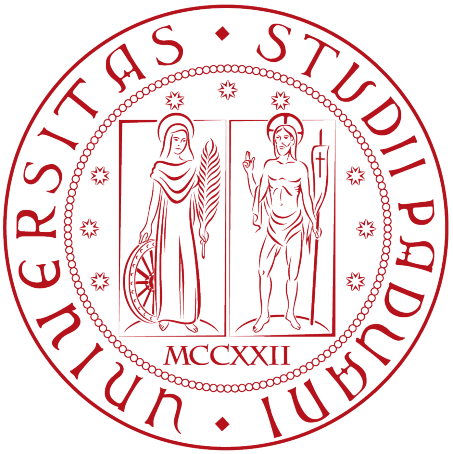
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Analisi e sviluppo di un'applicazione per la configurazione automatica di una chatbot professionale

Tesi di laurea triennale

Relatore
Prof.Ballan Lamberto

Laureando
Stefano Zanatta

Anno Accademico 2018-2019

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentododici ore, dal laureando Stefano Zanatta presso l'azienda PAT s.r.l. Gli obbiettivi principali del progetto erano:

* studio del motore semantico di Engagent, la chatbot professionale dell'azienda;
* studio dei risultati di un algoritmo di clustering, sviluppato da ricercatori esterni all'azienda;
* integrazione automatica dei cluster con Engagent, eseguendo operazioni di pos-tagging.

Ringraziamenti

Innanzitutto, vorrei ringrazione il Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura della tesi.
Vorrei ringraziare il tutor aziendale Davide Bastianetto, per avermi dato l'opportunità di svolgere lo stage alla PAT.
Desidero ringraziare con affetto i miei genitori e mio fratello per il sostegno morale (ed economico) dimostratomi durante questi anni.

Padova, Settembre 2019 Stefano Zanatta

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

PAT s.r.lè un'azienda italiana che da 25 anni sviluppa soluzioni software per altre aziende e privati.
L'azienda lavora su 5 diversi macro-progetti, uno dei quali è Engagent, la chatbot professionale oggetto dei miei due mesi di stage.
Dal 2013, PAT è entrata a far parte di Zucchetti Group.

1.1.1 Prodotti e servizi

L'azienda offre ai suoi clienti l'automatizzazione dei processi e il miglioramento dell'*user experience* dei loro prodotti. PAT concretizza quesi obbiettivi attraverso i seguenti prodotti:

* Engagent: chatbot semi-automatizzata per uso professionale;
* Helpdesk;
* Infinite: *CRM* software orientato alla relazione tra cliente e azienda;
* Brain *Interactive*: piattaforma per governare dei servizi personalizzati attraverso dei diagrammi di flusso;
* Teammee: piattaforma per la comunicazione tra i dipendenti in un'azienda, usando la logica dei social networks;

Engagent

Engagent è una chatbot orientata al business, con un agente virtuale integrato. In *backend*, un motore semantico permette di capire cosa sta chiedendo l'utente e trovare la risposta più coerente. Se la domanda è troppo complessa, il motore semantico estrate la categoria della domanda e la reinderizza all'operatore adeguato.

1.1.2 Organizzazione e Metodo di lavoro

L'azienda è divisa in più gruppi di lavoro, uno per ogni macro-progetto, oltre alla segreteria e direzione.

Ogni team è separato dagli altri, anche se la collaborazione tra le parti è necessaria. Tutti i team di sviluppo in PAT s.r.lseguono una metodologia Agile. Questa metodologia fa parte delle metodologie iterative. Le brevi iterazioni (o sprint, di circa 3-4 settimane) sono seguite dalla *review* del lavoro svolto. Il focus principale si trova nel cliente, difatti ci deve essere una interazione costante per capire quali sono le *feature* più importanti, ovvero quelli da sviluppare prima.
Il team a cui ho preso parte applica questa metodologia. Il contatto con il cliente è frequente, che sia manutenzione o nuove *features* da sviluppare. La piccola dimensione del team e le riunioni giornaliere permettono una buona collaborazione. Il team è gestito da un responsabile che organizza le riunioni e comunica con il manager dell'azienda.
Per quanto mi rigurarda, ho adottato senza difficoltà queste metodologie, perché molto simili a quelle utilizzate durante il progetto di ingegneria del software.

1.2 Strumenti e Tecnologie

1.2.1 Ambiente di lavoro

L'ambiente di lavoro utilizzato è Windows 10, assieme al pacchetto office per la maggior parte delle attività.
I team di sviluppo hanno libera scelta sugli editor. Per la codifica e la stesura dei docuemnti, ho scelto Visual studio. Per la creazione di diagrammi UML, ho utilizzato Astah UML.
Ogni sviluppatore ha a disposizione un PC fisso e un portatile per le riunioni.

1.2.2 Test

Le tecnologie utilizzate per eseguire i test sono:

* **Engagent:** test di accettazione e di sistema. Viene verificato che il motore semantico funzioni correttamente con la configurazione generata;
* **Postman:** test di integrazione. Permette di generare delle richieste http alle API sviluppate;
* **nose:** test di unità e integrazione. Ambiente di test specifico di Python. L'estensione nose-cov permette di calcolare il code coverage;
* **pylint:** analisi statica del codice. Test statici per Python;

1.3 Organizzazione del testo

Il primo capitolo contiene una panoramica dell' azienda e le tecnologie utilizzate PAT s.r.l;

Il secondo capitolo contiene la pianificazione progetto;

Il terzo capitolo approfondisce nel dettaglio i requisiti del progetto, descrivendo il processo di analisi che ha portato alla loro stesura;

Il quarto capitolo approfondisce l'architettura del software sviluppato, con il supporto di esempi specifici e schemi ad alto livello;

Il quinto capitolo contiene il resoconto del progetto e considerazioni personali sullo stage.

Capitolo 2

Descrizione dello stage

2.1 Il progetto

Il progetto è nato dalla necessità dell'azienda PAT s.r.ldi automatizzare il processo di configurazione del motore semantico di *Engagent*[g], il quale richiede la creazione manuale di *synset*[g]e *regole*[g]. Questo processo è economicamente fattibile e vantaggioso finché le dimensioni delle regole create sono ridotte, ma il costo cresce esponenzialmente con l'aumentare delle regole.
Più regole rendono più precisa la chatbot, ma incrementano di conseguenza i [g]synset da inserire, prolungando i tempi di compilazione del'*NLP*[g]da qualche giorno a settimane.
Per risolvere questo problema, PAT s.r.lha scomposto il processo nei seguenti *task* da automatizzare:

1. creazione delle *regole*[g];
2. creazione dei synset;
3. raffinamento dei risultati;
4. creazione del file di configurazione *NLP* per il motore semantico;

2.1.1 Creazione delle regole

Questo task è il più difficile da automatizzare, perché richiede la definizione di *match* contenenti categorie correlate tra loro. Inoltre, non esistono delle regole uguali per tutti, ma ogni settore ha regole diverse.
La soluzione è stata trovata nell'intelligenza artificiale, più in particolare nel clustering. Tramite l'analisi di *chat* e *FAQs* archiviate, è possibile generare delle regole allo stato grezzo.
Il problema è la bassa affidabilità dei risultati. Possibili soluzioni:

- più dati in input (poco realizzabile nel breve periodo);
- algoritmo più complesso (in lavorazione);
- raffinamento manuale dei risultati (anche se manuale, richiede meno tempo di creare l'NLP da zero, soluzione migliore nel breve termine);

* raffinamento automatico dei risultati (tramite *POS-tagging*) (buon compromesso tra il raffinamento manuale e le altre due soluzioni).

2.1.2 Creazione dei synset

La creazione dei *synset* è facilmente automatizzabile (se le regole sono già state create), in quanto basta trovare i sinonimi delle categorie.

2.1.3 Raffinamento dei risultati

I risultati dell'algoritmo di clustering devono essere ripuliti da *stop-words* e regole prive di significato.

2.1.4 creazione dell'NLP

Adattamento dell'output dell'algoritmo di clustering al motore semantico di Engagent.

2.2 Obbiettivi Aziendali

L'obbiettivo di automatizzare il processo di configurazione di Engagent non è nato con il progetto di stage, ma qualche anno fa, mentre l'*IA* diventava sempre più popolare. L'azienda attribuì questo compito a un team esterno. Il loro compito consisteva nel sviluppare un algoritmo di intelligenza artificiale, per la generazione di cluster contenenti gli ingredienti essenziali alla configurazione del motore semantico.
Verso l'inizio dell'anno, i progressi fatti da questo team erano convincenti, quindi PAT s.r.laveva l'intenzione di sperimentare l'integrazione di tali risultati con il proprio sistema.

2.3 Obbiettivi personali

Durante la ricerca dell'azienda per lo stage, volevo contribuire a un progetto software in ambito professionale, facendo contemporaneamente i primi passi nel mondo delle intelligenze artificiali.
Il progetto proposto da PAT racchiudeva queste prerogative: sarei stato inserito in un progetto maturo e, con l'aiuto di esperti nel settore, avrei potuto lavorare con degli algoritmi di clustering.

2.4 Pianificazione

Con l'aiuto del tutor aziendale ho redatto il piano di lavoro, che comprende 312 ore distribuite in 8 ore al giorno, per 5 giorni alla settimana (lunedì 24 luglio mi sono dovuto assentare da lavoro, con il consenso del tutor aziendale, per un esame universitario).
La pianificazione ha avuto delle modifiche durante l'avanzare del progetto, vista la sua natura "sperimentale". Per esempio, il linguaggio di programmazione Python è stato accordato assieme al tutor aziendale solamente dopo un'analisi approfondita del problema. Di seguito viene riportata l'ultima versione del piano di lavoro.

* **I settimana:** studio della piattaforma *Engagent*;

* **II settimana:** analisi e stesura del report riguradante il problema descritto in 2.1;
* **III settimana:** ricerca e sperimentazione di possibili soluzioni già esistenti per automatizzare la generazione di sinonimi; analisi e progettazione dell' applicazione NLP-Generator;
* **IV settimana:** preparazione dell'ambiente di lavoro; codifica di NLP-Generator; stesura di test di unità;
* **V settimana:** codifica e miglioramento delle prestazioni di *NLP-Generator*; verfica dei risultati di *NLP-generator* da parte del tutor aziendale
* **VI settimana:** analisi sul miglioramento dei risultati di *NLP-Generator* e codifica; documentazione;
* **VII settimana:** documentazione e validazione;
* **VIII settimana:** collaudo.

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

La progettazione dell'applicazione è iniziata con la stesura dei casi d'uso, supportati da diagrammi dei casi d'uso coerenti con lo standard UML.
I casi d'uso sono aumentati durante tutta la durata dello stage. Durante lo sviluppo, assieme al tutor, venivano individuate nuove funzionalità del programma per migliroare i risultati e aumentare l'automazione.

UC0: Scenario principale

Attori Principali: Utente.

Precondizioni: Il sistema è stato installato correttamente. L'utente ha aperto una *shell* posizionata nella root dell'applicazione. (stato principale del sistema).

Descrizione: Il sistema, tramite CLI (*command line interface*), permette di:

* 1 inserire un file json;
* 2 inserire un file xmlsx;
* 3 personalizzare i parametri in output;
* 4 attivare o disattivare le funzionalità del programma;
* 5 creare una configurazione per il motore semantico di Engagent;
* 6 caricare automaticamente l'output in Engagent;
* 7 inserire in input una configurazione già esistente

.

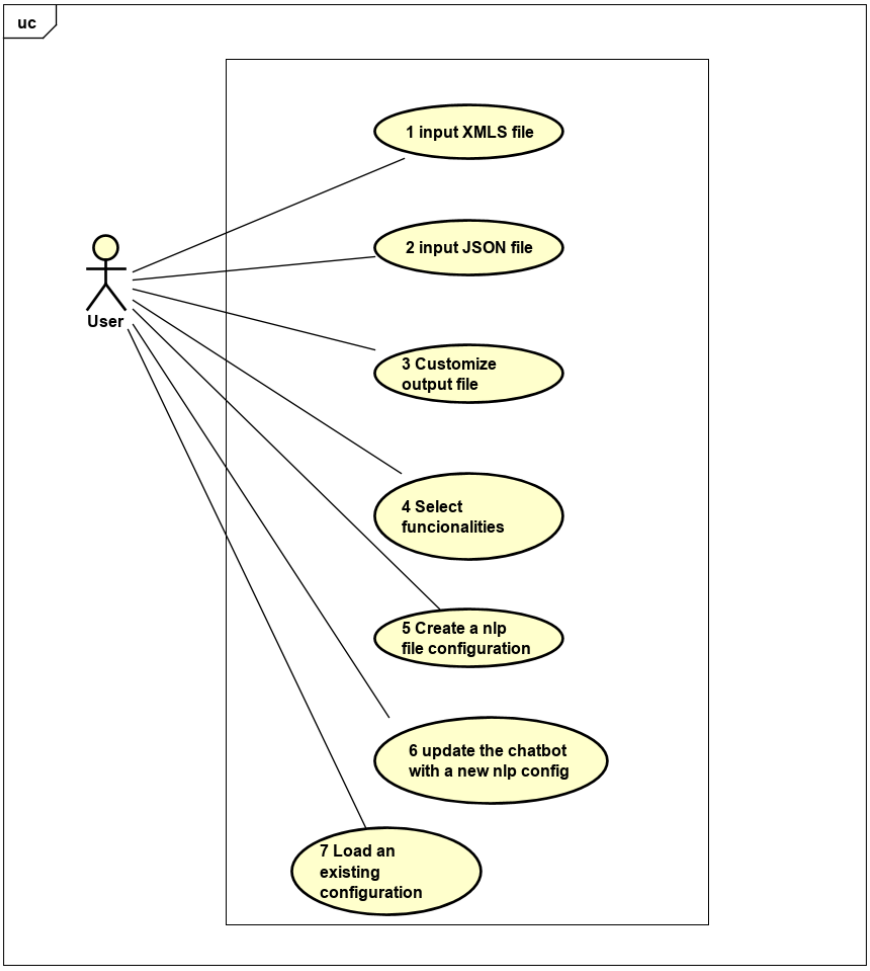


Figura 3.1: Use Case - UC0: Scenario principale

Postcondizioni: Il sistema è pronto per una nuova iterazione.

UC1/2: inserire un file json/xmlsx

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per l'input di un file json/xmlsx.

Descrizione: L'utente, tramite CLI, inserisce un file json/xmlsx.

Postcondizioni: Il sistema permette di inserire un nuovo file in input.

UC3: Personalizzazione parametri in output

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per la personalizzazione

dell'output.

Descrizione: L'utente, tramite CLI, personalizza i parametri di output:

* 3.1 dominio;
* 3.2 lingua;
* 3.3 prefissi;
* 3.4 priorità delle regole.

.

Postcondizioni: Il sistema torna allo stato principale.

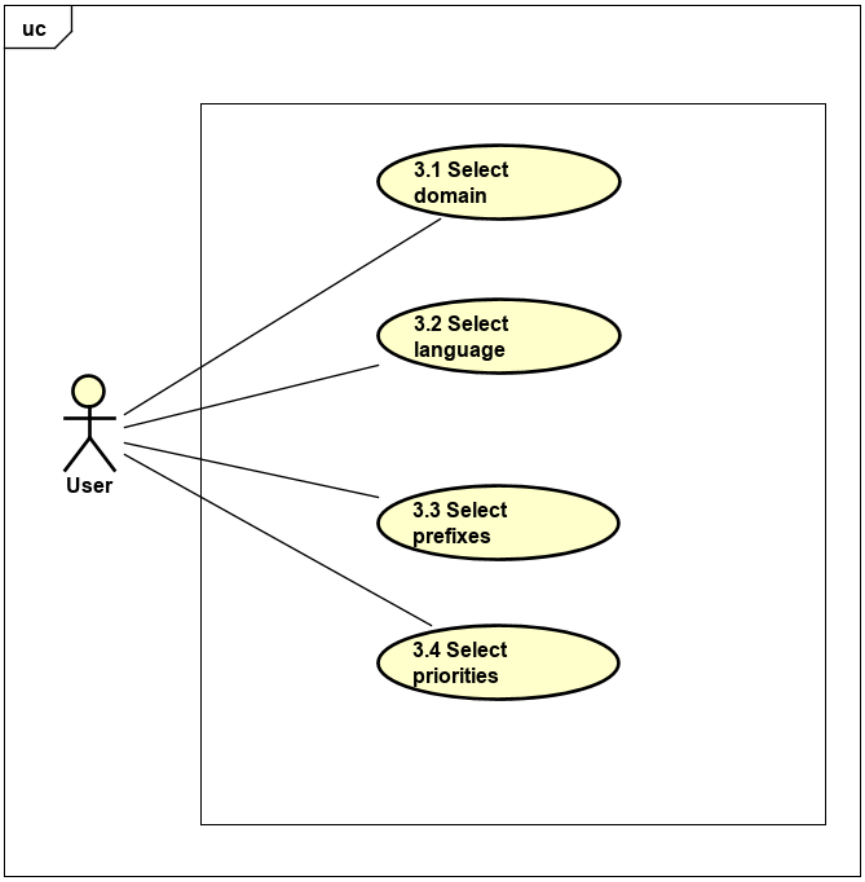


Figura 3.2: Use Case - UC3: Personalizzazione parametri in output

UC3.1, 3.2, 3.3, 3.4: Personalizzazione dominio/lingua/prefissi/priorità delle regole

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per la personalizzazione

dell'output.

Descrizione: L'utente, tramite CLI, modifica il dominio/lingua/prefissi/priorità delle regole.

Postcondizioni: Il sistema permette di personalizzare nuovi parametri.

UC4: Attivare o disattivare le funzionalità del programma

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per selezionare quali funzionalità del programma utilizzare.

Descrizione: L'utente, tramite CLI, attiva o disattiva le seguenti funzionalità:

* 4.1 stemming sulle categorie;
* 4.2 validazione delle categorie attraverso le domande associate;
* 4.3 rimozione delle parole presenti in blacklist;
* 4.4 creazione di *cluster*;
* 4.5 creazione di *intent*.

.

Postcondizioni: Il sistema torna allo stato principale.

UC5: Creare una configurazione compatibile con il motore semantico di Engagent

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per la creazione della configurazione.

Descrizione: L'utente, tramite CLI, crea una nuova configurazione.

Postcondizioni: É stato creato un nuovo file contenente la configurazione. Il sistema è tornato allo stato principale.

UC6: Caricare automaticamente la configurazione in Engagent

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per caricare automaticamente il file contenente la configurare in Engagent.

Descrizione: L'utente, tramite CLI, carica il file contenente la configurazione in Engagent.

Postcondizioni: Engagent contiene la nuova configurazione.

UC7: Inserire una configurazione già esistente

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per l'input di una

configurazione già esistente (vengono ereditate *regole* e *synset*).

Descrizione: L'utente, tramite CLI, carica una configurazione esistente.

Postcondizioni: Il sistema contiene la configurazione di partenza.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Il codice dei requisiti è così strutturato R[O/D][Num. requisito] dove:

R = requisito

O = obbligatorio

D = desiderabile

Tabella 3.1: Tabella del tracciamento dei requisti funzionali

Requisito	Descrizione	Use Case
RO-1	L'utente può inserire un file Excel	UC1
RO-2	L'utente può inserire un file Json	UC2
RO-3	L'utente può personalizzare la configurazione	UC3
RO-3.1	L'utente può modificare il dominio della configurazione	UC3.1
RO-3.2	L'utente può modificare la lingua della configurazione	UC3.2
RO-3.3	L'utente può modificare i prefissi delle regole nella configurazione	UC 3.3
RO-3.4	L'utente può modificare le priorità dei match nella configurazione	UC3.4
RO-4	L'utente può attivare o disattivare le funzionalità del programma	UC4
RO-4.1	L'utente può abilitare lo [g]stemming sulle categorie	UC4.1
RO-4.2	L'utente può disabilitare lo [g]stemming sulle categorie	UC4.1
RO-4.3	Valdiazione delle categorie, attraverso le frasi associate	UC4.2
RO-4.4	valdiazione delle categorie, attraverso le frasi associate	UC4.2
RO-4.5	Creazione dei *cluster*	UC4.3
RO-4.6	Creazione degli *intent*	UC4.5
RO-5	L'utente può creare un file contenente la configurazione	UC5
RO-6	le funzioni più importanti devono essere riutilizzabili	Tutor aziendale
RO-7	L'applicazione deve essere sviluppata in Python 3.7	Tutor aziendale
RD-1	creazione di una configurazione estendendone una già esistente	UC7
RD-2	caricare automaticamente una configurazione in *Engagent*	UC6
RD-3	esecuzione in tempo lineare rispetto alla grandezza del file	Tutor aziendale
RD-4	Il codice deve essere coperto da almeno 80% di test di unità	Tutor aziendale
RD-5	Il codice deve rispettare lo stile pep8	Obiettivo personale

Capitolo 4

Progettazione e codifica

4.1 Tecnologie e strumenti

Python

Python è un linguaggio di programmazione pensato per la ricerca. Per questo linguaggio sono stati sviluppati la maggior parte dei framework che trattano l'intelligenza artificiale (come TensorFlow). Questo vale anche per l'algoritmo di clustering presentato nell'introzione2.1.1.
Assieme al tutor aziendale, abbiamo scelto questo linguaggio per il progetto di stage per più motivi:

* per soddisfare il requisito di modularità del codice (RO-6) è necessario produrre del codice compatibile con l'algoritmo di clustering;
* durante l'analisi del problema, ho scoperto l'esistenza di framework per il postagging e la creazione dei synset sviluppati e ancora supportati in Python. (NLTK, TreeTagger);
* Python non è adatto a progetti di grossa portata, ma va bene per uno di soli due mesi.

NLTK

Framework di python per la creazione di sinonimi, attraverso dizionari italiani e inglese.

TreeTagger

Applicazione per l'estrazione dei lemma dalle parole. Viene adattato in Python attraverso *TreeTaggerWrapper*.

Engagent

Piattaforma sviluppata da PAT s.r.l formata dalla chatbot, il motre semantico e alcuni servizi di supporto. É servita per eseguire i test di sistema e accettazione, in quanto target dei file di configurazione generati dalla applicazione sviluppata durante lo stage.

4.2 Progettazione

Durante lo stage, la progettazione è stata inserita all'interno del Manuale dello sviluppatore (documento in possesso di PAT s.r.l). Di seguito riporto tale progettazione, tralasciando però alcuni dettagli di implementazione, come richiesto dal tutor aziendale.

model

Le classi in [g]model rappresentano una configurazione *NLP*.

NLP: Classe principale di *model*, rappresenta una configurazione NLP. Le altre classi sono dei componenti di questa. Il metodo principale è "to_string", che trasforma un oggetto NLP nella configurazione per *Engagent*.

Rule: Rappresenta una singola regola. Comprende il commento della regola, i *match*, le domande e le risposte.

Synset: Rappresenta un singolo synset. É composta da un titolo, alcuni valori di configurazione e i sinonimi.

Match: Rappresenta un singolo match di una regola. É composta da un insieme di categorie, la priorità del match e un titolo.

builders

Le classi in *builders* permettono di creare un oggetto NLP, senza preoccuparsi della logica di implementazione. Attraverso il design pattern *abstract method*, è possibile derivare la classe NLPBuilder, per creare builders che lavorano con formati diversi da JSON e XLSX.

NLPBuilder: Classe astratta per la creazione di un NLP. Contiene la logica principale di creazione delle configurazioni. Le classi che estendono questa classe devono solamente implementare i metodi astratti per standardizzare l'input (come definito nei commenti al codice e nel manuale dello sviluppatore).

NLPBuilderXLSX: Classe che estende NLPBuilder. Standardizza l'input nel formato xlsx (excel).

NLPBuilderJSON: Classe che estende NLPBuilder. Standardizza l'input nel formato json.

utils

Contiene moduli e classi di utilità.

SynsetGenerator: Permette la creazione di sinonimi a partire da una parola qualsiasi. Contiene gli algoritmi più importanti dell'applicazione per PAT s.r.l

Utils: Modulo che contiene funzioni di utilità, utilizzate da più classi non dipendenti tra di loro.

NLPStemmer: Esegue lo stemming su un oggetto di tipo NLP.

interface

Interfaccia dell'applicazione per l'utente. Permette l'interazione programmatica e a linea di comando.

api: Permette l'interazione programmatica con l'applicazione

cli: Permette l'interazione a linea di comando con l'applicazione

exceptions

Eccezioni personalizzate per l'applicazione.

EmptyCategoryException: Durante l'esecuzione del programma, è stata trovata una categoria vuota.

EmptySynonymException: Durante l'esecuzione del programma, è stato trovato un sinonimo vuoto.

4.3 Design Pattern utilizzati

L'applicazione è stata sviluppata utilizzando i seguenti design pattern:

* Builder Pattern: la creazione di un oggetto NLP può essere complicata, perché composta da almeno quattro componenti diverse. Il builder pattern permette di semplificare questo compito, rendendo di conseguenza le classi in *model* meno complesse;
* Abstract Pattern: permette di aggiungere nuovi formati in input all'applicazione senza dover riscrivere l'intera logica di creazione dell'NLP.

4.4 Codifica

La codifica si è intervallata con periodi di progettazione di dettaglio e analisi delle nuove richieste del tutor. Per ogni nuova componente da sviluppare, ho seguito questi passaggi:

* analisi e progettazione di dettaglio del problema;
* ricerca di soluzioni già esistenti per questo problema;
* codifica e sviluppo di test di unità specifici;
* esecuzione di tutti i test di unità e risoluzione di eventuali *bug*;
* verifica da parte del tutor aziendale;
* risoluzione di eventuali errori logici.

Capitolo 5

Conclusioni

5.1 Consuntivo finale

5.2 Raggiungimento degli obiettivi

5.3 Conoscenze acquisite

5.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia