



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

TRACE: Tool for community Repositories Analysis and Cultural dispersion Examination

RELATORE

Prof. Palomba Fabio

Dott. Lambiase Stefano

Università degli Studi di Salerno

CANDIDATO

Vitale Ciro

Matricola: 0512110719

Questa tesi è stata realizzata nel



A me, la mia famiglia, e a tutti coloro che mi hanno voluto e mi vogliono bene.

Abstract

In un'epoca caratterizzata dalla *digitalizzazione* e dalla *globalizzazione*, il software si è affermato come elemento centrale della nostra quotidianità. Tale situazione ha messo in evidenza l'intrinseca *natura sociale dello Sviluppo Software*, con un numero sempre crescente di team distribuiti ed eterogenei. Gestire tale diversità risulta cruciale, poiché può implicare il successo o il fallimento di un progetto. Di fronte a sfide come la mancanza di soluzioni automatizzate, le limitate informazioni messe a disposizione dalle piattaforme e la complessità intrinseca della *Cultura*, è stato progettato **TRACE**. Questo tool, sviluppato per l'analisi automatizzata di *community open-source*, mira a quantificare la *Dispersione Culturale*, sfruttando *formule matematiche, tecniche di Intelligenza Artificiale, chiamate API a servizi online e cinque moduli predittivi*. Infine, in applicazioni reali, TRACE ha dimostrato efficacia e precisione nell'identificare e quantificare la Dispersione Culturale, colmando le problematiche esistenti e rispondendo alle sfide attuali.

Indice

| | |
|--------------------------------------------------------------------------|-----------|
| Elenco delle Figure | iv |
| Elenco delle Tabelle | v |
| 1 Introduzione | 1 |
| 2 Stato dell'Arte | 3 |
| 2.1 Cultura nello Sviluppo Software | 3 |
| 2.1.1 Definizione e descrizione di cosa si intende per Cultura | 4 |
| 2.1.2 Definizione e descrizione di Dispersione Culturale | 4 |
| 2.1.3 Framework e Modelli per rappresentare la Cultura | 5 |
| 2.2 Aspetti Sociali nello Sviluppo Software | 7 |
| 2.3 Machine Learning | 7 |
| 2.3.1 Apprendimento Supervisionato | 8 |
| 2.3.2 Apprendimento Non Supervisionato | 11 |
| 2.3.3 Apprendimento Semi-Supervisionato | 11 |
| 2.3.4 Apprendimento Per Rinforzo | 11 |
| 2.4 Natural Language Processing | 11 |
| 2.4.1 NLU | 12 |
| 2.4.2 NLG | 13 |
| 2.4.3 Tecniche e Modelli Tradizionali di NLP | 13 |

| | | |
|----------|--------------------------------------------------------------------------------------------|-----------|
| 2.4.4 | Deep Learning e NLP | 14 |
| 3 | TRACE: Tool for community Repositories Analysis and Cultural dispersion Examination | 16 |
| 3.1 | Obiettivo e Requisiti del Tool | 17 |
| 3.1.1 | Requisiti Funzionali | 17 |
| 3.1.2 | Requisiti Non Funzionali | 17 |
| 3.2 | Architettura del Tool | 18 |
| 3.2.1 | React | 18 |
| 3.2.2 | Flask | 20 |
| 3.2.3 | GitHub API | 21 |
| 3.2.4 | OpenAI API | 22 |
| 3.2.5 | Google Cloud API | 22 |
| 3.2.6 | Modulo NameToCountry | 23 |
| 3.2.7 | Modulo UsernameToCountry | 26 |
| 3.2.8 | Modulo CVToCountry | 26 |
| 3.2.9 | Modulo CommitsToCountry | 29 |
| 3.2.10 | Modulo LocationToCountry | 29 |
| 3.3 | Limiti del Tool | 30 |
| 3.4 | Interfaccia Grafica | 31 |
| 3.5 | Manuale di Installazione | 36 |
| 3.5.1 | Configurazione Server | 36 |
| 3.5.2 | Configurazione Client | 37 |
| 3.6 | Manuale d'Uso | 38 |
| 4 | Caso d'Uso del Tool | 39 |
| 4.1 | Dataset Analizzato | 39 |
| 4.1.1 | Hikki00/Raptor-AI | 40 |
| 4.1.2 | cheshire-cat-ai/core | 40 |
| 4.1.3 | carbon-language/carbon-lang | 40 |
| 4.2 | Esecuzione del tool | 41 |
| 4.2.1 | Hikki00/Raptor-AI | 42 |
| 4.2.2 | cheshire-cat-ai/core | 42 |

| | |
|---------------------------------------------|-----------|
| 4.2.3 carbon-language/carbon-lang | 42 |
| 5 Conclusioni | 43 |
| 5.1 Conclusioni | 43 |
| 5.2 Sviluppi Futuri | 43 |
| Bibliografia | 45 |

Elenco delle figure

| | | |
|-----|-------------------------------------------------------------------|----|
| 2.1 | Dataset Etichettato (e.g., spam classification) [1] | 9 |
| 2.2 | Struttura NLP [2] | 12 |
| 3.1 | Architettura del Tool | 19 |
| 3.2 | Header | 31 |
| 3.3 | Campo di Input | 31 |
| 3.4 | Loader e Disabilitazione Submit | 32 |
| 3.5 | Compenente di Errore | 32 |
| 3.6 | Notifica di Rumore nel Calcolo | 33 |
| 3.7 | Compenente Developers List | 33 |
| 3.8 | Compenente Repository Info | 34 |
| 3.9 | Compenente User Info - Dati sensibili oscurati per <i>Privacy</i> | 35 |

Elenco delle tabelle

| | |
|---------------------------------------------------------|----|
| 2.1 Matrice di Confusione | 10 |
| 4.1 Repository Analizzati con TRACE | 39 |
| 4.2 Risultati Repository Analizzati con TRACE | 41 |

CAPITOLO 1

Introduzione

Contesto In un'epoca caratterizzata dalla *digitalizzazione* e dalla *globalizzazione*, il software si è consolidato come uno strumento quotidiano nella vita di ciascuno di noi. Questo ha portato a evidenziare l'*intersezione tra Software Engineering e Cultura* generando una serie di sfide e opportunità. La comunità accademica e scientifica ha condotto numerosi studi e ricerche sull'importanza di considerare la Cultura nella gestione di comunità di Software Engineering e nella progettazione dei relativi progetti [3, 4, 5]. Poiché riconoscere e gestire le differenze culturali può determinare il successo o il fallimento di un progetto, tali differenze devono essere prese in considerazione e sfruttate per creare soluzioni inclusive ed efficaci.

Problema e Limitazione Esistono limitate soluzioni automatizzate che permettono di analizzare e quantificare efficacemente la *Dispersione Culturale* all'interno delle comunità open-source. Uno dei primi ostacoli è che la maggior parte degli approcci esistenti si basano su sondaggi, che possono non essere particolarmente rappresentativi. Un altro problema significativo è la scarsità di informazioni disponibili, molti strumenti tentano di estrapolare il massimo dalle informazioni spesso scarse a disposizione, come quelle offerte da piattaforme come GitHub. Infine, complica ulteriormente la situazione la natura stessa della Cultura, essendo un costrutto

complesso, multifattoriale e in evoluzione costante, è intrinsecamente difficile da analizzare e identificare con precisione.

Obiettivo del Lavoro L’obiettivo principale di questa tesi è la progettazione di uno strumento software, **TRACE**: *Tool for community Repositories Analysis and Cultural dispersion Examination*. Software capace di analizzare le community open-source, in particolare community GitHub, allo scopo di quantificarne la Dispersione Culturale, sfruttando *formule matematiche*, *tecniche di Intelligenza Artificiale*, *chiamate API* a servizi online e *cinque moduli predittivi*. Inoltre, il tool presenta un’*interfaccia user-friendly*, mostrando i risultati sia in forma numerica che sottoforma di grafici. Infine, una *valutazione approfondita* è stata condotta su un dataset significativo, validando l’efficacia del software nell’identificazione delle Culture e nella quantificazione della Dispersione Culturale in comunità variegate, sia in termini di contesto che di dimensioni. TRACE è stato progettato con un *duplice obiettivo*. Da un lato, intende assistere i *ricercatori* che lavorano nel contesto della Cultura e del Software Engineering, offrendo loro un tool automatizzato per l’analisi rapida e accurata di community open-source. Dall’altro lato, fornendo una risorsa pratica ai *team manager*, permettendo loro di prendere decisioni informate al fine di mitigare i problemi, adottando approcci efficaci e inclusivi per ogni situazione specifica.

Struttura della Tesi La presente tesi è organizzata come segue, inizia con una *Introduzione*, offrendo una panoramica generale del contesto, delle limitazioni attuali e dell’obiettivo del tool. Segue il capitolo *Stato dell’Arte*, dove viene fornito un approfondimento dei vari aspetti, dando una visione generale dello stato attuale della ricerca nei campi legati allo Sviluppo Software, la Cultura e le varie tecnologie utilizzate per lo sviluppo del tool. Successivamente, il capitolo *TRACE* presenta il software e il lavoro svolto, descrivendone gli obiettivi e l’architettura, fornendo, inoltre, manuali dettagliati utili per l’installazione e l’utilizzo. A seguire, il capitolo *Caso d’Uso del Tool*, dove sono presentati esempi pratici effettuati, mostrando e analizzando i risultati ottenuti dalle esecuzioni effettuate dal tool, validandone l’accuratezza. Infine, il capitolo *Conclusioni*, riflette sui risultati ottenuti e propone sviluppi futuri per tale tool.

CAPITOLO 2

Stato dell'Arte

2.1 Cultura nello Sviluppo Software

L'attività di **Software Engineering** (SE) trascende la semplice programmazione e l'implementazione di codice. Nella sua essenza, è una pratica sociale e collaborativa. Attualmente, lo Sviluppo Software è un'attività diffusa globalmente, grazie all'evoluzione della tecnologia è stato possibile superare le barriere geografiche e permettere a membri dello stesso team di essere locati in paesi diversi. La globalizzazione, ha reso possibile la formazione di team eterogenei, quindi, composti da investitori, managers, developers e altre figure professionali aventi Cultura diversa, il che può portare sia benefici che svantaggi, poiché in questa attività è di fondamentale importanza la collaborazione: comunicazione, comprensione e interazione tra i vari attori [3].

La *Dispersione Culturale* può notevolmente influenzare l'attività di Software Engineering. Pertanto, la costruzione di team per questa attività non è una pratica semplice. I manager devono essere in grado di formare team di sviluppo eterogenei, massimizzando i benefici previsti e, al contempo, prevedendo i potenziali problemi che possano presentarsi, causati da tali diversità, per poterli gestire al meglio [6].

2.1.1 Definizione e descrizione di cosa si intende per Cultura

Nel corso degli anni sono state fornite varie definizioni di **Cultura**, tra cui: *Hofstede* la definisce come "la programmazione collettiva della mente che distingue i membri di un gruppo o di una categoria di persone dagli altri" [7], invece, *Kreitner et al.* la definiscono come "assunzioni socialmente derivate e date per scontate su come agire e pensare" [3, 8]. Per quanto la definizione di Cultura può variare in base all'autore, in generale, può essere definita come l'insieme delle abitudini, conoscenze, valori, norme, tradizioni, lingue e qualsiasi altra capacità e abitudine acquisita e trasmessa da un individuo in quanto membro di una società. Tuttavia, è importante notare che la Cultura non è statica, bensì evolve e si adatta in risposta ai cambiamenti, questa dinamicità è ciò che consente alle società di evolvere e crescere [3].

Nel dominio dello Sviluppo Software, la Cultura può avere un impatto notevole sull'efficienza e sulla produttività di un team, ma anche sulle modalità di collaborazione tra i suoi membri. Pertanto, *comprendere la Cultura* è fondamentale per il successo di qualsiasi progetto di SE [4].

2.1.2 Definizione e descrizione di Dispersione Culturale

La **Dispersione Culturale** è la misura delle differenze culturali all'interno di un team [4]. Per quantificarla, è stato utilizzato l'*Indice di Shannon* [9], una misura impiegata per valutare la diversità di specie facendo riferimento ad abbondanza e distribuzione di essa, in questo caso adottata per analizzare la distribuzione delle Culture in una comunità (vedi Equazione 2.1.1).

$$H' = - \sum_{i=1}^s p_i \ln(p_i) \quad (2.1.1)$$

Nel quale:

- H' : indice di Shannon.
- p_i : porzione della i-esima Cultura.
- s : numero totale di Culture.

L'indice raggiunge il suo picco, $H'_{max} = \ln(s)$, quando tutte le Culture sono ugualmente rappresentate e scende progressivamente verso 0 all'aumentare della predominanza di una singola Cultura nel campione analizzato. Alcuni studi riportano che diversità e rendimento sono positivamente correlati [10], invece, altre ricerche affermano che sono negativamente correlati [6, 11]. Taluni studi si sono incentrati sui *Community Smells*, Caballero-Espinosa et al. li definiscono come le connessioni tra le decisioni socio-tecniche e le conseguenze che queste hanno sugli individui all'interno di una comunità [5], essi tendono a verificarsi in diverse circostanze: mancanza di comunicazione, mancanza di collaborazione, cambiamenti nel team, conflitti interpersonali, mancanza o cambiamenti della leadership, oppure il sovraccarico di lavoro. Un'alta *Dispersione Culturale* all'interno di un team deve essere gestita poiché può portare a Community Smells.

Lambiase et al. [4] hanno mostrato, mediante l'utilizzo di un modello statistico, che la Dispersione Culturale, in progetti di Software Engineering, influenza la produttività *sia positivamente che negativamente*.

2.1.3 Framework e Modelli per rappresentare la Cultura

La Cultura rappresenta un tema complesso da strutturare con precisione, per questo sono stati definiti Modelli e Framework per analizzarla e rappresentarla in organizzazioni e nazioni. Tali framework presentano le loro limitazioni ma, nonostante ciò, possono fornire una base utile per analizzare e comprendere la Cultura in una comunità o, più specificatamente, in un team di Software Engineering. Tali modelli possono essere di ausilio per identificare i punti di forza e le aree di miglioramento per quanto riguarda la Cultura in un team di Sviluppo Software. Alcuni tra i modelli più conosciuti ed utilizzati sono: *Modello di Hofstede* e *Modello di Trompenaars*.

Modello di Hofstede

Il **Modello di Hofstede** [7] è basato su 6 dimensioni:

- *Distanza dal Potere*: quanto una società tollera le differenze di potere.

- *Individualismo vs Collettivismo*: se un'organizzazione è maggiormente individuista o collettivista.
- *Mascolinità vs. Femminilità*: la mascolinità sottolinea ruoli di genere definiti, invece, la femminilità evidenzia la cooperazione.
- *Rifiuto dell'Incertezza*: quanto una società accetta o declina l'incertezza.
- *Orientamento a Lungo o Breve Termine*: se una comunità è focalizzata maggiormente sul presente o sul futuro.
- *Indulgenza vs Restrizione*: se una società predilige il naturale bisogno di soddisfare i desideri oppure impone limiti ad essi.

I benefici di questo modello sono evidenti: è basato su un ampio campione di dati, è facilmente comprensibile ed è uno dei framework più utilizzati. Di contro, le limitazioni, sono: i dati utilizzati risultano datati e riferiti a una realtà aziendale ed è eccessivamente semplificativo, non tiene conto di differenze regionali o individuali. Nonostante le limitazioni che presenta tale modello, esso risulta attualmente il miglior modello per rappresentare la Cultura [12].

Modello di Trompenaars

Il **Modello di Trompenaars** [13] è basato su 7 dimensioni:

- *Universalismo vs Particularismo*: come una società applica le regole, se valgono ugualmente per tutti oppure se vengono adattate in base alle circostanze.
- *Individualismo vs Comunitarismo*: se una comunità è maggiormente individualista oppure predilige il gruppo.
- *Neutralità vs Affettività*: quanto una società esprime le emozioni apertamente o meno.
- *Specificità vs Diffusività*: grado che esprime quanto i membri di una comunità separano la vita personale da quella professionale.

- *Realizzazione vs Attribuzione*: come i componenti della società ottengono lo status, in base alle realizzazioni personali oppure assegnati in base a titolo, età o background.
- *Sequenziale vs Asincrono*: se una società è più propensa alla pianificazione e alla puntualità oppure se è più flessibile nella gestione del tempo.
- *Controllo Interno vs Controllo Esterno*: se il controllo è autonomo oppure proviene dall'esterno.

Per quanto riguarda questo framework i benefici sono molteplici, tra cui: offre un'ampia copertura di aspetti culturali, rendendolo uno strumento efficace per analizzare la Cultura in profondità. Al contempo le limitazioni sono svariate, tra cui: può portare a generalizzazioni eccessive e l'utilizzo può risultare complesso rispetto ad altri modelli.

2.2 Aspetti Sociali nello Sviluppo Software

Gli **aspetti sociali** rivestono un ruolo cruciale nel Software Engineering. Essi includono la gestione delle risorse umane, la gestione dei conflitti, la leadership, la collaborazione e la comunicazione. Nel corso degli anni, sono evolute anche le metodologie di sviluppo, passando da modelli più "isolati" a metodi fortemente collaborativi. Inoltre, si è passati da team di sviluppo formati da poche persone a grandi team distribuiti globalmente.

2.3 Machine Learning

Il **Machine Learning** (ML) è un sottoinsieme dell'*Intelligenza Artificiale* (AI). Il suo scopo è addestrare i computer a pensare e imparare dai dati, migliorando con l'esperienza, anziché fornire alla macchina un elenco dettagliato di istruzioni su come risolvere un problema. Nel Machine Learning gli algoritmi vengono addestrati a far emergere pattern da grandi dataset e a formulare le migliori decisioni e previsioni sulla base di tali analisi. Le applicazioni di ML migliorano con l'esperienza e diven-

tano più precise proporzionalmente alla crescita del volume di dati a cui possono accedere [14].

L'algoritmo di apprendimento "apprende" cercando i migliori parametri per il modello di apprendimento, il modello "apprende" nel senso che le sue previsioni si adattano e migliorano a seguito dell'addestramento con l'algoritmo. La premessa chiave del Machine Learning è l'esistenza di una correlazione matematica tra un insieme di dati di ingresso e di uscita. Il modello di Machine Learning non conosce a priori tale corrispondenza, bensì è in grado di inferirla se ha a disposizione un ampio numero di dati, ogni algoritmo di ML è fondato su una funzione matematica modificabile. Una *pipeline* di ML è una sequenza di operazioni sul dataset che va dal suo stato iniziale al modello [15].

Il Machine Learning è utilizzato in molteplici campi, da quello medico a quello finanziario, passando per quello manifatturiero e molti altri ancora. Esistono quattro tipi principali di apprendimento: *supervisionato*, *non supervisionato*, *semi-supervisionato* e *per rinforzo*.

2.3.1 Apprendimento Supervisionato

Nell'**apprendimento supervisionato**, l'algoritmo viene addestrato utilizzando dati etichettati (vedi Figura 2.1), le etichette rappresentano le risposte attese. Gli algoritmi più importanti di apprendimento supervisionato sono: *k-Nearest Neighbors* e *Linear Regression* [1]. Quindi, per ogni osservazione, oltre ai dati di input, ovvero le variabili indipendenti, è noto anche il valore della variabile dipendente.

La decisione di impiegare l'apprendimento supervisionato per prevedere la Cultura di uno sviluppatore è stata influenzata dalla disponibilità online di dataset etichettati, che associano nomi propri di persona al paese di provenienza di quest'ultime.

Dataset

Individuato il *dataset etichettato*, bisogna procedere alla pulizia dei dati, eliminare dati mancanti, duplicati e correggere errori, successivamente bisogna effettuare il preprocessing dei dati, ovvero normalizzare e standardizzare i dati ed effettuare

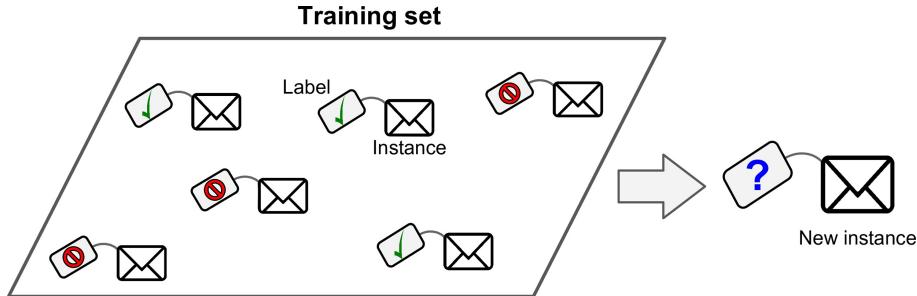


Figura 2.1: Dataset Etichettato (e.g., spam classification) [1]

la codifica delle variabili categoriali, per poi scegliere le features, per migliorare la capacità del modello di effettuare previsione accurate.

Il dataset iniziale viene diviso in *training set* e *test set*. Il *training set* utilizzato per addestrare il modello, invece, il *test set* utilizzato per valutare le prestazioni del modello dopo la fase di training [15].

Training

Durante la fase di *training*, l'algoritmo di apprendimento tenta di trovare una funzione (o un modello) che mappi in modo efficace gli input alle etichette corrispondenti. Questa funzione viene inizialmente impostata con dei parametri casuali. Il processo di addestramento consiste nell'adattare iterativamente i parametri di questa funzione per minimizzare una funzione di perdita, questo è un processo di ottimizzazione [15].

Errore

L'*errore* di un modello di ML è la differenza tra il valore stimato delle variabili da predire e il suo valore reale, può essere irriducibile oppure riducibile, il secondo è quello che si vuole e si può minimizzare. L'errore riducibile $\epsilon = \text{bias} + \text{varianza}$, è la somma di bias e varianza che sono inversamente correlate, il che rende difficile minimizzarle entrambe simultaneamente. Il bias indica l'errore di previsione dovuto alle semplificazioni del modello. La varianza rappresenta l'errore di previsione dovuto alla sensibilità del modello ai dati di addestramento. Alto bias implica *underfitting*, alta varianza implica *overfitting* [15].

Misure di Performance

La *Matrice di Confusione* (vedi Tabella 2.1) è una tabella spesso utilizzata su un modello di classificazione. Questa matrice presenta le seguenti informazioni:

- *True Positives* (TP): i casi in cui il modello ha predetto correttamente la classe positiva.
- *True Negatives* (TN): i casi in cui il modello ha predetto correttamente la classe negativa.
- *False Positives* (FP): i casi in cui il modello ha predetto erroneamente che la classe è positiva, mentre in realtà è negativa.
- *False Negatives* (FN): i casi in cui il modello ha predetto erroneamente che la classe è negativa, mentre in realtà è positiva.

| Classe Reale | Classe Predetta | |
|--------------|-----------------|----------|
| | Positiva | Negativa |
| Positiva | TP | FN |
| Negativa | FP | TN |

Tabella 2.1: Matrice di Confusione

Alcune misure di performance comuni nell'apprendimento supervisionato sono:

- *Precision* [1]: indica quanti errori ci saranno nella lista delle predizioni effettuate dal classificatore (vedi Equazione 2.3.1).

$$Precision = \frac{TP}{TP + FP} \quad (2.3.1)$$

- *Accuracy* [15]: indica il numero totale di predizione corrette sul totale delle istanze (vedi Equazione 2.3.2).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.3.2)$$

2.3.2 Apprendimento Non Supervisionato

Nell'**apprendimento non supervisionato**, i dati di addestramento sono privi di etichetta. Gli algoritmi più importanti di apprendimento non supervisionato sono: *Clustering, Anomaly Detection, Visualization and Dimensionality Reduction e Association Rule Learning* [1]. Il rischio in questa tecnica è di trovare falsi pattern, per via della mancanza della variabile dipendente associata, che rende anche difficile la valutazione dell'accuratezza del modello.

2.3.3 Apprendimento Semi-Supervisionato

Nell'**apprendimento semi-supervisionato**, l'algoritmo dovrà imparare facendo inferenza su dataset contenenti sia dati etichettati che non, solitamente sono in maggior numero quelli non etichettati, il che rappresenta un problema. Molti algoritmi di apprendimento semi-supervisionato sono combinazioni di algoritmi supervisionati e non supervisionati [1]. Il rischio persiste nel caso in cui i dati etichettati non siano rappresentativi dell'intero set, portando a un potenziale bias nel modello.

2.3.4 Apprendimento Per Rinforzo

Nell'**apprendimento per rinforzo** il sistema di apprendimento, chiamato agente, può osservare l'ambiente, selezionare azioni casuali ed eseguirle, ottenendo in cambio ricompense o penalità [1]. La ricompensa ha lo scopo di incoraggiare comportamenti corretti e, analogamente, la penalità quelli non corretti. Questa tecnica richiede molte iterazioni ed è sensibile alla funzione di ricompensa.

2.4 Natural Language Processing

Il **Natural Language Processing** (NLP) si riferisce alla branca dell'Intelligenza Artificiale che si occupa di dare ai computer la capacità di comprendere e interpretare le parole e il testo nello stesso modo in cui lo fanno gli esseri umani. Nel corso degli anni, l'NLP ha adottato vari approcci, attualmente combina la *linguistica computazionale*, ovvero la modellazione basata su regole del linguaggio umano, con

modelli statistici, di Machine Learning e di Deep Learning. Insieme, queste tecnologie consentono ai computer di elaborare il linguaggio umano, sotto forma di testo o dati vocali, e di "capire" il suo pieno significato, comprese le intenzioni e le emozioni di chi parla o scrive [16].

NLP è composto da *Natural Language Understanding* e *Natural Language Generation* (vedi Figura 2.2) che hanno il compito di capire e generare il testo. Khurana et al. [2] hanno mostrato che, nonostante i significativi progressi nell'ambito del Natural Language Processing negli ultimi anni, la complessità del linguaggio umano continua a rappresentare un ostacolo rilevante. L'ambiguità, ovvero le stesse parole e frasi che possono assumere significato diverso, rappresenta un task impegnativo per la macchina. Pertanto, l'NLP rimane un campo di studio altamente impegnativo e dinamico, caratterizzato da una ricerca costante e continua innovazione.

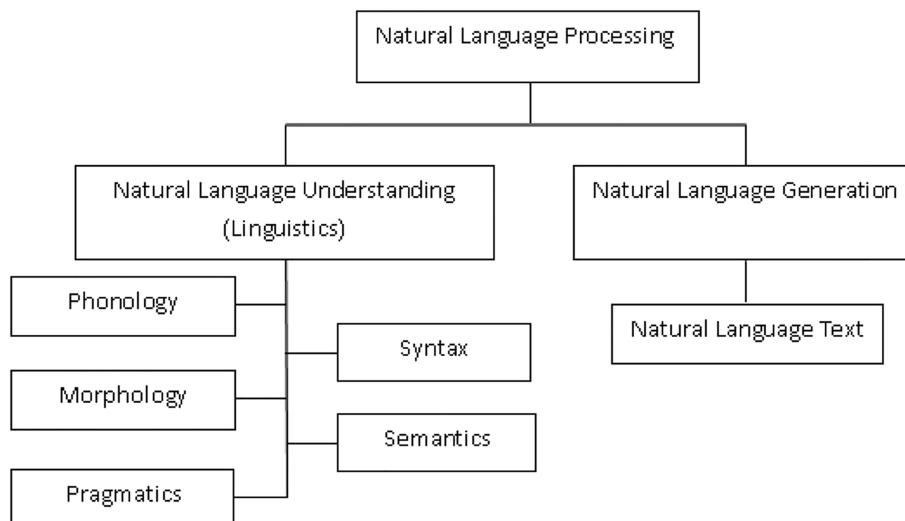


Figura 2.2: Struttura NLP [2]

2.4.1 NLU

Il **Natural Language Understanding** (NLU) è il processo che permette alla macchina di capire il linguaggio naturale, analizzandone sintassi e semantica estraendone concetti, entità, emozioni e keywords, come farebbe un essere umano. Inoltre, gli algoritmi di NLU sono in grado di pulire e trasformare il testo in strutture dati comprensibili alla macchina [2]. Il Natural Language Understanding è utilizzato in molteplici applicazioni, tra cui servizi di customer care e sentiment analysis.

2.4.2 NLG

Il **Natural Language Generation** (NLG) è il processo che si occupa della generazione del linguaggio naturale, in relazione ai dati in input, producendo frasi significative a partire da una struttura interna [2]. Esso è utilizzato in molteplici applicazioni, tra cui produzione di reportistica a partire da dati grezzi e riassunto di testi.

2.4.3 Tecniche e Modelli Tradizionali di NLP

Nel dominio del NLP, **tecniche e modelli tradizionali** combinavano regole linguistiche, conoscenza esperta della lingua e metodi statistici. Questi approcci hanno costituito per decenni le fondamenta del Natural Language Processing.

Rule-Based

Inizialmente, NLP era *rule-based*, basandosi su regole scritte a mano, utilizzando grammatiche formali. Sebbene questi metodi fossero precisi in specifici contesti, erano limitati dalla loro inflessibilità e dalla difficoltà nel cogliere le variegate sfumature delle lingue naturali.

Approcci Statistici

Gli *approcci statistici* utilizzano dati numerici e algoritmi probabilistici per analizzare e interpretare il linguaggio. Due dei più comuni approcci statistici sono: *Bag of Words* (BoW) [17] e *Term Frequency-Inverse Document Frequency* (TF-IDF) [18]. Nel modello di Machine Learning sviluppato per la previsione della Cultura di uno sviluppatore basandosi sul nome, è stato utilizzato il TF-IDF come `vectorizer` per le features di testo, convertendone il testo in forma numerica, rendendole utilizzabili dal modello.

Word Embedding

Il *word embedding* si riferisce alla rappresentazione di parole sotto forma di vettori continui ad alta dimensionalità, essi catturano la semantica e la relazione tra le parole, in base al contesto di riferimento. L'evoluzione verso queste tecniche e modelli ha

rivoluzionato l’NLP, consentendo il superamento di molte delle limitazioni che presentavano gli approcci statistici. Due tra le più comuni tecniche di word embedding sono: *Word2Vec* [19] e *Global Vectors for Word Representation* (*GloVe*) [20].

2.4.4 Deep Learning e NLP

L’introduzione del **Deep Learning nel NLP** ha portato a una trasformazione senza precedenti, ristrutturando in maniera significativa una vasta gamma di applicazioni, determinando significativi progressi in termini di capacità, precisione e applicabilità dei modelli.

Neural Network for NLP

Le *reti neurali* rappresentano un pilastro fondamentale per l’esponenziale crescita del NLP moderno, con la loro capacità di apprendere rappresentazioni complesse del linguaggio, comprendendo il significato intrinseco delle frasi e delle parole, in relazione ai relativi contesti. Le principali architetture sono: *Recurrent Neural Networks* (RNNs) [21], *Sequence to Sequence Model* (Seq2Seq) [22] e *Convolutional Neural Networks* (CNNs) [23].

Transformer

I modelli *Transformer*, introdotti da Vaswani et al. [24], hanno rappresentato un progresso notevole nel campo del NLP. Questi modelli offrono un meccanismo di *attenzione* per dare maggior importanza a determinate parole in input, a seconda del contesto. Tali modelli sono composti da:

- *Encoder*: riceve in input una sequenza di parole e produce in output una rappresentazione continua dell’input. Sebbene tenga conto delle dipendenze tra le parole, non ha consapevolezza della loro posizione intrinseca nella sequenza. Per superare questa limitazione, viene aggiunta una codifica posizionale.
- *Decoder*: utilizza sia l’output dell’encoder che una sequenza di input separata per generare l’output, durante questa fase, fa uso del meccanismo di attenzione.

Pre-Trained Language Models

Nell’ambito del NLP, i *Pre-Trained Language Models* hanno portato a una svolta epocale, raggiungendo prestazioni elevate in molteplici campi di applicazione. Essi sono addestrati su grandi quantità di testo, riuscendo a cogliere sfumature grammaticali e linguistiche dell’input. Possono essere facilmente adattati a nuovi campi di applicazione, con poco addestramento aggiuntivo [25]. I principali modelli sono:

- *Bidirectional Encoder Representations from Transformers* (BERT): questo modello analizza in maniera bidirezionale il testo, considerando sia il contesto precedente che successivo di ogni parola, consentendo di avere una comprensione più profonda del contesto [25].
- *Generative Pre-trained Transformer* (GPT): questo modello rappresenta uno dei pilastri nei Pre-Trained Language Models. GPT è un modello di decodifica, orientato alla previsione sequenziale di parole, rendendolo in grado di essere estremamente efficace nei compiti generativi. Quindi, gli aspetti distintivi di GPT includono la capacità generativa e la facilità di adattamento ai vari contesti. La famiglia di modelli GPT di OpenAI si è evoluta in pochissimo tempo, fino ad arrivare a GPT-4, che risulta attualmente il modello di linguaggio più potente sviluppato [25]. I modelli di OpenAI sono stati utilizzati nel tool sviluppato, per effettuare previsioni sulla Cultura degli sviluppatori (vedi Sezione 3.2.4).

CAPITOLO 3

TRACE: Tool for community Repositories Analysis and Cultural dispersion Examination

Il tool **TRACE** è stato ideato per *analizzare community open-source* nel campo del Software Engineering, con lo scopo principale di *calcolarne la Dispersione Culturale*. In maniera specifica, quest'analisi verrà effettuata sui repository *GitHub*. La sfida principale risiede nell'*identificazione della Cultura* di uno sviluppatore (vedi Sezione 2.1.1), a partire dalle limitate informazioni fornite da un account GitHub. Tale identificazione è fondamentale, poiché è utilizzata, successivamente, nel calcolo della Dispersione Culturale, mediante l'applicazione dell'Indice di Shannon (vedi Equazione 2.1.1) [9]. Si noti che ogni Cultura, nel tool, è rappresentata con il relativo codice ISO 3166-1 *alpha-2*.

TRACE è stato progettato per *due principali motivi*. Il primo è quello di offrire alla comunità accademica e scientifica un tool in grado di effettuare un'analisi rapida e accurata di una community open-source, quantificandone in particolare la Dispersione Culturale. La seconda ragione è quella di fornire ai contributors una visione chiara delle differenze culturali all'interno del repository, permettendo ai team manager di prendere decisioni informate, al fine di mitigare i problemi e di adottare approcci inclusivi. Queste necessità nascono dal crescente interesse verso le diversità

culturali nel mondo del Software Engineering, dettate dalla sua globalizzazione. Tali aspetti culturali stanno recentemente acquisendo sempre maggiore importanza nella comunità scientifica. Studi recenti sul tema evidenziano l'importanza di considerare la Cultura nella progettazione e gestione dei progetti di Sviluppo Software [3, 4, 5].

3.1 Obiettivo e Requisiti del Tool

Il tool TRACE è stato ideato e sviluppato con l'obiettivo di analizzare la Disersione Culturale in community open-source, nello specifico in repository GitHub.

3.1.1 Requisiti Funzionali

I **requisiti funzionali** emersi per TRACE sono:

- *Estazione dati da GitHub.*
- *Tentativo di predizione della Cultura per ogni contributor.*
- *Calcolo della Dispersione Culturale.*

3.1.2 Requisiti Non Funzionali

I **requisiti non funzionali** emersi per TRACE sono:

- *Usability:*
 - Interfaccia utente semplice e intuitiva.
 - Visualizzazione grafica delle informazioni.
- *Reliability:*
 - Gestione di errori e interruzioni.
- *Supportability:*
 - Facilità di manutenzione e aggiornamento.

3.2 Architettura del Tool

TRACE è stato progettato con un’architettura *client-server* sfruttando le potenzialità di due moderne tecnologie: *React* per il front-end e *Flask* per il back-end (vedi Figura 3.1). React funge da client e invia richieste a Flask, il quale funge da server e si occupa dell’elaborazione dei dati. Quest’ultimo sfrutta *formule matematiche, tecniche di Intelligenza Artificiale*, cinque *moduli predittivi* e *chiamate API* a servizi online per assicurare l’elaborazione di un risultato accurato (vedi Figura 3.1). Alcuni dei moduli predittivi, diventano operativi solo quando viene fornita la specifica chiave di configurazione relativa al servizio online API a cui sono collegati (vedi Sezione 3.5.1). Se vengono fornite tutte le chiavi di configurazione richieste, allora tutti i moduli possono essere attivati, consentendo al tool di operare al massimo delle sue potenzialità. In caso contrario, le sue potenzialità saranno limitate.

Durante la fase di progettazione e sviluppo, sono state seguite le regole di buona programmazione e codice pulito. L’utilizzo di strumenti ausiliari come il tool *Prettier*, ha garantito un’ulteriore formattazione chiara e pulita. Il codice è stato prodotto assicurando la modularità, dividendo in metodi e componenti specifici per ogni task, facilitando modifiche future ed eventuali manutenzioni del software. Ogni potenziale errore è stato gestito, assicurando un’esecuzione stabile del software. Inoltre, il codice è stato opportunamente commentato, al fine di facilitare il lavoro di futuri sviluppatori, rendendo loro più semplice la comprensione della logica e della struttura del codice. Infine, i log del server sono stati impostati in maniera tale da consentire il monitoraggio dell’esecuzione del tool mediante il *terminale*.

3.2.1 React

React è una libreria open-source di JavaScript per la creazione di interfacce web dinamiche e performanti [26]. Nel contesto di TRACE, il front-end presenta all’utente una pagina semplice e intuitiva, dando la possibilità di inserire il repository GitHub che si vuole analizzare, con formato *OWNER/NAME*. Successivamente, l’applicazione invia una richiesta al back-end, il quale si occupa di elaborare i dati. Quando quest’ultimo termina l’analisi, restituisce i risultati al front-end. L’interfaccia presenta i risultati dell’elaborazione all’utente, anche sotto forma di grafici. Inoltre, il

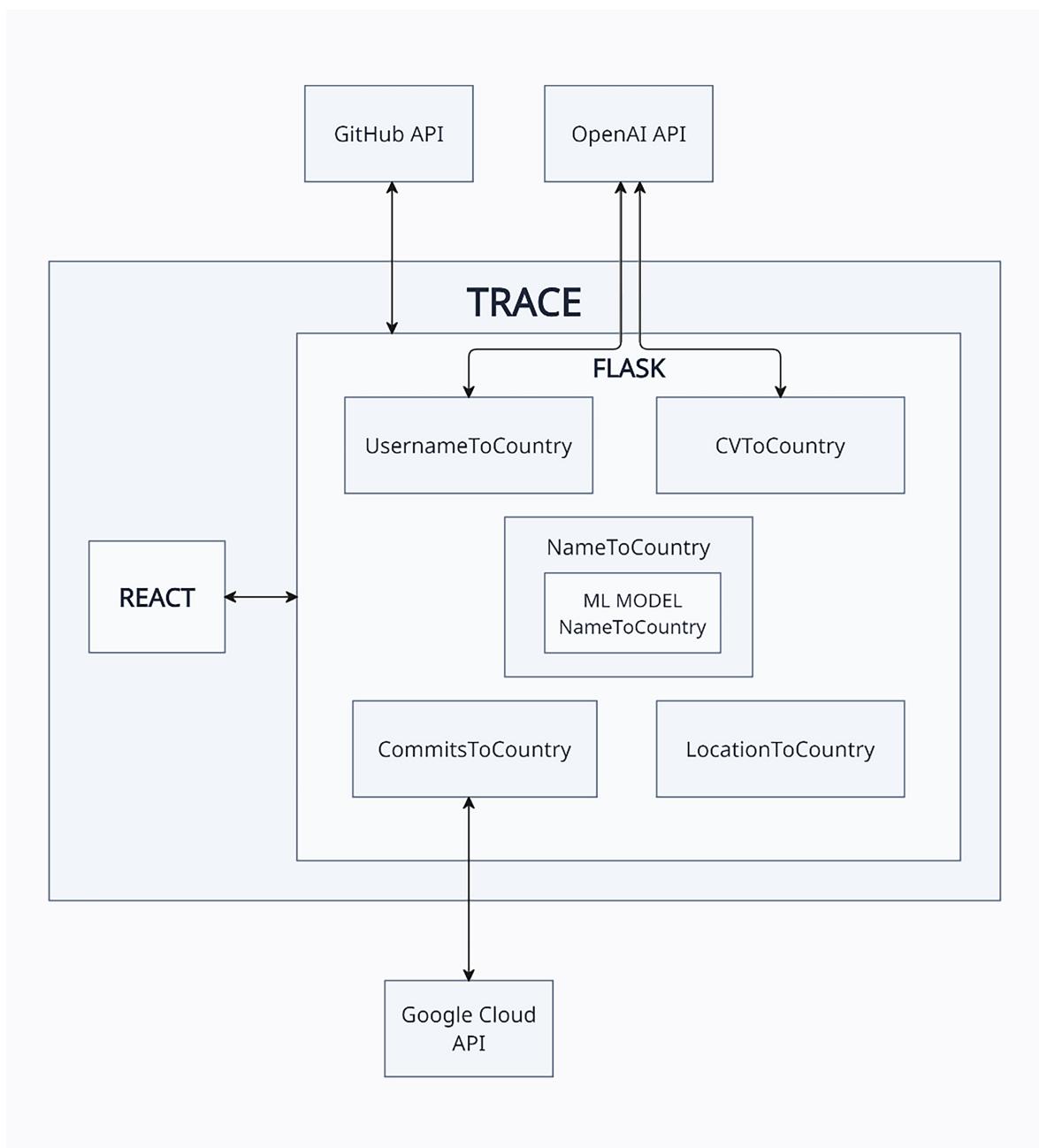


Figura 3.1: Architettura del Tool

front-end mostra un elenco dei contributors del repository, indicando per ciascuno di essi la Cultura stimata. Selezionando uno specifico sviluppatore, l’utente può comprendere attraverso quali dei *moduli predittivi* è stata determinata la Cultura associata. In caso di errori o problemi durante l’elaborazione da parte del back-end, un messaggio viene mostrato dall’applicazione, informando l’utente dell’errore e della natura di quest’ultimo.

Dal punto di vista tecnico, React è in esecuzione sulla porta 3000, invece, Flask sulla porta 5000. Per tale motivo, per garantire la comunicazione, mediante API, tra client e server, nel file `package.json` del progetto React è stato impostato l’indirizzo del server Flask, mediante la chiave `proxy`, per instradare le richieste al server.

3.2.2 Flask

Flask è un micro-framework scritto in Python, progettato per sviluppare back-end leggeri e performanti [27]. Nell’ambito di TRACE, Flask svolge un ruolo cruciale per l’elaborazione dei dati. Riceve richieste dal front-end ed effettua una serie di analisi sul repository in input. In particolar modo, effettua la predizione della Cultura di ogni sviluppatore, mediante cinque diversi moduli predittivi: *NameToCountry* (vedi Sezione 3.2.6), *UsernameToCountry* (vedi Sezione 3.2.7), *CVToCountry* (vedi Sezione 3.2.8), *CommitsToCountry* (vedi Sezione 3.2.9) e *LocationToCountry* (vedi Sezione 3.2.10). Dopodiché, applica una formula matematica pesata, allo scopo di stimare la Cultura dell’individuo in base ai risultati ottenuti dai vari moduli, la quale considera che ciascun modulo ha un peso diverso nella predizione, determinando la rilevanza di quest’ultimo nel calcolo finale. I pesi assegnati a ciascun modulo predittivo, dopo un’attenta analisi, sono i seguenti:

- *Modulo NameToCountry*: 0,25
- *Modulo UsernameToCountry*: 0,17
- *Modulo CVToCountry*: 0,21
- *Modulo CommitsToCountry*: 0,16
- *Modulo LocationToCountry*: 0,21

Infine, calcola la Dispersione Culturale della community (vedi Equazione 2.1.1) e invia i risultati al front-end.

Dal punto di vista tecnico, Flask è stato configurato per operare sulla porta 5000. Esso intercetta le richieste inviate dal front-end, processa le informazioni e, dopo l'elaborazione, risponde al client. Dal punto di vista della sicurezza, tutte le chiavi e i token menzionati nel corso di questo documento, sono memorizzati all'interno del file `.env`, garantendo la riservatezza e la protezione di tali dati sensibili.

3.2.3 GitHub API

Per condurre l'analisi del repository fornito in input, bisogna interfacciarsi con le **GitHub API**, permettendo al tool il recupero delle informazioni relative al suddetto repository e ai contributors che vi hanno partecipato. Le interazioni con le GitHub API sono implementate mediante chiamate HTTP, usufruendo della libreria `requests`. L'utilizzo di queste API è cruciale per il funzionamento dell'applicazione.

Limiti e Costi

GitHub pone limiti sull'utilizzo delle proprie API, questi limiti sono stabiliti per prevenire gli abusi. La piattaforma impone limiti specifici a seconda dell'utilizzo o meno di un token di autenticazione, reperibile gratuitamente accedendo all'area personale del proprio account GitHub. Nel caso in cui non venga utilizzato alcun token, il limite è di 60 richieste all'ora, invece, utilizzando il token personale di autenticazione, lo scenario cambia drasticamente, espandendo il limite a 5000 richieste all'ora. Il superamento di questi limiti induce a temporanee restrizioni nell'utilizzo delle API.

Gestione Errori

Durante l'interazione con GitHub mediante le API, sono stati implementati adeguati metodi per gestire errori e malfunzionamenti di diversa natura. Questi includono l'assenza o l'inaccessibilità al repository specifico, il superamento dei limiti di richiesta o eventuali malfunzionamenti durante l'interazione con le GitHub API.

3.2.4 OpenAI API

Alcuni dei moduli predittivi usufruiscono dell’interazione con **OpenAI** e, in particolare, con **ChatGPT**. Il modello specifico impiegato è `gpt-3.5-turbo`, l’interazione con quest’ultimo avviene mediante la libreria ufficiale di OpenAI, `openai`, mediante la funzione `ChatCompletion`. Durante questa interazione, vengono specificati parametri come il modello, il prompt e la temperature.

Limiti e Costi

L’utilizzo delle OpenAI API impone stringenti limiti in termini di numero di richieste, di lunghezza del prompt e delle risposte. L’utilizzo di ChatGPT attraverso l’API ha un costo associato, è possibile reperire il token registrandosi come developer sulla piattaforma di OpenAI, successivamente per ogni richiesta sarà addebitato il relativo costo. Un altro aspetto fondamentale da considerare è la natura *non deterministica* di ChatGPT, essendo una *AI generativa*, ciò implica che, dato lo stesso prompt in input, il modello potrebbe generare risposte leggermente diverse in occasioni diverse.

Gestione Errori

Nel processo di interazione con OpenAI tramite le API, è cruciale avere un robusto sistema di gestione degli errori. Le anomalie possono emergere da diversi scenari come il superamento dei limiti di richiesta, problemi di connessione oppure risposte inattese dall’API. Inoltre, per assicurare efficienza e stabilità, ogni richiesta è eseguita su un thread a sé stante, dotato di timeout, al fine di prevenire potenziali attese prolungate.

3.2.5 Google Cloud API

Google Cloud API, è stato integrato per permettere le interazioni, in particolar modo, con *Cloud Translation API*. Per usufruire e integrare questo servizio è necessario ottenere le chiavi di autenticazione accedendo alla *console* di sviluppatore di Google, da cui è possibile anche gestire e configurare le API che si vuole utilizzare.

Limiti e Costi

La fruizione di tale servizio comporta dei costi, si noti che Google offre un piano gratuito di 3 mesi per i nuovi sviluppatori che utilizzano Google Cloud API, successivamente sarà necessario fare riferimento ai costi previsti per le chiamate alle API utilizzate.

Gestione Errori

Sono gestiti gli eventuali errori che possono verificarsi durante le chiamate a Google Cloud API. In caso di malfunzionamenti, l'errore viene intercettato e notificato all'utente.

3.2.6 Modulo NameToCountry

Il modulo predittivo **NameToCountry** effettua la previsione della Cultura basandosi sul nome. Tale previsione è fatta mediante l'uso di tecniche di Intelligenza Artificiale, specificatamente mediante un *modello di Machine Learning* addestrato con l'algoritmo di *apprendimento supervisionato Random Forest* (vedi Sezione 2.3.1). Si noti che il modulo prende in considerazione solo le previsioni, effettuate dal *modello di Machine Learning*, con probabilità superiore al 50,00 %.

Modello di ML

Il **modello di ML** è stato addestrato sul dataset `name_country.csv`¹ composto da nomi di persone e i rispettivi paesi di provenienza. In *fase di preparazione*, il dataset è stato pulito rimuovendo le righe nulle e tutti i nomi sono stati convertiti in minuscolo per garantire uniformità. Inoltre, per bilanciare il dataset e prevenire l'*overfitting*, è stata applicata una tecnica di *undersampling*, basata sulla distribuzione media dei paesi presenti nel dataset. Poi, i nomi presenti nel dataset sono stati vettorizzati mediante il *TF-IDF Vectorizer*, essenziale per convertire i dati testuali in forma numerica, rendendoli utilizzabili per addestrare il modello. Successivamente, il dataset è stato diviso in *training set* e *test set*. La *fase di addestramento* è stata effettuata

¹<https://github.com/cirovitale/TRACE/tree/main/ml-training-scripts/nameToCountry>

mediante l'algoritmo classificatore *Random Forest*, con *numero di alberi fissato a 200 e cross-validation a 3 fold*.

Di seguito, è riportato il codice utilizzato per la preparazione del dataset, l'addestramento del modello e la valutazione delle prestazioni di quest'ultimo. Tutto ciò che concerne il modello di Machine Learning è stato sviluppato usufruendo delle seguenti librerie: `sklearn`, `numpy`, `pandas` e `joblib`.

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.model_selection import train_test_split, cross_val_score
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.metrics import accuracy_score, precision_score
7 from joblib import dump
8
9 # Undersample dataset
10 def undersampleDataset(dataset, column, targetCount):
11     return dataset.groupby(column).apply(lambda x: x.sample(min(len(x),
12                                                               → targetCount))).reset_index(drop=True)
13
14 # Load and clean dataset
15 def loadAndCleanDataset(filepath):
16     dataset = pd.read_csv(filepath)
17     datasetCleaned = dataset.dropna().copy()
18     datasetCleaned['Name'] = datasetCleaned['Name'].str.lower()
19     averageCount = int(datasetCleaned['Country'].value_counts().mean())
20     datasetBalanced = undersampleDataset(datasetCleaned, 'Country',
21                                           → averageCount)
22     return datasetBalanced
23
24 # Vectorize data
25 def vectorizeData(data, ngramRange=(1, 2)):
26     vectorizer = TfidfVectorizer(ngram_range=ngramRange)
27     X = vectorizer.fit_transform(data)
28     return X, vectorizer
29
30 # Train model
31 def trainRandomForest(X_train, y_train, nEstimators=200, cv=3):
32     model = RandomForestClassifier(n_estimators=nEstimators)
33     scores = cross_val_score(model, X_train, y_train, cv=cv)
34     model.fit(X_train, y_train)
35     return model, scores

```

```

36 # Evaluate model
37 def evaluateModel(model, X_test, y_test):
38     y_pred = model.predict(X_test)
39     accuracy = accuracy_score(y_test, y_pred)
40     precision = precision_score(y_test, y_pred, average='weighted')
41     return accuracy, precision
42
43 def main():
44     dataset = loadAndCleanDataset("name_country.csv")
45     X, vectorizer = vectorizeData(dataset['Name'])
46     y = dataset['Country']
47     # Split dataset in training and test set
48     X_train, X_test, y_train, y_test = train_test_split(X, y,
49             test_size=0.2)
50     model, scores = trainRandomForest(X_train, y_train)
51     accuracy, precision = evaluateModel(model, X_test, y_test)
52     print(f"Cross-Validation Scores: {scores}")
53     print(f"Mean Cross-Validation Scores: {np.mean(scores)}")
54     print(f"Accuracy: {accuracy}")
55     print(f"Precision: {precision}")
56     dump(model, 'random_forest_model.joblib')
57     dump(vectorizer, 'tfidf_vectorizer.joblib')
58
59 if __name__ == "__main__":
60     main()

```

Limiti e Accuratezza

Il modulo predittivo presenta limiti intrinseci, data la comune presenza di nomi in diverse Culture oppure nomi particolarmente rari e inusuali. Un ulteriore problema è che il dataset utilizzato, nonostante la grande dimensionalità, presenta i nomi solo per 63 Culture differenti, non coprendo la totalità delle Culture globali. La *fase di valutazione* effettuata sul classificatore mediante diverse metriche (vedi Sezione 2.3.1) ha evidenziato i seguenti *soddisfacenti risultati*:

- *Cross-Validation Scores:* [0,67 0,67 0,66]
- *Media Cross-Validation Scores:* 0,67
- *Precision:* 0,68
- *Accuracy:* 0,70

3.2.7 Modulo UsernameToCountry

Il modulo predittivo **UsernameToCountry** effettua il task di prevedere la Cultura di uno sviluppatore basandosi esclusivamente sul suo username. Questa analisi si avvale delle API di OpenAI e del modello gpt-3.5-turbo (vedi Sezione 3.2.4).

Limiti e Accuratezza

L’analisi della Cultura basandosi esclusivamente sull’username presenta limiti intrinseci. Nonostante ciò, ChatGPT è in grado di analizzare il nome utente per individuare nomi, abbreviazioni di nomi e parole al suo interno che indicano potenziali legami con la nazionalità o la Cultura dell’individuo. Tuttavia, nel caso in cui non venga riscontrato alcun legame significativo tra username fornito e potenziale nazionalità di appartenenza, l’analisi restituisce un risultato nullo.

Il modulo è stato sottoposto a test e ottimizzazioni raggiungendo *risultati soddisfacenti* in molte situazioni. Nella fase di test, sono stati presi in considerazione solo i risultati per cui il modulo ha predetto un valore non nullo, registrando un’accuratezza del 67,75 %. Questa percentuale mostra l’efficacia del modulo e la sua capacità di operare in diverse condizioni. Tuttavia, è importante notare, che l’accuratezza può variare a seconda della specificità del nome utente analizzato e dall’eventuale assenza di informazioni rilevanti in esso.

3.2.8 Modulo CVToCountry

Il modulo predittivo **CVToCountry** ha lo scopo di prevedere la Cultura di uno sviluppatore basandosi sul suo Curriculum Vitae (CV). Questo modulo sfrutta tecniche di web scraping per individuare ed analizzare il CV, cercando nel sito web, se specificato tra le informazioni di GitHub, del contributor in questione. Questa ricerca avviene tramite la libreria BeautifulSoup, cercando file in formato PDF. Identificati i documenti, mediante un’analisi su nome, numero di pagine e dimensione del documento, viene appurato se si tratta di un CV o meno. In caso di esito positivo del controllo, si procede con l’estrazione del contenuto del file, mediante la libreria PyPDF2. Successivamente, il contenuto del documento viene inoltrato a ChatGPT di OpenAI (vedi Sezione 3.2.4), per effettuare la predizione mediante il modello

gpt-3.5-turbo. Nel caso in cui vengono identificati più file che potrebbero essere CV, la previsione finale si basa sulla Cultura che risulta essere la più frequente rilevata in questi documenti.

Di seguito, è riportato un estratto di codice, del modulo *CVToCountry*, che mostra l'avvio di un thread dedicato, mediante la libreria `threading`, per interagire con ChatGPT di OpenAI, inviare il contenuto del PDF e attendere la risposta entro un tempo limite.

```

1 import openai
2 import threading
3 import json
4
5 def detectCVCountryOpenAI(textPdf, results):
6     try:
7         response = openai.ChatCompletion.create(
8             model="gpt-3.5-turbo",
9             messages=[
10                 {
11                     "role": "user",
12                     "content": f"Based on the information provided and
13                         doing your best, try to make a prediction. Given
14                         the following information from a developer's
15                         Curriculum Vitae, analyze Name, Addresses, Phone
16                         Numbers, Work Experience, Education, Language,
17                         Projects, and anything else that may be useful
18                         in trying to predict his or her nationality.
19                         Return the prediction in the given JSON format,
20                         using 'NULL' if you cannot make a prediction. Be
21                         sure to use ONLY single ('') or double (\")
22                         quotes in the JSON format. Do not use unescaped
23                         double quotes within JSON strings. ONLY and
24                         EXCLUSIVELY return a JSON with the following
25                         format: {json.dumps({ 'isoPredicted':
26                             '[PREDICTION ISO 3166-1 alpha-2 COUNTRY or
27                             NULL]', 'reasons': '[REASONS FOR THE CHOICE]',
28                             'completeAnswers': '[DETAILED ANSWER]' })} CV
29                         CONTENT: [[[ {textPdf} ]]]"
30                 }
31             ],
32             temperature=1
33         )
34         results['data'] = response
35     except Exception as e:
36         results['error'] = str(e)

```

```

20 def detectCountryFromCV(urlPdf):
21     results = {}
22     # Setup thread call and start it
23     timeout_seconds = 15
24     textPdf = getTextByPdf(urlPdf)
25
26     thread = threading.Thread(target=detectCVCountryOpenAI,
27                               args=(textPdf, results))
28     thread.start()
29     thread.join(timeout=timeout_seconds)
30
31     if thread.is_alive():
32         # Timeout handling
33         print("[OPENAI API] API call timed out")
34         return {
35             "error": "API call timed out",
36             "status": "408"
37         }
38
39     if 'error' in results:
40         print(f"[OPENAI API] Error: {results['error']}")  

41         return {
42             "error": results['error'],
43             "status": "403"
44         }
45
46     return results['data']['choices'][0]['message']['content']

```

Limiti e Accuratezza

Sebbene determinare la Cultura basandosi sulle sole informazioni del CV presenta limiti intrinseci, la varietà di informazioni presenti, in questa tipologia di documenti, come formazione, esperienze, lingue parlate e altre informazioni personali, possono fornire a ChatGPT utili indizi per effettuare la predizione della Cultura.

Il modulo è stato sottoposto a test e ottimizzazioni raggiungendo *ottimi risultati*. Nella fase di test, è stata registrata, per i Curriculum Vitae individuati nei siti web dei contributor, un'accuratezza pari al 85,97 %, considerando soltanto le predizioni non nulle. Questo evidenzia la capacità di ChatGPT nel comprendere ed analizzare la vasta gamma di informazioni presenti nel CV.

Gestione Errori

La gestione degli errori, in questo modulo, comprende la risoluzione di URL malformati, mediante la libreria `urllib`, e la gestione dei problemi nei tentativi di accesso ai siti web.

3.2.9 Modulo CommitsToCountry

Il modulo predittivo **CommitsToCountry** ha il compito di individuare la lingua utilizzata nel testo dei commits prodotti da uno specifico contributor. Tale modulo offre due metodi per effettuare la rilevazione della lingua dal testo, a seconda di se vengono fornite o meno le credenziali di Google Cloud API (vedi Sezione 3.2.5). Il fruitore dell'applicazione può decidere di fornire tali credenziali ed avere una predizione molto più precisa, effettuando richieste HTTP a *Google Cloud Translation API*, a pagamento, oppure, facendo affidamento alla libreria `langdetect`, soluzione pratica e gratuita che non ha bisogno di configurazione.

Limiti

Il limite principale emerso dopo un'attenta analisi, è che l'inglese, essendo una lingua internazionale, è spesso adottato nei progetti open-source da sviluppatori di diversa nazionalità. Pertanto, la rilevazione della lingua inglese potrebbe non fornire informazioni significative sulla Cultura del contributor in questione. Di conseguenza, è stata presa la decisione di ignorare il risultato di questo modulo se la lingua predetta risultasse esclusivamente l'inglese, negli n commits analizzati. In caso contrario, si tiene conto della lingua più frequente riscontrata tra i commits del developer in questione, supponendo che tale risultato rappresenti la lingua madre del contributor, dando un'informazione diretta sulla sua Cultura.

3.2.10 Modulo LocationToCountry

Il modulo predittivo **LocationToCountry** è progettato per identificare il codice ISO del paese, basandosi sulla posizione dichiarata dal contributor nel suo account GitHub. Il modulo utilizza la libreria `geopy` per effettuare tale task, analizzando

l’indirizzo fornito dal contributor e restituendo come risultato il *codice ISO 3166-1 alpha-2* relativo.

Limiti

Questa informazione è un indizio diretto sulla posizione geografica dello sviluppatore, ma potrebbe rivelarsi ambigua. Infatti, può indicare la provenienza geografica oppure la residenza attuale, che può essere diversa dalla nazionalità di appartenenza, pertanto è un’informazione che può indurre ad errore e deve essere valutata con cautela.

3.3 Limiti del Tool

TRACE presenta alcuni limiti intriseci legati principalmente alle informazioni disponibili negli account GitHub di ogni contributor preso in esame. Uno dei principali vincoli, quindi, è la *quantità e qualità* delle informazioni presenti in ogni account di cui si cerca di prevedere la Cultura. Infatti, GitHub richiede obbligatoriamente, in fase di registrazione, solo l’username, mentre tutte le altre informazioni sono facoltative. La mancanza di determinate informazioni limita notevolmente le capacità predittive del tool, in quanto riduce la possibilità di attivazione dei vari moduli predittivi.

Per le problematiche sopracitate e per i limiti di ogni modulo predittivo implementato, in alcuni casi, nonostante la combinazione dei vari moduli, il tool potrebbe non riuscire a prevedere con successo la Cultura di uno specifico contributor. In tali situazioni, questi sviluppatori vengono assegnati ad una Cultura nulla, identificata come “*N/A*”. Questi sviluppatori non sono inclusi nel calcolo finale della Dispersione Culturale. Inoltre, se essi rappresentano più del 30,00 % dei contributor totali del repository in esame, viene visualizzato un messaggio di avviso all’utente, di possibile rumore nel calcolo della Dispersione Culturale per il repository in questione.

3.4 Interfaccia Grafica

Nell’ambito di TRACE, l’interfaccia grafica gioca un ruolo fondamentale per garantire una navigazione semplice e intuitiva, essa è stata sviluppata utilizzando il framework React (vedi Sezione 3.2.1). Al caricamento del tool, l’utente viene accolto dall’*header* (vedi Figura 3.2) con logo e nome, che stabilisce un’identità visiva coerente per l’intera applicazione. Al di sotto dell’*header*, l’utente visualizza il *campo di input* (vedi Figura 3.3) che consente l’inserimento del repository GitHub che si vuole analizzare, con il seguente formato: OWNER/NAME. Dopo di che, fornito il repository, cliccando il pulsante “*submit*”, viene effettuata la validazione dell’input e, in caso di esito positivo, inviata la richiesta al back-end Flask (vedi Sezione 3.2.2) per avviare l’analisi.



Figura 3.2: Header

Enter OWNER/NAME of the repo to analyze:

Figura 3.3: Campo di Input

Durante la fase di elaborazione, appare un *loader animato* della forma del logo di TRACE, per segnalare all’utente che il processo è in corso e bisogna attendere. Questo feedback visivo è essenziale per evitare che l’utente possa pensare che il software si sia bloccato. Inoltre, viene disabilitato il pulsante “*submit*”, impedendo potenziali azioni inopportune da parte dell’utente, come tentativi ripetuti di invio (vedi Figura 3.4).

Enter OWNER/NAME of the repo to analyze:

cirovitale/TRACE

SUBMIT



Figura 3.4: Loader e Disabilitazione Submit

Se l’elaborazione venisse interrotta per un errore o un malfunzionamento, il tool rileva l’errore e notifica all’utente quest’ultimo mediante un *Compenente di Errore* (vedi Figura 3.5).

Enter OWNER/NAME of the repo to analyze:

lorem/lorem

Client Error: Not Found for url: https://api.git...

SUBMIT

! ERROR CODE: 404
 Client Error: Not Found for url: https://api.github.com/repos/lorem/lorem

Figura 3.5: Compenente di Errore

Altrimenti, in caso di esito positivo, una volta ricevuta la risposta dal server, il client presenta all’utente l’analisi del repository GitHub attraverso dati e grafici. Vengono visualizzati i seguenti componenti:

- *Componete Repository Info*: visualizza le principali informazioni relative al repository analizzato, inclusa la Dispersione Culturale calcolata. Presenta sia graficamente che numericamente il conteggio dei contributors associati ad ogni Cultura (vedi Figura 3.8). Nel caso in cui, oltre il 30,00 % dei contributors ha una predizione di Cultura nulla, viene notificato all’utente un avviso di potenziale imprecisione del calcolo della Dispersione Culturale (vedi Figura 3.6).

- *Componente Developers List*: mostra l'elenco dei contributors del repository, visualizzando nome, avatar e Cultura stimata (vedi Figura 3.7).
- *Componente User Info*: può essere attivato cliccando su uno specifico contributor mostrato nel *componente Developer List*. Esso visualizza le principali informazioni relative allo sviluppatore selezionato, mostrando i risultati intermedi dei moduli che hanno contribuito alla previsione della Cultura associata. Inoltre, posizionando il mouse sull'*icona di informazione* relativa al modulo predittivo, viene mostrata in un popup la motivazione associata a tale predizione (vedi Figura 3.9).

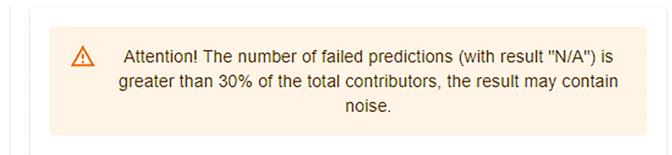
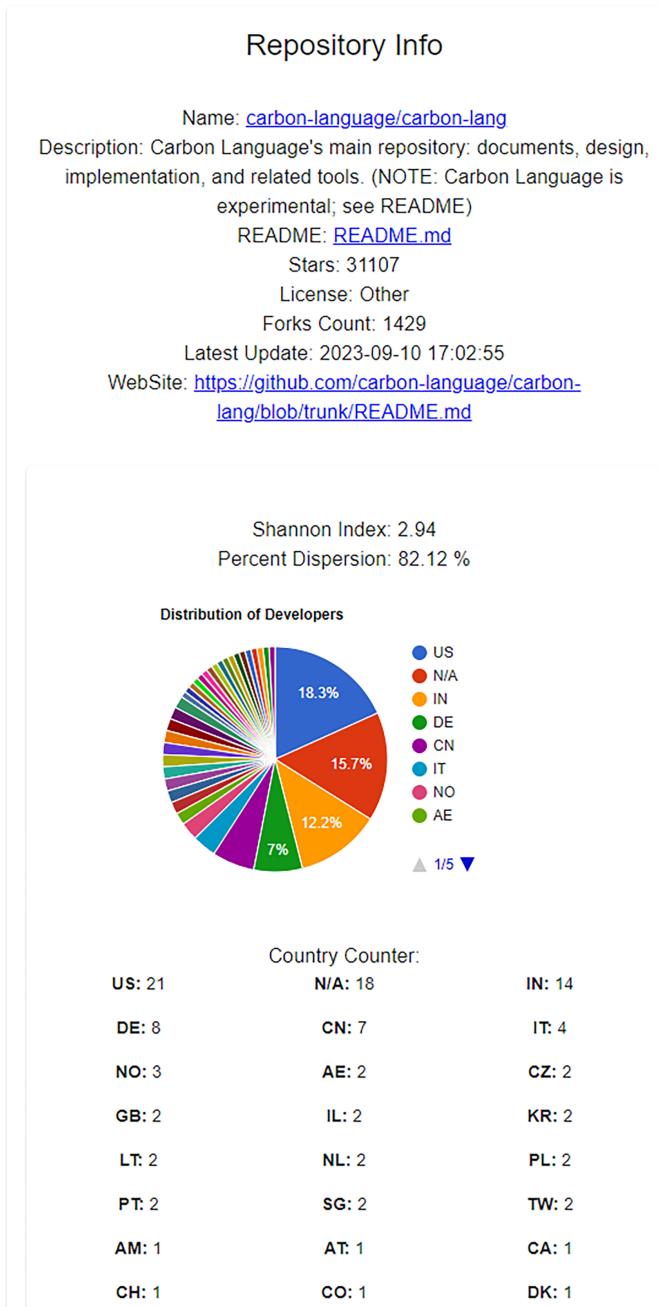


Figura 3.6: Notifica di Rumore nel Calcolo

A screenshot of the "Developers List" component. The title "Developers List" is at the top. Below it is a list of seven contributors, each with an avatar, name, and estimated country. The list is as follows:

- jonmeow (Estimated Country: US)
- zygoloid (Estimated Country: AE)
- geoffromer (Estimated Country: US)
- Pixep (Estimated Country: FR)
- jsiek (Estimated Country: US)
- prabhatexit0 (Estimated Country: IN)
- fasiddique (Estimated Country: PK)

Figura 3.7: Compenente Developers List

**Figura 3.8:** Compenente Repository Info

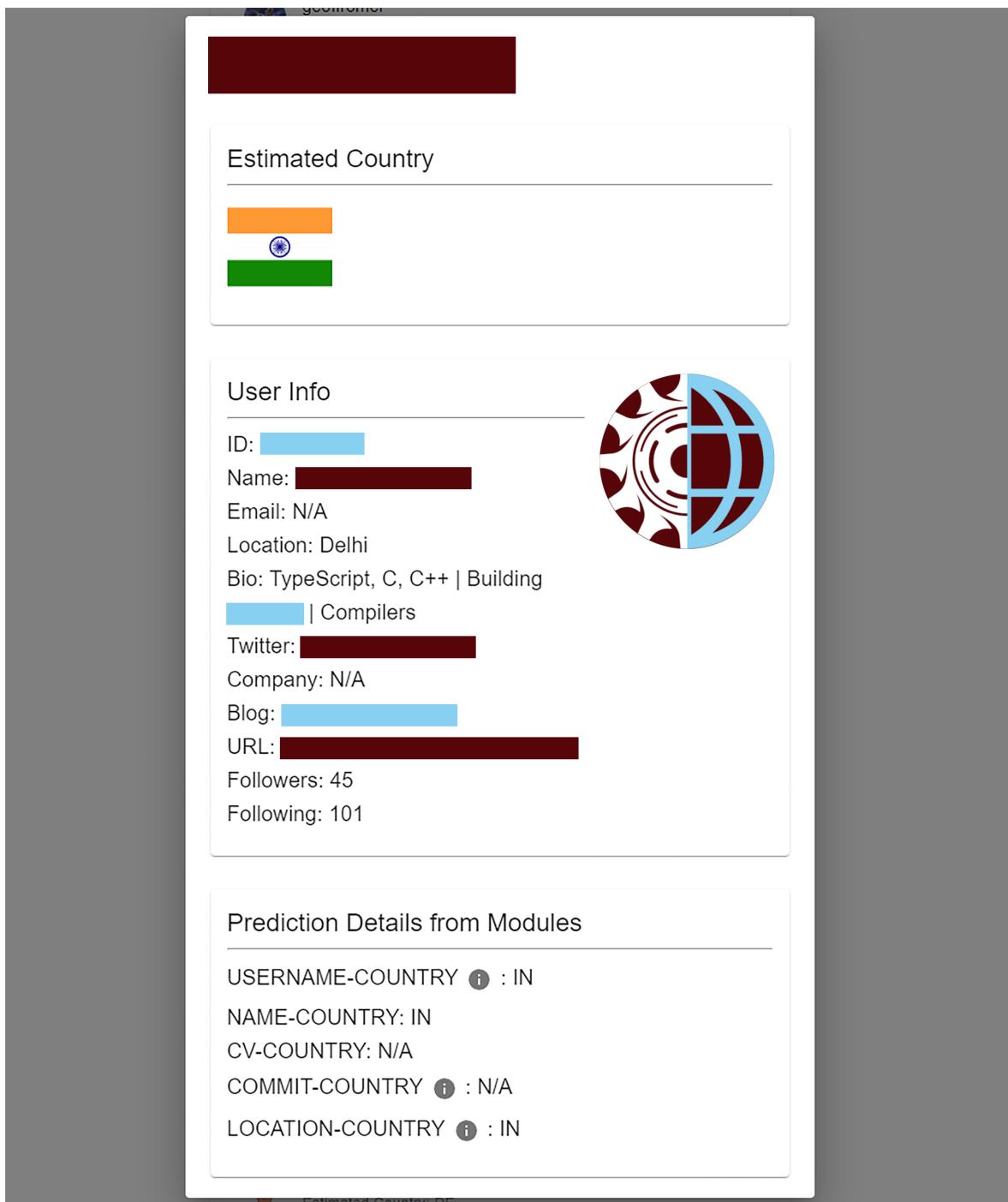


Figura 3.9: Componente User Info - Dati sensibili oscurati per *Privacy*

3.5 Manuale di Installazione

Le seguenti linee guida hanno lo scopo di agevolare il processo di installazione del tool TRACE. Prima di procedere con l'installazione di TRACE, è essenziale assicurarsi che il sistema soddisfi i seguenti *requisiti*:

- *Memoria*: minimo 16 GB di RAM, per supportare il caricamento in memoria del modello di ML.
- *Connessione Internet*: necessaria per le chiamate API.

Recupero Codice Effettuare il `clone` del repository GitHub² per recuperare il client e il server.

3.5.1 Configurazione Server

Download e Configurazione Assets Scaricare l'archivio TRACE_ML_ASSETS³ contenente le directory, `models` e `vectorizers`, da trasferire nella directory `server`.

Creazione file .env Creare un file denominato `.env` nella directory `server`, per conservare in modo sicuro le chiavi di configurazione e i token. Popolare il file con il seguente contenuto, inserendo le credenziali specifiche dove necessario. È importante notare che l'inserimento di tali credenziali è *facoltativo*. Tuttavia, fornendo tutte le credenziali richieste, il tool sarà in grado di operare al massimo delle sue potenzialità. In caso contrario, le potenzialità di TRACE saranno limitate.

```

1 GITHUB_API_TOKEN=""
2 OPENAI_API_KEY=""
3 OPENAI_ORGANIZATION_ID=""
4 GOOGLE_API_KEY=""

```

Creazione Ambiente Virtuale Creare l'*ambiente virtuale* `venv` utilizzando il seguente comando da *terminale* a livello della directory `server`.

²<https://github.com/cirovitale/TRACE>

³<https://cirovitale.me/ext/trace-ml-assets>

```
> python -m venv venv
```

Installazione Dipendenze Attivare l’ambiente virtuale `venv`, eseguendo il seguente comando da *terminale* a livello della directory `server`.

```
> venv\Scripts\activate
```

Infine, effettuare l’installazione delle dipendenze necessarie per garantire il corretto funzionamento del server Flask, attraverso il seguente comando da *terminale* a livello della directory `server`.

```
> pip install -r requirements.txt
```

3.5.2 Configurazione Client

Installazione Dipendenze È necessario effettuare l’installazione delle dipendenze richieste per assicurare il corretto funzionamento del client React, attraverso il seguente comando da *terminale* a livello della directory `client`.

```
> npm install
```

Configurazione Porta di Comunicazione Il client React, di default, è configurato per connettersi al server Flask, sulla porta 5000. Tuttavia, in alcuni ambienti, questa porta potrebbe essere già occupata da altre applicazioni o servizi. In tal caso, è possibile configurare il puntamento nel file `package.json` situato nella directory `client`, modificando la variabile `proxy` per puntare alla porta corretta.

```
"proxy": "http://127.0.0.1:5000"
```

3.6 Manuale d’Uso

Dopo aver completato le procedure di installazione (vedi Sezione 3.5), è possibile avviare il tool ed effettuare le analisi desiderate, seguendo le seguenti linee guide.

Avvio Server Per avviare il *server*, bisogna attivare l’ambiente virtuale `venv`, eseguendo il seguente comando da *terminale* a livello della directory `server`.

```
> venv\Scripts\activate
```

Dopodiché, è possibile avviare il server Flask mediante il seguente comando da *terminale* a livello della directory `server`, assicurandosi che sia in esecuzione sulla porta 5000 (vedi Sezione 3.5.2).

```
> flask run
```

Avvio Client Per avviare il *client*, bisogna eseguire il seguente comando da *terminale* a livello della directory `client`.

```
> npm start
```

Dopodiché, verrà automaticamente avviata l’applicazione web, aprendo una nuova finestra nel browser predefinito per visualizzare il tool.

Utilizzo del Tool Una volta avviati client e server, è possibile usufruire del tool, mediante la pagina web visualizzata (vedi Sezione 3.4). Per effettuare le predizioni bisogna inserire il repository che si intende analizzare nel campo di input, con il formato *OWNER/NAME*, e poi cliccare il pulsante "submit". Durante l’elaborazione, verrà visualizzato un loader animato. Successivamente, in caso di errore, viene visualizzato quest’ultimo, altrimenti, viene presentata l’analisi effettuata, in maniera dettagliata, sia per quanto riguarda il repository, sia per quanto riguarda i singoli contributor. Inoltre, è possibile monitorare il processo di elaborazione e visualizzare i risultati anche tramite i log del server.

CAPITOLO 4

Caso d'Uso del Tool

Nel contesto della ricerca e dello sviluppo di TRACE, è cruciale valutare le performance del tool *in applicazioni reali*. Tale valutazione è stata condotta su un dataset rappresentativo di repository GitHub, con caratteristiche differenti, per misurare la qualità delle predizioni del software, ma anche per osservare come il tool si adatta a diverse configurazioni e contesti di community. Si noti che alcuni moduli di elaborazione di TRACE sono non deterministici, per cui diverse esecuzioni potrebbero portare a risultati leggermente diversi. I risultati dell'analisi, effettuata in data 10/09/2023, sono stati esaminati e discussi in dettaglio.

4.1 Dataset Analizzato

| Nome | Contributors | Fork Attive | Stelle |
|------------------------------------|--------------|-------------|--------|
| <i>Hikki00/Raptor-AI</i> | 5 | 0 | 6 |
| <i>cheshire-cat-ai/core</i> | 45 | 136 | 875 |
| <i>carbon-language/carbon-lang</i> | 115 | 1.429 | 31.107 |

Tabella 4.1: Repository Analizzati con TRACE

I repository selezionati per valutare TRACE sono: *Hikki00/Raptor-AI*, *cheshire-cat-ai/core* e *carbon-language/carbon-lang*. Ognuno dei quali presenta caratteristiche uniche, rendendo ideale l’impiego di questo dataset per la *valutazione del tool* (vedi Tabella 4.1).

4.1.1 Hikki00/Raptor-AI

Il progetto *Raptor-AI* è il frutto della collaborazione tra il sottoscritto, Vitale Ciro, e gli altri membri del team: Consiglio Luigi, D’Antuono Francesco Paolo, Miron Roberto Andrei e Scermino Simone. Questo lavoro è stato ideato e sviluppato nell’ambito dell’esame *Fondamenti di Intelligenza Artificiale* dell’Università degli Studi di Salerno, focalizzato sulla creazione di agenti intelligenti capaci di competere contro un giocatore umano in un ambiente simulato in Unity. Trattasi, quindi, di un **progetto universitario open-source di piccole dimensioni**.

4.1.2 cheshire-cat-ai/core

Il progetto *cheshire-cat-ai/core* è un framework open-source sviluppato per *costruire intelligenze artificiali personalizzate* basandosi su qualsiasi *language model*. Nonostante il progetto goda di visibilità internazionale, ha solide radici italiane, in quanto il suo ideatore è *Piero Savastano*, data scientist e divulgatore scientifico italiano. Questo repository, considerato nel dataset di analisi, rappresenta un **progetto di medie dimensioni open-source**.

4.1.3 carbon-language/carbon-lang

Il progetto *carbon-language/carbon-lang* rappresenta un *linguaggio di programmazione general-purpose sperimentale* noto come Carbon. Tale linguaggio è sviluppato da Google, con l’intento di sostituire C++. Tale progetto, rappresenta un **grande progetto open-source**.

4.2 Esecuzione del tool

Dopo aver eseguito TRACE sui repository selezionati, sono stati ottenuti **ottimi risultati**. Sebbene questi possano leggermente variare a causa della natura non deterministica di alcuni moduli del tool, i risultati restano consistenti e rilevanti. La capacità di TRACE di analizzare diverse scale e tipologie di progetti è stata ampiamente dimostrata, confermandone l'efficienza (vedi Tabella 4.2).

La Tabella 4.2 presenta una sintesi dei risultati ottenuti attraverso l'analisi effettuata da TRACE. Ogni riga rappresenta un diverso repository analizzato, invece, le colonne forniscono le corrispondenti metriche chiave: la prima colonna è quella relativa al *nome*, poi ci sono quelle relative alla *predizione della Cultura*, dove è indicata sia la *copertura*, ovvero di quanti contributors il tool è riuscito ad effettuare una stima della Cultura, sia la *precisione* delle predizioni rispetto a una valutazione manuale. Infine, l'ultima colonna indica la *Dispersione Culturale* calcolata. La tabella fornisce, quindi, una panoramica su come TRACE ha perfomato sui repository selezionati, mostrando quanto spesso è stato in grado di effettuare una predizione e quanto queste sono state precise.

È importante notare quanto la *valutazione manuale* delle Culture dei contributors, soprattutto, si sia rivelata un processo estenuante. Essa è avvenuta mediante la valutazione manuale di nomi, siti web, social network e altre informazioni personali dello specifico contributor. Tale processo ha permesso di valutare la precisione del tool, evidenziando la validità delle previsioni di TRACE e confermando l'utilità di tale software automatizzato, che permette di risparmiare enormi quantità di tempo e risorse rispetto alle tecniche manuali.

| Nome | Predizione Cultura | | Disp. Culturale |
|-----------------------------|--------------------|------------|-----------------|
| | Copertura | Precisione | |
| Hikki00/Raptor-AI | 100,00 % | 80,00 % | 72,19 % |
| cheshire-cat-ai/core | 95,56 % | 81,40 % | 51,06 % |
| carbon-language/carbon-lang | 84,35 % | 85,57 % | 82,12 % |

Tabella 4.2: Risultati Repository Analizzati con TRACE

4.2.1 Hikki00/Raptor-AI

L'esecuzione del tool sul repository *Hikki00/Raptor-AI* ha portato alla stima della Cultura di 5 contributors su 5 totali. La *Dispersione Culturale* ottenuta è 72,19 %. Dopo una valutazione manuale, è stato confermato che il tool ha correttamente stimato 4 Culture su 5 effettive. Tali risultati hanno portato a una *precisione delle predizioni della Cultura* pari a 80,00 %.

4.2.2 cheshire-cat-ai/core

Il repository *cheshire-cat-ai/core*, ha permesso a TRACE di lavorare su un set di contributors più vasto. Il tool ha previsto la Cultura di 43 sviluppatori su 45 totali. La *Dispersione Culturale* calcolata è 51,06 %. Dopo un'attenta valutazione manuale, è stato constatato che il software ha identificato correttamente 35 Culture su 43. Ciò si traduce in una *precisione delle predizioni della Cultura* pari a 81,40 %.

4.2.3 carbon-language/carbon-lang

Il progetto *carbon-language/carbon-lang* ha permesso di effettuare un'esecuzione del tool su un set di contributors ben più vasto e variegato. Il software è riuscito a rilevare la Cultura per 97 contributors su 115 totali. La *Dispersione Culturale* risultante è 82,12 %. Dopo un'attenta ed estenuante valutazione manuale, è stato confermato che TRACE ha effettuato correttamente la predizione di 83 Culture, il che si traduce in una *precisione delle predizioni della Cultura* pari a 85,57 %.

CAPITOLO 5

Conclusioni

5.1 Conclusioni

TRACE ha dimostrato di essere uno strumento essenziale nell’analisi automatizzata di community open-source. L’importanza di comprendere e quantificare la Dispersione Culturale è notevole, non solo per favorire l’inclusione, ma anche per comprendere gli aspetti che possono influenzare la produttività in una comunità eterogenea. Attraverso un’accurata valutazione, il software ha dimostrato la sua capacità di adattarsi a vari contesti, producendo risultati ottimali, nonostante i limiti intrinseci nel contesto culturale e le limitate informazioni a disposizione.

5.2 Sviluppi Futuri

TRACE ha raggiunto traguardi significativi nel suo stato attuale, ma la ricerca deve continuare, esplorando le seguenti direzioni in cui il tool può ancora evolversi:

- *Migliorare le Prestazioni:* la priorità è continuare a *ottimizzare* il tool, sia in termini di accuratezza che di tempi di risposta.

- *Integrazione con altre Piattaforme*: espansione del tool per funzionare con altre piattaforme di sviluppo, come *Bitbucket* o *GitLab*.
- *Analisi Individualizzata*: orientare l'analisi anche al *singolo contributor*. Questo approccio permette di fornire informazioni utili ai reparti di *Human Resources*, nella selezione degli sviluppatori più adatti a un determinato progetto, considerando Dispersione Culturale del progetto e le specifiche esperienze culturali dello sviluppatore in questione.
- *Integrazione con LLama 2*: integrazione con il *LLM open-source* rilasciato da *Meta*, per rendere il *tool completamente open-source* [28].
- *Ricerca CV in Google Drive*: ampliare il modulo *CVToCountry* integrando la ricerca di potenziali CV in Google Docs o Google Drive, sfruttando le *Google Cloud API*, nei siti web dei contributor.

Bibliografia

- [1] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. O'Reilly Media, Inc., 2019. (Citato alle pagine iv, 8, 9, 10 e 11)
- [2] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, no. 3, 2023. [Online]. Available: <https://doi.org/10.1007/s11042-022-13428-4> (Citato alle pagine iv, 12 e 13)
- [3] S. Lambiase, G. Catolino, D. A. Tamburri, A. Serebrenik, F. Palomba, and F. Ferrucci, "Good fences make good neighbours? on the impact of cultural and geographical dispersion on community smells," in *Proceedings of the 2022 ACM/IEEE 44th International Conference on Software Engineering: Software Engineering in Society*, 2022. (Citato alle pagine 1, 3, 4 e 17)
- [4] S. Lambiase, G. Catolino, F. Pecorelli, D. A. Tamburri, F. Palomba, W.-J. Van Den Heuvel, and F. Ferrucci, ""there and back again?" on the influence of software community dispersion over productivity," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022, pp. 177–184. (Citato alle pagine 1, 4, 5 e 17)

- [5] E. Caballero-Espinosa, J. C. Carver, and K. Stowers, "Community smells—the sources of social debt: A systematic literature review," *Information and Software Technology*, p. 107078, 2022. (Citato alle pagine 1, 5 e 17)
- [6] M. Ortù, G. Destefanis, S. Counsell, S. Swift, R. Tonelli, and M. Marchesi, "How diverse is your team? investigating gender and nationality diversity in github teams," *Journal of Software Engineering Research and Development*, vol. 5, no. 1, pp. 1–18, 2017. (Citato alle pagine 3 e 5)
- [7] G. Hofstede, *Culture's Consequences: International Differences in Work-Related Values*, ser. Cross Cultural Research and Methodology. SAGE Publications, 1984. [Online]. Available: https://books.google.it/books?id=Cayp_Um4O9gC (Citato alle pagine 4 e 5)
- [8] R. Kreitner, M. Buelens, and A. Kinicki, *Organizational Behaviour: First European Edition*. McGraw-Hill, 1999. [Online]. Available: <http://lib.ugent.be/catalog/rug01:000455633> (Citato a pagina 4)
- [9] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, no. 1, p. 3–55, jan 2001. [Online]. Available: <https://doi.org/10.1145/584091.584093> (Citato alle pagine 4 e 16)
- [10] C. P. Earley and E. Mosakowski, "Creating hybrid team cultures: An empirical test of transnational team functioning," *Academy of Management journal*, vol. 43, no. 1, pp. 26–49, 2000. (Citato a pagina 5)
- [11] W. E. Watson, K. Kumar, and L. K. Michaelsen, "Cultural diversity's impact on interaction process and performance: Comparing homogeneous and diverse task groups," *Academy of management journal*, vol. 36, no. 3, pp. 590–602, 1993. (Citato a pagina 5)
- [12] R. T. Venkateswaran and A. K. Ojha, "Abandon hofstede-based research? not yet! a perspective from the philosophy of the social sciences," *Asia Pacific Business Review*, vol. 25, no. 3, pp. 413–434, 2019. (Citato a pagina 6)
- [13] F. Trompenaars and C. Hampden-Turner, "Riding the waves of culture," 01 1998. (Citato a pagina 6)

- [14] SAP, “What is machine learning?” *www.sap.com*, 2020. [Online]. Available: <https://www.sap.com/products/artificial-intelligence/what-is-machine-learning.html> (Citato a pagina 8)
- [15] A. Burkov, *Machine Learning Engineering*. True Positive Incorporated, 2020. [Online]. Available: <https://books.google.it/books?id=DSj9zQEACAAJ> (Citato alle pagine 8, 9 e 10)
- [16] K. Eda, “Nlp vs. nlu vs. nlg: the differences between three natural language processing concepts,” *www.ibm.com*, 2020. [Online]. Available: <https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/> (Citato a pagina 12)
- [17] W. A. Qader, M. M. Ameen, and B. I. Ahmed, “An overview of bag of words; importance, implementation, applications, and challenges,” in *2019 international engineering conference (IEC)*. IEEE, 2019, pp. 200–204. (Citato a pagina 13)
- [18] K. Sparck Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972. (Citato a pagina 13)
- [19] K. W. CHURCH, “Word2vec,” *Natural Language Engineering*, vol. 23, no. 1, p. 155–162, 2017. (Citato a pagina 14)
- [20] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543. (Citato a pagina 14)
- [21] L. R. Medsker and L. Jain, “Recurrent neural networks,” *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001. (Citato a pagina 14)
- [22] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_

- files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf (Citato a pagina 14)
- [23] D. Britz, "Understanding convolutional neural networks for nlp," URL: <http://www.wildml.com/2015/11/understanding-convolutional-neuralnetworks-for-nlp/> (visited on 11/07/2015), 2015. (Citato a pagina 14)
- [24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. (Citato a pagina 14)
- [25] H. Wang, J. Li, H. Wu, E. Hovy, and Y. Sun, "Pre-trained language models and their applications," *Engineering*, 2022. (Citato a pagina 15)
- [26] C. Gackenheimer, *Introduction to React*. Apress, 2015. (Citato a pagina 18)
- [27] M. Grinberg, *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018. (Citato a pagina 20)
- [28] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023. (Citato a pagina 44)

Ringraziamenti

È essenziale dedicare un momento a coloro che mi hanno accompagnato lungo il cammino della mia vita fino ad oggi. Mi scuso anticipatamente, se dovessi aver dimenticato di citare qualcuno ma, in generale, desidero ringraziare e dedicare questa tesi a tutti coloro che mi hanno voluto e che mi vogliono bene, *grazie!*

Ringrazio infinitamente la mia famiglia: mia mamma *Grazia*, mio padre *Tonino*, mia sorella *Anna Rita* e la mia nipotina *Ilary*. Loro che mi hanno insegnato i valori della vita, che mi hanno sempre sostenuto e che mi hanno dato tutto. Un grazie speciale a nonna *Anna* e nonna *Annamaria*, a nonno *Aurelio* e nonno *Ciro*. Grazie di cuore a *zia Cinzia*, *zio Massimo* e mia cugina *Annamaria*.

Grazie al mitico *Don Franco* e alla moglie *Giovanna*.

Grazie a *Luca* e a ogni singolo *collega aziendale*.

Grazie a tutti i *docenti* e i *compagni di classe* che ho avuto, dall'asilo all'università. In particolar modo grazie al mio Relatore, il *Prof. Palomba Fabio*, e al *Dott. Lambiase Stefano*, per avermi accompagnato in questo frangente finale accademico.

Grazie ai miei compagni di viaggio, che hanno reso fantastica questa avventura triennale universitaria: *Luigi, Francesco, Eljon, Roberto e Simone*.

Grazie a *Mirco* e a tutto il gruppo di amici che ho conosciuto grazie alla *Salernitana*.

Grazie di cuore ai miei amici di una vita, *Alessandro e Michele*.

Dulcis in fundo, un ringraziamento speciale alla donna che mi è sempre stata vicina e che mi ha dato la forza di non mollare mai, e alla sua famiglia. *Grazie Carmen!*