modo piu' naturale possibile, ovvero vi e' un significato per le parole a se stanti (esempio: spada = sword) e un terzo per l'unione delle due parole (esempio: spada laser = lightsaber).

1.2.3 Viterbi

Durante la fase di inizializzazione vi e' la lettura di tutte le informazioni del training set in memoria.Durante questo processo vengono precalcolate alcune informazioni utili come:

- il conteggio totale di ogni tag;
- una mappa che associa ogni parola col numero di volte in cui cooccorre con un relativo tag. (ed anche il numero totale di volte in cui compare questa parola);
- la lista di tutti i tag in sequenza (che ci agevolera' nel calcolo delle probabilita' di transizione).

Per quanto riguarda il calcolo delle probabilita' di transizione, questo e' di gran lunga il collo di bottiglia di tutto l'algoritmo, in quanto non c'e' modo di ottimizzare la ricerca se non scorrere ogni volta tutta la lista. Essendo pero' un'operazione abbastanza semplice e statica sarebbe possibile effettuarla a priori, una sola volta, per tutti i tag possibili e salvare i risultati in un file. Nonostante tutto cio' comporterebbe un vantaggio in termini di tempo, l' ammontare di tutte le possibili combinazioni da precalcolare e' esagerato, per cui ho optato per una soluzione piu' sottile.

Siccome in realta' sono molte meno le sequenze di tag che sono realmente necessarie, ho deciso di realizzare un sistema di "cache" che, una volta calcolate, le memorizzi per utilizzi futuri. In questo modo ho reso le operazioni su tutto il training set molto pi veloci (abbastanza da rendere non necessario prevedere un sistema di salvataggio su file di questa "cache", in quanto basta un calcolo di probabilit di sequenza di tag on demand in caso di "cache miss").

2

Risultati e ottimizzazioni

In questa sezione andiamo a vedere quali sono state le scelte implementative e i risultati ottenuti.

2.0.1 Processing del Data Set

Fin dalle prime fasi di sviluppo avevo delle perplessita' sul dataset:

- come posso calcolare la differenza tra la sequenza di tag del mio modello e quella del test set se non riesco a individuare correttamente la fine di una frase e l'inizio della successiva?
- nel calcolo del tag, il conoscere con quale probabilita' un certo tag pu comparire ad inizio frase o esserne l' ultimo, puo' rivelarsi un' informazione rilevante?

Per questi motivi ho deciso di scaricare le versioni originali del dataset e riprocessarle, andando ad aggiungere tag di START e di END ad inizio e fine frase. Durante questo processo ho anche corretto alcuni errori minori, come per esempio la presenza di caratteri di errore taggati come AUX, che sono stati corretti. (In realta' per, questo processo si rivelato quasi ininfluente nel calcolo del risultato finale).

2.0.2 Tuning e ottimizzazioni

A questo punto ho calcolato il livello di precisione del mio modello sul Dev Set. Nelle fasi iniziali non ho usato il Test set in quanto le calibrazioni degli iper parametri vanno fatte sul Dev set, altrimenti i valori finali potrebbero venire falsificati da approcci che funzionano meglio su casi specifici del Test set.

Inizialmente, utilizzando le indicazioni suggerite a lezione, ho ottenuto un risualto di:

Baseline: 88.03

Precision: 82.65
A questo punto per ho fatto delle piccole modifiche all' algoritmo:

1. Ottimizzazione celata: lo scomporre le preposizione articolata migliora le performance su casi reali, anche se non incide sul calcolo di precisone sul Test/Dev set in quanto li le preposizioni sono gi scomposte.
2. Nel caso in cui la parola sia conosciuta al modello, ma appaia con un tag mai associato ad essa, le assegno un valore di probabilita' infinitamente piccolo (per non avere probabilit nulle). In questo modo arrivo a:
   Precision: 90.06
3. Nel caso in cui la parola sia sconosciuta, probabilmente sara' un nome o un verbo (in quanto i tag aperti pi comuni) e allora quando la parola e' sconociuta ma il tag proposto e' uno di quei due, assegno una probabilita' del 50. In questo modo ho ottenuto:
   Precision: 91.04
4. A questo punto ho provato a vedere se la situazione migliorasse usando il Test set, ma prima di farlo ho unito Training e Dev set, per aumentare il numero di dati a disposizione. In questo modo sono arrivato a:
   Baseline: 92.28
   Precision: 94.34
5. Visto che aumentare il numero di dati a disposizione aveva comportato (senza troppe sorprese) un cosi netto miglioramento ho provato a farlo di nuovo, ma in un altro modo. Nel dataset originale vi erano pi informazioni oltre le parole, e nello specifico ho notato che vi erano anche le rispettive versioni lemmatizzate, come sottolineato dall' immagine qui di sotto. Allora ho provato a produrre un Dataset in cui oltre alle parole normali venivano usate anche le versioni lemmatizzate (quando vi era una differenza).
   In questo caso per, il risultato ottenuto e' stato deludente:
   Precision: 91.69
   La mia spiegazione e' che in questo modo andavo a creare frasi di training poco realistiche (come ad esempio: "e' essere stata essere la il giornata...") che andavano a falsare i calcoli di probabilita' di transizione da un tag al successivo.

2.0.3 Risultati finali

Dopo aver provato tutte le ottimizzazioni possibili, sono arrivato a questi risultati, presi direttamente dall'output del programma:

Calcolo della Baseline!
Baseline: 92.2812283039093
Tempo di computazione: 0.097 secondi.
Calcolo del livello di precisione del modello!
Precision: 94.34024095535278

```
# sent_id = isst_tanl-6
# text = una sala ha dovuto essere
1       una     uno     DET     RI
2       sala    sala    NOUN    S
3       ha      avere   AUX     VA
4       dovuto  dovere  AUX     VM
5       essere  essere  AUX     VA
6       sgomberata      sgomberare
7       per     per     ADP     E
8       una     uno     DET     RI
9       fuga    fuga    NOUN    S
10      di      di      ADP     E
11      gas     gas     NOUN    S
12      tossico tossico ADJ     A
13      da      da      ADP     E
```

Figura 2.1. Nella prima colonna troviamo la parola originale, nella seconda la versione lemmatizzata.

Tempo di computazione: 2.246 secondi.
BUILD SUCCESSFUL (total time: 3 seconds)

3

Pos Tagging e Traduzioni

3.0.1 Traduzione delle frasi

Qui di seguito verranno inseriti gli esiti delle traduzioni delle 3 frasi di esempio, presi direttamente dall' output del software:

Prima frase

Frase in input: [E', la, spada, laser, di, tuo, padre]
Pos Taggging:
E' = AUX
la = DET
spada = NOUN
laser = NOUN
di = ADP
tuo = DET
padre = NOUN
Traduzione intermedia: [is, the, lightsaber, of, your, father]
Traduzione finale: [it, is, the, lightsaber, of, your, father]

Seconda frase

Frase in input: [Ha, fatto, una, mossa, leale]
Pos Tagging:
Ha = AUX
fatto = VERB
una = DET
mossa = NOUN
leale = ADJ
Traduzione intermedia: [made, a, move, fair]
Traduzione finale: [he, made, a, fair, move]

Terza frase

Frase in input: [Gli, ultimi, avanzi, di, la, vecchia, Repubblica, sono, stati, spazzati, via]

Pos Tagging:
Gli = DET
ultimi = ADJ
avanzi = NOUN
di = ADP
la = DET
vecchia = ADJ
Repubblica = NOUN
sono = AUX
stati = AUX
spazzati = VERB
via = ADV

Traduzione intermedia: [the, last, remnants, of, the, old, republic, have, been, swept, away]

Traduzione finale: [the, last, remnants, of, the, old, republic, have, been, swept, away]