

Corso di Laurea in Informatica Umanistica

Tesi di Laurea

Costruzione di un corpus specialistico per analisi sul dominio linguistico amministrativo

Candidato: *Marco Durante*

Relatore: *Alessandro Lenci*

Correlatore: *Maria Simi*

Anno Accademico 2012-2013

# **INDICE**

[Introduzione 2](#_Toc372819830)

1. [IL WEB COME CORPUS 3](#_Toc372819831)

[1.1 Cosa sono i corpora 3](#_Toc372819832)

[1.2 Perché usare il web come corpus 4](#_Toc372819833)

[1.2.1BootCat 8](#_Toc372819834)

2. [Costruire PAWaC 9](#_Toc372819835)

[2.1 Corpora d’addestramento 9](#_Toc372819836)

[2.2 PAWaC - Public Administration Web as Corpus 9](#_Toc372819837)

[2.3 Costruzione di PAWaC 11](#_Toc372819838)

[2.4 Introduzione alla pipeline 12](#_Toc372819839)

[2.5 La pipeline 13](#_Toc372819840)

[2.5.1 Fase 1: raccolta degli URLs 14](#_Toc372819841)

[2.5.2 Fase 2: controllo degli URLs duplicati 15](#_Toc372819842)

[2.5.3 Fase 3 : download dei file 16](#_Toc372819843)

[2.5.4 Fase 4 : creazione del corpus 16](#_Toc372819844)

[2.6 Strategie di formazione delle tuple per costruire PAWaC 18](#_Toc372819845)

[2.7 Confronto dei corpora 20](#_Toc372819846)

[2.8 Lancio della pipeline 21](#_Toc372819847)

3. [CORPUS ANNOTATO 23](#_Toc372819848)

[3.1 La suite 24](#_Toc372819849)

[3.2 Formato CoNLL 26](#_Toc372819849)

[3.3 PAWaC in cifre 27](#_Toc372819850)

[Conclusioni 30](#_Toc372819851)

[Bibliografia 31](#_Toc372819851)

[Stografia 32](#_Toc372819851)

# **INTRODUZIONE**

Lo scopo del presente lavoro è la costruzione del corpus PAWaC, Public Administration Web as Corpus, un corpus specialistico di addestramento nell'ambito linguistico degli atti amministrativi.

Tale lavoro si inserisce nell'ambito del Progetto SEMPLICE che alcune aziende stanno sviluppando in collaborazione con Laboratorio di Linguistica Computazionale del Dipartimento di Filologia, Letteratura e Linguistica, per la gestione e l'organizzazione dei documenti della Pubblica Amministrazione mediante una piattaforma ad hoc.

Il corpus PAWaC è stato costruito seguendo la prassi, da poco più di un decennio diventata comune nella Linguistica Computazionale, dell'utilizzo del web come fonte di dati linguistici per la costruzione di corpora ed è stato successivamente annotato in maniera automatica.

Dopo una presentazione dei corpora come strumenti di studio e delle motivazioni che conducono all'utilizzo del web come fonte primaria di dati, abbiamo messo in evidenza i vantaggi e i problemi che questa scelta comporta, offrendo l'esempio di un *tool*, BootCat, che agevola la raccolta dei dati linguistici dal web.

Nel presente lavoro però abbiamo scelto di sviluppare un nostro strumento, capace di raccogliere dati selezionati dal web, superando al tempo stesso i problemi incontrati: una pipeline che trae spunto da BootCat e di cui descriviamo nel dettaglio le fase che la compongono.

Successivamente vengono esposte le metodologie che sono state esplorare e confrontate per la creazione del corpus PAWaC. Infine riportiamo il tipo di annotazione a cui esso è stato sottoposto e un'analisi dei risultati ottenuti.

**-1-**

# **IL WEB COME CORPUS**

* 1. **Cosa sono i corpora**

I corpora sono collezioni di testi, per lo più di grandi dimensioni, prodotti in contesti comunicativi reali (es. articoli di giornale, registrazione di discorsi, ecc.), selezionati e organizzati in base a criteri specifici per essere fonte di dati linguistici per determinate analisi linguistiche.

La costruzione di corpora ha origini anteriori all'avvento dell'informatica in quanto la raccolta di testi, puramente cartacea, era già una prassi comune nello studio del linguaggio come, per esempio, un corpus ottenuto come collezione delle opere di un certo autore. Lo sviluppo delle tecnologie informatiche ha dato nuovo vigore a questo metodo di indagine linguistica, potenziandone sia l'aspetto quantitativo (permettendo la raccolta di un numero sempre maggiore di testi), che quello funzionale (ottimizzando l'esplorazione dei testi e sviluppando modelli computazionali della lingua).

La costruzione di un corpus nasce da un'opera di selezione: i testi infatti devono essere scelti in base a criteri che condizionano la natura del corpus stesso e l'uso che ne può essere fatto. I testi possono essere interi o porzioni di lunghezza prefissata a seconda del tipo di indagine che si vuole condurre.

Possiamo avere corpora generali, nei quali i testi selezionati rispecchiano varietà diverse di una lingua, e corpora specialistici che includono testi appartenenti a uno specifico dominio tematico o una singola varietà linguistica (es. linguaggi settoriali, medico, giuridico, burocratico, ecc. oppure linguaggi particolari come il linguaggio infantile). I primi sono dunque caratterizzati da un elevato grado di Generalità per cui risultano plurifunzionali e spesso progettati come corpora di riferimento per lo studio di una lingua nel suo complesso. Al contrario i secondi hanno il grado minimo di Generalità.

Poiché nasce da un'opera di selezione, l'insieme di testi che compone un corpus si configura come un campione statistico di una lingua o di una sua varietà cioè rappresenta un sottoinsieme della popolazione più generale che è oggetto di studio.

In questa ottica dunque utilizzare come fonte di dati un corpus, richiede che esso soddisfi alcune condizioni, ovvero che sia adeguato come dimensione quantitativa e che i testi che lo compongono siano selezionati in modo da permettere generalizzazioni corrette e affidabili sulla popolazione intera di riferimento: il campione deve essere il più possibile una rappresentazione fedele in scala ridotta della popolazione. Solo a queste condizioni le conclusioni che si possono trarre dal corpus possono essere estese all'intera popolazione, il che equivale a dire che il corpus deve essere un campione rappresentativo. In particolare un corpus generale, per essere rappresentativo, deve essere bilanciato, ovvero diversificato rispetto a un ampio spettro di tipi testuali (D. Biber, 1993: 220).

La rappresentatività di un corpus è un vincolo sia quantitativo che qualitativo che ne condiziona fortemente la funzione di modello di una lingua, ma dipende strettamente dal grado di Generalità del corpus: i corpora specialistici, per esempio, costituiscono il caso più semplice poiché si riferiscono a varietà della lingua ristrette e ben definibili.

* 1. **Perché usare il web come corpus**

La linguistica dei corpora, intesa come la disciplina che ha introdotto l'analisi di testi per studiare le regolarità linguistiche di una lingua, si è concentrata prevalentemente sulla cura degli aspetti qualitativi nella selezione dei testi, in modo da ridurre la possibilità che tratti linguistici rilevanti non siano rappresentati nel campione.

Un corpus tuttavia, è comunque una porzione limitata di una popolazione linguistica potenzialmente infinita ed è quindi possibile che esistano in essa eventi linguistici che non si è in grado di osservare nel corpus stesso. Una strada alternativa si è aperta al crescere della capacità dei computer di gestire quantità sempre maggiori di dati: siccome è possibile effettuare indagini statistiche significative soltanto se un determinato evento si presenta un numero consistente di volte, diventa essenziale avere a disposizione corpora di grandi dimensioni che dovrebbero permettere di raccogliere dati linguistici sufficienti per lo sviluppo di modelli e analisi affidabili. Questo modo di procedere permette di tenere conto anche del fatto che la maggior parte dei fenomeni linguistici segue una distribuzione zipfiana, nella quale pochi elementi hanno una frequenza molto elevata e molti elementi, al contrario, hanno frequenze molto basse. Analizzare una grande quantità di testi aumenta dunque le probabilità di riuscire a cogliere fenomeni linguistici con bassa frequenza.

Dove trovare una quantità così elevata di dati? Risulta piuttosto naturale utilizzare fonti che per loro stessa natura raccolgono grandi quantità di dati in formato testuale, quali ad esempio giornali, agenzie di stampa, archivi istituzionali nazionali o internazionali, ecc.

Esiste però una fonte praticamente illimitata di dati per analisi linguistiche ed è il web.

Il web possiede una doppia natura in quanto da una parte rappresenta la più grande collezione di testi a cui attingere per la costruzione di corpora, dall'altra può esso stesso essere considerato un unico corpus.

Come risorsa di materiale testuale, secondo la stima di A. Kilgarriff e G. Grefenstette (2003) la quantità di testo accessibile sul web equivale a circa 2000 miliardi di parole, ovvero circa 20 terabyte. E sono stime ormai datate. Oggi si può dire che compaiono miliardi di nuove pagine web al giorno (Alpert e Hajaj, 2008).

Il web presenta inoltre il vantaggio di contenere i testi in formato digitale e di costituire una risorsa di informazione multilingue: si stima che il 75% delle pagine web indicizzate sia in inglese, 6.8% in giapponese, 1.5% cinese, 1.1% spagnolo e 0.9% in italiano (A. Lenci, S. Montemagni e V. Pirrelli, 2005). Anche questi dati sono soggetti a variazioni temporali, via via che nuove lingue acquistano progressivamente rilevanza (esempio il cinese). Per quanto riguarda l'italiano, nonostante abbia una presenza ridotta in percentuale, può comunque contare su quasi 2 miliardi di parole.

Questa enorme quantità di testi spazia su un'ampia gamma di generi, domini, ecc. che permettono la costruzione di corpora sia generali che specialistici, monolingue o multilingue, dando la possibilità di indagini linguistiche anche su lingue minoritarie, comunque presenti in maniera consistente nel web. Per di più i contenuti si presentano in diverse modalità: audio, video, puro testo.

Non da ultimo, il Web è costantemente aggiornato, e permette così di documentare fenomeni linguistici emersi anche in tempi recenti, che non è pertanto possibile studiare attraverso risorse tradizionali

Ciò che fa di questi vantaggi l'aspetto vincente del web è la sua notevole accessibilità, poiché con un personal computer e l'accesso a una connessione a banda larga "one can compile and process a multimillion-words corpus in minutes" (W. H. Flecther, 2011).

Come si può dunque utilizzare al meglio questa mole di dati? Il web però può diventare fonte diretta di dati linguistici fungendo esso stesso da corpus su cui compiere analisi computazionali. In realtà il web non rientra nella nozione di corpus come campione rappresentativo di una lingua. Infatti non ne presenta tutte le diverse varietà: risultano privilegiati alcuni domini come tecnologie, news e ne sono quasi del tutto assenti altri come il parlato spontaneo e le varietà regionali. Ne consegue che, come risorsa di riferimento per una lingua, la rappresentatività del web è inferiore a quella di un qualunque corpus bilanciato. Inoltre il web presenta, a prescindere dalla sua rappresentatività, l'impossibilità di essere sottoposto nel suo complesso a processi specifici, essenziali per l'analisi linguistica computazionale di un corpus, come ad esempio il *PoS-Tagging* di cui parleremo in seguito.

Pur con questo limite il web rimane una collezione di testi promettente come fonte di dati per analisi linguistiche.

Una volta stabilito che i dati ottenuti dal Web sono in grado di rispondere a determinate esigenze di ricerca, rimane tuttavia aperta la questione di come avervi accesso.

Il comune uso dei motori di ricerca come Google, negli studi linguistici, non è utile in quanto gli algoritmi che stanno alla base di essi non seguono criteri linguistici ma privilegiano determinati tratti quali il numero di accessi degli utenti, il numero di citazioni e riferimenti. Inoltre la maggior parte dei documenti ricavabili dal web è servita al pubblico in HTML, ossia in formato testo arricchito da tags. I dati quindi vanno "ripuliti".

La strategia più diffusa per costruire corpora dal web consiste nel reperire, salvare e manipolare offline dati provenienti da esso. A tale scopo una delle opzioni più promettenti per la costruzione di web corpora sembra essere quella di eseguire *crawling* su larga scala della rete con tecniche simili a quelle utilizzate dai motori di ricerca. Al contrario di quello che accade quando si ricorre a questi ultimi, un *crawler* autonomo permette di avere maggior controllo sulla procedura di raccolta dei dati. Ciò nonostante continua ad essere alto il rischio di raccogliere documenti indesiderati ("dati sporchi"), i quali però possono essere poi ripuliti eliminando, ad esempio, le pagine ripetute o generate automaticamente.

Un *crawler* è un software che analizza in forma automatizzata i contenuti della rete; il suo punto di partenza è una lista di URLs, chiamati *seeds*, da visitare. In questi siti esso esegue una copia del documento contenuto, individua tutti gli *hyperlinks* presenti e li aggiunge alla lista degli URLs da visitare. Il processo può essere concluso manualmente o dopo che un determinato numero di collegamenti è stato seguito. Tutto ciò è molto affascinante ma un po' pericoloso, perché, come detto sopra, per un *crawler* è difficile evitare contenuti duplicati e rischia, se non è ben progettato, di incorrere in malfunzionamenti o intralciare il funzionamento dei server o della rete se entra in *loop*.

* + 1. **BootCat**

BootCat (M. Baroni e S. Bernardini, 2004) è un tool veloce, efficace e versatile per la costruzione di corpora dal web, particolarmente utile per piccoli corpora specialistici come traduzioni, preparazione di materiale per l'insegnamento, ecc.

Inoltre BootCat si presta anche ad essere utilizzato per creare corpora generali di grandi dimensioni per lessicografia e ricerche linguistiche generali (S. Sharoff, 2006).

La procedura base prevede il seguente iter: il primo passo è la scelta di un piccolo insieme di parole chiave, dette *seeds*, che si ritengono tipiche del dominio di interesse. Con tali *seeds* vengono create le “tuple”, cioè gruppi di *seeds* (coppie, triplette, ecc.) combinati tra loro casualmente. La lunghezza delle tuple condiziona il tipo di corpus che si ottiene in quanto più corta è la tupla, maggiore sarà la dimensione del corpus e minore la sua accuratezza (intesa come precisione nel selezionare tipi di documento ritenuti validi ai propri scopi). E' necessario dunque trovare un equilibrio tra queste due opposte esigenze; nel tutorial di BootCat[[1]](#footnote-1) si suggerisce di costruire corpora generali con tuple con lunghezza pari a 2, mentre per corpora specialistici si consiglia di usare tuple di lunghezza 3.

Una volta costruite le tuple esse vengono utilizzate come valore di querie da effettuare sul motore di ricerca Bing[[2]](#footnote-2) sfruttando l'apposita API[[3]](#footnote-3) sviluppata da Microsoft, per ottenere una lista di URLs per ogni query effettuata. Si ottengono N risultati per query, dove N è un numero prefissato che non può superare 50 per motivi legati a Bing stesso. BootCat possiede un primo sistema di controllo sugli URLs che gli evita di recuperare duplicati.

Nell'ultima fase dagli URLs ottenuti viene scaricato il testo, ripulito e costruito il corpus. Anche in questa fase BootCat controlla che i documenti scaricati non siano "troppo" simili onde evitare possibili duplicati. Arrivati a questo punto del processo, si può effettuare iterativamente un numero potenzialmente illimitato di cicli estraendo nuovi *seeds* dal corpus ottenuto, basati su un paragone di frequenza (relativa) con corpora di riferimento, e ripetendo i passaggi a partire dalla formazione delle tuple con questi nuovi *seeds*, fino al raggiungimento del corpus desiderato.

**-2-**

# **COSTRUIRE PAWaC**

* 1. **Corpora d’addestramento**

L'applicazione di metodi statistici alla linguistica computazionale si basa sulla possibilità di costruire modelli di un certo fenomeno linguistico a partire dagli eventi osservati all'interno di un corpus che viene detto di addestramento. Gli eventi osservati possono essere le parole del corpus, i loro significati o categorie sintattiche, come anche strutture più complesse quali sequenze di parole, sintagmi, frasi, ecc. L'utilità di un corpus d'addestramento consiste nella possibilità di raccogliere dati quantitativi sull'occorrenza di determinati eventi linguistici e nel consentire ai metodi statistici a loro volta di ricavare delle regolarità dai dati analizzati.

Questa metodologia può essere applicata alla linguistica con l'assunzione di base che la lingua possa essere vista come un sistema probabilistico. Infatti è possibile spiegare un fenomeno assumendo che esso sia il prodotto di un sistema aleatorio, per il quale si costruisce un modello probabilistico a partire da un corpus di osservazioni sul sistema: il corpus di addestramento. I dati da esso estratti sono usati per addestrare il modello, cioè per stimare le probabilità del verificarsi degli eventi prodotti dal sistema.

La bontà del modello viene valutata in base alla sua capacità di prevedere correttamente il comportamento futuro del sistema. Elemento chiave è il grado di *Generalità* del modello che a sua volta dipende dalla *Rappresentatività* del corpus di addestramento rispetto al tipo di fenomeno da spiegare.

* 1. **PAWaC - Public Administration Web as Corpus**

PAWaC è il corpus di addestramento sviluppato in questo lavoro. Si tratta di un corpus specialistico che fa riferimento allo specifico dominio amministrativo della lingua italiana, avente lo scopo di addestrare e testare i modelli statistici che opereranno su tale dominio.

I corpora specialistici, per loro stessa natura, vengono costruiti all'interno di confini ben definiti di una lingua di riferimento, pertanto rispetto ai corpora generali possono risultare di dimensioni più ridotte senza peraltro perdere in rappresentatività. Quando il campo di ricerca è ristretto ad un particolare dominio linguistico (conversazioni telefoniche, giornali, notiziari, agenzie di stampa, ecc.) tali corpora possono essere addirittura preferibili, come dimostra il numero di corpora specialistici in aumento (A. Lenci, S. Montemagni e V. Pirrelli, 2005: 44).

.

Sono corpora di questo tipo, ad esempio:

* Reuters Corpus: rilasciato nel 2000, contiene una raccolta di notizie dell'agenzia fino al 2004 quando il progetto è cessato. Il corpus, interamente marcato in XML, è nato come strumento utile per la ricerca e lo sviluppo di tecnologie linguistiche computazionali.
* "La Repubblica" corpus: è una grande raccolta di testi in italiano del giornale omonimo, costruito dalla scuola per interpreti SSLMIT di Bologna. Il corpus è annotato sintatticamente e semanticamente e contiene approssimativamente 380 milioni di *tokens*.
* EPIC Corpus: corpus specialistico multilingue parallelo (italiano, spagnolo e inglese), contiene i discorsi del Parlamento Europeo e le corrispondenti traduzioni. Il corpus è stato costruito raccogliendo le registrazioni degli interventi dei vari speakers e delle traduzioni degli interpreti, effettuate durante un elevato numero di sessioni plenarie del 2004.

EPIC è composto da 3 subcorpora contenenti i testi dei discorsi originali nelle tre lingue e da 6 subcorpora con i testi delle relative traduzioni.

Il corpus PAWaC, rispetto ai corpora specialistici così descritti, presenta un livello di specializzazione ulteriore in quanto il campo di ricerca da noi studiato si configura come un sottodominio molto specifico all'interno del dominio linguistico degli atti amministrativi. Tale sottodominio è costituito infatti da atti esclusivamente prodotti dalla Pubblica Amministrazione ed in particolare solo da delibere e determine comunali dei comuni toscani.

Questa scelta dipende dal fatto che il presente lavoro si inserisce nell'ambito del Progetto SEMPLICE, progetto che alcune aziende, capofila della quali è 01S, stanno sviluppando per la gestione e l'organizzazione dei documenti della Pubblica Amministrazione mediante una piattaforma ad hoc.

Il Laboratorio di Linguistica Computazionale dell'Università di Pisa è tra i partners di tale progetto e il lavoro di questo studio contribuisce addestrando modelli statistici che saranno utilizzati dal laboratorio stesso per il progetto.

In questa ottica è stato posto il vincolo di privilegiare documenti di tipo delibere e determine, pur non considerando "rumore", e quindi errori, altri tipi di documenti che rientrano comunque negli atti amministrativi comunali.

In virtù dell'interesse per un sottodominio così ristretto, è stato curato in particolare l'aspetto di *precision[[4]](#footnote-4)* rispetto a *recall[[5]](#footnote-5)*: ciò significa tendere a ridurre al minimo il numero di testi che non rientrano nel dominio e a mantenere limitato il numero di testi che rientrano negli atti amministrativi ma non sono delibere e determine. In questa ottica ci si trova nella necessità di rinunciare ad avere un corpus di grandi dimensioni.

* 1. **Costruzione di PAWaC**

Nella costruzione di corpora l'uso tipico dei *crawlers* per scaricare dati dal web è considerato una prassi comune (M. Baroni, S. Bernardini, A. Ferraresi, E. Zanchetta, 2008, in questo articolo sono citati diversi esempi di costruzione di corpora tramite web *crawling*); nel caso particolare in cui, come nel presente lavoro, sia necessario privilegiare la precisione, l'utilizzo di questo strumento può risultare inefficace: un *crawler* solitamente scarica da un sito web spostandosi al suo interno attraverso gli *hyperlinks* presenti. Ciò potrebbe portare al recupero di molti documenti non inerenti al sottodominio di nostro interesse. Inoltre la struttura dei siti web comunali non è uniforme poiché spesso i documenti che vogliamo estrarre sono ospitati in un diverso dominio web; ciò ha contribuito quindi a rendere l'approccio mediante *crawling* poco utile.

Una caratteristica significativa dei documenti di nostro interesse è che parte consistente di essi è messa a disposizione del pubblico sul web da parte della Pubblica Amministrazione in formato *pdf*; questo ci ha portato a dover scartare uno strumento di lavoro come BootCat che è più improntato a lavorare con files di tipo testuale e HTML. BootCat si è rivelato comunque prezioso come spunto per la progettazione di un nostro software, adatto ad una ricerca dei documenti più mirata e in grado di lavorare con documenti di tipo *pdf*.

* 1. **Introduzione alla pipeline**

Lo strumento di cui avevamo bisogno doveva trovare in rete, scaricare e convertire in formato testo i documenti *pdf* di nostro interesse, quali delibere e determine dei comuni toscani.

Nella fase di progettazione del nostro strumento è risultato conveniente trarre spunto dalla procedura di BootCat per organizzare una pipeline, in modo da far compiere a programmi dedicati le operazioni di ricerca, recupero, e conversione files. Analogamente a BootCat ci è apparso naturale far coincidere le operazioni necessarie con altrettante fasi nella costruzione della pipeline, alle quali abbiamo aggiunto una fase di filtro degli URLs duplicati che anche BootCat prevede.

A differenza di BootCat, la nostra pipeline non necessita di una fase in cui si eliminano elementi web strutturali dai documenti scaricati in quanto il *target* della nostra fase di ricerca sono documenti esclusivamente pdf. Ciò ha richiesto però di introdurre una fase di conversione dei documenti scaricati in formato testuale prima di poter costruire il corpus.

La pipeline da noi sviluppata si caratterizza per la modalità di ricerca degli URLs indirizzata ad una significativa precisione: come BootCat sfruttiamo l’API di Bing, per ottenere gli URLs, senza però estendere la nostra ricerca all’intero dominio web che li contiene. Ciò significa fare maggiore affidamento sui risultati proposti dal motore di ricerca e scaricare esclusivamente i documenti contenuti negli URLs stessi.

Restringere il campo di ricerca comporta una riduzione di documenti scaricati per ogni URL visitato. Tale riduzione è stata da noi compensata incrementando sensibilmente il numero possibile di URLs ottenibili con ogni singola query, come verrà mostrato in seguito.

* 1. **La pipeline**

La presente pipeline è scritta interamente in Python 2.7[[6]](#footnote-6) e si articola in 4 fasi, come in figura 1.

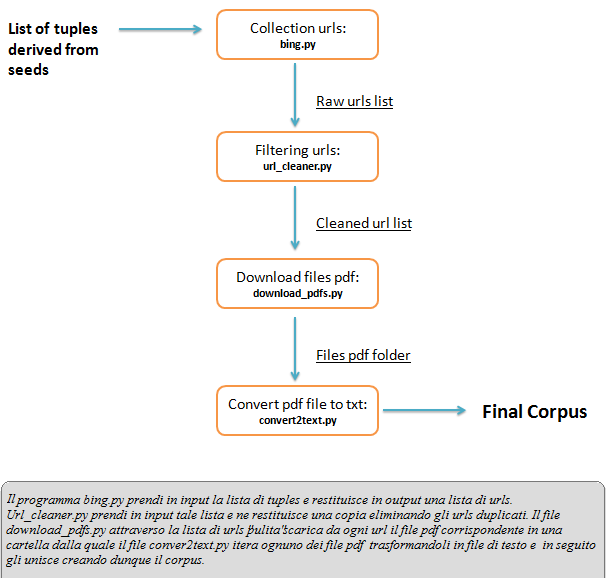


Figura 1. Diagramma di flusso della pipeline.

I dati di input del programma bing.py, il passo iniziale, consistono in un insieme di tuple ricavate da una lista di *seeds* che, come nel caso di BootCat, sono utilizzate come valore di query. In output viene restituita la lista di URLs che è l'insieme dei risultati di ogni query effettuata.

La lista di URLs diventa a sua volta input del programma Url\_cleaner.py che restituisce come output una nuova lista di URLs dalla quale sono stati eliminati gli URLs duplicati della lista precedente. Sulla lista ripulita opera il programma download\_pdfs.py che scarica da ogni URL il file pdf in esso rinvenuto e lo colloca in una cartella apposita.

Il programma convert2text.py converte in files di testo tutti i files pdf contenuti nella cartella, riunendoli in un unico grande file testuale. Quest'ultimo è il risultato finale della intera procedura, articolata in 4 fasi.

* + 1. **Fase 1: raccolta degli URLs**

Il primo passo della pipeline è il programma bing.py, che accetta in input la lista di tuple e restituisce in output una lista di URLs.

Tale operazione permette di ricavare in maniera automatica URLs che indirizzano a documenti web che rispondono il più possibile agli obiettivi di ricerca determinati da un'appropriata costruzione delle tuple.

I parametri necessari sono:

1. *Path* del file input: il percorso del file all'interno del computer.
2. Nome del file output.
3. Tipo di file: specifica il tipo di file che si vuole ricercare.
4. Lingua: serve per specificare la lingua in cui i documenti vengono ricercati.
5. Chiave di un account Microsoft Azure Marketplace[[7]](#footnote-7): per poter utilizzare l'API di Bing è necessario avere un account email Microsoft associato a un account di Azure Marketplace con cui iscriversi al servizio Bing Search API offerto, anche gratuitamente, da Bing. La chiave dell'account è indispensabile per eseguire ogni singola query.
6. Numero di risultati ottenuti per query: Bing impone un massimo di 50 risultati per query, che può rivelarsi un numero troppo piccolo. In tal caso, viene ripetuta la stessa query più volte in modo da ottenere il numero di risultati richiesti: per esempio se si richiedono 125 risultati si ripetere la stessa query tre volte, recuperando la prima volta i primi 50 risultati, la seconda i 50 risultati successivi e la terza volta i 25 seguenti.

|  |
| --- |
| > python bing.py -i input\_file -l language -t fileType -k key -n number -o output\_file |

Figura 2. Comando da shell bing.py.

Una volta definiti i parametri e lanciato il programma, questo legge ogni tupla del file input e la trasforma, attraverso il metodo.get() della libreria requests[[8]](#footnote-8) in una query per l'API di Bing; ogni singola query restituisce una lista di URLs. Ogni URL viene poi registrato nel file output.

* + 1. **Fase 2: controllo degli URLs duplicati**

Il programma Url\_cleaner.py accetta come input una lista di URLs, che può essere l'output della fase precedente o anche creata separatamente. Effettua un controllo sull'eventuale presenza di URLs duplicati all'interno della lista ed elimina quelli che riconosce come tali.

Questo passaggio della pipeline è indispensabile, considerato il grave problema di recuperare documenti identici che porterebbe ad un'analisi linguistica alterata, ad esempio fornendo falsi dati di frequenza.

Alla fine di quest'operazione di controllo viene restituita la lista di URLs iniziale opportunamente ripulita.

|  |
| --- |
| > python url\_cleaner.py –i input\_file |

Figura 3. Comando da shell url\_cleaner.py.

* + 1. **Fase 3 : download dei file**

Il programma download\_pdfs.py ha il compito di scaricare sul computer i file a cui accede tramite una lista di URLs.

Partendo da questa lista, per ognuno degli URLs applica lo stesso metodo.get() della libreria requests (già citata), precedentemente utilizzato per ottenere una lista di URLs e che ora serve per scaricare il contenuto degli URLs dagli URLs stessi. Il contenuto viene poi salvato in una cartella appositamente creata.

In questa fase viene creato inoltre un registro, chiamato registry.txt, di tutti gli URLs. A ciascuno di essi viene associato il nome del file che contengono, tenendo in questo modo traccia, per ogni documento salvato, del suo URL di origine. Viene tenuta traccia anche di quegli URLs dai quali non è stato possibile scaricare il contenuto (principalmente a causa di errori di connessione); essi vengono registrati nel file descarded\_urls.txt.

Il risultato output del programma è la cartella chiamata *pdf\_saved* in cui sono raccolti tutti i file scaricati.

|  |
| --- |
| > python download\_pdfs.py –i input\_file |

Figura 4. Comandi da shell dowload\_pdfs.py.

* + 1. **Fase 4 : creazione del corpus**

L'ultima fase della pipeline si occupa di trasformare i file in formato *pdf* in file di testo e unire tutti questi files in un unico file di testo che costituisce il corpus.

Il programma *convert2text.py* apre uno alla volta i files PDF contenuti nella cartella pdf\_saved e lo converte in testo utilizzando il programma **pdftotext**[[9]](#footnote-9)di cui sono dotati i sistemi Linux e Mac come applicazione predefinita. Il file convertito viene salvato in una cartella temporanea creata all'inizio del procedimento.

|  |
| --- |
| > python convert2text.py –i input\_folder |

Figura 5. Comando da shell convert2text.py.

Nel caso in cui non sia possibile convertire un PDF in file di testo, l'errore riscontrato viene riportato in un registro apposito chiamato Log\_error.txt che contiene il nome del PDF e il tipo di errore generato.

È importante a questo punto rilevare che l'utilizzo del programma **pdftotext** all'interno della pipeline ne costituisce un fattore limitante in quanto il suddetto programma opera solo su sistemi Linux e Mac; pertanto la fase finale della pipeline è utilizzabile solo se si opera con uno di essi.

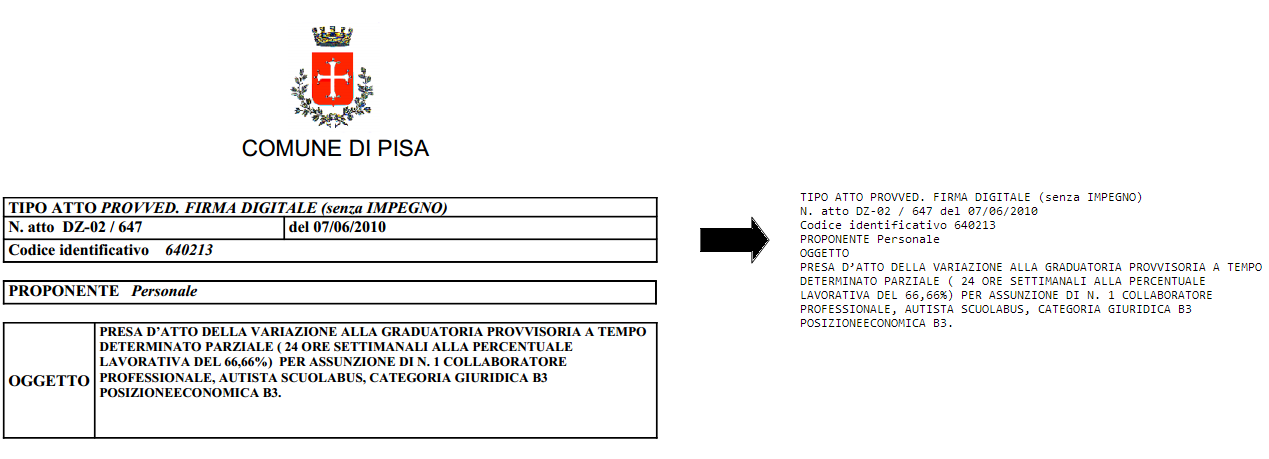


Figura 6. File pdf convertito in file di testo.

Una volta conclusa la procedura di conversione per tutti i files PDF, il programma compie due operazioni su ognuno dei files convertiti: in primo luogo il file viene processato attraverso la funzione *sentence\_fixer()* che serve a controllare e a stabilire se un "a capo" è frutto di un errore di conversione del file; questo processo avviene dapprima eliminando tutti gli "a capo" presenti nel testo originariamente convertito e poi decidendo di "andare a capo" solo quando sono realizzate precise condizioni da noi definite.

Questa funzione è utile per la buona riuscita della lettura del nostro corpus da parte dei programmi di analisi a cui è destinato, poiché essi hanno il compito, tra l'altro, di suddividere il testo in frasi.

In secondo luogo, sempre per facilitare le analisi successive, ad ogni file viene aggiunto un tag di inizio (<doc ..) e un tag di fine (</doc>) che ne delimitano il testo contenuto; il tag di inizio contiene due attributi il cui valore è rispettivamente il nome del PDF e l'URL dal quale è stato scaricato.

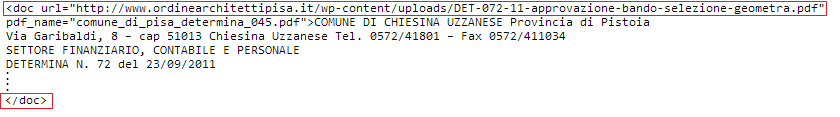


Figura 7. Tags di inizio e fine documento.

Nel corso di questa operazione il programma si occupa anche di ricercare ed eliminare tutti i caratteri contenuti nel file che non sono Unicode, ovvero quei caratteri presenti nel PDF d'origine -che non rientrano nel set di caratteri utf-8- che sono stati trasformati in carattere di errore durante la conversione in testo.

Alla fine di tutte queste procedure il programma riunisce tutti i singoli files di testo processati in un unico file complessivo che costituisce il corpus finale ed elimina la cartella temporanea in cui erano stati salvati.

* 1. **Strategie di formazione delle tuple per costruire PAWaC**

Come è stato detto a inizio capitolo, l'utilizzo della pipeline prevede la creazione preliminare di una lista di tuple, formate raggruppando casualmente i *seeds* che sono stati selezionati.

Comunemente ciò che viene fatto quando si cerca di delimitare un singolo dominio di una lingua è partire da una base di *seeds* consistente (dell'ordine delle centinaia di parole, come per esempio nel lavoro di M. Baroni, S. Bernardini, A. Ferraresi, E. Zanchetta già citato, oppure nel lavoro anch'esso già citato di S. Sharoff).

La necessità di ottenere risultati specificamente per un sottodominio, quello oggetto del nostro studio, ci ha portato a sperimentare strategie differenti, tutte aventi però come punto di partenza comune la scelta di usare un gruppo molto ristretto di *seeds* e di costruire delle tuple con abbinamenti non casuali, diverse per ognuna delle strategie.

Ogni strategia ha prodotto un corpus di cui abbiamo valutato un campione e il confronto tra i campioni ci ha consentito di individuare la strategia più adatta ad avviare la creazione di PAWaC.

La prima strategia sperimentata prevedeva di costruire le tuple abbinando due *tokens*, il primo dei quali è il nome di un comune scritto nella forma "comune\_di\_nomecomune" e il secondo è uno dei quattro nomi dei tipi di documento di interesse. Abbiamo così abbinato ogni singolo comune di volta in volta con i *seeds* "determina" "delibera" "determinazione" e "deliberazione" creando quindi quattro tuple per comune, in cui cambiava solo il secondo *token* e rimaneva invariato il comune (es. "comune\_di\_abbadia"\_delibera).

L'utilizzo delle virgolette (dette operatori di ricerca)[[10]](#footnote-10) nel costrutto "comune\_di\_nomecomune" fa sì che il motore di ricerca interpreti tale costrutto come un unico *token* che viene poi sommato al nome del documento (il secondo *token*) nella ricerca.

La seconda strategia da noi sperimentata prevedeva l'uso di un differente *token* per rappresentare di volta in volta il comune studiato; le tuple formate in quest'ultimo caso si presentavano così: site:comune.abbadia.it\_delibera. I motori di ricerca interpretano l'operatore "site:nome di dominio" effettuando la ricerca della parola "delibera" solo all'interno del dominio richiesto.

La struttura del nome di dominio dei siti dei comuni toscani è sempre la stessa, ad eccezione di 10 comuni. Essendo le tuple create automaticamente, abbiamo deciso di scartare i dieci comuni che non seguivano la struttura degli altri e questo ha condizionato la scelta del numero di comuni presi in esame per tutte e tre le strategie, in modo da poterle comparare al meglio. Quindi abbiamo ristretto il campo del nostro studio a 277 comuni toscani su 287 formando così 1108 tuple (277\*4).

L'ultima strategia sperimentata prevedeva l'abbinamento di due *token* molto simili a quelli utilizzati nella prima ma con una piccola differenza che ha portato significativi risultati. Le tuple erano composte così: "comune\_di\_abbadia"\_"delibera\_n".

Una volta ottenuti i tre gruppi di tuple abbiamo costruito tre corpora, uno per ogni gruppo, con lo scopo di valutarli e confrontare le strategie dai quali sono nati.

* 1. **Confronto dei corpora**

Ogni corpus è stato costruito utilizzando la pipeline descritta sopra.

Per valutare la bontà dei tre corpora abbiamo deciso di estrarre poi in maniera casuale 100 documenti da ognuno di essi e controllando manualmente, documento per documento, l'eventuale appartenenza al sottodominio di nostro interesse, dividendoli poi in tre categorie:

1. Documenti errati: quei documenti che non sono atti amministrativi o che erano atti amministrativi ma non emanati da comuni.
2. Atti amministrativi: tutti i documenti che sono atti amministrativi comunali.
3. Documenti corretti: i documenti che sono delibere o determine emanate da comuni.

L'analisi su un campione casuale di documenti ci permette di valutare la bontà dei corpora e dunque delle tuple che li hanno generati.

Il campione di documenti prelevato dal corpus formato con la prima strategia per la formazione delle tuple ha dato i seguenti risultati:

* Formato tuple: "comune\_di\_abbadia"\_delibera.
* Documenti errati: 54.
* Atti amministrativi: 35.
* Documenti corretti: 11.

Il numero di documenti errati nel campione è più della metà del campione stesso.

La dimensione del corpus è risultata di circa 60MB.

I risultati ottenuti da tale corpus sono stati valutati inadeguati per lo scarso numero di documenti corretti.

Il campione di documenti relativo alla seconda strategia ha prodotto questi risultati:

* Formato tuple: site:comune.abbadia.it\_delibera.
* Documenti errati: 18.
* Atti amministrativi: 63.
* Documenti corretti: 19.

In questo caso la valutazione del corpus è migliore del precedente. La dimensione in questo caso è scesa a poco più di 30MB e un esame attento dei risultati ha mostrato che erano sbilanciati. Abbiamo ottenuto infatti il massimo dei risultati dai siti dei grandi comuni e pochi dai siti di comuni di medie o piccole dimensioni. Inoltre non abbiamo ottenuto risultati dai comuni che rendono disponibili i loro documenti su piattaforme di hosting esterne (es. B.U.R.T)[[11]](#footnote-11).

L'ultimo campione estratto, relativo alla terza strategia ha dato i seguenti risultati:

* Formato tuple: "comune\_di\_abbadia"\_"delibera\_n.".
* Documenti errati: 18
* Atti amministrativi: 40
* Documenti corretti: 42

L'ultimo campione ha prodotto risultati da noi ritenuti soddisfacenti per l'elevato numero di risultati corretti e considerando comunque non errati, quindi accettabili, i documenti che rientrano nella categoria di Atti amministrativi. La dimensione di questo corpus è risultata di oltre 100MB, nonostante una parte consistente dei documenti (tutti i risultati dal comune di Poggibonsi in poi, in ordine alfabetico) non sia stata scaricata per problemi tecnici.

Abbiamo deciso dunque di utilizzare la terza strategia per la creazione di PAWaC.

* 1. **Lancio della pipeline**

Dopo il confronto descritto, la terza strategia ci ha permesso di formare le tuple idonee ad essere utilizzate come input per la pipeline. Abbiamo scelto di richiedere 300 risultati per ogni singola query, nel tentativo di bilanciare l'esigenza di creare un corpus di dimensioni adeguate con l'esigenza di ridurre il più possibile il rischio di raccogliere un numero eccessivo di documenti errati.

Al termine dell'esecuzione della pipeline abbiamo infine ottenuto il corpus PAWaC, la cui dimensione finale è risultata di 142MB, ottenuti processando 3985 documenti per un totale di 24181010 parole.

**-3-**

# **CORPUS ANNOTATO**

La creazione di PAWaC è il primo passo di un processo che si completa con la successiva annotazione del corpus che, come accennato nel paragrafo 1.1, serve a rendere esplicite informazioni di carattere linguistico che si possono ricavare dal corpus stesso.

L'annotazione è un procedimento necessario per consentire le elaborazioni automatiche su di esso poiché permette, per esempio, il conteggio del numero totale di parole che lo compongono o la definizione del vocabolario (composto da tutte le parole tipo presenti nel testo), oppure attività più complesse come il calcolo TTR (Type Token Ratio)[[12]](#footnote-12) per valutarne la ricchezza lessicale.

L'annotazione di un corpus può essere effettuata su più livelli, caratterizzati da un diverso grado di approfondimento dell'analisi linguistica.

Il livello base è costituito dall'annotazione morfo-sintattica (*PoS-Tagging*) che ha lo scopo di assegnare ad ogni *token* del testo un'etichetta che ne esplicita l'appartenenza ad una particolare categoria grammaticale o parte del discorso (es. nome, verbo, ecc). Allo stesso livello vi è anche la lemmatizzazione, spesso combinata all'annotazione morfo-sintattica a cui è correlata. La lemmatizzazione permette di ricondurre ogni parola al relativo lemma (es. i verbi nella loro forma dell'infinito), per compiere ricerche che prescindono da variazioni morfologiche dei vocaboli.

L'annotazione morfo-sintattica è il livello a cui è stato annotato PAWaC, anche se esistono ulteriori livelli di annotazione possibili.

Un livello più approfondito che è importante menzionare è l'annotazione sintattica (*parsing*) che richiede il precedente livello di annotazione come base di partenza. Questo tipo di annotazione si avvale di un software detto parser sintattico (o anche solo parser) che individua le relazioni gerarchiche tra i costituenti di una frase e la suddivide nei suoi componenti sintattici, come sintagmi nominali, verbali, preposizionali ecc. scomponendoli a loro volta in sotto-sintagmi e ottenendo così una struttura ad albero che termina sempre con le concrete parole della frase (le foglie). I corpora così annotati vengono definiti *Treebank*s, appunto perché il risultato di tale analisi è il diagramma ad albero di una frase.

## 3.1 La suite

Sia il *parsing* che il *PoS-tagging* ricevono in input una frase che poi ciascuno di essi elabora a seconda delle proprie funzioni. Da questo si evince che un corpus deve essere "preparato" per la suddetta analisi, qualunque essa sia. Tipicamente quindi, sia il *parser* che il *PoS-tagger* costituiscono solo una parte di un insieme di programmi (detto suite) che realizza l'annotazione di un corpus.

Considerato che il corpus, cioè il materiale di partenza, è in principio un testo libero, si ha come necessità prioritaria la sua suddivisione in frasi. Il programma che svolge questa funzione viene detto *Sentence Splitter*: esso suddivide il testo in frasi basandosi sull'interpretazione dei punti come separatori tra di esse.

Questo però non è ancora sufficiente per permettere al parser o al *PoS-tagger* di operare. È necessario infatti che all'interno di ogni frase vengano poi riconosciute le parole, o meglio i "pezzi" (*chunks*) che la compongono. Di questo si occupa il programma definito *Tokenizer*; quest'operazione è più complessa di quanto si potrebbe immaginare: a fronte di un input costituito semplicemente da una sequenza finita di caratteri (la frase), produce in output una lista di pezzi dotati di significato. Ogni *Tokenizer* segue regole proprie che gli forniscono i criteri per stabilire se due o più parole costituiscono un unico pezzo o se devono essere considerate singolarmente (es. una data, in cui gli spazi separano giorno, mese e anno ma costituiscono un unico pezzo di significato).

A questo punto, il testo risulta pronto per essere analizzato dal *parser* o dal *PoS-Tagger*.

Nel caso particolare di PAWaC, che è stato annotato morfo-sintatticamente, sono stati utilizzati due *PoS-Tagger*: il primo ha effettuato una selezione grossolana che si limita a individuare nel testo solo 14 categorie grammaticali di base.

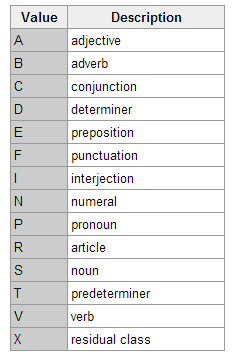
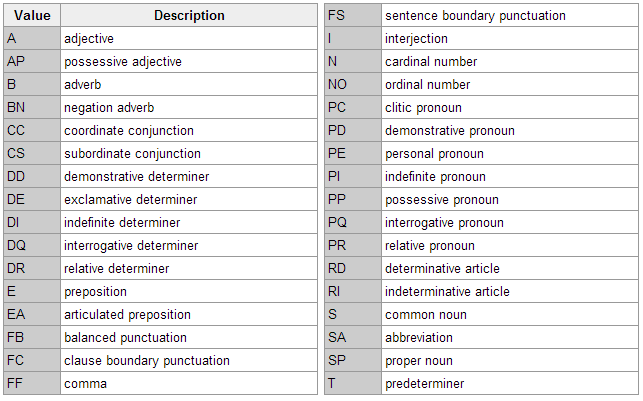


Figura 8. Le 14 categorie del primo PoS.

Il secondo è dotato di un *tag-set* molto più ampio ed effettua una marcatura più accurata, in grado di segnalare sotto-tipi differenti della stessa categoria grammaticale.

**Figura 9. Le categorie del secondo PoS**.

* 1. **Formato CoNLL**

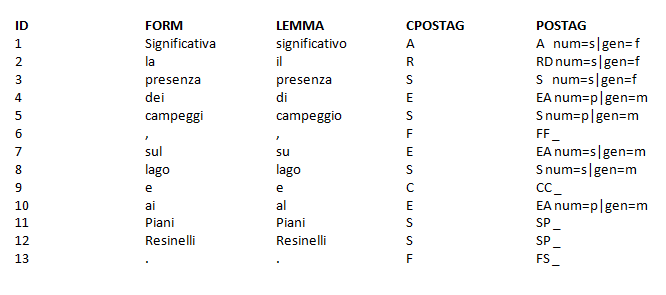
Il processo di annotazione, per sua natura, si presta ottimamente all'utilizzo del formato XML che permette di definire una gerarchia nelle etichette.

Al termine di questo processo dunque, il testo contenuto nel corpus appare profondamente modificato nel formato, a causa delle varie fasi di etichettatura.

La prima fase, già presente nella creazione del corpus, separa i documenti l'uno dell'altro, all'interno del corpus stesso.

Nella fase successiva i documenti suddivisi in frasi contengono tags che segnalano la fine e l'inizio di ognuna di esse(<s> frase </s>).

Nell'ultima fase vengono annotati gli elementi di ogni frase.



**Figura 10. Esempio di frase formattata all’interno di PAWaC**.

Ogni riga corrisponde ad un singolo elemento della frase con le etichette che lo descrivono:

* ID: è il contatore dei *token*; parte da 1 per ogni nuova frase.
* FORM: indica lo specifico *token*.
* LEMMA: indica il lemma a cui risale il *token*.
* CPOSTAG: rappresenta la categoria grammaticale a cui appartiene il *token* assegnata con il primo *PoS-Tagger*.
* POSTAG: rappresenta la categoria grammaticale più raffinata derivata dal secondo *PoS-Tagger*.
  1. **PAWaC in cifre**

L'utilizzo di procedure automatizzate per la costruzione dei corpora non permette di avere un completo controllo sul risultato finale. Per comprendere la composizione effettiva di un corpus è in realtà necessario svolgere analisi appositamente predisposte a posteriori.

Nel caso di PAWaC il nostro primo interesse è verificare le prestazioni della pipeline. Successivamente abbiamo effettuato un'analisi descrittiva del corpus in termini numerici.

In seguito, un'analisi più sofisticata ci ha permesso di avere un'indicazione della ricchezza lessicale del nostro corpus tramite il calcolo del TTR e della percentuale di hapax[[13]](#footnote-13).

In figura 11 sono riportati dati tecnici che consentono di compiere la verifica suddetta.

|  |  |
| --- | --- |
| Numero di tuple | 1108 |
| Numero di urls raccolti | 37850 |
| Numero di urls unici | 5105 |
| Numero di documenti scaricati | 3985 |
| Numero di documenti convertiti in pdf | 3871 |

Figura 11. Dati tecnici della pipeline.

Come è facile verificare, il numero di urls unici di documenti pdf di nostro interesse è notevolmente inferiore al numero di urls ottenuti considerando anche i duplicati. Da tali urls unici a loro volta non è stato possibile scaricare una quantità significativa di documenti per problemi principalmente dovuti alla connessione, il che ha comportato una riduzione del numero di risultati utili. Un'ulteriore riduzione si è avuta con la conversione dei documenti pdf a file di testo, poiché abbiamo riscontrato 71 problemi di codifica e 43 files corrotti.

La figura 12 presenta i dati dimensionali di PAWaC.

|  |  |
| --- | --- |
| Dimensioni del corpus di partenza | 142 MB |
| Dimensioni del corpus annotato | 632 MB |
| Numero di *token* | 24181010 |
| Numero di parole tipo (types) | 557654 |
| Numero di lemmi | 521122 |

Figura 12. Dati dimensionali di PAWaC.

Nelle figure 13 e 14 è riportato rispettivamente il numero di occorrenze delle categorie grammaticali determinate con il primo processo di PoS-tagging e quelle derivate dal secondo processo.

|  |  |
| --- | --- |
| S | 8290335 |
| E | 4276850 |
| F | 2898223 |
| N | 1973475 |
| V | 1863780 |
| A | 1667808 |
| R | 1236505 |
| C | 796204 |
| B | 492793 |
| P | 438156 |
| D | 116411 |
| X | 107673 |
| T | 20327 |
| I | 2470 |

**Figura 13. Frequenze primo PoS.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S | 5077549 | FC | 663282 | CS | 96040 | T | 20327 |
| SP | 2851053 | FB | 636852 | BN | 94768 | PE | 15791 |
| E | 2255034 | FS | 624235 | VM | 92254 | PQ | 4273 |
| EA | 2021816 | B | 398025 | DI | 79504 | I | 2470 |
| N | 1939383 | SA | 303741 | SW | 57851 | DQ | 1946 |
| A | 1631733 | VA | 235224 | PD | 44912 | DR | 1335 |
| V | 1536302 | PR | 210335 | AP | 36075 | DE | 1328 |
| RD | 1104156 | PC | 140713 | NO | 34092 | PP | 152 |
| FF | 973854 | RI | 132349 | DD | 32298 | SM | 141 |
| CC | 700164 | X | 107673 | PI | 21978 | P | 2 |

**Figura 14. Frequenze secondo PoS.**

La figura 15 riporta i risultati del calcolo del TTR e della percentuale id hapax presenti nel corpus.

|  |  |
| --- | --- |
| Type Token Ratio (TTR) | 2% |
| Percentuale di Hapax | 57,1% |

**Figura 15. TTR e percentuale di Hapax.**

Come mostrano questi dati la ricchezza lessicale del nostro corpus è risultata estremamente limitata, il che non meraviglia in quanto il linguaggio burocratico-amministrativo tende ad utilizzare parole e costrutti standardizzati e ripetuti.

# **CONCLUSIONI**

La costruzione del corpus PAWaC ha colto l'obiettivo di predisporre un corpus specialistico del dominio linguistico amministrativo formato recuperando dal web atti emanati da un insieme scelto di Pubbliche Amministrazioni comunali.

È importante sottolineare che la creazione di PAWaC non si è avvalsa esclusivamente di strumenti già esistenti, ma ha sollecitato lo sviluppo mirato di uno strumento ad hoc che si è rivelato efficace nel recuperare le risorse di cui avevamo bisogno e convertirle secondo le nostre necessità.

PAWaC è un corpus di addestramento, tipologia di corpora per la quale esiste un particolare interesse nato dall'introduzione di metodi statistici nelle indagini linguistiche. Tali metodi si basano sulla possibilità di costruire modelli di un certo fenomeno linguistico a partire dagli eventi osservati all'interno di un corpus, il corpus di addestramento.

In tale prospettiva il nostro corpus si presta dunque a usi futuri per addestrare un modello probabilistico che stimi le probabilità di eventi linguistici ritenuti rilevanti, basandosi sui dati estratti da esso. è proprio questo l'utilizzo che ne farà il laboratorio di Linguistica Computazionale dell'Università di Pisa nell'ambito del progetto SEMPLICE.

# **BIBLIOGRAFIA**

* D. Biber (1993), Using *Registered-Diversified Corpora for Generale Language Studies* in Computational Linguistics, n.19, pp. 220.
* A. Kilgarriff e G. Grefenstette (2003) *Introduction to the Special Issue on the Web as Corpus*, in Computational Linguistics n. 23, pp. 333-47.
* A. Lenci, S. Montemagni e V. Pirrelli (2005), *Testo e computer - Elementi di linguistica computazionale,* Carrocci editore, pp.47.
* W. H. Flecther (2011) *Corpus analisys of the world wide web*, pubblicato in Chapelle Carol A. (2011). Encyclopedia of Applied Linguistics. Wiley – Blackwell.
* M. Baroni and S. Bernardini (2004), *BootCaT: Bootstrapping corpora and terms from the web*. Proceedings of LREC 2004.
* S. Sharoff *Creating General-Purpose Corpora Using Automated Search Engine Queries.* In Baroni and Bernardini (eds.) Wacky! Working papers on the Web as Corpus. Bologna, Gedit.
* M. Baroni (CIMEC University of Trento), S. Bernardini, A. Ferraresi, E. Zanchetta (SITLeC University of Bologna) (2008), *The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora*.
* I. Chiari (2007) *Introduzione alla linguistica computazionale*, editori Laterza.

# **SITOGRAFIA**

* Alpert e Hajaj (2008), *We knew the web was big…*

<http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html>

* Corpora di Italiano dell'enciclopedia Treccani, <http://www.treccani.it/enciclopedia/corpora-di-italiano_(Enciclopedia_dell'Italiano)/>
* Tutorial di BootCat, <http://docs.sslmit.unibo.it/doku.php?id=bootcat:tutorials:basic_2>

Motore di ricerca Bing,

http://www.bing.com

* API di Bing,

<http://www.bing.com/developers/s/APIBasics.html>

* Python 2.7,

http://www.python.org

* Azure Microsoft,

http://datamarket.azure.com/

* Libreria Requests,

https://pypi.python.org/pypi/requests

* Pdftotext,

http://manpages.ubuntu.com/manpages/lucid/man1/pdftotext.1.html

* Guida alla ricerca di google, https://support.google.com/websearch/answer/136861?hl=it
* Progetto B.U.R.T. della Regione Toscana,

http://www.regione.toscana.it/burt

1. , <http://docs.sslmit.unibo.it/doku.php?id=bootcat:tutorials:basic_2> [↑](#footnote-ref-1)
2. http://www.bing.com [↑](#footnote-ref-2)
3. <http://www.bing.com/developers/s/APIBasics.html> [↑](#footnote-ref-3)
4. Rapporto tra il numero di documenti attinenti recuperati e il numero di documenti recuperati totale. [↑](#footnote-ref-4)
5. Rapporto tra il numero di documenti attinenti recuperati e il numero di documenti attinenti esistenti. [↑](#footnote-ref-5)
6. http://www.python.org [↑](#footnote-ref-6)
7. <http://datamarket.azure.com/> [↑](#footnote-ref-7)
8. https://pypi.python.org/pypi/requests [↑](#footnote-ref-8)
9. <http://manpages.ubuntu.com/manpages/lucid/man1/pdftotext.1.html> [↑](#footnote-ref-9)
10. <https://support.google.com/websearch/answer/136861?hl=it> [↑](#footnote-ref-10)
11. http://www.regione.toscana.it/burt [↑](#footnote-ref-11)
12. Rapporto tra parole tipo e numero di *token*; è una misura della ricchezza lessicale di un testo ed ha valore compreso tra 0 e 1. Più alto è il valore maggiore è la ricchezza lessicale. [↑](#footnote-ref-12)
13. Token che ricorrono una volta sola nel testo. [↑](#footnote-ref-13)