TaLTaC 3.0: un software multi-lessicale e uni-testuale ad architettura web [1]

Sergio Bolasco[1], Francesco Baiocchi[2] Alessio Canzonetti[3], Giovanni De Gasperis[4]

[1] SAPIENZA Università di Roma, [2] ISTAT Istituto Nazionale di Statistica,

[3] Consorzio MIPA, [4] Università dell'Aquila

Abstract

In the present paper, we introduce TaLTaC version 3.0. For this updated version, the software's architecture has been entirely restructured, and the code completely rewritten. The main aims of these changes were *i*) to enable TaLTaC to handle large textual corpora and *ii*) to introduce a web-based (HTML5/CSS/Javascript) user interface. The backend code has been ported to the Python/PyPy language, and the implementation of a NoSQL approach for the data persistence layer allowed for improved performance. In particular, advanced automatic memory caching services through SSD memory devices are exploited, and results are obtained that are comparable to those achieved by using only the computer's main memory. TaLTaC 3.0 runs on Windows, Linux and Os X, without any dependency from proprietary software technologies. The graphical user interface (GUI) has been redesigned and reserved a software sub-system decoupled from the inner processing core (CORE). CORE extracts a 3-layered lexical vocabulary from each given corpus, taking advantage of the parallel operations that can be carried out by current personal computers. The three lexical layers are *i*) basic parsing output with extraction of original graphic forms, *ii*) pre-normalization, and *iii*) named entities and multi-words recognition; all lexical layers are based on one and the same textual parsing. The paper also presents and compares the actual performances that can be obtained on different operating systems/machines and working with different data set sizes.

Riassunto

La versione 3 di TaLTaC, qui presentata, include una revisione globale dell'architettura software e una riscrittura totale del codice, con l'obiettivo di elaborare grandi corpora testuali e introdurre una interfaccia utente basata sul web (HTML5/CSS/Javascript). Il codice di calcolo di base è scritto in Python e nei suoi derivati ad alte prestazioni, come PyPy. Inoltre l'adozione di un approccio NoSQL per lo strato di persistenza dei dati, ha permesso di ottenere prestazioni paragonabili all'uso della sola memoria centrale, potendo sfruttare tecniche avanzate automatiche di memoria cache con l'uso dei dispositivi allo stato solido (SSD). TaLTaC 3.0 può girare su Windows, Linux e Os X, non presentando dipendenze da tecnologie software proprietarie. L'interfaccia grafica (GUI) è stata riprogettata, implementando un sottosistema disaccoppiato dal nucleo centrale di elaborazione (CORE). Il CORE estrae simultaneamente dal corpus un vocabolario a 3 livelli lessicali, sfruttando il parallelismo disponibile negli attuali computer: scansione con estrazione delle forme grafiche originarie, pre-normalizzazione e riconoscimento di entità e multiwords predefinite. Questi tre parsing lessicali insistono tutti su un'unica scansione testuale. Nel lavoro vengono confrontati i tempi di elaborazione ottenibili nei diversi sistemi operativi per corpora di vari tipi e dimensioni.

Key words : software engineering, lexical analysis, JSON, NoSQL, high performance computing.

1. Introduzione

La nuova versione del software Taltac qui illustrata rappresenta una significativa evoluzione della idea iniziale concepita nel 1999 e sviluppata in due tempi: nel 2000 Taltac (versioni

[1] Il paper è frutto del lavoro comune degli Autori e scritto rispettivamente da: S. Bolasco (par. 1, 3 e 5), G. De Gasperis (par. 2 e 2.1), F. Baiocchi (par. 2.2), A. Canzonetti (par. 4).

1.##) centrato sull'analisi di tipo lessicale e nel 2005 TaLTaC2 (versioni 2.##) orientato all'analisi di tipo testuale[2]. La prima offre strumenti sia per l'annotazione grammaticale e semantica del vocabolario di un corpus sia per l'individuazione di sottoinsiemi di keyword (Bolasco, 2013) al fine di applicare tecniche statistiche multidimensionali per una visione d'insieme del "senso" presente nel corpus. La seconda utilizza tecniche di estrazione d'informazione basate su grammatiche locali (Silberztein, 1993; Poibeau, 2011, p. 66-67) per l'individuazione di entità di interesse mediante espressioni regolari. Questo tipo di analisi è spesso finalizzata a classificazioni e categorizzazioni supervisionate dei documenti.

Dal 2010, anno dell'ultimo aggiornamento di TaLTaC2 nella versione 2.10 (Bolasco, 2010b), Taltac ha vissuto un lungo periodo di incubazione nel progetto di rinnovamento tecnologico complessivo dell'architettura, mirato al potenziamento delle capacità di elaborazione, ottenendo tempi di calcolo ragionevoli su corpora di grandi dimensioni (*big data*) e all'adozione di linguaggi e ambienti di sviluppo propri del mondo web. In questi ultimi cinque anni, anche altri software di analisi di dati testuali hanno rinnovato la loro architettura tecnologica[3].

Fra le caratteristiche di Taltac, consolidate in vari studi e ricerche (Bolasco, 2013), vanno evidenziate l'utilizzo di risorse esterne, la possibilità di disporre liberamente di una varietà di attrezzi di base per l'analisi automatica dei testi, le sinergie tra momenti di studio lessicale e di indagine testuale molto utili per costruire procedure riproducibili nelle soluzioni di text mining. D'altro canto i limiti di questo approccio si riscontrano in alcune procedure gerarchizzate, nella lessicalizzazione fisica delle multiwords, nella impossibilità di procedere nella disambiguazione grammaticale a partire dalle concordanze, e nella dimensione massima dei corpora analizzabili (files non superiori a 150MB, o con non più di 100.000 documenti). Per superare questi limiti siamo stati spinti a produrre un salto di qualità nella struttura tecnologica.

2. Architettura software

L'architettura della nuova versione di TaLTaC 3.0 (nel seguito T3.0) si ispira alla struttura *multi-tier* (Allier et al., 2015). Offre complessi strumenti di analisi dei dati testuali, mantenendo prestazioni elevate e offrendo una interfaccia familiare all'utente del web. La base di codice è stata interamente riscritta in linguaggio Python/PyPy[4], riferimento utilizzato per l'elaborazione dei testi e del linguaggio naturale da parte di un'ampia comunità di sviluppatori.

In particolare sono ottenibili alte prestazioni dalle strutture dati aggregate e indicizzate del Python, su una macchina a 64bit di parallelismo, che permettono di operare efficacemente macro operazioni su grosse moli di dati. La dimensione dei corpora analizzati è teoricamente limitata dalle sole capacità della macchina su cui viene installato e utilizzato il software, incluso il sistema operativo. Le massime prestazioni si ottengono nei sistemi dove la versione Python/PyPy è disponibile a 64bit (Linux e OS X), tramite i quali è stato possibile effettuare

[2] www.taltac.it (link visitato il 30/03/2016).

[3] Limitandoci all'ambito della comunità JADT, sono da citare software quali TXM (Heiden: http://textometrie.ens-lyon.fr/?lang=fr), Hyperbase (Brunet: http://ancilla.unice.fr/; http://bcl.cnrs.fr/rubrique38), IRAMUTEQ (Ratinaud: http://www.iramuteq.org/). (link visitati il 30/03/2016).

[4] http://www.pypy.org (link visitato il 30/03/2016).

oggi ad esempio un parsing a 3 livelli su corpora di estensione maggiore di 1.5GB in pochi minuti, come si può vedere in tabella 1.

Inoltre si è scelto di utilizzare una soluzione di persistenza della memoria a lungo termine nella modalità NoSQL *key/value data store*, come quella fornita da un server Redis[5], e tecniche di mappatura diretta in RAM dei file testo con tempi di accesso limitati soltanto dalla tecnologia disponibile per la memoria secondaria, potendo sfruttare al meglio ad esempio la presenza di unità disco allo stato solido (SSD).

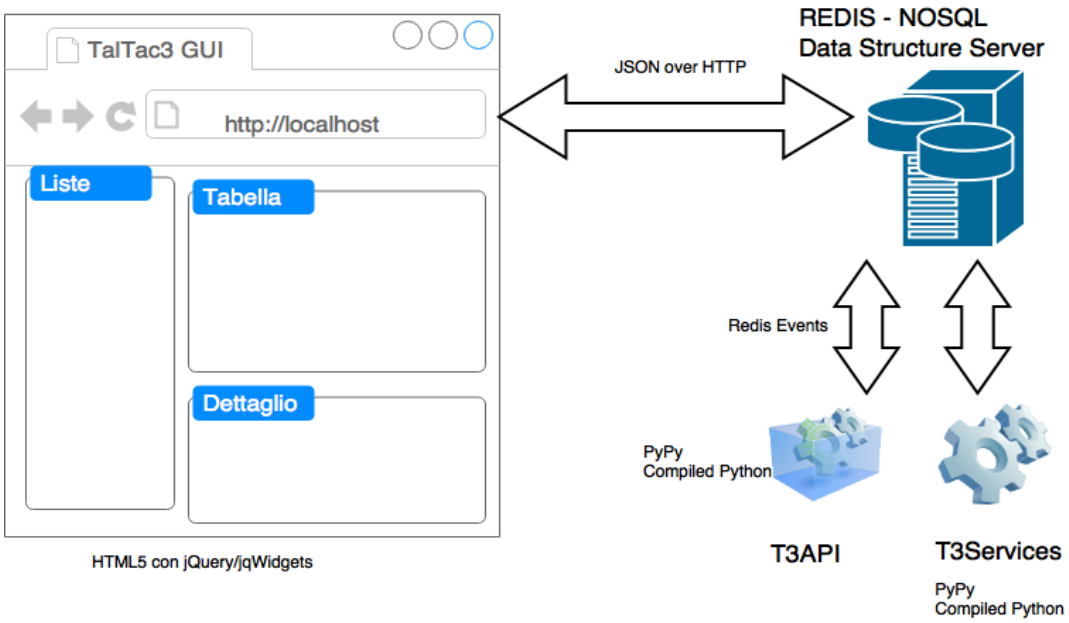
La scelta del linguaggio Python ha permesso inoltre di ottenere una applicazione multipiattaforma (Windows, Linux, OS X), accompagnata da una interfaccia utente basata sugli standard del web (HTML5/Javascript) che astrae totalmente dai vincoli imposti da ciascun sistema operativo.

2.1. Infrastruttura dei processi componenti

L'architettura di T3.0 è suddivisa tra interfaccia utente (nel seguito GUI: Graphical User Interface) e nucleo di calcolo (CORE), disaccoppiati in modalità completamente asincrona, comunicanti fra di loro tramite protocollo HTTP. Il nucleo di calcolo è in grado di elaborare i dati testuali in modalità multi-processo e quindi di sfruttare i multi-core della macchina ospitante, qualora siano presenti.

Come illustrato nella figura 1, T3.0 non è una applicazione software monolitica, ma è stata scomposta in un insieme di processi di elaborazione che cooperano tra loro in modalità del tutto asincrona, sulla base di eventi originati dall'interfaccia utente. In questo modo, l'applicazione può rispondere a diverse modalità di installazione: singolo desktop oppure client/server, con l'unico vincolo per la parte client, ove risiede la GUI, di poter utilizzare un visualizzatore compatibile HTML5, che può essere un web browser, oppure una applicazione desktop con browser integrato.

Figura 1 – L'architettura di TaLTaC 3.0: GUI e CORE



[5] http://redis.io (link visitato il 30/03/2016).

Questo tipo di realizzazione dell'interfaccia verso l'utente, libera totalmente T3.0 dai vincoli imposti dai diversi sistemi operativi. Analogamente, nella parte server, i diversi processi che compongono il CORE, sono stati sviluppati in sintassi Python 2.7.x e possono essere configurati per sistemi ad alte prestazioni basati sul compilatore in tempo reale PyPy.

I processi principali di questa parte server CORE sono: C1- server di persistenza NoSQL Redis *key/value data store*; C2- server di smistamento delle chiamate asincrone T3API (Application Programming Interface); C3- server di calcolo in background degli strati lessicali/semantici T3Services; C4- server del canale REST; C5- server di distribuzione files HTTP.

L'insieme dei processi di base della parte CORE, C2 e C3, in aggiunta al processo GUI ospitato nel visualizzatore, comunicano tra loro tramite il meccanismo asincrono PUBLISH/SUBSCRIBE offerto dal processo server C1, il quale a sua volta esegue comandi diretti provenienti dalla GUI di lettura e scrittura nella memoria delle chiavi. Tutti i messaggi da e verso la GUI sono standardizzati in formato JSON, trasmessi e ricevuti tramite un canale HTTP su una porta di comunicazione dedicata, il cui intermediario è realizzato tramite il processo C4 implementato tramite PyRESTRedis, pacchetto rilasciato in open source da membri del gruppo di sviluppo su GitHub[6].

Gli strati lessicali di T3.0 sono implementati sul server persistente, tramite chiavi *hash* con campi, ciascuno contenente i dati in formato JSON, direttamente utilizzabili tramite serializzazione dai processi CORE e GUI. In questo modo quando l'utente fa una richiesta dalla GUI, come ad esempio avviare un algoritmo di parsing su un corpus molto esteso, un unico comando asincrono viene inviato tramite il canale REST al CORE, il quale a sua volta essendo in ascolto di comandi da eseguire, avvia i processi interni per ottenere il risultato; il calcolo può impiegare anche qualche minuto, a seconda dell'estensione dei dati da analizzare (cfr. paragrafo 6). In questo modo l'utente può consultare i dati già presenti nel sistema navigando le diverse sessioni aperte, e/o altri stati lessicali presenti sugli stessi dati calcolati in precedenza, senza perdere il controllo della GUI. Quando il risultato è reso disponibile dal CORE, la GUI viene notificata e l'utente può visualizzarlo.

Anche tutte le opzioni scelte dall'utente, come selezione dell'alfabeto, opzioni di parsing, e altro, vengono memorizzate dal CORE in forma JSON e sono gestite allo stesso modo dei dati di calcolo.

Le chiamate al sistema CORE sono state implementate introducendo un meccanismo di RPC , Remote Procedure Call, basato sul canale REST e il comando PUBLISH del server di persistenza sul canale appositamente dedicato. In questo modo i processi CORE che sono sottoscritti a quello stesso canale ricevono il messaggio di un nuovo comando da eseguire e a seconda della disponibilità a eseguirlo, in funzione di avere o meno la propria porzione di CPU già occupata a fare altro. Quando il dato è disponibile a fine calcolo, viene salvato in formato JSON in chiavi persistenti che la GUI può andare a leggere in istanti successivi. In questo modo avendo un numero di processi CORE di “smaltimento” della coda comandi pari ad almeno il numero dei core fisici (o virtuali) della macchina a disposizione, si possono ottenere sostanziali miglioramenti delle prestazioni rispetto a una ipotetica versione del software monolitica e senza una infrastruttura di chiamata differita. Inoltre in questo modo, per sua natura, il CORE si presta a poter essere distribuito su un cluster di macchine connesse

[6] Reperibile su http://github.com/AAAI-DISIM-UnivAQ/PyRESTRedis (link visitato il 30/03/2016).

da una rete locale ad alta velocità su fibra ottica, potendo costruire sistemi in grado di analizzare moli di dati tipiche del mondo "*big data*" (decine di Gigabytes).

2.2. Frontend

Come il *backend*, anche il *frontend* di T3.0 è sviluppato per essere utilizzato senza limitazioni o differenze sui principali sistemi operativi esistenti (Windows, Linux, OS X). Per ottenere tale obiettivo si è operata una scelta consolidata nell'ambito dello sviluppo applicativo, ovvero uno sviluppo basato solo su Javascript e CSS, con l'ausilio di alcune fra le più conosciute librerie Javascript attualmente esistenti: *i)* jQuery, libreria generalizzata per manipolare il DOM della pagina HTML, gestire gli eventi, eseguire chiamate Ajax; *ii)* jQWidgets, framework per la costruzione di interfacce utente; *iii)* Knockout, libreria che permette di utilizzare nell'ambito di jQWidgets il modello MVVM (*Model-View-View Model*): in questo approccio al problema dello sviluppo di una GUI, alla classica View (l'insieme dei componenti visuali che l'utente vede e con i quali può interagire) viene associato un View Model che ne rappresenta un'astrazione e sarà quest'ultimo componente a interagire con il Model (la rappresentazione dei dati provenienti dal *backend*), gestendo gli eventi che provocano cambiamenti da/verso la View.

Come già descritto nel paragrafo precedente la comunicazione tra il *frontend* e il *backend* di T3.0 avviene scrivendo e leggendo, da ambo i lati, nel server di persistenza Redis.

Mentre il *backend*, essendo scritto in Python, è nativamente fornito degli strumenti necessari per connettersi al server Redis, il *frontend*, sviluppato in Javascript ed elaborato direttamente da un browser, non ha nativamente a disposizione alcuno strumento per connettersi a Redis. In un primo momento si è risolto questo problema grazie all'utilizzo di un componente di terze parti (Flask[7]) che espone un server REST e permette l'accesso in lettura/scrittura a Redis con semplici chiamate su protocollo HTTP che sono eseguibili senza problemi da qualunque browser (server C4 citato nel paragrafo precedente).

Per comprendere il come e il perché sia stato poi superato questo approccio iniziale basato su Flask, occorre sottolineare che un processo Javascript eseguito all'interno di un browser è sottoposto, per motivi di sicurezza, ad alcune limitazioni: la più grave delle quali consiste nell'impossibilità di accedere liberamente al *file system* della macchina, su cui sta girando il browser che ospita la GUI di T3.0.

Il *perché* è dovuto al fatto che per un'applicazione come T3.0 è fondamentale poter accedere alle risorse dell'utente presenti sulla macchina. E' quindi un aspetto critico non poter fornire all'utente *widget* di facile utilizzo (come una finestra di dialogo per sfogliare cartelle) per utilizzare le proprie risorse senza costringerlo a compiere operazioni scomode o a posizionare tutti i suoi file in una cartella arbitrariamente decisa dall'applicazione e non da lui.

Per realizzare il *come* si è introdotto nell'architettura di T3.0 un altro componente Javascript: *Node.js*. Con l'adozione di Node.js di fatto è cambiato il paradigma di fruizione dell'applicazione: mentre in precedenza era il motore Javascript del browser dell'utente a elaborare i contenuti della GUI, ora è il motore Javascript contenuto nel framework Node.js a elaborarli e a passarli al browser. Questo fatto permette di superare i limiti imposti (per la sicurezza) all'esecuzione di codice Javascript all'interno del browser e quindi ora è possibile fare due cose (prima impossibili): *i)* accedere liberamente al *file system* della macchina sulla

[7] http://flask.pocoo.org/ (link visitato il 30/03/2016).

quale si sta utilizzando T3.0; *ii)* interagire direttamente con il server Redis, tramite l'utilizzo di un *add-on* di Node.js.

L'introduzione di Node.js ha suggerito un ultimo passaggio relativo alla distribuzione dell'applicazione: l'utilizzo di Electron[8]. Quest'ultimo non è altro che un framework Node.js fornito di un browser (*Chromium*) adattato *ad hoc* al contesto in cui opera. Questa strategia permette di incapsulare tutto il *frontend* e tutto il *backend* in un unico pacchetto, mascherando all'utente il fatto che nell'ambito di T3.0 vengono lanciati più processi e rendendone più facile e consistente la gestione rispetto al sistema operativo.[9]

3. Capacità computazionali

In questa nuova edizione di Taltac è possibile elaborare l'informazione su differenti strati (layers) di vocabolario. Fin dal primo processo di tokenizzazione (*parsing*) T3.0 genera tre strati lessicali. Si tratta rispettivamente del "vocabolario base" (parsing grezzo per tokens semplici), del "vocabolario L0" (parsing con pre-normalizzazione per riconoscere e uniformare parole accentate e/o scritte in maiuscolo a inizio frase), del "vocabolario L1" (parsing sulla base di risorse predefinite di nomi propri ed entità complesse (*multiwords*), che identificano toponimi, terminologia tecnico-scientifica o espressioni idiomatiche). La pluralità di strati di vocabolario consente di procedere nell'analisi a differenti livelli, in modo non necessariamente "gerarchico", utilizzando di volta in volta l'informazione più appropriata.

Infatti il primo livello, detto base, ha il carattere dell'analisi documentaria, in quanto permette di mantenere intatta la collezione dei testi, così come questi risultano a livello della raccolta; pertanto il corpus è analizzabile tal quale è allo stato di documentazione. Conserva la grafia Alto/basso delle parole, non corregge spazi o eventuali errori. La tokenizzazione del corpus è realizzata basandosi solo sull'insieme dei caratteri delimitatori pre-scelti per separare le parole del testo.

Il secondo livello, corrispondente al vocabolario L0 (layer zero), produce la lista dei types corretti negli accenti (trasforma ad esempio *liberta'* in *libertà*) e nelle maiuscole a inizio frase (*Il* in *il*, *La* in *la* ecc.). Questo livello ottimizza il riconoscimento di parole del linguaggio naturale, eliminando varianti di forme grafiche non significative e cumulando le rispettive occorrenze: ciò consente buoni confronti con risorse linguistiche esterne, quali i dizionari di lingua e i lessici di frequenza.

Il terzo livello, corrispondente allo strato L1 (layer uno), produce un vocabolario "capito" i cui elementi sono unità di senso più vicine al significato reale dei contenuti del testo. Ciò avviene con il riconoscimento di "*named entities*" come i nomi propri di persona, celebrità, società, luoghi o altre entità, espressi sia con parole singole (*Mario*) sia con "multiwords" (*Anna Maria*), o il riconoscimento di locuzioni di vario tipo grammaticale: preposizionali (*al fine di*), avverbiali (*in parte*), aggettivali (*di carta*), nominali (*dato di fatto*, *impronte digitali*), verbali (*andare a male*). Tali riconoscimenti sono possibili a partire da liste predefinite disponibili come risorse esterne in processi di normalizzazione del corpus o mediante

[8] http://electron.atom.io/ (link visitato il 30/03/2016)

[9] Poiché anche Electron è disponibile per tutti i principali sistemi operativi, con la sua adozione il ciclo di rilascio di T3.0 viene drasticamente semplificato e sarà possibile pubblicare le nuove versioni sempre in simultanea per tutti i sistemi operativi.

algoritmi specifici come quello dei verbi frasali (Bolasco, 2010a), facilitando le operazioni di tagging grammaticale e lemmatizzazione in successivi step di analisi.

Esaminiamo alcune analisi applicabili ai primi tre strati di vocabolari. Ad esempio, l'estrazione del linguaggio peculiare - che si effettua operando il confronto del vocabolario del corpus con un lessico di frequenza, costruito generalmente sulla base di forme flesse semplici - può essere svolta sui primi due strati di vocabolario (parsing grezzo o parsing di pre-normalizzazione), comunque prima del riconoscimento di multiwords. Al contrario, la selezione di keywords caratteristiche o specifiche di una parte, oppure rilevanti per catturare un documento (pesate mediante l'indice *tf-idf*), è più informativa circa il reale significato delle unità lessicali, se effettuata a livello di unità di senso riconosciute, quindi sulla base dello strato L1 o successivi del vocabolario. La pluralità di strati libera la ricerca da fasi di studio gerarchizzate, in quanto i suddetti esempi di estrazione di keyword sono operabili in qualsiasi ordine temporale fra loro.

Un quarto livello lessicale, definibile come strato L2, prende in considerazione gli aspetti propriamente linguistici di ogni type (quindi le annotazioni della categoria grammaticale e del lemma), mentre un quinto livello (strato L3) considera le dimensioni semantiche dei type. Entrambi i livelli introducono un cambiamento radicale della unità lessicale, poiché "mettono in discussione" l'unicità del type in quanto forma grafica nel vocabolario. Una forma come *italiano* può occorrere alcune volte come aggettivo e altre come sostantivo; oppure una forma come *fine* può indicare in talune occorrenze del corpus il significato di "termine" e altre quello di "scopo". Pertanto in questi strati, per conservare l'unicità della unità lessicale occorre passare a considerare non semplicemente il type ma delle *t-uple* di informazioni. Nel caso linguistico l'insieme delle informazioni espresse dalla tripla {flessione, categoria grammaticale, lemma}, nel caso semantico quelle espresse dalla coppia {flessione, categoria semantica/diverso significato}. Si può verificare anche un progressivo accumulo di riconoscimenti semantici (del tipo L3a, L3b, L3c) tesi a definire successivamente unità "atomiche" diverse formanti, mediante operazioni di lessicalizzazione, strutture via via più complesse ("molecole"). Si può trovare un esempio di questo tipo in una ricerca sul linguaggio della critica gastronomica, che misura l'innovazione *vs* tradizione di un ristorante passando dall'analisi dei cibi a quella dei piatti del menu (Bolasco, 2013, p. 352-356).

4. Relazione fra l'analisi lessicale multi-livello e l'analisi testuale uni-livello

In questo quadro, di analisi lessicali multilivello non gerarchizzate, le analisi di tipo testuale insistono sempre sulla prima tokenizzazione, mantenendo di fatto unico il livello di studio testuale. Ognuno dei vari strati di vocabolario rappresenta, infatti, di volta in volta un differente indice di un medesimo e unico corpus testuale. Vale a dire che le informazioni principali sulle singole occorrenze (token) - type e posizione nel testo - vengono raccolte solo in fase di primo parsing del testo e successivamente non vengono più modificate. Le trasformazioni che i vari types subiscono nei passaggi tra i vari strati (ad esempio la trasformazione di *liberta'* in *libertà*) avvengono solo al livello della lista del vocabolario, e non producono una modifica fisica del testo, ovvero sono delle trasformazioni virtuali.

Questa scelta ha origine dal fatto che, volendo mantenere attivi, sempre e comunque, tutti i vari strati di analisi, una trasformazione fisica del testo imporrebbe la duplicazione del corpus per ogni strato, cosa che porterebbe a due conseguenze non desiderabili. La prima è che si avrebbero evidenti ripercussioni sullo spazio occupato dai dati, cosa che rappresenterebbe un grave inconveniente, considerando che T3.0 consente di lavorare con corpus di vaste

dimensioni, anche di alcuni Gigabytes. La seconda è che la duplicazione del corpus per ogni strato, di fatto, porta a una situazione simile alla creazione di una nuova sessione di lavoro, parallela a quella/e già esistente/i, cosa in fondo già possibile. Mantenere un unico corpus, ovvero un unico set di informazioni testuali, consente peraltro una maggiore "coordinazione" tra i diversi strati: tutte le operazioni effettuate su uno strato (si pensi all'attribuzione di una etichetta a un set di forme) possono essere riportate anche sugli altri e da qui potranno essere utilizzate anche a livello testuale, data l'unicità del corpus, mentre con la duplicazione delle informazioni testuali si avrebbero degli strati a "compartimenti stagni".

Occorre gestire una serie di situazioni affinché un vocabolario "dinamico" (trasformato dal primo parsing di base negli strati successivi da L0 a L3) possa correttamente "dialogare" con un livello testuale che invece è "congelato" alla forma originaria. Per il primo strato, che è effettivamente il vocabolario autentico del livello testuale (nel senso di riprodurre il carattere *documentale* del testo), questo dialogo è diretto: il vocabolario in questo caso contiene le forme così come sono presenti nel testo (nel software è chiamato "vocabolario base"). Pertanto, quando si vogliono ottenere le concordanze della parola *liberta'*, le singole occorrenze di questa forma verranno cercate e ovviamente trovate nel testo, perché è proprio in questa forma grafica (nel seguito fg) che vi si trovano scritte.

Passando agli strati successivi, questo non è più scontato. In L0, verosimilmente, la parola *liberta'* non esisterà più, ma esisterà la sua forma trasformata (per via della pre-normalizzazione) *libertà*. In questo strato, l'utente non può più vedere l'aspetto originario di questa parola, per cui un'eventuale ricerca delle concordanze di *libertà* produrrebbe un risultato vuoto o parziale, poiché non esiste una tale parola nel testo oppure perché non verrebbero recuperate tutte le sue occorrenze[10]. Il dialogo tra il vocabolario dello strato L0 ed il livello testuale deve essere quindi mediato da una mappatura delle forme di L0 sulle forme del vocabolario originario: occorre, cioè, registrare la corrispondenza *libertà* (L0) = *liberta'* (vocabolario base). Questo metodo consente di operare - in maniera del tutto trasparente all'utente - la ricerca di *liberta'* anche quando egli sta lavorando con la parola *libertà*. Non solo: T3.0 permette anche di creare i vocabolari degli strati da L0 in poi non come effettiva duplicazione del vocabolario dello strato precedente, ma solo come registrazione delle modifiche e delle mappature che intervengono nel passaggio fra uno strato e l'altro, con evidenti vantaggi nello spazio occupato in memoria e nella "coordinazione" tra i diversi strati di analisi.

Quella appena accennata nell'esempio - modifica della grafia di una forma - è, tuttavia, la casistica di più semplice risoluzione tra tante possibili. I casi di trasformazione di un vocabolario possono ricadere nelle seguenti altre tipologie:

i) una fg cambia grafia in un'altra non già esistente (è il caso appena visto nell'esempio). Nel vocabolario della struttura dello strato successivo, occorrerà mappare la nuova forma sulla vecchia. Tutte le singole occorrenze potranno essere recuperate nel corpus tramite questa mappatura, che di fatto concretizza il concetto:

libertà (L0) = *liberta'* (vocabolario base) e viceversa;

ii) una fg cambia grafia in un'altra già esistente, ad esempio una fg con errore ortografico o con grafia "non standard" viene corretta riportandola alla giusta forma già presente nel corpus. Nella struttura dello strato successivo, la forma deve accogliere, nel vocabolario, le

[10] Nel testo potrebbero esserci alcune occorrenze della parola con l'accento e altre con l'apostrofo.

occorrenze della forma che dovrà sparire, e rendere conto di quale forma ha dato luogo a questa "aggiunta". Quest'ultima informazione permette di mappare le occorrenze della vecchia forma nel testo e assimilarle a quelle della esistente. La mappatura in questo caso concretizza il concetto:

libertà (L0) = *libertà* (vocabolario base) + *liberta'* (vocabolario base) e viceversa.

In questo caso, particolare attenzione dovrà essere rivolta ai calcoli che coinvolgono il numero di unità lessicali (types).

iii) disambiguazione (grammaticale o semantica), in cui bisogna distinguere due casi:

a) una fg viene disambiguata totalmente, ovvero la disambiguazione coinvolge tutte le sue occorrenze. In tal caso nello strato successivo si generano due (o più) fg che "esauriscono" le occorrenze della forma di origine, che quindi cessa di esistere. La mappatura realizza il concetto:

governo[N](L0) + *governo[V]*(L0) = *governo[J]* (vocabolario base) e viceversa;

b) una fg viene disambiguata parzialmente, ovvero viene disambiguata solo una parte delle sue occorrenze, con le altre che rimangono ambigue e continuano ad appartenere alla forma di origine, che quindi non cessa di esistere. Nella struttura dello strato successivo, verrà aggiunta una nuova entrata nel vocabolario e dovranno essere aggiornate le informazioni sulle posizioni del testo della nuova entrata aggiunta. Le occorrenze di questa nuova entrata dovranno essere detratte da quelle della forma di origine. La mappatura realizza il concetto:

governo[J](L0) + *governo[V]*(L0) = *governo[J]* (vocabolario base) e viceversa;

iv) lessicalizzazione (creazione di una *multiword*). I poliformi rinvenuti nel corpus, verosimilmente sulla base di una lista, vengono inseriti nel vocabolario dello strato di pertinenza, con il conteggio delle rispettive occorrenze. A livello testuale, questa operazione viene realizzata semplicemente "reindirizzando" le opportune occorrenze dei types componenti la multiword alla forma della multiword stessa. In questo caso, la mappatura concretizza il concetto:

carta di credito (L1) = *carta* (L0)[*x, y, z*] *di* (L0)[*a, b, c*] *credito* (L0)[*s, t, u*],

dove con *carta* (L0)[*x, y, z*], *di* (L0)[*a, b, c*] e *credito* (L0)[*s, t, u*] si indicano solo quelle occorrenze (in questo caso 3) delle tre forme coinvolte dalla lessicalizzazione. Chiaramente, queste occorrenze cesseranno di far parte delle rispettive forme (semplici) di origine.

5. Prestazioni

In primo luogo i limiti dimensionali di elaborazione con TaLTaC2 - file di 150 MB e/o corpus con max 100.000 documenti - sono ampiamente superati. Finora, con T3.0 sono stati elaborati, su PC con 16GB di RAM, files fino a 3 GB (equivalenti a corpus con oltre mezzo miliardo di tokens), visualizzando interamente nell'interfaccia utente vocabolari di oltre 2 milioni di types diversi e quattrocentomila documenti. Non vi sono motivi per pensare che tali dimensionalità non possano essere superate con macchine di prestazioni adeguate alle necessità.

In secondo luogo si riscontra che la maggiore velocità di T3.0 rispetto alla ultima versione di T2.10 è dell'ordine di 300/400%, senza contare la maggior complessità del triplo parsing di T3.0 (vocabolario base e vocabolari degli strati L0 e L1) rispetto all'unico parsing di TaLTaC2.

In tabella 1 si riportano alcuni risultati sulle prestazioni di T3.0 sotto i tre principali sistemi operativi Windows, Linux, OS X misurate su computer equivalenti, in corpora confrontabili. Dai tempi di calcolo si possono riscontrare i seguenti ordini di grandezza di elaborazione: per un file di circa 40MB, pari a corpus di 7-8 milioni di occorrenze, T3.0 impiega fra i 16 e i 35 secondi per eseguire il parsing dei tre strati; mentre per un file 10 volte maggiore (400MB, 74 milioni di tokens) impiega fra i 200 e i 270 secondi, ovvero fra poco più di 3 minuti e 4 minuti e mezzo, a seconda delle macchine utilizzate.

Tabella 1 – Risultati di elaborazioni in differenti corpus con i tre sistemi operativi

Id corpus	Nome Corpus	File size	Num framm	Misure del parsing base		Secondi di elaborazione		
				types (V)	tokens (N)	Wind (1)	Linux (2)	Os X (3)
1	LeMonde 100 gruppi di articoli	39 MB	100	199.640	8.039.078	17,3	25,1	16,3
2	Repubblica 10 mila articoli		10.000	166.107	7.376.303	35,0	25,4	21,6
3	Repubblica 20 mila articoli	78 MB	20.000	232.440	14.783.665	54,1	47,9	40,5
4	LeMond100 + Rep 10M (**1+2**)		10.100	349.184	15.394.935	42,2	50,1	38,8
5	Repubblica anno 1992 (*)	155 MB	37.440	324.838	28.889.331	135,2	93,9	78,2
6	Repubblica 100 mila articoli	393 MB	100.000	531.357	74.183.669	270,4	242,3	202,1
7	Repubblica 150 mila articoli	564 MB	150.000	653.612	106.417.935	406,0	355,2	284,0
8	Repubblica 10 anni (*Rep90*)	1,51 GB	400.276	1.167.723	278.940.753	*n.d.*	986,0	844,0
9	*Rep90* + Stampa Francese	2,81 GB	402.175	2.227.191	526.316.751	*n.d.*	2.573,2	1.522,2

Legenda

(1) Windows7 , Intel Core i7, 2.6 Ghz, 16GB RAM, SSD 512GB (Toshiba Z30)

(2) Linux Mint MATE 64 8(Rose), Intel Core Xeon, 2.93 Ghz, 24Gb RAM, HDD 1TB (Dell 5500)

(3) Mac Os X10.5, Intel Core i7, 2.7Ghz, 16GB RAM, SSD 512GB (Mac Pro)

(*) corpus con 3 sezioni

In TaLTaC2 i tempi di calcolo -a parità di N (ampiezza del corpus)- dipendono dalla dimensione degli elementi della collezione: cento grossi gruppi di articoli (corpus 1, LM100 in tabella 1) contro diecimila singoli articoli di stampa (corpus 2, Rep10mila) -100 volte di più- richiedono per il parsing il 45% di tempo in più. Al contrario, in T3 l'elaborazione degli stessi due richiede il 50% di tempo in meno, per cui un minor numero di documenti a prescindere dalla loro grandezza richiede minor tempo di tokenizzazione. Dalla tabella 1 si evince anche una certa linearità nei tempi di elaborazione, al crescere delle dimensioni dei files. Infatti per un file di 1,5 GB relativo a un corpus di riferimento di 278,9 milioni di tokens, utilizzato per costruire il dizionario di frequenza *Rep90*[11], il triplo parsing di 400mila articoli ha richiesto fra i 14 e i 15 minuti a seconda del sistema operativo; mentre per un file di 2,8 GB contenente un corpus bilingue di articoli di giornale, in italiano e francese[12], per un ammontare di quasi mezzo miliardo di tokens e oltre 2 milioni di types, T3.0 ha impiegato meno di mezzora sul PC più performante delle nostre prove.

Questi risultati confermano in particolare che la sostanziale linearità dei tempi di elaborazione di T3.0, al variare della dimensione del corpus, è indipendente dalla sua struttura: pochi frammenti grandi *vs* molti frammenti piccoli; oppure corpus monolingue *vs* multilingue. Questa circostanza non si registrava nella precedente versione di Taltac.

[11] Bolasco, 2013, p. 256, 262-264.

[12] La parte in italiano corrisponde al corpus di Rep90, la parte in francese corrisponde a una raccolta

6. Riferimenti

Allier S., Barais O., Baudry B., Bourcier J., Daubert E., Fleurey F., Monperrus M., Song H., Tricoire M. (2015). Multi-tier Diversification in Web-Based Software Applications. *IEEE Software*, vol.(32.1): 83-90.

Bolasco S. (2010a). Il riconoscimento automatico di locuzioni verbali con l'ausilio del software Taltac2. *Rassegna italiana di Linguistica Applicata*, XLI, vol.(1-2): 39-56.

Bolasco S. (2010b). *Taltac2.10: Sviluppi, esperienze ed elementi essenziali di analisi automatica dei testi.* LED, Milano.

Bolasco S. (2013). *L'analisi automatica dei testi. Fare ricerca con il text mining*. Carocci Ed., Roma.

Heiden S., Magué J. P. and Pincemin B. (2010). TXM: Une plateforme logicielle open-source pour la textométrie - conception et développement. In Bolasco S., Chiari I. and Giuliano L. editors, *Proc. of 10th International Conference on the Statistical Analysis of Textual Data - JADT 2010* (Vol. 2: pages 1021-1032). Edizioni LED, Milano.

Poibeau T. (2011). *Traitement automatique du contenu textuel*. Lavoisier, Paris.

Silberztein M. (1993), *Dictionnaires électroniques et analyse automatique de textes. Le système INTEX*. Masson, Paris.