UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Costruzione di Modelli di Predizione di Vulnerabilità con Algoritmi Genetici: un'Indagine Preliminare

RELATORE

Prof. **Fabio PALOMBA**

SECONDO RELATORE

Dott. **Emanuele IANNONE**

Università degli studi di Salerno

CANDIDATO

Alfonso CANNAVALE

Matricola: 0512108068

Anno Accademico 2021-2022

"Follia è fare sempre la stessa cosa aspettandosi risultati diversi"

Albert Einstein

Sommario

Gli applicativi (o software) informatici sono soggetti a vulnerabilità causate da errori commessi nelle fasi che costituiscono il ciclo di vita del software. Tali errori rappresentano un pericolo per tutti i sistemi informatici nonché per la sicurezza dei dati. Nella società moderna, il numero di persone che fruisce di prodotti software è notevolmente cresciuto, costringendo gli sviluppatori e gli ingegneri del software ad accelerare le varie fasi dello sviluppo, ciò comporta una minore attenzione per l'analisi delle rilevazioni di eventuali vulnerabilità. Tutto questo ha portato ad un aumento dei rischi nella vita digitale degli utenti ed è diventato fondamentale, quindi, identificare eventuali vulnerabilità per proteggere la sicurezza di tutti. Esistono diverse tecniche per l'identificazione delle vulnerabilità, tra le quali è presente una tecnica incentrata sull'utilizzo di Vulnerability Prediction Models, ovvero l'utilizzo di algoritmi di apprendimento automatico basato su metriche e attributi software.

Questo lavoro di tesi si concentra sull'utilizzo di una tecnica di identificazione delle vulnerabilità basata su algoritmi genetici, in particolare l'obiettivo è quello di determinare in che modo l'utilizzo di algoritmi genetici, può influenzare le prestazioni dei vulnerability prediction models. Per raggiungere questo obiettivo, è stato implementato un tool che tramite l'utilizzo di algoritmi genetici definisce un Decision Tree che classifica se un commit introduce o meno una vulnerabilità all'interno del software.

I risultati ottenuti da questo lavoro hanno evidenziato come l'utilizzo di algoritMi genetici, con determinati parametri, abbia portato alla definizione di modelli con prestazioni leggermente più elevate rispetto i modelli definiti in modo tradizionale. La scelta dei parametri e della funzione obiettivo dell'algoritmo genetico gioca un ruolo cruciale, da queste scelte dipendono infatti le metriche di perfomance dei modelli definiti tramite GA.

Ulteriori studi dovrebbero concentrarsi sull'introduzione di individui diversi dal decision tree e sull'analisi delle strutture dei modelli ottenuti tramite algoritmi genetici.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

In questo capitolo si fornisce una panoramica del lavoro. In primo luogo, viene introdotto il contesto in cui si opera e le motivazioni della ricerca. Poi, sono presentati gli obiettivi insieme ad una breve discussione sui risultati ottenuti. Infine, si spiega la struttura di questa tesi.

1.1 Contesto applicativo

Il problema della sicurezza informatica, per i sistemi di calcolo, risale ai primi anni '60; ma il problema della *"sicurezza delle informazioni"* è antico quanto l'uomo. Inizialmente questo problema riguardava prettamente il settore militare, oggigiorno la sicurezza delle informazioni riguarda ogni settore.
La sicurezza informatica è la pratica di proteggere i sistemi informatici e le infomazioni sensibili da attacchi digitali, che possono causare danni ingenti. Con l'incremento dei servizi digitale, la nostra società si affida sempre di più al software. Questo utilizzo massiccio da parte di tutta la popolazione mondiale del software, porta con sé dei rischi per la sicurezza associati alla presenza di vulnerabilità nel software. A titolo di esempio, consideriamo l'attacco *WannaCry*, un attacco informatico mondiale del maggio 2017 che ha preso di mira i computer che eseguono il sistema operativo Microsoft Windows crittografando i dati e richiedendo pagamenti di riscatto nella criptovaluta Bitcoin. In un giorno è stato riferito che il codice ha infettato più di 230.000 computer in oltre 150 paesi, sfruttando una vulnerabilità del sistema operativo Microsoft Windows [1].

É fondamentale, quindi, capire cosa sia una vulnerabilità e soprattutto capire come identificarla. Le vulnerabilità informatiche sono mal funzionamenti, configurazioni sbagliate o semplicemente errori presenti in un sistema che lo espongono a dei rischi; la loro presenza rende un sistema vulnerabile ed esposto ad attacchi. Il mondo delle vulnerabilità in informatica è un settore in continua espansione e che riguarda ogni singola componente di un sistema informatico, quindi l'identificazione e la gestione delle vulnerabilità è un aspetto fondamentale della sicurezza.

Attualmente per l'identificazione delle vulnerabilità esistono diverse tecniche, tra le quali:

- Analisi Statica: revisione del codice sorgente;
- Analisi Dinamica: osservazione del comportamento del software in esecuzione;
- Rilevamento automatico delle vulnerabilità: dove è previsto l'utilizzo di modelli di apprendimento automatico basati su attributi software, chiamati *Vulnerability Prediction Models*.

Si ritiene che i modelli di previsione delle vulnerabilità (VPMs), siano promettenti per fornire agli ingegneri del software una guida su dove dare la priorità alle preziose risorse di verifica per l'identificazione di esse [2]. Tuttavia, gli studi si concentrano per lo più sull'individuazione delle variabili indipendenti correlate alla presenza di vulnerabilità, sulle metriche da utilizzare; ma non sono stati individuati studi che si concentrassero sulla definizione di tali modelli tramite l'utilizzo di algoritmi di ottimizzazione come gli algoritmi gentici.

1.2 Motivazioni ed Obiettivi

I Vulnerability Prediction Models sono modelli di apprendimento automatico basati su metriche e attributi software, ed il loro utilizzo facilita l'identificazione delle vulnerabilità. Ad oggi, la definizione dei modelli predittivi utilizzati per l'identificazione delle vulnerabilità, avviene solo tramite l'utilizzo di apposite framework, come *Scikit-learn*, *TensorFlow*, *Kersa*, etc. Non sono stati individuati studi relativi all'utilizzo di algoritmi di ottimizzazione, come gli algoritmi genetici, per la definizione di VPMs; proprio per questo lo studio si concentrerà sull'utilizzo di tali algoritmi per la definizione di questi modelli. L'obiettivo principale della tesi è quello di implementare un tool che tramite l'utilizzo di algoritmi genetici porti alla definizione di VPMs.
Questo tipo di approccio offrirebbe:

- un'esplorazione, in maniera rapida, di gran parte dello spazio di ricerca (quindi esplorazione di tutte le possibili definizione del modello predittivo);
- una tecina di ricerca veloce, che garantirebbe risultati sub-ottimali (vicini all'ottimo) in tempi ragionevoli.

Lo scopo del presente lavoro di tesi è quello di confrontare i modelli predittivi definiti tramite gli algoritmi genetici, con i modelli di apprendimento definiti in modo tradizionale ed individuare la tecnica che fornisce i risultati migliori.

1.3 Risultati

I risultati dello studio rivelano che è possibile utilizzare gli algorimti genetici per la definizione di modelli predittivi e che, a seconda di alcuni parametri, è possibile ottenere anche dei risultati migliori rispetto alle tradizionali costruzioni di questi modelli. Infatti, è possibile osservare come l'utilizzo dell'algorimto genetico ha portato alla definizione di un Vulnerability Prediction Model con prestazioni leggermente più elevate rispetto al modello definito tramite CART. É importante evidenziare che questo risultato è stato ottenuto ottimizzando la F-measure. I risultati ottenuti ottimizzando le altre metriche (precision, recall e accuracy) hanno portato alla definizione di modelli con prestazioni più basse rispetto al modello definito tradizionalmente.

Questo lavoro dovrebbe essere ampliato, un possibile ampliamento potrebbe essere apportato analizzando l'utilizzo di algorimti genetici per la definizione di altri modelli predittivi diversi dal Decision Tree. Inoltre, negli studi successivi potrebbero essere confrontate le strutture dei modelli per analizzare e comprendere le differenze.
In sintesi, il lavoro di tesi ha fornito i seguenti contributi:

- studio e analisi di una tecnica di identificazione di vulnerabilità tramite l'utilizzo di algoritmi genetici;
- un tool che è possibile utilizzare per la definizione di modelli predittivi (Decision Tree) tramite l'utilizzo di algoritmi genetici.

1.4 Struttura della tesi

La Sezione 2 illustra la letteratura correlata all'identificazione delle vulnerabilità e approfondisce degli aspetti relativi ai concetti trattati nella tesi. La Sezione 3 riporta la metodologia e le tecniche utilizzate per raggiungere gli obiettivi della tesi, mentre la Sezione 4 riporta i risultati ottenuti dalle varie esecuzioni. La sezione 5 conclude la tesi con l'analisi dei risultati e alcune riflessioni.

CAPITOLO 2

Stato dell'arte

Questo capitolo illustra lo stato dell'arte e i lavori presenti in letteratura sugli aspetti di ricerca trattati nello studio dell'utilizzo di algoritmi di apprendimento per l'identificazione di vulnerabilità del software.

2.1 Che cos'è una vulnerabilità

Quando gli errori del software mettono a rischio la sicurezza dei nostri dati allora si parla di **vulnerabilità** del software e bisogna fare molta attenzione.

La sicurezza del software gioca un ruolo cruciale nello sviluppo del software moderno [3], e rappresenta l'idea di progettare il software in modo che continui a funzionare correttamente sotto un attacco malevolo [4].
La necessità di lanciare un nuovo prodotto o servizio battendo sul tempo la concorrenza spesso ne compromette la qualità, il funzionamento e la sicurezza, che sono elementi fondamentali. La richiesta di software sicuri è in aumento, sia per una maggiore consapevolezza da parte dell'utente finale sia per la capacità delle aziende di stimare i costi derivanti da cybercrimini o da malfunzionamenti software.
Le organizzazioni più attente sono consapevoli che sia più proficuo prevenire l'errore e il danno, piuttosto che correggerlo o ripararlo [5].

La vulnerabilità può essere intesa come una componente (esplicita o implicita) di un sistema, in corrispondenza alla quale le misure di sicurezza sono assenti, ridotte o compromesse, il che rappresenta un punto debole del sistema e consente ad un eventuale aggressore di compromettere il livello di sicurezza dell'intero sistema.
Il codice sorgente dovrebbe essere progettato per essere resiliente agli attacchi esterni: sfortunatamente, le vulnerabilità del software rappresentano minacce alla sicurezza che possono potenzialmente essere sfruttate da esterni per causare perdita di dati, escalation di privilegi, condizioni di gara e altri effetti indesiderati che possono colpire il codice sorgente [6].
Parlando di vulnerabilità, risulta molto complesso stabilire l'origine di ognuna di esse. Possiamo tuttavia dire che esistono 3 macro-categorie nelle quali catalogare le vulnerabilità in informatica:

1. **Vulnerabilità software**: anche definita *bug software*, si presenta ovunque ci sia un difetto di progettazione, codifica, installazione e configurazione del software. Sono letteralmente mal funzionamenti e errori di scrittura del software.

2. **Vulnerabilità dei protocolli**: si manifestano quando i protocolli di comunicazione non contemplano il problema legato alla sicurezza; l'esempio classico di vulnerabilità consiste nel permettere una connessione in chiaro (non crittografata) consentendo a possibili malintenzionati di intercettare le informazioni scambiate (**eavesdropping**) (es. Telnet).

3. **Vulnerabilità hardware**: parliamo di vulnerabilità degli apparati hardware quando qualsiasi elemento causa un'oggettivo pericolo al corretto funzionamento di una macchina.

Si discuterà nel dettaglio delle *vulnerabilità software*.

2.1.1 Vulnerabilità software

Le **vulnerabilità software** vengono catalogate nel *National Vulnerability Database* (NVD), all'interno del quale sono presenti database di riferimenti a elenchi di controllo di sicurezza, difetti software relativi alla sicurezza, configurazioni errate, nomi di prodotti e metriche di impatto.
Alcuni noti esempi di vulnerabilità software:

- **CodeRed**: rilevato, per la prima volta, il 17 luglio del 2001; secondo alcune stime, tale worm ha complessivamente infettato circa 300.000 computer, impedendo, di fatto, ad una moltitudine di imprese, di condurre le normali attività di business; in tal modo, esso ha causato considerevoli danni finanziari ad un elevato numero di società, ubicate in vari paesi.
  Nonostante Microsoft abbia rilasciato, assieme al Security Bulletin MS01-033, un'apposita patch destinata a chiudere la vulnerabilità utilizzata dal suddetto worm di rete, alcune versioni di *CodeRed* continuano tuttora a diffondersi attraverso Internet.

- **Spida**: worm di rete individuato quasi un anno dopo la comparsa di *CodeRed*, utilizzava, per la propria diffusione, un'exposure in MS SQL. Alcune installazioni standard del server MS SQL, in effetti, non proteggevano tramite apposita password l'account di sistema "SA", permettendo così a chiunque avesse accesso al sistema attraverso la rete di eseguire comandi arbitrari all'interno dello stesso.
  Utilizzando tale esposizione, il worm configura l'account "Guest" in maniera tale da ottenere il pieno accesso ai file custoditi nel computer; in seguito, esso provvede all'upload di se stesso sul server da infettare.

- **Slammer**: rilevato alla fine del mese di gennaio 2003, si avvaleva di un metodo ancor più semplice per generare l'infezione informatica sui computer provvisti di sistema operativo Windows e relativo server MS SQL; si trattava, nella fattispecie, dello sfruttamento di una vulnerabilità a livello di buffer overflow in una delle subroutine di elaborazione dei pacchetti UDP.
  Secondo alcune stime, il worm *Slammer* ha infettato all'incirca 75.000 computer in tutto il mondo nei primi 15 minuti in cui si è sviluppata l'epidemia informatica in questione.

Un noto esempio di vulenrabilità software ha interessato la libreria Java *log4j (versione 2)*. Questa vulnerabilità, anche nota come *"Log4Shell"*, consentiva l'esecuzione di codice in modalità remota non autenticata ed è stata pubblicata nel NVD come CVE-2021-45046.

È riportato di seguito un codice di esempio:

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class VulnerableLog4jExampleHandler implements HttpHandler {

  static Logger log = LogManager.getLogger(VulnerableLog4jExampleHandler.
     class.getName());

  /**
   * Un semplice endpoint HTTP che legge l'intestazione x-api-version
      della richiesta e la registra.
   * Questo e' uno pseudo-codice per spiegare la vulnerabilita', e non un
      esempio completo.
   * @param he HTTP Request Object
   */
  public void handle(HttpExchange he) throws IOException {
    String apiVersion = he.getRequestHeader("X-Api-Version");

    // Questa linea innesca l'RCE (Remote Code Execution) registrando l'
       intestazione HTTP controllata dall'attaccante.
    // L'attaccante puo' impostare l'intestazione X-Api-Version su: ${jndi
       :ldap://some-attacker.com/a}
    log.info("Requested␣Api␣Version:{}", apiVersion);

    String response = "<h1>Hello␣from:␣" + apiVersion + "!</h1>";
    he.sendResponseHeaders(200, response.length());
    OutputStream os = he.getResponseBody();
    os.write(response.getBytes());
    os.close();
  }
}
```

Un attaccante sfrutta semplici richieste http, verso un sito web, per iniettare un input non ben validato all'interno del messaggio di log, ovvero un tag appositamente forgiato che impone una specifica interpretazione da parte della libreria. Il tag in oggetto è chiamato JNDI (Java Naming and Directory Interface) e può essere utilizzato per caricare codice da remoto. Nell'esempio è riportato un attacco con iniezione di un RCE (Remote Control Execution) malevolo, che verrà automaticamente eseguito sul sistema vittima e quindi sul server web; questo permetterà all'attacante di eseguire in remoto codice dannosso sul server web ottenendone il pieno controllo.

Tra gli attacchi alle app Web, Remote Code Execution (RCE) è una delle minacce più dannose; infatti Secondo Open Web Application Security Project (OWASP), RCE è il problema di sicurezza PHP più diffuso dal luglio 2004 e quindi è stato classificato come la minaccia numero uno nell'elenco dei problemi di sicurezza delle app Web [7].

2.2 Identificare vulnerabilità

Un software sicuro richiede tecniche efficaci per il rilevamento delle vulnerabilità durante il suo ciclo di sviluppo. La pratica di rilevare le falle di sicurezza prima della fase di implementazione mitiga i rischi che le vulnerabilità possono imporre.
Per mitigare le minacce alle applicazioni Web esistono delle soluzioni lato client, come ad esempio Noxes [8] che mitiga attacchi di scripting cross-site; e delle soluzioni lato server, che hanno il vantaggio di scoprire una gamma più ampia di vulnerabilità.
Le tecniche di *detection vulnerability* lato server possono essere classificate in:

- **Analisi Statica**: prevede la revisione del codice sorgente o binario eliminando la necessità di eseguirlo.
  L'analisi statica è semplice, veloce e può essere efficace per trovare bug nel codice [9]. Tuttavia, le analisi statiche sono piuttosto imprecise e generano enormi falsi positivi e falsi negativi [10].

- **Analisi Dinamica**: implica l'esecuzione del software.
  Il problema dei falsi positivi e negativi è minore nel caso dell'analisi dinamica perché analizzano il software eseguendo i casi di test. Ma questo approccio richiede un gran numero di casi di test per garantire un certo livello di confidenza nel rilevamento dei bug di sicurezza [11].

La maggior parte degli approcci sono basati sull'analisi statica del codice sorgente e/o sull'analisi dinamica e sul fuzz-testing [12].

2.2.1 Vulnerability Prediction Models

Le soluzioni attuali sono ancora raramente efficaci nella pratica poiché soffrono di alti tassi di falsi positivi e/o problemi di scalabilità [13; 14]. Per questo motivo sono stati introdotti approcci per il rilevamento automatico delle vulnerabilità software con machine learning [15; 16], che hanno portato alla nascita dei ***Vulnerability Prediction Models*** *(VPMs)*.
I **Vulnerability Prediction Models** (VPMs) sono un approccio per dare priorità alle ispezioni di sicurezza e agli sforzi dei test per trovare e correggere le vulnerabilità [17], sono infatti responsabili dell'identificazione di componenti software (ad es. classi) che possono contenere vulnerabilità critiche. I risultati dei modelli di previsione della vulnerabilità sono estremamente utili per sviluppatori e project manager, in quanto consentono loro di dare priorità, e concentrare le risorse disponibili, ai casi di test delle aree ad alto rischio (cioè potenzialmente vulnerabili).

I VPMs sono principalmente modelli di apprendimento automatico basati su attributi software recuperati principalmente dal codice sorgente del software analizzato (ad es. metriche del software, caratteristiche del testo, ecc.) e sono stati creati sulla base di una varietà di metriche e approcci [17].
Gli studi relativi alla costruzioni di modelli di previsione delle vulnerabilità del software si concentrano essenzialmente sull'individuazione delle variabili indipendenti correlate alla presenza di vulnerabilità. Quasi tutti i lavori hanno coinvolto metriche come le dimensioni dei file sorgenti o binari (ad esempio, le linee di codice) o le metriche strutturali.
Numerosi studi hanno affermato che i VPMs costruiti usando metriche strutturali come predittori, relative a complessità, accoppiamento e coesione (CCC), hanno raggiunto risultati più precisi riguardo all'identificazione di vulnerabilità [18].

La previsione della vulnerabilità è stata basata su diversi modelli di apprendimento automatico supervisionato, come Decision Trees, Support Vector Machines (SVM), Naïve Bayes e Random Forests.

2.2.1.1 VPMs con granularità a livello di file

La maggior parte degli studi sono stati condotti su VPMs che fanno previsioni a livello di file sorgente [18; 19; 20; 21]. I VPMs con granularità a livello di file classificano le componenti del software (come moduli, file ecc.) in due classi: vulnerabile e pulito. Questi tipi di modelli sono addestrati con vulnerabilità note e con le loro caratteristiche. I dettagli delle vulnerabilità con le loro caratterstiche sono raccolti dal ***National Vulnerability Database (NVD)***.
Per la costruzione dei VPMs vengono estrarre le caratteristiche dal software sviluppato in passato, come ad esempio le metriche di costruzione, che poi vengono mappate alle vulnerabilità corrispondenti (vulnerabili o meno). Le caratteristiche del software agiscono come input del modello di predizione durante i test.
I modelli di predizioni si basano principalmente sulle metriche del software o sull'analisi del testo, alcuni invece utilizzano una combinazione di analisi del testo e metriche del software. La tabella 2.1 elenca alcuni dei lavori nel campo della previsione delle vulnerabilità.

Tabella 2.1: Confronto con i lavori precedenti riguardanti i modelli di previsione della vulnerabilità. L'attenzione si concentra sul livello di granularità (cioè, i componenti che sono soggetti alle previsioni).

Studio	Granularità
Neuhaus et al. [22]	Funzioni
Zimmermann et al. [23]	File
Binario *Sultana et al. [24]*	Classi/Metodi
Theisen e Williams [17]	File
Nguyen e Tran [25]	File
Chowdhury et al. [26]	File
Shin et al. [27]	File
Scandariato et al. [28]	File
Zhang et al. [29]	File
Perl et al. [30]	Commit
Yang et al. [31]	Commit

2.2.1.2 VPMs just-in-time

La maggior parte dei VPMs, come discusso nella sezione precedente, rilevano vulnerabilità solo dopo il rilascio del software, tuttavia, come riportato da Lomio et al. [6], è stato dimostrato che un'identificazione a livello di commit si adatta meglio alle esigenze degli sviluppatori che potrebbero così accellerare la loro risoluzione. La tabella 2.2 riporta le variabili indipendenti su cui lo studio *Just-in-Time Software Vulnerability Detection: Are We There Yet?*[6] si è concentrato per la costruzione del modello predittivo. Il loro studio ha evidenziato che la combinazione di più metriche non migliora necessariamente la capacità di classificazione. Inoltre, è stato dimostrato che gli algoritmi di apprendimento di base raramente funzionano bene.

Tabella 2.2: Elenco delle metriche utilizzate come variabili indipendenti (caratteristiche) per i machine learners. La tabella riporta una descrizione e la logica dietro la loro selezione

Nome	Descrizione	Motivazione
Linee Aggiunte	Il numero di linee di codice aggiunte nel commit.	Un'elevata quantità di linee di codice aggiunte indica un grande commit, che ha un rischio maggiore di introdurre difetti o vulnerabilità.
Linee Cancellate	Il numero di linee di codice rimosse nel commit.	Uguale a *Linee Aggiunte*.
Metodi Aggiunti	Il numero di nuove funzioni/metodi aggiunti nel commit.	Nuove funzioni o metodi possono aggiungere nuovi controlli di sicurezza o aumentare la superficie di attacco.
Metodi Rimossi	Il numero di funzioni/metodi rimossi nel commit.	L'eliminazione di funzioni o metodi critici per la sicurezza possono rimuovere i controlli di sicurezza o ridurre la superficie di attacco.
Metodi Modificati	Il numero di funzioni/metodi modificati nel commit.	La rimozione di funzioni o metodi critici per la sicurezza può modificare il profilo di sicurezza.
Condizioni Aggiunte	Il numero di espressioni condizionali (es. if) aggiunte nel commit.	Uguale a *Metodi Aggiunti*.
Condizioni Rimosse	Il numeor di espressioni condizionali rimosse nel commit.	Uguale a *Metodi Rimossi*.
Chiamate Metodi Aggiunte	Il numero di chiamate di funzioni o metodi aggiunti nel commit.	Uguale a *Metodi Aggiunti*.
Chiamate Metodi Rimosse	Il numero di chiamate di funzioni o metodi rimossi nel commit.	Uguale a *Metodi Rimossi*.

2.3 Research-Based Machine Learning

Per quanto ne sappiamo, non ci sono studi relativi all'utilizzo di algoritmi di ricerca per la definizione di modelli predittivi; per questo motivo la mia ricerca mira a far luce sull'utilizzo di algoritmi genetici per la definizione di modelli predittivi con lo scopo di studiare il comportamento di tali algoritmi in questo ambito.
In particolare mi concentrerò sulla creazione di un tool a supporto della definizione del modello predittivo ideato nell'articolo di Lomio et al.[6], con l'obiettivo di analizzare le prestazioni di tale modello.

CAPITOLO 3

Una Tecnica di Identificazione di Vulnerabilità basata su Algoritmi Genetici

Il presente capitolo tratterà un'approccio basato sull'utilizzo di algoritmi genetici adottato per la costruzione di modelli predittivi, quali i *Decision Tree*, per l'identificazione di vulnerabilità software.

3.1 Gli algoritmi evolutivi

Dai fondamenti della *teoria dell'evoluzione*, dove organismi della medesima specie **evolvono** tramite un processo che prende il nome di **selezione naturale**, nascono quelli che sono gli **algoritmi evolutivi *(EAs - Evolutionary Algorithms)***, questi tipi di algoritmi di ottimizzazione sono ispirati, nella loro struttura, ai meccanismi della teoria dell'evoluzione. Gli ***algoritmi evolutivi*** sono strategie **meta-euristiche** che imitano i processi di evoluzione naturale per risolvere problemi di ricerca globale. In generale, i pilastri fondamentali su cui si basano queste tecniche sono i seguenti:

- riproduzione: è il processo di generazione di nuovi elementi di una popolazione a partire dai genitori;
- variazione: è una sorta di mutazione, spesso inattesa e casuale che si ha nel processo di generazione della nuova progenie;
- competizione e selezione: la *competizione* tra i nuovi elementi e la *selezione* dei migliori sono i processi inevitabili di sopravvivenza in un ambiente con risorse limitate.

3.1.1 Gli algoritmi genetici

Gli **algoritmi genetici *(GAs - Genetic Algorithms)*** sono il tipo più noto ed utilizzato di EAs. Un ***Algoritmo Genetico*** è un algoritmo meta-euristico, ispirato alla selezione naturale, utilizzato per tentare di risolvere problemi di ottimizzazione per i quali non si conoscono altri algoritmi efficienti di complessità lineare o polinomiale. É possibile mappare i concetti della teoria dell'evoluzione con un problema di ottimizzazione, come evidenziato nella tabella 3.1.

Tabella 3.1: Mappatura dei concetti di un problema di ottimizzazione con la teoria dell'evoluzione

Ottimizzazione	Teoria dell'evoluzione
Problema di Ottimizzazione	Ambiente in cui sono calati gli individui
Funzione obiettivo	Capacità di Adattamento
Soluzione candidata	Individuo
Insieme di soluzioni candidate	Popolazione

Un GA fa evolvere iterativamente una **popolazione di *individui***, producendo di volta in volta nuove generazioni di individui migliori, rispetto ad una cosiddetta ***misura di fitness***, finché uno o più criteri di arresto non sono soddisfatti. La generazione di nuovi individui avviene per mezzo di tre operatori di ricerca, quali:

- ***Selezione***: operatore ispirato alla selezione naturale, che si occupa di selezionare gli individui più forti, con fitness più alta, poichè risolvono meglio di altri il problema di ricerca dato. La selezione crea un sottoinsieme della popolazione che viene *candidato alla riproduzione*, sfruttando il criterio di selezione che dovrebbe favorire gli individui con alta fitness (più forti), ciò permette di sfruttare (**exploit**) le buone soluzioni. Infatti, le soluzioni più promettenti sono favorite nella speranza che esse permettano la generazione di altre più promettenti.

- ***Crossover***: operatore ispirato alla genetica, che si occupa di combinare i genitori per creare prole. Gli individui selezionati dall'operatore di selezione (parents) sono *abbinati casualmente* per generare della prole (offsprings) incrociando in qualche modo la loro codifica. I figli generati rimpiazzeranno i genitori della generazione intermedia.

- ***Mutazione***: operatore ispirato alla genetica, che si occupa di apportare, con una bassa probabilità, una variazione ai nuovi individui nati dal crossover. Alcuni geni, ovvero singoli elementi della codifica dell'individuo, presenti tra tutti gli individui sono *mutati casualmente* (con bassa probabilità). La mutazione, di solito, non deve dar luogo ad individui con codifiche fuori dal dominio del problema.

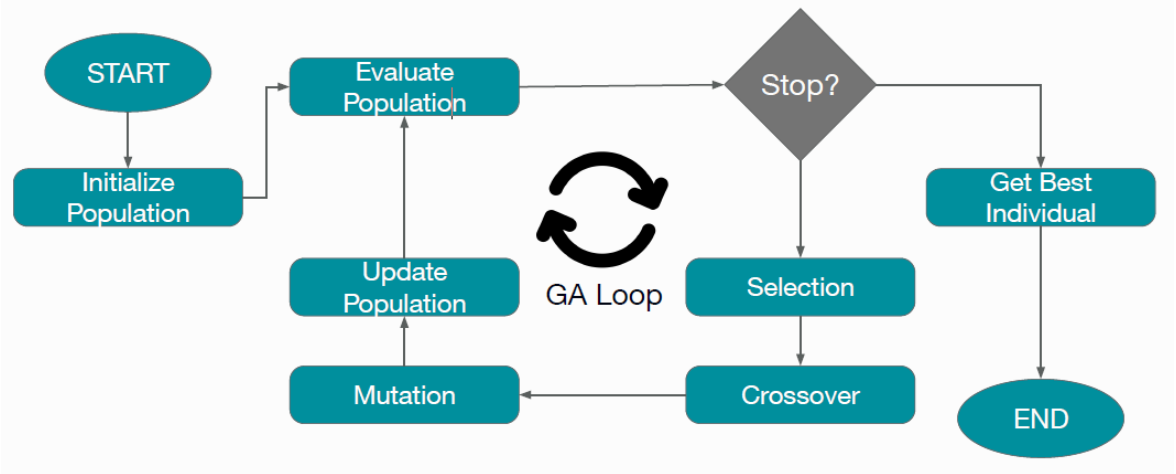

Figura 3.1: Il loop degli algoritmi genetici standard

L'obiettivo del lavoro di tesi è quello di definire, tramite l'utilizzo di algoritmi genetici, modelli predittivi per l'identificazione di vulnerabilità, per poi analizzare il risultato e l'efficienza di tali modelli. Questo perchè, probabilmente, l'utilizzo di algoritmi genetici che arrivano alla definizione di un intero albero decisionale, potrebbe essere molto più efficace rispetto alla creazione standard del modello predittivo.

La ricerca si è concentrata principalmente sulla scelta di un'appropriata *codifica degli individui* e sulla definizione di *operatori di ricerca (selezione, crossover e mutazione)*. É stato inoltre fondamentale concetrarsi sulla scelta di un'appropriata *misura di fitness* per la valutazione degli individui.

3.2 Codifica degli individui

L'individuo che si è scelto di codificare è il *Decision Tree - albero di decisione*, ovvero un modello predittivo, utilizzato sia per attività di classificazione che di regressione. É stato scelto proprio questo modello in quanto è un modello molto semplice da comprendere, infatti sono molto utili per la loro *facilità di lettura*; inoltre ha una struttuta ricorsiva facilmente codificabile in individui ricombinabili. Esso, infatti, si compone di una struttura ad albero gerarchica, che consiste di un nodo radice, di rami, nodi interni e nodi foglia.

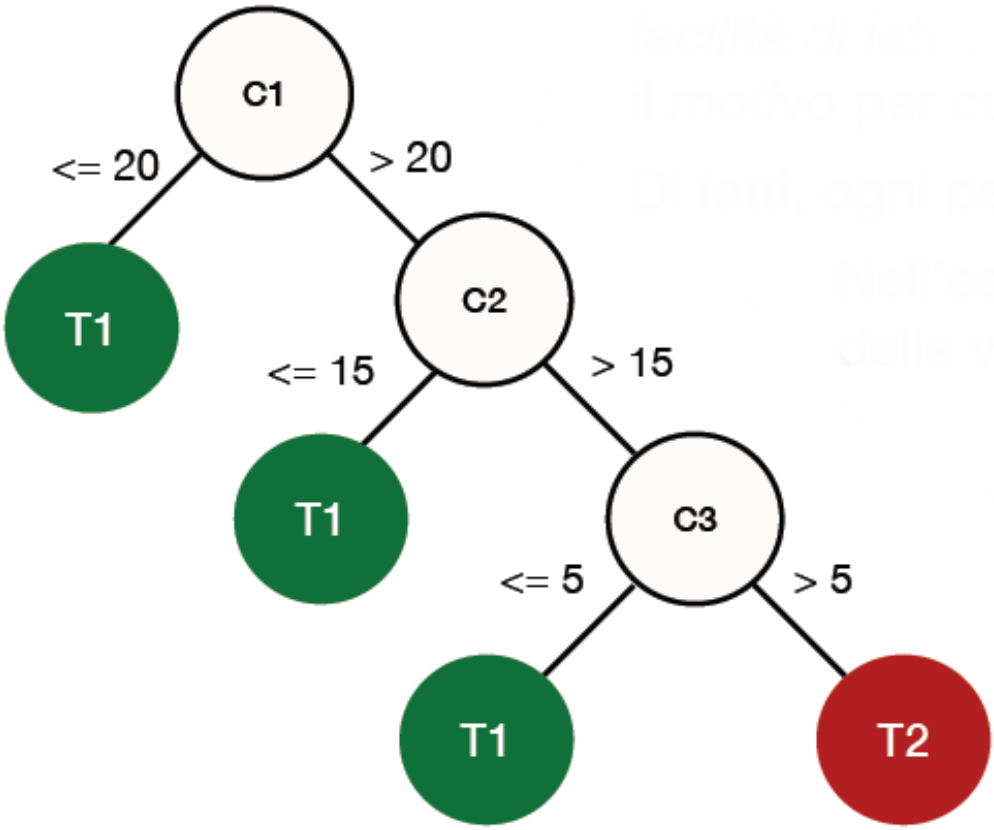

Figura 3.2: La struttura del Decision Tree

Come è possibile riscontrare in figura 3.2, in questo modello predittivo i dati presi come input vengono continuamente splittati in base a certi criteri. Due concetti chiave per capire il loro funzionamento sono nodi e foglie. I nodi interni sono i luoghi in cui, in base a certe regole, i dati vengono splittati. Le foglie sono invece i risultati finali, i luoghi in cui finiscono i dati una volta separati. L'obiettivo è quello di trovare i valori e le caratteristiche per cui otteniamo il migliore split, ovvero quelle caratteritiche e quei valori che meglio riescono a distinguire le varie classi target. Quindi ricapitolando, sulla base delle funzionalità disponibili, entrambi i tipi di nodo conducono valutazioni per formare sottoinsiemi omogenei, che sono rappresentati da nodi foglia (o nodi terminali) i quali rappresentano rappresentano tutti i possibili risultati all'interno del set di dati.
É importante notare che grazie alla sua facilità di lettura, navigando l'albero dalla radice fino alle foglie potremmo derivare delle *regole di classificazione*. Guardando l'albero in figura 3.2 possiamo facilmente intuire la regola che porta ad esempio alla classificazione di un valore come T2. Si potrebbe motivare la predizione della classe T2 nel seguente modo:
C1 > 20 AND C2 > 15 AND C3>5 -> T2.
Dopo aver introdotto cosa sia un Decision Tree e come funziona, analizziamo come rappresentarlo all'interno dell'algoritmo genetico. Un algoritmo genetico (GA) necessita di un'appropriata rappresentazione codificata degli individui, comunemente si adoperano stringhe binarie, ma in questo caso per la codifica degli individui la scelta è stata quella di avvalersi del concetto di Composite Pattern [32].
Il **Composite Pattern** è un design pattern di progettazione strutturale che consente di comporre oggetti in strutture ricorsive, in modo da trattare singoli oggetti e oggetti composti alla stessa maniera.

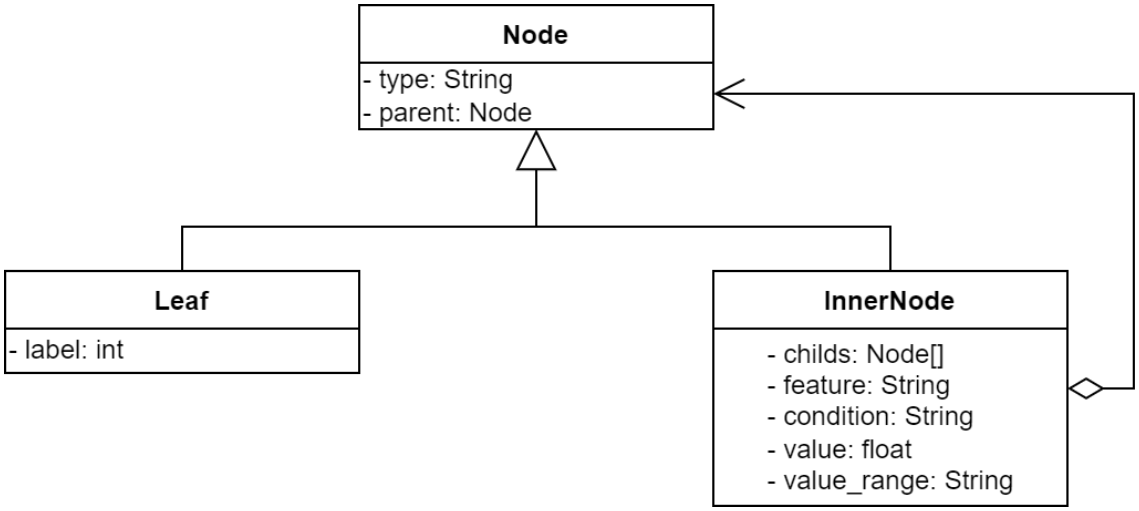

Figura 3.3: La struttura dell'individuo

L'individuo è definito dalla classe `Node` che descrive gli attributi e le operazioni comuni sia per gli oggetti Leaf sia per gli oggetti InnerNode.
La classe `InnerNode`, è un elemento che ha sottoelementi: Leaf o altri InnerNode. Rappresenta i *nodi interni* del Decision Tree, ha come attributi:

- i nodi figli (nodo dx e nodo sx);
- il nome della feature associata al nodo;
- la condizione da verificare, quindi '<=' o '>';
- il valore soglia del nodo;
- il range che può assumere il valore soglia, utilizzato principalmente nell'operatore di mutazione.

La classe `Leaf`, anch'esso sottoclasse dell'ogetto Node, è un elemento di base di un albero e non ha sottoelementi. Rappresenta le foglie del Decision Tree, chiamati anche nodi terminali, ovvero nodi che non si dividono in altri nodi e che si occupano di classificare le istanze; infatti, hanno un unico attributo utilizzato per identificare la classe dell'istanza dopo aver attraversato l'albero decisionale.

3.3 Funzioni obiettivo e valutazione degli individui

Le funzioni obiettivo del problema da ottimizzare sono le seguenti:

- $f_1 = \max(precision)$
- $f_2 = \max(recall)$
- $f_3 = \max(accuracy)$
- $f_4 = \max(F - measure)$

Precision e recall sono due delle metriche principali nei processi di classificazione di dati. La *precision* inidica il numero di predizioni corrette per la classe 'true' rispetto a tutte le predizioni fatte dal classificatore, ovvero quanti errori ci sono nella lista delle predizioni fatte.

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

La *recall* indica il numero di predizioni corrette per la classe 'true' rispetto a tutte le istanze positive di quella classe, ovvero quante istanze positive nell'intero dataset il classificatore può determinare.

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

É stata poi utilizzata l'*accuracy* che indica il numero totale di predizioni corrette (sia della classe positiva che negativa), la quale però potrebbe creare problemi di interpretazione nel caso di dataset sbilanciati dove il numero di casi positivi è molto basso e il classificatore sarà sicuramente molto più abile a riconoscere i veri negativi.

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

Infine abbiamo la *F-measure*, che è una media armonica di precision e recall. Rispetto ad una media convenzionale, quella armonica attribuisce un peso maggiore ai valori piccoli, questo fa sì che un classificatore ottenga un alto punteggio F-measure solo quando precision e recall sono entrambi alti.

$$F - measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Quindi l'obiettivo dell'ottimizzazione è quello di massimizzare alcune delle metriche di valutazione di un modello di machine learning, ovvero precision, recall, accuracy e F-measure. Queste metriche danno informazioni riguardanti le predizioni del modello e ci permettono di analizzare se il modello definito riuscirà a predire correttamente l'istanza, quindi ad individuare o meno l'introduzione di una vulnerabilità a livello di commit.
Per la valutazione degli individui, dopo alcune prove empririche, si è evidenziato come i risultati migliori si ottenessero quando si procedeva all'ottimizzazione del problema a singolo obiettivo, quindi ottimizzando solo una delle metriche; mentre l'ottimizzazione di un problema multi-obiettivo ha portato a risultati non ottimali e, soprattutto, ad una convergenza dell'algoritmo molto lenta e quindi inefficiente. Per questi motivi, la valutazione è stata effettuata cercando di ottimizzare una tra le metriche sopra citate. La scelta della metrica da utilizzare viene effettuata a seconda delle caratteristiche del dataset.
La valutazione del modello avviene in fase di "addestramento" dove viene utilizzato un apposito ***dataset di training***, questa scelta è stata effettuata in quanto si cerca di emulare l'escuzione di una tradizionale pipeline di machine learning, in cui il dataset iniziale viene splittato in un dataset di training e in un dataset di testing.

Per eseguire la valutazione è stata creata la classe ***TreeModel***, che implementa la funzione `train` e la funzione `predict`. La funzione `train` esamina tutte le righe del dataset di training, memorizzando il valore della colonna *target*, per poi procedere alla predizione dell'istanza sul modello definito dall'algoritmo gentico.

```
def train(self):
    expected = []
    predicted = []
    for index, row in self.df_train.iterrows():
        expected.append(row['target'])
        '''
        the decision_of_tree function takes care of following
        the path of the tree by checking the various conditions
        '''
        predicted.append(self.__decision_of_tree(self.tree, row))
```

La funzione `predict` è identica alla funzione precedente, l'unica differenza è che le predizioni in questo caso vengono effettuate su un dataset di testing. Questo ci permette di analizzare il comportamento e le prestazioni del modello con dati differenti da quelli del dataset utilizzato per la definzione di esso.

3.4 Criteri di Arresto

Ogni volta che viene eseguito l'algoritmo genetico, è necessario decidere in ogni iterazione se l'esecuzione dell'ottimizzazione deve essere proseguita o meno. Esistono molti modi diversi per determinare quando deve essere terminata l'esecuzione di un algoritmo.

Per scegliere il criterio di arresto migliore sono state effettuate diverse prove empiriche, che hanno evidenziato come i criteri con i quali è stato possibile ottenere i risultati migliori sono i seguenti:

1. criterio di arresto che limitava il numero di generazioni a 50 generazioni;
2. criterio di arresto che limitava il tempo di esecuzione dell'algoritmo, impostandolo ad un'ora.

3.5 Operatori di ricerca

Dopo aver definito l'individuo ed il problema, bisogna definire gli operatori di ricerca.

3.5.1 Selezione

L'operatore di selezione che è stato utilizzato è il **Tournament Selection**, che sceglie delle coppie di individui dalla popolazione e confronta la loro funzione di fitness andando a favorire l'individuo "più forte", quindi con funzione di fitness più alta. Il vincitore di ogni torneo viene selezionato per il crossover; il numero di tornei dipende dal numero di individui della popolazione, se la taglia della popolazione è 50 allora ci saranno 50 tornei.
Questo operatore di selezione utilizza la funzione obiettivo, dalla quale deriva un valore di fitness per ogni individuo, che guiderà la selezione (*fitness-guided selection*). L'utilizzo di questo specifico operatore ha portato a buoni risultati sia per quanto riguarda la "velocità" di convergenza, sia per quanto riguarda il valore della soluzione ottima trovata alla fine della computazione.

3.5.2 Crossover

La codifica scelta per gli individui ha influenzato la scelta dell'algoritmo di crossover. L'operatore che è stato utilizzato ed implementato è il ***Tree Crossover*** e si ispira all'operatore noto come *One-Point Crossover*; questo operatore sceglie casualmente un punto sui cromosomi di entrambi i genitori che viene designato come "punto di taglio", utilizzato per incrociare le due parti e dare vita a due offspring. L'operatore TreeCrossover opera su coppie di genitori e per ogni genitore, sceglie casualmente un numero compreso tra 1 e l'altezza massima dell'albero; questo numero viene poi utilizzato per visitare in profondità l'albero, scegliendo ad ogni iterazione, con il 50% di probabilità, se visitare il ramo sinistro o il ramo destro dell'albero. Questa visita porterà al raggiungimento di un nodo interno o di una foglia.
Una volta scelto in modo casuale un qualsiasi nodo per ogni genitore e quindi dopo aver individuato un sottoalbero per ognuno di esso, l'operatore provvede all'incrocio scambiando i sottoalberi, così come riportato nella figura 3.4.

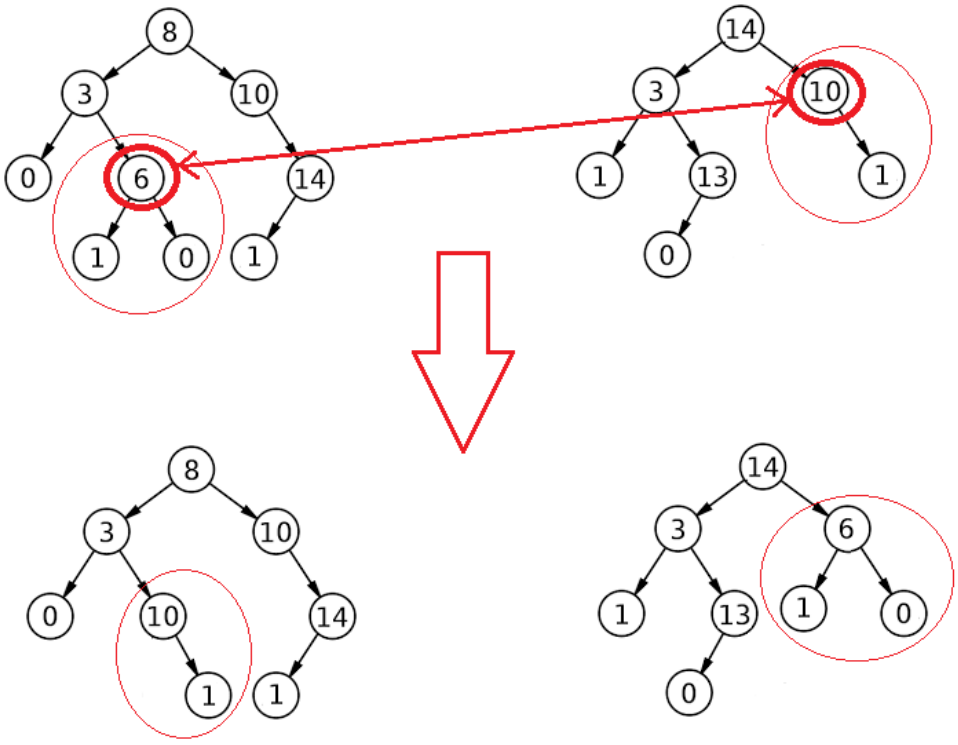

Figura 3.4: Esempio di esecuzione dell'operatore di crossover

3.5.3 Mutazione

L'operatore di mutazione è il secondo passo per la creazione di nuove soluzioni per esplorare aree sconosciuste dello spazio di ricerca. Sebbene in misura minore, anche la mutazione è sensibile alla codifica degli individui. L'operatore di mutazione che è stato implementato è il ***Tree Mutation*** e si ispira all'operatore di mutazione noto come *Bit-Flip Mutation*, questo operatore considera tutti i geni presenti nella popolazione e con una bassa probabilità applica una variazione del gene di un individuo con un altro valore ammissibile. L'operatore TreeMutation sceglie casualmente un numero compreso tra 1 e l'altezza massima dell'albero che viene utilizzato per visitare in profondità l'albero, come nel caso dell'operatore di selezione.
Una volta raggiunto un qualsiasi nodo in modo casuale, è possibile che si verifichi uno dei seguenti due casi:

- il nodo raggiunto è un **nodo interno (InnerNode)**, in questo caso, la mutazione può avvenire in due modi:
  - modificando il valore soglia del nodo con un valore casuale compreso tra l'intervallo indicato nell'attributo *value_range* del nodo;
  - oppure, modificando la condizione da verificare da '<' a '>=' o viceversa.

  La scelta tra queste due casi avviene in modo probabilistico, con il 50% di probabilità per ognuno.
- il nodo raggiunto è un **nodo foglia (Leaf)**, in questo caso la mutazione avviene modificando il valore dell'attributo *label* con un altro valore, scelto in modo casuale dal vettore dei nodi foglia.

3.6 Popolazione Iniziale

Per la generazione della popolazione iniziale è stata implementata un'apposita funzione, che in base alla grandezza della popolazione, genera degli oggetti `Node` che rappresentano alberi casuali; questi alberi vengono costruiti sulla base di un file contenente le features del dataset e le possibili classificazioni. Il file è così strutturato:

- il tipo di nodo, quindi se è un nodo interno o un nodo foglia;
- se è un nodo interno la riga contiene il nome della feature e il range che può assumere il valore soglia del nodo;
- se è un nodo foglia, invece, viene indicato il valore della label.

Si è scelto di generare casualmente la popolazione iniziale, in questo modo l'algoritmo ha a disposizione un insieme di soluzioni che gli permettono di far partire l'esplorazione da più punti dello spazio di ricerca. Per quanto riguarda la taglia della popolazione iniziale, essa è stata impostata a 50 individui, perchè ha portato a risultati migliori rispetto a popolazioni iniziali composte da 10, 20 e 25 individui.

3.7 Caratteristiche del dataset

Il problema così come è stato definito si basa sulla costruzione dei Decision Tree, quindi, il dataset su cui lavora, deve già avere a disposizione dati che possono essere fruibili ad un machine learning.
I dati devono subire una pre-elaborazione, questo perchè spesso i dataset sono:

- incompleti: hanno valori nulli per alcuni attributi o mancano del tutto attributi importanti e/o interessanti.
- inaccurati: contengono valori errati o che si discostano sensibilmente da valori attesi;
- incosistenti: contengono dati che rappresentano la stessa informazione ma hanno valori diversi.

É importante effettuare la pre-elaborazione dei dataset prima dell'esecuzione dell'algoritmo genetico e quindi della definizione del modello predittivo, perchè se i dati in input non sono di buona qualità, neanche le analisi basate su di questi lo possono essere. Nel dettaglio il dataset deve aver già effettuato il processo di **Data Preparation** che prevede quattro passi fondamentali:

- ***data cleaning***: è un processo che viene per lo più fatto quando si ha a che fare con dei dati semi strutturati, per cui ci si trova in situazioni dove per un certo dato mancano delle informazioni;
- ***feature scaling***: è un processo che prevede di normalizzare o scalare l'insieme di valori di una caratterstica, questo perchè se l'insieme dei valori per una determinata caratteristica è molto diverso rispetto ad un altro, c'è il rischio che l'algoritmo di apprendimento sottostimi o sovrastimi l'importanza di quella caratteristica;
- ***feature selection***: è un processo che ha l'obiettivo di definire delle caratteristiche, anche chiamate feature, metriche, o variabili indipendenti che possano caratterizzare gli aspetti principali del problema in esame e, quindi, avere una buona potenza predittiva;
- ***data balancing***: sono un insieme di tecniche che convertono un dataset sbilanciato in un dataset bilanciato, queste tecniche sono fondamentiali perchè se non viene considerato il problema dello sbilanciamento dei dati, è molto probabile che venga definito un modello di machine learning capace di caratterizzare correttamente solo gli esempi della classe più popolosa.

Considerando che l'algoritmo progettato si focalizza solo sulla definizione del modello predittivo, diventa di fondamentale importanza importare, all'interno del tool, un dataset che abbia già subito la fase di data preparation. In questo modo la definizione del modello predittivo, e quindi "l'addestramento" tramite algoritmo genetico, avviene utilizzando dati *accurati, completi e consistenti* assicurandoci di ottenere come risultato un modello predittivo più efficiente.

3.8 Framework pymoo

Per la realizzazione del tool per la tecnica di individuazione delle vulnerabilità tramite l'utilizzo di algoritmi genetici è stato utilizzato il framerwork **pymoo - *Multi-objective Optimization in Python*** [33].
Questo framework offre algoritmi di ottimizzazione singoli e multi-obiettivo ed è stato scelto in quanto permette la personalizzazione degli individui e degli operatori di ricerca del GA.

3.8.1 Definizione del problema

Il framework *pymoo* prevede la definizione del problema di ottimizzazione. Il problema definito *ProblemDecisionTree* estende la classe *ElementwiseProblem*, ed implementa la funzione

`evaluate` che valuta una **singola** soluzione (soluzione candidata - individuo) alla volta.

3.8.2 Esecuzione Genetic Algorithm

Per eseguire l'ottimizzazione, bisogna prima di tutto, creare e settare l'algoritmo gentico. Il framework *pymoo* mette a disposizione la classe **`GA`**, dove bisogna indicare la taglia della popolazione ed i tipi di operatori di ricerca da utilizzare, si procede poi con l'esecuzione dell'ottimizzazione. Pymoo mette a disposizione la funzione `minimize`, dove bisogna indicare l'algoritmo da utilizzare per eseguire l'ottimizzazione e il problema da ottimizzare; nel caso specifico questa funzione utilizza l'algortimo genetico per minimizzare la funzione obiettivo (funzione `evaluate`) del problema creato.

É possibile consultare il codice al seguente link GA-for-ML.

3.9 Come vengono valutati i risultati

Per valutare i risultati ottenuti è stato definito un Decision Tree tramite l'utilizzo di *CART* (CART è l'algoritmo utilizzato da Scikit Learn per la costruzione di un decision tree), le cui metriche di performance sono state messe a confronto con le metriche di performance del modello definito tramite l'algoritmo genetico.

CAPITOLO 4

Risultati

In questo capitolo vengono illustrati i risultati ottenuti dalle ottimizzazioni delle diverse metriche.

4.1 Ottimizzazione della Precision

La tabella 4.1 mostra i risultati ottenuti definendo il modello con l'obiettivo di ottimizzare la precision. Possiamo notare come la precision del decision tree definito tramite GA, uguale allo 0.013, è molto bassa rispetto allo 0.08 del modello definito tramite CART. Analizzando le altre metriche notiamo che la recall per il modello definito tramite GA è uguale ad 1, ciò vuol dire che il modello classifica tutte le istanze come positive senza mai predire un'istanza come negativa. Osserviamo, inoltre, che il modello definito tramite CART ha un accuracy molto alta, uguale allo 0.96 rispetto allo 0.013 del modello definito tramite GA. Questo aspetto è collegato all'analisi effettuata per la metrica di recall, perchè, siccome nel dataset il 95% delle istanze sono etichettate come non vulnerabili, il modello ha predetto tutte le istanze come vulnerabili e quindi ciò ha portato ad azzerare il numero dei falsi negativi e ad avere una recall pari a 1.
Infine, è interessante notare come la metrica più alta non sia la precision, nonostante come funzione obiettivo del problema era impostata la massimizzazione di quest'ultima. Infatti, i risultati sembrano mostrare un'ottimizzazione della recall piuttosto che della precision.

Tabella 4.1: Confronto tra le metriche dei modelli predittivi definiti tramite CART e tramite gli algoritmi genetici. La colonna *Decision Tree* GA_1 e la colonna *Decision Tree* GA_2 contengono le metriche dei modelli definiti tramite GA che ha come funzione obiettivo la massimizzazione della *precision*, i modelli si differenziano per il criterio di arresto scelto per l'esecuzione del GA. Per *Decision Tree* GA_1 il criterio di arresto è la limitazione del numero di generazioni impostata ad un massimo di 50 generazioni, mentre per *Decision Tree* GA_2 è la limitazione del tempo di esecuzione impostato ad un'ora.

Metrica	Decision Tree CART	Decision Tree GA_1	Decision Tree GA_2
Precision	**0.082**	0.013	0.062
Recall	0.171	**1.000**	0.042
Accuracy	0.961	0.013	**0.978**
F-measure	**0.114**	0.026	0.050

Considerando questi risultati possiamo affermare che questo esperimento mostra dei risultati che vanno contro le aspettative. Così si è deciso di modificare alcuni parametri del GA per analizzarne il comportamento. Sono stati infatti ottenuti dei risultati migliori modificando il criterio di arresto e limitando l'esecuzione dell'algoritmo per un tempo massimo di un'ora, la colonna *Decision Tree* GA_2 della tabella 4.1 mostra i risultati ottenuti. Le prestazioni si avvicinano di più al modello definito tramite CART, anzi la metrica di accuracy è uguale allo 0.98 e risulta essere migliore, ma questa metrica non può essere presa come riferimento per valutare il modello visto l'utilizzo di un dataset sbilanciato.
Questi risultati mostrano, in ogni caso, che il modello migliore è il modello definito tramite CART.

4.2 Ottimizzazione della Recall

La tabella 4.2 mostra i risultati ottenuti definendo il modello con l'obiettivo di massimizzare la recall. Dai risultati si evince come tutte le metriche del modello definito tramite GA siano uguali a 0, tranne per l'accuracy uguale allo 0.98. Questi risultati evidenziano la definizione di un modello "pessimistico", ovvero un modello che restituisce tutte predizioni negative. L'accuracy è invece uguale allo 0.98, superiore allo 0.96 del modello definito tramite CART, in quanto il dataset è sbilanciato e sono in numero maggiore le istanze negative.
Sono stati poi eseguiti ulteriori esperimenti, nei quali sono stati modificati alcuni parametri dell'algoritmo genetico come il criterio di arresto e la probabilità di crossover e mutazione, ma i risultati non sono cambiati.

Tabella 4.2: Confronto tra le metriche dei modelli predittivi definiti tramite CART e tramite gli algoritmi genetici. La colonna *Decision Tree* GA_1 e la colonna *Decision Tree* GA_2 contengono le metriche dei modelli definiti tramite GA che ha come funzione obiettivo la massimizzazione della *recall*, i modelli si differenziano per il criterio di arresto scelto per l'esecuzione del GA. Per *Decision Tree* GA_1 il criterio di arresto è la limitazione del numero di generazioni impostata ad un massimo di 50 generazioni, mentre per *Decision Tree* GA_2 è la limitazione del tempo di esecuzione impostato ad un'ora.

Metrica	Decision Tree CART	Decision Tree GA_1	Decision Tree GA_2
Precision	**0.082**	0.000	0.000
Recall	**0.171**	0.000	0.000
Accuracy	0.961	**0.986**	**0.986**
F-measure	**0.114**	0.000	0.000

4.3 Ottimizzazione dell'Accuracy

La tabella 4.3 mostra i risultati ottenuti definendo il modello con l'obiettivo di massimizzare l'accuracy. I risultati evidenziano come il modello definito tramite GA abbia una precision inferiore rispetto al modello definito tramite CART. Invece, la recall è uguale allo 0.95, che è molto più alta rispetto allo 0.17 del modello definito in modo "tradizionale". Possiamo notare, infine, che il modello definito tarmite GA ha un'accuracy uguale allo 0.014, che è molto più bassa rispetto allo 0.96 dell'altro modello. Questi risultati evidenziano che il modello definito tramite GA restituisce per lo più predizioni positive e questo spiega il perchè di un'accuracy così bassa. Infine, l'F-measure uguale allo 0.025 rispetto allo 0.11, conferma che il modello migliore è quello definito tramite CART.

É interessante notare che, nonostante l'ottimizzazione fosse incentrata sull'accuracy, i risultati hanno portato ad ottenere il valore più basso in assoluto per questa metrica in confronto a tutti gli altri risultati. Sono stati ottenuto gli stessi risultati anche modificando i parametri del GA.

Tabella 4.3: Confronto tra le metriche dei modelli predittivi definiti tramite CART e tramite gli algoritmi genetici.La colonna *Decision Tree* GA_1 e la colonna *Decision Tree* GA_2 contengono le metriche dei modelli definiti tramite GA che ha come funzione obiettivo la massimizzazione dell'*accuracy*, i modelli si differenziano per il criterio di arresto scelto per l'esecuzione del GA. Per *Decision Tree* GA_1 il criterio di arresto è la limitazione del numero di generazioni impostata ad un massimo di 50 generazioni, mentre per *Decision Tree* GA_2 è la limitazione del tempo di esecuzione impostato ad un'ora.

Metrica	Decision Tree CART	Decision Tree GA_1	Decision Tree GA_2
Precision	**0.082**	0.013	0.013
Recall	0.171	**0.958**	**0.958**
Accuracy	**0.961**	0.014	0.014
F-measure	**0.114**	0.025	0.025

4.4 Ottimizzazione della F-measure

La tabella 4.4 mostra i risultati ottenuti definendo un modello con l'obiettivo di ottimizzazione la F-measure. É possibile notare dai risultati che il valore di precision per il modello definito tramite GA, uguale allo 0.23, è più elevato rispetto allo 0.08 del modello definito tramite CART; anche la F-measure, uguale allo 0.16, è più alta rispetto allo 0.11; mentre, il valore di recall è più basso: 0.125 rispetto allo 0.17. Considerando questi valori, è possibile affermare che il modello definito tarmite GA è un modello che fornisce spesso delle predizioni corrette quando l'istanza viene classificata come positiva; mentre il valore di recall più basso ci informa che il modello è sensibile ai falsi negativi, ovvero alle previsioni negative sbagliate. Infine, l'F-measure è più alta in quanto il modello definito tramite GA ha aumentato le performance per la precision. Le metriche di accuracy sono molto alte in entrambi i modelli, ma come è stato già evidenziato precedentemente è sconsigliato usare questa metrica per valutare il modello in quanto le classi nel dataset sono distribuite in maniera impari.

Tabella 4.4: Confronto tra le metriche dei modelli predittivi definiti tramite CART e tramite gli algoritmi genetici. La colonna *Decision Tree* GA_1 e la colonna *Decision Tree* GA_2 contengono le metriche dei modelli definiti tramite GA che ha come funzione obiettivo la massimizzazione della *F-measure*, i modelli si differenziano per il criterio di arresto scelto per l'esecuzione del GA. Per *Decision Tree* GA_1 il criterio di arresto è la limitazione del numero di generazioni impostata ad un massimo di 50 generazioni, mentre per *Decision Tree* GA_2 è la limitazione del tempo di esecuzione impostato ad un'ora.

Metrica	Decision Tree CART	Decision Tree GA_1	Decision Tree GA_2
Precision	0.082	**0.230**	0.225
Recall	**0.171**	0.125	0.125
Accuracy	0.961	**0.983**	0.982
F-measure	0.114	**0.166**	0.162

4.5 Considerazioni finali

Osservando i risultati ottenuti dalle 4 tipologie di ottimizzazione, possiamo evincere che i risultati migliori sono stati ottenuti ottimizzando la F-measure. Questa osservazione non può essere generalizzata, in quanto le meriche di performance del modello definito tramite GA dipendono anche dal tipo di dataset utilizzato. In ogni caso, bisognerebbe effettaure sempre tutte le tipologie di ottimizzazione, per poi prendere come riferimento il modello con prestazioni più alte. Inoltre, è stato possibile dedurre che spesso i parametri dell'algoritmo genetico influiscono sulle metriche di perfomance del modello, e quindi sui risultati. In alcuni casi, è possibile che i parametri dell'algoritmo abbiano falsato in qualche modo le metriche.

Ci sono alcuni possibili miglioramenti che possono essere attuati per quanto riguarda l'implementazione dell'algoritmo genetico; ad esempio potrebbe essere introdotta una funzione multi-obiettivo, oppure potrebbero essere introdotti nuovi operatori di ricerca.

CAPITOLO 5

Conclusioni

Questo capitolo vuole riassumere il lavoro svolto ed analizzare i risultati ottenuti, proponendo diversi spunti di riflessione che potrebbero essere utilizzati in futuro per migliorare il lavoro.

5.1 Analisi dei risultati

I risultati principali indicano che il problema dovrebbe essere ulteriormente approfondito. In primo luogo, il problema maggiore sono i valori delle metriche di valutazione molto basse, dovute allo sbilanciamento del dataset e/o all'utilizzo di caratteristiche non adatte per l'identificazione delle vulnerabilità. Questi aspetti sono stati analizzati da Lomio et al. [6].
In secondo luogo, i risultati indicano che l'utilizzo di algorimti genetici per la definizione di modelli predittivi, in alcuni casi, ha un impatto positivo sulle prestazioni del modello stesso. Infatti, a seconda dei parametri impostati per l'algoritmo genetico e a seconda della funzione obiettivo scelta, i modelli definiti tramite GA hanno prestazioni più elevate rispetto ai modelli definiti tradizionalmente.
Infine, è da evidenziare come in alcuni casi l'ottimizzazione di una determinata funzione obiettivo, e quindi la massimizzazione di una determinata metrica, portasse quella metrica ad avere un valore più basso rispetto ai valori ricevuti dagli altri esperimenti. Questo aspetto va approfondito negli studi futuri.

5.2 Riflessione sui risultati complessivi

In questa tesi è stato presentato uno studio empirico sull'utilizzo degli algoritmi genetici per la definizione di modelli predittivi. Giunti al termine del presente lavoro di tesi, possiamo trarre delle conclusioni riguardanti la tecnica illustrata nel capitolo 3.
I risultati migliori sono stati ottenuti impostando come funzione obiettivo la massimizzazione della metrica *F-measure* e come criterio di arresto la limitazione del numero di generazioni impostata ad un massimo di 50 generazioni. É possibile, quindi, affermare che l'utilizzo di algoritmi genetici ha portato alla definizione di un modello predittivo con prestazioni migliori rispetto ad un modello definito tramite l'utilizzo di una tradizionale pipeline di machine learning.

Al fine di migliorare ulteriormente i risultati ottenuti, bisogna ampliare questo studio. Inoltre, i futuri lavori dovranno concentrarsi sulla struttura dei modelli definiti dagli algoritmi genetici, per comprendere e analizzare le differenze con la struttura dei Decision Tree definiti tramite i tradizionali framework. Inoltre, bisogna appronfondire l'utilizzo di tale tecnica per la definizione di altri modelli predittivi diversi dal Decision Tree.

Ringraziamenti

A conclusione di questo elaborato, vorrei dedicare qualche riga a tutti coloro che mi sono stati vicini in questo percorso di crescita personale e professionale.

Innanzitutto, ringrazio il prof. Fabio Palomba e dott. Emanuele Iannone che in questi mesi di lavoro hanno saputo guidarmi con suggerimenti pratici nelle ricerche e nella stesura dell'elaborato, trasmettendomi tutta la loro passione. Grazie a loro ho accresciuto le mie conoscenze e le mie compotenze e sono state fonte di ispirazione per la trascizione di questa tesi.

Ringrazio i miei genitori, mia sorella Giusy e mio nonno Michele per essere stati sempre al mio fianco e per avermi aiutato a superare i periodi più difficili, ma soprattutto vi ringrazio per avermi reso la persona che sono oggi.

Un ringraziamento particolare va ai miei amici fraterni, Alessandro, Mario, Giuseppe, Salvatore e Francesco, che mi hanno sempre incorraggiato fin dall'inizio.

Inoltre, vorrei ringraziare tutti i miei colleghi, in particolare Antonio, Mario, Natale, Stefano e Thomas, con i quali ho trascorso dei momenti stupendi in questi tre anni.

Mi dispiace non poter citare ad uno ad uno tutti coloro che sono stati presenti e che hanno reso speciale questo percorso, ma siete davvero tanti e per voi posso solo scrivere un semplicissimo, ma immenso, grazie.

Infine, ringrazio me stesso e dedico questo piccolo traguardo a me e alla persona che sono.

Bibliografia

[1] Wikipedia, "Wannacry ransomware attack." (Citato a pagina 2)

[2] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," 04 2015. (Citato a pagina 3)

[3] M. Dowd, J. McDonald, and J. Schuh, *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education, 2006. (Citato a pagina 6)

[4] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004. (Citato a pagina 6)

[5] M. Montanile, S. Voci, G. Tortora, and C. S. Malavenda, "La sicurezza del software: guida alla progettazione e allo sviluppo," *La sicurezza del software*, pp. 1–197, 2021. (Citato a pagina 6)

[6] F. Lomio, E. Iannone, A. De Lucia, F. Palomba, and V. Lenarduzzi, "Just-in-time software vulnerability detection: Are we there yet?," *Journal of Systems and Software*, p. 111283, 2022. (Citato alle pagine 6, 13, 14 e 36)

[7] P. Bulusu, H. Shahriar, and H. M. Haddad, "Classification of lightweight directory access protocol query injection attacks and mitigation techniques," in *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 337–344, IEEE, 2015. (Citato a pagina 10)

[8] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, "Noxes: a client-side solution for mitigating cross-site scripting attacks," in *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 330–337, 2006. (Citato a pagina 10)

[9] P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in *2010 IEEE international conference on information theory and information security*, pp. 521–524, IEEE, 2010. (Citato a pagina 10)

[10] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. R. Raje, and J. H. Hill, "Identifying and documenting false positive patterns generated by static code analysis tools," in *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, pp. 55–61, IEEE, 2017. (Citato a pagina 10)

[11] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," in *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 1, pp. 343–350, IEEE, 2006. (Citato a pagina 11)

[12] B. Jiang, Y. Liu, and W. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 259–269, IEEE, 2018. (Citato a pagina 11)

[13] N. Antunes and M. Vieira, "Benchmarking vulnerability detection tools for web services," in *2010 IEEE International Conference on Web Services*, pp. 203–210, IEEE, 2010. (Citato a pagina 11)

[14] L. N. Q. Do, J. Wright, and K. Ali, "Why do software developers use static analysis tools? a user-centered study of developer needs and motivations," *IEEE Transactions on Software Engineering*, 2020. (Citato a pagina 11)

[15] B. Chernis and R. Verma, "Machine learning methods for software vulnerability detection," in *International Workshop on Security and Privacy Analytics*, pp. 31–39, 2018. (Citato a pagina 11)

[16] J. A. Harer, L. Y. Kim, R. L. Russell, O. Ozdemir, L. R. Kosta, A. Rangamani, L. H. Hamilton, G. I. Centeno, J. R. Key, P. M. Ellingwood, *et al.*, "Automated software vulnerability detection with machine learning," *arXiv preprint arXiv:1803.04497*, 2018. (Citato a pagina 11)

[17] C. Theisen and L. Williams, "Better together: Comparing vulnerability prediction models," *Information and Software Technology*, vol. 119, p. 106204, 2020. (Citato alle pagine 11 e 12)

[18] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011. (Citato alle pagine 11 e 12)

[19] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, pp. 1–8, 2010. (Citato a pagina 12)

[20] B. Smith and L. Williams, "Using sql hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pp. 220–229, IEEE, 2011. (Citato a pagina 12)

[21] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pp. 315–317, 2008. (Citato a pagina 12)

[22] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in *Conference on Computer and Communications Security*, p. 529–540, 2007. (Citato a pagina 12)

[23] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *International Conference on Software Testing, Verification and Validation*, pp. 421–428, 2010. (Citato a pagina 12)

[24] K. Z. Sultana, V. Anu, and T. Chong, "Using software metrics for predicting vulnerable classes and methods in java projects: A machine learning approach," *Journal of Software: Evolution and Process*, vol. 33, 2020. (Citato a pagina 12)

[25] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," in *International Workshop on Security Measurements and Metrics*, 2010. (Citato a pagina 12)

[26] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011. (Citato a pagina 12)

[27] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?," *Empirical Software Engineering*, vol. 18, pp. 25–59, 2011. (Citato a pagina 12)

[28] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, 2014. (Citato a pagina 12)

[29] Y. Zhang, D. Lo, X. Xia, B. Xu, J. Sun, and S. Li, "Combining software metrics and text features for vulnerable file prediction," in *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 40–49, 2015. (Citato a pagina 12)

[30] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits," in *ACM SIGSAC Conference on Computer and Communications Security*, p. 426–437, 2015. (Citato a pagina 12)

[31] L. Yang, X. Li, and Y. Yu, "Vuldigger: A just-in-time and cost-aware tool for digging vulnerability-contributing changes," in *IEEE Global Communications Conference*, pp. 1–7, 2017. (Citato a pagina 12)

[32] J. Hunt, "Gang of four design patterns," in *Scala design patterns*, pp. 135–136, Springer, 2013. (Citato a pagina 19)

[33] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020. (Citato a pagina 27)