UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un sistema di Sentiment Analysis delle recensioni degli utenti e sua integrazione in un servizio di navigazione

Design and implementation of a Sentiment Analysis system for user reviews and its integration into a navigation service

Relatore

Dott. Enrico Corradini

Candidato

Samuele Sacchetti

ANNO ACCADEMICO 2023-2024

Al mio papà Federico e alla mia mamma Sabrina, per avermi dato la vita.
Alla mia fidanzata Matunikka, per l'amore che mi doni costantemente.

Sommario

Al giorno d'oggi, l'Intelligenza Artificiale svolge un ruolo cruciale in molteplici settori della vita quotidiana, dall'automazione industriale agli assistenti virtuali, dalla medicina personalizzata all'analisi economica. Questa tesi si focalizza sull'implementazione di un sistema di Sentiment Analysis per l'analisi delle recensioni di un'applicazione mobile. Utilizzando RoBERTa, un modello avanzato di Intelligenza Artificiale, noto per la sua capacità di estrarre informazioni essenziali da testi complessi, il sistema classifica le recensioni in positive, negative o neutre. Inoltre, mediante l'uso del modello KeyBERT, è stato possibile estrarre le parole chiave dalle recensioni stesse. Infine, è stata sviluppata una dashboard web per visualizzare in modo efficace queste informazioni, fornendo agli amministratori uno strumento intuitivo per monitorare il feedback degli utenti e prendere decisioni mirate per migliorare l'esperienza dell'applicazione e il servizio offerto.

Indice

Introduzione **1**

1 Introduzione all'intelligenza artificiale **2**
1.1 Definizione di intelligenza artificiale . . . 2
1.2 Transformers . . . 3
1.2.1 Architettura del modello . . . 5
1.2.2 Applicazioni per NLP . . . 7
1.2.3 Language Modeling . . . 7
1.2.4 Vision Models . . . 7
1.3 Sentiment Analysis . . . 8

2 NavigaUNIVPM **9**
2.1 Introduzione e idea . . . 9
2.2 Analisi dei requisiti . . . 9
2.2.1 Requisiti funzionali . . . 10
2.3 Diagrammi di analisi . . . 10
2.3.1 Diagrammi dei casi d'uso . . . 10
2.3.2 Diagrammi di attività . . . 11
2.4 Diagrammi di progettazione . . . 13
2.4.1 Diagramma delle classi . . . 15
2.5 Esperienza utente . . . 17
2.6 Tecnologie utilizzate . . . 18
2.7 Gestione della navigazione e realtà aumentata . . . 19

3 Raccolta e analisi delle recensioni **23**
3.1 Descrizione del problema . . . 23
3.1.1 Analisi dei requisiti . . . 24
3.2 Soluzione proposta . . . 24
3.2.1 Integrazione della progettazione . . . 25
3.2.2 Esperienza utente . . . 28
3.3 Integrazione nel sistema . . . 31
3.3.1 Tecnologie utilizzate . . . 31
3.3.2 Modifiche al codice . . . 32
3.4 RoBERTa . . . 34
3.4.1 Processo di fine-tuning . . . 35

3.4.2 Inferenza del modello 37

4 Valutazione del sistema **39**
4.1 Funzionamento 39
4.1.1 Applicazione mobile 39
4.1.2 Dashboard web 40
4.2 Qualità del modello 40
4.2.1 Test effettuati 40
4.2.2 Generazione delle recensioni utente 42
4.2.3 Parametri del modello 42
4.3 Analisi e discussione 45

5 Conclusione **47**
5.1 Discussione 47
5.2 Sviluppi futuri 47

Bibliografia **49**

Ringraziamenti **50**

Elenco delle figure

1.1 Processo di encoding di un input testuale 5
1.2 Diagramma dell'architettura dei Transformers 6

2.1 Diagramma dei casi d'uso: gestione navigazione 11
2.2 Diagramma dei casi d'uso: gestione dati ambienti 11
2.3 Diagramma dei casi d'uso: gestione area amministratore 12
2.4 Diagramma di attività: scelta partenza 12
2.5 Diagramma di attività: scelta destinazione 13
2.6 Diagramma di attività: caricamento del percorso 13
2.7 Diagramma di attività: avvio della navigazione 14
2.8 Diagramma di attività: visualizzazione del percorso 14
2.9 Diagramma di attività: terminazione della navigazione 15
2.10 Diagramma delle classi: repository 15
2.11 Diagramma delle classi: service 16
2.12 Diagramma delle classi: model 16
2.13 Diagramma delle classi: model 17
2.14 Diagramma delle classi: model 17
2.15 Mockup applicazione mobile 18
2.16 Autenticazione al database Firestore 19
2.17 Implementazione dell'algoritmo di Dijsktra 20
2.18 Implementazione della funzione calculateRoute 21
2.19 Implementazione della classe Navigation 21
2.20 Panoramica generale IDE Unity 22
2.21 Implementazione del metodo per la visualizzazione del percorso in Unity . . 22

3.1 Diagramma di flusso per la soluzione proposta 25
3.2 Diagramma dei casi d'uso: gestione delle recensioni 26
3.3 Classe Feedback 26
3.4 Classe Feedback Repository 27
3.5 Classe Feedback Service 28
3.6 Diagramma di attività: inserimento e visualizzazione feedback 29
3.7 Mockup app mobile 29
3.8 Mockup dashboard web 30
3.9 Struttura di Firebase Realtime Database 31
3.10 Tecnologie utilizzate 32

3.11 Funzione C# da eseguire al click del bottone 33
3.12 Funzione Python per estrazione delle keyword dalle recensioni 34
3.13 Caricamento del dataset 35
3.14 Caricamento del tokenizer e definizione della funzione per la "tokenizzazione" 35
3.15 Caricamento del modello RoBERTa 36
3.16 Definizione degli argomenti per il training 36
3.17 Definizione delle metriche di valutazione del modello 36
3.18 Creazione dell'oggetto Trainer per la gestione dell'addestramento 37
3.19 Avvio dell'addestramento 37
3.20 Risultati di valutazione del modello 37
3.21 Metodo per inferenza del modello RoBERTa 38

4.1 Implementazione del form su app mobile 40
4.2 Implementazione della dashboard web 41
4.3 Risultati delle metriche di valutazione del modello 44
4.4 Confusion matrix 45

Elenco delle tabelle

4.1 Valori dei parametri di valutazione del modello 43

Introduzione

L'intelligenza artificiale (IA) rappresenta un campo dell'informatica dedicato alla creazione di sistemi e algoritmi in grado di emulare funzioni cognitive umane come il ragionamento e l'apprendimento, seppur in modo limitato. Formalmente nato nel 1956 con il convegno di Dartmouth, il campo dell'IA ha visto pionieri come John McCarthy, Claude Shannon e Marvin Minsky delineare la visione di macchine in grado di simulare l'intelligenza umana. Da allora, la ricerca e lo sviluppo nell'IA hanno generato progressi significativi, trasformando profondamente il panorama tecnologico moderno. Oggi, l'IA permea numerosi aspetti della vita quotidiana, dai motori di ricerca agli assistenti virtuali, dai veicoli autonomi alla medicina personalizzata, rappresentando una risorsa cruciale con implicazioni etiche e regolamentari in continua evoluzione.

Questa tesi è il risultato di un tirocinio svolto presso il Dipartimento di Ingegneria dell'Informazione (DII) dell'Università Politecnica delle Marche, focalizzato sull'implementazione di un sistema di Sentiment Analysis per l'analisi delle recensioni. Questo sistema è stato integrato nell'applicazione NavigaUNIVPM, sviluppata durante il corso di Ingegneria del Software. Nata come strumento di supporto per la navigazione interna all'università, l'applicazione rappresenta il contesto pratico per l'implementazione del sistema. Quest'ultimo mira ad analizzare e classificare le recensioni degli utenti per fornire informazioni utili al miglioramento continuo dell'applicazione stessa. Conoscere il pensiero degli utenti riguardo a un prodotto o servizio è fondamentale per rispondere adeguatamente alle loro esigenze. Per affrontare questo compito è stato adottato RoBERTa, un modello avanzato di IA noto per la sua capacità di estrarre informazioni essenziali da testi complessi. Esso ha dimostrato di classificare efficacemente le recensioni come positive, negative o neutre. In aggiunta, il modello KeyBERT è stato utilizzato per l'estrazione delle parole chiave dalle recensioni. Infine, è stata sviluppata una dashboard web che consente agli amministratori di monitorare e interpretare il feedback generale dell'applicazione, visualizzando recensioni, parole chiave e diagrammi che rappresentano le informazioni di classificazione fornite da RoBERTa.

La tesi si articola in cinque capitoli. Nel Capitolo 1 vengono presentate le basi teoriche della Sentiment Analysis, con un focus sull'IA e sull'architettura dei Transformers, fondamentali per il modello utilizzato. Nel Capitolo 2 si descrive l'applicazione NavigaUNIVPM prima dell'integrazione del sistema di Sentiment Analysis. Nel Capitolo 3 ci si concentra sulla raccolta e analisi delle recensioni, presentando RoBERTa come soluzione principale e dettagliando i passaggi per l'integrazione del sistema. Nel Capitolo 4 si valuta il modello con test e parametri per valutarne le performance e l'affidabilità. Infine, nel Capitolo 5 vengono discussi i risultati ottenuti e propone sviluppi futuri per migliorare ulteriormente il sistema di Sentiment Analysis e l'applicazione nel suo complesso.

CAPITOLO 1

Introduzione all'intelligenza artificiale

L'intelligenza artificiale è attualmente un argomento molto studiato e siamo in pieno periodo innovativo e di ricerca. Gli ambiti di applicazione dell'intelligenza artificiale sono svariati e differenti fra loro: dalla medicina all'economia fino all'insegnamento. Come ci sono aspetti positivi di questa relativamente nuova branca dell'informatica, dall'altro lato della medaglia, ci sono gli aspetti negativi che, nella maggior parte delle volte, demoralizzano nell'avanzamento della ricerca. In questo capitolo verrà presentata brevemente una definizione dell'intelligenza artificiale mentre si andrà ad approfondire un aspetto di questa disciplina ossia la Sentiment Analysis o, in maniera più ampia, il campo del Natural Language Processing.

1.1 Definizione di intelligenza artificiale

L'intelligenza artificiale (IA) rappresenta una delle aree più dinamiche e attualmente studiate della scienza e della tecnologia moderna. Negli ultimi anni, l'IA ha guadagnato sempre più rilevanza in una grande varietà di settori, influenzando cosi le nostra vite e trasformando il modo in cui interagiamo con il "mondo digitale".

Sebbene negli ultimi decenni siano emerse numerose definizioni di intelligenza artificiale, possiamo introdurre l'argomento trattato riportando una citazione di John McCarthy che offrì la seguente definizione in una intervista nel 2004:

> *[L'intelligenza artificiale] È la scienza e l'ingegneria della creazione di macchine intelligenti, in particolare di programmi informatici intelligenti. Si tratta di un compito simile a quello di utilizzare i computer per comprendere l'intelligenza umana, ma l'IA non deve limitarsi a metodi biologicamente osservabili.*[1]

John McCharty propose il termine "Intelligenza Artificiale" come riferimento ad una macchina la quale fosse in grado di simulare gli aspetti del pensiero e dell'apprendimento umano.

I rischi e i benefici di questa nuova branca dell'informatica sono molteplici ed entrambi derivano dalle modalità e dagli scopi di utilizzo. Ad ogni modo, la crescente attenzione creatasi attorno a questa disciplina è motivata dai risultati acquisiti grazie al livello di maturità tecnologica raggiunta, sia nel calcolo computazionale, sia nella capacità di analisi in real-time ed in tempi brevi, di enormi quantità di dati in ogni formato. Basti pensare agli enormi passi avanti fatti nel campo della medicina grazie a modelli in grado di riconoscere tumori e consigliare cure mirate per determinate patologie. L'impiego dell'intelligenza artificiale

[1] https://www-formal.stanford.edu/jmc/whatisai.pdf

spazia anche in settori che, purtroppo, non mirano al miglioramento della societa, come quello bellico, ad esempio droni in grado di riconoscere un obiettivo nemico ed abbatterlo. Constatato ciò, è importante comprendere ed essere consapevoli di tutte le potenzialità e le prospettive di impiego dell'IA che sono in continua evoluzione. In questa tesi gli aspetti principalmente analizzati fanno riferimento all'ambito del natural language processing (NLP) e, più precisamente, della Sentiment Analysis, ossia l'analisi del testo e la sua polarizzazione.

1.2 Transformers

I Transformers sono una classe di modelli di intelligenza artificiale sviluppata da ricercatori di Google e della Universtiy of Toronto intorno al 2017. I Transformers erano stati inizialmente progettati per la traduzione del testo ma si sono rivelati anche molto utili in compiti differenti. Per Transformers ci si riferisce ad una architettura di rete neurale utilizzata per analizzare dati complessi come testi, immagini, audio e video. Una rete neurale è un metodo di intelligenza artificiale che insegna ai computer a elaborare i dati in un modo che si ispira al cervello umano. Si tratta di un processo di deep learning che utilizza nodi interconnessi o neuroni[2]; questi neuroni sono organizzati in strati e interconnessi da pesi, consentendo alla rete di apprendere da dati di input e di generare output complessi. Questo meccanismo permette alle reti neurali di essere dei potenti strumenti utili nei compiti di predizione, classificazione e elaborazione dei dati. Ovviamente ci sono differenti reti neurali ottimizzate in base al dato da analizzare. Ad esempio:

- **Convolutional Neural Networks (CNNs)**: utilizzate nelle analisi di immagini
- **Recurrent Neural Networks (RNNs)**: utilizzate nelle analisi testuale e audio
- **Graph Convolutional Neural Networks (GCNNs)**: utilizzate nelle analisi di dati strutturati rappresentati come grafi

Prima dell'avvento dei Transformers, l'analisi testuale veniva effettuata attraverso le Recurrent Neural Networks che presentavano, però, alcuni problemi e fra questi: *(i)* la difficoltà nell'analizzare testi molto lunghi e con svariati paragrafi, *(ii)* l'elevata complessità e *(iii)* durata della fase di training. Questo perchè le parole venivano processate in maniera sequenziale e ciò portava alla difficoltà di allenare la rete con una grande mole di dati. I Transformers, al contrario, permettono una efficiente parallelizzazione che porta ad un aumento delle prestazioni e, di conseguenza, ad una migliore fase di training. Le principali innovazioni introdotte da questa classe di modelli sono:

1. Positional encoding.
2. Attention.
3. Self-attention.

La tecnica del positional encoding sostituisce connessioni ricorrenti o operazioni convoluzionali all'interno del modello. Quindi, per determinare l'ordine della sequenza, ovvero per assegnare una posizione alle parole all'interno di una determinata frase o documento, si inseriscono informazioni sull'ordine relativo o assoluto dei tokens (parola o insieme di parole). Di conseguenza, prima di far elaborare il testo alla rete, ad ogni token si assegna un numero che ne identifica la posizione al suo interno; così facendo, la rete neurale impara l'importanza dell'ordine delle parole all'interno di una frase. Ed è proprio questa la differenza

[2]https://aws.amazon.com/it/what-is/neural-network

che ha fatto dei Transformers un modello più semplice rispetto alle RNNs soprattutto nella fase di training (Vaswani *et al.* [2017]). Le formule utilizzate per il positional encoding, utili per catturare informazioni sulla posizione dei token all'interno di un testo, fanno uso di funzioni seno e coseno a frequenze differenti:

$$PE_{(pos,2i)} = sin(pos/1000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/1000^{2i/d_{model}})$$

dove $d_{model} = 512$ è la dimensione dei vettori utilizzati per rappresentare la posizione dei tokens, ossia determina quanti valori vengono generati per ciascuna posizione nella sequenza di input, *pos* è la posizione di un token nella sequenza di input e i è l'indice utilizzato per generare i valori sinusoidali e cosinusoidali. (Vaswani *et al.* [2017]. Le formule generano valori sinusoidali e cosinusoidali per ciascuna posizione nella sequenza di input.

Il meccanismo di Attention, invece, permette ad un modello di tenere in considerazione ogni singola parola nella frase di origine in maniera da comprendere meglio il contesto e le relazioni implicite delle parole. Questo è risultato molto nelle traduzioni fra lingue diverse. Infatti, considerando una qualunque frase, le parole al suo interno ricoprono posizioni differenti in base alla lingua. Una traduzione della frase con un ordinamento delle parole scorretto porterebbe ad errori di comprensione o, addirittura, a frasi completamente errate. Il meccanismo di Attention nei Transformers permette di considerare una "finestra di riferimento" molto più ampia rispetto a quella utilizzata dalle RNNs, consentendo così di mettere in relazione tutte le parole presenti nella sequenza di input, anziché considerare un ordine temporale di generazione delle parole. Una finestra di riferimento rappresenta l'insieme di token che vengono considerati insieme durante l'analisi o l'elaborazione dei dati; più è ampia più contesto è possibile considerare, riuscendo a catturare relazioni più ampie e complesse tra le parole nella sequenza. Ma la reale innovazione introdotta dai Transformers è detta self-attention la quale si focalizza sulla comprensione e, più nello specifico, sul significato implicito di una frase. Questo meccanismo consente di catturare relazioni globali tra le unità di testo all'interno di un contesto e permette al modello di concentrarsi su differenti parti di una sequenza di input.

Di seguito viene riportato un esempio in cui il sequence-based task, ovvero un compito o un'attività che coinvolge l'elaborazione o l'analisi di sequenze di dati, è la traduzione di una frase. Per questo compito utilizziamo un modello di Transformers con self-attention. Supponiamo di voler tradurre una frase qualunque. I passaggi che vengono effettuati dal modello sono:

1. **Encoding dell'input**: l'input viene codificato in una sequenza di cosiddetti "input embeddings", ovvero dei vettori che hanno l'obiettivo di catturare le relazioni semantiche fra le parole. Questo processo è descritto brevemente in Figura 1.1

2. **Creazione di queries, keys e values**: il meccanismo di self-attention genera tre versioni dell'input embedding: queries, key e values. La query è l'informazione che il modello sta cercando, la key è il contesto o il riferimento e il value è il contenuto associato ad ogni token (significato semantico, contesto, e cosi via). Questi tre valori rappresentano le proiezioni lineari degli embedding dell'input originali e sono utilizzati per calcolare l'attention-score. L'attention-score indica l'importanza relativa di ciascun elemento all'interno di una sequenza rispetto agli altri elementi.

3. **Calcolo dell'attention-score**: l'attention-score viene calcolato considerando il prodotto scalare fra query e key. Questi valori rappresentano l'importanza o rilevanza di ogni parola rispetto alla parola corrente che viene elaborata.

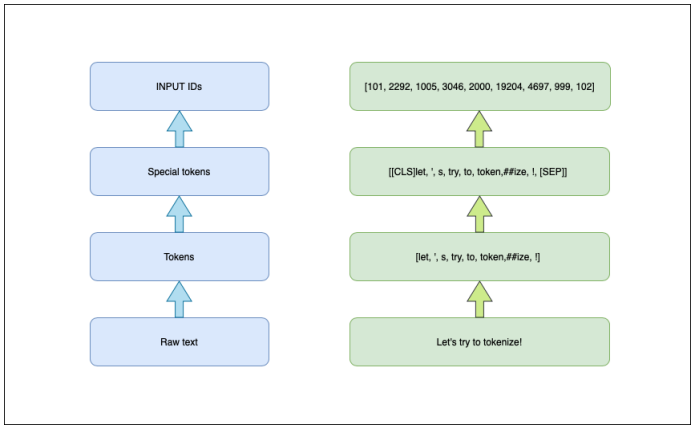

Figura 1.1: Processo di encoding di un input testuale

4. **Applicazione della funzione softmax**: viene applicata la funzione softmax ai valori dell'attention-score convertendoli in probabilità.

5. **Somma pesata dei valori**: viene eseguito il prodotto scalare fra values e i softmax attention-score. Questa operazione ci da una sorta di "media pesata" delle parole, dove le parole più "importanti" o "rilevanti" secondo gli attention-score hanno un peso maggiore nella media. Il vettore risultate, che si ottiene attraverso questa media, è noto come "rappresentazione consapevole del contesto della parola corrente", che tiene conto delle relazioni di una parola con le altre parole nella sequenza.

In conclusione, applicando il meccanismo di self-attention, il modello di Transformers è in grado di catturare le dipendenze presenti fra parole differenti in una sequenza di input. Ciò aiuta a comprendere il contesto e migliora la qualità sia nel caso di traduzione tra lingue che, in generale, nell'analisi testuale (sequence-based tasks) [3].

Uno dei più popolari modelli basato sui transformers è BERT (Bidirectional Encoder Representations from Transformers), pubblicato dai ricercatori di Google nel 2018, il quale può essere adattato ad una enorme varietà di tasks per la NLP. A differenza dei modelli che lo hanno preceduto, BERT è un modello bidirezionale in grado di processare una sequenza di input non in una unica direzione bensì di comprendere il contesto e le relazioni grazie a quest'architettura.

1.2.1 Architettura del modello

L'architettura dei Transformers utilizza un modello strutturato con encoder e decoder. L'encoder mappa una rappresentazione di una sequenza di simboli di input $(x_1, \ldots, x_n)$ in una rappresentazione di una sequenza continua $\mathbf{z} = (z_1, \ldots, z_n)$. L'output dell'encoder è una rappresentazione vettoriale continua dell'input. Dato $\mathbf{z}$ in input, il decoder genera una sequenza di simboli di output $(y_1, \ldots, y_m)$ in maniera sequenziale, un elemento alla volta. Il modello è "auto-regressive", quindi prende in considerazione la sequenza dei simboli già generati per la generazione del simbolo successivo, mantenendo in questo modo una sorta di memoria. Il decoder prende come input gli output precedentemente ottenuti fin quando non si arriva al token di "end of sentence" $< end >$.

In Figura 1.2 viene mostrata l'architettura dei Transformers, la quale è composta da layer di self-attention e layer interamente connessi, sia nell'encoder che nel decoder, permettendo di catturare le dipendenze fra le parole in maniera efficiente anche in documenti testuali molto

[3] https://jalammar.github.io/illustrated-transformer/

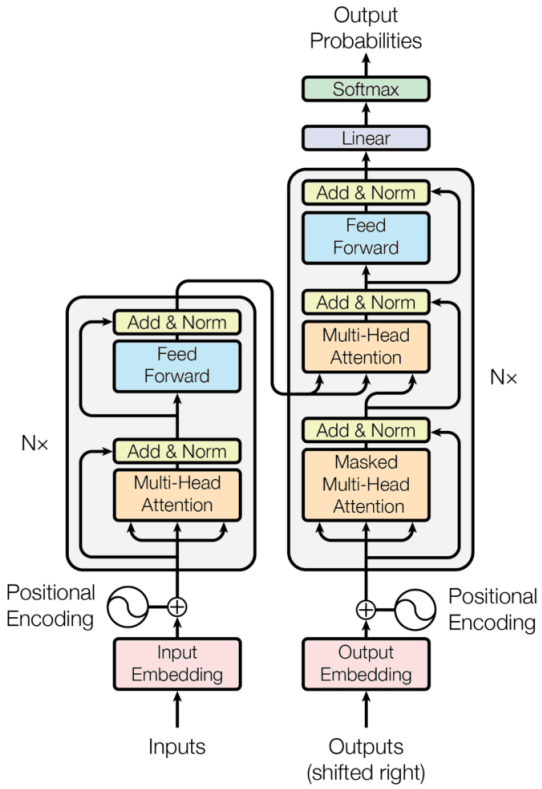

Figura 1.2: Diagramma dell'architettura dei Transformers

lunghi. L'encoder è composto da 6 layer, ognuno dei quali è diviso in 2 sub-layer: nel primo è presente un meccanismo di multi-head self-attention, mentre nel secondo layer è presente una rete feed-forward completamente connessa. L'approccio multi-head self-attention consente al modello di guardare diverse parole in una frase da più "testate" (heads) simultaneamente; ogni testata apprende dalla frase una diversa relazione tra le parole. Il layer feed-forward completamente connesso, invece, è costituto da uno strato di neuroni (nodi) che è connesso a tutti i neuroni del layer di self-attention. Questo layer permette di trattare ogni parola in maniera indipendente e di trasformarla, attraverso una rete a più strati – ad esempio con una funzione di attivazione (di solito ReLU) –, aiutando il modello a catturare relazioni non lineari fra le parole. Attorno ai due sub-layer viene applicata una "residual connection", che consiste nell'aggiungere l'input originale del layer all'output del layer stesso, seguita da una normalizzazione definita come:

$$LayerNorm(x + Sublayer(x))$$

dove $Sublayer(x)$ è la funzione implementata in ogni sub-layer, come ad esempio le operazioni di multi-head self-attention oppure quelle eseguite all'interno del layer di feed-forward. Quindi la funzione $Sublayer(x)$ applica delle operazioni all'input x di ogni layer.

Il decoder è composto da 6 layer ma, a differenza dell'encoder, ogni layer è composto da 3 sub-layer. Il terzo sub-layer esegue un multi-head attention sull'output dell'encoder stack.(Vaswani *et al.* [2017]). L'operazione di multi-head attention è differente da quella di multi-head self-attention che viene eseguita nell'encoder. Il meccanismo di multi-head attention, infatti, è capace solamente di catturare relazioni fra una singola sequenza di input, a differenza del multi-head attention che può essere utilizzato per relazioni tra sequenze diverse.

1.2.2 Applicazioni per NLP

La Natural Language Processing (NLP) è un campo di linguistica e apprendimento automatico focalizzato sulla comprensione di tutto ciò che è correlato al linguaggio umano. Lo scopo dei compiti di NLP non è semplicemente quello di comprendere singole parole, ma di essere in grado di comprenderne il contesto. [4]

I Transformers in NLP hanno un ampio range di applicazioni riuscendo a gestire con facilità delle dipendenze a lungo raggio (long-range dependencies). Le reti neurali tradizionali (come le RNNs), infatti, possono avere difficoltà a catturare dipendenze a lungo raggio perchè l'informazione deve fluire attraverso una serie di passaggi ricorsivi e il segnale può attenuarsi o scomprarire durante questo processo. I modelli Transformers, invece, riescono ad essere molto efficienti grazie al meccanismo di self-attention che, come descritto nelle sezioni precedenti, ci permette di creare connessioni fra parole all'interno di una stessa frase senza l'utilizzo di convoluzione o connessioni ricorrenti [5]. Di seguito vengono riportate una serie di applicazioni in cui i Transformers sono stati scelti come "backbone models", ovvero modelli utilizzati come base o struttura di partenza per altre attività di addestramento o per la creazione di modelli più complessi. Come ultima applicazione verrà considerata anche la Sentiment Analysis che rappresenta il core del lavoro presentato in questa tesi.

1.2.3 Language Modeling

Il language modeling è un task NLP che permette di prevedere il token successivo, dati i tokens precedenti. Viene formulato come un probema di stima della distribuzione dei tokens alla posizione $i+1$ dati i token dalla posizione 0 alla posizione i. Il miglior token predetto è quello che massimizza la probabilità, data da

$$\hat{x}_{i+1} = \arg\max_{x_{i+1} \in V} Pr(x_i | x_0, \ldots, x_i)$$

Dove V è il vocabolario, ossia l'insieme di tutti i token che il modello può predire. I token $x_0, \ldots, x_i$ rappresentano il contesto e sono fondamentali per stimare la distribuzione di probabilità condizionata del token successivo $\hat{x}_{i+1}$. Quest'ultimo token diventa parte del contesto per la generazione degli altri token. Quindi, attraverso i token iniziali $x_1, \ldots, x_i$, il modello è in grado di generare dei token $\hat{x}_{i+1}, \ldots, \hat{x}_{k+1}$ per completare la sequenza in maniera coerente.

1.2.4 Vision Models

I Transformers sono stati inizialmente utilizzati per risolvere problemi di NLP ma nell'ultimo periodo, nell'ambito della Computer Vision (CV), è presente una tendenza al cambiamento e si sta passando dall'utilizzo delle CNNs (che sono state estremamente efficaci in una vasta gamma di compiti di CV) ai Transformers come "backbone models". Una applicazione interessante dei Transformers in questo ambito è la classe di modelli chiamata Vision Transformers (ViT). L'idea principale dietro ViT è quella di dividere l'immagine in patch e trattare ciascuna patch come un token. Una immagine può essere quindi rappresentata come:

$$H \cdot W \cdot C$$

dove H è l'altezza dell'immagine, W e la larghezza dell'immagine e C è il numero di canali Una patch di questa immagine, di conseguenza, potrebbe essere un quadrato di lato P e tutte

[4] https://huggingface.co/learn/nlp-course/chapter1/2
[5] http://nlp.seas.harvard.edu/annotated-transformer/

le patch possono essere rappresentate come:

$$P \cdot P \cdot C$$

Queste patch vengono date in input all'encoder che genera una rappresentazione contestuale dell'immagine. Questa rappresentazione, poi, potrà essere utilizzata per differenti task come classificazione di immagini, individuazione di oggetti, e cosi via (Xiao e Zhu [2023]).

1.3 Sentiment Analysis

La Sentiment Analysis (SA) è una tecnica utile per identificare, analizzare ed estrarre keywords, stati d'animo, emozioni da dati in forma testuale.

Nell'era della digitalizzazione e dei social network, l'importanza dei dati e, più precisamente, del feedback degli utenti, dell'emozione o dello stato d'animo suscitati da un determinato contenuto online, ha assunto un peso davvero importante. Da parte delle aziende sviluppatrici di software il pensiero dell'utente riguardo un determinato prodotto è essenziale ma con una mole elevata di dati è impossibile andare ad analizzare ogni singola recensione degli utenti. E' qui che entrano in gioco i modelli di intelligenza artificiale che, se ben allenati, permettono di analizzare in maniera efficace ed efficiente una grande mole di dati e restituire in output le feature richieste. Per esempio, è possibile estrarre la polarizzazione (sentiment) di un testo, andando a capire se è positivo, neutro o negativo. Inoltre è possibile comprendere il contesto della frase ed estrarre le parole fondamentali ai fini di un'analisi più ampia. La Sentiment Analysis, quindi, è una componente chiave del Natural Language Processing e rientra nell'ambito del "text classification", permettendo di analizzare singole frasi o addirittura interi documenti. Il compito della Sentiment Analysis comprende essenzialmente quattro step principali:

1. Pre-processing
2. Feature extraction
3. Classification
4. Interpretation of results

Fra gli step menzionati, quello della feature extraction svolge un ruolo fondamentale in quanto utile a migliorare l'efficienza della classificazione. Ci sono due metodi per eseguire la feature extraction: il primo è basato sul lessico (lexicon-based method), mentre il secondo si basa sul Machine Learning oppure Deep Learning. Nel caso dei metodi basati sul lessico, vengono forniti al modello degli elenchi di parole positive e negative e, andando a contare il numero, viene calcolato il sentiment della frase. Dunque, se ci sarà maggiore presenza di parole che sono state definite come positive nell'elenco fornito al modello, la frase sarà classificata con un sentiment positivo; viceversa, se c'è maggioranza di parole definite come negative, la frase sarà classificata con un sentiment negativo. Per quanto riguarda le tecniche basate sul Machine-Learning e Deep Learning, si possono suddividere in tecniche supervisionate e non supervisionate Le tecniche supervisionate necessitano un addestramento del modello su dati etichettati prima che sia valutato su dati non visti durante la fase di training per determinare le prestazioni del modello. D'altra parte, le tecniche non supervisionate addestrano il modello utilizzando dati non etichettati, consentendo così al modello di estrarre conoscenza dai dati in modo autonomo (Kokab *et al.* [2022]).

CAPITOLO 2

NavigaUNIVPM

L'applicazione NavigaUNIVPM è stata sviluppata per risolvere il problema, comune a molti studenti e docenti, di orientarsi all'interno degli edifici universitari. L'obiettivo principale è quello di guidare gli utenti da un punto di partenza fino a un punto di destinazione all'interno dell'università. L'idea originale dell'applicazione è nata come progetto per un esame del corso di Ingegneria del Software, successivamente ampliato durante il lavoro di tirocinio. Nei paragrafi che seguono viene introdotta sia l'idea alla base dell'applicazione sia il suo funzionamento.

2.1 Introduzione e idea

L'idea di questa applicazione nasce da un progetto per l'esame del corso di "Ingegneria del software". Il progetto è stato ampliato e migliorato per il lavoro di tirocinio e verrà presentato, in maniera più dettagliata, nei successivi capitoli della tesi.

L'applicazione NavigaUNIVPM è nata da una semplice necessità: molti studenti, matricole o docenti possono trovarsi smarriti all'interno degli edifici dell'università, senza sapere in modo preciso come raggiungere le aule dove si svolgono lezioni o esami, gli uffici dei docenti o altri luoghi di interesse come ad esempio bar, segreteria, portineria, e cosi via. Spesso, chiedere indicazioni ad altri studenti non è sufficiente per ottenere una descrizione dettagliata del percorso da seguire. L'obiettivo principale di NavigaUNIVPM è di guidare gli utenti da una posizione di partenza a una di arrivo all'interno degli edifici universitari.

Nel momento in cui l'utente avvia l'applicazione, potrà selezionare la posizione di partenza (quella in cui si trova, ad esempio) e quella di destinazione attraverso un menu a tendina dedicato; grazie ad un bottone, poi, sarà possibile avviare la navigazione. Una volta che la navigazione è avviata, attraverso un sistema di realtà aumentata, l'utente visualizzerà, sullo schermo del suo cellulare, l'ambiente circostante con evidenziato il percorso che lo guiderà lungo il tragitto fino al raggiungimento della destinazione prescelta.

2.2 Analisi dei requisiti

Attraverso la descrizione del sistema è stato possibile eseguire una analisi dei requisiti, considerando delle macro aree in modo da poter associare ogni requisito ad una funzionalità più generale che è stata implementata, successivamente, nel sistema. Sono state considerate come aree quelle relative alla gestione della navigazione, degli ambienti e dell'amministratore. Di seguito vengono elencati i requisiti funzionali, ossia quei requisiti che specificano le

funzionalità o le azioni che il sistema deve essere in grado di svolgere. In altre parole, descrivono ciò che il sistema deve fare o fornire agli utenti. Questi requisiti si concentrano sul comportamento del sistema e su ciò che gli utenti possono aspettarsi di ottenere utilizzando l'applicazione.

2.2.1 Requisiti funzionali

Di seguito sono elencati i requisiti funzionali suddivisi per area di gestione:

Area: Gestione Navigazione	
RF1: Seleziona partenza	Il sistema deve gestire la selezione del punto di partenza da parte dell'utente.
RF2: Selezione destinazione	Il sistema deve gestire la selezione del punto di destinazione da parte dell'utente.
RF3: Termina navigazione	Il sistema deve permettere la terminazione della navigazione quando l'utente è arrivato a destinazione oppure se l'utente decide di interromperla manualmente.
RF4: Visualizza percorso	Il sistema deve gestire la visualizzazione del percorso per raggiungere la destinazione desiderata dall'utente.
Area: Gestione dati ambienti	
RF5: Fornisci orari aule	Il sistema deve fornire all'utente gli orari di occupazione delle aule.
RF6: Fornisci dati docenti	Il sistema deve fornire all'utente informazioni riguardanti i docenti.
RF7: Fornisci dati segreteria	Il sistema deve fornire all'utente informazioni riguardanti la segreteria.
RF8: Modifica mappa	Il sistema deve garantire la possibilità di modificare la mappa dell'Università in caso di necessità.
Area: Gestione amministratore	
RF9: Recupera credenziali	Il sistema deve garantire all'amministratore la possibilità di recuperare le proprie credenziali in caso di smarrimento.
RF11: Gestisci autenticazione	Il sistema deve gestire l'autenticazione dell'amministratore per consentire l'accesso alla sua area riservata.
RF11: Modifica orari segreteria	Il sistema deve permettere all'amministratore di modificare gli orari dei vari ambienti dell'Università.
RF12: Gestione richieste amministratori	Il sistema deve permettere agli amministratori, già autenticati, di accettare o rifiutare le richieste di altri utenti che desiderano autenticarsi come amministratori.

2.3 Diagrammi di analisi

I diagrammi di analisi sono uno strumento di modellazione utilizzato per rappresentare in maniera grafica le componenti, le interazioni e le funzionalità di un sistema durante la fase di analisi dei requisiti del software. La fase di analisi è fondamentale in quanto permette di comprendere a fondo le esigenze degli utenti (o del committente per cui si sta lavorando) e di tradurle in requisiti specifici che guideranno lo sviluppo del sistema.

I diagrammi di analisi possono includere diversi tipi di diagrammi, tra cui: diagramma dei casi d'uso, diagramma delle classi, diagramma di sequenza, diagramma di attività. In seguito, come diagrammi di analisi, vengono mostrati i diagrammi dei casi d'uso e quelli di attività delle principali funzionalità che il sistema fornisce all'utente. Queste ultime comprendono la selezione della partenza e della destinazione, il caricamento e la visualizzazione del percorso, l'avviamento e la conclusione della navigazione.

2.3.1 Diagrammi dei casi d'uso

A partire dall'analisi dei requisiti effettuata, sono stati modellati i diagrammi dei casi d'uso per ogni "macro sezione". Il diagramma dei casi d'uso è un diagramma UML (Unified Modeling Language) comportamentale utile per analizzare i diversi tipi di ruoli che possono essere ricoperti all'interno di un sistema e per descrivere e comprendere come questi ruoli si interfacciano con il sistema stesso.

Il diagramma dei casi d'uso in Figura 2.1 descrive le azioni che l'utilizzatore, ossia l'attore principale, può compiere per gestire la navigazione interna all'università. Esso può, quindi,

caricare il percorso dopo aver scelto una posizione di partenza e una di destinazione e può, successivamente, avviare la navigazione.

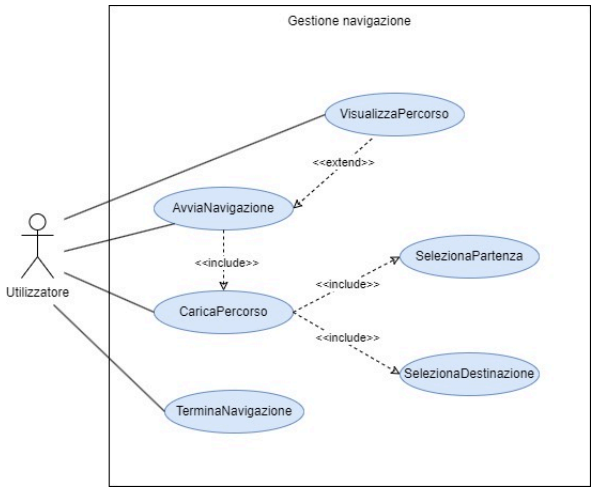

Figura 2.1: Diagramma dei casi d'uso: gestione navigazione

Il diagramma in Figura 2.2, invece, descrive le azioni che l'utilizzatore dell'applicazione può svolgere nella gestione dei dati degli ambienti dell'università. Come si nota, infatti, esso può visualizzare gli orari dei vari punti di interesse e le informazioni relative alla segreteria e agli insegnanti. Affinché questi dati possano essere messi a disposizione dell'utilizzatore, c'è bisogno che venga fatta una ricerca di questi dati all'interno di un database: l'utilizzatore non è in grado di visualizzare direttamente questo processo.

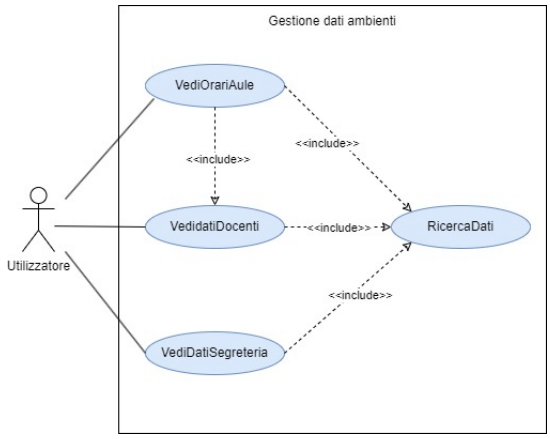

Figura 2.2: Diagramma dei casi d'uso: gestione dati ambienti

In Figura 2.3 viene descritto, infine, il diagramma dei casi d'uso per la gestione dell'area dell'amministratore che permette di visualizzare come l'attore principale, ossia l'amministratore può interfacciarsi con il sistema. Tale attore può: richiedere o recuperare, in caso di smarrimento, le proprie credenziali, accettare nuovi amministratori, consultare le statistiche dell'applicazione, modificare la mappa dell'università o gli orari della segreteria che vengono visualizzati dall'utilizzatore.

2.3.2 Diagrammi di attività

I diagrammi di attività descrivono il flusso di lavoro all'interno di un sistema, mostrando le attività da svolgere e le relazioni tra di esse.

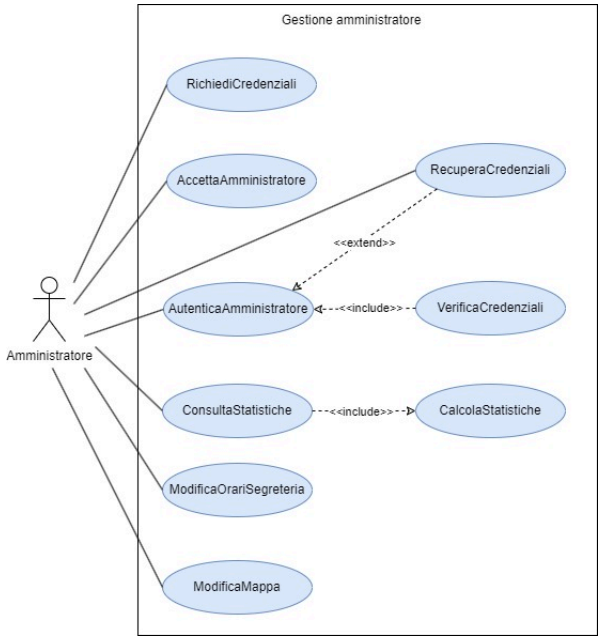

Figura 2.3: Diagramma dei casi d'uso: gestione area amministratore

Partendo dal diagramma di attività in Figura 2.4 per la selezione della partenza, si nota il flusso delle attività che l'utente può svolgere. Infatti, la partenza può essere selezionata sia in modo manuale – attraverso un menu a tendina, ad esempio – sia attraverso la scannerizzazione di un QR code posizionato nei punti di interesse interni all'università.

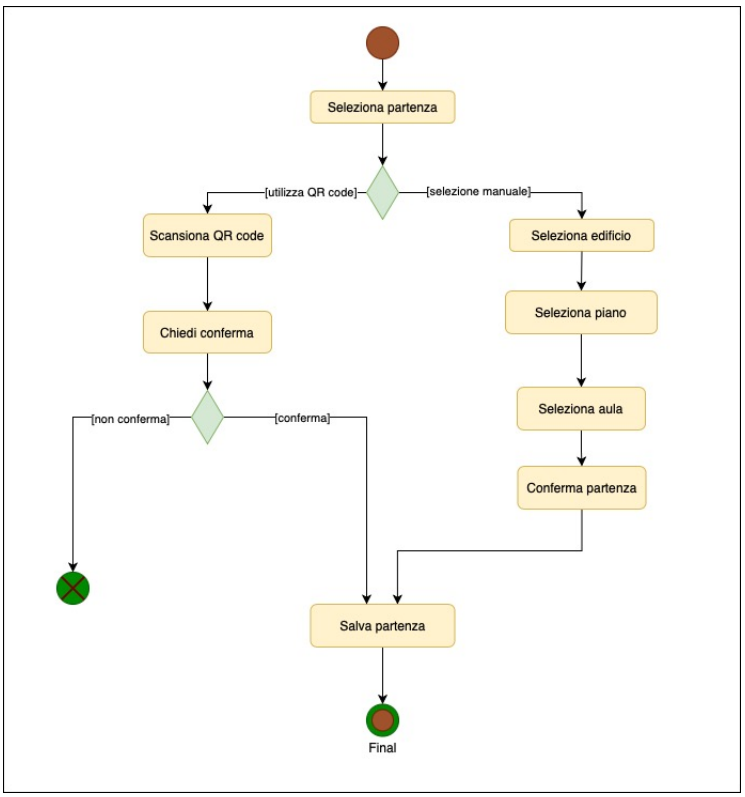

Figura 2.4: Diagramma di attività: scelta partenza

Il diagramma di attività in Figura 2.5, invece, rappresenta le azioni che l'utente deve

svolgere per scegliere e confermare la destinazione in cui vuole arrivare.

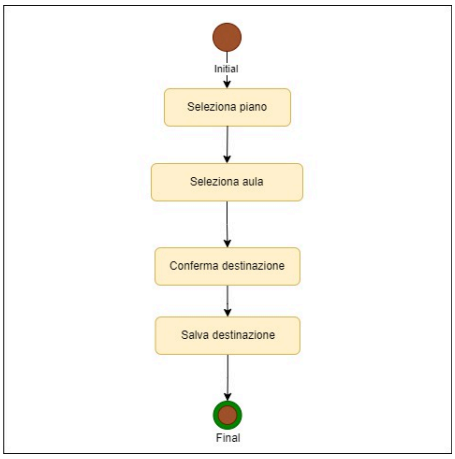

Figura 2.5: Diagramma di attività: scelta destinazione

Il diagramma di attività in Figura 2.6 per il caricamento del percorso contiene le azioni di scelta del punto di partenza e di destinazione che fanno riferimento alle azioni da svolgere descritte nei diagrammi di attività visti in precedenza. Dopo aver fatto ciò, il sistema elabora il percorso più veloce, avendo preso in input il punto di partenza e di arrivo.

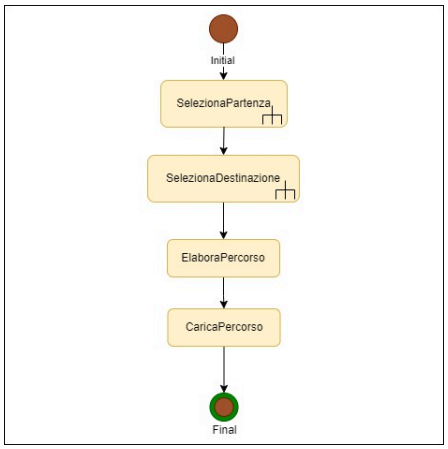

Figura 2.6: Diagramma di attività: caricamento del percorso

Le azioni necessarie per l'avvio della navigazione sono descritte nel diagramma di attività in Figura 2.7. Dopo che il percorso viene caricato, l'utente potrà scegliere se visualizzare o meno la mappa con il percorso evidenziato.

Il diagramma di attività per la visualizzazione del percorso, in Figura 2.8, descrive le azioni che l'utente può svolgere per "bloccare" momentaneamente la navigazione per visualizzare il percorso completo e per riavviare la navigazione nascondendo la mappa.

Il diagramma di attività in Figura 2.9, infine, descrive le azioni che permettono di terminare la navigazione. L'utente, infatti, può terminare senza il bisogno di arrivare a destinazione per poi selezionare un nuovo percorso di navigazione.

2.4 Diagrammi di progettazione

La fase di progettazione è successiva al completamento dei processi relativi alla fase di analisi. Tuttavia, in molti casi, queste due fasi possono essere eseguite in parallelo, sebbene i

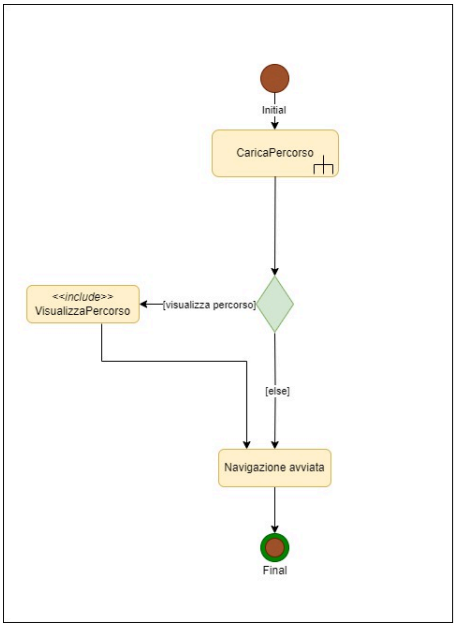

Figura 2.7: Diagramma di attività: avvio della navigazione

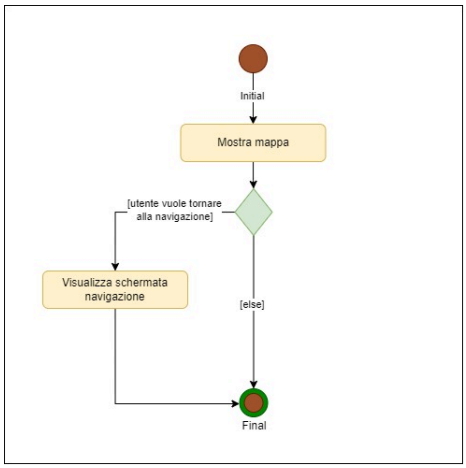

Figura 2.8: Diagramma di attività: visualizzazione del percorso

manufatti risultanti, ovvero il modello di analisi e il modello di progettazione, siano distinti.

Lo scopo delle attività di progettazione è quello di definire in modo completo l'implementazione delle funzionalità identificate durante la fase di analisi.
Il modello di progettazione è composto da:

- Sottosistema di progettazione
- Classi di progettazione
- Interfacce
- Progettazione delle realizzazioni dei casi d'uso
- Diagramma di deployment

Nella sottosezione successiva verrà mostrato il diagramma delle classi e delle interfacce.

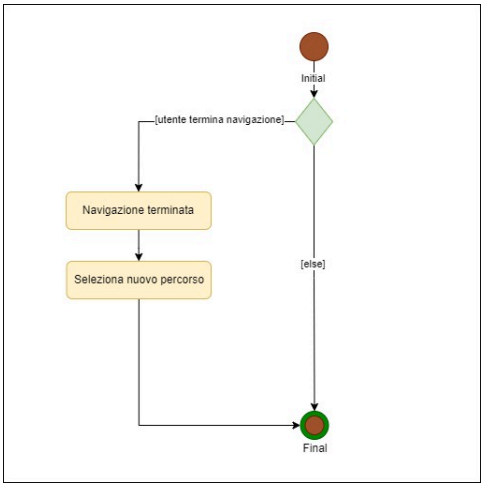

Figura 2.9: Diagramma di attività: terminazione della navigazione

2.4.1 Diagramma delle classi

Il diagramma delle classi, soprattutto nella fase di progettazione, è importante al fine di modellare al meglio la realtà considerata. Le varie classi possono essere assimilate a persone fisiche, oggetti o concetti e comprendere in che modo questi sono in relazione tra loro e da quali attributi o azioni si distinguono.

Le classi di progettazione sono classi le cui specifiche sono talmente complete che possono essere effettivamente implementate. L'analisi serve a modellare cosa il sistema dovrebbe fare mentre la progettazione serve a modellare come tale comportamento debba essere implementato.

Si è scelto di utilizzare il pattern architetturale Service-Repository per organizzare, successivamente, il codice in maniera modulare e scalare. Questo pattern permette di suddividere la logica di business dalla gestione dei dati; esso è composto da due livelli principali, ossia il layer repository e il layer service. Si sono suddivise, di conseguenza, le classi rispetto a quelle che appartengono a: repository, model e service. In Figura 2.10 viene riportato il diagramma delle classi per il package "repository". Queste classi servono per recuperare i dati dal database e di fornirli alle classi che formano il "service layer".

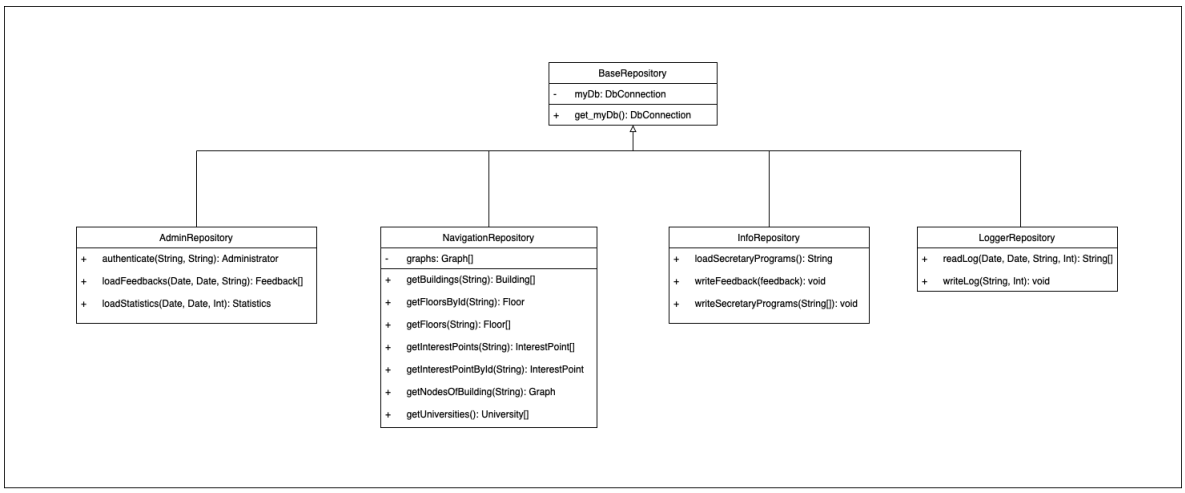

Figura 2.10: Diagramma delle classi: repository

In Figura 2.11 viene riportato il diagramma delle classi per il package "service". Queste

classi implementano la logica di business complessa, coordinando le interazioni tra i "model" e altre componenti dell'applicazione.

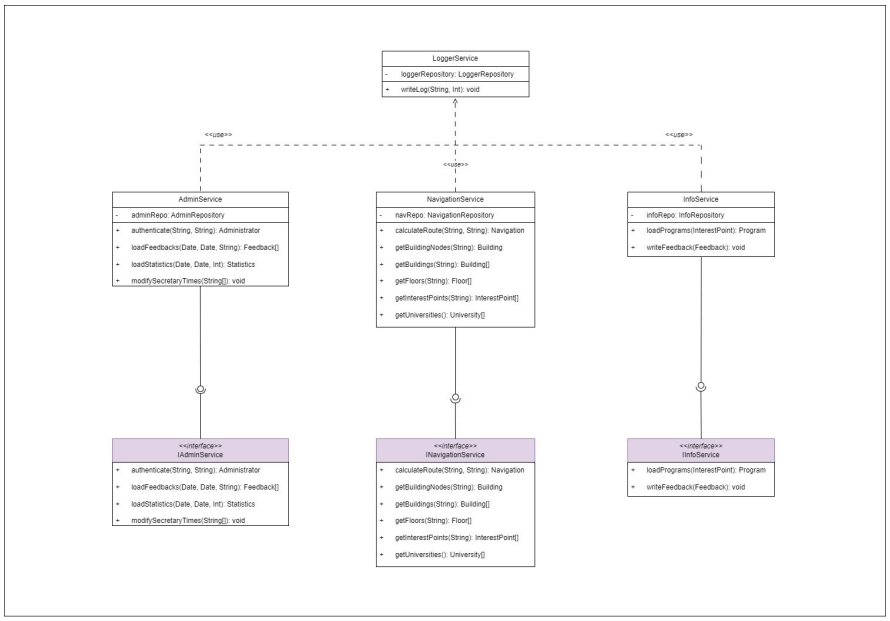

Figura 2.11: Diagramma delle classi: service

In Figura 2.12 viene riportato il diagramma delle classi per il package "model". Queste classi rappresentano i dati e la logica di business di base. Definiscono la struttura dei dati e gestiscono le interazioni con il database.

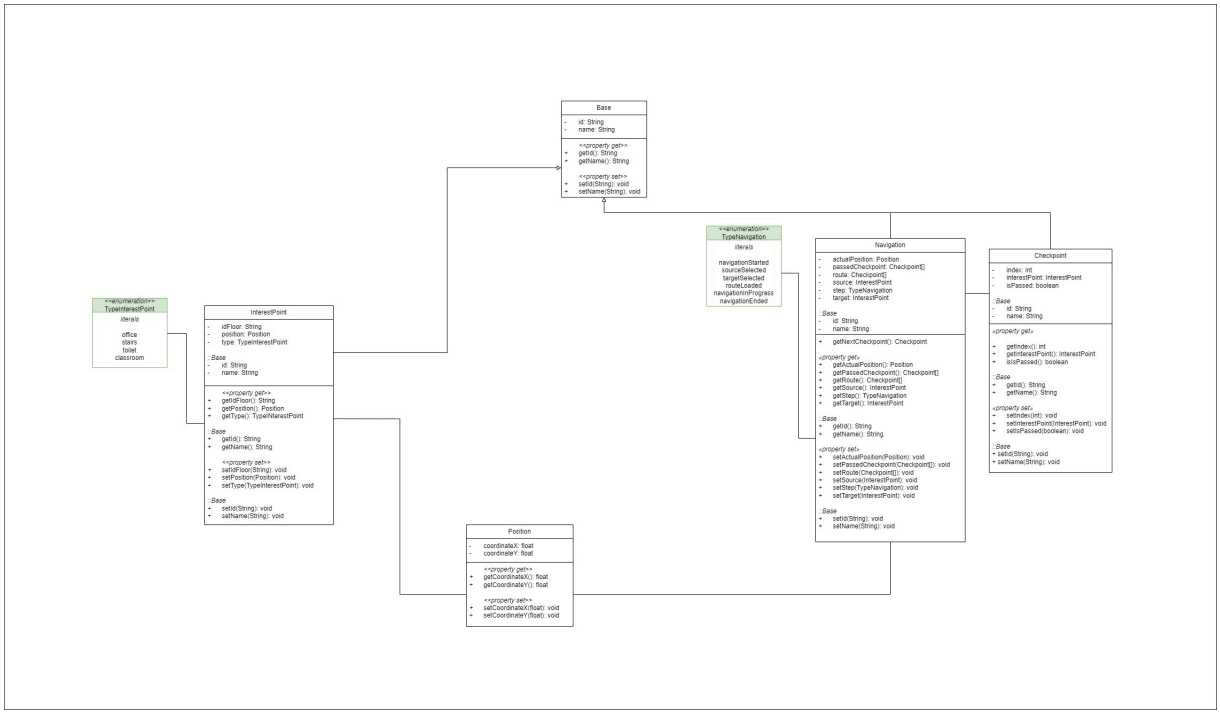

Figura 2.12: Diagramma delle classi: model

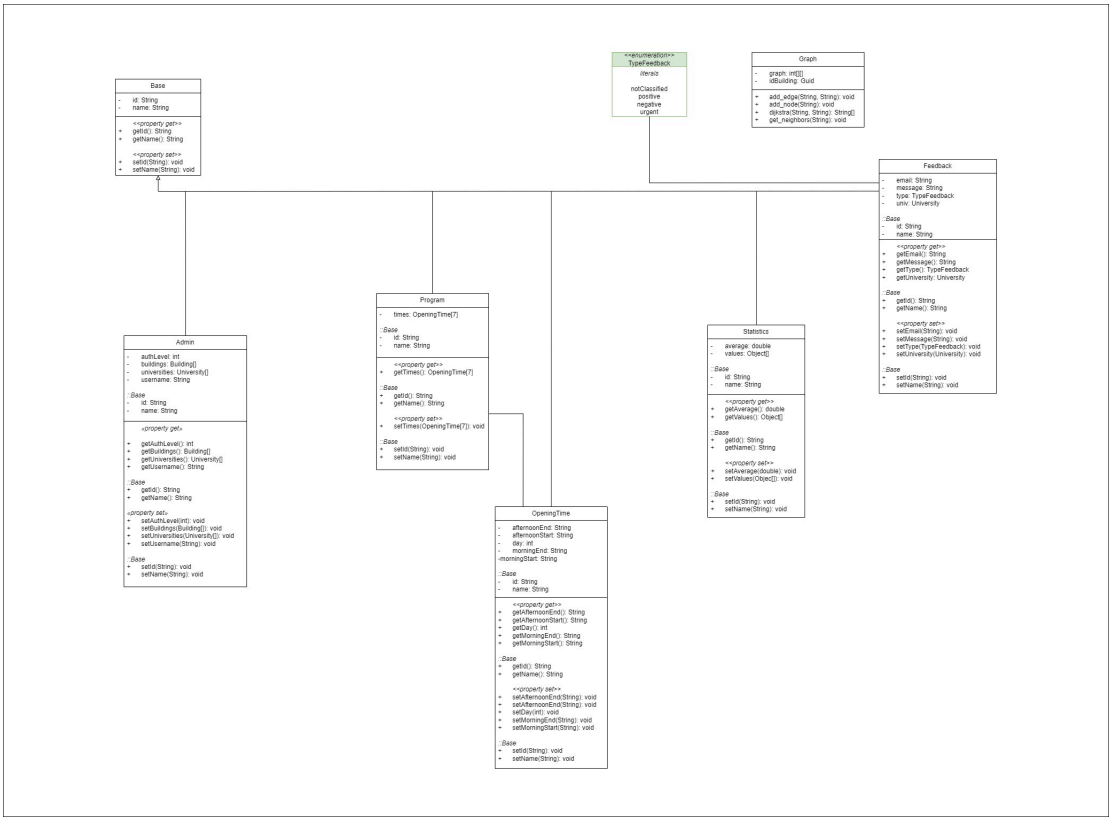

Figura 2.13: Diagramma delle classi: model

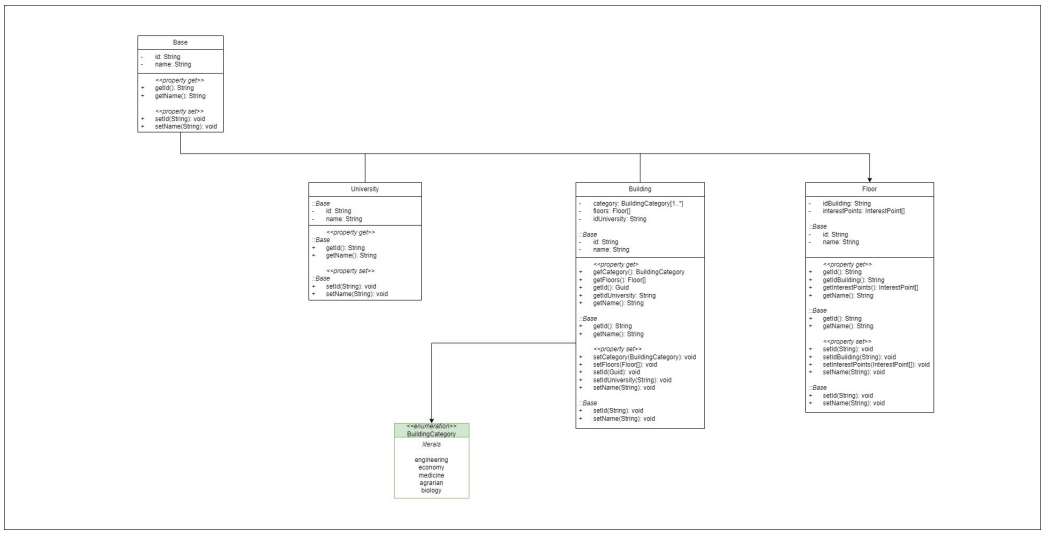

Figura 2.14: Diagramma delle classi: model

2.5 Esperienza utente

L'applicazione è stata progettata e sviluppata in modo tale da essere il più possibile "user-friendly" e accessibile da qualunque tipo di utente. Le schermate di avvio, di scelta della partenza e della destinazione e quella di navigazione, sono state pensate seguendo una armonia cromatica coerente. L'esperienza utente, inoltre, è stata progettata per guidare gli utenti attraverso l'intero flusso dell'applicazione in modo chiaro e senza ambiguità. Ciò è stato ottenuto attraverso l'uso di terminologie e componenti grafiche chiare, garantendo

un'esperienza intuitiva per tutti gli utenti. In Figura 2.15 è possibile visualizzare il mockup dell'applicazione mobile, realizzato grazie al software Figma. Come accennato precedentemente, possiamo notare la presenza di poche componenti grafiche ma che permettono all'utente di non confondersi e di riuscire a navigare all'interno dell'applicazione in maniera facile.

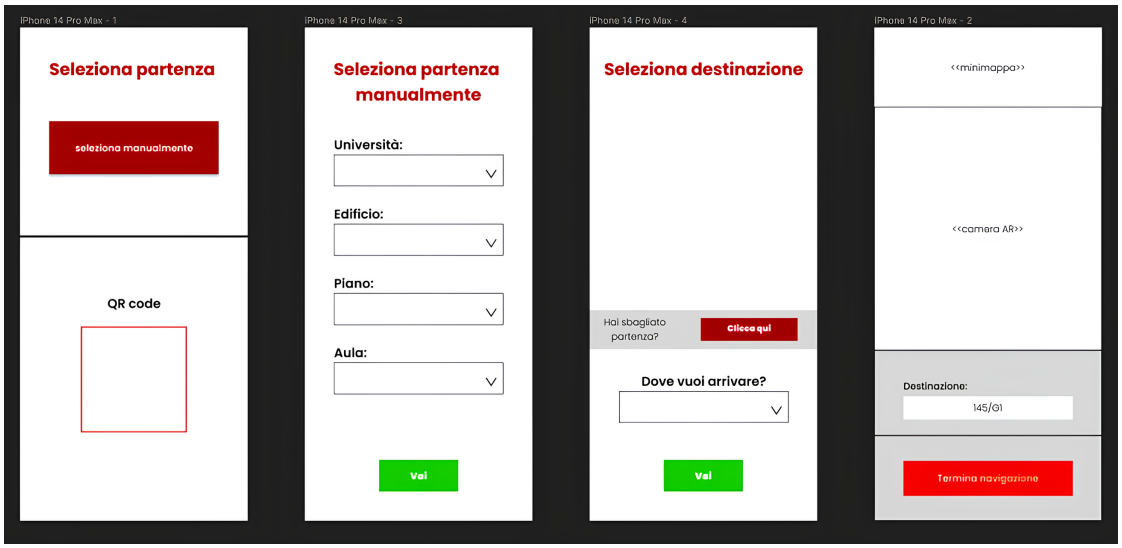

Figura 2.15: Mockup applicazione mobile

All'avvio dell'applicazione, viene visualizzata una schermata che presenta una grafica esplicativa dello scopo per cui è stata sviluppata, ossia la navigazione interna all'università. Premendo il bottone "inizia navigazione", è possibile passare alla schermata successiva in cui l'utente può scegliere il punto di partenza sia attraverso l'inquadratura di un QR code, sia attraverso un menù a tendina. Il punto di destinazione che vuole raggiungere, invece, può essere selezionato solamentre attraverso un menù a tendina apposito. Gli elementi selezionabili dal menù sono stati inseriti precedentemente in fase di sviluppo. Successivamente all'avvio della navigazione, viene visualizzato sullo schermo ciò che viene catturato dalla fotocamera del cellulare con, in aggiunta, il percorso evidenziato, grazie alla realtà aumentata, che porta alla destinazione selezionata in precedenza dall'utente. Di conseguenza, basterà seguire le indicazioni per arrivare a destinazione, cercando di puntare la fotocamera verso il sentiero tracciato dall'applicazione in maniera da poter navigare al meglio.

2.6 Tecnologie utilizzate

In questa sezione vengono presentate le tecnologie utilizzate per l'implementazione dell'applicazione "NavigaUNIVPM".

Come database si è scelto di utilizzare Firestore, messo a disposizione da Firebase. Firebase è una piattaforma di sviluppo mobile e web fornita da Google. Essa offre una vasta gamma di servizi, fra cui:

- Autenticazione: fornisce un modo sicuro e facile per gli utenti per accedere alle app tramite email, password, o account social come Google o Facebook.
- Database in tempo reale: il Firebase Realtime Database consente alle organizzazioni di memorizzare e sincronizzare dati in tempo reale su tutti i dispositivi degli utenti, facilitando la creazione di app sempre aggiornate anche offline.

- Messaggistica cloud: Firebase Cloud Messaging (FCM) consente alle aziende di inviare messaggi ai dispositivi degli utenti, inclusi notifiche push e aggiornamenti dell'app.
- Crashlytics: Fornisce report dettagliati sui crash delle app, aiutando le organizzazioni a identificare rapidamente e risolvere i problemi.
- Monitoraggio delle prestazioni: Firebase Performance Monitoring fornisce informazioni sulle prestazioni dell'app, inclusi dati su utilizzo della CPU, memoria e traffico di rete.
- Test Lab: Consente agli sviluppatori di testare le proprie app su una vasta gamma di dispositivi e configurazioni, garantendo che funzionino correttamente in diverse situazioni di utilizzo[1].

Come linguaggio di programmazione per la parte backend si è deciso di utilizzare Python, noto per la sua flessibilità su contesti differenti e per la sua facilità d'uso. Per recuperare le informazioni dal database attraverso Python, è stata utilizza l'API Firebase. Firebase, infatti, fornisce un SDK (Software Development Kit) specifico per Python chiamato "firebase-admin". Questo SDK consente di interagire con i servizi Firebase, incluso il database Firestore. In Figura 2.16 viene riportato lo script Python, utilizzato per il collegamento al database di Firebase:

```python
def __init__(self):
    if not firebase_admin._apps:
        cred_path = 'navigaunivpm-firebase-adminsdk-mnhmg-f5e1ddd076.json'
        bundle_dir = getattr(sys, '_MEIPASS', path.abspath(path.dirname(__file__)))
        path_to_dat = path.abspath(path.join(bundle_dir, cred_path))
        cred = credentials.Certificate(path_to_dat)
        firebase_admin.initialize_app(cred)

    self.myDb = firestore.client()
```

Figura 2.16: Autenticazione al database Firestore

Nello specifico *cred_path* contiene le credenziali necessarie per il collegamento al servizio di Firebase. Se non ci sono applicazioni Firebase inizializzate nel contesto corrente, viene inizializzato un oggetto Firestore client per la comunicazione con il database.

Per la parte di frontend viene utilizzato il motore grafico Unity, il quale permette di costruire l'interfaccia grafica in maniera facile, inserendo tutti i componenti necessari come i bottoni, gli input text, e cosi via. La logica dietro l'interfaccia, ossia il comportamento "dinamico" di questi componenti grafici, è stata inserita utilizzando il linguaggio di programmazione C#. Durante la fase di implementazione si è utilizzato il sistema di controllo delle versioni "Git". Attraverso questo sistema è possibile lavorare su porzioni di codice differenti in maniera simultanea. Inoltre, è possibile tenere traccia di tutte le modifiche al codice che vengono effettuate dai membri del team.

2.7 Gestione della navigazione e realtà aumentata

La gestione della navigazione interna all'applicazione, da un punto di partenza ad un punto di destinazione, viene gestita attraverso l'utilizzo dell'algoritmo di Dijkstra. L'algoritmo di Dijsktra è un algoritmo per la ricerca del percorso più veloce fra due nodi di un grafo pesato, il quale potrebbe rappresentare, ad esempio, una rete stradale[2]. Nel caso dell'applicazione NavigaUNIVPM i nodi rappresentano i punti di interesse (aule, uffici, bar, e così via) mentre

[1] https://firebase.google.com
[2] https://w.wiki/4xCS

gli archi sono tutti i possibili percorsi che li collegano. L'idea alla base è stata quella di creare un grafo pesato per ogni edificio dell'università. Il peso di ogni arco è stato assegnato utilizzando, come criterio di attribuzione del valore opportuno, la distanza euclidea fra due punti (x_1, y_1) e (x_2, y_2). In formula, la distanza è data da:

$$d_{1,2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Dove $d_{1,2}$ è la distanza fra i due punti, ossia il peso assegnato all'arco che collega il nodo 1 con il nodo 2. Il collegamento fra diversi piani di uno stesso edificio, ossia le scale, è stato modellato come un nodo. Una volta scelti sia il punto di partenza p che quello di destinazione d e avviata la navigazione, l'algoritmo di Dijkstra prende in input il grafo e i parametri p e d. Quindi, partendo dal punto di partenza p, esamina tutti gli archi disponibili (ovvero tutti i collegamenti fra i nodi), calcolando la distanza da ciascuno fino alla destinazione. Si tiene traccia della distanza minima per raggiungere ogni nodo lungo il percorso. L'algoritmo, quindi, sceglie il nodo successivo più vicino alla destinazione e continua questo processo fino a quando non si raggiunge la meta prescelta. Come output viene restituita una lista di nodi che l'utente deve attraversare per arrivare a destinazione nel minor tempo possibile. In Figura 2.17 è possibile visualizzare l'implementazione in Python dell'algoritmo di Dijsktra. Il metodo *dijsktra* prende come parametri un grafo, il nodo di partenza e quello di destinazione e restituisce una lista di nodi che devono essere attraversati per arrivare al nodo di destinazione.

```
def dijkstra(graph, start, target):
    distances = {vertex: float('inf') for vertex in graph}
    distances[start] = 0
    previous = {vertex: None for vertex in graph}
    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_vertex == target:
            break

        if current_distance   distances[current_vertex]:
            continue

        for neighbor in graph.get_neighbors(current_vertex):
            weight = graph.get_weight(current_vertex, neighbor)
            distance = current_distance   weight

            if distance   distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_vertex
                heapq.heappush(priority_queue, (distance, neighbor))

    if previous[target] is None:
        return None

    path = []
    current = target
    while current is not None:
        path.insert(0, current)
        current = previous[current]

    return path
```

Figura 2.17: Implementazione dell'algoritmo di Dijsktra

Il metodo *dijsktra* della classe *Graph*, viene chiamato all'interno del metodo *calculateRoute* appartenente alla classe *NavigationService*, la cui implementazione in Python può essere

visualizzata in Figura 2.18. Questo metodo prende come parametri in input il punto di partenza e quello di destinazione, entrambi di tipo stringa. Esso restituisce in output un oggetto *Navigation*, la cui implementazione è definita in Figura 2.19.

```
def calculateRoute(self, idSource: str, idTarget: str) -> Navigation:
    intPoint = self.navRepo.getInterestPointById(idSource)
    floor = self.navRepo.getFloorById(intPoint.idFloor)
    nodes = self.navRepo.getNodesOfBuilding(floor.idBuilding)
    allCheckpoints = dijkstra(nodes, idSource, idTarget)
    route: List[Checkpoint] = []
    for i in range(0, (len(allCheckpoints)  1), 3):
        intPointCheck = self.navRepo.getInterestPointById(allCheckpoints[i])
        check = Checkpoint(intPointCheck, i)
        route.append(check)

    target = self.navRepo.getInterestPointById(idTarget)
    check = Checkpoint(target, i)
    route.append(check)
    nav = Navigation(str(uuid.UUID), intPoint, target, route)
    return nav
```

Figura 2.18: Implementazione della funzione calculateRoute

```
class Navigation:
    id = ""
    source: InterestPoint
    target: InterestPoint
    position: Position
    route: List[Checkpoint]
    checkpointPassed: List[Checkpoint]
    type: TypeNavigation

    def __init__(self, id, source, target, route) -> None:
        self.id = id
        self.source = source
        self.target = target
        self.route = route
        self.checkpointPassed = []
        self.position = Position(source.coordinateX, source.coordinateY)
        self.type = TypeNavigation.RouteLoaded.value
```

Figura 2.19: Implementazione della classe Navigation

L'obiettivo dell'applicazione NavigaUNIVPM è quello di fornire all'utente finale una esperienza coinvolgente e di guidarlo in maniera facile e intuitiva da un punto di partenza a uno di arrivo. Infatti, come tecnologia per lo sviluppo della navigazione nella parte frontend si è scelto di utilizzare il motore grafico Unity. Esso ha rivestito un ruolo fondamentale nella creazione dell'applicazione in realtà aumentata. In Figura 2.20 è possibile visualizzare una panoramica generale dell'IDE (Integrated Development Environment) Unity. In alto a sinistra sono presenti le componenti che costituiscono la schermata, responsabili della gestione degli eventi e dell'interazione con l'utente. Nella parte inferiore è presente una anteprima di come saranno visualizzate le varie schermate su un dispositivo mobile.

La realtà aumentata (AR) è un'esperienza interattiva che combina il mondo reale con i contenuti generati al computer. Tali contenuti possono spaziare attraverso molteplici modalità sensoriali, comprese quelle visive, uditive, tattili, somatosensoriali e olfattive. Grazie alla piattaforma di sviluppo di Unity, è stato possibile creare e gestire contenuti AR, tra cui modelli 3D, animazioni e testi, integrandoli in modo realistico nell'ambiente reale.

In Figura 2.21 è possibile visualizzare il metodo che è stato implementato in Unity utilizzando il linguaggio di programmazione C#, il quale gestisce la visualizzazione del percorso

Figura 2.20: Panoramica generale IDE Unity

durante la navigazione da un punto di partenza a uno di destinazione. Questo metodo calcola la posizione del prossimo checkpoint e istanzia il prefabbricato appropriato nella scena Unity. Infatti, a seconda del numero di checkpoint rimanenti nella lista *Navigation.chekpoints*, viene istanziato il prefabbricato del checkpoint o del punto di arrivo nella posizione calcolata precedentemente. Dopo aver visualizzato il checkpoint, il primo elemento nella lista viene rimosso e la posizione del prossimo checkpoint viene impostata come la nuova *TargetPosition* per il componente *CalculatePath*.

```
public void SetNextCheckpoint(){
    Vector3 pos;

    try{
        pos =
            new Vector3(
                float.Parse(Navigation.checkpoints[0].coordinateX, CultureInfo.InvariantCulture),
                0,
                float.Parse(Navigation.checkpoints[0].coordinateY, CultureInfo.InvariantCulture)
            );
    }
    catch{
        pos = Vector3.zero;
    }

    var x = Navigation.checkpoints;

    if (Navigation.checkpoints.Count == 1)
        Instantiate(prefabArrivo, pos, Quaternion.identity);
    else
        Instantiate(prefabCheckpoint, pos, Quaternion.identity);

    try{
        Navigation.RemoveFirstCheckpoint();
    }
    catch{
    }

    calculatePath.TargetPosition = pos;
}
```

Figura 2.21: Implementazione del metodo per la visualizzazione del percorso in Unity

CAPITOLO 3

Raccolta e analisi delle recensioni

Nei paragrafi che seguono viene presentato il problema della raccolta e dell'analisi delle recensioni degli utenti. Viene proposta, successivamente, una soluzione che utilizza tecniche di Machine Learning per il processamento del linguaggio naturale (NLP), e vengono mostrate le modifiche effettuate al progetto iniziale dell'applicazione, presentata nel Capitolo 2.

3.1 Descrizione del problema

Nella seguente sezione viene presentato il problema principale che è stato affrontato durante il lavoro di tirocinio. Una funzionalità essenziale mancante all'applicazione NavigaUNIVPM è la raccolta e l'analisi delle recensioni utente. Ciò è importante per conoscere il pensiero di chi utilizza l'applicazione e, a seguito di una analisi più approfondita, per scoprire l'esistenza di qualche malfunzionamento.

La raccolta delle recensioni può essere considerata come un problema più semplice da affrontare e risolvere rispetto a quello dell'elaborazione e dell'analisi delle stesse. Infatti, ci si è trovati a dover scegliere un modello di intelligenza artificiale che fosse capace di "comprendere" le recensioni in lingua italiana e di fornire delle informazioni importanti come, ad esempio, la classificazione della loro polarizzazione in negativa, neutra o positiva.

La raccolta e l'analisi delle recensioni rappresentano dei processi rilevanti perché oggigiorno è diventato davvero importante conoscere il pensiero dell'utente su un determinato servizio o prodotto (in questo caso l'applicazione NavigaUNIVPM). Le applicazioni vengono sviluppate per essere vendute – o messe a disposizione in maniera gratuita – ad un pubblico sempre più ampio ed è importante, quindi, conoscere il feedback dell'utente e cercare di risolvere eventuali errori o "insoddisfazioni" nell'utilizzo delle stesse. Ovviamente sarebbe impossibile, avendo un'enorme quantità di recensioni, passarle in rassegna una ad una. Di conseguenza, l'alternativa più semplice per farlo in maniera efficiente è di affidarsi a delle soluzioni che permettano di effettuare una analisi e di estrapolare le informazioni essenziali in pochissimo tempo e senza richiedere un grande sforzo da parte di una persona umana. Il problema principale, quindi, oltre alla raccolta delle recensioni, è quello di trovare una soluzione che permetta di analizzare ed elaborare una grande mole di dati, prevalentemente testuali, in poco tempo e con una elevata – o soddisfacente – accuratezza senza alcun intervento umano. Il risultato di questo processo (quindi la classificazione del feedback in positivo, negativo o neutro), poi, dovrà poter essere visualizzabile in maniera chiara da un amministratore di sistema tramite una piattaforma (web o mobile). L'amministratore dovrà essere in grado di

riuscire a comprendere le informazioni necessarie per una analisi sull'andamento generale dell'applicazione e sul pensiero degli utilizzatori finali.

Il lavoro presentato è stato principalmente quello di trovare un modello di intelligenza artificiale che permettesse di analizzare un input testuale e di fornire come output la classificazione dell'input stesso. Per adattare il modello nella comprensione e nella classificazione delle recensioni lasciate dall'utente nell'applicazione, è stato necessario eseguire un processo di "fine tuning". Questo consiste nell'allenamento del modello su un set di dati appropriato come ad esempio, nel corrente contesto, un set di recensioni in lingua italiana. Tutto ciò è importante perché permette al modello di adattarsi ad un compito specifico che gli viene richiesto.

3.1.1 Analisi dei requisiti

Dopo aver compreso appieno le esigenze e individuato il problema da risolvere, è stata condotta un'ulteriore analisi dei requisiti per valutare la necessità di eventuali aggiunte. Durante questo processo, è emersa l'esigenza di introdurre nuovi requisiti relativi alla raccolta, alla classificazione e alla visualizzazione delle recensioni lasciate dall'utente nell'applicazione. I requisiti che sono stati introdotti, scritti in maniera appropriata, sono:

Area: Gestione feedback utente	Descrizione requisito
RF13: Raccogli feedback utente	Il sistema deve gestire la raccolta del feedback da parte dell'utente e dei dati di ricontatto.
RF14: Classifica feedback utente	Il sistema deve gestire la classificazione del feedback da parte dell'utente, categorizzandolo come negativo, positivo o neutro.
RF15: Visualizza statistiche	Il sistema deve visualizzare a schermo le statistiche riguardanti l'utilizzo dell'applicazione, fra cui la polarizzazione generale delle recensioni lasciate dagli utenti tramite l'applicazione mobile.

È fondamentale, infatti, sia che le recensioni vengano elaborate in maniera appropriata dal modello di intelligenza artificiale, sia che le informazioni restituite possano essere visualizzabili da un amministratore. Tutto ciò è importante affinché sia possibile comprendere quali sono le possibili mancanze o gli errori, in maniera tale da poterli risolvere in un secondo momento. Non sono stati aggiunti ulteriori requisiti funzionali oltre a quelli già evidenziati in fase di analisi.

3.2 Soluzione proposta

In Figura 3.1, è possibile visualizzare il flusso delle azioni che devono essere compiute affinché il feedback possa essere inserito dall'utente, salvato, elaborato dal modello e visualizzato in una dashboard apposita.

La soluzione proposta per soddisfare i requisiti riguardanti l'inserimento e l'analisi delle recensioni degli utenti prevede che, quando l'utente arriva a destinazione, l'applicazione mostri a schermo un form in cui è possibile inserire una recensione, sia sotto forma di input testuale che attraverso la selezione di un numero di stelle in base all'esperienza d'uso dell'applicazione. Una volta che la recensione è inviata, deve essere salvata all'interno del database di Firebase (database non relazionale). Le recensioni devono poi essere recuperate nella parte di backend per l'elaborazione in modo da poterne estrapolare le informazioni essenziali ai fini dell'analisi, come ad esempio la classificazione delle recensioni (sentiment analysis) oppure l'estrazione delle keywords (parole presenti con maggiore frequenza nelle recensioni). I risultati dell'elaborazione di questi dati devono essere resi disponibili, tramite degli endpoint, in maniera che possano essere recuperati attraverso chiamate HTTP in un secondo momento. I dati, successivamente, devono essere riordinati e visualizzati su una dashboard web per poter essere fruibili dagli amministratori.

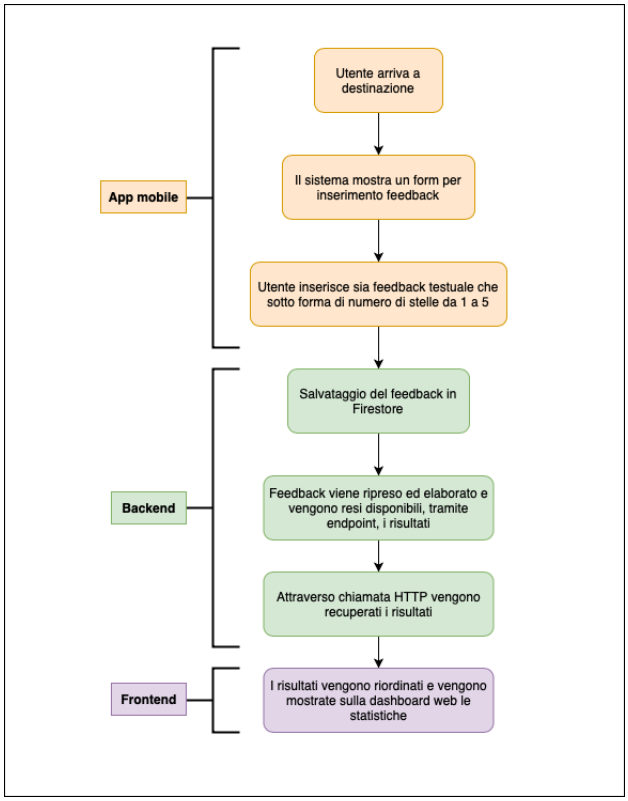

Figura 3.1: Diagramma di flusso per la soluzione proposta

Il processo di analisi del sentiment è svolto da un modello di intelligenza artificiale in grado di comprendere ed estrarre informazioni significative dalle recensioni. Per adattare il modello al compito ad esso assegnato, è stato necessario eseguire l'addestramento del modello stesso. Da questa considerazione nasce l'esigenza di trovare un dataset adeguato su cui eseguire il processo di "fine-tuning" del modello.

Si dovrà pensare, infine, anche ad un modo per inserire delle recensioni in maniera fittizia su cui eseguire la valutazione delle prestazioni del modello. Infatti, essendo l'applicazione NavigaUNIVPM non in commercio, dovranno essere gli stessi sviluppatori a dover inserire le recensioni tramite un form apposito.

3.2.1 Integrazione della progettazione

A seguito dei nuovi requisiti emersi, è stato necessario integrare alcune modifiche alla progettazione iniziale. Queste hanno riguardato, prevalentemente, i diagrammi dei casi d'uso, quelli delle classi e di attività. Il diagramma dei casi d'uso inserito nella progettazione si riferisce alla gestione delle recensioni utente. Il nuovo diagramma, rappresentato in Figura 3.2, è utile per descrivere come l'utilizzatore può interfacciarsi con il sistema, sia per inserire la valutazione sotto forma di selezione di un numero di stelle da 1 a 5, sia per inserire la recensione in modo testuale. Quest'ultima deve essere poi analizzata e classificata in maniera opportuna.

Per quanto riguarda il diagramma delle classi di progettazione si è eseguita la modifica della classe *Feedback*. Da come si può osservare nella Figura 3.3, sono stati aggiunti i seguenti attributi privati: l'identificativo della recensione, il suo contenuto testuale, il timestamp in cui viene inserita, i punti di partenza e di destinazione e il numero di stelle. Per tutti questi attributi, inoltre, sono stati aggiunti i metodi pubblici: get, utile per il recupero dei dati, e set per la modifica o inizializzazione dei valori degli attributi stessi.

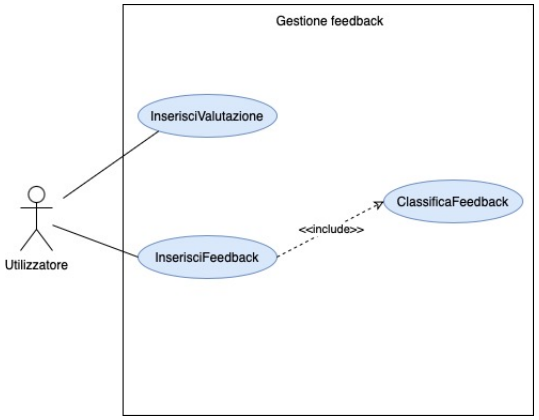

Figura 3.2: Diagramma dei casi d'uso: gestione delle recensioni

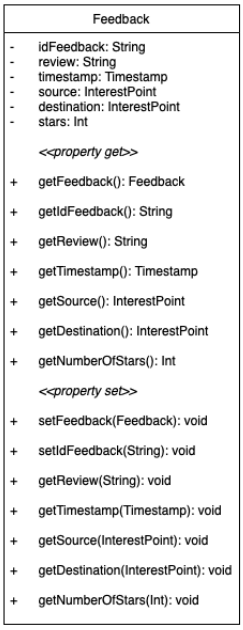

Figura 3.3: Classe Feedback

Seguendo il pattern architetturale Service Repository, è stata creata la classe *FeedbackRepository* che definisce la repository per recuperare i dati dal database. Questa classe eredita dalla classe *BaseRepository* il collegamento con il database e quindi può riferirsi ad esso senza doverlo istanziare nuovamente. I metodi creati sono:

- $getAllFeedbacks()$: restituisce un array contenente tutte le recensioni memorizzate nel database.
- $getFeedbacksByStar(Int)$: prende come parametro un intero (da 1 a 5), rappresentante il numero di stelle, e restituisce tutte le recensioni che hanno una valutazione con quel numero di stelle.
- $getFeedbackById(String)$: prende come parametro una stringa e ritorna la recensione (se esistente) con essa come identificativo.

- *getStarsNumber*(): restituisce un array di dimensione pari a 5 in cui, ad ogni posizione, è associato il numero di stelle; ad esempio, nella prima posizione, è presente il numero di stelle pari ad 1.

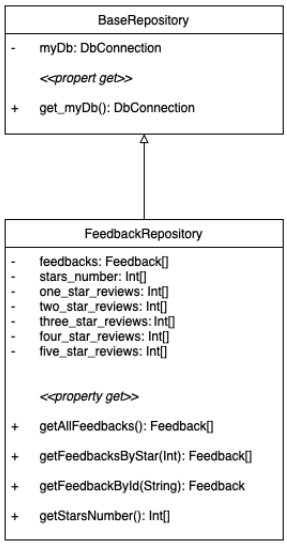

Figura 3.4: Classe Feedback Repository

Come fatto anche per la classe *Navigation* (vedere cap.2), è stata creata la classe *FeedbackService* che implementa l'interfaccia *IFeedbackService*, utile per effettuare tutte le operazioni necessarie per l'elaborazione delle recensioni. I metodi presenti nella classe *FeedbackService* saranno chiamati nell'API (Application Programming Interface) che riceverà i risultati delle operazioni presenti in ogni metodo. Nello specifico, i metodi presenti in questa classe sono:

- *getFeedbacks*(): richiama il metodo *getAllFeedbacks*() della classe "FeedbackRepository" e ottiene, quindi, un array contenente tutte le recensioni presenti nel database.
- *calculateSentiment*(*String*): prende come parametro una stringa, rappresentante la recensione inserita dall'utente durante la compilazione del form nell'applicazione, e restituisce un intero che sarà pari a 1 se la recensione viene classificata come positiva, 2 se neutra oppure 3 se negativa. Questo metodo utilizza un modello di intelligenza artificiale per l'inferenza.
- *calculateSentimentNumbers*(*DictInt*, *String*): prende come input un dizionario che ha come chiave una stringa rappresentante la recensione e un intero rappresentante il sentiment associato, calcolato nel precedente metodo tramite il modello di IA. Questo metodo restituisce un array di interi di dimensione tre, contenente, rispettivamente, il numero di recensioni positive, neutre e negative.
- *extractKeywordsFromFeedbacks*(): restituisce un dizionario contenente tutte le parole chiave rispetto a tutte le recensioni inserite dagli utenti. Ad ogni keyword è associato il numero di occorrenze trovate e la media del sentiment associato. Se consideriamo, ad esempio, un numero totale di tre recensioni in cui è presente in tutte e tre la keyword

"navigazione" e i sentiment associati sono positivi per la prima e la seconda recensione mentre neutro per la terza, allora si avrà:

$$\{\text{“}keyword\text{”} : navigazione, \text{“}occurences\text{”} : 3, \text{“}mean - sentiment\text{”} : positive\}$$

Anche per l'esecuzione di questa operazione si dovrà far ricorso ad un modello di intelligenza artificiale.

- $calculateSentimentNumbersByStar(Int)$: prende come parametro un intero che rappresenta il numero di stelle (da 1 a 5). Il metodo restituisce un array di interi di dimensioni tre contenente, rispettivamente, il numero di recensioni positive, neutre e negative che hanno come numero di stelle quello in input al metodo stesso.

Tutti questi metodi elencati sono fondamentali per visualizzare, successivamente, le informazioni nella dashboard web e per avere una visione generale e chiara del pensiero degli utenti riguardo l'applicazione.

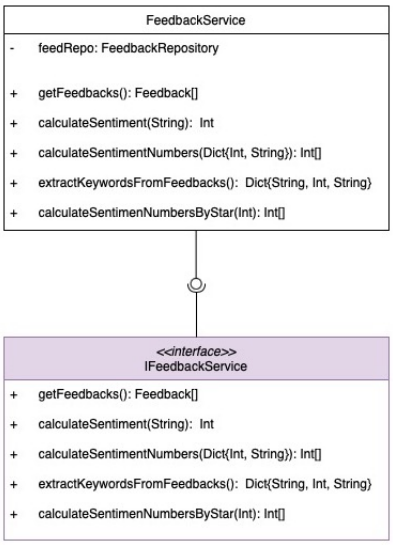

Figura 3.5: Classe Feedback Service

La modifica essenziale nei diagrammi di attività è stata l'aggiunta del diagramma in Figura 3.6. Esso descrive il flusso delle attività per l'inserimento e la visualizzazione della recensione lasciata dall'utente. Si nota che, fin quando l'utente non ha inserito sia la recensione testuale che sotto forma di numero di stelle, il sistema "non avanza" e continua a chiedere all'utente di eseguire le due operazioni. Dopo aver fatto ciò, tralasciando le operazioni per la memorizzazione della recensione all'interno del database, viene eseguita la sentiment analysis per poi effettuare la visualizzazione, delle statistiche corrispondenti, su una dashboard web.

3.2.2 Esperienza utente

A seguito delle modifiche nella parte di progettazione, con i relativi diagrammi, sono state effettuate anche delle aggiunte nella parte di interfaccia utente. Si è deciso di seguire la direzione presa inizialmente, ossia progettare un design il più "user-friendly" possibile. In questa sezione vengono presentati i mockup sia della schermata per l'inserimento del feedback nell'app mobile, sia della dashboard web.

Il mockup della schermata dell'applicazione mobile, sviluppata per la raccolta delle recensioni, ha come obiettivo principale quello di fornire agli utenti un'interfaccia intuitiva

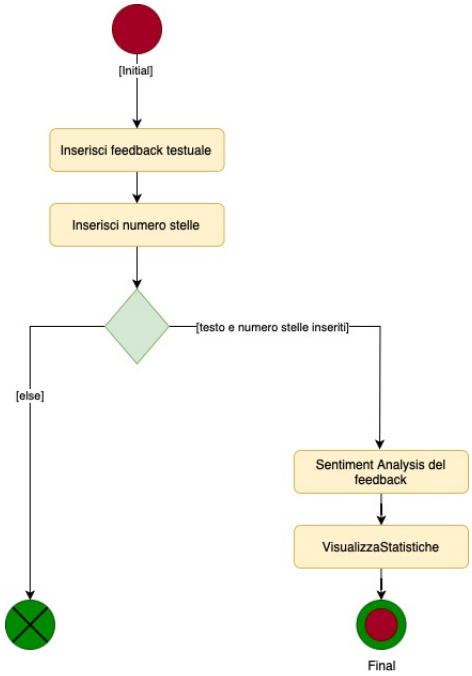

Figura 3.6: Diagramma di attività: inserimento e visualizzazione feedback

ed efficiente per esprimere le loro opinioni e le loro esperienze con l'applicazione. Si può notare la presenza di pochi componenti grafici ma efficaci allo scopo di inserimento del feedback, in maniera tale che l'utente non si trovi confuso e senza comprendere bene cosa deve fare. Nello specifico, l'interfaccia offre una semplice area testuale dove gli utenti possono digitare la loro recensione e cinque icone cliccabili a forma di stella, che consentono loro di valutare l'applicazione in base alla propria esperienza.

Figura 3.7: Mockup app mobile

Di seguito, invece, viene presentato il design della dashboard web, progettata per la visualizzazione delle recensioni degli utenti e delle informazioni ottenute grazie al processamento delle stesse. La dashboard web ha come obiettivo quello di fornire agli amministratori uno strumento potente per monitorare e comprendere le opinioni degli utenti, nonché per estrarre insight significativi per migliorare continuamente l'esperienza.

Figura 3.8: Mockup dashboard web

Da come è possibile vedere in Figura 3.8, nella dashboard web si è deciso di inserire una tabella per la visualizzazione di tutte le recensioni con specificate: la data e l'ora in cui sono state inserite, il contenuto testuale insieme al numero di stelle e il punto di partenza e di destinazione dell'istanza di navigazione in cui è stata inserita. È presente anche una tabella in cui vengono visualizzate le parole chiave estrapolate dal modello di intelligenza artificiale insieme al numero di occorrenze (quante volte vengono incontrate in tutte le recensioni) e al sentimento medio a cui sono associate. Come grafici, si è optato per un "donut chart" per visualizzare la distribuzione percentuale del sentiment e un istogramma in cui, per ogni stella, sono presenti tre colonne. Quest'ultime specificano il numero di recensioni categorizzate come positive, neutre e negative. Utilizzando l'istogramma è possibile verificare il funzionamento del modello di intelligenza artificiale. Ad esempio, se si nota un'elevata quantità di recensioni positive associate a una sola stella o un elevato numero di recensioni negative associate a cinque stelle, ciò potrebbe indicare un malfunzionamento del modello. Ci si aspetta, infatti, che la maggioranza delle recensioni positive sia associata a cinque stelle, mentre le recensioni negative siano principalmente collegate a una stella.

3.3 Integrazione nel sistema

A seguito delle modifiche alla progettazione iniziale e allo sviluppo dei mockup, sia per la parte mobile che web, ci si è concentrati sulla fase di implementazione. Anzitutto, è stato necessario decidere le tecnologie e i modelli di intelligenza artificiale che più avrebbero soddisfatto le esigenze del problema considerato. Una volta fatte queste scelte è stato necessario integrare le parti di codice già presenti per ottenere il risultato finale.

3.3.1 Tecnologie utilizzate

In questa sezione vengono presentate le tecnologie utilizzate per l'implementazione della soluzone proposta. Si andrà più nello specifico sulle tecnologie non menzionate nel capitolo precedente. Verranno descritte, infatti, solamente quelle che hanno ricoperto un ruolo fondamentale nel processo di implementazione e integrazione della soluzione proposta per la Sentiment Analysis.

A differenza della modalità di salvataggio dei dati che viene utilizzata in tutte le parti dell'applicazione NavigaUNIVPM, in questo caso, per il salvataggio dei dati relativi alle recensioni, si è scelto di utilizzare "Firebase Realtime Database", messo a disposizione sempre da Firebase. Si è scelta questa alternativa perché ha permesso di memorizzare e di recuperare i dati in maniera molto agevole. I dati memorizzati, infatti, da come si può vedere in Figura 3.9, hanno una struttura molto simile al formato json e possono essere sia importati che esportati nello stesso formato. Ciò permette una manipolazione molto più semplice.

Figura 3.9: Struttura di Firebase Realtime Database

Come linguaggio di programmazione per la parte backend si è continuato ad utilizzare Python. Esso viene utilizzato anche per il recupero dei dati relativi alle recensioni nel database. Come già scritto nel capitolo precedente, per il recupero dei dati dal database, viene utilizzata l'API Firebase, il quale fornisce un SDK specifico per Python chiamato "firebase-admin".

Per la parte di frontend, viene utilizzato Unity per la realizzazione della schermata in cui è presente il form per l'inserimento della recensione da parte dell'utente. Grazie a Unity, infatti, è stato possibile costruire l'interfaccia grafica andando ad inserire tutti i componenti necessari come i bottoni, gli input text, e cosi via. Successivamente, è stata inserita la logica, ossia il comportamento di questi componenti grafici, utilizzando C# come linguaggio di programmazione. Ad esempio, al click del bottone di invio della recensione, è stata inserità una funzionalità che esegue un controllo per verificare se l'utente ha inserito sia la recensione testuale sia se ha scelto il numero di stelle. Sempre per la parte di frontend, inoltre, per la costruzione dell'interfaccia grafica della dashboard, sono stati utilizzati HTML e CSS, mentre JavaScript è stato impiegato come linguaggio di scripting per l'integrazione di grafici e tabelle dinamiche. Inoltre, si è scelto di utilizzare il framework Django nella parte di backend per

eseguire chiamate HTTP agli endpoint e recuperare i dati da visualizzare nella dashboard. Django è un web framework scritto in Python, progettato per semplificare lo sviluppo web raggruppando le varie funzionalità in un'ampia collezione di moduli riutilizzabili chiamati framework di applicazioni web. Gli sviluppatori utilizzano Django per organizzare e scrivere il codice in modo più efficiente, riducendo significativamente i tempi di sviluppo web.[1].

Infine, per quanto riguarda i modelli di intelligenza artificiale, si è scelto RoBERTa per il compito di Sentiment Analysis e KeyBERT per l'estrazione delle keyword. Il modello RoBERTa si basa sull'architettura dei Transformers. RoBERTa rappresenta un'evoluzione ottimizzata del modello BERT (Bidirectional Encoder Representations from Transformers) di Google. Questa versione è stata progettata per massimizzare le prestazioni attraverso una serie di miglioramenti nell'addestramento e nell'ottimizzazione dei parametri. Si è scelto questo modello per eseguire, prevalentemente, la classificazione delle recensioni. Infatti, adattando opportunamente RoBERTa al compito di Sentiment Analysis, è possibile passare in input un testo e ottenere, come output, la classificazione dello stesso come positivo, negativo o neutro. In Figura 3.10 è possibile visualizzare tutte le tecnologie utilizzate, sia per l'implementazione della soluzione per la Sentiment Analysis, sia per le altre funzionalità dell'applicazione NavigaUNIVPM.

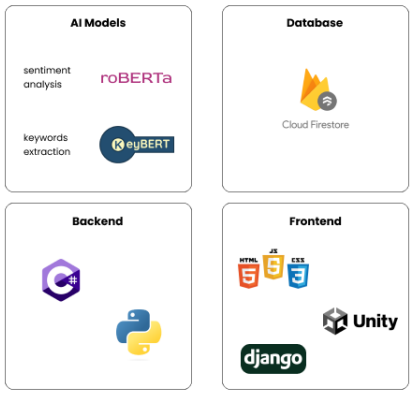

Figura 3.10: Tecnologie utilizzate

3.3.2 Modifiche al codice

Le modifiche effettuate al codice hanno riguardato sia la parte di frontend sviluppata in Unity e C#, sia la parte di backend che quella delle API (Application Programming Interface).

Per la parte di frontend, grazie a Unity, è stato possibile costruire la schermata per l'inserimento del feedback in maniera molto semplice. È bastato, infatti, utilizzare le componenti grafiche come bottoni o input text, messe a disposizione dal motore grafico. Per aggiungere il comportamento dinamico all'interfaccia, è stato utilizzato il linguaggio di programmazione C#. Per gestire la fase di inserimento e di salvataggio della recensione nel database, si è deciso di creare la classe *FeedScript.cs*. Ad esempio, da come si può vedere in Figura 3.11, è stata creata la funzione $feedBtnOnClick()$ che viene eseguita quando l'utente clicca il bottone di invio della recensione. Viene fatto un controllo per verificare che sia stata inserita sia la recensione testuale che sotto forma di stelle, altrimenti il sistema non avanza. Se, invece, l'utente ha svolto entrambe le azioni, vengono eseguiti i seguenti passaggi:

1. Vengono impostati i valori degli attributi *review* e *stars* della classe *Feedback.cs*.

[1]https://aws.amazon.com/it/what-is/django/

2. Viene chiamato il metodo *toJSON()* della classe *Feedback.cs* che trasforma in un formato Json gli attributi della classe stessa, ossia *source*, *destination*, *timestamp*, *review*, *stars*.

3. Il risultato del metodo *toJSON()* viene passato alla funzione *SaveData()* che salva i dati, in formato Json, all'interno del database.

4. Attraverso il comando *SceneManager.LoadScene("GreetingScene")*, viene caricata una nuova schermata in cui viene visualizzato un messaggio di ringraziamento per la recensione inserita.

```
public void feedBtnOnClick()
{
    UnityEngine.Debug.Log("Bottone cliccato");
    if (string.IsNullOrEmpty(feedInputTMP.text) || currentRating == 0)
    {
        UnityEngine.Debug.Log("Recensione non inserita oppure stelle non messe");
    }
    else
    {
        string review = feedInputTMP.text;
        feed.SetReview(review);
        feed.SetRating(currentRating);
        string json = feed.toJSON()
        SaveData(json);
        UnityEngine.Debug.Log("Recensione inserita: " + review);
        SceneManager.LoadScene("GreetingScene");
    }
}
```

Figura 3.11: Funzione C# da eseguire al click del bottone

Le modifiche effettuate nel codice del backend non hanno coinvolto parti già esistenti. Infatti, seguendo il pattern Service-Repository, è stato necessario creare solamente le classi *FeedbackRepository*, per il recupero dei dati dal database, *FeedbackService*, per l'esecuzione di funzioni e metodi e *Feedback* per modellare in maniera efficace ogni singola recensione. Ad esempio, nella classe *FeedbackService* è stato creato il metodo *extractKeywordsFromFeedbacks*, in Figura 3.12, per l'estrazione delle keyword dalle recensioni inserite dagli utenti, utilizzando il modello di intelligenza artificiale KeyBERT. Questo metodo, come insieme ad altri per l'elaborazione dei dati, viene chiamato nella parte di API per rendere disponibili le informazioni nella dashboard web.

Per la parte di API, gli endpoint che sono stati esposti per il recupero dei dati tramite chiamate HTTP sono:

- **/calculateSentimentRoBERTa**

  Questo endpoint esegue l'analisi del sentiment sulle recensioni degli utenti utilizzando il modello RoBERTa. Restituisce i risultati del sentiment analysis, compresi i sentiment positivi, neutri e negativi.

- **/getFeedbacks**

  Questo endpoint restituisce tutte le recensioni degli utenti con le relative informazioni, come id, timestamp, corpo del testo della recensione, numero di stelle, punto di partenza e di destinazione.

- **/extractKeywordsFromFeedbacks**

  Questo endpoint estrae le parole chiave dalle recensioni degli utenti utilizzando il modello KeyBERT.

- **/getStarsNumber**

  Questo endpoint restituisce un dizionario contenente cinque numeri che rappresentano il numero di stelle, da 1 a 5, inserite dagli utenti.

```python
def extractKeywordFromFeedbacks(self) -> dict:
    feedbacks = self.feedbackRepository.getAllFeedbacks()
    kw = KeyBERT()
    italian_stopwords = set(stopwords.words('italian'))

    word_occurrences = defaultdict(int)
    word_sentiments = defaultdict(list)
    result = {}

    for feedback in feedbacks.values():
        if isinstance(feedback, dict):
            review = feedback.get('review', "")
            if review:
                sentiment = self.calculateSentiment(review)
                review_words = review.split()
                clean_review = [word for word in review_words if word.lower() not in italian_stopwords]
                keywords = kw.extract_keywords(" ".join(clean_review), keyphrase_ngram_range=(1, 1))
                for word, _ in keywords:
                    if word == 'None Found':
                        continue
                    word_sentiments[word].append(sentiment)
                    word_occurrences[word] += 1

    for word, occurrences in word_occurrences.items():
        sentiments = word_sentiments[word]
        mean_sentiment_score = round(sum(sentiments) / len(sentiments))
        if mean_sentiment_score == 1:
            mean_sentiment = "Positive"
        elif mean_sentiment_score == 2:
            mean_sentiment = "Neutral"
        elif mean_sentiment_score == 3:
            mean_sentiment = "Negative"
        else:
            mean_sentiment = "Unknown"
        result[word] = {"occurrences": occurrences, "meanSentiment": mean_sentiment}

    return result
```

Figura 3.12: Funzione Python per estrazione delle keyword dalle recensioni

- **/calculateSentimentNumbersByStar/numberOfStar**

  Questo endpoint restituisce informazioni sul numero di sentiment positivi, neutri e negativi associati a una determinata stella. Il parametro `numberOfStar` rappresenta il numero di stelle da 1 a 5.

  Esempio di utilizzo:
  `GET /calculateSentimentNumbersByStar/3`

3.4 RoBERTa

Come accennato nelle precedenti sezioni, il modello di intelligenza artificiale utilizzato per il compito di Sentiment Analysis è RoBERTa (Robustly optimized BERT approach), sviluppato da Facebook AI. È un modello linguistico basato sull'architettura dei Transformers ed è ottimizzato rispetto a BERT, sviluppato dal team di Google. Ci sono diversi aspetti che differenziano questi due modelli, tra cui:

1. **Dimensione del modello**: RoBERTa ha un numero maggiore di parametri rispetto a BERT, rendendolo più potente ma anche più complesso e con la necessità di maggiori risorse di calcolo.

2. **Processo di pre-training**: RoBERTa ha avuto accesso a una mole di dati testuali significativamente maggiore rispetto a quelli utilizzati per BERT, migliorando notevolmente la sua capacità di comprendere e generalizzare il linguaggio naturale. Infatti, è stato

utilizzato un dataset molto ampio che include testi provenienti da diverse fonti come libri, articoli scientifici e l'ampio corpus di Common Crawl [2].

3. **Etichette di mascheramento (masking labels)**: BERT utilizza una tecnica di "static masking" delle etichette, mentre RoBERTa impiega un approccio dinamico. In quest'ultimo, le parole vengono mascherate seguendo una logica specifica e non in maniera casuale come avveniva con BERT.

RoBERTa utilizza il meccanismo di self-attention per elaborare sequenze di input e generare rappresentazioni contestualizzate delle parole in una frase. Per l'analisi delle keywords, invece, è stato utilizzato il modello KeyBERT.

3.4.1 Processo di fine-tuning

A seguito di alcuni test con delle recensioni in lingua italiana, si è osservato che il modello RoBERTa non riusciva a comprenderle con sufficiente precisione. Di conseguenza, il processo di classificazione non era abbastanza soddisfacente. Per risolvere questo problema è stato eseguito un processo di fine-tuning per adattare il modello al contesto di interesse. Questo processo ha coinvolto l'addestramento su un dataset di recensioni in lingua italiana utilizzando PyTorch come framework di Deep Learning. PyTorch permette di rappresentare dati molto complessi tramite i cosiddetti "tensori". I tensori possono essere pensati come una generalizzazione delle matrici, aventi un numero arbitrario di dimensioni.

Le operazioni principali che sono state svolte nel processo di addestramento sono:

- **Caricamento del dataset**: il dataset utilizzato per la fase di addestramento è stato scelto per la sua struttura e per le recensioni in lingua italiana[3]. In Figura 3.13 viene riportato il codice per il caricamento del dataset, in formato csv, utilizzando la libreria Pandas messa a disposizione da Python per l'analisi dei dati.

```
dataset = pd.read_csv("raw_data.txt", on_bad_lines='skip', engine='python')
dataset = dataset[['review_text', 'review_stars']]
```

Figura 3.13: Caricamento del dataset

- **Caricamento del tokenizer**: il tokenizer è importante durante le fasi di addestramento e di inferenza del modello perché permette di convertire il testo grezzo in un formato comprensibile e processabile dal modello stesso. In Figura 3.14 si può notare la definizione della funzione $tokenize\_fn(batch)$, a cui viene passato un testo e viene restituito, in output, il testo "tokenizzato", comprensibile dal modello.

```
task = 'sentiment'
MODEL = f'cardiffnlp/twitter-roberta-base-{task}'
```

```
tokenizer = AutoTokenizer.from_pretrained(MODEL)
```

```
def tokenize_fn(batch):
    return tokenizer(batch['text'], truncation=True, padding=True, max_length=256)
```

Figura 3.14: Caricamento del tokenizer e definizione della funzione per la "tokenizzazione"

[2] https://commoncrawl.org
[3] https://www.github.com/AlessandroGianfelici/italian_reviews_dataset

- **Caricamento del modello**: il modello RoBERTa viene caricato direttamento dal "Model Hub" fornito da HuggingFace. Da come si può notare in Figura 3.15, per il caricamento viene utilizzato il metodo $from\_pretrained()$ messo a disposizione dalla classe $AutoModelForSequenceClassification$ della libreria $Transformers$.

```
model = AutoModelForSequenceClassification.from_pretrained(MODEL, num_labels=3)
```

Figura 3.15: Caricamento del modello RoBERTa

- **Definizione degli argomenti per il training**: viene definito un oggetto $TrainingArguments$ che contiene tutti gli argomenti necessari per l'addestramento del modello. Nello specifico:
  - *output_dir*: specifica la directory su cui salvare i risultati dell'addestramento.
  - *evaluation_strategy*: definisce la strategia di valutazione durante l'addestramento. In questo caso indica che la valutazione viene eseguita alla fine di ogni epoca.
  - *per_device_train_batch_size*: definisce la dimensione del batch di addestramento per dispositivo. In questo caso significa che su ogni dispositivo (CPU, GPU) vengono elaborati 16 esempi contemporaneamente durante l'addestramento.
  - *per_device_eval_batch_size*: specifica la dimensione del batch di valutazione per dispositivo. In questo caso è impostato su 64, indicando che per ogni didpositivo vengono valutati 64 esempi contemporaneamente.

```
training_args = TrainingArguments(
    output_dir='training_dir',
    evaluation_strategy='epoch',
    save_strategy='epoch',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
)
```

Figura 3.16: Definizione degli argomenti per il training

- **Definizione delle metriche**: viene definita una funzione *compute_metrics* che calcola le metriche di valutazione del modello. Questa funzione restituisce l'accuratezza e il punteggio F1. Quest'ultima misura tiene in considerazione precisione e recupero del test, dove la precisione è il numero di veri positivi diviso il numero di tutti i risultati positivi, mentre il recupero è il numero di veri positivi diviso il numero di tutti i test che sarebbero dovuti risultare positivi (ovvero veri positivi più falsi negativi).

```
def compute_metrics(logits_and_labels):
        logits, labels = logits_and_labels
        predictions = np.argmax(logits, axis=-1)
        acc = np.mean(predictions == labels)
        f1 = f1_score(labels, predictions, average = 'micro')
        return {'accuracy': acc, 'f1_score': f1}
```

Figura 3.17: Definizione delle metriche di valutazione del modello

- **Creazione dell'oggetto Trainer**: viene creato un oggetto *Trainer* che si occupa di gestire l'addestramento del modello. Da come si può notare in Figura 3.18, gli attributi passati sono: il modello da addestrare, gli argomenti di addestramento definiti in precedenza, il dataset di addestramento e quello di valutazione, il tokenizer e la funzione per i calcolo delle metriche di valutazione del modello.

```
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset = tokenized_dataset["train"],
    eval_dataset = tokenized_dataset["test"],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

Figura 3.18: Creazione dell'oggetto Trainer per la gestione dell'addestramento

- **Avvio del training**: come descritto in Figura 3.19, l'addestramento del modello può essere avviato tramite il metodo *train*() della classe *Trainer*.

```
trainer.train()
```

Figura 3.19: Avvio dell'addestramento

3.4.2 Inferenza del modello

A seguito della fase di addestramento del modello RoBERTa, si sono valutate le sue prestazioni fornendo in input un dataset di valutazione. Il modello è riuscito a classificare le recensioni nel dataset con un buon livello di precisione. Di conseguenza, si è salvato il modello in un formato binario per l'utilizzo successivo nella fase di inferenza. In Figura 3.20 è possibile visualizzare i risultati di valutazione del modello su un set di dati di test durante la fase di valutazione. I parametri principali sono:

- *eval_loss*: rappresenta la perdita, ossia una misura dell'errore del modello rispetto ai dati di valutazione.
- *eval_accuracy*: misura della percentuale di esempi classificati correttamente dal modello.
- *eval_f1_score*: misura che combina i valori di precisione e di recall del modello in un solo punteggio. Il valore di recall indica la proporzione di veri positivi tra tutti i campioni che sono effettivamente positivi mentre la precisione indica la proporzione di veri positivi tra tutti i campioni classificati come positivi.

```
{'eval_loss': 0.18968340754508972,
 'eval_accuracy': 0.9478186484174508,
 'eval_f1_score': 0.9478186484174508,
 'eval_runtime': 208.5537,
 'eval_samples_per_second': 134.527,
 'eval_steps_per_second': 1.055,
 'epoch': 3.0}
```

Figura 3.20: Risultati di valutazione del modello

Seguendo i diagrammi delle classi di progettazione, si è creata la classe *FeedbackService* in cui è stato definito, oltre a quelli già descritti in Figura 3.5, il metodo *calculateSentiment*. Questo metodo si occupa di effettuare l'operazione di Sentiment Analysis su una stringa passata come parametro al metodo. Viene restituito, come output, un intero che sarà pari a 1 se la recensione viene classificata come positiva, 2 se neutra oppure 3 se negativa. Nello specifico, i passaggi che vengono effettuati sono:

1. **Caricamento del modello**: il modello su cui è stato eseguito il fine-tuning viene caricato grazie al metodo *load*() messo a disposizione dal framework PyTorch.

2. **Caricamento del tokenizer**: così come fatto in fase di addestramento, anche per l'inferenza c'è necessità di caricare il tokenizer per convertire l'input in un formato comprensibile e processabile da RoBERTa.

3. **Inferenza del modello**: vengono passati i dati "tokenizzati" al modello per eseguire l'inferenza. La variabile $output$ conterrà i logit, ossia i punteggi grezzi degli ultimi layer del modello.

4. **Estrazione e trasformazione dei punteggi**: i punteggi vengono estratti e trasformati in un array NumPy tramite le funzioni $detach()$ e $numpy()$. Successivamente, attraverso la funzione $softmax()$, vengono estratte le probabilità associate alla polarizzazione della recensione in positiva, neutra o negativa.

La variabile *max_sentiment* conterrà il valore della probabilità più alta fra i tre differenti elementi della lista *scores*. La definizione del metodo è rappresentata in Figura 3.21.

```
def calculateSentiment(self, review) -> any:
        MODEL_PATH = "D:/JupyterHub/ssacchetti/NavigaUNIVPM_backend/Api/pytorch_roberta_sentiment.bin"
        model = torch.load(MODEL_PATH, map_location=torch.device('cpu'))

        task = 'sentiment'
        MODEL = f'cardiffnlp/twitter-roberta-base-{task}'
        tokenizer = AutoTokenizer.from_pretrained(MODEL)

        feedback = review

        if feedback:
            encoded_input = tokenizer(str(feedback), return_tensors='pt')
            output = model(**encoded_input)
            scores = output[0][0].detach().numpy()
            scores = softmax(scores)
            scores = scores.tolist()
            max_sentiment = max(scores[0], scores[1], scores[2])

            if(max_sentiment == scores[2]):
                sentiment = 1
            else:
                if(max_sentiment == scores[1]):
                    sentiment = 2
                else:
                    if(max_sentiment == scores[0]):
                        sentiment = 3

        return sentiment
```

Figura 3.21: Metodo per inferenza del modello RoBERTa

CAPITOLO 4

Valutazione del sistema

In questa ultima sezione vengono presentati i risultati finali derivanti dall'implementazione del sistema di Sentiment Analysis nell'applicazione NavigaUNIVPM. Inoltre, viene dedicato un paragrafo alla valutazione della qualità del modello RoBERTa nella classificazione delle recensioni degli utenti, evidenziando i parametri utilizzati per eseguire una valutazione complessiva delle sue performance.

4.1 Funzionamento

Il sistema di Sentiment Analysis è stato integrato nell'applicazione NavigaUNIVPM per aggiungere una funzionalità essenziale al giorno d'oggi. Comprendere cosa pensano gli utenti di un determinato servizio o prodotto è fondamentale per il miglioramento continuo e per soddisfare eventuali richieste. In ambito imprenditoriale, è celebre la frase "il cliente ha sempre ragione". Questo principio, sebbene valido solo fino a un certo punto, deve essere considerato anche nello sviluppo di software. Un'applicazione è, infatti, paragonabile a qualsiasi altro prodotto sul mercato, e l'obiettivo è venderla soddisfacendo le aspettative dei potenziali clienti *target*.

Il funzionamento generale del sistema prevede che, una volta conclusa la navigazione da un punto di partenza a uno di destinazione interni all'università, l'utente possa inserire, a sua discrezione, una recensione sia sotto forma testuale sia tramite la selezione di un numero di stelle da 1 a 5 per esprimere la sua valutazione e il suo pensiero sull'utilizzo dell'applicazione. Questa recensione viene poi memorizzata all'interno del database per essere successivamente elaborata dal modello di intelligenza artificiale RoBERTa. I risultati saranno infine resi disponibili nell'interfaccia della dashboard web. Al momento, non è presente un'autenticazione per accedere alla dashboard, ma questa funzionalità potrebbe essere implementata in futuro affinché solo amministratori autorizzati possano visualizzare il pannello.

4.1.1 Applicazione mobile

In Figura 4.1 è possibile visualizzare il risultato finale dell'implementazione del form per l'inserimento delle recensioni utente nell'applicazione mobile. Come si era precedentemente deciso, l'utente può inserire sia una recensione sotto forma testuale che attraverso la scelta di un numero di stelle in base alla propria esperienza di navigazione fornita dall'applicazione stessa.

Figura 4.1: Implementazione del form su app mobile

4.1.2 Dashboard web

In Figura 4.2 è possibile visualizzare il risultato finale della dashboard web. Così come per l'implementazione della schermata per l'inserimento delle recensioni nell'applicazione mobile, anche in questo caso si è seguito il design del mockup che era stato sviluppato precedentemente. Sono stati inseriti, infatti, anche tutti i diagrammi e tabelle che erano state descritte nelle fasi precedenti all'implementazione vera e propria. Per uno sviluppo rapido e pulito dell'interfaccia grafica si è utilizzato Bootstrap, un framework open source che offre modelli di progettazione basati su HTML e CSS.

4.2 Qualità del modello

La valutazione della qualità del modello è una fase essenziale perché permette di comprendere le sue performance e di identificare eventuali criticità che potrebbero portare a risultati non accettabili. L'obiettivo del modello RoBERTa, nel contesto considerato, è quello di classificare con la massima accuratezza e precisione possibile le recensioni che gli utenti lasciano nell'applicazione mobile NavigaUNIVPM.

Una qualità elevata del modello è cruciale perché è fondamentale evitare che recensioni positive vengano erroneamente classificate come negative e viceversa. Analizzare correttamente i risultati ottenuti dall'elaborazione delle recensioni è importante per comprendere il pensiero generale degli utenti sull'applicazione e per identificare e correggere eventuali errori o bug segnalati dagli utenti stessi. A tal fine, nelle seguenti sezioni vengono presentati i passaggi seguiti per valutare le performance e la qualità del modello RoBERTa. Verranno illustrati i test effettuati, il modo in cui sono state generate le recensioni e i parametri del modello considerati in fase di valutazione.

4.2.1 Test effettuati

Per valutare le performance del modello, è stato anzitutto utilizzato un set di recensioni non incluso nella fase di addestramento. Questo approccio simula un contesto reale, nel

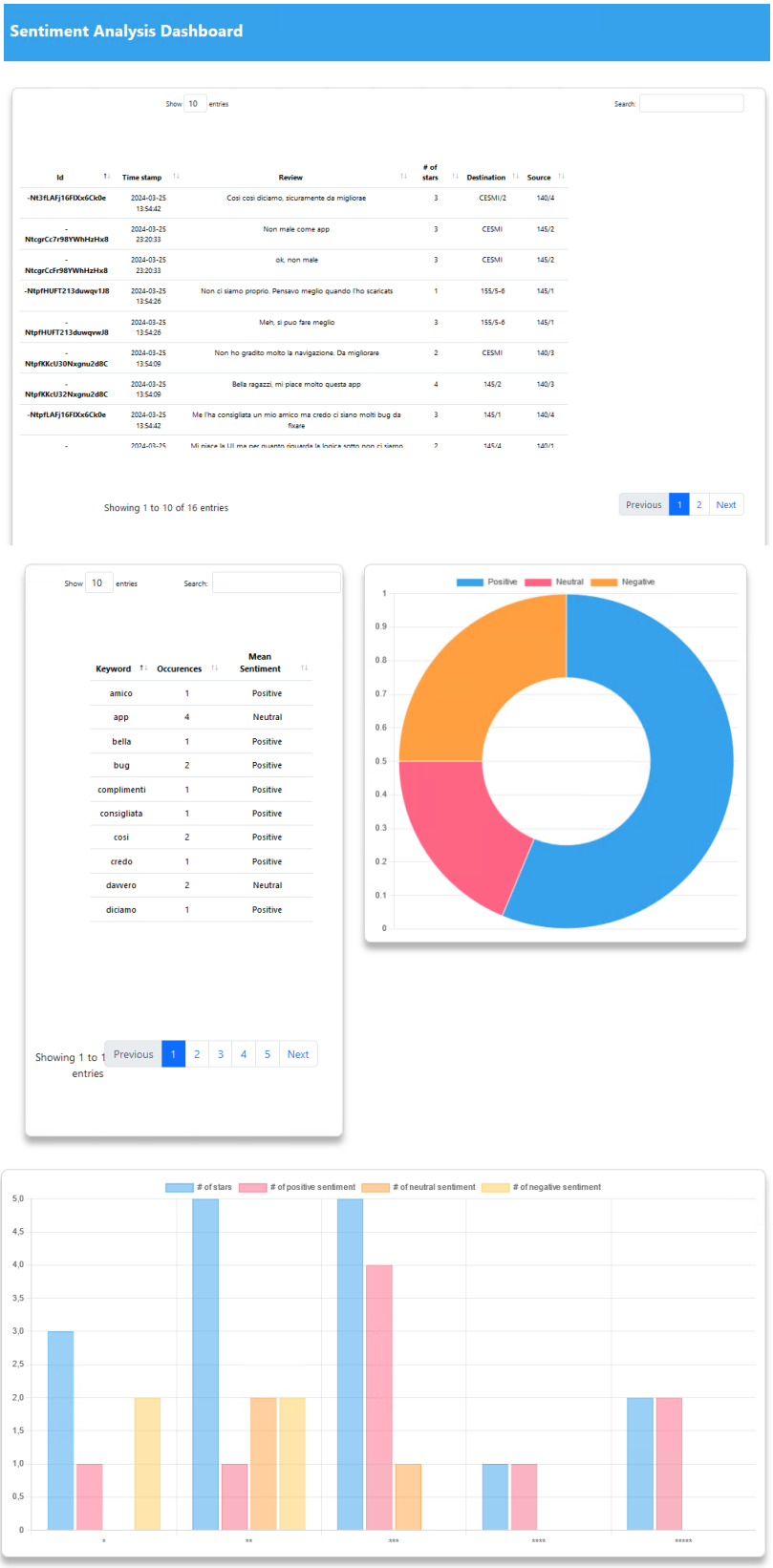

Figura 4.2: Implementazione della dashboard web

quale il modello elabora recensioni inserite dagli utenti, assicurando che il set di dati sia rappresentativo delle recensioni che il modello incontrerà nel suo utilizzo effettivo.

Successivamente, sono stati condotti test su dati rumorosi, ossia recensioni contenenti errori ortografici e grammaticali. Questo test è stato eseguito per simulare condizioni reali in cui gli utenti potrebbero commettere errori durante l'inserimento della recensione testuale. Le metriche di accuracy, precision e F1-score sono state confrontate tra i dati rumorosi e i dati puliti (ossia, recensioni senza errori) per valutare la robustezza del modello in presenza di rumore.

Infine, sono stati effettuati test sul tempo di risposta del modello. Questo aspetto è essenziale per garantire l'efficienza del modello, in modo che sia in grado di elaborare una recensione o un set di recensioni in un periodo di tempo ragionevole. Ciò è fondamentale per evitare ritardi significativi per l'amministratore che visualizzerà i dati nella dashboard.

4.2.2 Generazione delle recensioni utente

L'applicazione NavigaUNIVPM non è ancora disponibile nei vari store per essere scaricata da tutti e, di conseguenza, per valutare il comportamento del modello e del sistema in generale, è stato necessario aggiungere delle recensioni tramite il form nell'applicazione mobile in maniera "fittizia". Si è cercato di aggiungere recensioni che fossero il più differenti possibile tra loro, in modo da ottenere una panoramica generale e comprendere tutti i vari scenari. Di seguito vengono presentati i principali approcci seguiti nell'inserimento delle recensioni:

1. Inserimento di recensioni positive con numero di stelle appropriato: sono state inserite recensioni esplicitamente positive, con un numero di stelle che riflettesse correttamente la positività del contenuto (3, 4 o 5 stelle per recensioni particolarmente positive).

2. Inserimento di recensioni negative con numero di stelle appropriato: allo stesso modo, sono state inserite recensioni esplicitamente negative, con un numero di stelle che riflettesse la negatività del contenuto (3, 2 o 1 stella per recensioni particolarmente negative).

3. Inserimento di recensioni con incoerenza tra testo e numero di stelle: per testare ulteriormente il modello, sono state create recensioni in cui il sentiment del testo non corrispondeva al numero di stelle assegnato. Ad esempio, recensioni positive con 1 o 2 stelle e recensioni negative con 4 o 5 stelle.

4. Inserimento di recensioni con testo molto lungo: sono state inserite recensioni con un corpo testuale molto lungo per mettere alla prova il modello nella comprensione e classificazione di testi estesi.

Le recensioni inserite, dopo essere state salvate nel database, sono state elaborate dal modello RoBERTa per eseguire una valutazione complessiva delle performance e per comprendere il suo comportamento in un contesto il più realistico possibile.

4.2.3 Parametri del modello

La Figura 4.3 mostra l'andamento, durante le varie epoche, delle principali metriche di valutazione utilizzate per analizzare le prestazioni del modello. Le immagini riportate includono i punteggi di Recall, F1, MSE e RMSE, Accuracy e Precision. Queste metriche forniscono una visione dettagliata sull'accuratezza e l'efficacia del modello nel classificare correttamente le recensioni degli utenti.

In particolare, la metrica di Recall descrive la capacità del modello di identificare correttamente tutte le istanze positive, misurando quindi la proporzione di veri positivi rispetto al

totale delle istanze realmente positive. Un alto valore di Recall indica che il modello è efficace nel catturare la maggior parte delle istanze positive.

La metrica di Precision descrive la capacità del modello di identificare correttamente le istanze positive tra tutte quelle predette come positive. Un alto valore di Precision indica che il modello ha un basso tasso di falsi positivi, risultando affidabile nella classificazione delle istanze positive. Si può notare come, in questo caso, il valore di Precision raggiunge circa 0.80 nella terza epoca di addestramento e si stabilisce, infine, intorno a 0.75. Questo non è un valore altissimo ma possiamo definirlo come soddisfacente.

La metrica di F1 descrive la media armonica tra Precision e Recall, fornendo un equilibrio tra queste due metriche. L'F1 score è particolarmente utile quando è necessario un compromesso tra la capacità del modello di evitare falsi negativi e falsi positivi, soprattutto in presenza di classi sbilanciate, ossia classi in cui non si ha lo stesso numero di record.

La metrica di Accuracy descrive la proporzione di predizioni corrette rispetto al totale delle istanze valutate. È una metrica globale che misura la percentuale di tutte le predizioni, sia positive che negative, che il modello ha classificato correttamente.

Le metriche MSE (Mean Squared Error) e RMSE (Root Mean Squared Error) descrivono la differenza media quadratica tra i valori predetti e i valori reali. MSE calcola la media dei quadrati degli errori, mentre RMSE è la radice quadrata di MSE. Queste metriche sono utili per quantificare la precisione delle predizioni continue e indicano quanto distanti sono, in media, le predizioni dagli effettivi valori osservati. È possibile osservare come, in questo caso, alla decima epoca, i valori delle metriche MSE e RMSE siano abbastanza bassi e ciò si traduce in previsioni del modello abbastanza vicine ai valori reali.

Infine, il parametro Loss descrive la funzione di perdita utilizzata per ottimizzare il modello durante l'addestramento. La funzione di perdita quantifica quanto le predizioni del modello si discostano dai valori reali, fornendo un valore numerico che il modello cerca di minimizzare. Un valore di Loss basso indica una migliore performance del modello in termini di capacità di predire correttamente i dati di addestramento. In questo contesto, monitorare attentamente il parametro Loss durante l'addestramento è cruciale per garantire che il modello apprenda in modo efficace senza incorrere in fenomeni di overfitting, ossia l'adattamento eccessivo ai dati di addestramento, o underfitting, che indica una capacità limitata nel generalizzare il pattern dei dati.

La Tabella 4.1 riporta i valori finali dei parametri del modello considerati durante la fase di valutazione. Si può affermare come i risultati ottenuti siano abbastanza soddisfacenti.

Parametro di valutazione	**Valore finale**
Recall	0.73
Precision	0.74
F1 Score	0.73
Accuracy	0.94
MSE	0.30
RMSE	0.15
Loss	0.12

Tabella 4.1: Valori dei parametri di valutazione del modello

In Figura 4.4 è possibile visualizzare la Confusion Matrix. La Confusion Matrix è uno strumento utilizzato in Machine Learning e statistica per valutare le prestazioni di un modello di classificazione. Essa fornisce una rappresentazione tabellare delle predizioni effettuate dal modello rispetto ai valori reali osservati, permettendo di visualizzare non solo la precisione delle predizioni, ma anche i tipi di errori commessi.

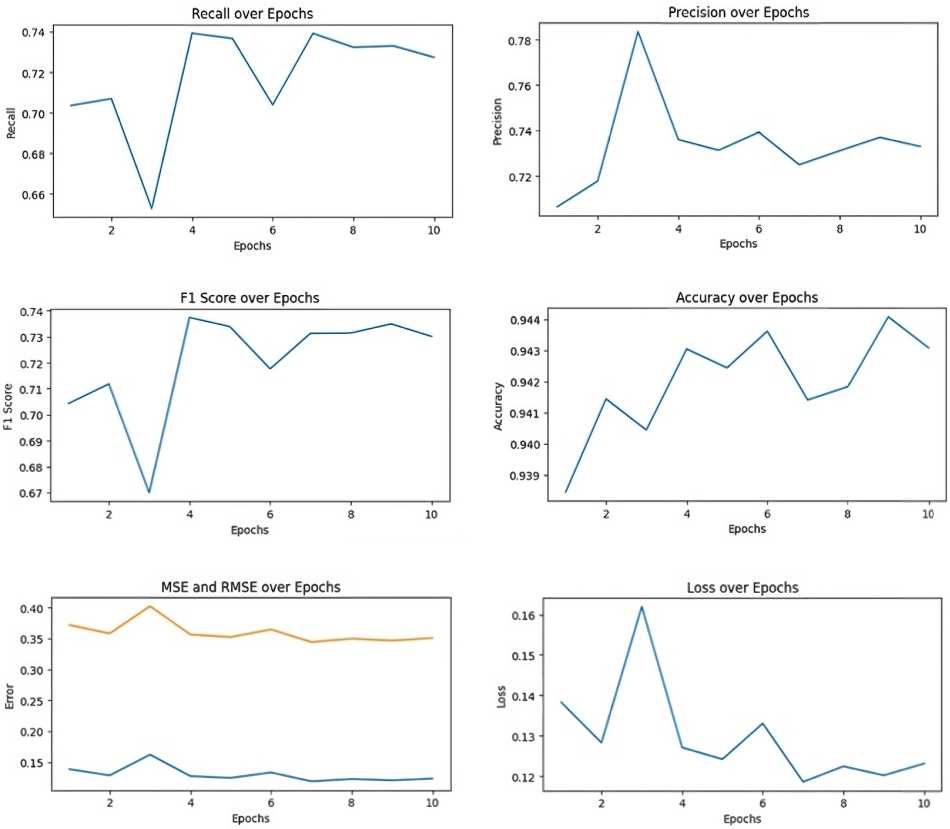

Figura 4.3: Risultati delle metriche di valutazione del modello

In questo caso, la matrice è di ordine 3x3, dato che le classi utilizzate nella classificazione sono tre: neutra, positiva e negativa.

È possibile osservare come il modello riesca a classificare in maniera abbastanza precisa sia le recensioni realmente neutre sia quelle realmente negative. Infatti la precisione del modello nella classificazione è pari a 89% per le recensioni negative e 98% per quelle neutre. Tuttavia, il modello si comporta peggio nella classificazione delle recensioni positive. Infatti, le recensioni realmente positive vengono predette come neutre nel 60% dei casi e come negative nel 30% dei casi. Quest'ultimo risultato non è accettabile, e sarebbe necessario eseguire una ulteriore fase di addestramento per migliorare la classificazione delle recensioni positive.

In conclusione, è possibile affermare che il modello si è comportato bene in fase di addestramento e nella valutazione delle sue performance, riuscendo a predire in maniera abbastanza efficace le classi reali a cui appartengono le recensioni degli utenti. Tuttavia, c'è ancora margine per migliorare, soprattutto per quanto riguarda la predizione delle recensioni positive. Come si può osservare dalla Confusion Matrix, il modello tende a classificare le recensioni realmente positive principalmente come neutre e, in misura minore, come negative. Una possibile soluzione per migliorare questa situazione potrebbe essere quella di effettuare un'ulteriore fase di addestramento del modello, modificando il dataset per includere un maggior numero di recensioni positive. Questo potrebbe aiutare il modello a riconoscere meglio le caratteristiche distintive delle recensioni positive, migliorando così la sua capacità di classificazione complessiva.

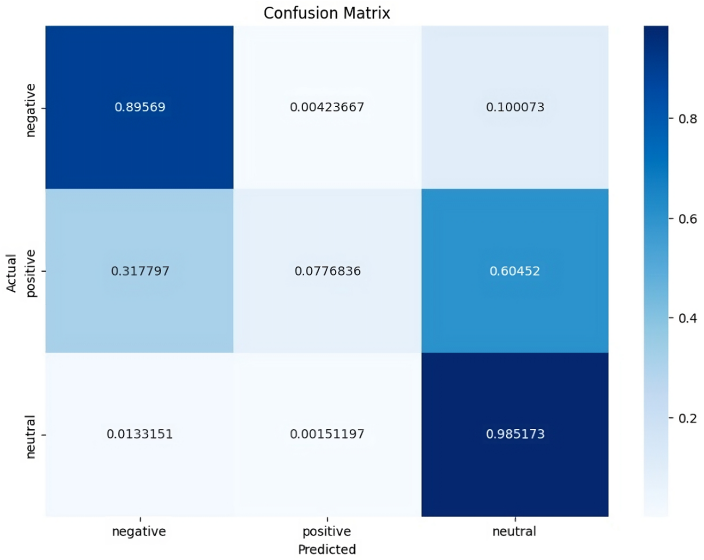

Figura 4.4: Confusion matrix

4.3 Analisi e discussione

Lo sviluppo del sistema di Sentiment Analysis da integrare nell'applicazione NavigaUNIVPM ha presentato alcune difficoltà, soprattutto nelle fasi iniziali. Questo tipo di sistemi richiede una conoscenza approfondita dei meccanismi di Machine Learning. Inoltre, per adattare un modello di Intelligenza Artificiale a svolgere un determinato compito, è necessario seguire dei passaggi che possono sembrare superficiali a un neofita. Ad esempio, la scelta di un dataset è cruciale per il successo del processo di addestramento del modello. Scegliere un dataset con un numero insufficiente di record può portare a un problema di overfitting: il modello risponde perfettamente ai contesti già visti durante l'addestramento, ma è incapace di generalizzare e quindi non risponde accuratamente a input differenti.

Appena si è iniziato a "mettere le mani" sul progetto, ci si è ritrovati confusi e senza un punto da cui partire. Di conseguenza, si è scelto di suddividere il *task* in *sotto-task* più piccoli, cosi da poterne affrontare uno alla volta in maniera più semplice. Successivamente, questi *sotto-task* sono stati ordinati per priorità e anche secondo un ordine che sembrava il più corretto. Sono stati evidenziati i seguenti passaggi principali:

1. Sviluppo dell'interfaccia grafica nell'applicazione mobile per l'inserimento delle recensioni degli utenti.
2. Implementazione del metodo per il salvataggio delle recensioni nel database.
3. Ricerca di un modello di Machine Learning per l'elaborazione del linguaggio naturale (NLP).
4. Ricerca di un dataset di recensioni appropriato.
5. Addestramento del modello sul dataset selezionato e successivo salvataggio del modello addestrato.
6. Recupero delle recensioni dal database nel backend Python.
7. Implementazione dei metodi per la classificazione delle recensioni e di altre funzionalità importanti, come la ricerca delle parole chiave.

8. Sviluppo della dashboard web per la visualizzazione dei risultati dell'elaborazione delle recensioni.

9. Implementazione delle richieste HTTP per il recupero dei dati dal backend e la successiva visualizzazione nella dashboard.

Adottando questo approccio di suddivisione in *sotto-task*, è stato possibile affrontare e risolvere ogni parte separatamente, raggiungendo così il risultato finale prefissato. L'ostacolo principale che ha richiesto maggior tempo è stato il cosiddetto "periodo di adattamento". Questo coincide con la parte iniziale della *curva di apprendimento*, durante la quale è necessario un periodo più o meno lungo per cominciare a orientarsi e comprendere meglio una nuova disciplina.

CAPITOLO 5

Conclusione

5.1 Discussione

L'integrazione del sistema di Sentiment Analysis all'interno dell'applicazione NavigaUNIVPM ha significativamente arricchito le sue funzionalità esistenti, consentendo agli amministratori di valutare il feedback generale sull'applicazione. Questa capacità di rilevare errori o insoddisfazioni condivise dagli utenti promuove un miglioramento continuo del servizio.

Lo sviluppo del sistema ha portato con sé una serie di difficoltà, soprattutto nelle fasi iniziali. Prima di arrivare a un risultato finale soddisfacente, sono stati utilizzati differenti approcci. Ad esempio, il primo modello che è stato utilizzato per eseguire la Sentiment Analysis è stato BERT. Solo successivamente, a seguito di una ricerca più approfondita, si è optato per l'utilizzo di RoBERTa che, come descritto nei capitoli precedenti, si è rivelato maggiormente efficiente e affidabile.

Gestire l'intero ciclo di sviluppo del progetto - dall'analisi dei requisiti iniziali alla valutazione delle performance e all'implementazione effettiva - ha fornito preziose lezioni su come gestire efficacemente le risorse, pianificare il lavoro in modo efficiente e affrontare sfide tecniche complesse. Questa esperienza non solo ha rappresentato un'opportunità di crescita personale, ma ha anche contribuito significativamente al percorso formativo nel campo dell'intelligenza artificiale e del machine learning.

L'esperienza acquisita attraverso questo progetto ha sottolineato l'importanza della flessibilità e della continua valutazione dei metodi utilizzati, evidenziando come approcci differenti possano condurre a miglioramenti significativi nei risultati. In conclusione, il lavoro svolto non solo ha portato a un'applicazione più completa e funzionale, ma ha anche fornito importanti spunti per ulteriori sviluppi e ottimizzazioni future.

5.2 Sviluppi futuri

Allo stato attuale, l'applicazione NavigaUNIVPM dispone delle principali funzionalità che permettono un utilizzo completo e soddisfacente da parte degli utenti. Tuttavia, per il futuro, si potrebbero sviluppare ulteriori funzionalità utili a migliorare ancora di più l'esperienza d'uso. In particolare, alcuni possibili sviluppi potrebbero includere:

- **Personalizzazione della dashboard**: Implementare una maggiore personalizzazione della dashboard per consentire agli amministratori di eseguire analisi dettagliate, come

il confronto tra periodi temporali e l'identificazione di trend. Questo permetterebbe una gestione più precisa e informata dei dati raccolti.

- **Intelligenza artificiale per il rilevamento dei bug**: Introdurre un modello di intelligenza artificiale in grado di rilevare automaticamente i bug menzionati nelle recensioni, oltre alla classificazione del sentiment e all'estrazione delle parole chiave. Ciò faciliterebbe la comprensione dei feedback degli utenti e l'identificazione rapida delle aree da migliorare.
- **Sistema di autenticazione**: Implementare un sistema di autenticazione per l'accesso alla dashboard, garantendo ai soli amministratori autorizzati la visualizzazione delle informazioni sensibili in modo sicuro. Questo aggiungerebbe un livello di sicurezza essenziale per la protezione dei dati.

Queste proposte mirano a migliorare ulteriormente la funzionalità e l'utilità di NavigaUNIVPM, rappresentando un significativo passo avanti nella gestione delle operazioni, in particolare nell'analisi delle recensioni. Con questi sviluppi, l'applicazione potrebbe offrire un supporto ancora più robusto e innovativo agli amministratori e agli utenti.

Bibliografia

KOKAB, S. T., ASGHAR, S. e NAZ, S. (2022), «Transformer-based deep learning models for the sentiment analysis of social media data», *Array*, vol. 14, p. 100 157. (Cited at page 8)

VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł. e POLOSUKHIN, I. (2017), «Attention is all you need», *Advances in neural information processing systems*, vol. 30. (Cited at pages 4 e 6)

XIAO, T. e ZHU, J. (2023), «Introduction to Transformers: an NLP Perspective», *arXiv preprint arXiv:2311.17633*. (Cited at page 8)

Ringraziamenti

Un ringraziamento speciale va al mio relatore, Dott. Enrico Corradini, per la disponibilità e la professionalità mostrate durante il lavoro di tirocinio e durante la stesura di questa tesi.

Ringrazio i miei genitori, Sabrina e Federico, i miei fratelli, Giacomo e Gioele, e mia sorella Mariastella, per il costante supporto e per la profonda stima che nutrono nei miei confronti. Vi ringrazio per credere sempre in me e nelle mie scelte di vita, siete il motore che mi da la forza per superare le sfide della vita.

Ringrazio la mia fidanzata Matunikka, per avermi aiutato a superare molti ostacoli durante questo percorso di studi e per essere stata luce nei momenti più bui. Ti ringrazio per aver sempre creduto in me e per avermi riportato con i piedi per terra, evitandomi la fine di Icaro, bruciato dal Sole per la sua superbia. Ti ringrazio per i tanti consigli che mi hai dato e continui a darmi, anche se spesso faccio fatica a metterli in pratica a causa del mio "bel caratterino".

Ringrazio tutti i miei amici dello studentato di Brecce Bianche, per avermi donato momenti indimenticabili e per avermi strappato un sorriso nei momenti in cui ne avevo più bisogno. Vi ringrazio per avermi distratto dallo studio disperato, permettendomi di riposare il cervello e di ricaricarmi. In special modo, ringrazio i miei amici Fatou e Aleandro per essere stati sempre disponibili, gentili e premurosi con me e per essere stati una spalla fedele nei momenti belli così come in quelli brutti.

Infine, voglio ringraziare me stesso per non aver mai mollato e per essere riuscito a raggiungere questo obiettivo nonostante i mille impegni di cui mi sono fatto carico in questi tre anni. Spero che questo percorso mi sia di esperienza e che mi aiuti ad affrontare gli ulteriori ostacoli e sfide che la vita mi metterà di fronte, consapevole di non camminare solo.