SULLA CORREZIONE ORTOGRAFICA AUTOMATICA

LUCA CHIODINI

Un'applicazione concreta partendo da Wikipedia

ITIS P. Paleocapa
Esame di Stato 2015

Luca Chiodini: *Sulla correzione ortografica automatica,* Un'applicazione concreta partendo da Wikipedia
Esame di Stato 2015

SOMMARIO

Obiettivo del lavoro è costruire un sistema che proponga una correzione automatica di parole e frasi, con un approccio statistico. La sorgente dei dati su cui lavora l'algoritmo di correzione è la celebre enciclopedia Wikipedia, in lingua italiana. Si espone un servizio di API e una semplice applicazione web per gli utenti finali. Si pone particolare accento all'intera implementazione con un occhio di riguardo per l'ottimizzazione.

INDICE

1 INTRODUZIONE

> All men are really most attracted by the beauty of plain speech.
>
> Thoreau

1.1 NATURAL LANGUAGE PROCESSING

L'elaborazione del linguaggio naturale (da qui in avanti *NLP*, Natural Language Processing) è la parte delle scienze informatiche che si occupa dell'interazione tra macchina e linguaggio umano. Da questa premessa segue immediatamente che NLP è correlata all'ambito dell'interazione uomo-macchina.

Con "linguaggio umano" intendiamo una lingua che viene usata per le comunicazioni di tutti i giorni dagli umani, per esempio l'italiano, l'inglese o l'hindi. A differenza dei linguaggi artificiali, come i linguaggi di programmazione o le notazioni matematiche, i linguaggi naturali sono difficilmente fissabili in un insieme finito di regole. Questi ultimi infatti hanno un'infinità di varianti e, per di più, evolvono con il passare del tempo.

NLP può variare dall'essere estremamente facile all'estremamente difficile: da un lato potremmo confrontare due diversi stili di scrittura semplicemente contando le frequenze delle parole, dall'altro potremmo voler essere in grado di dare risposte sensate a qualsiasi quesito pronunciato da un umano.

Per capire quanti problemi ci sono all'interno dell'NLP proviamo a elencarne alcuni tra i più comuni:

- Traduzione automatica: siamo abituati a poter tradurre automaticamente del testo da un linguaggio umano all'altro, ad esempio con Google Translate.
- Generazione di linguaggio naturale: gli assistenti vocali recuperano informazioni da svariate basi di dati e le presentano sotto forma di testo in un linguaggio umano.

- OCR (riconoscimento ottico dei caratteri): data un'immagine contenente del testo, stampato o scritto, ricavare il testo corrispondente.
- POS (Part Of Speech): data una frase, determinare la parte del discorso di ciascuna parola (verbo, nome, ...) in modo analogo a quanto si fa nell'analisi grammaticale.
- Divisione in frasi, riconoscendo gli opportuni segni di interpunzione.
- Riconoscimento vocale: partendo da una sorgente audio si vuole ottenere una trascrizione fedele a quanto pronunciato. Questo compito richiede a sua volta che sia risolto il problema di individuare, all'interno di un frammento audio, la corretta divisione delle parole.
- Classificazione del testo: se arriva una mail il cui oggetto contiene parole come "Diventa subito milionario" oppure "Hai ereditato in Nigeria una somma di denaro", il messaggio dovrebbe essere automaticamente riconosciuto come spam.
- Sentiment analysis: identificare la posizione soggettiva (cosiddetta polarizzazione) partendo da un testo. Un esempio può essere l'individuare lo stato emotivo di una persona (triste, arrabbiato, felice, ansioso, ...) partendo da un tweet.

1.2 IL PROBLEMA DELLA CORREZIONE AUTOMATICA

Ogni elaboratore di testi moderno che si rispetti include oggigiorno un potente correttore ortografico. Simili strumenti sono integrati anche nei browser e soprattutto nelle tastiere intelligenti degli smartphone. Nonostante sembrino utili, tutti odiano i correttori ortografici perché sono ritenuti "stupidi".

Il grosso equivoco di fondo è che il pubblico generalista non conosce i retroscena della correzione automatica e tende quindi a considerarlo un compito come gli altri, avente un esito binario: funziona oppure non funziona. Spesso si sottovaluta che i correttori ortografici automatici sono uno degli ambiti dell'intelligenza artificiale (AI) e nei lavori di AI, allo stato dell'arte, non è possibile ottenere una precisione del 100%. Si punta, per quanto possibile, a rendere questa percentuale più alta possibile per fornire un risultato di qualità.

Kukich, nel suo articolo pubblicato nel 1992 [5], divide il problema generale della correzione in tre sottoproblemi via via più ampi e difficili:

- Rilevamento dei "non-word error": rilevare errori che diventano parole sconosciute (come *esmpio* al posto di *esempio*).
- Correzione dei "non-word error": correggere le parole che diventano parole sconosciute osservando l'errore di per sè (correggere *esmpio* in *esempio*).
- Rilevamento e correzione basata sul contesto: usare il contesto per decidere se rilevare e correggere una parola anche qualora questa sia stata sfortunatamente trasformata in un'altra parola esistente ("real-word error"). Consideriamo come esempio il caso in cui *tre* è stato scritto senza la prima lettera (*re*).

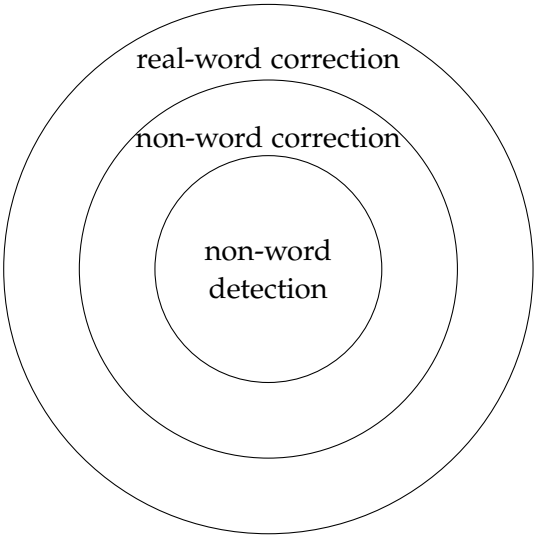

Figura 1: Diversi tipi di correzione automatica

Obiettivo di questo lavoro è arrivare a un sistema che provi a rispondere a tutte le richieste sopra descritte.

2

PRIMA DI METTERSI ALL'OPERA

Si supponga che l'obiettivo sia correggere *amre*, una parola palesemente sbagliata perché non contenuta in nessun dizionario della lingua italiana. È evidente come sia impossibile stabilire con certezza quale debba essere la miglior soluzione al quesito: per esempio, *amre* dovrebbe essere corretta in *amore* oppure in *amare*, oppure ancora in *mare*? Questo semplice esempio suggerisce come la correzione ortografica si debba basare sulle probabilità.

2.1 TEORIA DELLA PROBABILITÀ

Nel corso degli anni è stato sviluppato un modello probabilistico il cui fine è cercare la parola intesa dall'utente dato un input in cui vi è un'alterazione di qualche tipo. Il modello prende il nome di "noisy channel" [2].

Il nostro compito è quello quindi di trovare una parola w, quella che originariamente l'utente intendeva, che sostituisca la parola errata nel miglior modo possibile. Ciò significa che si è alla ricerca di una correzione c all'interno di un dizionario D tale che venga massimizzata la probabilità che la parola c sia esattamente quella che l'utente voleva (w).

$$\arg\max_{c \in D} P(c|w) \tag{1}$$

Per il teorema di Bayes, ciò equivale a:

$$\arg\max_{c \in D} \frac{P(w|c) \cdot P(c)}{P(w)} \tag{2}$$

Possiamo ulteriormente semplificare l'equazione osservando che il denominatore $P(w)$ dipende esclusivamente dalla parola originale e quindi sarà costante indipendentemente dall'ipotesi di correzione. Prese due correzioni c_1 e c_2 distinte tra loro, ciò che massimiz-

za l'equazione (2) massimizza sicuramente anche (3), essendo $P(w)$ costante.

$$\arg\max_{c \in D} \underbrace{P(w|c)}_{\text{error}} \cdot \underbrace{P(c)}_{\text{language}} \tag{3}$$

Il risultato mostra quindi come la probabilità che c sia la miglior correzione è data dal prodotto tra due fattori: $P(w|c)$ e $P(c)$. Il primo viene chiamato "error model" e ci dice quanto probabile è che c sia la parola che si intendeva rispetto a w. Il secondo invece viene detto "language model" e ci dice quanto probabile è che c sia una parola, all'interno del dizionario D che contiene tutti i candidati.

2.2 PROBABILITÀ BASATA SUL CONTESTO

In non poche situazioni fare una scelta basata solo su una singola parola può portare a una valutazione errata. Si consideri ad esempio la frase: *Nel mare si nuota*. Riprendendo l'esempio fatto a inizio capitolo, supponiamo che la parola *mare* sia stata erroneamente digitata come *amre*. L'algoritmo di correzione potrebbe riportare come parole nel dizionario D, tra le altre, *amore* (con una 'o' mancante) e *mare* (con 'm' e 'a' invertite). Seguendo l'algoritmo mostrato nel paragrafo precedente è possibile che venga riportata come soluzione *amore*, che compare spesso nella lingua italiana, ma che nella frase *Nel amore si nuota* è evidentemente fuori posto.

Per risolvere questo genere di problemi estendiamo il "language model" in modo che consideri anche ciò che viene prima e dopo la parola che stiamo analizzando. Nel caso di esempio possiamo affermare con sicurezza che $P(\text{Nel}\,|\,\text{mare}) > P(\text{Nel}\,|\,\text{amore})$ (visto che *nel mare* è una sequenza di parole corretta, mentre *nel amore* non lo è).

Per evitare un appesantimento eccessivo dell'algoritmo, in questo progetto ci limiteremo a considerare, data una possibile correzione c, la parola precedente (w_{i-1}) e quella seguente (w_{i+1}). Sequenze di due parole sono detti bigrammi, in contrapposizione alle singole parole (unigrammi). Generalizzando, una sequenza di N parole viene definita N-gramma.

Analizziamo la sottoparte della formula che dovrebbe considerare c (in sostituzione di w_i) e w_{i-1}. Applicando la teoria della probabilità condizionata si ha che:

$$P(w_{i-1}, c) = P(c|w_{i-1}) \cdot P(w_{i-1}) \tag{4}$$

Per la "regola della catena" (nota anche come teorema della probabilità composta) si ha che:

$$P(w_1, w_2, \ldots, w_n) = P(w_1|w_2, \ldots, w_n) \cdot \ldots \cdot P(w_{n-1}|w_n) \cdot P(w_n) \quad (5)$$

Se consideriamo i bigrammi, concludiamo che la probabilità di una sequenza è semplicemente il prodotto delle probabilità condizionate dei suoi bigrammi.

Recuperando l'equazione (3) e adattandola per essere usata con i bigrammi si ottiene la seguente formula (implementata nell'algoritmo di correzione):

$$\arg\max_{c \in D} \underbrace{P(w|c)}_{\text{error}} \cdot \underbrace{P(w_{i-1}|c) \cdot P(c) \cdot P(c|w_{i+1})}_{language} \quad (6)$$

2.3 SCELTA DEL DATASET

Come abbiamo visto nei paragrafi precedenti, la probabilità gioca un ruolo essenziale nell'algoritmo di correzione. Diventa quindi cruciale stabilire dove prendere una grande mole di dati su cui calcolare probabilità significative. Per la legge dei grandi numeri si avranno migliori risultati tanto più l'insieme dei dati sarà ampio.

Google, che fa un ampio uso di N-grammi (sono celebri l'autocorrezione del motore di ricerca e il traduttore automatico), ha rilasciato pubblicamente un insieme di dati raccolti dai suoi crawler che scandagliano quotidianamente la rete[1]. Questi dati sono però purtroppo relativi solo alla lingua inglese (dove sono presenti oltre un miliardo di occorrenze). Google rilascia anche un database estratto da Google Books, questa volta anche in lingua italiana: purtroppo però le osservazioni sono minori e viene aggiornato con poca frequenza[2].

Costruire un proprio crawler che scandagli il web e memorizzi tutte le informazioni può essere un'attività tecnicamente interessante ma poco realizzabile nella pratica. Si può invece pensare di sfruttare Wikipedia in lingua italiana. Essa contiene un'enorme quantità di pagine (all'atto della stesura di questo testo se ne contano circa 1,2 milioni), scritte perlopiù da umani e in continuo aggiornamento.

1 https://catalog.ldc.upenn.edu/LDC2006T13
2 http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

2.4 ERROR MODEL

L'uso di Wikipedia, che abbiamo scoperto essere conveniente allo scopo, risolve in realtà solo parte del problema. Attraverso l'analisi del testo possiamo calcolare le probabilità degli unigrammi e dei bigrammi, che costituiscono il "language model".

Finora non è stato affrontato l'"error model", ovvero la probabilità che una parola w sia diventata un errore x per un certo motivo. Calcolare sperimentalmente questa probabilità è difficile, perché servirebbe un corpus di parole errate con la loro correzione. Ve ne sono diversi, anche di buona qualità, per la lingua inglese mentre mancano per la lingua italiana.

Per questi motivi adotteremo un semplice modello: la distanza Damerau–Levenshtein [3, 6]. Essa rappresenta il numero di modifiche per passare da una stringa a a una stringa b. Si considera una modifica un'eliminazione (rimuovere una lettera), un'inserzione (aggiungere una lettera), un'alterazione (cambiare una lettera in un'altra) e una trasposizione (scambiare due lettere adiacenti).

3

ALL'OPERA

> Tutto ciò che è bello e nobile è il risultato della ragione e di calcoli.
>
> Baudelaire

In questo capitolo verranno affrontati i principali passaggi per arrivare a dati utilizzabili per calcolare le probabilità, come richiesto dall'equazione (6).

3.1 DOWNLOAD DEL DATASET DI WIKIPEDIA

Wikipedia è una sorgente di dati ben nota nell'ambito della ricerca sull'NLP. All'indirizzo `http://dumps.wikimedia.org/` la Wikimedia Foundation ospita una serie di file pronti al download, generati periodicamente da bot, che consentono di reperire con più facilità i dati quando si tratta di una grande mole. In questo particolare scenario l'utilizzo delle API (si veda il paragrafo 3.5.1) è fortemente sconsigliato in quanto genera un inutile overload sui server e risulta comunque più lento rispetto al primo metodo.

All'indirizzo `http://dumps.wikimedia.org/itwiki/` sono ospitati i dump relativi alla versione italiana di Wikipedia. La base da cui si è partiti è la versione `20150316` nella variante che comprende tutte le pagine correnti, senza cronologia e senza file multimediali associati. La scelta è stata fatta sia per avere una dimensione dei dati gestibile (al momento della stesura il dump indicato ha una dimensione di circa 2 GB) sia per avere dati sempre "freschi": non avrebbe senso fare statistica su un testo che gli utenti hanno già corretto, migliorato o comunque rivisto.

3.2 DAL DATASET AL PLAINTEXT

Il dataset è compresso in formato `bzip2`. Per decomprimerlo è sufficiente eseguire da riga di comando:

```
$ tar xjf itwiki-20150316-pages-articles.xml.bz2
```

Verrà creato un file XML che contiene il testo di tutte le pagine insieme ad alcune informazioni aggiuntive (i cosiddetti *metadati*, per

esempio l'autore originale, la data di creazione e quella di ultima modifica, la categoria, ...). Nel tag `<text>` è contenuto il testo sorgente della pagina codificato in un miscuglio di HTML e XML ("MediaWiki Markup Language") che rende difficile il parsing. Poiché a noi interessa solo testo in chiaro senza informazioni ulteriori, tra i vari progetti disponibili possiamo sfruttare *Wikipedia Extractor* [7], un lavoro di alcuni ricercatori dell'Università di Pisa che "ripulisce" il contenuto della pagina.

Wikipedia in lingua italiana è una grossa enciclopedia con oltre un milione di voci: è quindi fondamentale che l'elaborazione dei dati avvenga in tempi accettabili. Il lodevole progetto dell'Università di Pisa è stato quindi migliorato da un altro studente che ne ha realizzato una versione multithread [10], sfruttando a pieno la grande potenza computazionale delle CPU odierne. Sulla postazione utilizzata per lo sviluppo di questo progetto, equipaggiata con un processore ad 8 core, si è osservato un incremento prestazionale circa del 600%.

Nel corso del progetto lo script è stato migliorato nei seguenti aspetti:

- Aggiunta la modalità di output in plaintext.
- Aggiunta la possibilità di specificare il numero di thread da utilizzare.
- Aggiunta la possibilità di specificare il nome dell'unico file di output (in origine era prevista una cartella con file divisi e rinominati casualmente).
- Lievi miglioramenti al supporto UTF-8.
- Revisione completa dello script per la piena compatibilità con lo standard di scrittura codice Python PEP8.

Quest'ultima versione è disponibile nel repository pubblico del progetto (cfr. 5.2).

3.3 dal plaintext agli n-grammi

Come si è visto nel capitolo 2, per raccogliere statistiche usabili nel determinare il "language model" abbiamo bisogno delle frequenze con cui unigrammi e bigrammi appaiono nel testo appena estratto. Questo compito soffre naturalmente degli stessi problemi di quello precedente: abbiamo bisogno di uno script che faccia il proprio lavoro in tempi ragionevoli e con un consumo di memoria accettabile.

La prima versione del programma realizzato svolgeva egregiamente il suo compito limitatamente a input di modeste dimensioni; si scontrava invece con il limite della memoria RAM disponibile sulla mac-

china di sviluppo (8 GiB) quando eseguito con input più consistenti. Al fine di migliorare l'efficienza della soluzione si è intervenuti su alcuni fronti:

- L'esecuzione del codice è stata resa parallela. Ciò comporta non pochi problemi in Python, in quanto la maggior parte delle implementazioni degli interpreti usano GIL[1] rendendo inutile l'uso dei thread. Si è quindi fatto ricorso alla creazione di processi figli (analogamente a quanto avviene in C con l'istruzione `fork()`) a cui si demanda il calcolo delle frequenze delle parole. L'architettura complessiva prevede quindi un processo che legge il file in ingresso una riga alla volta. Ciascuna riga diventa un "compito" da svolgere e viene messa in una coda che viene via via svuotata dai vari processi (worker). Quando la coda 1 è vuota, i worker terminano il loro lavoro e salvano i risultati nella coda 2, che verrà elaborata da un altro processo. Quest'ultimo unisce i risultati parziali calcolando le frequenze totali che vengono scritte in un file di output.

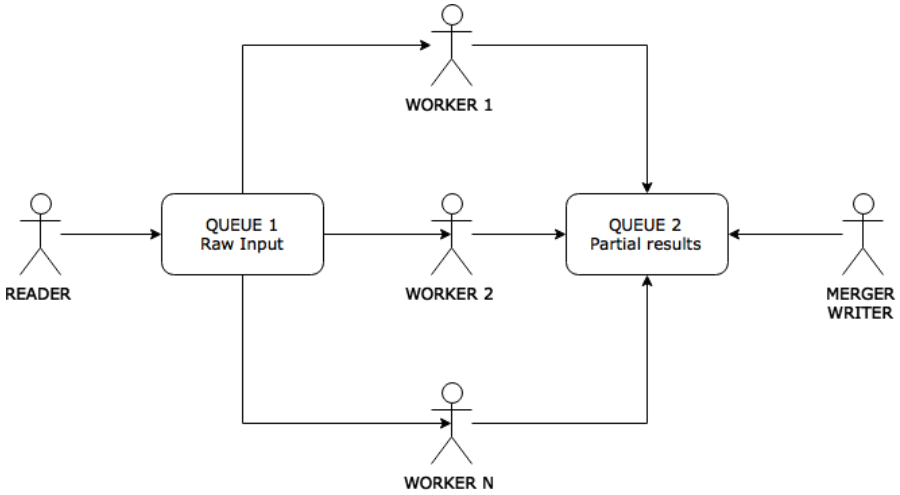

Figura 2: Schema logico di computeFrequencies.py

- Il codice del progetto è stato migrato alla versione 3.4 di Python che include un'interessante funzionalità: i dizionari a chiave condivisa[2]. Questa caratteristica consente all'interprete Python di allocare memoria una sola volta per memorizzare le chiavi dei dizionari tra più istanze della stessa classe. Poiché lo script prevede una classe per rappresentare un processo e `Counter`, la classe utilizzata per contare le frequenze, è derivata dalla classe dizionario, questa modifica porta un significativo miglioramen-

1 Global Interpreter Lock. È un meccanismo di sincronizzazione usato nei linguaggi interpretati che regola l'esecuzione di thread multipli affinché ne venga eseguito solo uno alla volta.

2 https://www.python.org/dev/peps/pep-0412/

to del consumo di RAM dello script. Da alcuni test di verifica, il risparmio di memoria si è mostrato superiore al 50%.

- Nonostante la miglioria precedente, per il calcolo di tutte le frequenze dei bigrammi lo script richiede un quantitativo di RAM superiore a quello disponibile nella macchina di sviluppo. Si è quindi scelto di eseguirlo su una macchina virtuale Amazon EC2 di tipo `c3.2xlarge` con 8 CPU virtuali e 15 GiB di RAM. Il programma ha quindi svolto il suo compito in 1018 secondi (circa 17 minuti), con un picco di occupazione RAM intorno ai 13 GiB.

3.4 Scelta del DBMS

Le occorrenze calcolate sono memorizzate in file con la seguente sintassi:

```
Ngramma Frequenza
```

Le frequenze sono quindi da memorizzare in una base di dati che conservi queste coppie (chiave, valore) e ne consenta l'interrogazione in modo estremamente efficiente. Per stabilire infatti la migliore correzione possibile per una parola, come abbiamo visto nel capitolo 2, è necessario calcolare un grande numero di probabilità. Per questo il DBMS Redis[3] fa al caso nostro [9]. Redis è un database NoSQL ultra efficiente per memorizzare nella RAM coppie chiave-valore, che possono facoltativamente essere salvate su disco per garantire la persistenza.

Le prestazioni di Redis variano, come è ovvio, dalla potenza dell'host su cui il server è installato ma anche dalla tipologia di connessione tra client e server (via TCP oppure UNIX socket). In linea di massima è comunque possibile affermare che Redis è in grado di servire oltre 100 000 query/s.

Poiché per sua natura Redis mantiene tutte le informazioni direttamente in memoria è imporante che il server sia dimensionato correttamente in termini di RAM. In particolare, per questa installazione che contiene tutti gli unigrammi e i bigrammi il processo `redis-server` occupa (a caricamento completato) circa 1.7 GiB. La soluzione scelta per la produzione è su una VPS dedicata allo scopo con 2 GiB di RAM, disco SSD e connettività a 1 GBps.

3 http://redis.io/

A LOOK INTO THE WORLD OF NOSQL DATABASES

NoSQL databases provide a mechanism for storing and retrieving data that is memorized in a different way than the tabular relations used in relational DBs.
NoSQL databases are designed in contrast to relational databases, which were first introduced by Codd, an English computer scientist. He used the definition of a mathematical relation to organize data into one or more tables of rows and columns.
NoSQL databases feature highly optimized key-value operations (i.e., storing and reading a value given a key). Datastore of this kind may not require fixed table schemas and usually avoid join operations.
The term gained popularity in early 2009 with the advent of big data. In fact NoSQL DBs are often employed in real-time web applications and those requiring fast computation of a huge amount of data. The benefits include:

- Scalability and superior performances;
- Capability to memorize large volumes of structured and unstructured data;
- An efficient architecture that can easily scale out;
- Most implementations are open source.

Instead, the downsides are:

- Immaturity;
- No support for a well-known and standardized language, as SQL;
- No ACID transactions;
- Many data structures (objects) cannot be easily represented as key-value pairs without a complex serialization.

3.5 AGGIORNAMENTI IN TEMPO REALE

Per fare in modo che il sistema proposto migliori con il tempo proponiamo una soluzione che migliori costantemente la base di dati su cui lavora l'algoritmo. Come già spiegato, tutto il lavoro di correzione si basa sulle probabilità ottenute dall'elaborazione di Wikipeida. La sorgente di dati non è stata scelta a caso: Wikipedia è continuamente aggiornata dagli utenti che correggono (inevitabili) errori e aggiungono contenuti. Aggiornando quindi il nostro database i dati in esso contenuti vengono continuamente raffinati e, speranzosamente, offriranno una qualità della correzione sempre migliore.

3.5.1 *MediaWiki API*

Wikipedia, oltre al servizio di dump descritto nel paragrafo 3.1, offre anche un insieme di API che consente a bot e terze parti di recuperare ed eventualmente modificare i dati. Il servizio di API, per la versione italiana, è disponibile presso `http://it.wikipedia.org/w/api.php`. Esistono diversi wrapper per l'API, ma spesso sono incompleti e poco aggiornati rispetto alle ultime versioni dell'interfaccia. Pertanto è stata creata una nuova classe Python che astrae l'interfaccia offrendo due metodi:

- `getRecentChanges`: restituisce una lista di (al più 500) oggetti JSON dove ciascuno rappresenta una modifica a una pagina di Wikipedia. È possibile specificare anche la data e l'ora da cui partire a recuperare le modifiche.
- `getPageReviews`: a partire da una lista di identificativi di revisione, recupera il testo di ciascuna delle pagine a cui le revisioni fanno riferimento prima e dopo di esse.

La classe è disponibile nel repository del progetto.

4

IMPLEMENTAZIONE DEL SISTEMA

4.1 SCHEMA ARCHITETTURALE COMPLETO

Come si è potuto evincere dal capitolo precedente, il sistema è composto da diverse parti. Ciascuna di esse interagisce con le altre in un modo ben definito. Chiariamo meglio l'architettura completa con uno schema, spiegato nel dettaglio di seguito.

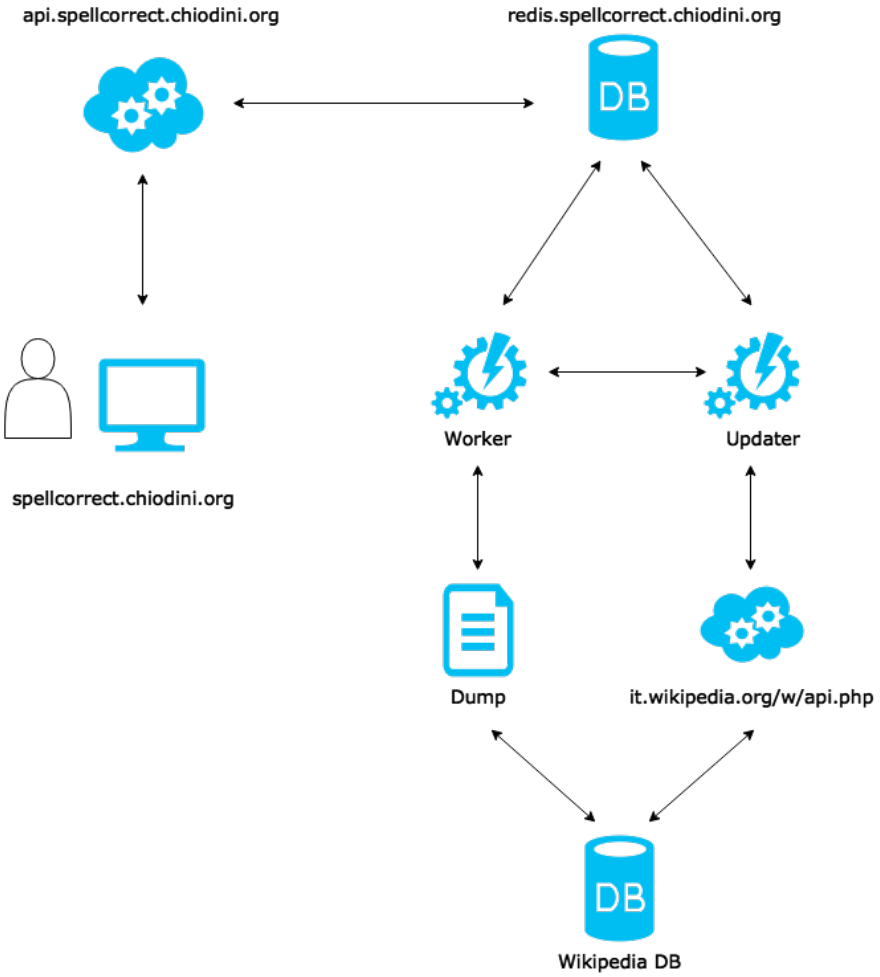

Figura 3: Architettura del sistema

- `spellcorrect.chiodini.org` è servito da Google App Engine[1] ed è costituito da una pagina statica (e alcuni asset). Essa contiene un'area dove l'utente inserisce la parola o la frase da correggere; questa verrà inviata al servizio di API tramite richiesta AJAX.
- `api.spellcorrect.chiodini.org` è una web API, servita sempre da Google App Engine. Essa è tecnicamente realizzata tramite Flask[2], un microframework che consente di realizzare un web server usando come linguaggio Python.

  La definizione dell'API è semplice: è necessario fare una richiesta a

  `http://api.spellcorrect.chiodini.org/correct/<str>`

  sostituendo `<str>` con la parola o la frase da correggere. Il servizio risponde nella notazione JSON. Ecco un esempio tipico di risposta:

```
{
 "cache": false,
 "corrected": "eccezione",
 "elapsed_time": "0:00:01.719220",
 "queries": 36,
 "input": "eccezzione"
}
```

  I campi della risposta hanno i seguenti significati:

  - `input` contiene l'input ricevuto;
  - `corrected` contiene la miglior correzione che il sistema è riuscito a calcolare per quell'input;
  - `elapsed_time` indica il tempo, con precisione in microsecondi, impiegato per l'elaborazione;
  - `queries` indica il numero di query effettuate al database Redis;
  - `cache` indica se la risposta è stata calcolata al momento oppure recuperata dalla cache (cfr. 4.3.2).
- `redis.spellcorrect.chiodini.org` ospita il DBMS Redis che si mette in ascolto sulla porta `TCP/6379`. Il DBMS è stato installato su una VPS (cfr. 3.4).
- La restante parte dello schema rappresenta quanto già ampiamente discusso nel capitolo precedente. Le frequenze iniziali

1 https://cloud.google.com/appengine/
2 http://flask.pocoo.org/

vengono memorizzate in Redis dopo essere state ottenute elaborando il dump. Gli aggiornamenti continui sono invece realizzati tramite un modulo ("Updater" nello schema in figura) che periodicamente interroga le API di Wikipedia e processa le eventuali modifiche delle frequenze inviandole sempre a Redis.

4.2 ALGORITMO

L'algoritmo di base è tratto da un articolo di Peter Norvig [8], che è l'attuale direttore del dipartimento di Ricerca presso Google. Rispetto alla versione originale sono stati modificati diversi punti, in accordo alla teoria definita nel capitolo 2.

Lo schema riporta i passi essenziali dell'algoritmo di correzione:

- L'input viene ricevuto dall'applicazione Flask.
- Per ciascuna parola presente nell'input, si procede alla generazione di tutte le parole candidate a sostituire quella presente. Per fare ciò, all'avvio viene precaricato in memoria tutto il dizionario delle parole conosciute. Tutte le parole presenti nel dizionario, aventi una distanza di Damerau–Levenshtein (cfr. 2.4) inferiore o uguale a due rispetto all'input, vengono considerate possibili candidati.
- Per ogni candidato vengono effettuate tre query al database che misurano la probabilità del candidato di per sè e quella rispetto alla parola precedente e seguente (naturalmente ove applicabile). Viene inoltre considerata anche la probabilità basata sulla distanza Damerau–Levenshtein a cui la parola si trova rispetto a quella originale. Per un approccio rigoroso, si riveda il paragrafo 2.2.
- Il candidato con la probabilità più alta viene considerato il migliore e restituito come soluzione. È importante notare che anche la parola stessa originaria costituisce un candidato: essa infatti potrebbe essere corretta e non necessitare di alcuna alterazione.

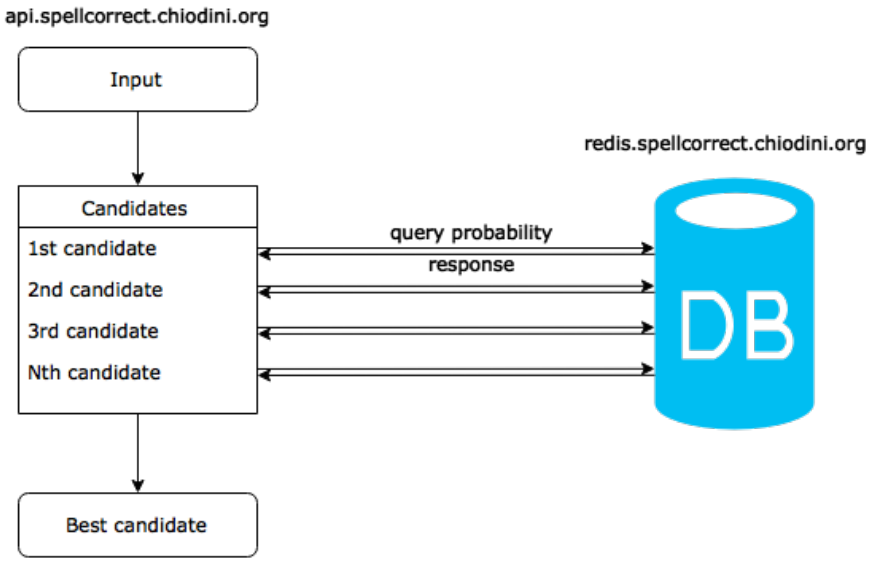

Figura 4: Algoritmo in dettaglio

4.3 EFFICIENZA

I paragrafi seguenti spiegano i principali accorgimenti che sono stati adottati per rendere il sistema il più efficiente possibile.

4.3.1 *Ottimizzazione dell'algoritmo*

A un lettore attento non sarà sfuggito che l'algoritmo mostrato in figura 4 esegue un gran numero di interrogazioni sul database. Poiché client e server sono due host distinti, ogni query avrà un tempo di esecuzione pari alla latenza tra i due host sommata al tempo di esecuzione dell'interrogazione stessa.

Redis è in grado di rispondere a una query in un tempo quasi sempre inferiore al millisecondo, ma la latenza tra due host su una rete geografica può essere anche 100 volte tanto. Essendo molto alto il numero delle interrogazioni è necessario evitare che per ciascuna sia necessario lo scambio di pacchetti TCP andata e ritorno: ciò porta a tempi di risposta complessivi inaccettabili.

Per risolvere il problema e ridurre drasticamente il tempo complessivo di risposta sfruttiamo una funzionalità di Redis, "pipeline", che consente di bufferizzare le query ed eseguirle tutte in un'unica volta.

Supponiamo che per correggere una certa frase servano 5000 query, che la latenza tra i due host sia di 50 ms e che Redis risponda a tutte

le query in 1 ms: la tabella mostra un confronto tra i tempi dei due algoritmi.

Metodo	Latenza TCP	Tempo query	Tempo globale
Originale	$5000 \cdot 2 \cdot 50$ ms	$5000 \cdot 1$ ms	~ 8 min
Ottimizzato	$1 \cdot 2 \cdot 50$ ms	$5000 \cdot 1$ ms	~ 5 sec

Tabella 1: Confronto tra i tempi di risposta dei due algoritmi

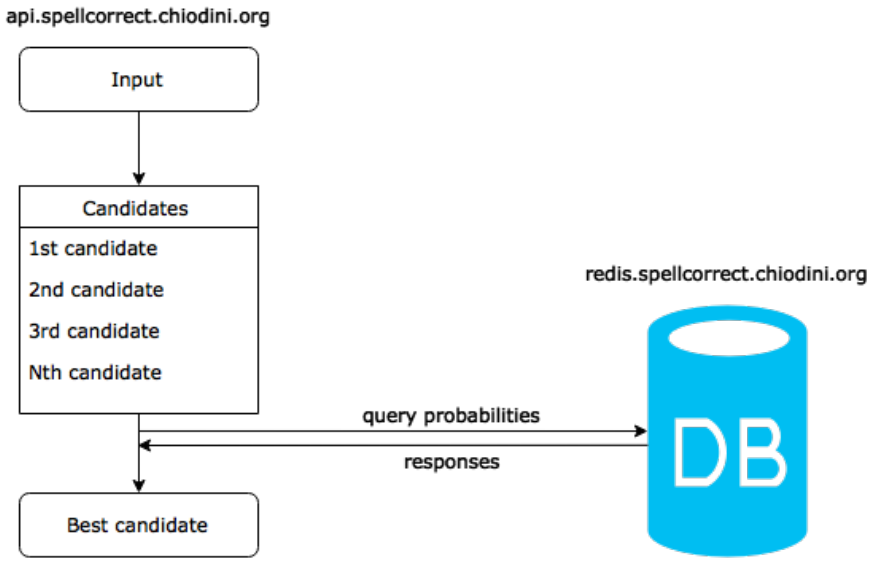

Figura 5: Algoritmo ottimizzato

4.3.2 *Cache*

Poiché più persone potrebbero richiedere la correzione delle stesse parole nel giro di poco tempo, ha senso mantenere una cache dei risultati. Questo permette di evitare il ricalcolo delle probabilità, che risulta essere molto oneroso.

All'interno della piattaforma App Engine è possibile sfruttare Memcached[3], un sistema distribuito ad alte prestazioni per salvare in memoria oggetti. Per quanto il progetto Memcached nasca e resti generico, esso è frequentemente impiegato come meccanismo di cache per alleviare il carico del database e velocizzare applicazioni web dinamiche e servizi di API.

Il funzionamento è tanto potente quanto semplice. Memcached prevede una chiave (in questo caso l'input), un valore (salviamo l'intero og-

3 http://memcached.org/

getto della risposta) e una scadenza. Ad ogni chiamata all'API viene eseguito questo codice (riportato nella sua versione essenziale):

```
# Try to save resources and reduce response time retrieving
# the answer from the cache.
res = memcache.get(words_str)

# If the response has not been found in the cache:
if res is None:

    # Compute it.
    res = self.parse(words_str)

    # Add it to the cache (expire time 1 day).
    memcache.add(words_str, res, 86400)
```

4.3.3 *Scalabilità*

Oltre alle misure descritte nei due paragrafi precedenti si è ritenuto opportuno tenere in considerazione particolare la scalabilità del servizio di API. La piattaforma di produzione scelta, Google App Engine, consente infatti di lavorare sul concetto di istanza per garantire alta affidabilità ed elevate prestazioni.

Il meccanismo è semplice: la piattaforma monitora continuamente svariati parametri sulle risposte fornite dall'applicazione, di cui il più importante è il tempo medio di risposta. Quando quest'ultimo valore aumenta oltre una soglia ritenuta inaccettabile è evidente che un'instanza dell'applicazione non è sufficiente a soddisfare il carico corrente sul server e ne viene pertanto lanciata un'altra. In modo analogo, quando l'utilizzo medio delle risorse di un'instanza (CPU, RAM) diventa irrisorio App Engine decide di terminare l'istanza riducendo il costo complessivo.

Tutti i parametri di configurazione relativi alle istanze sono manipolabili nel file `app.yaml` e includono, oltre a quanto già visto, la scelta delle caratteristiche hardware virtuali dell'istanza e la possibilità di averne un certo numero sempre attive, pronte a rispondere alle richieste.

5 RISULTATI

Valutare quanto un correttore ortografico automatico sia buono non è facile. La maggior parte della letteratura su questo tema esegue dei test partendo da un elenco noto di errori ortografici e della loro esatta correzione, valutando quindi la percentuale di casi in cui il sistema ha dato l'esito atteso.

È però quasi impossibile fare una simile operazione con il nostro correttore in lingua italiana, banalmente perché non esiste un insieme di dati di questo tipo disponibile. Sarebbe possibile crearlo da zero, ma si rischierebbe un forte effetto di bias[1]: i dati rischierebbero di essere costruiti appositamente per questo algoritmo di correzione. Di fatto verrebbe quindi valutata una correttezza specifica ma molto poco generica, rendendo così il risultato privo di interesse statistico.

5.1 SI PUÒ FARE DI MEGLIO?

Alcuni aspetti del progetto possono essere estesi e migliorati:

- Tutti i programmi per l'elaborazione del codice supportano completamente la codifica UTF-8 dello standard Unicode, tuttavia lo stesso non si può dire per l'algoritmo di correzione in sè. Esso infatti, per ridurre il numero possibile di candidati alla correzione, considera solo caratteri minuscoli dell'alfabeto latino. Almeno il supporto alle lettere maiuscole e alle lettere accentate dovrebbe essere offerto. Dustin Boswell, in una sua ricerca del 2004 [1], esplora alcune strutture dati avanzate che sembrano essere promettenti per questo impiego.
- L'algoritmo per la divisione del testo in parole utilizzato è banale: il carattere spazio ha la funzione di separatore. Questo non è evidentemente sempre vero anche considerando solo la lingua italiana: due parole potrebbero essere separate anche da un apostrofo. Questo problema è noto in letteratura come "Word segmentation" ed è un incubo se si considera, ad esempio, la lingua cinese dove alcune nostre frasi sono rappresentate con sequenze di una manciata di caratteri senza spazio.

  La divisione in parole resta comunque un problema anche considerando solo la lingua italiana: si prendano come esempio i

1 http://it.wikipedia.org/wiki/Bias_induttivo

due spezzoni di frase “insufficienti modi” e “in sufficienti modi”. Un errore ortografico potrebbe essere anche il mancato inserimento dello spazio (o viceversa la sua aggiunta involontaria). Già questo piccolo esempio mostra come la divisione in parole sia a sua volta un sottoproblema estremamente vasto e risolto, ancora una volta, in modo statistico.

- In un mondo globalizzato è oggi difficile pensare di realizzare un programma che funzioni solo in una sua piccola parte. Eppure per alcuni compiti piuttosto specifici è ancora necessario ragionare per area geografica. Uno di questi è sicuramente quello della correzione ortografica, il che lascia spesso delusi gli utenti, desiderosi di una soluzione universale. I risultati più promettenti in questo campo sembrano arrivare dai laboratori IBM [4].

5.2 PUBBLICAZIONE DEL LAVORO

L’intera piattaforma di correzione, che comprende tutti i programmi Python che compongono i vari passaggi, è stata pubblicata su un repository di GitHub nel profilo dell’autore[2] sotto licenza AGPL. L’impiego del sistema di controllo versione Git sin dalle prime fasi della programmazione ha consentito uno sviluppo particolarmente efficace, grazie alle funzionalità di branch e di revert delle ultime modifiche.

Gli indirizzi di riferimento a cui trovare l’applicazione sono:

- Sito web per utenti finali: `http://spellcorrect.chiodini.org`
- API: `http://api.spellcorrect.chiodini.org`

2 `https://github.com/lucach/spellcorrect`

BIBLIOGRAFIA

[1] Dustin Boswell. Generating candidate spelling corrections. http://dustwell.com/PastWork/NearestNeighborWordsPresent.pdf, 2004.

[2] Eric Brill and Robert C Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293. Association for Computational Linguistics, 2000.

[3] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964. ISSN 0001-0782. doi: 10.1145/363958.363994. URL http://doi.acm.org/10.1145/363958.363994.

[4] Ahmed Hassan, Sara Noeman, and Hany Hassan. Language independent text correction using finite state automata. In *IJCNLP*, pages 913–918, 2008.

[5] Karen Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, December 1992. ISSN 0360-0300. doi: 10.1145/146370.146380. URL http://doi.acm.org/10.1145/146370.146380.

[6] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[7] Università di Pisa Medialab. Wikipedia extractor. http://medialab.di.unipi.it/wiki/Wikipedia_extractor.

[8] Peter Norvig. How to write a spelling corrector. http://norvig.com/spell-correct.html.

[9] Jacob Perkins. *Python 3 Text Processing with NLTK 3 Cookbook*. Packt Publishing Ltd, 2014.

[10] Leonardo Souza. Multithreaded wikipedia extractor. https://bitbucket.org/leonardossz/multithreaded-wikipedia-extractor/wiki/Home.