Progetto di Intelligenza Artificiale
POS Tagging con Brill Tagger

Diego Mazzalovo
Matr. 1236592

A.A. 2019-2020

1 Approcci Basati sulle Regole

1.1 Introduzione

I primi sistemi di taggatura del testo, sono stati i sistemi a regole, dove veniva manualmente creato un set di regole e poi applicato al testo, che inizialmente veniva taggato con un *lexicon*[1] per migliorarlo. Le regole vengono usate per correggere i tag delle parole che risultavano avere più tag e le quali avessero vicine parole con un solo tag. Il tagging finale, viene ricavato dall'applicazione di multiple regole per correggere gli errori presenti in fase di addestramento.

1.2 Apprendimento Basato sulla Trasformazione

Nel 1995, Eric Brill [Brill(1995)], sperimentò un nuovo approccio nei sistemi basati sulle regole. Invece di scrivere manualmente le regole, crea un sistema in grado di dedurre una serie di regole di correzione dal testo, inventando così la trasformation-based learning o TBL.

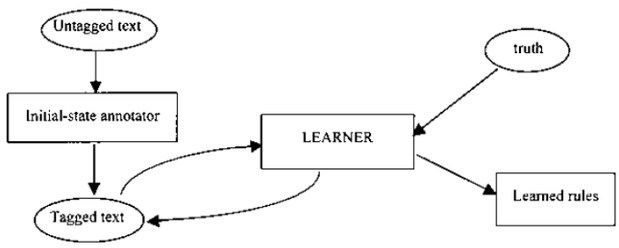

Figure 1: Struttura degli Algoritmi di tipo TBL

Inizialmente, il *corpus*[2] viene taggato con uno tra vari metodi possibili: scegliere un tag a caso tra quelli possibili per la parola, assegnare il tag visto più spesso per quella parola durante il *training* o semplicemente assegnare il tag nome, in quanto risulta essere il tag più comune. Successivamente, usando un *set di template di regola*, il sistema inizializza le regole basandosi sul testo in esame, per poi applicare tutte le regole una alla volta e temporaneamente al testo, al fine di vedere quale di esse esegua il maggior numero di correzioni nel testo. Viene poi presa la regola che effettua il maggior numero di correzioni, che viene aggiunta al set di regole apprese e poi applicata al testo, per poi ripetere questo procedimento fino ad a raggiungere una determinata soglia di correttezza decisa in fase di training.

1.3 Template di Regola

I template di regola si riferiscono ad un contesto vicino alla parola, come ad esempio le parole o i tag precedenti e successivi. Ogni regola è composta di 2 parti:

- Un ambiente di trigger(condizione necessaria per far attivare la regola);

[1] Dizionario dove ad ogni parola viene associato un relativo tag
[2] File di addestramento.

- Regola di riscrittura(azione da intraprendere).

Un esempio di regola, potrebbe essere:

```
Cambia da tag da "verbo" a "nome" se il tag precedente è "articolo"
```

Riprendendo l'esempio 1, ad esempio, un tag iniziale potrebbe essere :

```
La/articolo sposa/verbo si/verboModale sposa/verbo
```

In questo caso, alla parola sposa verrebbe assegnato ogni volta il tag verbo, in quanto risulta essere quello più comune per quella parola. Dopo aver applicato la regola sopra citata, il tagging diverrebbe quindi:

```
La/articolo sposa/nome si/verboModale sposa/verbo
```

Correggendo così l'errore inizialmente commesso. E' importante specificare che vengono imparate diversi tipi di regole per le parole contenute nel *lexicon* e quelle invece non contenute nel *lexicon* anche dette sconosciute.

1.4 Regole per Parole Conosciute

Change the tag from A to B if		
$t_{i-1} = \mathtt{X}$	$t_{i-2} = \mathtt{X}$ and $t_{i+1} = \mathtt{Y}$	$w_i = \mathtt{X}$ and $t_{i+1} = \mathtt{Y}$
$t_{i+1} = \mathtt{X}$	$w_{i-1} = \mathtt{X}$	$w_i = \mathtt{X}$
$t_{i-2} = \mathtt{X}$	$w_{i+1} = \mathtt{X}$	$w_{i-1} = \mathtt{X}$ and $t_{i-1} = \mathtt{Y}$
$t_{i+2} = \mathtt{X}$	$w_{i-2} = \mathtt{X}$	$w_{i-1} = \mathtt{X}$ and $t_{i+1} = \mathtt{Y}$
$t_{i-2} = \mathtt{X}$ or $t_{i-1} = \mathtt{X}$	$w_{i+2} = \mathtt{X}$	$t_{i-1} = \mathtt{X}$ and $w_{i+1} = \mathtt{Y}$
$t_{i+1} = \mathtt{X}$ or $t_{i+2} = \mathtt{X}$	$w_{i-2} = \mathtt{X}$ or $w_{i-1} = \mathtt{X}$	$w_{i+1} = \mathtt{X}$ and $t_{i+1} = \mathtt{Y}$
$t_{i-3} = \mathtt{X}$ or $t_{i-2} = \mathtt{X}$ or $t_{i-1} = \mathtt{X}$	$w_{i+1} = \mathtt{X}$ or $w_{i+2} = \mathtt{X}$	$w_{i-1} = \mathtt{X}$ and $t_{i-1} = \mathtt{Y}$ and $w_i = \mathtt{Z}$
$t_{i+1} = \mathtt{X}$ or $t_{i+2} = \mathtt{X}$ or $t_{i+3} = \mathtt{X}$	$w_{i-1} = \mathtt{X}$ and $w_i = \mathtt{Y}$	$w_{i-1} = \mathtt{X}$ and $w_i = \mathtt{Y}$ and $t_{i+1} = \mathtt{Z}$
$t_{i-1} = \mathtt{X}$ and $t_{i+1} = \mathtt{Y}$	$w_i = \mathtt{X}$ and $w_{i+1} = \mathtt{Y}$	$t_{i-1} = \mathtt{X}$ and $w_i = \mathtt{Y}$ and $w_{i+1} = \mathtt{Z}$
$t_{i-1} = \mathtt{X}$ and $t_{i+2} = \mathtt{Y}$	$t_{i-1} = \mathtt{X}$ and $w_i = \mathtt{Y}$	$w_i = \mathtt{X}$ and $w_{i+1} = \mathtt{Y}$ and $t_{i+1} = \mathtt{Z}$

Figure 2: Template di regola per le parole conosciute usati nei TBL

In figura 2 si possono vedere i template di regola comunemente usati per le parole conosciute, dove t sta per tag e w sta per parola. Essi, basano i loro cambiamenti principalmente sulla base delle parole e dei tag in contesti vicini.

1.5 Regole per Parole non Conosciute

Change the tag of an unknown word from $\boldsymbol{a}$ to $\boldsymbol{b}$ when:

1. Deleting the prefix (suffix) $\boldsymbol{x}$ results in a word
2. The prefix (suffix) of the word is $\boldsymbol{x}$
3. Adding the prefix (suffix) $\boldsymbol{x}$ results in a word
4. The preceding (following) word is $\boldsymbol{w}$
5. The character $\boldsymbol{c}$ appears in the word

where $\boldsymbol{x}$ is a string of length 1 to 4.

Figure 3: Template di regola per le parole sconosciute usati nei TBL

Le regole per le parole sconosciute, si basano invece principalmente sulla morfologia della parola corrente come si può notare in figura 3, cercando di modificare il tag inizialmente attribuito senza basarsi sui tag vicini, in quanto applicando le stesse regole applicate per le parole conosciute, si potrebbe incorrere in errori o in riduzioni di correzioni non desiderate. Siccome spesso si utilizzano corpus completamente taggati, vengono trattate come parole sconosciute le parole con una bassa frequenza, come ad esempio meno di 5 occorrenze. In fase di addestramento, vengono inoltre imparate prima le regole per le parole sconosciute e, una volta apprese vengono applicate in ordine al tagging iniziale eseguito come in 1.2 per poi procedere con l'apprendimento delle regole per le parole conosciute.

1.6 Funzione di Costo

La funzione di costo più comunemente usata per apprendere le regole, risulta essere l'assegnare un punteggio ad ogni regola sulla base delle "buone" trasformazioni fatte, ovvero prendendo quella per cui #giuste - #sbagliate risulta essere massimo. Sono possibili anche altri tipi di funzione costo, come ad esempio prendendo la regola che corregge il maggior numero di errori senza però commetterne.

1.7 Processo di Taggatura

Inizialmente, viene creato quindi un *lexicon*, contenente quante più parole possibili, dove ad ogni parola, nel caso avesse più tag possibili, viene assegnato il tag più probabile per quella parola. Spesso, il dizionario viene creato direttamente dal file di addestramento, quindi il numero che una volta appare con un determinato tag, viene calcolato sul *corpus* di addestramento. Si procede quindi a taggare il testo con il *lexicon*, mentre per le parole sconosciute, normalmente, viene assegnato il tag "nome" in quanto tag più comune o il tag "nome proprio" se la parola inizia con la maiuscola. Successivamente, vengono quindi applicate le regole per le parole sconosciute alle parole non conosciute e poi le regole normali a tutto il testo.

2 Implementazione

2.1 Corpus e Tags

In ognuno dei sistemi seguenti, è stato usato per l'addestramento il corpus LaRepubblica [3], composto da 14.167 frasi e 298.344 parole. É stato inoltre utilizzato lo standard di Universal Dependency[4], composto da 17 tags :

- **ADJ**: aggettivo;
- **ADP**: adposizioni. Sono l'insieme delle preposizioni e postposzioni;
- **ADV**: avverbio;
- **AUX**: ausiliari, ovvero parole che accompagnano i verbi, ad esempio in "ho mangiato", la parola "ho" è AUX;
- **CCONJ**: congiunzione coordinative;
- **DET**: sono parole che modificano il significato di un nome o di una frase nominale, comprendono tra gli altri: articoli determinativi,pronomi numerali, pronomi possesivi..;
- **INTJ**: interiezioni, ovvero parole usate come esclamazioni;
- **NOUN**: sostantivi;
- **NUM**: numeri;
- **PART**: parole che accompagnano altre per darci significato, come ad esempio il "'s" in inglese;

[3]https:// universaldependencies.org/ #download
[4]https:// universaldependencies.org/ u/ pos/ index.html

- **PRON**: pronomi;
- **PROPN**: nomi propri;
- **PUNCT**: punteggiatura;
- **SCONJ**: congiunzioni subordinanti;
- **SYM** : simboli;
- **VERB**: verbi;
- **X**: altro.

2.2 Lexicon

Per lo sviluppo del Tagger, ho utilizzato un dizionario preso da un progetto svolto all'università di bologna chiamato Morph- IT [Zanchetta(2004-2007)], il quale è composto da 505.074 parole italiane e da 667 tag diversi, dati dai singoli tag con le varie forme che possono assumere. Per prima cosa, ho ridotto il numero complessivo di tag conformandolo con lo standard di Universal Dependencies come descritto in 2.1. Successivamente, al fine di creare il dizionario per il Brill Tagger, ho estratto per ogni parola, solo il tag più comune tra quelli presenti per tale parola.

2.3 Template di Regola Applicati

2.3.1 Regole per Parole Sconosciute

Nel tagger, sono stati applicati i seguenti template di regola per le parole sconosciute:

- Il suffisso di lunghezza 3 della parola è x;
- Il suffisso di lunghezza 4 della parola è x;
- Il suffisso di lunghezza 5 della parola è x;
- La parola prima di quella attuale è p;
- La parola dopo quella attuale è p.

2.3.2 Regole Generali

Nel tagger, sono stati applicati i seguenti template di regola per le parole:

- Il tag precedente è t;
- Il tag successivo è t;
- Il tag 2 posizioni prima è t;
- Il tag 2 posizioni dopo è t;
- Il tag precedente è t1 e il tag successivo è t2;
- Il tag precedente è t1 e il tag 2 posizioni prima è t2;
- Il tag successivo è t1 e quello 2 posizioni dopo è t2;
- Il tag t è presente in uno dei 2 tag prima;
- Il tag t è presente in uno dei 2 tag dopo; Il tag t è presente in uno dei 3 tag prima;
- Il tag t è presente in uno dei 3 tag dopo;
- La parola prima è p;
- La parola successiva è p;

- La parola 2 posizioni prima è p;
- La parola 2 posizioni dopo è p;
- La parola p si trova in una delle 2 parole prima;
- La parola p si trova in una delle 2 parole dopo.

2.4 Miglioramenti

Per questione di tempo, sono state applicate delle ottimizzazioni come in [Larsson and Norelius()]. Per avere una velocità maggiore ho applicato le seguenti ottimizzazioni:

- Tutti i tag durante l'addestramento e successivamente la taggatura, sono tradotti in numeri;
- Le regole vengono istanziate solo una volta all'inizio, invece che ricrearle dopo ogni taggatura. Ad ogni interazione viene semplicemente rivalutato il loro numero di correzioni eventuale;
- Durante il processo di addestramento, quando viene valutata una regola, non vengono anche eseguite le trasformazioni, così da non dover copiare il file ogni volta.

2.5 Codice

2.5.1 File Addestra.js

Il file `Addestra.js`, si occupa di addestrare le regole per la trasformazione dei tag del brill tagger. Esso utilizza i 2 file `generaRegoleSconosciute.js` e `generaRegole.js`, i quali sono composti dai seguenti metodi:

- **TaggaFrase()** : esegue il tag iniziale della frase, in particolare, se trova la parola nel Morph, assegna il tag corrispondente, altrimenti: assegna il tag _NUM qualora sia un numero, _PROPN se la parola inizia con una lettera maiuscola, altrimenti _NOUN, ovvero il tag più comune in assoluto;
- **Stagga()** : toglie i tag dal file di train, per poi poter riutilizzare il file così ottenuto ritaggandolo;
- **TaggaPerBrill()** : usa il metodo TaggaFrase() per taggare il file di train staggato dal metodo Stagga();
- **TraduciTrainPerBrill()** : estrae i tag dal file di train e li traduce in numeri al fine di aver maggiore velocità nel paragonare i tag;
- **CreaSconosciute()** : crea un file dove ogni parola viene sostituita da un 1 se risulta essere una parola sconosciuta o da uno 0 altrimenti;
- **InizializzaPerRegole()** : chiama i metodi precedentemente descritti;
- **ValutaPossibilitaPerSconosciute()** : data una frase e la posizione della parola, valuta tutti i template di regola per quella parola al fine di istanziare le corrispondenti regole per le parole sconosciute;
- **TrovaRegolaMiglioreSconosciuta()** : valuta tutte le regole istanziate per le parole sconosciute, salvando all'interno della regola il numero di cambiamenti corretti che tale regola applicherebbe qualora utilizzata;
- **EstraiPiuPromettenteSconosciuta()** : ordina l'array contenente le regole(regoleSconosciute), sulla base della valutazione eseguita con il metodo TrovaRegolaMiglioreSconosciuta() e cancella dall'array quelle che hanno un numero di cambiamenti negativo o pari a 0. Successivamente, se la prima regola ha un numero di cambiamenti positivo, la elimina tra quelle possibili e la pusha nell'array delle regole valide(valideScon), ritornando true. Se nessuna regola risulta avere un numero positivo di cambiamenti, ritorna false;

- **RitaggaSconosciute()** : utilizza l'ultima regola trovata con il metodo EstraiPiuPromettenteSconosciuta() per ritaggare la variabile ritagPerBrill contenente i tag delle parole;
- **ValutaRegoleSconosciute()** : istanzia tutte le possibili regole per correggere i tag delle parole sconosciute e le salva nell'array regoleSconosciute;
- **TrovaRegoleSconosciute()** : trova un numero di regole valide fino al numero passato come parametro o fino a quando le regole hanno ancora un apporto positivo. Chiama in sequenza i metodi ValutaRegoleSconosciute(), TrovaRegolaMiglioreSconosciuta(), EstraiPiuPromettenteSconosciuta(), RitaggaSconosciute() e a ripetizione fino ad aver raggiunto il numero desiderato di regole per le parole sconosciute;
- **ValutaPossibilita()** : l'equivalente del metodo ValutaPossibilitaPerSconosciute() per le regole normali;
- **TrovaRegolaMigliore()** : l'equivalente di TrovaRegolaMiglioreSconosciuta() per le regole normali;
- **EstraiPiuPromettente()** : l'equivalente di EstraiPiuPromettenteScono- sciuta() per le regole normali;
- **Ritagga()** : l'equivalente di ritaggaSconosciute() per le regole normali, solo che ritagga tutto il file;
- **ValutaRegole()** : l'equivalente di ValutaRegoleSconosciute() per le regole normali;
- **TrovaRegole()** : l'equivalente di TrovaRegoleSconosciute() per le regole normali;
- **Addestra()** : addestra il sistema, trovando un numero di regole e regole sconosciute pari a quanto passato come parametro.

Quando viene invocato il metodo addestra, esso salva le regole trovate su file, in quanto il processo di addestramento risulta essere molto lungo.
Per addestrare il brill tagger usare il seguente comando da riga di codice: node addestra.js "nome Cartella" "nome da dare ai file" "nome del file di train" "percorso completo del dizionario" numero di regole da imparare numero di regole per le parole sconosciute da imparare. Ad esempio con la configurazione attuale, il comando: node addestra.js "train" "500" "train" "./morphSingoliTag.json" 10 5 , addestra utilizzando il file di train presente nella cartella train, arrivando ad imparare 5 regole per parole sconosciute e 10 regole generali, salvando i file nella cartella train.

2.5.2 File usaBrill.js

Il file `usaBrill.js` si occupa di applicare l'algoritmo di trasformazione per eseguire l'effettiva analisi grammaticale della frase. Il tagging iniziale della frase, avviene grazie al metodo `taggaFrase()` condiviso con il file `Addestra.js` ed è composto dai seguenti metodi:

- **ProvaRegole()** : richiede in input la frase sotto forma di stringa e i tag iniziali della frase in un array. Applica le regole in ordine di come sono state inserite nell'array regole a tutto il testo e restituisce i tag aggiornati;
- **ProvaRegoleSconosciute()** : come ProvaRegole(), solo che richiede anche la posizione degli elementi ai quali dover applicare le regole di trasformazione per le parole sconosciute;
- **ApplicaRegole()** : per prima cosa, applica il tagging iniziale alla frase con il metodo taggaFrase(), per poi chiamare i metodi provaRegoleSconosciute(), dove gli elementi da controllare sono le parole non presenti nel dizionario e successivamente il metodo provaRegole() da applicare a tutto il testo per ottenere il risultato finale.

Per utilizzare il tagger, sono necessari i file con le regole imparate dall'addestramento. Dal momento che il train richiederebbe troppo tempo con un numero valido di frasi, ho fornito nella cartella preTrained i file con il tagger addestrato su 5000 frasi, con 150 regole generali e 50 per le parole sconosciute.

Per utilizzare il tagger, usare il seguente comando: node usaBrill.js "nome cartella con i file contenenti le regole" "nome dato ai file, ad esempio 5k con le regole già addestrate" "percorso del file con il file di test sul quale usare il tagger". Ad esempio con la configurazione attuale: node usaBrill.js "preTrained" "5k" "./train/test.txt".
Altrimenti, è possibile usarlo con singole frasi, ad esempio: node usaBrill.js "nome cartella con i file contenenti le regole" "nome dato ai file, ad esempio 5k con le regole già addestrate" "questo è il progetto per intelligenza artificiale" "altra frase da taggare" "ulteriore frase da taggare". Con la configurazione attuale: node usaBrill.js "preTrained" "5k" "questo è il progetto per intelligenza artificiale" "altra frase da taggare" "ulteriore frase da taggare".

2.5.3 File utils.js

Il file `utils.js` contiene dei metodi di utilità utilizzati dagli altri file, come ad esempio il metodo per controllare se una parola sia un numero o meno o metodi per confrontare le regole.

2.6 Risultati

Il tagger è stato addestrato su una parte del file di train di Universal Dependencies, composto da 5000 frasi. Sono state trovate 50 regole per le parole sconosciute e 150 regole generali. Il tempo di addestramento è stato circa 1h30m per le regole sconosciute e circa 10h per le regole complessive. Il tagging iniziale risulta essere 88,42% sul file di test di Universal Dependencies. Dopo aver applicato le regole per le parole sconosciute trovate, raggiunge rispettivamente l'89,22%. Infine, applicando le regole generali, raggiunge un'accuratezza complessiva del 90.92%.
Il brill tagger, è uno dei primi tagger creati, e risulta essere uno dei meno efficienti. In particolare, al giorno d'oggi, dei tagger molto performanti sono quelli basati su hidden markov models oppure quelli che utilizzano reti neurali convoluzionali.

References

[Brill(1995)] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging, 1995. URL https://www.aclweb.org/anthology/J95-4004.

[Larsson and Norelius()] M. Larsson and M. Norelius. Part-of-speech tagging using the brill method. URL https://pdfs.semanticscholar.org/ae31/1cb0f1d821840d4309c9d33c0da0525f621d.pdf.

[Zanchetta(2004-2007)] M. B. E. Zanchetta. Morph-it! a free morphological lexicon for the italian language, 2004-2007. URL https://docs.sslmit.unibo.it/doku.php?id=resources:morph-it.