UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Estrazione di requisiti non funzionali di sicurezza tramite tecniche di NLP

PRIMO RELATORE

Prof. **Fabio Palomba**

SECONDO RELATORE

Dott. **Francesco Casillo**

Università degli studi di Salerno

CANDIDATO

Dario Mazza

Matricola: 0512106742

Anno Accademico 2021-2022

"La vita è come andare in bicicletta: se vuoi stare in equilibrio devi muoverti"

Albert Einstein

Sommario

I requisiti non funzionali (NFR) sono un insieme di attributi e vincoli di qualità che un sistema software deve soddisfare. Alcuni di questi, come quelli di sicurezza, sono di cruciale importanza poiché la loro mancata individuazione nelle prime fasi della realizzazione del sistema comporta la creazione di software con meccanismi di sicurezza inadeguati, e quindi con molte vulnerabilità che possono comprometterne il corretto funzionamento oltre che causare continue revisioni e correzioni del sistema che rallentano lo sviluppo e incompatibilità con specifici standard governativi e commerciali.

Lo scopo di questa tesi è quello di cercare di automatizzare il processo di riconoscimento di tali requisiti, in modo da avere un punto di partenza su cui gli ingegneri del software potranno basarsi per la fase di elicitazione e definizione dei requisiti di sicurezza.

Gli studi effettuati ci hanno permesso di creare un tool, che attraverso tecniche di *natural language processing* (NLP) e *di machine learning*, è capace di riconoscere e successivamente classificare i requisiti di sicurezza a partire da documenti scritti in linguaggio naturale.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

L'ingegneria dei requisiti [1] è un processo di raccolta e definizione dei servizi che il sistema deve fornire. L'ingegneria dei requisiti si concentra sulla valutazione dell'utilità e fattibilità del sistema da sviluppare per l'azienda (studio di fattibilità), sulla scoperta dei requisiti (elicitazione e analisi), sulla conversione di questi requisiti in un formato standard (specifica) e sulla verifica che i requisiti definiscano il sistema desiderato dal cliente (validazione). I requisiti del software sono classificati in requisiti funzionali e requisiti non funzionali:

- **Requisiti funzionali:** riguardano le funzioni principali che devono essere fornite dal sistema. Possono descrivere anche il comportamento del sistema a fronte di particolari input e come esso dovrebbe reagire in determinate situazioni.
- **Requisiti non funzionali:** rappresentano i vincoli e le proprietà relative al sistema, come il numero di processi che il sistema può gestire (prestazioni), quali sono i problemi (di sicurezza) di cui il sistema deve occuparsi ecc.

I requisiti non funzionali sono spesso più critici di quelli funzionali. Di solito gli utenti possono trovare il modo di aggirare una funzione del sistema che non soddisfa le loro esigenze.

Tuttavia, il mancato rispetto di un requisito non funzionale può rendere inutilizzabile l'intero sistema. Infatti l'elicitazione e il riconoscimento di quest'ultimi costituiscono una fase fondamentale del processo di sviluppo del software, poiché tutti gli errori effettuati in questa fase hanno un impatto rilevante sui costi del progetto e questo, come dimostrano gli studi [2], è uno dei motivi più comuni per cui il progetto fallisce.

1.2 Motivazioni e Obiettivi

Tra i requisiti software, una tipologia in particolare determina la robustezza e l'affidabilità del sistema: i **requisiti di sicurezza**. Nello sviluppo del progetto, spesso gli ingegneri tendono a concentrarsi sui requisiti funzionali e i requisiti relativi alla sicurezza vengono considerati in fasi più avanzate del ciclo di vita del software, come ad esempio durante la fase di implementazione. Questo comporta la creazione di sistemi con meccanismi di sicurezza inadeguati e quindi con molte vulnerabilità che possono comprometterne il corretto funzionamento. I requisiti di sicurezza sono requisiti non funzionali, ma che riguardano il funzionamento del sistema e per questo motivo sono spesso distribuiti in tutta la documentazione relativa.
Per di essi si possono suddividere in altrettante sottocategorie:

- Confidentiality;
- Integrity;
- Availability;
- Accountability;
- Operational;
- Access control & Identity;
- Privacy.

Questa natura pervasiva ed eterogenea che caratterizza i requisiti di sicurezza, rende ancora più difficile la loro corretta individuazione e dichiarazione da parte degli analisti.
Il modo più tipico per esprimere i requisiti software è il linguaggio naturale, questo approccio è soggetto a molteplici errori poiché si affida alla capacità umana di comprendere il linguaggio. Lo scopo di questa tesi è quello di cercare di automatizzare il processo di riconoscimento di tali requisiti, in modo da avere un punto di partenza su cui gli ingegneri del software potranno basarsi per la fase di elicitazione e definizione dei requisiti di sicurezza.

1.3 Risultati

Gli studi effettuati hanno portato alla realizzazione di uno strumento che, prendendo in input degli artefatti scritti in linguaggio naturale, è in grado di identificare i requisiti di sicurezza e successivamente di classificarli nelle varie tipologie.

Per realizzare questo tool è stato creato un dataset a partire da sei documenti pre-elaborati, per un totale di circa 10.000 istanze, utilizzati in uno studio condotto da *Riaz et al.* . Questi documenti sono stati processati effettuando una fase di pulizia dei dati ottenendo un dataset contenente circa 6000 requisiti. Successivamente sono state utilizzate varie tecniche di NLP per elaborare i dati testuali e renderli analizzabili dai diversi modelli di machine learning implementati in questo studio. I risultati più promettenti sono stati quelli dei modelli:

- **Logistic Regression** e **LinearSVC** per quanto riguarda l'identificazione di un requisito di sicurezza;
- **RandomForest** e **DecisionTree** per la classificazione dei requisiti nelle sottocategorie di appartenenza.

1.4 Struttura della tesi

Nel capitolo 2, si trova una panoramica sui requisiti non funzionali, vengono trattate tematiche relative allo stato dell'arte sull'identificazione e classificazione dei requisiti non funzionali in generale. Successivamente vengono riportate le informazioni presenti in letteratura sul riconoscimento e classificazione dei requisiti di sicurezza nello specifico.
Nel capitolo 3, sono descritte l'architettura del progetto, l'analisi del dataset e le successive fasi di elaborazione dei dati, la scelta dei modelli di machine learning e tutte le tecniche e le metriche di validazione di quest'ultimi.
Nel capitolo 4 sono presenti tutte le informazioni relative ai risultati ottenuti dai modelli. In particolare verranno descritte e motivate le metriche utilizzate e saranno confrontati tutti i modelli implementati.
Nel capitolo 5, è presente una descrizione sommaria di ciò che è stato fatto a partire dalla creazione del dataset fino alla discussione dei risultati finali e vengono effettuate delle considerazioni in merito ai possibili sviluppi futuri.

CAPITOLO 2

Background Stato dell'arte

2.1 Background

2.1.1 Requisiti non funzionali

L'ingegneria dei requisiti è un processo fortemente incentrato sull'uomo. Di conseguenza, la conoscenza del dominio applicativo da parte degli stakeholders è un fattore determinante nell'elicitazione dei requisiti del software. Essi sono principalmente espressi in linguaggio naturale [3]. L'estrazione dei requisiti non funzionali (NFR) viene effettuata indagando su quali sono i problemi che vanno ad influire sulla qualità del software attraverso questionari, template, checklists ecc... rivolti agli stakeholders [4].

Nonostante l'importanza che i NFR hanno, tutt'ora non esiste una definizione formale e rigorosa che li descriva. Vengono molto spesso definiti come requisiti che non descrivono cosa il sistema deve fare (requisiti funzionali) ma piuttosto come lo farà [5]. Sono anche noti come caratteristiche o vincoli di qualità. Alcuni tipici requisiti non funzionali sono: *Performance, Scalability, Capacity, Availability, Reliability, Recoverability, Maintainability, Serviceability, Security, Regulatory, Manageability, Environmental, Data Integrity, Usability, Interoperability*.

2.1.2 Identificazione e classificazione dei requisiti non funzionali

L'estrazione e la classificazione automatica dei NFR da documenti è stata al centro dell'attenzione di diversi ricercatori nel campo dell'ingegneria dei requisiti. Le specifiche dei requisiti sono solitamente scritti in linguaggio naturale. I documenti contengono sia requisiti funzionali e sia requisiti non funzionali in maniera mista. Esistono diversi approcci per estrarli dai documenti.

L'estrazione automatica dei NFR da documenti di grandi dimensioni nelle fasi di sviluppo iniziali assume notevole rilevanza, poiché essa risulta essere particolarmente complessa nel ciclo di vita del software. Inoltre le fonti da analizzare sono molteplici (specifiche dei requisiti, linee guida, SOP[1], manuali utente ecc...), rendendo il processo ancora più articolato. Riuscire ad estrarre i NFR in maniera efficace consente agli analisti di evitare eventuali revisioni e ristrutturazioni ma anche di risparmiare tempo.

Una volta estratti correttamente i requisiti, è opportuno anche classificarli poiché essi sono disposti maniera eterogenea all'interno dei documenti, ed è importante identificarli separatamente [6].

Cleland-Huang *et al.* (2006) [7] hanno descritto un approccio automatizzato per la classificazione dei NFR presenti nei vari artefatti prodotti durante le diverse fasi del processo di sviluppo del software. L'approccio proposto si basa sul fatto che i diversi tipi di NFR possono essere distinti da alcune parole chiave. Tali parole chiave vengono estratte manualmente dai documenti e utilizzati per addestrare un modello che verrà successivamente utilizzato per classificare altri requisiti. L'approccio proposto è stato testato su 30 progetti software realizzati da studenti universitari e ha mostrato una recall di classificazione del 70%. Tuttavia, nonostante l'obiettivo principale era massimizzare la recall rispetto alla precision, è stata registrata una percentuale troppo alta di falsi positivi.

Nel 2010 Casamayor *et al.* [8] hanno ripetuto l'esperimento utilizzando un approccio semi-supervisionato, basato su un classificatore Bayesiano multinomiale, in grado di minimizzare il lavoro svolto dagli analisti per etichettare il dataset. Rispetto all'approccio supervisionato la tecnica usata in questo lavoro ha permesso di automatizzare la fase di categorizzazione manuale dei requisiti per l'addestramento del modello, riducendo di molto lo sforzo richiesto dagli ingegneri del software. Questa metodologia ha prodotto buoni risultati con un'accuratezza superiore al 70%.

[1] Standard Operating Procedure: sono istruzioni step-by-step destinate ai dipendenti per guidarli nel completamento e nell'esecuzione dei principali processi aziendali.

Zhang *et al.* [9] nel 2011 hanno replicato nuovamente l'esperimento, ma hanno utilizzato come algoritmo di machine learnig Support Vector Machine con un kernel lineare. Hanno riportato una precision significativamente più alta, anche se una recall più bassa rispetto al lavoro svolto da di Cleland-Huang *et al.* ma non hanno fornito dettagli in merito.

Slankas e Williams (2013) [10] hanno sviluppato un tool chiamato *NFRLocator*[2] capace di estrarre i requisiti non funzionali da documenti in linguaggio naturale. La classificazione dei requisiti è stata effettuata sulla base delle categorie dei NFR (14 in questo caso). Sono stati utilizzati molteplici classificatori in questo studio, riscontrando le performance migliori (F-measure del 54%) utilizzando il k-nearest neighbors.

Nel 2016 Mahmoud e Williams [11] hanno utilizzato tecniche di apprendimento non supervisionato per individuare e classificare i requisiti non funzionali. Le parole chiave sono determinate attraverso il clustering e la classificazione dei NFR viene effettuata con il metodo della Normalized Google Distance (NGD). I ricercatori hanno hanno valutato il loro lavoro ottenendo una precisione media del 53% ed una recall dell'83%.

2.2 Stato dell'arte

2.2.1 Indentificazione e classificazione dei requisiti di sicurezza

Classificazione binaria

Knauss *et al.* [12] (2011) hanno sviluppato un classificatore binario per stabilire se un requisito fosse o meno relativo alla security. In particolare è stato utilizzato un classificatore Bayesiano ottenendo una F-measure massima dell'84%, che però scende al di sotto del 60% se si sceglie di usare un dominio applicativo diverso da quello utilizzato per l'addestramento del modello.

Li *et al.* [13] (2017) hanno proposto un tool per identificare in modo efficiente i requisiti di sicurezza, che combina l'analisi linguistica con tecniche di machine learning. Nella fattispecie, hanno applicato un approccio sistematico per identificare caratteristiche linguistiche sulla base di ontologie e conoscenze linguistiche di requisiti di sicurezza esistenti. Queste caratteristiche vengono estratte automaticamente dai requisiti testuali, che vengono poi utilizzate per addestrare i classificatori usando le tipiche tecniche di machine learning. Il modello ha ottenuto performance inferiori rispetto all'articolo citato in precedenza (77% di F-measure) ma si

[2] `https://github.com/RealsearchGroup/NFRLocator`

è rivelato più stabile se usato su altri domini applicativi ottenendo un valore di F-measure del 61%.

Palacio *et al.* [14] (2019) hanno realizzato un classificatore chiamato *SecureReqNet* per l'identificazione dei requisiti di sicurezza. Questo classificatore binario combina tecniche di word embedding con varie architetture di CNN il numero di neuroni negli strati completamente connessi. Il modello è stato testato sia su dataset ristretti (insieme di user stories di progetti aziendali) sia su dataset di dimensioni notevoli (requisiti da open source estratti da varie repository su GitHub e GitLab) ottenendo rispettivamente un'accuratezza del 71% e 96%.

Classificazione Multi-label & Multiclasse

Gli stessi Slankas, Williams *et al.* (2013) [15] hanno proposto anche un tool in grado di identificare e classificare requisiti relativi alla security in base a degli specifici obiettivi di sicurezza forniti dagli analisti. Lo strumento prende in input un insieme di documenti in linguaggio naturale da elaborare. Sono stati usati calssificatori k-NN e naïve Bayes ottenendo una precision dell'82% e una recall del 79%.

Jindal *et al.* (2016) [16] hanno eseguito un'analisi automatizzata di vari documenti di specifiche di requisiti dall' archivio PROMISE[3]. Il dataset è costituito da specifiche dei requisiti raccolte da 15 progetti sviluppati dagli studenti della DePaul University. Gli autori dell'articolo hanno creato un classificatore capace di categorizzare diversi tipi di requisiti di sicurezza (Authentication-Authorization, Access Control, Cryptography- Encryption e Data Integrity) usando solo un algoritmo di ML (J48 Decision Tree) e varie tecniche di text mining, ottenendo risultati promettenti.

Nel 2021 Varenov *et al.* [17] hanno effettuato uno studio sull'elicitazione e classificazione dei requisiti di sicurezza. In particolare la classificazione di questi avviene tramite modelli di deep learning. A tal fine i ricercatori hanno usato come modello di deep learning DistilBERT, una versione modificata del noto trasformatore BERT. Il modello è stato utilizzato in prima battuta su vari dataset già esistenti, ottenendo risultati discreti in termini di precision e F-measure. Successivamente gli autori hanno deciso di creare il proprio dataset per minimizzare i problemi che hanno riscontrato con quelli utilizzati in precedenza, ottenendo il 78% di F-measure. Inoltre questi esperimenti hanno mostrato che i modelli usati funzionano bene anche su dataset di dimensioni ridotte, effettuando opportune operazioni di tuning.

[3] `http://promisedata.org/repository`

CAPITOLO 3

Metodologia

3.1 Architettura del progetto

Il sistema sarà composto da due agenti intelligenti:

1. Il primo sarà un classificatore binario che, dato un requisito, si occuperà di stabilire se è di sicurezza o meno.

2. Il secondo sarà un classificatore multi-label che dovrà essere in grado di stimare l'appartenenza del requisito a una o più categorie di sicurezza.

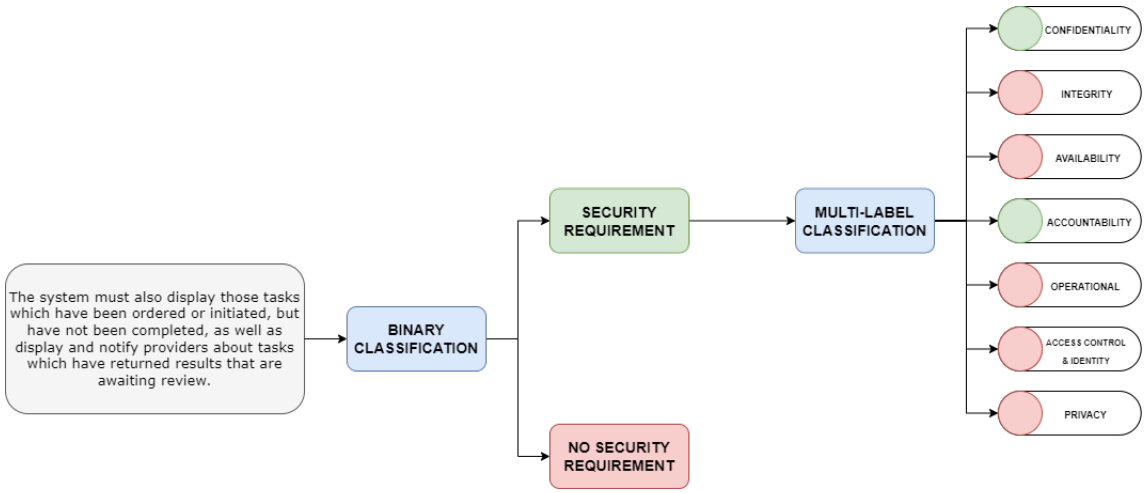

Figura 3.1: Esempio di identificazione e classificazione di un requisito. In verde le categorie del requisito predette dal modello di machine learning.

3.2 Analisi del Dataset

Per questa ricerca è stato utilizzato il dataset realizzato da Riaz *et al.* [15]: per crearlo si sono basati su sei documenti relativi al dominio applicativo dell'*healthcare*. I ricercatori hanno letto manualmente ogni frase in linguaggio naturale presente nei documenti e i requisiti che venivano riconosciuti come di sicurezza sono stati sottoposte ad un'ulteriore analisi per la categorizzazione. Il dataset è composto da circa 10.000 frasi, ed ogni requisito di sicurezza appartiene ad una o più delle seguenti categorie:

- **Access control & identity**
- **Confidentiality**
- **Integrity**
- **Availability**
- **Accountability**
- **Privacy**
- **Technical**
- **Management**
- **Operational**
- **Database**

Inoltre per semplificare la fase di estrazione dei requisiti dal dataset, gli autori hanno aggiunto informazioni aggiuntive per ogni frase, come il campo *sentenceType* che può essere:

- **TITLE**: se la frase si riferisce ad un titolo di una sezione o di un paragrafo;
- **LIST_START**: se la frase è la parte iniziale di un elenco puntato o numerato;
- **LIST_MEMBER**: se la frase corrisponde ad un elemento di un elenco;
- **NORMAL**: se il testo corrisponde ad una normale frase dei documenti originari.

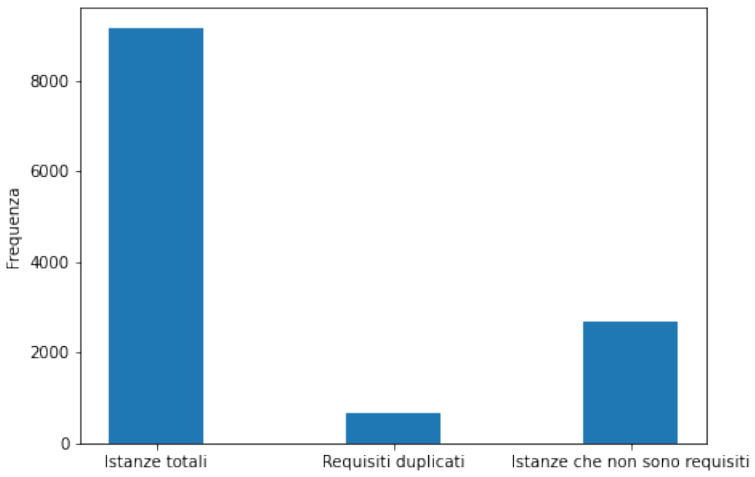

Figura 3.2: Dataset prima di effettuare l'operazione di pulizia dei dati. Tra le istanze che non sono requisiti si trovano tutte quelle contenenti esclusivamente caratteri speciali, email, url, errori, log ecc.

3.3 Elaborazione dei dati

3.3.1 Data cleaning

In questa fase sono state analizzati tutti i requisiti valutandone la consistenza, la correttezza sintattica e semantica dei contenuti, cercando di ottenere per quanto possibile un dataset privo di istanze che non si riferiscono propriamente ad un requisito del sistema o senza un senso logico.

In particolare le operazioni di pulizia dei dati che sono state effettuate sono:

- Esclusione di tutte le istanze contenenti solo caratteri speciali;
- Concatenazione degli elementi *LIST_MEMBER* con i corrispettivi elementi *LIST_START*;
- Riconoscimento di tutte le istanze contenenti titoli di paragrafi o sezioni (molti titoli non erano correttamente etichettati come tali nei file originali);
- Esclusione di tutte le istanze contenenti un numero di parole inferiore a due;
- Esclusione di tutte le istanze contenenti solo URL, identificativi di *user stories* o email;
- Esclusione delle istanze contenenti solo informazioni temporali, come orari, date ecc...
- Esclusione delle istanze espresse come prototipi di funzioni, metodi o procedure, o come errori e messaggi log;
- Rimozione delle istanze espresse come didascalie di immagini o tabelle;
- Inoltre sono state effettuate anche operazioni di **normalizzazione** dei requisiti:
  - Rimozione di caratteri non riconosciuti presenti in molti requisiti;
  - Rimozione di spazi bianchi all'inizio del requisito;
  - Aggiunta del punto alla fine di ogni requisito se non presente;
  - Rimozione degli apici, doppi apici o parentesi se racchiudono l'intero requisito, o se vengono aperti e mai chiusi;
  - Modifica dei requisiti affinché tutti inizino con la lettera maiuscola;

Tutte le operazioni riportate sopra sono state attuate mediante l'uso di **espressioni regolari**. Alla fine di questa procedura sono state rimosse righe duplicate dal dataset in quanto molto frequenti, ottenendo un dataset contenente 5980 requisiti di cui 4032 sono requisiti di sicurezza.

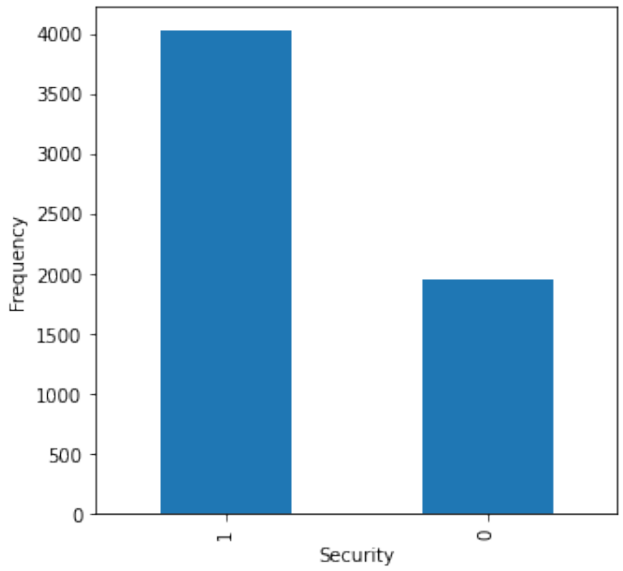

Figura 3.3: Dataset dopo aver effettuato l'operazione di pulizia dei dati. I requisiti di sicurezza sono contrassegnati dall'etichetta "1"

3.3.2 Dizionario Security Keywords

Per ottenere informazioni aggiuntive riguardo la classificazione dei requisiti di sicurezza, abbiamo usato un dizionario di *security keywords*, contenente una lista di parole chiave strettamente legate alla sicurezza: per ogni requisito nel dataset sono state aggiunte tutte le keywords che sono presenti all'interno della frase in un'apposita colonna.

Le parole chiave che si trovano nel dizionario sono state estratte da due glossari online:

- SysAdmin, Audit, Network, and Security (SANS Institute) [1]
- National Initiative for Cybersecurity Careers and Studies (NICCS) [2]

Per creare il dizionario di parole chiave è stata utilizzata le libreria *BeautifulSoup*[3] per estrarre dai glossari i vari termini di sicurezza e aggiungerli ad una lista. Le keyword ottenute vengono ulteriormente elaborate cercando eventuali acronimi. Ad esempio la parola *"Cyclic Redundancy Check (CRC)"* viene divisa in due parole separate, *"Cyclic Redundancy Check"* e *"CRC"* che vengono aggiunte entrambe alla lista. Questo procedimento è ripetuto per entrambi i glossari evitando eventuali ripetizioni.

[1] `https://www.sans.org/security-resources/glossary-of-terms/`
[2] `https://niccs.cisa.gov/cybersecurity-career-resources/glossary#body`
[3] `https://www.crummy.com/software/BeautifulSoup/`

3.3.3 Applicazione tecniche di NLP

Per poter utilizzare i dati testuali come *features* dei vari modelli, è necessario modellarli al fine di renderli utilizzabili dagli agenti intelligenti. Per tale scopo utilizziamo le tecniche di Natural Language Processing (NLP), ovvero degli algoritmi di intelligenza artificiale in grado di analizzare, rappresentare e quindi comprendere il linguaggio naturale. Per applicare queste tecniche è stato usato come toolkit *spaCy*[4], uno dei framework NLP più recenti e potenti in circolazione. In particolare per questo studio sono state utilizzate le seguenti funzionalità:

1. **Tokenization**: la tokenization consiste nella suddivisione di un testo in più parti, dette token. Quando svolgiamo questa operazione, suddividiamo un testo di partenza in diversi token. Nel nostro caso divideremo il testo in token basandoci sulle parole, evitando segni di punteggiatura.

```
def get_tokens(sentence):
    doc = nlp(sentence)
    tokens = []
    for token in doc:
        if str(token) not in punctuation:
        tokens.append(token.text)
    return tokens

sentence = "This is a security requirement."
print(get_tokens(sentence))

#Output:
#['This', 'is', 'a', 'security', 'requirement']
```

Esempio di tokenization con spaCy

2. **Parts of Speech**: l'operazione di *PoS* (Part-of-Speech) *tagging* è il processo di assegnazione di etichette diverse alle parole di una frase, permettendoci di classificarle in base alle proprietà morfologiche o alle funzioni sintattiche di queste. La PoS di una parola fornisce informazioni fondamentali per determinare il ruolo della parola stessa e di quelle vicine nella frase.

[4] https://spacy.io/

3. **Dependecy Parsing**: Il *Dependency parsing* è il processo di analisi della struttura grammaticale di una frase, e delle varie dipendenze che intercorrono tra ogni parola presente, mediante l'uso di particolari etichette chiamate *Dependency tag*. Questa fase è fondamentale in quanto permette di riconoscere frasi che sono sintatticamente diverse ma che hanno lo stesso significato semantico come la stessa frase.

Text	Part of speech	Dependecy
The	DET	det
system	NOUN	nsubj
should	AUX	aux
provide	VERB	ROOT
a	DET	det
means	NOUN	dobj
for	SCONJ	mark
a	DET	det
provider	NOUN	nsubj
to	PART	aux
display	VERB	relcl
patient	NOUN	compound
records	NOUN	dobj
needing	VERB	acl
attention	NOUN	dobj
or	CCONJ	cc
completion	NOUN	conj

Tabella 3.1: Esempio di PoS e Dependency tags per un requisito

4. **Entity Recognition**: La Named Entity Recognition è un insieme di operazioni che ci permettono di di identificare specifiche entità nel testo e associarle alle corrispondenti categorie semantiche come persone, organizzazioni, entità di tipo geopolitico, geografico, numeri, espressioni temporali ecc...

   La lista delle entità può anche essere allargata aggiungendone di nuove manualmente.

Figura 3.4: Esempio di Named Entity Recognition con spaCy

Dopo aver applicato le tecniche di NLP, il dataset costruito si presenta con le seguenti informazioni:

- *Sentence*: il requisito in forma testuale;
- *Entities*: le entità rilevate;
- *Dependencies*: Dependency tags;
- *Parts of Speech*: PoS tags;
- *File*: File di provenienza del requisito;
- *Categories*: categorie del requisito ;
- *Security Words*: parole chiave di sicurezza associate al requisito;
- *Security*: valore numerico che indica se il requisito sia di sicurezza o meno (1 = Sicurezza, 0 altrimenti).

3.3.4 Scelta delle categorie di sicurezza

Pulito il dataset, ci siamo focalizzati sulla scelta delle categorie di sicurezza, basandoci principalmente sullo studio effettuato da Varenov *et al.* [17]. A differenza del loro lavoro, nel quale hanno selezionato solo la prima categoria per ogni requisito (ottenendo un problema **multiclasse**), in questo studio è stato deciso di usarle tutte, ottenendo quindi un problema **multi-label**. Le categorie che abbiamo deciso di usare sono le seguenti:

- **Confidentiality**: Questa proprietà è soddisfatta in un sistema quando i dati scambiati tra mittente e destinatario non possono essere decifrati da una terza parte, anche nel caso in cui dovesse intercettare il messaggio.
  - Es: *The system shall provide a means to edit discharge instructions for a particular patient.*
- **Integrity**: Questa proprietà è soddisfatta quando i dati privati non possono essere modificati o cancellati da un terzo senza i permessi appropriati.
  - Es: *All users should be able to see and view deleted and active documents.*
- **Availability**: Questa proprietà è soddisfatta quando il sistema è in grado di funzionare regolarmente anche in caso di attacchi DoS (Denial of Service), DDoS (Distributed Denial of Service), guasti hardware o software ecc..
  - Es: *VLER DAS stores event descriptions in an audit log for a minimum of six (6) years.*

- **Accountability**: Questa proprietà è soddisfatta quando il sistema è in grado di proteggersi contro un individuo che neghi di aver eseguito una determinata azione dichiarando il falso. Le azioni degli utenti sono registrate e questi non possono in alcun modo modificare l'esito delle loro azioni.

  – Es: *The system should provide the ability to check medications against a list of drugs noted to be ineffective for the patient in the past.*

- **Operational**: Si tratta di una categoria di requisiti che non si riferiscono direttamente al software in sé, ma al modo in cui dovrebbe essere gestito per rimanere sicuro, ad esempio come gestire le terze parti, i providers ecc.

  – Es: *Nurses need to advocate for patients and mitigate the risk of miscommunication possibly and possibly misinterpretation of patient information.*

- **Access control & Identity**: Questa categoria specifica come l'autenticazione degli utenti nel sistema dovrebbe essere gestita, e quali risorse sono a disposizione di un utente autenticato.

  – Es: *In the event that a system does not support pre-login capabilities, the system shall display the banner immediately following authorization.*

- **Privacy:** Questa proprietà è soddisfatta quando le informazioni private degli utenti sono protette da accessi non autorizzati, misura necessaria ad evitare danni in caso di furto. Può essere considerata come un'estensione della categoria *Confidentiality*

  – Es: *Informs nursing and inter-professional practice by obtaining anonymized data for the purpose of health system use.*

Analisi Keywords

Scelte le categorie di sicurezza nel nostro dataset, è stata effettuata un'analisi statistica sulla possibile correlazione tra le parole chiave di sicurezza estratte nelle fasi precedenti. Per prima cosa abbiamo rappresentato la frequenza delle keywords associate ad almeno una categoria di sicurezza nel dataset tramite una *word cloud* (Figura 3.5), appurando che le tre parole chiave più frequenti sono *access*, *user* e *client*.

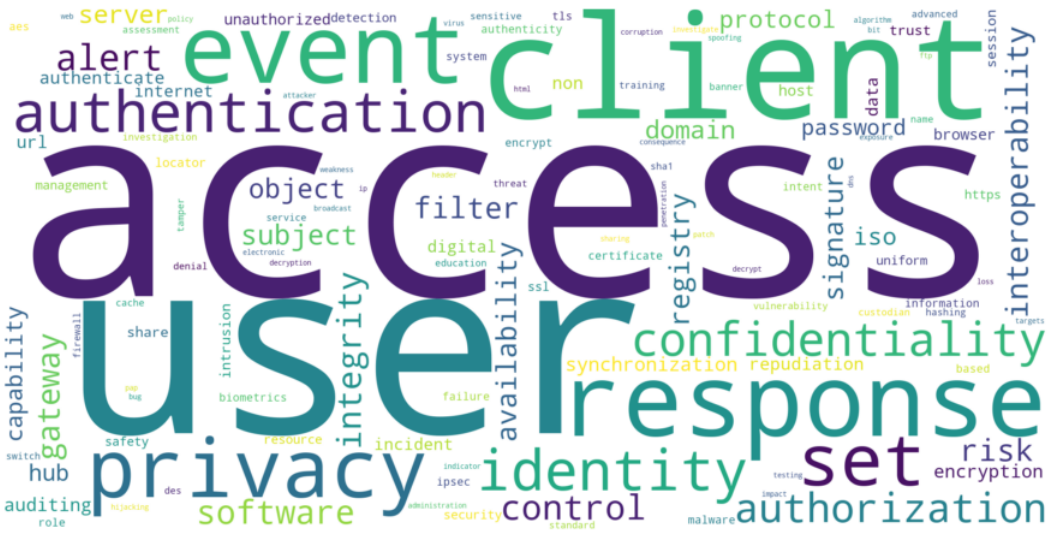

Figura 3.5: Keywords associate ai requisiti di sicurezza del dataset

A seguire abbiamo considerato per ogni categoria di sicurezza il numero di requisiti senza nessuna parola chiave associata, riscontrando che è di molto superiore alla metà del numero totale di requisiti appartenenti alla categoria presa in considerazione. Quindi abbiamo molti requisiti di sicurezza senza alcuna keyword associata.

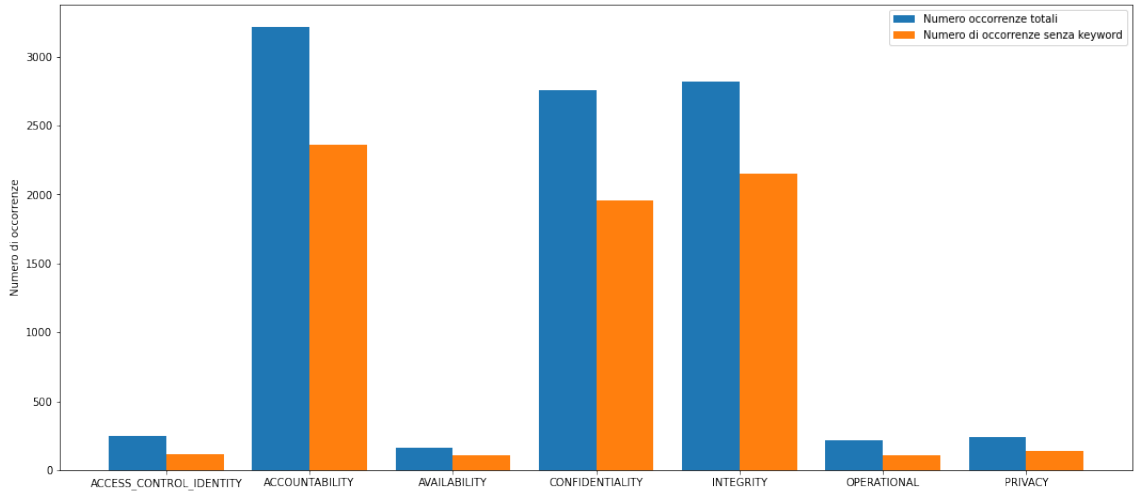

Figura 3.6: In arancione le categorie senza keywords associate, in blu il numero totale di requisiti per ogni categoria.

Inoltre analizzando le frequenze più alte delle parole chiave per ogni categoria di sicurezza, la più alta registrata ammonta al 17% circa (Tabella 3.2) mentre il resto ha dei valori molto bassi.

Possiamo ipotizzare che, siccome la maggior parte dei requisiti di sicurezza non presenta parole chiave associate e in generale la frequenza di queste nelle varie categorie è in media molto bassa, le keywords non aggiungono conoscenza significativa ai modelli di machine learning e che se esistesse una correlazione tra le parole chiave e le categorie sarebbe comunque trascurabile.

Categoria	Keyword	Frequenza (%)
ACCESS_CONTROL_IDENTITY	user	17,2%
	authentication	14,0%
	access	8,4%
	identity	6,8%
	password	3,6%
ACCOUNTABILITY	user	5,8%
	access	5,3%
	client	3,3%
	response	2,4%
AVAILABILITY	access	11,3%
	availability	7,5%
	user	4,4%
	server	3,7%
	integrity	3,1%
CONFIDENTIALITY	access	7,0%
	user	6,0%
	client	3,1%
	response	2,6%
INTEGRITY	user	4,7%
	client	3,4%
	access	2,6%
	response	2,4%
OPERATIONAL	access	11,1%
	user	9,7%
	privacy	8,8%
	software	3,7%
	confidentiality	3,7%
PRIVACY	privacy	17,2%
	access	10,9%
	authorization	5,4%
	client	4,6%
	user	3,3%

Tabella 3.2: Frequenza delle keywords per ogni categoria

3.3.5 Selezione e codifica delle features

In questa fase è stato deciso quali features prendere in cosiderazione, che nel nostro caso sono la lista di Entities, Dependencies e PoS e tag come variabili dipendenti:

- Il campo "Security" per il classificatore binario

- Il campo "Categories" per il classificatore multi-label

Prima di passare alla predizione di Security e di Categories è necessario codificare le features presenti nel dataset. A tale scopo,sono stati analizzati due approcci: Vettorizzazione e trasformazione delle features tramite *TfidfVectorizer*, e codifica delle features in numeri interi tramite un *Tokenizer pre-allenato*[5].

Vettorizzazione tramite TfidfVectorizer

Tramite *TfidfVectorizer* i dati vengono convertiti in una matrice di feature TF-IDF (Term Frequency-Inverse Document Frequency). Il processo consiste nel calcolare il TF-IDF score per ogni parola che successivamente viene messo in un vettore. Sia:

- t il termine di cui sta calcolando il TF-IDF score
- d un record del dataset, nel nostro caso un requisito
- D l'insieme di tutte le frasi.

Per calcolare il TF-IDF score, si calcola dapprima TF ovvero la frequenza dei termini nel testo.

$$TF(t,d) = \frac{f_{t,d}}{|d|}$$

dove $f_{t,d}$ indica il numero di occorrenze del termine t nella frase d. Successivamente calcoliamo la IDF (Inverse Document Frequency) un valore che associa un peso ad ogni parola, indicando quanto spesso appare nel testo, più una parola è frequente più il suo peso sarà basso, e di conseguenza la sua importanza sarà minore:

$$IDF(t) = \log \frac{|D|}{1 + |d : t \in d|}$$

dove $|D|$ corrisponde alla cardinalità del dataset e $1 + |d : t \in d|$ è il numero di record in cui appare il termine t. Possiamo quindi calcolare il valore TF-IDF nel seguente modo:

$$TFIDF(t,d) = TF(t,d) \times IDF(t)$$

[5] `https://huggingface.co/docs/transformers/main_classes/tokenizer`

Questa tecnica è molto efficace quando si hanno delle features in forma testuale ma ha degli svantaggi: siccome il vettore di termini viene costruito in base alle parole e alla loro frequenza nel dataset, i modelli allenati con questo approccio non funzioneranno bene se usati in un contesto applicativo differente da quello in cui sono stati allenati (healthcare nel nostro caso), in quanto termini mai apparsi prima non troverebbero una corrispondenza.

Codifica in interi tramite Tokenizer

Con questo approccio le features vengono codificate in una sequenza di numeri interi che rappresentano il testo. Per effettuare questa codifica è stato usato un Tokenizer con un transformer pre-allenato: *BERT cased*[6].

```
#caricamento del tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
model = AutoModel.from_pretrained("bert-base-cased")



#esempio di encoding dei token
sentence = "The system shall provide a means to edit discharge instructions for a
                                          particular patient."
encoded = tokenizer(sentence)['input_ids']
print(encoded)

#Output:
#[101,1109,1449,4103,2194,170,2086,1106,14609,12398,7953,1111,170,2440,119,102]
```

Esempio di encoding di un requisito con Tokenizer

3.4 Scelta dei modelli di machine learning

Dopo la fasi di pulizia, processing e selezione dei dati si è passati alla fase di implementazione, nella quale sono stati progettati vari modelli di machine learning per la predizione delle variabili indipendenti.

Gli algoritmi di machine learning scelti per questo lavoro sono sei:

- Random Forest Classifier
- Logistic Regression
- k-Nearest Neighbors Classifier
- LinearSVC
- Gaussian Naive Bayes
- Decision Tree Classifier

[6] https://huggingface.co/bert-base-cased

Per ogni modello è stata effettuata una fase preliminare di "tuning" degli iperparametri, utilizzando *RandomizedSearchCV*, una libreria di Scikit Learn[7] che permette di cercare randomicamente la combinazione di parametri ottimale tra un insieme di possibili valori preimpostati.

3.4.1 Random Forest Classifier

Quest'algoritmo, esegue *n* volte un altro modello di machine learning: l'algoritmo Decision Tree Classifier. È un tipo algoritmo di insieme, ogni albero decisionale è creato in modo autonomo ed effettua le sue predizioni. I risultati finali ottenuti sono una media di quelle effettuate dai singoli alberi.

Iperparametri con Tokenizer

```
Config: {'n_estimators': 19, 'min_samples_split': 2, 'min_samples_leaf': 1,
'max_features': 'sqrt', 'max_depth': 40, 'bootstrap': False}
```

Iperparametri con TfidfVectorizer

```
Config: {'n_estimators': 20, 'min_samples_split': 10, 'min_samples_leaf': 1,
'max_features': 'sqrt', 'max_depth': 70, 'bootstrap': False}
```

3.4.2 Logistic Regression

La logistic regression è un modello di regressione utilizzato quando la variabile indipendente è di tipo binaria. Il modello stima la probabilità del verificarsi di un evento in base alla correlazione che hanno le variabili dipendenti con la variabile indipendente. Poiché il risultato è una probabilità, la variabile dipendente è vincolata tra 0 e 1.

In questo algoritmo le probabilità vengono trasformate in probabilità logaritmiche, ossia , la probabilità di successo divisa per la probabilità di fallimento.

Iperparametri con Tokenizer

```
Config: {'solver': 'saga', 'penalty': 'l1', 'max_iter': 100, 'C': 100}
```

Iperparametri con TfidfVectorizer

```
Config: {'solver': 'newton-cg', 'penalty': 'l2', 'max_iter': 500, 'C': 10}
```

[7] https://scikit-learn.org/stable/

3.4.3 Gaussian Naive Bayes

L'algoritmo Gaussian Naive Bayes prevede il valore della variabile indipendente calcolando la probabilità che le features facciano parte di una determinata classe usando il teorema di Bayes. Successivamente la classe con la probabilità maggiore viene fornita come output. L'algoritmo è considerato *naive* (ingenuo) perché si basa sulla forte assunzione che le variabili dipendenti non sono correlate tra di loro. In particolare questa versione dell'algoritmo fa un'ulteriore assunzione: i valori associati ad ogni classe sono distribuiti seguendo la distribuzione Gaussiana.

Iperparametri con Tokenizer

```
Config: {'var_smoothing': 1.0}
```

Iperparametri con TfidfVectorizer

```
Config: {'var_smoothing': 0.01}
```

3.4.4 LinearSVC

SVM o Support Vector Machine [18] è un modello lineare per problemi di classificazione e regressione. Può risolvere problemi lineari e non lineari e funziona bene per molti problemi pratici. L'idea di SVM è semplice: L'algoritmo crea una linea o un iperpiano che separa i dati in classi. Questa linea è la retta di separazione delle classi che massimizza il margine tra le classi stesse, dove con margine si intende la distanza minima.

Iperparametri con Tokenizer

```
Config: {'penalty': 'l2', 'loss': 'squared_hinge', 'fit_intercept': True,
'dual': False, 'C': 0.1}
```

Iperparametri con TfidfVectorizer

```
Config: {'penalty': 'l2', 'loss': 'hinge', 'fit_intercept': True,
'dual': True, 'C': 1}
```

3.4.5 k-Nearest Neighbors Classifier

L'algoritmo k-nearest neighbors [19], noto anche come KNN o k-NN, è un classificatore che utilizza la prossimità per effettuare classificazioni o previsioni sul raggruppamento di

un singolo punto di dati. Sebbene possa essere utilizzato sia per problemi di regressione che di classificazione, è tipicamente usato come algoritmo di classificazione, partendo dal presupposto che punti simili possono essere trovati vicini tra loro.

Per i problemi di classificazione, l'etichetta della classe viene assegnata sulla base di un voto di maggioranza, cioè viene utilizzata l'etichetta più frequentemente rappresentata intorno a un determinato punto di dati.

Iperparametri con Tokenizer

```
Config: {'weights': 'distance', 'p': 1, 'n_neighbors': 10,
'metric': 'minkowski', 'leaf_size': 40}
```

Iperparametri con TfidfVectorizer

```
Config: {'weights': 'distance', 'p': 2, 'n_neighbors': 1,
'metric': 'minkowski', 'leaf_size': 1}
```

3.4.6 Decision Tree Classifier

Decision Tree Classifier è un algoritmo che utilizza un insieme di regole per prendere decisioni, in modo simile a quello degli esseri umani. L'intuizione alla base dell'algoritmo consiste nell'utilizzare le features del dataset per creare nodi di una struttura ad albero, dividendo continuamente il dataset fino a isolare tutti i punti di dati appartenenti a ciascuna classe. L'obiettivo è continuare a suddividere lo spazio delle features e ad applicare regole fino a quando non ci sono più regole da applicare o non ci sono più datapoint. A quel punto, è il momento di assegnare una classe a tutti i nodi foglia.

Iperparametri con Tokenizer

```
Config: {'max_features': 'log2', 'max_depth': 8, 'criterion': 'gini',
'ccp_alpha': 0.001}
```

Iperparametri con TfidfVectorizer

```
Config: {'max_features': 'sqrt', 'max_depth': 7, 'criterion': 'entropy',
'ccp_alpha': 0.001}
```

3.5 Tecniche di classificazione multi-label

Per effettuare la classificazione dei vari requisiti di sicurezza è necessario sviluppare un classificatore multi-label, poiché ogni requisito può appartenere a più categorie simultaneamente. Affinché i classici modelli di machine learning possano essere usati in questa situazione è necessario modellare il problema. Sono state usate tre tecniche:

- Binary Relevance
- Classifier Chain
- Label Powerset

3.5.1 Binary Relevance

Binary Relevance [20] è probabilmente la soluzione più intuitiva per i problemi multi-label. Funziona scomponendo il problema in una serie di sotto problemi di classificazione binaria indipendenti (uno per etichetta). Il problema di questa tecnica è che ignora l'eventuale correlazione tra le etichette.

X	Class1	Class2	Class3
X_1	0	0	1
X_2	0	0	0
X_3	1	0	1

X	Class1
X_1	0
X_2	0
X_3	1

X	Class2
X_1	0
X_2	0
X_3	0

X	Class 3
X_1	1
X_2	0
X_3	1

Tabella 3.3: Tratta ogni etichetta come un singolo problema di classificazione separato. La correlazione tra le etichette potrebbe perdersi.

3.5.2 Classifier Chain

In questa tecnica [21], abbiamo più classificatori collegati in una catena. Il primo classificatore viene costruito utilizzando le features. I classificatori successivi vengono addestrati utilizzando le features combinate alla variabile indipendente usata nei classificatori precedenti di una determinata catena. Si tratta di un processo sequenziale in cui l'output di un classificatore viene utilizzato come input del classificatore successivo della catena. In questo modo la correlazione tra le variabili viene preservata.

X	Class1	Class2	Class3
X_1	0	0	1
X_2	0	0	0
X_3	1	0	1

X	Class
X_1	0
X_2	0
X_3	1

X	Y_1	Class
X_1	0	0
X_2	0	0
X_3	1	0

X	Y_1	Y_2	Class
X_1	0	0	1
X_2	0	0	0
X_3	1	0	1

Tabella 3.4: Vengono creati classificatori in catena aggiungendo come caratteristica le etichette già usate. Preserva la correlazione tra le etichette.

3.5.3 Label Powerset

Con questa tecnica si trasforma il problema multilabel in un problema multiclasse. Vengono assegnati dei valori unici a tutte le possibili combinazioni delle etichette.

X	Class1	Class2	Class3
X_1	0	0	1
X_2	0	0	0
X_3	1	0	1

X	Class
X_1	1
X_2	2
X_3	3

Tabella 3.5: Trasforma il problema in un problema multiclasse, assegnando un valore unico per ogni combinazione delle etichette.

3.6 Tecniche di validazione

Per valutare le performance dei due classificatori, sono state usate come tecniche di validazione la *stratified k-fold cross-validation* per il classificatore binario e la *k-fold validation* per il classificatore multi-label, in quanto la versione stratified non è supportata in questo caso.

La cross fold validation [22] è una metrica di validazione usata nel machine learning per scoprire quanto il modello sia in grado di prevedere il risultato di dati non visti. Si tratta di un metodo di facile comprensione, che funziona bene per un campione di dati limitato e che offre una valutazione meno distorta, il che lo rende una scelta popolare. Il campione di dati viene suddiviso in un numero "k" di campioni più piccoli, da cui il nome: K-fold Cross Validation.

La differenza sostanziale della versione *stratified* è che questa versione garantisce che in ogni suddivisione si abbia una percentuale precisa di campioni per ogni classe, ideale per dataser sbilanciati. Nel nostro caso abbiamo scelto un numero di fold pari a 5.

3.7 Metriche di valutazione

Per valutare le prestazioni dei modelli di machine learning si usano solitamente le metriche di *precision, recall e F-measure* (o *F1-score*). Per stabilire questi valori è necessario innanzitutto dividere le predizioni del modello in quattro categorie (Figura 3.7):

- **True Positive (TP):** sono le previsioni positive corrette.
- **False Positive (FP):** sono le previsioni positive errate
- **True Negative (TN):** sono le previsioni negative corrette
- **False Negative (FN):** sono le previsioni negative errate.

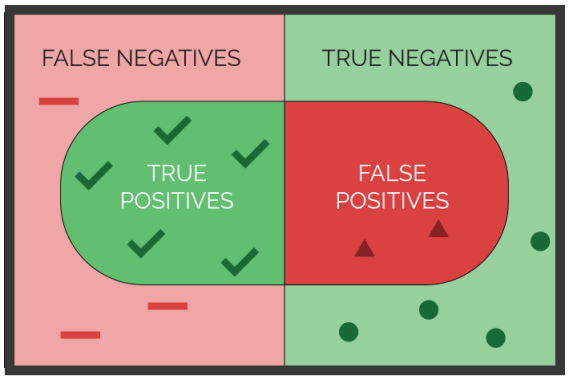

Figura 3.7: Dati necessari al calcolo della precision e recall.

Avendo questi valori, possiamo definire la precision come il rapporto tra il numero delle previsioni corrette (true positive) di una classe sul totale delle volte che il modello lo prevede:

$$Precision, P = \frac{TP}{TP + FP}$$

Definiamo la recall come il rapporto tra le previsioni corrette per una classe sul totale dei casi in cui si verifica effettivamente:

$$Recall, R = \frac{TP}{TP + FN}$$

L'accuratezza è definita come il rapporto tra le classificazioni corrette e il totale delle previsioni effettuate dal modello.

$$Accuracy, A = \frac{TP + TN}{TP + TN + FP + FN}$$

L'F-measure (o F_1) viene calcolato tramite la media armonica di precision e recall:

$$F_1 = 2 \times \frac{P \times R}{P + R}$$

Nel nostro caso il recall assume maggiore importanza rispetto alla precision in quanto vogliamo estrarre tutti i NFR, ma anche la precision non è da trascurare poiché una precisione molto bassa causa un grande numero di falsi positivi che potrebbero frustare gli utenti [10].

3.7.1 Metriche nell classificazione Multi-label

Nei problemi di classificazione multi-label, ossia nei problemi in cui ci sono più di una etichetta e le varie classi non sono mutualmente esclusive (ogni elemento può appartenere a più classi). Di conseguenza l'output del modello potrà essere anche *parzialmente corretto* e non solo *corretto* o *errato*. Considerando quanto detto le metriche menzionate sopra possono essere calcolate effettuando dei piccoli accorgimenti.

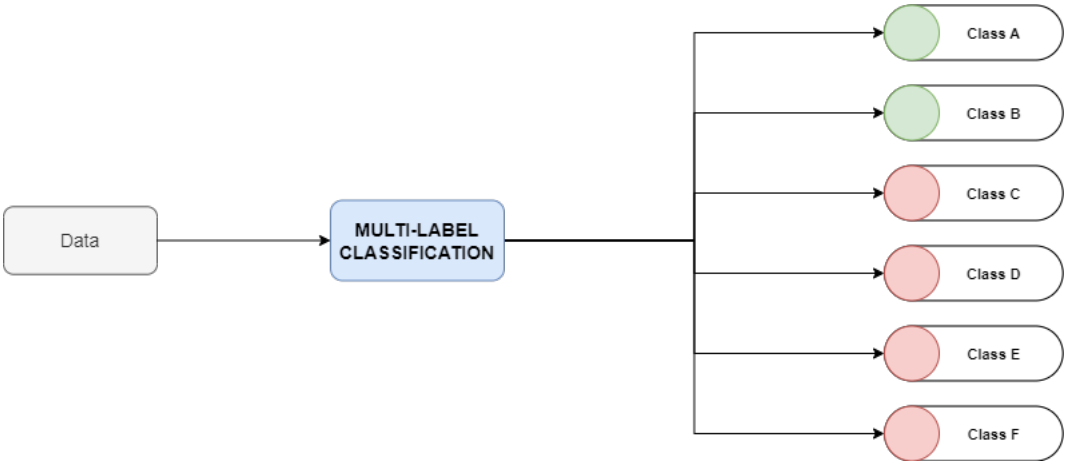

Figura 3.8: Esempio di classificazione multi-label. In verde le classi predette dal modello.

Siano Y_i e Z_i rispettivamente l'insieme di etichette reali e predette dal modello per l'i-esima istanza del dataset e N la lunghezza del dataset.

$$Accuracy, HammingScore = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

$$Precision, P = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{Z_i}$$

$$Recall, R = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_i \cap Z_i|}{Y_i}$$

$$F1 = \frac{1}{N}\sum_{i=1}^{N}\frac{2|Y_i \cap Z_i|}{|Y_i| + |Z_i|}$$

$$HammingLoss = \frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{N} I[Y_j \neq Z_j]$$

Dove L è la lunghezza dell'etichetta e I è pari al numero di valori di etichetta diversi tra quella reale e predetta.

3.7.2 Matrice di confusione

Le matrici di confusione sono una misura delle prestazioni per i problemi di classificazione del machine learning in cui l'output può essere costituito da due o più classi. Si tratta di una tabella con 4 diverse combinazioni di valori previsti e reali. La matrice presenta sulle righe la classe che si aspettava come predizione e sulle colonne la classe che è stata predetta, quindi osservando la diagonale principale troveremo il numero di istanze predette correttamente.

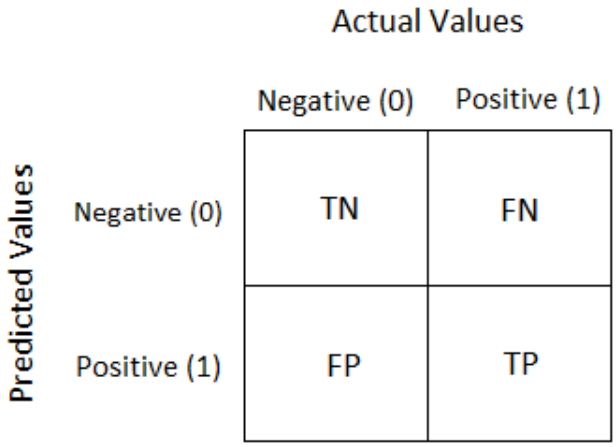

Figura 3.9: Esempio di matrice di confusione.

CAPITOLO 4

Risultati

4.1 Risultati classificatore binario

Analizzando i risultati ottenuti e ricordando che per il nostro problema vogliamo massimizzare la recall (mantenendo comunque una precision accettabile) possiamo constatare che nel primo approccio il modello migliore risulta essere ***LogisticRegression***, ottenendo una recall del 90%, una precision del 71%, e un F1-score del 79%. Nel secondo approccio invece il modello migliore è ***LinearSVC*** con il 91% di recall, 86% di precision e 89% di F1-score.

	TOKENIZER				TFIDF			
Classifer	**Accuracy**	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
RandomForest	72,5%	74,6%	89,7%	81,5%	80,5%	81,3%	92,3%	86,4%
LogisticRegression	**69,3%**	**71,5%**	**90,5%**	**79,9%**	84,8%	87,4%	90,5%	88,9%
GaussianNB	59,0%	76,4%	56,8%	65,1%	81,7%	82,9%	91,6%	81,7%
LinearSVC	69,2%	71,6%	90,2%	79,8%	**85,0%**	**86,8%**	**91,8%**	**89,2%**
k-NearestNeighbors	70,7%	73,2%	89,2%	80,4%	79,4%	81,1%	90,5%	85,5%
DecsionTree	70,3%	72,8%	90,1%	80,2%	68,3%	69,2%	95,7%	80,3%

Tabella 4.1: Risultati classificatore binario.

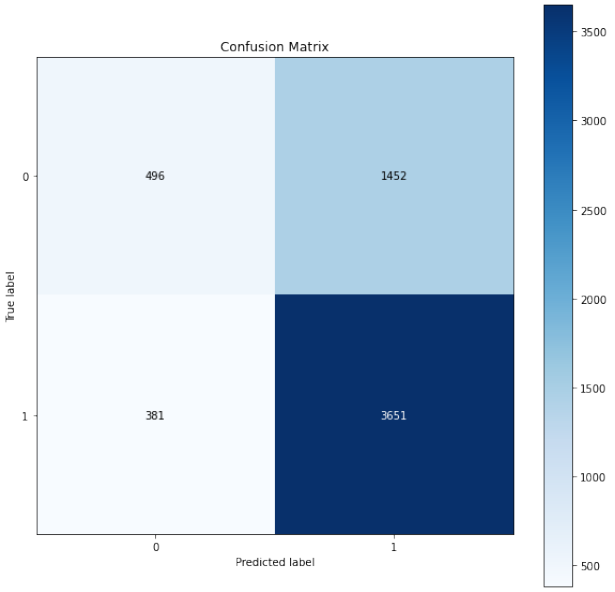

Figura 4.1: Matrice di confusione LogisticRegression nella classificazione binaria con Tokenizer.

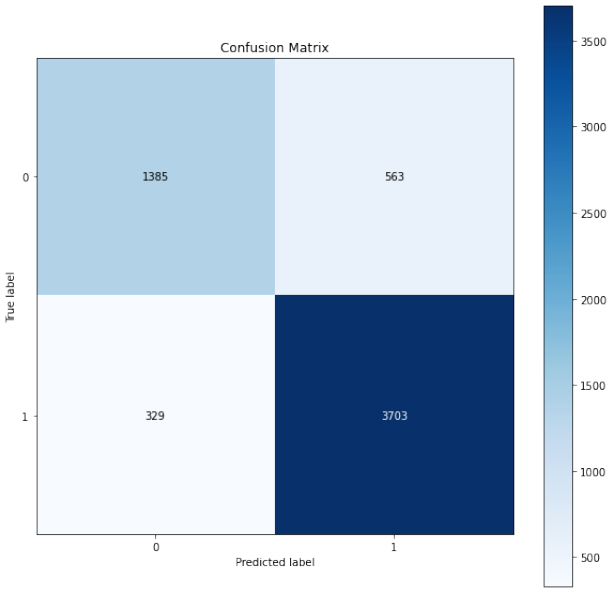

Figura 4.2: Matrice di confusione LinearSVC nella classificazione binaria con TfidfVectorizer.

Come si evince dalle figure 4.1 e 4.2 i modelli con le prestazioni migliori hanno classificato:

- **Tokenizer:** 5103 requisiti di sicurezza (di cui 3651 effettivi e 1452 predetti come falsi positivi);
- **TfidfVectorizer:** 4266 requisiti di sicurezza (di cui 3703 effettivi e 563 predetti come falsi positivi).

Il numero requisiti di sicurezza effettivi classificati dai modelli non è molto distante dal numero totale dei requisiti di sicurezza presenti nel dataset (4032), ciò rispecchia i valori di recall elevati che i modelli hanno ottenuto. Notiamo che usando Tokenizer, riscontriamo un numero notevole di falsi positivi, questo accade perché abbiamo una precision inferiore di circa il 15% rispetto all'approccio con TfidfVectorizer che presenta un numero di falsi positivi minore.

4.1.1 Risultati classificatore multi-label

Per quanto riguarda il secondo classificatore usando Tokenizer il modello migliore risulta essere ***DecisionTreeClassifier*** con la tecnica ***LabelPowerset***, ottenendo una recall del 90%, una precision del 72%, e un F1-score del 80%. Nel secondo approccio invece il modello migliore è ***RandomForestClassifier*** con la tecnica ***ClassifierChain*** ottenendo il 90% di recall, 75% di precision e 84% di F1-score.

4.1.2 BinaryRelevance

	TOKENIZER					TFIDF				
Classifer	**HS**	**Precision**	**Recall**	**F1**	**HL**	**HS**	**Precision**	**Recall**	**F1**	**HL**
RandomForest	69,3%	75,6%	85,6%	80,3%	14,3%	74,6%	80,6%	89,4%	84,4%	11,2%
LogisticRegression	68,4%	73,2%	87,0%	79,5%	15,3%	76,3%	85,6%	87,0%	86,3%	9,4%
GaussianNB	59,6%	62,5%	76,8%	68,9%	23,6%	54,1%	53,8%	82,3%	65,1%	30,1%
LinearSVC	68,3%	73,0%	86,9%	79,3%	15,4%	76,2%	84,1%	88,5%	86,2%	9,6%
k-NearestNeighbors	67,1%	74,6%	83,5%	78,8%	15,3%	74,0%	81,6%	84,4%	83,0%	11,7%
DecsionTree	68,6%	73,4%	87,0%	79,6%	15,1%	70,3%	74,0%	89,3%	80,9%	14,3%

Tabella 4.2: Risultati classificatore multilabel con Binary Relevance (HS indica l'Hamming-Score, mentre HL indica l'Hamming-Loss).

4.1.3 ClassifierChain

	TOKENIZER					TFIDF				
Classifer	**HS**	**Precision**	**Recall**	**F1**	**HL**	**HS**	**Precision**	**Recall**	**F1**	**HL**
RandomForest	70,0%	75,0%	87,1%	80,6%	14,2%	75,2%	79,9%	90,0%	84,6%	11,1%
LogisticRegression	68,0%	73,1%	86,4%	79,2%	15,4%	77,0%	84,1%	88,3%	86,1%	9,6%
GaussianNB	59,3%	62,4%	76,4%	68,7%	23,7%	50,5%	55,0%	75,7%	63,7%	29,4%
LinearSVC	68,2%	72,9%	86,7%	79,2%	15,5%	77,0%	82,4%	89,8%	85,9%	10,0%
k-NearestNeighbors	67,3%	74,6%	83,7%	78,9%	15,2%	74,4%	81,8%	84,9%	83,3%	11,5%
DecsionTree	68,9%	73,6%	87,2%	79,8%	15,0%	70,5%	73,8%	89,8%	81,0%	14,3%

Tabella 4.3: Risultati classificatore multilabel con Binary Relevance (HS indica l'Hamming-Score, mentre HL indica l'Hamming-Loss).

4.1.4 LabelPowerSet

	TOKENIZER					TFIDF				
Classifer	**HS**	**Precision**	**Recall**	**F1**	**HL**	**HS**	**Precision**	**Recall**	**F1**	**HL**
RandomForest	70,0%	75,1%	86,6%	80,4%	14,3%	74,8%	77,7%	90,3%	83,5%	12,1%
LogisticRegression	68,1%	72,7%	86,8%	79,1%	15,6%	77,8%	83,6%	88,6%	86,0%	9,8%
GaussianNB	59,7%	62,5%	77,0%	69,0%	23,6%	50,9%	55,2%	76,2%	64,0%	29,2%
LinearSVC	60,7%	67,6%	79,1%	72,9%	20,0%	77,3%	82,6%	88,9%	85,6%	10,1%
k-NearestNeighbors	68,4%	73,6%	85,7%	79,2%	15,3%	74,1%	81,7%	84,6%	83,2%	11,6%
DecsionTree	70,4%	72,9%	90,4%	80,7%	14,7%	69,9%	73,2%	89,4%	80,5%	14,7%

Tabella 4.4: Risultati classificatore multilabel con LabelPowerset (HS indica l'Hamming-Score, mentre HL indica l'Hamming-Loss).

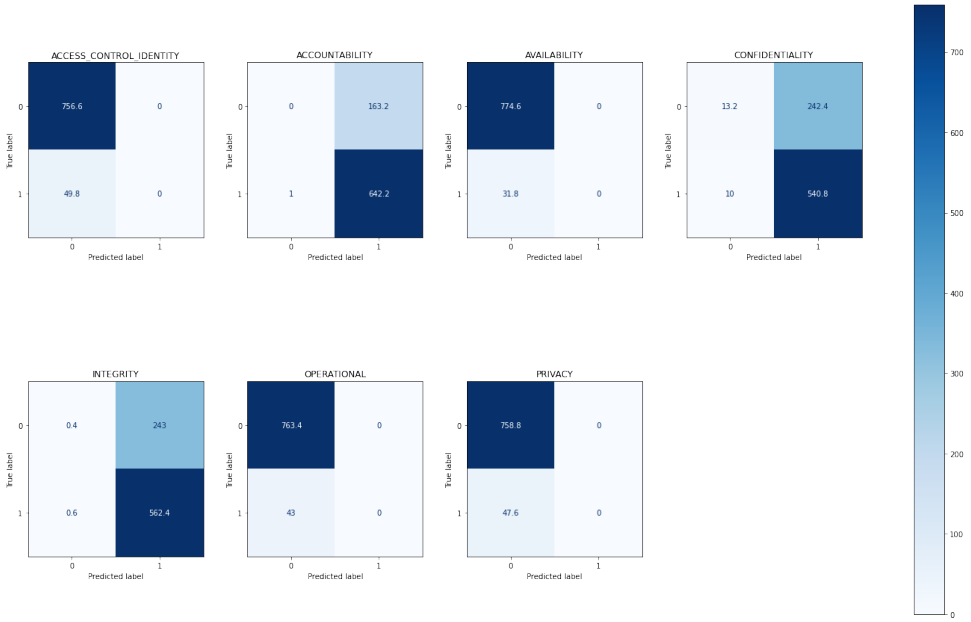

Figura 4.3: Matrice di confusione DecisionTree nella classificazione multi-label con LabelPowerset e Tokenizer

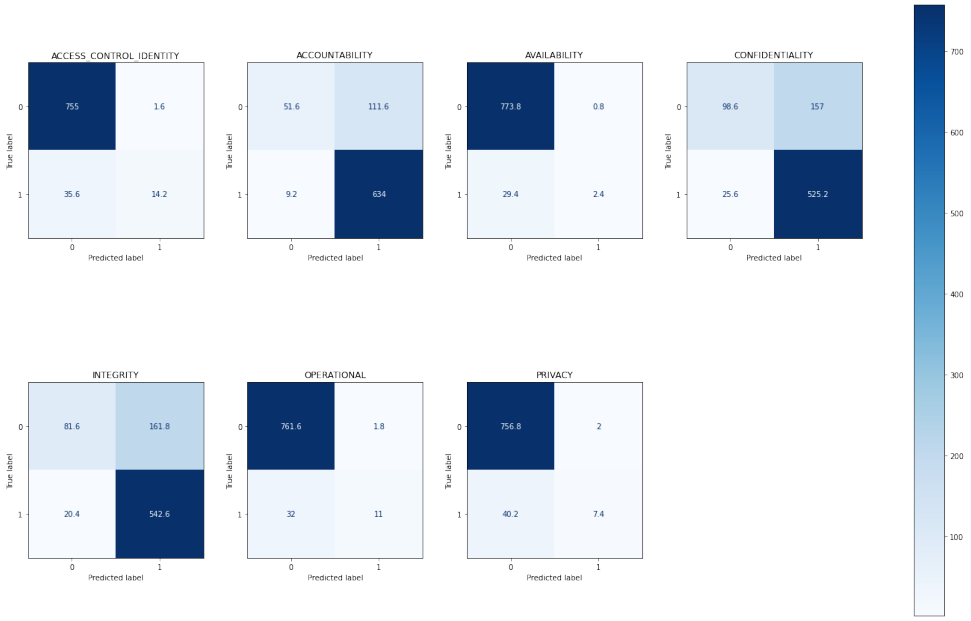

Figura 4.4: Matrice di confusione RandomForest nella classificazione multi-label con ClassifierChain e TfidfVectorizer

Osservando le matrici di confusione delle singole categorie, notiamo che alcune di esse (ad esempio *AVAILABILITY*) presentano un numero esiguo di veri positivi, il motivo di questo fenomeno è che il numero di categorie nel dataset non è bilanciato (Figura 4.5): molte categorie hanno poche occorrenze rispetto al numero totale dei requisiti di sicurezza, ed è per questo che ci troviamo in casi dove il numero di veri negativi è molto maggiore rispetto al numero di veri positivi.

Come ci aspettavamo i modelli con TfidfVectorizer hanno ottenuto le performance migliori, ma come è stato detto in precedenza, questo approccio avrà delle buone performance solo se i nuovi dati da predirre appartengono al contesto applicativo dell'healthcare.

Con l'encoding tramite Tokenizer invece, le prestazioni risultano inferiori (soprattutto per quanto riguarda la precision) ma non dipenderanno dal dominio applicativo.

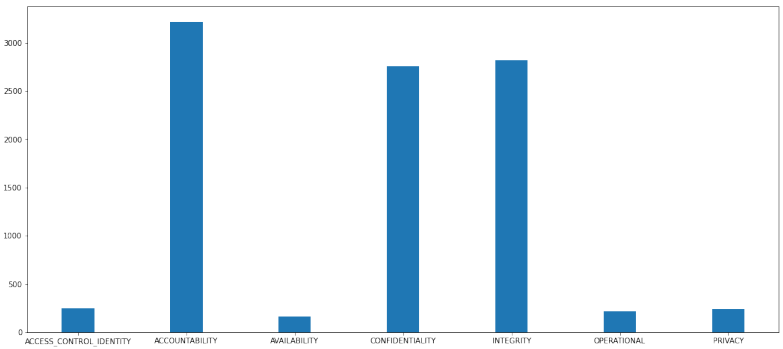

Figura 4.5: Numero di occorrenze di categorie nel dataset

CAPITOLO 5

Conclusioni e sviluppi futuri

Gli attacchi informatici che minano la sicurezza delle risorse e informazioni sensibili sono molto diffusi al giorno d'oggi. È imperativo progettare sistemi sicuri e robusti per evitare che ciò accada. Per costruire un software sicuro è necessario identificare e classificare i requisiti di sicurezza in modo appropriato durante le fasi preliminari dello sviluppo del prodotto. A tal proposito il lavoro svolto in questa tesi descrive lo sviluppo di un tool per l'identificazione e classificazione automatica dei requisiti non funzionali di sicurezza. Sono stati utilizzati sei documenti di software relativi all'healthcare opportunamente processati e sono state provate due tecniche di codifica delle caratteristiche, ognuno con i propri vantaggi e svantaggi. Per il riconoscimento e la classificazione sono stati impiegati diversi modelli di machine learning, ottenendo una precision che varia dal 70% all'80% a seconda dell'approccio di codifica utilizzato e una recall del 90% circa.

Lo strumento implementato seppur non perfetto, visto il numero di falsi positivi non trascurabili e una piccola percentuale di requisiti non correttamente identificati, non può certamente sostituire l'analisi umana, ma costituisce comunque un importante supporto e un punto di partenza su cui gli ingegneri del software potranno basarsi per la fase di ingegneria dei requisiti.

Possibili miglioramenti futuri a questo progetto potrebbero essere:

- L'utilizzo di un dataset appartenente ad un dominio applicativo più generico e con un numero di istanze per ogni categoria bilanciato.

- Utilizzare un NER (Named Entity Recognition) sviluppato ad hoc per questo contesto applicativo, migliorando la fase di pre processing NLP.
- Provare ad utilizzare modelli di Deep Learning per incrementare le prestazioni.
- Sviluppare strumenti analoghi per l'identificazione di altri requisiti non funzionali.

Il codice sorgente del progetto è disponibile al seguente link:
`https://github.com/xDaryamo/NFR-Security-Extraction-Classification`

Ringraziamenti

Ci tengo a ringraziare tutte le persone che mi hanno sostenuto ed accompagnato in questo percorso durato tre anni.
Inizio ringraziando il professore Fabio Palomba per la sua grande disponibilità e professionalità, grazie ai suoi insegnamenti mi sono appassionato ulteriormente a questo campo di studi. Un sentito grazie anche al dott. Francesco Casillo, secondo relatore, e alla dott.ssa Valeria Pontillo per i preziosi consigli e attenzioni che mi hanno fornito durante lo svolgimento di questa tesi.

Un infinito grazie va ai miei genitori per avermi sempre sostenuto, spronato e per tutti i sacrifici che hanno fatto per non farmi mancare mai niente.

Ringrazio i miei nonni, i miei zii, i miei cugini e tutta la mia famiglia in generale, per essermi stati sempre vicini nel momento del bisogno.

Ringrazio Daniele, Giuseppe, Ciro, Emmanuel e tutti gli amici per essere stati presenti nei momenti più difficili, per avermi sempre ascoltato e per aver provato sempre a tirarmi su di morale, soprattutto negli ultimi periodi.

Ringrazio Tommaso, Nicolò, Aurelio, Daniele, Simone, Leonardo e tutti i miei amici conosciuti all'università che hanno affrontato insieme a me questo percorso, e che mi hanno sempre aiutato senza esitare.

Bibliografia

[1] Omar Elgabry, "Requirements engineering — introduction (part 1)," 2016. https://medium.com/omarelgabrys-blog/requirements-engineering-introduction-part-1-6d49001526d3. (Citato a pagina 1)

[2] I. Attarzadeh and S. H. Ow, "Project management practices: Success versus failure," in *2008 International Symposium on Information Technology*, vol. 1, pp. 1–8, IEEE, 2008. (Citato a pagina 2)

[3] M. Younas, D. N. Jawawi, I. Ghani, and M. A. Shah, "Extraction of non-functional requirement using semantic similarity distance," *Neural Computing and Applications*, vol. 32, no. 11, pp. 7383–7397, 2020. (Citato a pagina 4)

[4] J. Doerr, D. Kerkow, T. Koenig, T. Olsson, and T. Suzuki, "Non-functional requirements in industry-three case studies adopting an experience-based nfr method," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, pp. 373–382, IEEE, 2005. (Citato a pagina 4)

[5] H. M. Abd El, N. A. abd el Azim, and N. Ramadan, "Challenges of non-functional requirements extraction in agile software development using machine learning," *International Journal of Computer Applications*, vol. 975, p. 8887. (Citato a pagina 4)

[6] R. U. Khan and M. Khan, "A review on automatic extraction and classification of non-functional requirements," *International Journal of Advanced and Applied Sciences*, vol. 4, no. 6, pp. 35–42, 2017. (Citato a pagina 5)

[7] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 39–48, IEEE, 2006. (Citato a pagina 5)

[8] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010. (Citato a pagina 5)

[9] W. Zhang, Y. Yang, Q. Wang, and F. Shu, "An empirical study on classification of non-functional requirements," in *The twenty-third international conference on software engineering and knowledge engineering (SEKE 2011)*, pp. 190–195, 2011. (Citato a pagina 6)

[10] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*, pp. 9–16, IEEE, 2013. (Citato alle pagine 6 e 25)

[11] A. Mahmoud and G. Williams, "Detecting, classifying, and tracing non-functional software requirements," *Requirements Engineering*, vol. 21, no. 3, pp. 357–381, 2016. (Citato a pagina 6)

[12] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting requirements engineers in recognising security issues," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 4–18, Springer, 2011. (Citato a pagina 6)

[13] T. Li, "Identifying security requirements based on linguistic analysis and machine learning," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 388–397, IEEE, 2017. (Citato a pagina 6)

[14] D. N. Palacio, D. McCrystal, K. Moran, C. Bernal-Cárdenas, D. Poshyvanyk, and C. Shenefiel, "Learning to identify security-related issues using convolutional neural networks," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 140–144, 2019. (Citato a pagina 7)

[15] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *2014 IEEE 22nd international requirements engineering conference (RE)*, pp. 183–192, IEEE, 2014. (Citato alle pagine 7 e 9)

[16] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2027–2033, 2016. (Citato a pagina 7)

[17] V. Varenov and A. Gabdrahmanov, "Security requirements classification into groups using nlp transformers," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 444–450, IEEE, 2021. (Citato alle pagine 7 e 14)

[18] Rushikesh Pupale, "Support vector machines(svm) — an overview," 2018. `https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989`. (Citato a pagina 21)

[19] IBM, "What is the k-nearest neighbors algorithm?." `https://www.ibm.com/topics/knn`. (Citato a pagina 21)

[20] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: an overview," *Frontiers of Computer Science*, vol. 12, no. 2, pp. 191–202, 2018. (Citato a pagina 23)

[21] Charles Kariuki, "Multi-label classification with scikit-multilearn." `https://www.section.io/engineering-education/multi-label-classification-with-scikit-multilearn/`. (Citato a pagina 23)

[22] Praveen Nellihela, "What is k-fold cross validation?," 2022. `https://towardsdatascience.com/what-is-k-fold-cross-validation-5a7bb241d82f`. (Citato a pagina 24)