UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Applicazione di tecniche di NLP e QNLP per la classificazione dei requisiti di sicurezza.

RELATORE

Prof. **Fabio Palomba**

Dott. **Francesco Casillo**

Università degli Studi di Salerno

CANDIDATO

Adriano Emanuele Califano

Matricola: 0512110212

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

"In un mondo dominato dalla forza, è la volontà umana a spingere le persone a compiere azioni incredibili."

Abstract

L'identificazione e la raccolta dei requisiti di un sistema software è di fondamentale importanza per la riuscita del sistema stesso e della sua durata nel tempo. Rivestono una particolare importanza i requisiti non funzionali (NFRs) che esprimono le qualità di un sistema in termini di efficienza, sicurezza, consistenza, e così via. La loro identificazione rappresenta quindi un grande aiuto per sviluppare del software di qualità. Tra le varie categorie esistenti di NFRs, quella dei requisiti di sicurezza è senza dubbio la categoria cui prestare maggiormente attenzione perchè da qui si misura la robustezza e sicurezza di un sistema. Ad oggi i tool e le ricerche effettuate si sono incentrate principalmente sulla teoria classica che ha dato risultati soddisfacenti e che è in costante miglioramento. D'altro canto l'ascesa recente del quantum computing ha portato nuovi stimoli ai ricercatori e quindi le nuove ricerche si stanno spostando in parte su questo nuovo campo. La presente tesi è incentrata sullo studio e sperimentazione di tecniche di Quantum Natural Language Processing (QNLP) tramite la libreria Lambeq, che è stata studiata e testata nelle sue funzionalità per poi essere utilizzata per la classificazione binaria di requisiti non funzionali di sicurezza. I risultati ottenuti hanno mostrato valori nella media riguardo ai modelli che seguono un approccio classico, in particolare le metriche nella combinazione migliore di iperparametri utilizzati hanno registrato valori tra il 50% e il 55%. Leggermente diversi sono stati i risultati dei modelli che seguono la teoria quantistica, le cui metriche hanno registrato valori intorno al 30% con qualche eccezione sul 70%.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

L'**ingegneria dei requisiti** [1] è un processo che si occupa della raccolta, documentazione e revisione dei requisiti e dei servizi che un sistema deve fornire. Riconoscere ed identificare i requisiti in un sistema non è sempre una procedura agevole in quanto possono esprimere sia funzionalità osservabili dal punto di vista dell'utente finale, sia caratteristiche qualitative che sono più nascoste e che vanno a rappresentare i vincoli del sistema. Dal punto di vista della progettazione, quindi, la loro identificazione può aiutare senza dubbio all'ideazione ed allo sviluppo di software di qualità. Una particolare attenzione va comunque alla loro corretta individuazione, che può semplificare il processo di sviluppo. Vista la loro natura eterogenea, i requisiti sono stati suddivisi nel tempo in due grandi categorie, ovvero quella dei **Requisiti funzionali** e dei **Requisiti non funzionali**.
I requisiti non funzionali possono dividersi a loro volta in altre sotto-categorie che toccano ogni aspetto qualitativo che il software deve o dovrebbe avere. Tra queste vi è la categoria dei **requisiti di sicurezza** che va a stabilire la robustezza, protezione agli attacchi ed affidabilità del sistema che si sta sviluppando.

1.2 Motivazioni ed Obiettivi

L'individuazione e la raccolta dei requisiti è spesso effettuata da esperti del settore, il cui compito è la lettura e comprensione dei documenti di specifica dei requisiti. La forma più tipica di rappresentazione e descrizione di un requisito è il linguaggio naturale, che porta con sè tutte le difficoltà relative alla comprensione ed interpretazione del testo. Da questa problematica sono nati moltissimi spunti per la ricerca e la classificazione automatica dei requisiti non funzionali, con particolare attenzione a quella dei requisiti di sicurezza.
Lo scopo di questa tesi è quindi lo studio della libreria **Lambeq** [2] che si colloca nel contesto del **Natural Language Processing** e del **Quantum Natural Language Processing**. Verrà quindi testata una sua applicazione alla classificazione binaria dei requisiti di sicurezza, effettuando dei test sui modelli da essa forniti ed effettuando anche delle comparazioni tra le funzionalità che seguono sia la teoria classica che quella quantistica.

1.3 Risultati ottenuti

La sperimentazione ha portato all'analisi di diversi modelli, appartenenti sia alla teoria classica sia a quella quantistica, e all'utilizzo di vari iperparametri per testare appieno le potenzialità di ciascun modello.
I dati utilizzati sono stati presi da dataset noti in letteratura per la classificazione binaria di requisiti di sicurezza, su cui sono stati applicate delle modifiche per renderli compatibili con le funzionalità della libreria.
La validazione dei modelli è stata suddivisa in due fasi. Nella prima è stato utilizzato un algoritmo Grid-Search per quantificare la bontà dei modelli con ogni combinazione di iperparametri selezionati. Nella seconda fase è stata effettuata un K-Fold Cross Validation con le due combinazioni di modelli migliori ottenuti dalla fase precedente.
Al termine delle due fasi di validazione effettuate sono stati individuate le seguenti combinazioni di iperparametri:

- **BCEWithLogitsLoss - AdamW - 3e-2**, tripla di iperparametri relativi alla pipeline classica i cui risultati hanno superato, seppur di poco, il **50%** di score in ogni

metrica salvo alcune eccezioni dovute ad errori in fase di addestramento.

- **SPSAOptimizer - 5e-2**, coppia di iperparametri relativi alla pipeline quantistica, i cui risultati non hanno prodotto numeri sullo stesso livello della loro controparte classica. Tuttavia sono presenti alcuni valori interessanti di precision che in alcuni casi hanno toccato il **70%** e il **77%**.

1.4 Struttura della tesi

La tesi è strutturata nei seguenti capitoli:

- **Capitolo 2: Background e Stato dell'Arte**: contiene una panoramica sullo stato dell'arte in cui questa tesi si colloca. Viene fatta un'introduzione sull'**ingegneria dei requisiti**, il **Machine Learning** con un richiamo anche al **Quantum Machine Learning** e il **Natural Language Processing**. Inoltre è presente una sezione dedicata agli studi ed i lavori effettuati in letteratura sulla classificazione di NFRs in generale ed anche di requisiti di sicurezza.

- **Capitolo 3: Metodologia**: contiene e spiega il lavoro effettuato in questa sperimentazione. In particolare verranno esplicitati gli obiettivi, gli studi effettuati, i dataset e le strategie di validazione.

- **Capitolo 4: Risultati**: in cui sono mostrati e commentati i risultati che sono stati ottenuti in seguito alle due fasi di validazione effettuate sulle due pipeline implementate, discutendo anche sulle differenze e sulle problematiche che sono emerse.

- **Capitolo 5: Conclusioni**: dedicato alle conclusioni tratte da questa sperimentazione, i risultati ottenuti in generale e gli spunti per eventuali studi e sviluppi futuri.

CAPITOLO 2

Background e Stato dell'arte

2.1 Background

2.1.1 Ingegneria dei Requisiti

La progettazione e lo sviluppo di sistemi software ha come punto principale quella della raccolta e revisione dei requisiti. La branca dell'Ingegneria del Software che si occupa della definizione, documentazione e revisione dei requisiti prende il nome di **Ingegneria dei Requisiti** [1].
La raccolta dei requisiti di un sistema non sempre risulta essere una pratica semplice in quanto essi possono essere sia esplicitati direttamente dall'utente, quindi parliamo delle funzionalità che il prodotto finale deve offrire, ma possono anche essere "nascosti", ovvero deducibili a seconda dei vincoli del problema e dei comportamenti delle singole componenti.
Considerando la quantità di funzionalità e caratteristiche qualitative che possono andare ad identificare, li possiamo suddividere in due grandi categorie:

- **Requisiti Funzionali**, ovvero le funzionalità che il sistema deve poter fornire e che vengono esplicitate direttamente dall'utente.

- **Requisiti Non Funzionali**, la cui definizione non sempre trova d'accordo tutti [3] ma in linea generale molto spesso si riduce alle qualità e caratteristiche che un sistema software deve avere.

A loro volta i requisiti non funzionali (**NFR**) si suddividono in altre categorie, come ad esempio *Sicurezza*, *Usabilità*, *Performance*, *Etici*, *Scalabilità*, *Manutenzione*, *Compatibilità*.

2.1.2 Identificazione dei requisiti di un sistema software

L'identificazione dei requisiti richiede una buona conoscenza della materia e del caso di studio oltre che ad esperienza nel settore. È una pratica che si trova al centro di numerosi studi e la bravura dei ricercatori e degli esperti ha portato all'ideazione di **patterns** utili alla ricerca e all'identificazione di NFR basati su convezioni e caratteristiche generali individuate nel corso degli anni [4].
Altri approcci per la ricerca di NFR riguardano metodi **experience-based**, mostrati da Joerg Doerr *et al.* (2005) [5] in uno studio in cui, collaborando con aziende ed enti appartenenti ad aree di studio differenti, sono riusciti ad ottenere dei risultati piuttosto soddisfacenti.

Requisiti di sicurezza

Una categoria fondamentale tra i NFR è senza dubbio quella dei **requisiti di sicurezza**, senza i quali un sistema software risulta essere altamente vulnerabile ad attacchi e ad azioni malevole. L'identificazione dei requisiti di sicurezza viene effettuata allo stesso modo degli altri, con alcune aggiunte interessanti. Infatti un articolo di Sindre *et al.* (2001) [6] ha mostrato la possibilità dell'individuazione dei requisiti di sicurezza a partire dai cosiddetti **misuse cases**. Lo studio condotto ha infatti mostrato che molto spesso i singoli casi d'uso risultano essere inefficaci se presi singolarmente per estrarre i requisiti di sicurezza di un sistema software, tuttavia integrando la loro controparte che esprime i comportamenti non desiderati si riesce a riconoscere una maggiore quantità di requisiti.

2.1.3 Natural Language Processing

Il **Natural Language Processing**, anche spesso denominato **NLP**, è una branca dell'informatica ed in particolare dell'*Intelligenza Artificiale* (IA) il cui obiettivo è quello di far imparare ai computer, ed alle macchine in generale, il linguaggio naturale scritto e parlato quotidianamente dagli essere umani. La sottomissione di testi scritti in languaggio naturale alle macchine che devono apprendere sintassi e semantica del linguaggio è la parte centrale di questa pratica che però nel corso degli anni ha dovuto superare diverse sfide e complicazioni.

Analisi del linguaggio e Grammatica

La componente su cui si basa principalmente il natural language processing è senza dubbio l'**analisi del linguaggio** [7], che ha come core la scomposizione delle frasi in componenti del discorso (*Parts of Speech*) e l'etichettatura di essi seguendo delle regole grammaticali ben precise.
L'analisi di una frase viene inizialmente effettuata nella sua sintassi, quindi viene scomposta nelle sue componenti fondamentali e successivamente viene eventualmente **regolarizzata**, ovvero semplificata eliminando quelle componenti PoS che non fanno cambiare il senso del discorso. In seguito viene effettuata una fase di analisi della semantica, in cui si va a ricercare il significato vero e proprio di quello che si sta analizzando.
Di uguale importanza è senza dubbio la grammatica, senza la quale non sarebbe possibile condurre le analisi precedentemente menzionate. Essa gioca un ruolo fondamentale nella scomposizione ed etichettatura in quanto è possibile andare a distinguere la natura stessa delle frasi analizzate, ovvero:

- **Noun Phrases** (NP), in cui la componente che fa da capo del discorso è un nome (in letteratura indicato con **Noun, N**).
- **Verb Phrases** (VP), in cui la componente che fa da capo del discorso è un verbo (indicato con **Verb, V**).
- **Prepositional Phrases** (PP), in cui la componente che fa da capo del discorso è una preposizione.

- **Adjective Phrases** (ADJP), in cui la componente che fa da capo del discorso è un aggettivo (indicato con **Adjective, Adj**).
- **Adverb Phrases** (ADVP), in cui la componente che fa da capo del discorso è un avverbio (indicato con **Adverb, Adv**).

Utilizzi del Natural Language Processing

L'NLP copre molte aree relative al linguaggio scritto e parlato. Alcune applicazioni che nel corso degli anni sono state create e migliorate sono lo *Speech-to-text*, ovvero il riconoscimento vocale per mezzo di audio e la conversione in testo scritto, *Etichettatura e Disambiguazione* delle parti del discorso, il *Text-to-speech*, ovvero la generazione di audio a partire da testi scritti, *Sentiment Analysis*, ovvero l'analisi dei testi e l'estrazione di emozioni, atteggiamenti, ecc... da un testo.

Sfide e complicazioni del linguaggio naturale

L'utilizzo dell'NLP ha sin da subito dovuto affrontare tutte le complicazioni che il linguaggio scritto e parlato porta con se. Regole sintattiche particolari, metafore ed omografi sono alcuni esempi che evidenziano la complessità del linguaggio naturale che hanno costretto i ricercatori a trovare delle soluzioni. Nadkarni *et al.* [8] hanno evidenziato alcune di queste problematiche fornendo come esempio l'utilizzo dell'NLP sui testi in ambito medico, mostrando come molti termini medici e scientifici di uso comune possano aver portato non pochi problemi con il riconoscimento delle parole.

2.1.4 Machine Learning

Il **Machine Learning** (ML) fa riferimento ad una branca dell'informatica che tratta l'apprendimento e la capacità dei computer di migliorare le proprie capacità grazie all'utilizzo di dati. Secondo la teoria *classica* [9], il machine learning si differenzia in tre tipi di apprendimento [10]:

- **Supervisionato**, in cui i modelli vengono inizialmente guidati all'apprendimento per mezzo di dati già etichettati con le relative classi di appartenenza.

L'apprendimento supervisionato viene impiegato specialmente nelle task di **classificazione**.

- **Non Supervisionato**, dove i modelli vengono addestrati con dati la cui etichetta è nascosta per tutto il processo di apprendimento. Questo implica la capacità del modello di estrarre le caratteristiche comuni ai dati di input e creare dei *cluster* su cui poi possono essere effettuate delle predizioni.

- **Per rinforzo**, in cui il modello viene guidato per mezzo regole ed obiettivi, con relativi "premi" e "punizioni", che sono utili per il processo di apprendimento. L'addestramento per rinforzo viene adoperato principalmente per lo sviluppo di *agenti intelligenti*.

Quantum Machine Learning

L'introduzione dei sistemi quantistici [11] nel mondo del machine learning ha portato nuove prospettive e spunti di studio per lo sviluppo e l'ottimizzazione di nuovi sistemi o sistemi già esistenti. Questi presentano delle differenze rispetto ai sistemi che seguono le architetture classiche, infatti si è creato una sorta di "ponte" tra i modelli e i sistemi quantistici e il loro corrispettivo classico. Le differenze le notiamo principalmente nelle unità fondamentali come ad esempio:

- **qbit**, che a differenza del classico bit il cui stato può assumere il valore 0 e 1, presenta anche una **sovrapposizione lineare** [9] di questi ultimi.

- **Parameterized Quantum Circuits** (PQCs), che sono considerati come la versione quantistica delle reti neurali per il ML classico.

La forza delle componenti quantistiche risiede nel fatto che possono essere usate anche per rappresentare informazioni ed incorporare modelli appartenenti alle architetture classiche. Stringhe binarie di bit possono essere convertite in stringhe equivalenti di qbit, **QNN** (Quantum Neural Networks) possono sfruttare modelli e reti neurali classiche. Grazie a questa caratteristica e al progredire della ricerca e delle tecnologie, gli studi si sono concentrati nel trovare spunti per l'ottimizzazione di sistemi di machine learning classici, che potenzialmente potrebbero ridurre drasticamente i tempi di esecuzione ed incrementare la precisione. Inoltre i campi di

applicazione vanno dalla semplice task di classificazione alle applicazioni in ambito chimico e scientifico.

2.2 Stato dell'Arte

Gli studi di identificazione e classificazione dei requisiti non funzionali, in particolare quelli di sicurezza, sono da sempre al centro delle ricerche ed hanno come obiettivo l'**automatizzazione** del processo di identificazione e la ricerca di modelli sempre più **precisi**. Tra i vari studi che sono stati effettuati bisogna però fare una distinzione tra quelli che seguono gli approcci classici relativi al machine learning, che nel corso degli anni si sono consolidati in termini di efficienza e di risultati effettivi, e quelli che si inseriscono nel mondo della sperimentazione quantistica, che ad oggi è ancora in corso di miglioramento ma che già sta dando risultati notevoli se confrontati con i loro corrispettivi classici.

2.2.1 Identificazione di NFRs

Haque *et al.* (2019) [12] hanno effettuato uno studio comparativo su 7 diversi modelli, come ad esempio **Support Vector Machines** (SVM), **Naive Bayes** (NB), **K-NearestNeighbors** (K-NN), ecc..., in combinazione con alcune tecniche di feature extraction, tra cui **Bag of Words** (BoW) e **Term Frequency–Inverse Document Frequency** (TF-IDF). Nell'esperimento, condotto utilizzando il dataset PROMISE contenente diverse categorie di NFRs, è stata effettuata una classificazione multiclasse che ha prodotto risultati altalenanti ma che comunque possono essere una baseline valida per ulteriori studi futuri.
Un altro studio condotto in precedenza da Slankas *et* Williams (2013) [13] sulla classificazione dei NFRs ha coinvolto l'utilizzo dello stesso dataset come base per la suddivisione in categorie di requisiti. Nel loro esperimento hanno realizzato il tool **NFR Locator k-NN** che è suddiviso in una parte di riconoscimento dei requisiti a partire da documenti scritti in linguaggio naturale e da una successiva che consiste nella vera e propria classificazione.

2.2.2 Identificazione dei requisiti di sicurezza

Jindal *et al.* (2016) [14] hanno sviluppato dei modelli di predizione basati sul **Decision Tree J48** [15] a partire da un dataset pubblico contenenti alcuni Requirements Specification Documents. Su questi dati sono stati effettuati dei lavori di text mining per estrarre i requisiti di sicurezza e successivamente sono stati costruiti dei modelli di predizione binari, uno per ogni categoria identificata, che hanno ottenuto risultati promettenti e che lasciano spunti per sviluppi futuri, come anche esplicitato dagli autori stessi, per estendere questo metodo anche ad altre categorie di NFRs.
Riaz *et al.* (2014) [16] hanno creato un tool chiamato **Security Discoverer**, le cui performance hanno senza dubbio portato ad un ulteriore step in avanti nell'identificazione dei NFRs. Nel dettaglio, questo tool permette all'utente di inserire artefatti in linguaggio naturale, possibilmente well-formed e senza errori vari, e di estrarne le categorie di sicurezza sia implicite che esplicite. Per la creazione è stato utilizzato un modello basato su **K-NearestNeighbors** [17] che in seguito ai vari addestramenti ha riportato valori di recall e precision abbastanza alti.
Wang *et al.* (2019) [18] invece hanno proposto un modello basato su un **Linear Classifier**, che ha dato risultati ottimi se confrontati con altri lavori e studi già consolidati nel corso degli anni. In questo lavoro sono stati utilizzati progetti importanti relativi ad Apache, GeoServer ecc... e sono stati costruiti dei dataset di requisiti a partire dalle documentazioni fornite da ciascun progetto. Il modello basato su regressione lineare è risultato ottimo, in tutte le metriche analizzate ha ottenuto gli score più alti mantenendo un grande similarità nei risultati per ciascun dataset utilizzato.
Varenov *et al.* (2021) [19] hanno posto invece il problema dell'utilizzo di **NLP Transformers** [20] nel contesto della requirements elicitation. Nel loro studio hanno utilizzato e testato un modello basato su BERT, i cui risultati hanno mostrato una solida base da cui partire su determinate classi di dataset. Nell'esperimento infatti sono stati utilizzati sia *labeled* che *unlabeled* datasets che hanno prodotto risultati buoni da un lato e scarsi dall'altro. Con una successiva "rivisitazione" del modello e dei dataset utilizzati sono riusciti ad ottenere risultati che si trovano alla pari con altre tecniche di classificazione già utilizzate e consolidate.

2.2.3 Computazione quantistica

Ganguly *et al.* (2022) [21] hanno condotto uno studio sull'utilizzo della computazione quantistica in ambito NLP basato sulla stessa libreria di riferimento utilizzata in questa sperimentazione . In questo caso la libreria **Lambeq** [2] è stata impiegata per una classificazione binaria di **Sentiment Analysis**, utilizzando dataset con un numero ristretto di entry, se paragonati ad altri utilizzati in molti altri lavori classici, ottenendo su 4 test condotti dei risultati in metriche di loss e accuracy molto promettenti.
Senokosov *et al.* (2023) [22], invece, hanno applicato i concetti del quantum computing alla classificazione di immagini utilizzando come core del loro esperimento le **Hybrid Quantum Neural Networks** (HQNNs). Nel test è stato effettuato un confronto tra questa nuova tecnologia in forte evoluzione e la sua controparte "classica", migliorando ulteriormente quelli che sono i risultati ottenuti con la computazione classica e riducendo anche il numero di calcoli effettuati per ciascuna task effettuata.
Kavitha *et* Kaulgud (2022) [23] hanno condotto uno studio comparativo sulle **Support Vector Machines** (SVM) e il loro corrispettivo quantistico. Dallo studio è emerso che per i dataset utilizzati, provando e cambiando i vari kernel quantistici disponibili, sono stati ottenuti risultati migliori sulle versioni quantistiche rispetto alle versioni classiche, sia in termini di accuracy che di efficienza. Lo studio inoltre si è concentrato anche sulla comparazione con altri lavori svolti in precedenza, mostrando come la scelta dei kernel quantistici per i modelli abbia inciso in maniera significativa sui risultati e sulle prestazioni stesse.

CAPITOLO 3

Metodologia

3.1 Motivazioni e Obiettivi

L'obiettivo principale della sperimentazione è quello di effettuare uno studio sulla libreria Lambeq nel contesto dell'**ingegneria dei requisiti**. L'intento è partire da alcuni dataset contenenti requisiti di sicurezza e non, e testare i metodi e le funzionalità della libreria per capire se è possibile implementare ed addestrare dei classificatori binari in grado di riconoscerli correttamente.
Vista la possibilità di utilizzare metodi e modelli che seguono sia la teoria **classica** che quella **quantistica** nascono qui le tre domande di ricerca che motiveranno il procedere della sperimentazione:

RQ_1: *In che misura è possibile classificare i requisiti di sicurezza utilizzando una pipeline classica?*

La prima domanda di ricerca ha come obiettivo quello di verificare con quali metodi e con che risultati è possibile classificare i requisiti di sicurezza utilizzando una pipeline costruita con lambeq che segue un approccio classico.

RQ_2: *In che misura è possibile classificare i requisiti di sicurezza utilizzando una pipeline quantistica?*

La seconda domanda di ricerca ha lo stesso obiettivo della prima, con la differenza che in questo caso verranno utilizzati modelli appartenenti alla teoria quantistica.

RQ_3: *In cosa e come si differenziano la pipeline classica e la pipeline quantistica nell'identificazione di requisiti di sicurezza?*

La terza domanda di ricerca mira ad una comparazione tra i risultati che saranno ottenuti dalle pipeline alla fine delle due fasi di validazione.

Sono state quindi implementate e testate entrambe le tipologie di pipeline, effettuando anche delle fasi di validazione per una ricerca degli iperparametri migliori tra quelli forniti dalla libreria di riferimento. Infine sono stati commentati i risultati ottenuti, facendo dei confronti su entrambi gli approcci.

3.1.1 Approccio al problema

Per il corretto utilizzo della libreria è stata molto importante una fase preliminare di studio dei package, delle classi e delle funzionalità che vanno a comporla.
Nel corso dell'esplorazione della libreria e della documentazione ad essa riferita sono stati approfonditi gli oggetti principali che permettono la creazione e l'addestramento di un classificatore binario. Inoltre sono state testare tutte le funzionalità in modo da capirne i meccanismi e i comportamenti per ogni step della pipeline.

3.1.2 Hardware utilizzato

Per tutta la sperimentazione, compresa la parte di ricerca della teoria e dello studio della libreria, è stata utilizzata un macchina con processore AMD Ryzen 5 5600X, ram DDR4 16GB e scheda video NVidia GeForce RTX 3060.
Il sistema operativo utilizzato è Arch Linux, che durante le fasi di test è risultato ottimo in quanto alcune funzionalità della libreria sono state implementate utilizzando librerie di ottimizzazione, come ad esempio *JAX* [24], che sono solo disponibili per

Linux. Trattando anche delle computazioni quantistiche sarebbe stato necessario anche l'utilizzo di macchine apposite, tuttavia non disponendo delle risorse necessarie è stata comunque effettuata sulla stessa macchina una **simulazione quantistica**.

3.2 Introduzione ai dataset utilizzati

I dataset utilizzati per la sperimentazione sono CPN, GPS ed ePurse [25], divenuti uno standard dato che vengono impiegati molto spesso come benchmark di classificatori binari.
Questi forniscono una serie di stringhe rappresentanti delle frasi in linguaggio naturale che, a seconda della categoria di appartenenza, vengono contrassegnate con le seguenti etichette al fine di suddividerli per una classificazione binaria:

- **;sec**, scritto in corrispondenza di stringhe riguardanti requisiti di sicurezza.
- **;nonsec**, scritto in corrispondenza di stringhe non riguardanti i requisiti di sicurezza.

Le stringhe su cui abbiamo lavorato sono inizialmente 511, di cui il 36.6% sono requisiti di sicurezza, ma come vedremo nelle sezioni successive tenderanno a diminuire in quanto è stata effettuata una fase di pulizia dei dati e riformattazione che ha eliminato duplicati e stringhe che non vengono correttamente interpretate dai parser della libreria.

3.2.1 Formattazione e preparazione dei dati

Le entry di ciascun dataset, a causa della formattazione e di altri problemi relativi alla sintassi dei periodi, non sono risultate idonee sin da subito come dataset di test per la sperimentazione, quindi sono state necessarie delle operazioni di preparazione e formattazione dei dati.

1. **Modifica delle etichette**:
   Le etichette relative a ciascuna entry dei dataset, come anche esplicitato nella sezione precedente, sono anch'esse delle stringhe in codice che vanno ad evidenziare la natura di ciascuna entry. Per poter utilizzare queste etichette

all'interno dei modelli forniti dalla libreria, sono state sostituite con altre in formato numerico dove:

- è presente un bit a 0 in corrispondenza delle entry aventi l'etichetta iniziale ;nonsec (quindi non requisito di sicurezza).
- è presente un bit a 1 in corrispondenza delle entry con l'etichetta ;sec (requisiti di sicurezza)

```
Tokens shall be verified by the appropriate Security Domain .;sec
1  Tokens shall be verified by the appropriate Security Domain .

(e.g. VoIP phones, Video phones, STBs, etc) .;nonsec
0  (e.g. VoIP phones, Video phones, STBs, etc) .
```

Figura 3.1: Esempi di entry prima e dopo la formattazione

Essendo poco più di 500 entry, per velocizzare il processo di modifica è stato scritto un piccolo frammento di codice che, preso in input il percorso relativo al file da modificare effettua le modifiche sopra citate.

2. **Normalizzazione degli acronimi e delle numerazioni**
   Avendo a che fare con dataset che trattano requisiti di sicurezza, sono molti gli acronimi e le versioni degli standard utilizzati in campo informatico e giuridico. Essendo parole difficilmente interpretabili dai parser, per far sì che vengano trattate come un'unica entità da analizzare sono stati modificati aggiungendo dei caratteri di spaziatura quali **-** .

```
ISO/IEC 7816-4 -> ISO/IEC-7816-4
ISO/IEC 14443-3 -> ISO/IEC-14443-3
```

Figura 3.2: Esempi di entry prima e dopo la normalizzazione

3. **Aggiunta di alcuni soggetti mancanti**
   Un altro problema riscontrato con i dati iniziali è stato che per un ristretto

numero tra tutte le entry dei dataset, circa il 7.6%, si generavano errori in fase di parsing a causa della presenza di aggettivi o complementi riferiti ad uno stesso soggetto.
In linguaggio naturale questi non sono errori del punto di vista logico o di sintassi ma nel momento in cui andiamo a passare frasi di questo tipo a delle funzioni di parsing, si manifestano dei crash oppure degli errori; questo perchè i parser stessi non riescono a costruire un opportuno string diagram perchè non riescono a "trovare" il soggetto per alcuni degli aggettivi/complementi.
Una soluzione è stata individuare i fault in fase di parsing, grazie a del codice scritto appositamente per risolvere questa problematica, e salvare in file di testo esterni le entry "difettose". Dopo di ciò sono stati corretti gli errori manualmente, in quanto individuare questi "errori" e risolverli con dell'altro codice sarebbe risultato difficoltoso e poco efficiente.

```
The CNG should detect replayed user and/or device credentials.
The CNG should detect replayed user credentials and/or device credentials.

        ("credentials" è riferito sia ad "user" che a "device")
```

Figura 3.3: Esempio di entry prima e dopo l'aggiunta dei soggetti

4. **Modifica delle spaziature ed eliminazione dei duplicati**
   Le ultime due modifiche apportate ai dati riguardano una semplice formattazione del testo. Per rimanere in linea con i dati utilizzati nella documentazione della libreria, sono state aggiunte delle spaziature tra tutti gli elementi di ogni frase. Inoltre sono state rimosse delle entry duplicate ed anche alcune entry che dopo la normalizzazione e l'aggiunta dei soggetti continuavano a generare errori.

Eseguite queste azioni di formattazione e preparazione dei dati, da un numero iniziale di 511 entry su cui andare a lavorare siamo passati a 438 entry totali, con una perdita del 14,2% sul totale.

3.3 Struttura della pipeline

La pipeline di Machine Learning che è stata creata segue le regole NLP implementate dalla libreria Lambeq. Gli step che vanno a costituire la pipeline completa hanno una propria teoria come background e forniscono diverse strategie di implementazione a seconda dell'utilizzo e della complessità del problema. La struttura tipica delle pipeline costruite con Lambeq assume lo schema logico mostrato in figura **3.4**.

Figura 3.4: Fasi della pipeline in Lambeq

Nelle prossime sezioni vengono spiegate nel dettaglio le varie fasi illustrate nella figura **3.4**, mostrando le strategie di utilizzo insieme al codice ed agli output che producono.

3.3.1 Tokenization

Una fase preliminare che è fondamentale per la riuscita di un modello di NLP è quella della **tokenizzazione** che consiste nello "scomporre" una frase, o un intero testo, nei suoi elementi base, ovvero le parole e i segni di punteggiatura, che in ambito tecnico sono definiti **tokens**.
Nel caso di questa sperimentazione, Lambeq fornisce una classe ad-hoc chiamata SpacyTokeniser che si basa sul pacchetto *spaCy* [26], molto famoso nell'ambito NLP, che permette la tokenizzazione singola o in blocco di testi in linguaggio naturale. Un'applicazione del tokeniser è mostrata in figura **3.5**:

```
from lambeq import SpacyTokeniser

tokeniser = SpacyTokeniser()
sentence = "Composing a work isn't easy at all."
tokens = tokeniser.tokenise_sentence(sentence)
#OUTPUT: ['Composing', 'a', 'work', 'is', "n't", 'easy', 'at', 'all', '.']


sentences = ["Composing a work isn't easy at all.",
             "However, some of them have proved the opposite."]
tok_sentences = tokeniser.tokenise_sentences(sentences)
#OUTPUT: [['Composing', 'a', 'work', 'is', "n't", 'easy', 'at', 'all', '.'],
#         ['However', ',', 'some', 'of', 'them', 'have', 'proved', 'the', 'opposite', '.']]
```

Figura 3.5: Utilizzo dello SpacyTokeniser

3.3.2 Parsing & Encoding

Le frasi in linguaggio naturale vengono date in input ad un analizzatore grammaticale, in letteratura definito **parser**, che le va prima a scomporre in parole e segni di punteggiatura e che successivamente le converte in **diagrams**.
I parser forniti dalla libreria sono suddivisi per modelli e possono essere utilizzati sia per il caso classico che per quello quantistico.

Syntax-based Model

Il modello di tipo syntax-based implementato fornisce una trasformazione da frase a **string diagram** secondo le regole del modello DIStributional COmpositional CATegorical (**DisCoCat**) [27]. Le singole parti di ogni frase vengono marcate con un'opportuna categoria, a seconda del ruolo che assumono nel contesto in cui si trovano. Il parser che implementa questo modello è **BobcatParser** e fornisce in output diagrammi come quello mostrato in figura **3.6**.

```
from lambeq import BobcatParser, pregroups

sentence = 'Ludwig plays the piano'
parser = BobcatParser(verbose = 'progress')
diagram = parser.sentence2diagram(sentence)
pregroups.draw(diagram, figsize=(14,3), fontsize=12)
```

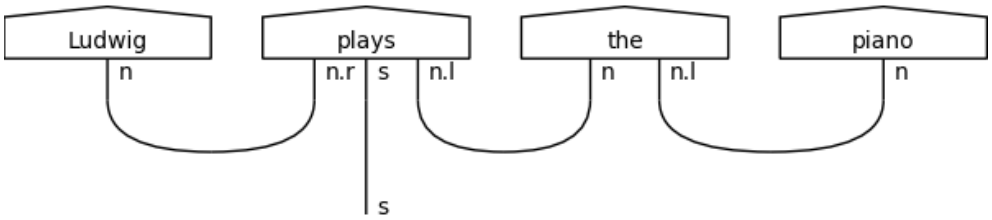

Figura 3.6: Parsing con BobcatParser

Bag-of-Words Model

Il modello di tipo bag-of-words non va a considerare le parole di una frase tenendo conto anche della sintassi e del significato, ma le tratta così come sono e le va ad unire in un unico grande "**spider diagram**". L'implementazione di questo modello viene fornita tramite il parser **spiders_reader** e dà in output diagrammi nella forma mostrata in figura **3.7**.

```
from lambeq import spiders_reader

sentence = 'Ludwig plays the piano'
spiders_diagram = spiders_reader.sentence2diagram(sentence)
spiders_diagram.draw(figsize=(13,6), fontsize=12)
```

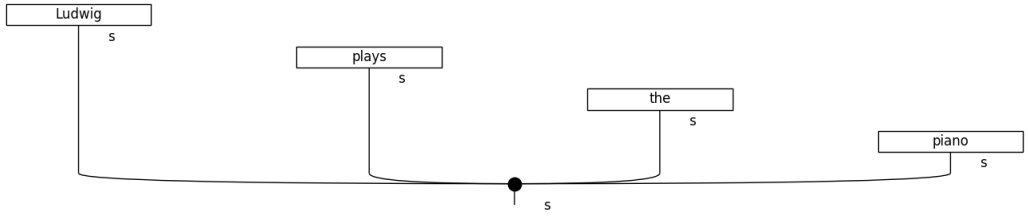

Figura 3.7: Parsing con spiders_reader

Word-Sequence Model

Il modello di tipo word-sequence costruisce i diagram man mano che le parole vengono lette ed analizzate; lambeq fornisce l'implementazione di questo modello con due parser differenti:

- **cups_reader**, genera un "**tensor_train**" come output che viene composto aggiungendo parole da destra verso sinistra ed espandendolo mano mano con sentences sempre più complesse. Il seguente codice mostra il necessario per ottenere in output un diagram del tipo della figura **3.8**.

```
from lambeq import cups_reader, pregroups

sentence = 'Ludwig plays the piano'
cups_diagram = cups_reader.sentence2diagram(sentence)
pregroups.draw(cups_diagram, figsize=(14,3), fontsize=12)
```

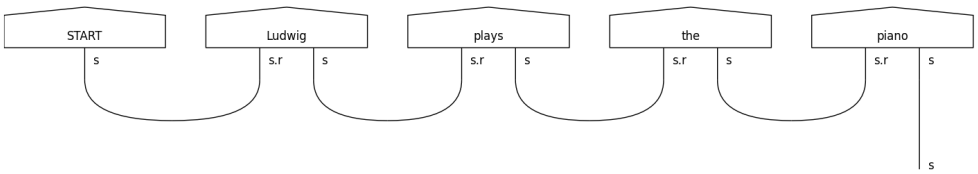

Figura 3.8: Parsing con cups_reader

- **stairs_reader**, il cui funzionamento è simile a quello di cups, con la differenza che vengono creati dei diagrammi a blocchi, come mostrato in figura **3.9**.

```
from lambeq import stairs_reader

sentence = 'Ludwig plays the piano'
stairs_diagram = stairs_reader.sentence2diagram(sentence)
stairs_diagram.draw(figsize=(14,4), fontsize=12)
```

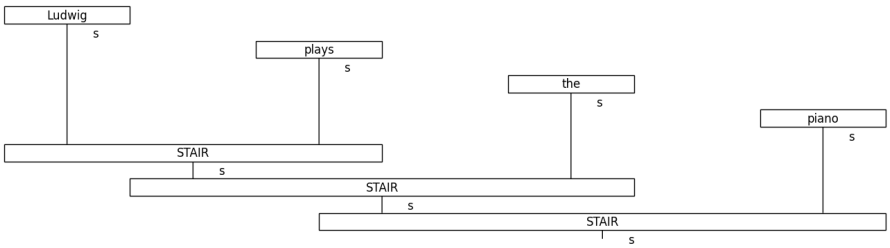

Figura 3.9: Parsing con cups_reader

3.3.3 Parametrization

Le frasi tokenizzate e successivamente convertite in string-diagrams non sono certo un input consono per il modello che si vuole addestrare. La fase che produrrà i dati di input per la fase di training è quella relativa alla **parametrizzazione** dei diagrammi in stringhe compatibili.
La conversione dei diagrammi viene affidata agli **ansätz** che, dalla traduzione tedesca applicata all'ambito NLP, rappresentano degli approcci riguardo ai dati in linguaggio naturale. In particolare, parlando di conversioni di diagrammi di stringhe, gli approcci sono la conversione dei diagrammi di stringhe in **tensor networks**, relativi alle pipeline classiche, oppure **quantum circuits**, che riguardano le pipeline quantistiche.

Ansatz per il caso classico

Gli ansatz per la conversione di diagrammi di stringhe in tensori che vengono implementati dalla libreria sono principalmente tre:

- **TensorAnsatz**: ansatz base per la conversione in tensori, nel caso di studio è risultata la scelta migliore per questioni di compatibilità. Un esempio del suo utilizzo è mostrato in figura **3.10**.
- **MPSAnsatz** e **SpiderAnsatz**: estensioni del TensorAnsatz che implementano il concetto di **Matrix Product States**, ovvero una funzione che fattorizza un tensore in più tensori con una taglia fissata in modo arbitrario.

```
from lambeq import TensorAnsatz, AtomicType

tensor = TensorAnsatz({AtomicType.NOUN: 2, AtomicType.SENTENCE: 2})
tensor_diagram = tensor(diagram)
tensor_diagram.draw(figsize=(10,4), fontsize=13)
```

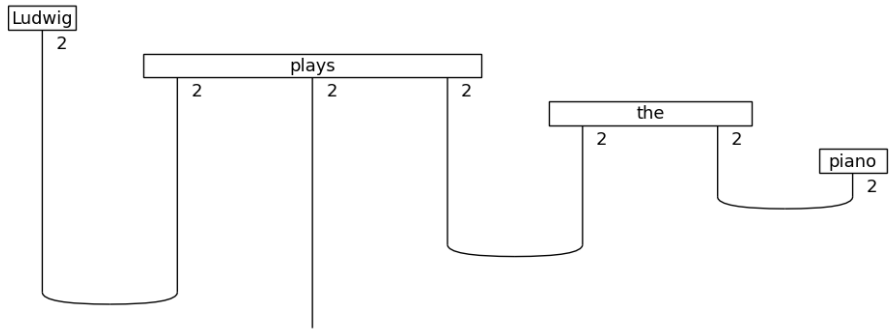

Figura 3.10: Esempio di conversione utilizzando TensorAnsatz

Ansatz per il caso quantistico

Gli ansatz per la conversione in circuiti quantistici che sono forniti dalla libreria sono vari e derivano dalla classe base **CircuitAnsatz**, che introduce i parametri relativi al numero di layer e i qbits per ogni parametro.
La scelta dell'ansatz da utilizzare nella sperimentazione è stata fatta a partire da uno studio di Nikhil Khatri [28], che evidenzia nel suo lavoro le differenze ed i risultati ottenuti utilizzando varie configurazioni tra gli ansatz disponibili.
Gli ansatz utilizzati nello studio sono **IQPAnsatz** (Instantaneous Quantum Polynomial), **Sim14Ansatz**, **Sim15Ansatz**, **StronglyEntanglingAnsatz**.
Dallo studio emerge che sebbene i tre nuovi ansatz hanno delle prestazioni in accuracy accettabili, la scelta di IQP rimane comunque quella ottimale, questo anche grazie

ai numerosi studi e ricerche che sono stati effettuati per ottimizzare al meglio questo approccio di conversione. Un esempio della conversione utilizzando IQPAnsatz è mostrato in figura **3.11**.

```
from lambeq import IQPAnsatz, AtomicType

iqp = IQPAnsatz({AtomicType.NOUN: 2, AtomicType.SENTENCE: 2},
                n_layers = 2,
                n_single_qubit_params = 3)
circuit = iqp(diagram)
circuit.draw(figsize=(14,10))
```

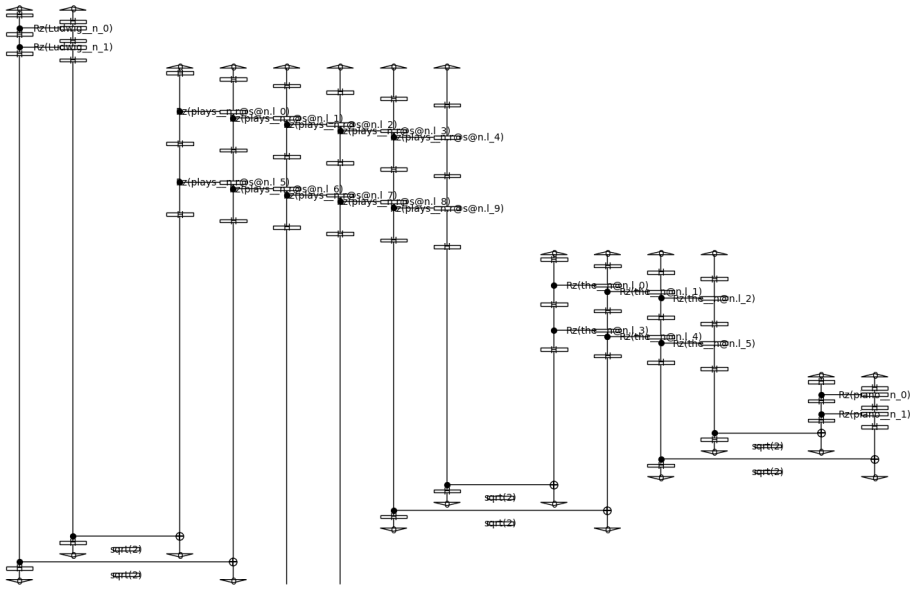

Figura 3.11: Esempio di conversione utilizzando IQPAnsatz

Il diagramma utilizzato per la conversione in circuito quantistico è quello risultato dal codice della figura **3.6**.

3.3.4 Training

La fase di training è composta da cinque sezioni:

- Inizializzazione di un modello sui circuiti e i label estratti nelle fasi precedenti
- Inizializzazione di un oggetto Trainer, passando tutti gli iperparametri necessari

- Inizializzazione degli oggetti Dataset da passare al trainer
- Addestramento del modello con i dati di train ed eventualmente una validazione con i dati di test
- Plot su grafico dei risultati ottenuti

Tra l'applicazione quantistica e quella classica non ci sono differenze, tranne per gli oggetti Model e Trainer che cambiano nell'architettura ma che rimangono uguali per iperparametri dati in input e metodi per l'esecuzione dell'addestramento.

3.4 Implementazione delle pipeline

Per l'implementazione delle pipeline è stato scelto un approccio di tipo object-oriented in modo tale da avere una suddivisione più chiara delle funzionalità e degli step descritti in precedenza. Insieme alle classi che racchiudono le funzionalità di lambeq, sono state scritte altre porzioni di codice per il testing sui dataset e alcune funzioni di utility quali l'estrazione e la formattazione dei dati oppure il salvataggio dei diagrammi su file, con la libreria *pickle*.

3.4.1 Pipeline e funzionalità

Le classi adibite alle funzionalità usate per l'approccio classico e quantistico prendono il nome di ClassicPipeline e QuantumPipeline rispettivamente. All'interno sono presenti i metodi che eseguono i passi utili per preparare i dati, convertirli prima in diagrammi e poi in circuiti, ed infine formattarli per far partire un addestramento.

Estrazione delle etichette e costruzione dei circuiti

Il primo step riguarda l'estrazione delle etichette relative a ciascuna entry dei vari dataset e della conversione dei requisiti in diagrammi di stringhe e poi in tensori (o circuiti quantistici). Gli output in questa fase sono quindi due oggetti collection che includono in maniera ordinata etichette e circuito corrispondente.

```
from lambeq import *
from pipeline import *

pipeline = Pipeline(parser, ansatz)
train_labels, train_circuits = pipeline.create_circuits_and_labels("path/to/file")
test_labels, test_circuits = pipeline.create_circuits_and_labels

#OUTPUT LABELS: [[1.0, 0.0], [1.0, 0.0], [1.0, 0.0], [1.0, 0.0], ..., [0.0, 1.0]]
#OUTPUT CIRCUITS: [A list of circuits objects]
```

Figura 3.12: Estrazione e creazione dei circuiti

Costruzione dei dataset e inizializzazione del modello

A partire dalle etichette e dai circuiti della prima fase si vanno a costruire degli oggetti Dataset compatibili con i trainer della libreria. Gli stessi circuiti vengono dati in input al modello che verrà addestrato, in modo tale da creare il dizionario delle parole su cui il modello dovrà essere capace di fare delle predizioni.

```
from lambeq import *
from pipeline import *

train_dataset = pipeline.create_dataset(train_labels, train_circuits)
test_dataset = pipeline.create_dataset(test_labels, test_circuits)
model = pipeline.create_model(train_circuits, test_circuits)

#OUTPUT: due oggetti dataset e il modello da addestrare
```

Figura 3.13: Dataset e modello da addestrare

Inizializzazione ed esecuzione del training

L'ultima fase riguarda l'inizializzazione del trainer e l'esecuzione dell'addestramento sui dataset creati in precedenza. I risultati che vengono mostrati in output sono le epoche di training che possono essere plottate su grafico per una maggiore comprensione dell'andamento dei valori ottenuti nella sessione di addestramento.

```
from lambeq import *
from pipeline import *

trainer = pipeline.create_trainer(model, loss_function, optimizer, learning_rate, epochs)
pipeline.train_and_evaluate(train_dataset, test_dataset)

#OUTPUT: Epoch 1: train/loss: 0.6933 train/acc: 0.4970 ... valid/rec: 0.4113
#        Epoch 2: train/loss: 0.6931 train/acc: 0.4970 ... valid/rec: 0.4032
#          ...
#        Epoch 100: train/loss: 0.252 train/acc: 0.8570 ... valid/rec: 0.7356
```

Figura 3.14: Blocco di training

Per un maggiore approfondimento, il codice prodotto per la sperimentazione è disponibile alla seguente repository github[1].

3.5 Validazione di un modello di Machine Learning

Una fase fondamentale nel processo di creazione di un modello di machine learning è la **validazione**, che ci dà una visione più ampia del modello che stiamo sviluppando e ne evidenzia il comportamento e soprattutto le criticità.

3.5.1 Criticità in un modello

Identificare e affrontare le criticità è essenziale per sviluppare modelli di machine learning accurati e affidabili. Esse possono essere numerose e possono manifestarsi in qualsiasi modello, a partire dai dati fino ai comportamenti interni del modello.
Le criticità più comuni tra quelle che possono manifestarsi sono:

- **overfitting e underfitting**: un modello di machine learning manifesta problemi di overfitting quando riesce a fornire risultati molto precisi sui dati di training ma non riesce a dare risultati soddisfacenti con i dati di test. In altre parole è un modello che si "abitua" ai dati di training e che inizialmente fornirà risultati precisi, ma se si considerano dati leggermente differenti i risultati saranno molto spesso imprecisi.
  Al contrario, quando si manifestano problemi di underfitting, il modello sarà

[1] https://github.com/adriano22jr/Tesi-Quantum-NLP

troppo semplice per i dati o per il problema che deve risolvere e quindi avrà una bassa precisione sia tra i dati di training che tra quelli di test.

- **qualità dei dati**: a volte i dati che si stanno utilizzando non sono adatti al modello che si sta sviluppando e quindi possono creare dei problemi. Questi possono presentarsi sia per la scarsa quantità di informazioni, che non permetterebbe al modello di apprendere al meglio le relazioni tra i dati, sia per la qualità vera e propria, ovvero l'assenza di valori oppure valori errati.

- **scelta dei parametri**: la scelta dei parametri ottimali per un modello di machine learning è fondamentale per i risultati che si vogliono ottenere. Infatti i risultati e le capacità predittive di un modello possono cambiare drasticamente se i parametri sono più o meno ottimizzati.

3.5.2 Tecniche di validazione

Le tecniche di validazione sono lo strumento principale per valutare il comportamento di un modello di Machine Learning.
In questa sperimentazione è stata effettuata una fase di validazione suddivisa in due parti: nella prima sono stati cercati i parametri ottimali tramite un algoritmo Grid-Search mentre nella seconda è stato effettuato un test di validazione vero e proprio mediante una K-Fold Cross-Validation.

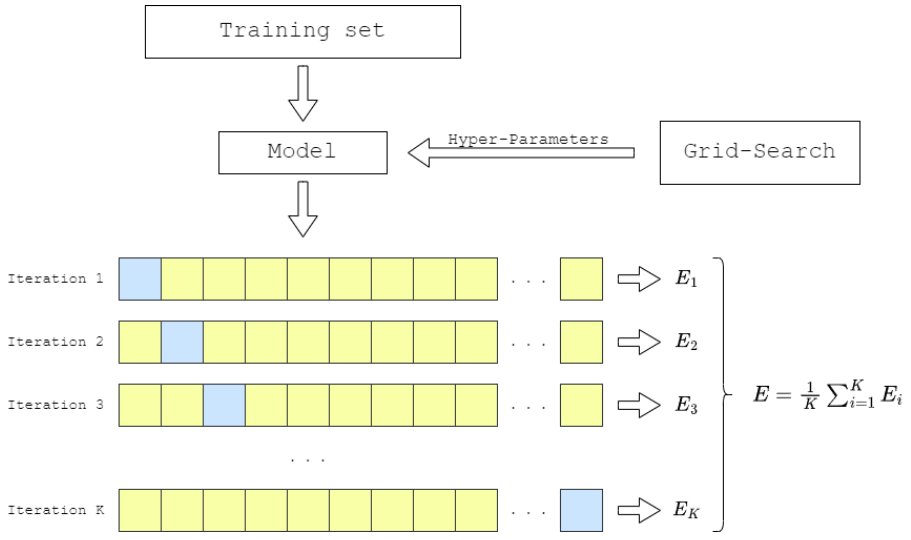

Figura 3.15: Processo di validazione usato nella sperimentazione

Gridsearch e ricerca dei parametri

La Grid-Search [29] è una tecnica di validazione che consiste nel definire un set di valori per ogni parametro e provare tutte le possibili combinazioni di valori. Questa tecnica è molto utilizzata nel campo del Machine Learning in quanto, se implementata a dovere, riesce a dare in output il valore migliore per ogni parametro che si vuole ottimizzare, massimizzando così la resa del modello che si sta sviluppando.
La tecnica Grid-Search è stata particolarmente utile in questa fase di validazione dei modelli scritti con la libreria di riferimento in quanto vengono messi a disposizione dei trainer con numerosi parametri da poter ottimizzare. Alla fine dell'esecuzione sono state selezionate le combinazioni di valori che hanno dato come input i risultati più equilibrati, sia nei valori di perdita che negli score delle metriche.
Gli oggetti trainer che la libreria fornisce sono pochi, sia per il caso di studi classico che per quello quantistico, quindi la scelta del trainer da utilizzare è stata semplice e rapida.

1. **Caso Classico**: il trainer selezionato per l'addestramento della pipeline classica è stato il **PytorchTrainer** che mette a disposizione diversi parametri.
   Nel dettaglio, i parametri su cui è stata effettuata la gridsearch riguardano gli **optimizer** (utilizzando quelli forniti dalla libreria pytorch), la **funzione di perdita**, il **numero di epoche** e il **learning-rate**.

```
import torch

epochs = [100]
loss_functions = [torch.nn.BCEWithLogitsLoss(), torch.nn.HingeEmbeddingLoss()]
optimizers = [torch.optim.AdamW, torch.optim.Adagrad, torch.optim.Adam, torch.optim.Adamax]
learning_rates = [3e-1, 3e-2, 3e-3]
```

Figura 3.16: Parametri su cui è stata effettuata la gridsearch per il caso classico

L'algoritmo di Grid-Search è stato implementato manualmente ed effettua un numero di addestramenti pari a tutte le possibili combinazioni dei parametri nelle liste elencate sopra, inoltre è stato scritto in modo tale da poter funzionare sia per i test classici che per quelli quantistici. Il modello di riferimento è stato costruito a partire dai 3 dataset formattati descritti nella sezione precedente.

```
model = pip.create_model(train_circuits, test_circuits, eval_circuits)

for loss in loss_functions:
    for optimizer in optimizers:
        for lr in learning_rates:
            print(f"\nCurrent triple: \n -LEARNING RATE: {str(lr)}\n" +
                                        f"-LOSS FUNCTION: {str(loss)}\n" +
                                        f"-OPTIMIZER: {str(optimizer)}")
            pip.create_trainer(model = model, loss = loss, optimizer = optimizer,
                               n_epochs = 100, lr = lr, evaluate = True)
            pip.train_and_evaluate(train_set, eval_set, 1, 1)
```

Figura 3.17: Implementazione dell'algoritmo di Grid-Search

2. **Caso Quantistico**: per l'addestramento quantistico è stato selezionato il **QuantumTrainer** che, come per il trainer del caso classico, permette di fare ricerca su alcuni **optimizers**, alcuni iperparametri relativi o che vengono influenzati dal **learning-rate** e il **numero di epoche**.
   Sfortunatamente però a causa dell'hardware che non è prettamente indicato per delle task di simulazione quantistica, non è stato possibile effettuare dei test con gli optimizer Rotosolve e NelderMead in quanto i tempi di attesa per una singola epoch superavano l'ora.

```
epochs = [100]
optimizers = [SPSAOptimizer, RotosolveOptimizer, NelderMeadOptimizer]
learning_rates = [5e-1, 5e-2, 5e-3]
```

Figura 3.18: Parametri su cui è stata effettuata la gridsearch per il caso quantistico

L'algoritmo di Grid-Search utilizzato per il caso quantistico è lo stesso utilizzato per il caso classico, tranne per qualche ciclo interno mancante.

K-Fold Cross Validation

La K-Fold Cross Validation [30] è una tecnica di validazione di modelli di Machine Learning che si basa sull'iterare per K volte degli addestramenti e di dare in output una media degli score delle metriche e della funzione di perdita. Più nel dettaglio, il dataset di riferimento viene diviso in K gruppi che, a turno, andranno a costituire i dati di test mentre i restanti k-1 comporranno i dati di training.

Nel caso della seconda fase della sperimentazione è stata applicata un 5-Fold Cross Validation, il che significa che il dataset è stato diviso in 5 gruppi di cui 1 va a costituire il test set e i restanti 4 il train set. I parametri utilizzati sono stati quelli ottenuti nella prima fase di Grid-Search, e dopo aver eseguito la Cross-Validation per ogni combinazione dei 3 dataset di riferimento, sono state calcolati i risultati medi di ciascuna metrica di valutazione.

3.5.3 Metriche di valutazione

La fase successiva alla validazione del modello è quella della valutazione. Tutti i modelli di machine learning infatti vengono valutati secondo delle metriche che ne stimano la capacità di predizione.
Una fase che viene però ancora prima del calcolo di queste metriche è quella che riguarda la suddivisione delle predizioni effettuate dal modello in esame in 4 diverse categorie:

1. **True Positives (TP)**: rappresentano le predizioni corrette sulle istanze positive del problema.

2. **False Positives (FP)**: rappresentano le predizioni che classificano un dato come appartenente alle istanze positive ma che in realtà appartiene alle istanze negative.

3. **True Negatives (TN)**: rappresentano le predizioni corrette sulle istanze negative del problema..

4. **False Negatives (FN)**: rappresentano le predizioni che classificano un dato come appartenente alle istanze negative ma che in realtà appartiene alle istanze positive.

Con questa suddivisione in gruppi possiamo successivamente ad andare a calcolare le metriche che daranno una stima delle capacità del modello. In questo caso di studio le metriche che sono state trattate sono:

- **Accuracy**

  È senza dubbio la metrica più diffusa e ci indica le predizioni corrette rispetto a tutte le istanze del dataset. In formula:

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

- **Precision**

  La precision ci indica la correttezza con la quale riusciamo a predire correttamente le istanze positive tra le sole istanze positive. In formula:

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

- **Recall**

  La recall indica quante istanze effettivamente positive sono state identificate correttamente. In formula:

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

- **F1-Score**

  Calcolato come combinazione di precision e recall, va ad effettuare una sorta di media delle due metriche. Nell'ambito dei modelli di classificazione binaria è senza dubbio la più usata. In formula:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

La definizione delle funzioni relative alle metriche di validazione è stata effettuata grazie all'utilizzo della libreria *tochmetrics* [31] per il caso classico e *scikit-learn* [32] per il caso quantistico, che forniscono una grande quantità di funzioni di validazione da utilizzare secondo le necessità di progetto.

CAPITOLO 4

Risultati

4.1 Risultati Grid-Search

I risultati relativi alla prima fase di validazione sono mostrati per tipo di computazione utilizzata. Nel caso della computazione classica, dato l'elevato numero di round di gridsearch effettuati, verranno mostrate le 4 combinazioni che hanno dato i risultati più equilibrati in ogni metrica di valutazione.

4.1.1 Computazione classica

Delle 24 combinazioni testate, utilizzando come dataset una combinazione dei tre, quelle che hanno ottenuto i risultati migliori sono le seguenti 4 triple di valori (funzione di perdita, optimizer, learning rate):

- **BCEWithLogitsLoss - AdamW - 3e-2**, nell'iterazione migliore ha ottenuto valori percentuali intorno al **85% - 87%** nelle metriche di accuracy, recall ed f1 nei dati di training. Le percentuali relative ai dati di test si aggirano intorno al **52% - 55%**.
- **BCEWithLogitsLoss - AdamW - 3e-3**, le cui percentuali sui dati di test sono leggermente inferiori rispetto a quelle precedenti ma che comunque si aggirano

intorno al **75%** in quasi tutte le metriche. Stesso discorso per i valori di test che si aggirano intorno al **50%**.

- **BCEWithLogitsLoss - Adagrad - 3e-3**, con il cambio di optimizer ha peggiorato drasticamente le prestazioni ottenendo solamente valori intorno al **49%** per i dati di train e **43%** per i dati di test.
- **BCEWithLogitsLoss - Adam - 3e-2**, ha ottenuto più o meno valori simili alla prima configurazione sia con i dati di train che con i dati di test.

Per una maggiore chiarezza dei risultati ottenuti in questa fase, le figure **4.1, 4.2, 4.3, 4.4** mostrano gli andamenti completi di ciascun addestramento sulle 100 epoche di train effettuate.

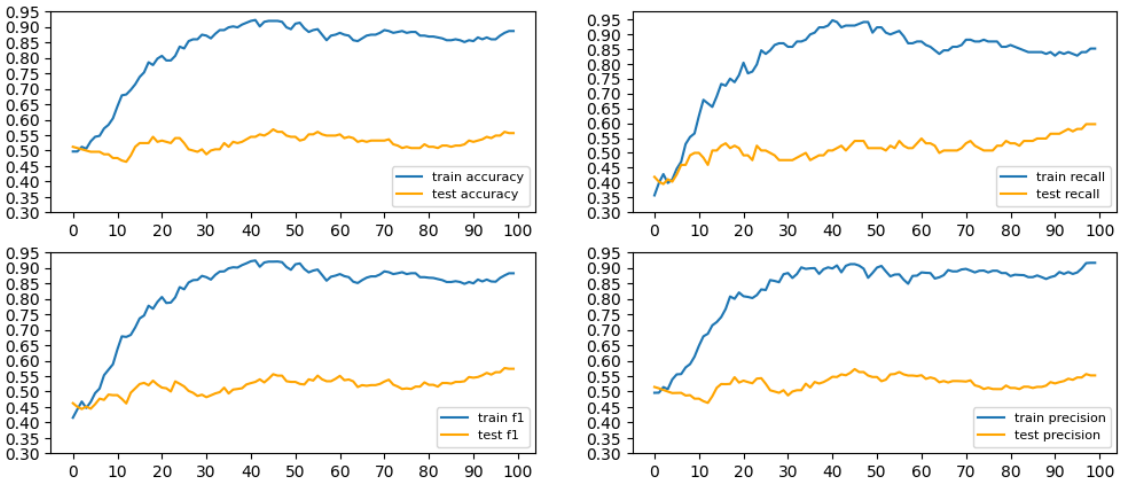

Figura 4.1: Andamento metriche di valutazione su 100 epochs per la tripla **BCEWithLogitsLoss - AdamW - 3e-2**

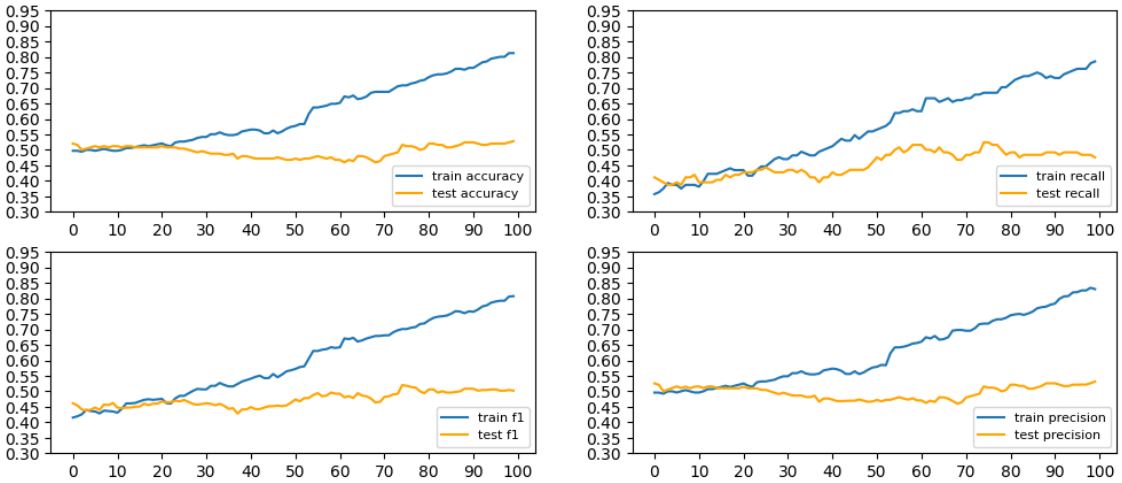

Figura 4.2: Andamento metriche di valutazione su 100 epochs per la tripla **BCEWithLogitsLoss - AdamW - 3e-3**

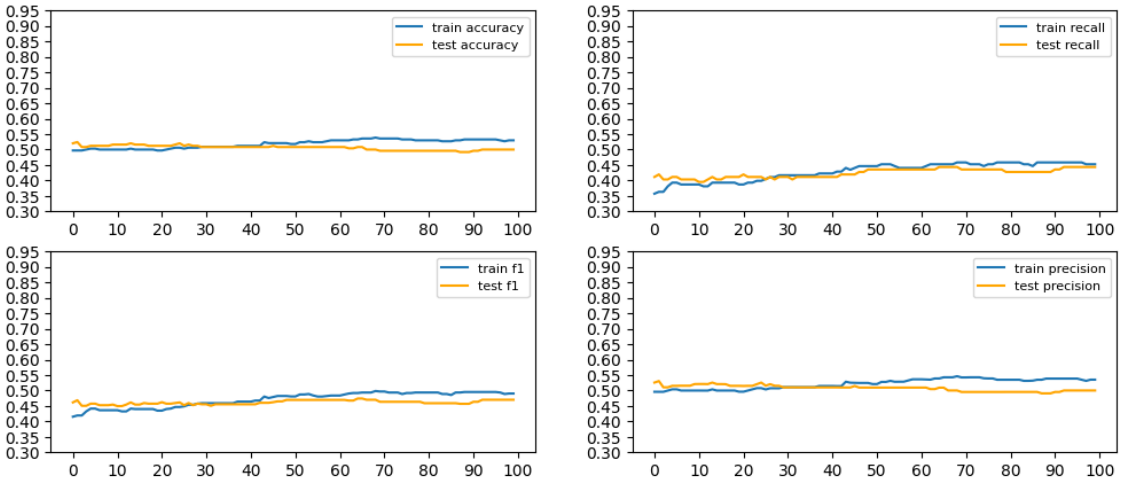

Figura 4.3: Andamento metriche di valutazione su 100 epochs per la tripla **BCEWithLogitsLoss - Adagrad - 3e-3**

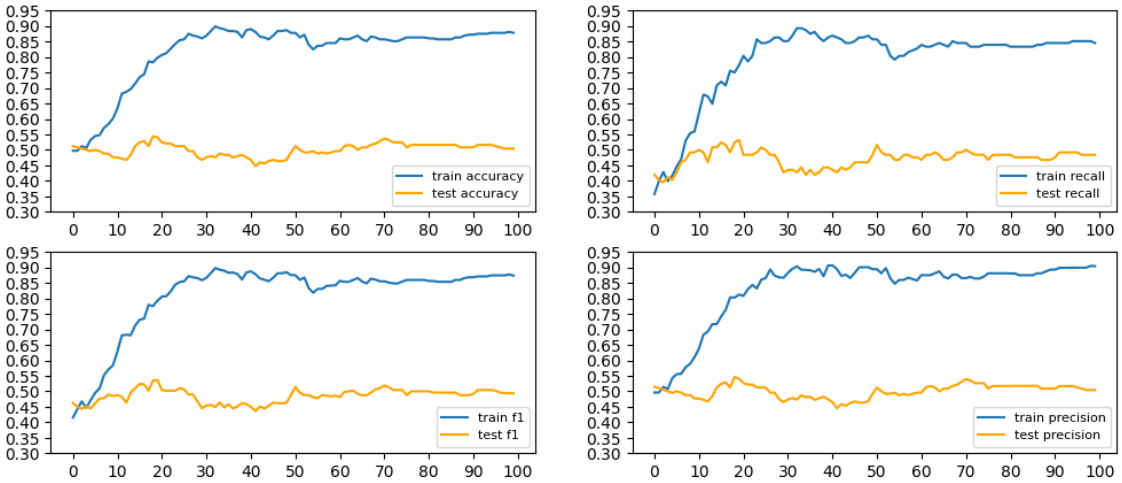

Figura 4.4: Andamento metriche di valutazione su 100 epochs per la tripla **BCEWithLogitsLoss - Adam - 3e-2**

Valori funzioni di perdita

La scelta delle combinazioni indicate nella sezione precedente sono giustificate anche dai risultati ottenuti nei loro valori di perdita sui dati di train e di test. Sebbene l'analisi dell'andamento dei valori di perdita ottenuto con la scelta effettuata mostra picchi in alcuni punti, i risultati ottenuti sono stati i migliori, come evidenziato dalle figure **4.5, 4.6, 4.7, 4.8**. Altre triple di valori, tendenzialmente quasi tutte quelle che hanno utilizzato la funzione **HingeEmbeddingLoss** per il calcolo della loss, presentano valori molto discontinui e molto spesso tendenti a valori negativi sempre più bassi.

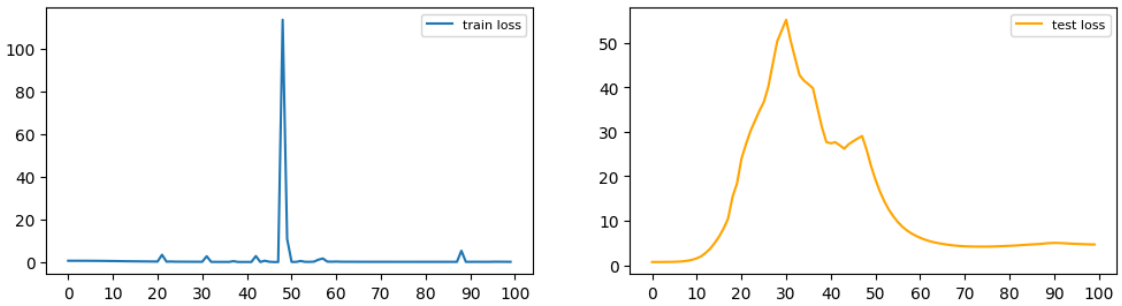

Figura 4.5: Andamento funzione di perdita su 100 epochs per la tripla **BCEWithLogitsLoss - AdamW - 3e-2**

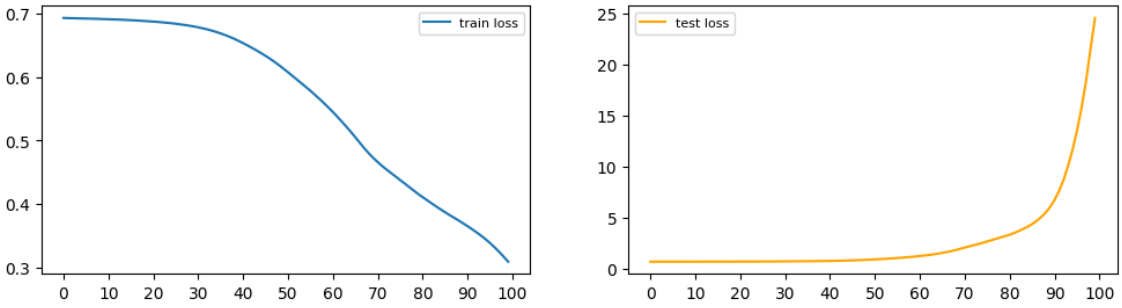

Figura 4.6: Andamento funzione di perdita su 100 epochs per la tripla **BCEWithLogitsLoss - AdamW - 3e-3**

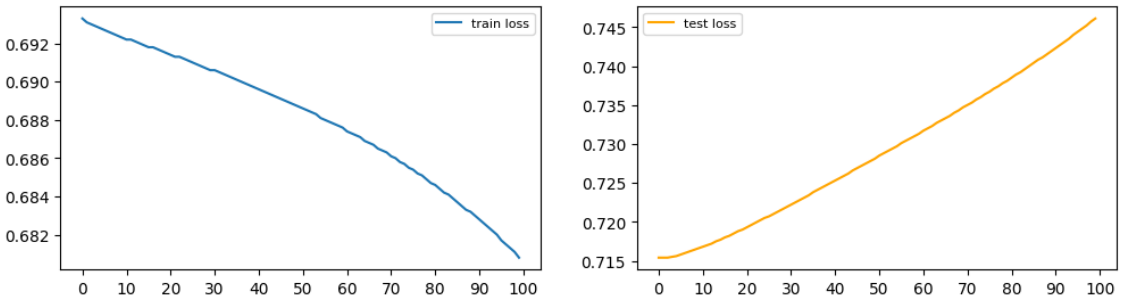

Figura 4.7: Andamento funzione di perdita su 100 epochs per la tripla **BCEWithLogitsLoss - Adagrad - 3e-3**

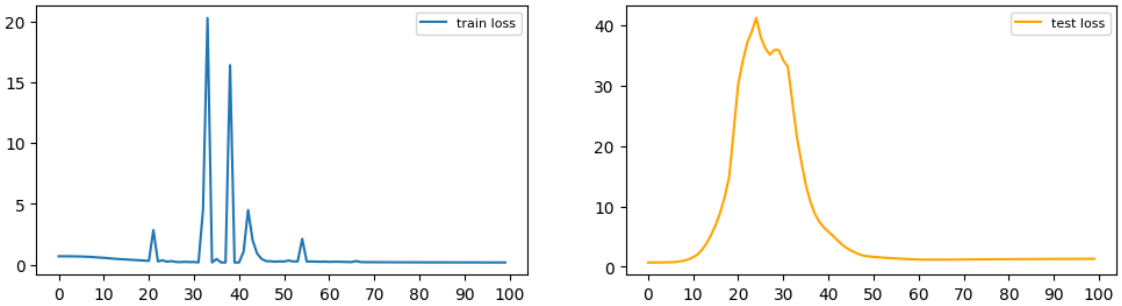

Figura 4.8: Andamento funzione di perdita su 100 epochs per la tripla **BCEWithLogitsLoss - Adam - 3e-2**

4.1.2 Computazione quantistica

Le coppie di valori testate in questa fase sono 3 e riguardano l'utilizzo del SPSAOptimizer e dei tre learning rates individuati. Dai test è emerso una minore capacità di predizione rispetto al modello classico, con valori nelle metriche di valutazione

che si aggirano intorno al **20% - 30%**, ad eccezione della precision che ha registrato valori compresi tra il **40% - 70%**. Le figure **4.9**, **4.10** e **4.11** mostrano gli andamenti delle metriche di ciascuna coppia di iperparametri utilizzati. Dai grafici è possibile notare che all'aumentare del parametro relativo al learning rate si verifica una sorta di appiattimento dei valori relativi a ciascuna metrica.

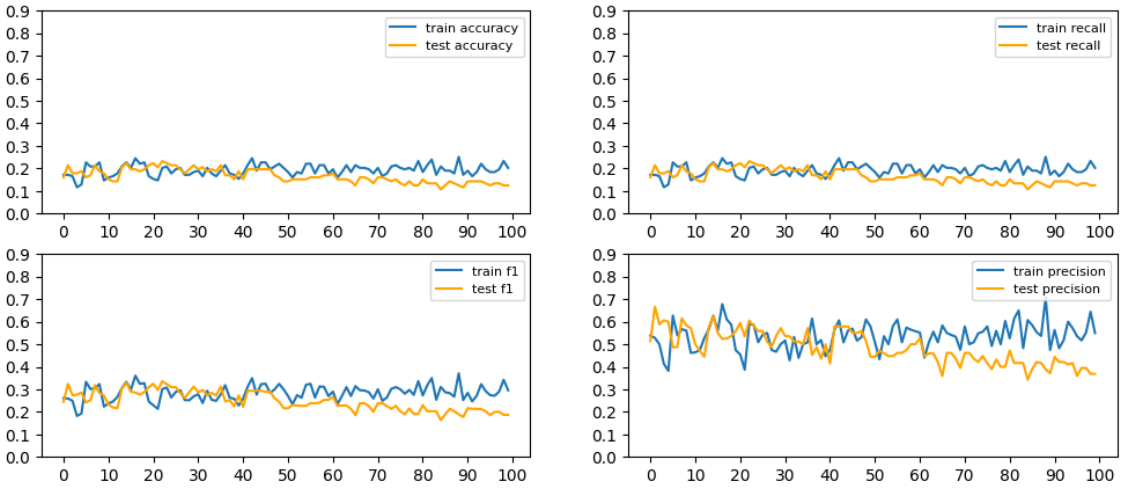

Figura 4.9: Andamento metriche di valutazione su 100 epochs per la coppia **SPSAOptimizer - 5e-1**

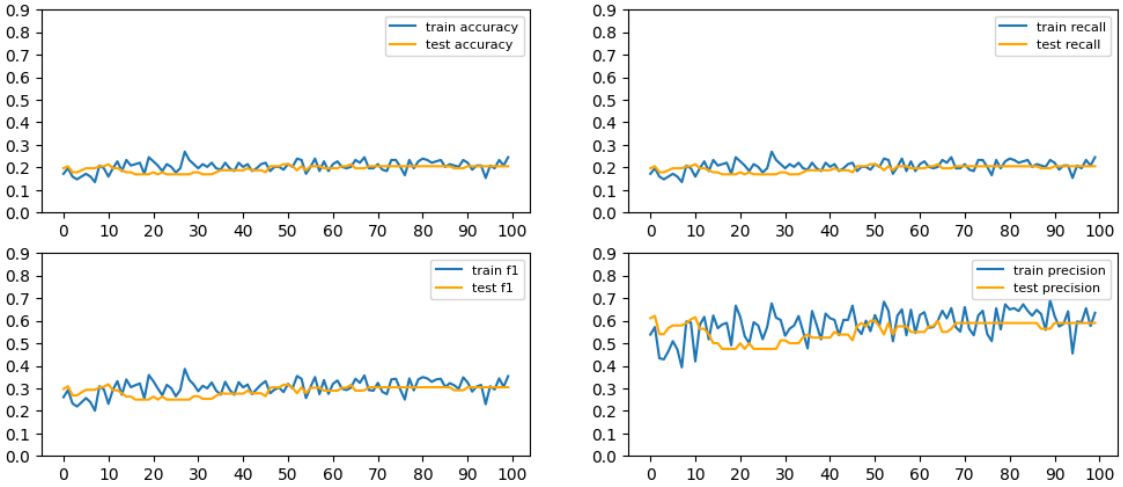

Figura 4.10: Andamento metriche di valutazione su 100 epochs per la coppia **SPSAOptimizer - 5e-2**

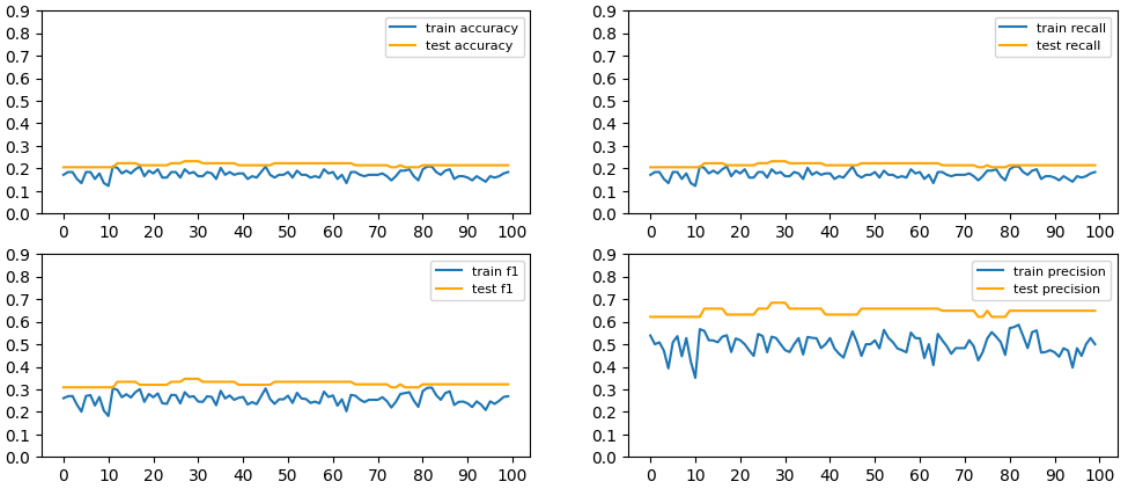

Figura 4.11: Andamento metriche di valutazione su 100 epochs per la coppia **SPSAOptimizer - 5e-3**

Valori funzioni di perdita

Anche i valori delle funzioni di perdita sono molto differenti rispetto a quelli ottenuti nel caso classico, infatti è possibile notare sia una maggiore variabilità dei risultati, come ad esempio nella figura **4.14**, sia valori oscillanti tra i dati di train e test rispettivamente, figure **4.12** e **4.13**.

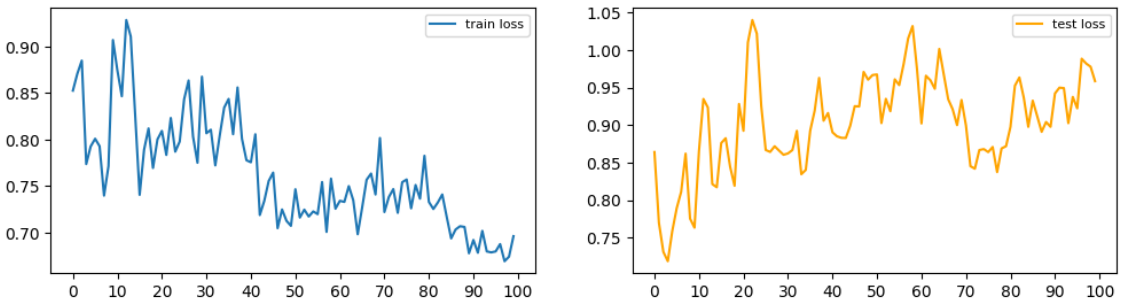

Figura 4.12: Andamento funzione di perdita su 100 epochs per la coppia **SPSAOptimizer - 5e-1**

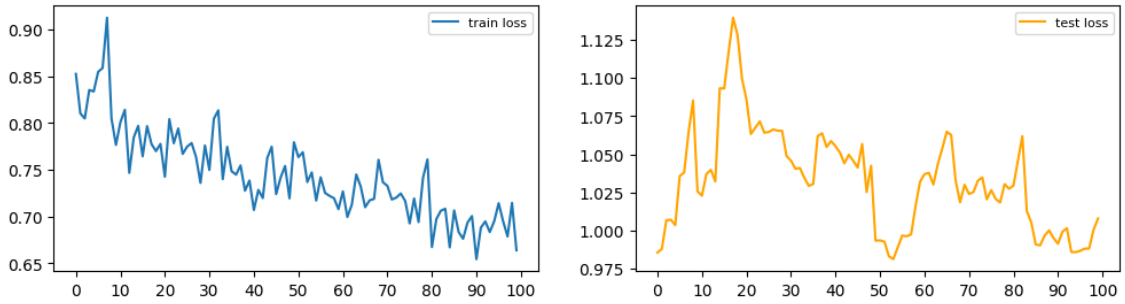

Figura 4.13: Andamento funzione di perdita su 100 epochs per la coppia **SPSAOptimizer - 5e-2**

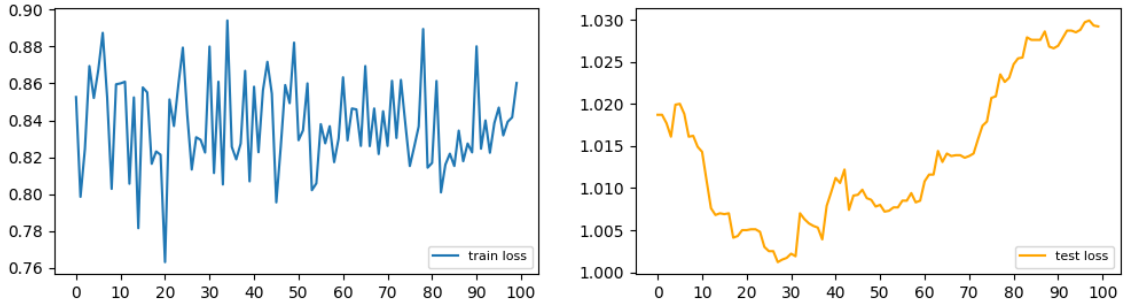

Figura 4.14: Andamento funzione di perdita su 100 epochs per la coppia **SPSAOptimizer - 5e-3**

4.2 Risultati K-Fold Cross-Validation

La seconda fase della validazione è stata effettuata seguendo due semplici step:

1. Selezione della tripla/coppia che ha ottenuto gli score migliori in fase di grid-search. Nel caso della pipeline classica è stata selezionata la tripla **BCEWithLogitsLoss - AdamW - 3e-2**, per la pipeline quantistica invece è stata scelta la coppia **SPSAOptimizer - 5e-2**.

2. Cross-Validation utilizzando tutte le combinazioni di dataset possibili, in modo tale da verificare in che misura cambiano i risultati e se la lunghezza e la complessità dei dataset incide sulle prestazioni del modello.

Le tabelle **4.1** e **4.2** mostrano i risultati medi ottenuti dai 5 fold per ogni combinazione di dataset utilizzata.

Dataset	Accuracy	Precision	Recall	F1-Score
GPS	0.4973	0.4945	0.4641	0.4753
CPN	0.4897	0.4896	0.4866	0.4877
ePurse	0.5036	**0.5014**	0.4363	0.4533
GPS + CPN	**0.5079**	0.1084	0.1015	0.1049
GPS + ePurse	**0.5100**	**0.5102**	**0.5478**	**0.5276**
CPN + ePurse	0.500	0.4997	**0.4999**	**0.4983**
ALL	0.4874	0.3870	0.3771	0.3818
Media	**0.4994**	**0.4272**	**0.4161**	**0.4184**

Tabella 4.1: Risultati medi ottenuti in seguito alla cross validation su pipeline classica

Dataset	Accuracy	Precision	Recall	F1-Score
GPS	**0.3326**	0.5956	**0.3326**	**0.4263**
CPN	**0.5131**	**0.7031**	0.5131	**0.5924**
ePurse	0.2746	**0.7733**	0.2746	0.4047
GPS + CPN	0.2951	0.4485	0.2951	0.3556
GPS + ePurse	0.2022	0.5114	0.2022	0.2891
CPN + ePurse	0.3148	0.5459	**0.3148**	0.3956
ALL	0.1090	0.2939	0.1090	0.1586
Media	**0.2916**	**0.4873**	**0.2916**	**0.3746**

Tabella 4.2: Risultati medi ottenuti in seguito alla cross validation su pipeline quantistica

Come si evince dalle tabelle **4.1** e **4.2**, i risultati ottenuti dalla cross-validation sono in linea con i risultati ottenuti nella prima fase della validazione con la gridsearch. Non sono sicuramente paragonabili ai risultati che si ottengono con i dati di esempio forniti dalla libreria a scopo didattico, ma c'è da dire che i dataset utilizzati in questa sperimentazione contengono dati molto più complessi da analizzare e classificare. Sebbene i risultati relativi alla pipeline classica si aggirano sul **50%** su quasi ogni

metrica quelli relativi alla pipeline quantistica non sono ancora dello stesso livello, fatta eccezione per alcuni valori che hanno toccato il **70%** in precision con alcuni dataset utilizzati. Alcune validazioni effettuate, in particolare quelle utilizzando il dataset **GPS + CPN** per la classica e **ALL** per la quantistica, presentano valori molto bassi in ogni metrica per errori dovuti alle funzione di perdita o crash inaspettati delle esecuzioni. I valori medi complessivi purtroppo non sono migliori infatti sono molto condizionati dai round di cross-validation che hanno ottenuto valori bassi nelle metriche a causa dei crash o dei valori di loss elevati. Un dato interessante da notare è il **48%** di precision ottenuto con il modello quantistico, che ha superato il modello classico che ha ottenuto solo il **42%** medio.

Fatte queste considerazioni è possibile rispondere alle domande di ricerca che ci siamo posti:

Risposta a RQ_1.
A fronte dei test e dei risultati ottenuti è possibile classificare con una pipeline classica i requisiti di sicurezza, prestando comunque attenzione ai risultati. I valori delle metriche di valutazione, infatti, non hanno registrato percentuali che lasciano la quasi certezza matematica di una predizione corretta per le istanze future che saranno fornite al modello.

Risposta a RQ_2.
Riguardo la pipeline quantistica, visti i risultati altalenanti e discontinui ottenuti, non è possibile ancora dire che è possibile classificare correttamente. Infatti con i dataset utilizzati e con la metodologia seguita non sono stati ottenuti i risultati sperati. Con alcuni accorgimenti e soprattutto con l'hardware adeguato si potranno ottenere sicuramente risultati migliori.

Risposta a RQ_3.
I modelli forniti dalla libreria sembrano essere migliori nella loro versione classica. Se paragonati con la versione quantistica risultano più equilibrati e meno soggetti ad errori o crash improvvisi. Inoltre la varietà di iperparametri disponibili per i modelli classici lascia spazio a numerosi tentativi e spunti per effettuare test diversi.

CAPITOLO 5

Conclusioni

L'dentificazione e l'analisi corretta dei NFRs è di fondamentale importanza per il corretto sviluppo di qualsiasi software. Una maggiore attenzione va senza dubbio ai requisiti di sicurezza che giocano un ruolo fondamentale per la stabilità e per la longevità del software. Risulta quindi necessario sviluppare ed affinare sistemi in grado di automatizzare questo processo di identificazione. Il lavoro svolto in questa tesi presenta un'analisi sulla libreria Lambeq per la classificazione dei requisiti a partire da dataset di uso comune per il testing di modelli. Le funzionalità ed i modelli offerti dalla libreria sono stati testati ed hanno prodotto dei risultati che lasciano spunti di miglioramento ed affinamento dei modelli stessi.
La pipeline classica testata è un buon punto di partenza e senza dubbio può essere migliorata, producendo comunque risultati equilibrati in ogni metrica di valutazione utilizzata (superando il 50% di score) e nella funzione di perdita.
La pipeline quantistica, salvo alcune eccezioni dove si sono presentati valori di precision oltre il 70%, risulta ancora non adatta per la classificazione di requisiti, a fronte dei test effettuati e dei dataset utilizzati. I risultati eterogenei nelle metriche e nei dataset lasciano spazio a lavori futuri di raffinamento del modello utilizzato.
Un altro aspetto da tenere in considerazione è l'hardware utilizzato per questo studio che senza dubbio ha giocato un ruolo importantissimo e che non è sicuramente adatto

a lavori di quantum computing. I miglioramenti che si possono fare su questo lavoro possono innanzitutto partire dall'utilizzo di un hardware appropriato, soprattutto per la pipeline quantistica, e più performante. In aggiunta, l'utilizzo di ulteriori dataset di requisiti con cui addestrare il SW basato su questa libreria potrà essere un contributo fondamentale per ottenere risultati ancora più accurati delle sue performances.

Bibliografia

[1] O. Elgabry, "Requirements engineering — an introduction to software requirements engineering." https://medium.com/omarelgabrys-blog/requirements-engineering-introduction-part-1-6d49001526d3, 2016. (Citato alle pagine 1 e 4)

[2] D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. de Felice, K. Meichanetzidis, S. Clark, and B. Coecke, "lambeq: An Efficient High-Level Python Library for Quantum NLP," *arXiv preprint arXiv:2110.04236*, 2021. (Citato alle pagine 2 e 11)

[3] S. Supakkul, T. Hill, L. Chung, T. T. Tun, and J. C. S. do Prado Leite, "An nfr pattern approach to dealing with nfrs," in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 179–188. (Citato a pagina 5)

[4] M. Glinz, "On non-functional requirements," in *15th IEEE international requirements engineering conference (RE 2007)*. IEEE, 2007, pp. 21–26. (Citato a pagina 5)

[5] 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)*. IEEE, 2005, pp. 373–382. (Citato a pagina 5)

[6] G. Sindre and A. L. Opdahl, "Capturing security requirements through misuse cases," *NIK 2001, Norsk Informatikkonferanse 2001, http://www. nik. no/2001*, vol. 74, 2001. (Citato a pagina 5)

[7] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020. (Citato a pagina 6)

[8] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011. (Citato a pagina 7)

[9] M. Schuld, I. Sinayskiy, and F. Petruccione, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015. (Citato alle pagine 7 e 8)

[10] C. Donalek, "Supervised and unsupervised learning," in *Astronomy Colloquia. USA*, vol. 27, 2011, p. 8. (Citato a pagina 7)

[11] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles, "Challenges and opportunities in quantum machine learning," *Nature Computational Science*, vol. 2, no. 9, pp. 567–576, 2022. (Citato a pagina 8)

[12] M. A. Haque, M. A. Rahman, and M. S. Siddik, "Non-functional requirements classification with feature extraction and machine learning: An empirical study," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*. IEEE, 2019, pp. 1–5. (Citato a pagina 9)

[13] 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)*. IEEE, 2013, pp. 9–16. (Citato a pagina 9)

[14] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2016, pp. 2027–2033. (Citato a pagina 10)

[15] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision tree analysis on j48 algorithm for data mining," *Proceedings of international journal of advanced research in computer science and software engineering*, vol. 3, no. 6, 2013. (Citato a pagina 10)

[16] 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)*. IEEE, 2014, pp. 183–192. (Citato a pagina 10)

[17] K. Chomboon, P. Chujai, P. Teerarassamee, K. Kerdprasop, and N. Kerdprasop, "An empirical study of distance metrics for k-nearest neighbor algorithm," in *Proceedings of the 3rd international conference on industrial application engineering*, vol. 2, 2015. (Citato a pagina 10)

[18] W. Wang, K. R. Mahakala, A. Gupta, N. Hussein, and Y. Wang, "A linear classifier based approach for identifying security requirements in open source software development," *Journal of Industrial Information Integration*, vol. 14, pp. 34–40, 2019. (Citato a pagina 10)

[19] V. Varenov and A. Gabdrahmanov, "Security requirements classification into groups using nlp transformers," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2021, pp. 444–450. (Citato a pagina 10)

[20] 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 10)

[21] S. Ganguly, S. N. Morapakula, and L. M. P. Coronado, "Quantum natural language processing based sentiment analysis using lambeq toolkit," in *2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*. IEEE, 2022, pp. 1–6. (Citato a pagina 11)

[22] A. Senokosov, A. Sedykh, A. Sagingalieva, and A. Melnikov, "Quantum machine learning for image classification," *arXiv preprint arXiv:2304.09224*, 2023. (Citato a pagina 11)

[23] S. Kavitha and N. Kaulgud, "Quantum machine learning for support vector machine classification," *Evolutionary Intelligence*, pp. 1–10, 2022. (Citato a pagina 11)

[24] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/google/jax (Citato a pagina 13)

[25] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens, "Supporting requirements engineers in recognising security issues," in *Requirements Engineering: Foundation for Software Quality: 17th International Working Conference, REFSQ 2011, Essen, Germany, March 28-30, 2011. Proceedings 17*. Springer, 2011, pp. 4–18. (Citato a pagina 14)

[26] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017. (Citato a pagina 17)

[27] B. Coecke, M. Sadrzadeh, and S. Clark, "Mathematical foundations for a compositional distributional model of meaning," *arXiv preprint arXiv:1003.4394*, 2010. (Citato a pagina 18)

[28] N. Khatri, "Experimental comparison of ansätze for quantum natural language processing." (Citato a pagina 22)

[29] R. Joseph, "Grid search for model tuning," https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e, 2018, published in Towards Data Science. (Citato a pagina 28)

[30] J. Brownlee, "A gentle introduction to k-fold cross-validation," https://machinelearningmastery.com/k-fold-cross-validation/, 2020. (Citato a pagina 29)

[31] N. S. Detlefsen, J. Borovec, J. Schock, A. H. Jha, T. Koker, L. D. Liello, D. Stancl, C. Quan, M. Grechkin, and W. Falcon, "Torchmetrics - measuring reproducibility in pytorch," *Journal of Open Source Software*, vol. 7, no. 70, p. 4101, 2022. [Online]. Available: https://doi.org/10.21105/joss.04101 (Citato a pagina 31)

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. (Citato a pagina 31)

Ringraziamenti

Ed eccoci arrivati a questa parte di tesi un po' più informale e molto più personale. Il giorno dell'immatricolazione al corso di laurea mi sembra di averlo vissuto il mese scorso ed invece sono già volati tre anni meravigliosi in cui ho avuto modo di conoscere professori e compagni di corso fantastici.

Innanzitutto ci tengo a ringraziare il prof. Fabio Palomba, che grazie alle sue interessanti lezioni di Fondamenti di Intelligenza Artificiale mi ha fatto avvicinare a questo campo dell'informatica. Ricordo che all'inizio del terzo anno la scelta del suo corso era stata una sorta di "ripiego" per un mancato tirocinio esterno, ma dopo poche lezioni mi sono subito reso conto di aver fatto la scelta giusta. Grazie per la sua disponibilità e professionalità mostrata sia ai corsi che nel periodo di tesi e tirocinio e grazie anche per i preziosi consigli mi ha saputo dare.

Un ringraziamento speciale va anche al dott. Francesco Casillo, che si è sempre reso disponibile fin dall'inizio del periodo di tirocinio e che mi ha dato preziosi consigli sia sugli approcci da utilizzare sia sulle metodologie per condurre il lavoro di tesi.

I prossimi ringraziamenti per i quali delle semplici parole non basteranno mai vanno alla mia famiglia. In particolare un grazie enorme e speciale a mia mamma e mio

padre, che da sempre mi sostengono e credono in me in qualsiasi cosa che faccio. Grazie per avermi cresciuto con amore e per avermi sempre permesso di coltivare tutte le passioni e gli hobby che ho sviluppato sin da piccolo. Grazie anche per aver sempre mostrato interesse in tutto quello che ho fatto, in particolare per l'università e per la materia che sto studiando. Grazie per avermi dato la possibilità di esplorare il mondo, per i viaggi che abbiamo fatto insieme e per tutti i bei momenti che abbiamo trascorso. Un grazie va anche ai miei nonni, che sin da piccolo mi hanno sempre sostenuto e cresciuto con amore. Nonostante l'età e la "lontananza" verso le tecnologie moderne non vi siete mai arresi nel cercare di capire cosa stessi studiando e cosa mi sarebbe piaciuto fare in futuro. Grazie per la pazienza che avete avuto da quando sono piccolo e per tutti i momenti in cui avete avuto bisogno del vostro "tecnico" per risolvere anche i problemi più semplici. Grazie a mia zia Mariella che facendo parte anche lei di questo campo mi ha sempre saputo dare consigli preziosi e aiuti in ogni cosa. Grazie anche per tutti i buoni consigli che mi hai dato nel corso degli anni e grazie per quelli che mi darai in futuro. Riassumendo, e parlando a tutti voi che siete la mia famiglia: GRAZIE, GRAZIE e ancora GRAZIE.

Un grazie enorme va alla mia fidanzata Grazia, che da 5 anni mi sopporta e mi supporta in ogni cosa che faccio. Posso dire che siamo cresciuti insieme sebbene ci conosciamo da relativamente poco tempo, se paragonato a quello che abbiamo vissuto. Ci supportiamo a vicenda in ogni scelta da sempre e insieme ci siamo sempre scambiati amore e affetto, anche nei momenti un po' più difficili. Spero di averti dato buoni consigli fino ad ora, come tu me ne hai dati in tutto questo tempo.

Uno ringraziamento doveroso va anche al mio amico Roberto, che conosco sin dai tempi del liceo. Sebbene non abbiamo frequentato la stessa scuola, grazie agli amici che abbiamo in comune ci siamo conosciuti e dopo poco abbiamo scoperto di avere tantissime passioni in comune, prima tra tutte quella per l'informatica. Grazie per tutti i consigli in ambito informatico che mi hai saputo dare e grazie per le avventure che abbiamo vissuto insieme, a partire dalle biciclettate pazze in mezzo ai boschi insieme a Michele, Agostino ed Armando, fino al viaggio in America che è stato l'apice delle nostre uscite.

Grazie a Michele, Armando, Agostino e di nuovo Roberto, con i quali ho trascorso interi pomeriggi su discord, giocando e facendo le cose più stupide insieme. Nel corso degli anni abbiamo conosciuto altri ragazzi con cui abbiamo trascorso intere giornate divertendoci e scherzando, ma alla fine siamo rimasti sempre noi quelli che hanno mantenuto un'amicizia più salda. Piano piano abbiamo fatto crescere sempre di più la nostra amicizia che spero continui a durare nel tempo, anche se la scelta di corsi di laurea e carriere lavorative diversi ci hanno ultimamente un po' giustamente allontanato.

Grazie anche ai nuovi amici, prima di tutto colleghi universitari, che in questi ultimi tre anni si sono aggiunti alla banda. In particolare grazie ad Anthony, con cui da subito ho legato sia per interessi comuni sia per modi di pensare. Grazie anche a Giovanni, Luca e Federico (anche se ci fa strano chiamarti così perchè per noi sei Palomba) che hanno portato sul server e nella vita un'ulteriore dose di pazzia e divertimento. Grazie per tutte le partite che abbiamo fatto e che spero continueremo a fare insieme. Grazie a tutti voi per i momenti divertenti che abbiamo trascorso insieme, un giorno anche se dovessimo prendere strade diverse spero che il legame che si è creato continui a rimanere forte.

Grazie a tutti i compagni di corso in particolare Carlo, Johnny, Ciro, Maria, Michele, Saso, Melania, Ermanno, Torz, David, Ghetty, Gigi e Ciccio. Voi siete i ragazzi con cui ho legato di più nel corso di questi tre anni e con cui ho passato tantissimi momenti di studio e anche di divertimento, soprattutto le pasquette di 20 e passa persone.

L'ultimo giro di ringraziamenti devo farli a me stesso. Il cambio di mentalità e di ambiente che ci sono tra il liceo e l'università li ho notati sin dai primi esami, che mi hanno fatto inizialmente vacillare e pensare di non essere all'altezza. Piano piano ho saputo trovare il mio metodo di studio e sono riuscito, anche grazie alle persone che mi sono vicine e che mi vogliono bene, a superare tutte le difficoltà che si sono presentate nella mia avventura in questa laurea triennale. Il ringraziamento più importante è quello di non aver mai mollato e di essere sempre riuscito a cavarmela,

a volte con non pochi problemi. Grazie anche per essere sempre stato me stesso e di non essermi mai fatto influenzare dalle idee degli altri. Sembra una cosa stupida, ma riflettendoci bene credo sia indice di una personalità forte, che però purtroppo non sempre mostro o mi rendo conto di avere.

Cosa mi aspetto adesso dal post-laurea? Non saprei veramente. Una cosa è certa: qualsiasi cosa decida di fare e soprattutto quale strada decida di intraprendere come laurea magistrale ed in futuro come lavoro, la prenderò in maniera diversa. Forte anche delle piccole esperienze mi raccomando da solo di prendere le mie scelte con decisione e con ottimismo verso il futuro. Penso possa bastare per ora, è stata dura trovare le parole adatte per ringraziare tutti, anche per non essere ripetitivo e banale. Concludo inserendo un'ulteriore frase che spero possa ispirare me e chiunque legga questa tesi:

"La vita è piena di scelte e incroci. Siamo noi a decidere quale strada prendere."