

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

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 obiettivi 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 post-tagging.

Ringraziamenti

Innanzitutto, vorrei ringraziare 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 un magnifico stage in PAT s.r.l.

Desidero ringraziare con affetto i miei genitori e mio fratello, per il sostegno morale dimostrato durante questi anni.

Ringrazio tutti i compagni di corso con i quali ho svolto i progetti universitari, per aver svolto un eccellente lavoro.

Padova, Settembre 2019

Stefano Zanatta

Indice

1	Introduzione	1
1.1	L'azienda	1
1.1.1	Prodotti e servizi	1
1.1.2	Organizzazione e Metodo di lavoro	2
1.2	Strumenti e Tecnologie	3
1.2.1	Ambiente di lavoro	3
1.2.2	Linguaggi di programmazione	3
1.3	Organizzazione del testo	3
2	Descrizione dello stage	5
2.1	Il progetto	5
2.2	Obiettivi Aziendali	5
2.3	Obiettivi personali	5
2.4	Pianificazione	6
3	Analisi dei requisiti	7
3.1	Analisi del problema	7
3.2	Casi d'uso	8
3.3	Tracciamento dei requisiti	13
4	Progettazione e codifica	15
4.1	Framework e tecnologie per lo sviluppo	15
4.1.1	pip - PyPI	17
4.2	Progettazione	17
4.2.1	Struttura dell'applicazione	17
4.3	Design Pattern utilizzati	20
4.4	Codifica	21
4.4.1	Task della codifica	21
4.4.2	Stile del codice	21
4.4.3	Codice significativo	21
5	Conclusioni	25
5.1	Consuntivo	25
5.1.1	Requisiti	25
5.1.2	Metriche del codice	27
5.1.3	User satisfaction	28
5.2	Raggiungimento degli obiettivi	28
5.3	Conoscenze acquisite	28
5.4	Valutazione personale	28

Bibliografia

31

Elenco delle figure

1.1	logo <i>PAT s.r.l</i>	1
1.2	logo Engagent	2
1.3	Agile	2
3.1	Use Case - UC0: Scenario principale	9
3.2	Use Case - UC3: Personalizzazione parametri in output	10
4.1	nose	15
4.2	Pylint	16
4.3	NLTK	16
4.4	Diagramma dei package	17
4.5	model	18
4.6	builders	19
4.7	utils	20
5.1	Requisiti soddisfatti	26
5.2	Requisiti totali	26
5.3	code coverage	27
5.4	Test di unità e integrazione	27
5.5	pycodestyle	28

Elenco delle tabelle

3.1	Tracciamento dei requisiti funzionali	14
3.2	Tracciamento dei requisiti di vincolo	14
5.1	requisiti soddisfatti	25

5.1	requisiti soddisfatti	26
-----	---------------------------------	----

Capitolo 1

Introduzione

1.1 L'azienda

PAT s.r.l è 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.



Figura 1.1: logo *PAT s.r.l*

1.1.1 Prodotti e servizi

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

- * Engagent: chatbot semi-automatizzata per uso professionale, l'unico prodotto di *PAT s.r.l* con cui sono entrato in contatto;
- * Helpdesk: service *desk*^[g];
- * 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 estrae la categoria della domanda e la inoltra all'operatore adeguato.



Figura 1.2: logo Engagent

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.l* seguono una metodologia Agile¹. Questa fa parte delle metodologie iterative, caratterizzata da brevi iterazioni (o sprint, di circa 3-4 settimane) seguite dalla *review* del lavoro svolto. Il focus principale si trova nel cliente: ci deve essere una interazione costante per capire quali sono le *feature* più importanti, che hanno precedenza sulle altre.

Il team a cui ho preso parte applica questa metodologia. Il contatto con il cliente è frequente, che sia manutenzione o nuove *features* da implementare. 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 riguarda, ho adottato senza difficoltà queste metodologie, perché molto simili a quelle utilizzate durante il progetto di ingegneria del software.

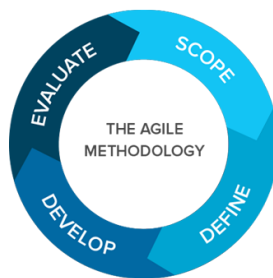


Figura 1.3: Agile

¹Manifesto Agile. URL: <http://agilemanifesto.org/iso/it/>.

1.2 Strumenti e Tecnologie

Questa sezione descrive, ad alto livello, le tecnologie utilizzate durante lo stage.

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 documenti, 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 Linguaggi di programmazione

I linguaggi di programmazione che ho utilizzato sono:

- * **Python:** per la codifica dell'applicazione;
- * **Shell:** per la creazione di script per automatizzare i seguenti task:
 - esecuzione del programma tramite linea di comando;
 - pulizia della cache del programma;
 - pulizia dei file di output del programma.

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.l* di 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 *synset*^[g] da inserire, prolungando i tempi di compilazione dell'*NLP*^[g] da qualche giorno a settimane.

2.2 Obiettivi Aziendali

L'obiettivo di automatizzare il processo di configurazione di Engagent non è nato con il progetto di stage, ma qualche anno fa, mentre il *Machine Learning* diventava sempre più popolare. *PAT s.r.l* attribuì questo compito a un team esterno. Il loro compito consisteva nell'implementazione di un algoritmo di *ML*, per la generazione di cluster contenenti gli ingredienti essenziali alla configurazione di Engagent. Verso l'inizio dell'anno, i progressi fatti da questo team erano convincenti, quindi *PAT s.r.l* aveva l'intenzione di sperimentare l'integrazione di tali risultati con il proprio sistema.

Dallo stage, l'azienda si aspettava i seguenti prodotti:

- * **Studio di fattibilità:** analisi del problema³ e *scouting*^[g] di soluzioni già esistenti;
- * **NLP Generator:** un'applicazione per la creazione del modello per il motore semantico di Engagent;
- * **manuale:** manuale tecnico per l'utilizzo e la manutenzione dell'applicazione.

2.3 Obiettivi 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 di un report, riguardante il problema della creazione automatica del file di configurazione^{2.1};
- preparazione dell'ambiente di lavoro;

*** III settimana:**

- ricerca e sperimentazione di possibili soluzioni già esistenti per automatizzare la generazione di sinonimi;
- analisi e progettazione (al alto livello) di NLP Generator;
- progettazione di dettaglio e implementazione del model^{4.2.1};

*** IV settimana:**

- implementazione di NLP Generator;
- stesura di test di unità;

*** V settimana:**

- implementazione e miglioramento delle prestazioni di *NLP Generator*;
- verifica dei risultati di *NLP Generator* da parte del tutor aziendale;

*** VI settimana:**

- analisi per il miglioramento dei risultati di *NLP Generator*
- progettazione di dettaglio e implementazione;
- documentazione;

*** VII settimana:**

- documentazione e validazione;

*** VIII settimana:**

- collaudo.

Capitolo 3

Analisi dei requisiti

3.1 Analisi del problema

Per risolvere il problema descritto in nella descrizione del progetto^{2.1}, ovvero l'automazione della configurazione del motore semantico di Engagent, ho scomposto tale problema nei seguenti *macro-task*, quindi ho studiato come automatizzarli:

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

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*^[g] archiviate, è possibile generare delle regole allo stato grezzo.

Rimane il problema della bassa affidabilità dei risultati. Possibili soluzioni:

- * **più dati in input:** non realizzabile nel breve periodo;
- * **algoritmo più complesso:** task attribuito a ricercatori esterni all'azienda;
- * **raffinamento manuale dei risultati:** richiede meno tempo rispetto alla creazione dell'*NLP* da zero. Questa soluzione è la più semplice nel breve periodo;
- * **raffinamento automatico dei risultati:** buon compromesso tra il raffinamento manuale e le altre due soluzioni.

Creazione dei synset

La creazione dei *synset* è automatizzabile trovando i sinonimi delle categorie che compongono le regole. Ho scomposto questo *macro-task* nei seguenti *task*:

- * estrazione delle parole che compongono le regole in una lista;
- * portare la parola a una forma standard (attraverso *TreeTagger*);
- * per ogni parola, trovare i suoi sinonimi (attraverso *NLTK Wordnet*).

Raffinamento dei risultati

I risultati dell'algoritmo di clustering devono essere ripuliti da *stop-words*^[6] e regole prive di significato. Le *stop-words* sono una lista di parole che non possono far parte del modello, definite da *PAT s.r.l.* Una regola è significativa quando permette di individuare un gruppo di domande con lo stesso significato.

creazione del modello NLP

La creazione del modello per il motore semantico di Engagent richiede di aver creato delle regole e *synset* corretti. Inoltre, per garantire flessibilità all'applicazione, è necessario permettere la manipolazione del modello attraverso l'aggiunta, la modifica e la rimozione di elementi in ogni momento.

3.2 Casi d'uso

Ho preceduto la progettazione dell'applicazione dalla 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, abbiamo individuato nuove funzionalità per raffinare i risultati e migliorare l'automazione di NLP Generator.

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 xlsx;
- * 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;
- * 8 utilizzo dell'applicazione attraverso una *REST API*;
- * 9 visualizzazione delle informazione di esecuzione del sistema (*log*).

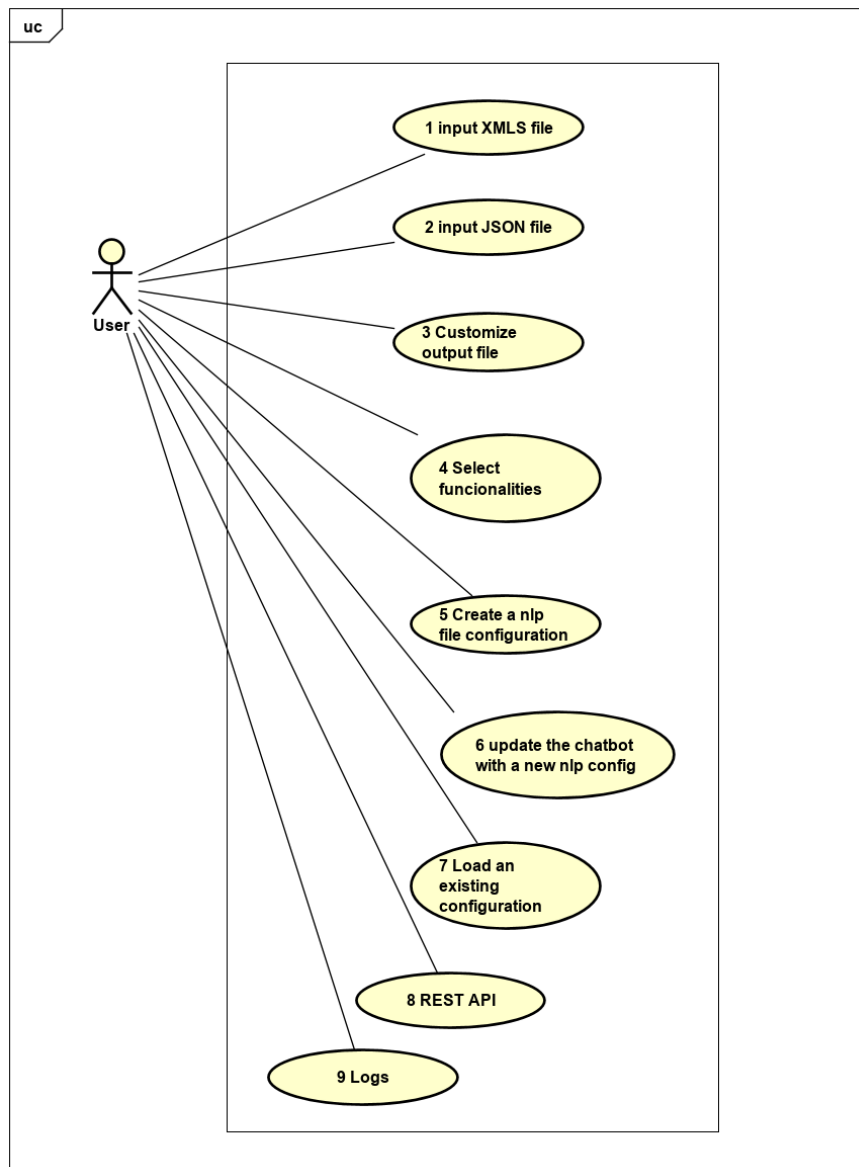


Figura 3.1: Use Case - UC0: Scenario principale

Postcondizioni: Il sistema è pronto per una nuova iterazione.

UC1/2: inserire un file json/xlsx

Attori Principali: Utente.

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

json/xlsx.

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

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 modifica del dominio;
- * 3.2 modifica della lingua;
- * 3.3 modifica dei prefissi;
- * 3.4 modifica della priorità delle regole.

Postcondizioni: Il sistema torna allo stato principale.

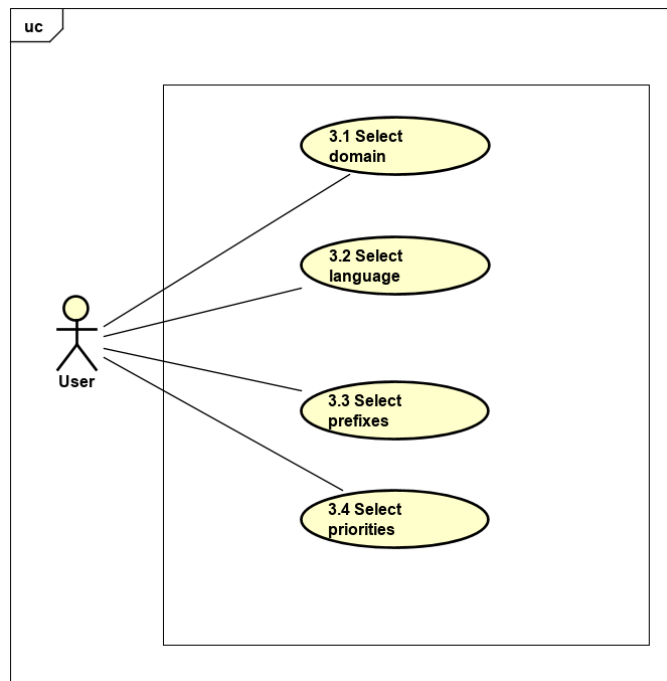


Figura 3.2: Use Case - UC3: Personalizzazione parametri in output

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: 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 stopwords^[g];
- * 4.4 creazione di *cluster*;
- * 4.5 creazione di *intent*.

.

Postcondizioni: Il sistema torna allo stato principale.

UC4.1: Stemming sulle categorie

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per l'attivazione (o disattivazione) dello stemming. Lo stemming consiste nell'estrazione della radice da una parola..

Descrizione: L'utente, tramite CLI, attiva o disattiva lo stemming.

Postcondizioni: Il sistema permette all'utente di modificare altre funzionalità.

UC4.2: Validazione delle categorie attraverso le domande associate

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per l'attivazione (o disattivazione) della validazione delle categorie. Questa funzione permette di filtrare le categorie, escludendo quelle assenti nelle domande associate, aumentando la precisione del risultato..

Descrizione: L'utente, tramite CLI, attiva o disattiva la validazione delle categorie.

Postcondizioni: Il sistema permette all'utente di modificare altre funzionalità.

UC4.3: rimozione delle *stopwords*

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per l'attivazione (o disattivazione) del filtro delle *stopwords*. Le stopwords sono delle parole che il sistema deve ignorare durante l'esecuzione del programma..

Descrizione: L'utente, tramite CLI, attiva o disattiva il filtro delle stopwords.

Postcondizioni: Il sistema permette all'utente di modificare altre funzionalità.

UC4.4: Creazione dei cluster

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per l'attivazione (o disattivazione) della creazione dei *cluster*. I *cluster* devono essere definiti nel file di input..

Descrizione: L'utente, tramite CLI, attiva o disattiva la creazione dei cluster.

Postcondizioni: Il sistema permette all'utente di modificare altre funzionalità.

UC4.5: Creazione degli intent

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per l'attivazione (o disattivazione) della creazione degli *intent*. Un intent è una regola composta da un singolo match..

Descrizione: L'utente, tramite CLI, attiva o disattiva la creazione degli intent.

Postcondizioni: Il sistema permette all'utente di modificare altre funzionalità.

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: È 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 configurazione 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.

UC8: Utilizzo dell'applicazione attraverso una *REST API*

Attori Principali: Utente.

Precondizioni: L'applicazione espone dei comandi REST per l'esecuzione dell'applicazione in remoto.

Descrizione: L'utente esegue dei comandi REST indirizzati a un server remoto.

Postcondizioni: Il sistema permette l'esecuzione di nuovi comandi REST.

UC9: visualizzazione delle log

Attori Principali: Utente.

Precondizioni: L'applicazione permette la generazione dei *log* durante l'esecuzione.

Descrizione: L'utente esegue l'applicazione.

Postcondizioni: Il sistema genera automaticamente i *log* e li salva in un *database*.

3.3 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][F/V][Num. requisito] dove:

R = requisito

O = obbligatorio

D = desiderabile

F = Funzionale

V = Vincolo

Requisito	Descrizione	Fonte
ROF-1	L'utente può inserire un file Excel	UC1
ROF-2	L'utente può inserire un file Json	UC2
ROF-3	L'utente può personalizzare la configurazione	UC3
ROF-3.1	L'utente può modificare il dominio della configurazione	UC3.1
ROF-3.2	L'utente può modificare la lingua della configurazione	UC3.2
ROF-3.3	L'utente può modificare i prefissi delle regole nella configurazione	UC 3.3
ROF-3.4	L'utente può modificare le priorità dei match nella configurazione	UC3.4
ROF-4	L'utente può attivare o disattivare le funzionalità del programma	UC4
ROF-4.1	L'utente eseguire lo <i>ls</i> stemming sulle categorie	UC4.1
ROF-4.2	Validazione delle categorie, attraverso le frasi associate	UC4.2
ROF-4.3	L'utente filtrare le <i>stopwords</i>	UC4.3
ROF-4.4	L'utente può creare i <i>cluster</i>	UC4.4
ROF-4.5	L'utente creare gli <i>intent</i>	UC4.5
ROF-5	L'utente può creare un file contenente la configurazione	UC5
RDF-1	creazione di una configurazione estendendone una già esistente	UC7
RDF-2	caricare automaticamente una configurazione in <i>Engagent</i>	UC6
RDF-3	L'applicazione deve esporre un servizio REST API	UC8
RDF-4	L'applicazione genera i <i>log</i>	UC9

Tabella 3.1: Tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
ROV-1	le funzioni più importanti devono essere riutilizzabili	Tutor aziendale
ROV-2	L'applicazione deve essere sviluppata in Python 3.7	Tutor aziendale
ROV-3	esecuzione in tempo lineare rispetto alla grandezza del file	Tutor aziendale
ROV-4	Il codice deve essere coperto da almeno 80% di test di unità	Tutor aziendale
ROV-5	Il codice deve rispettare lo stile pep8	Obiettivo personale

Tabella 3.2: Tracciamento dei requisiti di vincolo

Capitolo 4

Progettazione e codifica

4.1 Framework e tecnologie per lo sviluppo

Python

Python¹ 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'introduzione^{3.1}.

Assieme al tutor aziendale, abbiamo scelto questo linguaggio per i seguenti motivi:

- * permette di soddisfare il requisito di modularità del codice (requisito RO-6): il codice prodotto è compatibile con quello dell'algoritmo di clustering;
- * possiede dei framework per il pos-tagging e la generazione di sinonimi (NLTK e TreeTagger).;

Nose

Nose² è un framework per l'esecuzione di Test di unità e integrazione in Python. L'estensione nose-cov permette di calcolare il code coverage.

Nella maggior parte dei casi, ho utilizzato il tool automatico di *Visual Studio Code* per eseguire i test. Con l'opzione di eseguire i test a ogni salvataggio, è possibile accorgersi subito se sono stati inseriti dei *bug*. Per calcolare il ^[g]code coverage è necessario eseguire il seguente comando:

```
$ nosetests --with-cov --cov src tests/
```

dove *src* è la cartella contenente il codice.



Figura 4.1: nose

¹Python. URL: <https://www.python.org/>.

²Nose. URL: <https://nose.readthedocs.io/en/latest/>.

pycodestyle (pep8) - pylint

PEP8³ esegue l'analisi statica del codice per Python. Rileva, a ogni salvataggio, errori di formattazione e di stile nel codice. Questo strumento mi ha permesso di mantenere un codice pulito, rispettando lo standard *PEP8*^[g].



Figura 4.2: Pylint

NLTK - Wordnet

NLTK⁴ è un framework di Python per processare il linguaggio umano, attraverso dizionari in diverse lingue.

Wordnet⁵ è una libreria di NLTK, per l'estrazione dei sinonimi dalle parole.



Figura 4.3: NLTK

TreeTagger

TreeTagger⁶ è un'applicazione per l'estrazione dei lemma dalle parole. *TreeTaggerWrapper*⁷ permette l'utilizzo di questa applicazione in modo programmatico in Python.

Engagent

Engagent⁸ è una piattaforma sviluppata da *PAT s.r.l* formata dalla chatbot, il motore semantico e servizi di supporto. L'ho utilizzata principalmente per eseguire i test di sistema e accettazione, in quanto target dei file di configurazione generati da NLP Generator.

³PEP8. URL: <https://www.python.org/dev/peps/pep-0008/>.

⁴NLTK - Natural Language Toolkit. URL: <http://www.nltk.org/>.

⁵Wordnet. URL: <http://www.nltk.org/howto/wordnet.html>.

⁶TreeTagger. URL: <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/>.

⁷TreeTagger wrapper. URL: <https://treetaggerwrapper.readthedocs.io/en/latest/>.

⁸Engagent. URL: <https://www.pat.eu/prodotti-software-e-soluzioni-it/engagent/>.

4.1.1 pip - PyPI

pip⁹ è un sistema di gestione di pacchetti di Python. Semplifica il processo di installazione, aggiornamento e rimozione di pacchetti Python, attraverso semplici comandi. Permette il tracciamento delle dipendenze utilizzate dal programma. Tramite *pip-env*, è possibile utilizzare un ambiente virtuale per garantire la portabilità dell'applicazione. *pip* mi ha permesso di eseguire efficientemente i seguenti task:

- * individuazione, installazione e tracciamento dei pacchetti di Python;
- * installazione dell'applicazione nel server aziendale remoto, contenente il sistema operativo Linux.

4.2 Progettazione

4.2.1 Struttura dell'applicazione

NLP Generator è composto principalmente da due parti:

- * configurazione dell'NLP (package *model*);
- * costruttori dell'NLP (package *builders*).

Gli altri package contengono funzionalità di supporto ai primi due:

- * funzioni di utilità (package *utils*);
- * funzioni per l'interfacciamento con gli utenti (package *interface*);
- * eccezioni (package *exceptions*).

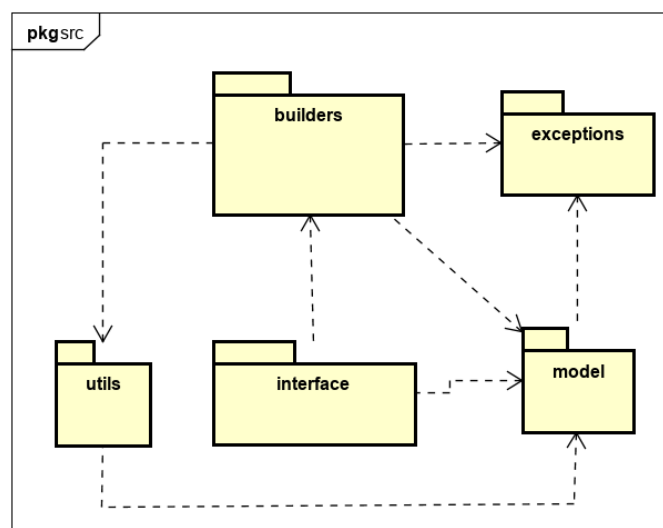


Figura 4.4: Diagramma dei package

⁹ *PyPI*.

model

Le classi in *model*^[g] rappresentano una configurazione *NLP*. La progettazione del **model** mi ha richiesto lo studio approfondito del motore semantico di *Engagent*, in particolare dei suoi file di configurazione. Ho progettato questo *package* includendo tutte le informazioni presenti nel file di configurazione del motore semantico, per facilitare la manutenzione perfetta del *software*.

Tutte le altre funzionalità del programma dipendono da questa (direttamente o indirettamente), per questo motivo le ho dato maggiore priorità.

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. È composta da un titolo, alcuni valori di configurazione e i sinonimi.

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

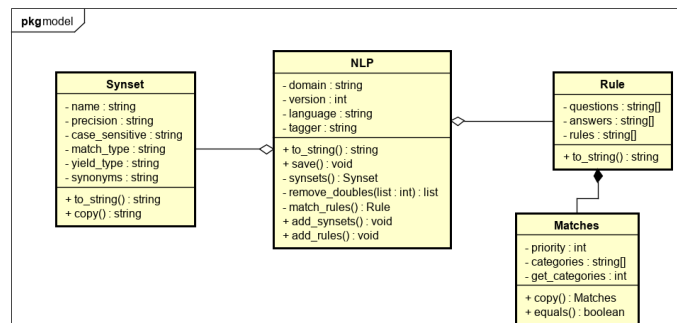


Figura 4.5: model

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.

I builder sono la parte del programma da estendere per aggiungere nuovi tipi di input (oltre a formati JSON ed Excel). Ho posto particolare attenzione durante la progettazione della classe astratta *NLPBuilder*, riducendo al minimo la logica richiesta dalle sue implementazioni, commentando nel dettaglio le sue funzioni astratte (con esempi) e dedicandole una sezione dedicata nel manuale.

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.

Nel diagramma è stato inserito anche il package *model* per specificare cosa crea ogni builder.

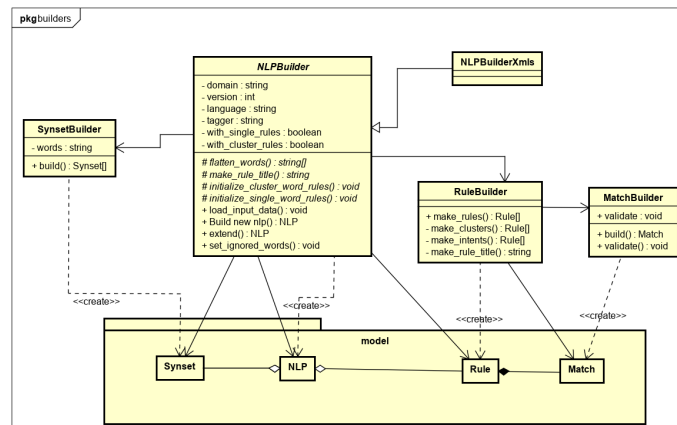


Figura 4.6: builders

utils

Contiene moduli e classi di utilità.

SynsetGenerator: Questa classe contiene la logica di *business* del programma, ovvero quella che esegue la vera trasformazione dell'input in *synset* e regole (a differenza del *model*, che si limita a tradurre tali risultati in qualcosa di compatibile con Engagent). Questa classe utilizza la libreria NLTK e TreeTagger.

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.

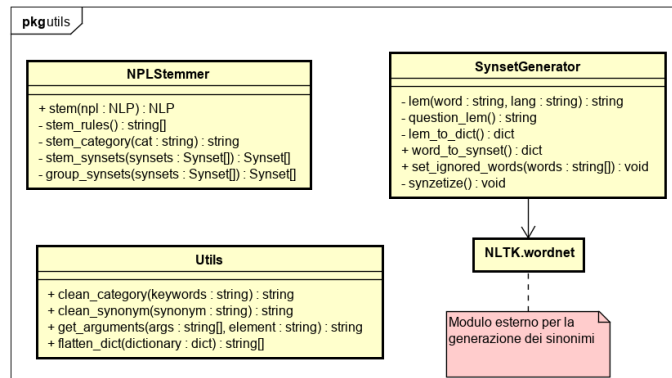


Figura 4.7: utils

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

4.4.1 Task della codifica

La codifica è stata intervallata da periodi di progettazione e analisi delle nuove richieste del tutor.

Per ogni nuova componente da implementare, ho seguito questi passaggi:

- * analisi e progettazione di dettaglio del problema;
- * ricerca di soluzioni già esistenti per questo problema;
- * codifica di quanto progettato 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.

4.4.2 Stile del codice

Per facilitare il lavoro di chi dovrà mantenere il progetto, ho seguito le linee guida definite in *PEP8*¹⁰ per la stesura del codice:

- * i metodi più significativi sono documentati con il seguente commento:

```
"""[Descrizione]

Arguments:
    arg1 {[tipo]} -- [descrizione]
Returns:
    [tipo] -- [descrizione]
Raises:
    [exception] -- [descrizione]
"""
```

- * le variabili private iniziano con un doppio underscore '`__`';
- * le variabili protette iniziano con un singolo underscore '`_`';
- * le variabili sono scritte interamente in minuscolo, variabili composte da più parole sono separate da underscore;
- * le costanti globali sono scritte in maiuscolo

4.4.3 Codice significativo

Di seguito, ho riportato alcuni esempi di codice significativo.

¹⁰ *PEP8*.

Implementazione dell'*abstract method pattern*

L'*abstract method pattern* è stato utilizzato nella classe NLPBuilder. Il ruolo dei metodi astratti è lasciare alle classi derivate il compito di implementare la logica di trasformazione dell'input in un formato standard. Questo è il minimo necessario richiesto per implementare un nuovo builder (per un nuovo formato di input).

L'implementazione del metodo `load_input_data` richiede la creazione di una variabile contenente il contenuto del file di input. Questa variabile verrà utilizzata solamente dalle implementazioni degli altri metodi astratti, quindi non è predefinita.

```
@abstractmethod
def load_input_data(self, path: str):
    """load the configuration from a file

    Arguments:
        path {str} -- relative path to the file
    """
    pass
```

Il metodo `_flatten_words` estrae le categorie dall'input definito in `load_input_data`.

```
@abstractmethod
def _flatten_words(self) -> list:
    """Flatten a list of categories in the input variable
    defined by load_input_data.

    Returns:
        words {list} -- of categories. E.g. ['dog','cat',
        plain','airplain']
    """
    pass
```

Gli altri metodi astratti definiti in NLPBuilder seguono una logica simile.

Implementazione del *builder pattern*

Il metodo principale del *builder* è il seguente. Questo inizializza i due tipi di regole (se richiesti), crea le regole, i *synset* e alla fine l'oggetto di configurazione NLP; se richiesto, viene applicato lo stemming.

```
def build(self) -> NLP:
    """Create a new NLP object.

    Returns:
        {NLP} -- new NLP object
    """
    intent_rules, cluster_rules = None, None
    if self.__with_cluster_rules:
        cluster_rules = self._initialize_cluster_rules()
    if self.__with_intent_rules:
        intent_rules = self._initialize_intent_rules()
    rules = self._rule_builder.make_rules(
        cluster_rules=cluster_rules, intent_rules=intent_rules)
```

```
synsets = self.__synset_builder.build()
nlp = NLP(domain=self.__domain, version=self.__version,
          language=self.__language, tagger=self.__tagger,
          synsets=synsets, match_rules=rules)

if self.__stem:
    nlp = self.stemmer.stem(nlp)
```


Capitolo 5

Conclusioni

5.1 Consuntivo

Questa sezione descrive i risultati dello stage sotto forma di metriche. Nel complesso, il progetto si è concluso positivamente.

5.1.1 Requisiti

Risultato: Passato

L'applicazione soddisfa tutti i requisiti obbligatori (funzionali e vincolo)^{3.3} (19/19) e metà dei requisiti desiderabili (2/4). I requisiti desiderabili non implementati, per questioni di tempo, sono RD-3 e RD-4^{3.3}, ovvero l'esposizione dell'applicazione attraverso un'API e la generazione delle informazioni di compilazione. Ho implementato questi requisiti in un breve periodo di post-stage in azienda.

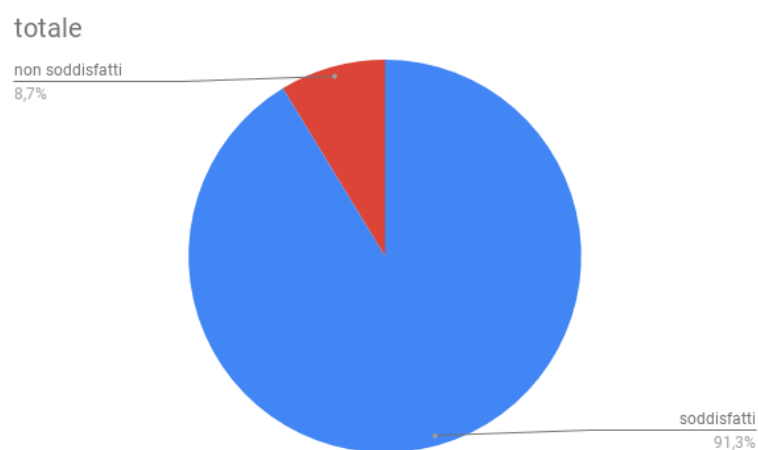
Tabella 5.1: requisiti soddisfatti

Requisito	Risultato
ROF-1	soddisfatto
ROF-2	soddisfatto
ROF-3	soddisfatto
ROF-3.1	soddisfatto
ROF-3.2	soddisfatto
ROF-3.3	soddisfatto
ROF-3.4	soddisfatto
ROF-4	soddisfatto
ROF-4.1	soddisfatto
ROF-4.2	soddisfatto
ROF-4.3	soddisfatto
ROF-4.4	soddisfatto
ROF-4.5	soddisfatto
ROF-5	soddisfatto
RDF-1	soddisfatto
RDF-2	soddisfatto
RDF-3	non soddisfatto
RDF-4	non soddisfatto
ROV-1	soddisfatto

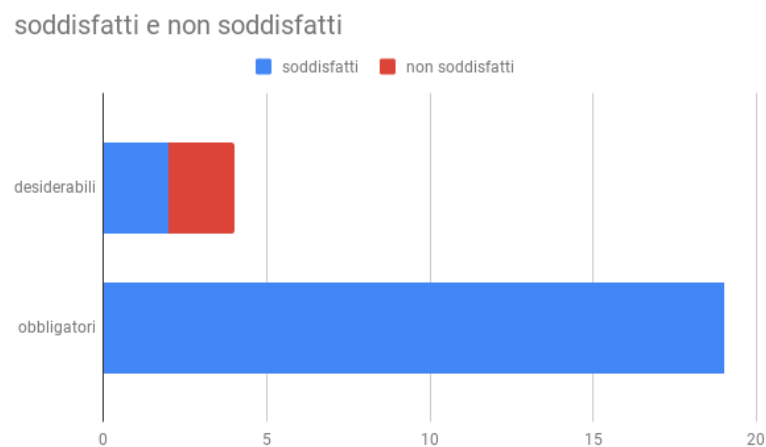
Tabella 5.1: requisiti soddisfatti

Requisito	Risultato
ROV-2	soddisfatto
ROV-3	soddisfatto
ROV-4	soddisfatto

Il seguente grafico rappresenta la relazione tra requisiti totali soddisfatti e requisiti non soddisfatti.

**Figura 5.1:** Requisiti soddisfatti

Il seguente grafico rappresenta la relazione tra requisiti soddisfatti e non soddisfatti, separati per desiderabili e obbligatori.

**Figura 5.2:** Requisiti totali

5.1.2 Metriche del codice

Code coverage

Risultato: Passato

Il *code coverage* è pari all'87%, superiore dell'80% richiesto dal requisito ROV-4^{3.3}. I metodi non coperti sono molto semplici e non necessitano di essere testati.

```
----- coverage: platform win32, python 3.7.2-final-0 -----
```

Name	Stmts	Miss	Cover
src__init__.py	0	0	100%
src\builders__init__.py	0	0	100%
src\builders\match_builder.py	24	3	88%
src\builders\nlp_builder.py	97	10	90%
src\builders\nlp_builder_xlsx.py	50	3	94%
src\builders\rule_builder.py	66	8	88%
src\builders\synset_builder.py	24	0	100%
src\exceptions__init__.py	0	0	100%
src\exceptions\exceptions.py	6	0	100%
src\interface__init__.py	0	0	100%
src\interface\api.py	68	21	69%
src\interface\cli.py	35	35	0%
src\interface\config.py	5	0	100%
src\model__init__.py	0	0	100%
src\model\match.py	17	1	94%
src\model\nlp.py	90	5	94%
src\model\rule.py	58	4	93%
src\model\synset.py	39	1	97%
src\wordnet_utils__init__.py	0	0	100%
src\wordnet_utils\config_to_nlp.py	88	0	100%
src\wordnet_utils\nlp_stemmer.py	93	9	90%
src\wordnet_utils\synset_generator.py	103	12	88%
src\wordnet_utils\utils.py	42	6	86%
TOTAL	905	118	87%

```
-----
Ran 12 tests in 13.111s
```

Figura 5.3: code coverage

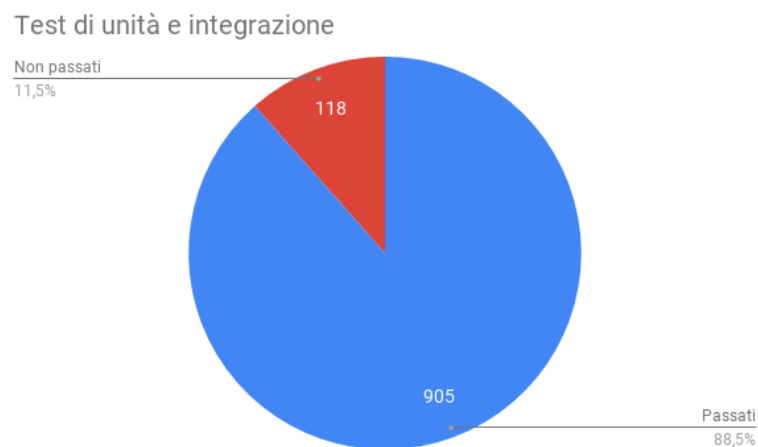
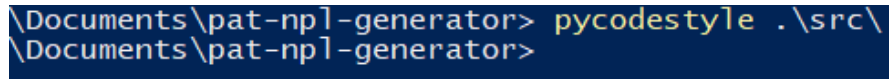


Figura 5.4: Test di unità e integrazione

Analisi statica

Risultato: Passato

Durante la codifica, ho utilizzato *pycodestyle* per rispettare lo stile definito da PEP8, come da metrica ROV-5^{3.3}. Questo ha portato a un codice privo di *warning* da parte di *pycodestyle*, relativi allo stile PEP8.



```

\Documents\pat-npl-generator> pycodestyle .\src\
\Documents\pat-npl-generator>

```

Figura 5.5: *pycodestyle*

5.1.3 User satisfaction

Risultato: Passato

Questa metrica, seppure meno oggettiva delle precedenti, è basata sulla riunione tecnica tenuta con i ricercatori della Zucchetti S.R.L e il tutor aziendale.

L'incontro è avvenuto l'ultima settimana di stage, per capire come integrare NLP Generator alla loro applicazione. Attraverso dei diagrammi UML, ho spiegato il funzionamento degli algoritmi implementati.

La presentazione si è conclusa con successo. I ricercatori hanno capito come migliorare l'output della loro applicazione, fornito in input a NLP Generator.

Il tutor aziendale ha apprezzato particolarmente l'automatismo fornito da NLP Generator. Per questo motivo, ha deciso di integrarlo ad *Engagent* attraverso una REST API.

5.2 Raggiungimento degli obiettivi

5.3 Conoscenze acquisite

- * gestione di un progetto in **python**, attraverso ambienti virtuali (per la gestione delle dipendenze) e principali librerie;
- * importanza e il ruolo del *pre-processing* dei dati, prima di essere elaborati da un algoritmo di machine learning. Nel mondo reale, i dati sono pochi e preziosi, quindi è necessario sfruttarli al massimo e assicurarsi che l'output dell'algoritmo di *machine learning* sia filtrato e analizzato, prima di essere esposto agli utenti del servizio.
Questo si contrappone con l'esperienza universitaria, dove il focus principale viene posto sul metodo, invece che sul risultato.
- * presentazione del software orientata a esperti (tecnica) e agli utenti (funzionale).
- * ho potuto osservare nella pratica come funziona una *software house* di piccole dimensioni, iniziando ad acquisire la professionalità richiesta per questo ambiente;

5.4 Valutazione personale

Prima di iniziare lo stage, sapevo che avrei voluto continuare il mio percorso di studi con la Laurea Magistrale, quindi ho considerato questa esperienza come un mezzo per

imparare qualcosa di nuovo, invece di applicare banalmente quello che ho imparato negli anni.

Ho potuto, per la prima volta, partecipare a dei colloqui di lavoro, qualcosa di cui ho sempre avuto timore, ma che ho imparato a gestire fin da subito.

Lavorare in gruppo è sempre stato un mio limite, che ritengo di aver superato grazie ai progetti svolti durante l'ultimo anno di triennale e ovviamente con lo stage. Nella maggior parte delle situazioni, unire le conoscenze per svolgere un singolo compito può velocizzare considerevolmente il lavoro, soprattutto durante la codifica (ad esempio con il peer programming, attuato giornalmente durante il progetto di Ingegneria del Software, per le parti più delicate dell'applicazione) e durante l'analisi e progettazione del software (durante lo stage, le sessioni di brainstorming con i colleghi erano frequenti e hanno portato ottimi risultati).

Dopo la fine dello stage, ho svolto un periodo extra per portare il software in produzione, dato che erano soddisfatti del risultato. Durante questo periodo, ho imparato come generare automaticamente i *log* durante l'esecuzione di un programma, quindi ho creato una REST API per renderlo disponibile alla rete aziendale. Questa esperienza di post-stage è qualcosa che non ho potuto sperimentare durante la laurea, ma essenziale in ambiente professionale.

Anche se non ho lavorato direttamente ad un algoritmo di machine learning, ho potuto capire la sua importanza nell'ambito professionale. Le aziende che investono (o intendono investire) in questa tecnologia sono portate ad acquisire sempre più dati dai propri clienti e/o partner, in quanto elemento fondamentale per un buon risultato. *PAT s.r.l* sta migliorando in questo compito, come molte altre aziende che vogliono rimanere competitive in un futuro poco lontano. Vedere quante risorse stanno investendo, mi ha motivato ancora di più a seguire un piano di studi orientato all'intelligenza artificiale: studierò qualcosa che mi piace, richiesto dal mercato e che acquisirà sempre più importanza in futuro.

Bibliografia

Siti web consultati

Engagent. URL: <https://www.pat.eu/prodotti-software-e-soluzioni-it/engagent/> (cit. a p. 16).

Manifesto Agile. URL: <http://agilemanifesto.org/iso/it/> (cit. a p. 2).

NLTK - Natural Language Toolkit. URL: <http://www.nltk.org/> (cit. a p. 16).

Nose. URL: <https://nose.readthedocs.io/en/latest/> (cit. a p. 15).

PEP8. URL: <https://www.python.org/dev/peps/pep-0008/> (cit. alle pp. 16, 21).

PyPI (cit. a p. 17).

Python. URL: <https://www.python.org/> (cit. a p. 15).

TreeTagger. URL: <https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/> (cit. a p. 16).

TreeTagger wrapper. URL: <https://treetaggerwrapper.readthedocs.io/en/latest/> (cit. a p. 16).

Wordnet. URL: <http://www.nltk.org/howto/wordnet.html> (cit. a p. 16).