UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TESI DI LAUREA

Predizione dei difetti in Terraform mediante l'utilizzo di metriche di prodotto

RELATORE

Prof. Dario Di Nucci

CORRELATORE

Dott.ssa Valeria Pontillo

Università degli Studi di Salerno

CANDIDATO

Gerardo Brescia

Matricola: 0522501272

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

"Diventiamo grandi grazie ai sogni. Tutti i grandi uomini sono dei sognatori. Vedono cose nella leggera foschia primaverile, o nel fuoco rosso della sera d'un lungo inverno. Alcuni di noi lasciano morire questi grandi sogni, ma altri li nutrono e li proteggono; abbiatene cura nei giorni brutti affinché portino il sole e la luce che viene sempre a chi spera col cuore che i propri sogni si avverino."

Woodrow Wilson.

Abstract

Nell'attuale panorama aziendale, caratterizzato dalla rapida evoluzione degli ambienti operativi e di sviluppo, la collaborazione tra i team è essenziale. L'adozione della metodologia DevOps ha facilitato tale integrazione e, all'interno di questo approccio, l'Infrastructure as Code (IaC) assume un ruolo cruciale. Tuttavia, mentre la predizione dei difetti è stata ampiamente studiata nei linguaggi di programmazione tradizionali, è fondamentale notare una minore attenzione ai linguaggi infrastrutturali. Questa mancanza ha rappresentato una significativa opportunità di studio e innovazione.

L'obiettivo del presente lavoro è stato quello di adattare e specializzare metriche, che riuscissero a descrivere le caratteristiche di qualità di codice, in formato Ansible per il linguaggio di IaC Terraform. Le metriche sono state implementate all'interno di RADON Defuse, uno strumento per la predizione di difetti per codice infrastrutturale con l'obiettivo finale di estenderlo a Terraform. Il tool esteso è stato impiegato per condurre uno studio empirico. In particolare, si è scelto di valutare le prestazioni del classificatore Random Forest nel contesto di Terraform. In parallelo, è stata condotta un'analisi sull'efficacia delle nuove metriche nell'identificazione dei difetti nei progetti Terraform.

I risultati emersi dall'analisi di 66 repository hanno evidenziato l'alta efficacia del classificatore Random Forest nella predizione dei difetti nel codice Terraform, ottenendo punteggi medi di AUC-PR e MCC rispettivamente di circa 0.90 e 0.80. È emerso, infine, che le metriche più efficaci per la predizione dei difetti sono risultate essere TextEntropy, LineSourceCode e NumTokens.

In conclusione, questi risultati aprono la strada a miglioramenti significativi nella predizione dei difetti nel contesto di Terraform e nell'ambito più ampio dell'IaC.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Nell'attuale panorama aziendale, caratterizzato da ambienti in continua evoluzione, i team operativi e di sviluppo lavorano sempre più insieme. L'approccio metodologico e culturale conosciuto come DevOps è stato adottato per favorire tale collaborazione e integrazione [1]. In questo ambito, l'Infrastructure as Code (IaC) assume un ruolo di notevole rilevanza, poiché rappresenta una pratica volta ad agevolare l'implementazione del continuous deployment attraverso la gestione e il provisioning dell'infrastruttura tramite la scrittura di codice. Questo approccio sostituisce l'esecuzione manuale di una serie di passaggi per la creazione e la configurazione di un ambiente, consentendo invece l'utilizzo di script o configurazioni dichiarative che descrivono l'infrastruttura desiderata. Nonostante l'Infrastructure as Code (IaC) stia diventando sempre più diffuso, non è ancora del tutto chiaro quali siano le migliori pratiche per garantire la manutenibilità del codice infrastrutturale, aspetto che invece è ben definito e ampiamente studiato per il codice sorgente di un software. Questa problematica è di particolare interesse per le organizzazioni in cui i malfunzionamenti dell'infrastruttura rappresentano una criticità, poiché i sistemi IT rivestono un ruolo vitale e non è accettabile alcuna interruzione operativa.

L'adozione di strumenti di predizione dei difetti per il codice infrastrutturale può fornire un notevole supporto ai team operativi, offrendo indicazioni sulle possibili presenze di errori negli script e permettendo una pianificazione più efficiente delle attività di testing e ispezione del codice.
L'obiettivo del lavoro di tesi consiste nell'introdurre nuove metriche per la creazione di un modello di predizione dei difetti nel linguaggio di IaC Terraform.
Il proposito di questa estensione è quello di condurre uno studio empirico di natura scientifica, in cui una selezione di progetti Terraform sarà oggetto di un'analisi dettagliata. Tale analisi mira a valutare le prestazioni di un classificatore basato su Random Forest all'interno del contesto specifico di Terraform.
In secondo luogo, l'intenzione è quella di valutare l'efficacia delle metriche di prodotto nell'identificazione di difetti nei progetti Terraform.

1.2 Motivazioni ed obiettivi

Sebbene la predizione dei difetti sia stata ampiamente esplorata e studiata, è importante notare che gran parte di questa ricerca si è concentrata principalmente sui linguaggi di programmazione convenzionali. Al contrario, la predizione dei difetti nei linguaggi di Infrastructure as Code (IaC) ha ricevuto una quantità notevolmente inferiore di attenzione e analisi.
Questa differenza di enfasi mette in evidenza un'opportunità significativa per approfondire e investigare l'applicazione di tecniche di predizione dei difetti specificamente nel contesto dei linguaggi di IaC. Nel presente studio, ci si è focalizzati su Terraform come linguaggio di IaC di rilievo, scelto per la sua crescente adozione nell'ambito aziendale.
L'obiettivo principale è adattare e ampliare lo studio condotto da Dalla Palma et al. [2] per applicarlo al contesto di Terraform. In particolare, si mira a valutare le prestazioni di un classificatore basato su Random Forest all'interno dell'ambito specifico di Terraform e a identificare le metriche di prodotto più efficaci per la predizione dei difetti specifici di Terraform.

1.3 Risultati ottenuti

I risultati ottenuti all'interno del corrente lavoro di tesi indicano che il classificatore Random Forest ha dimostrato la sua efficacia anche all'interno del contesto specifico di Terraform. In particolare, è emerso un punteggio medio di AUC-PR (Area Under the Precision-Recall Curve) di circa 0.90 e un punteggio medio di MCC (Matthews Correlation Coefficient) di circa 0.80. Questi risultati suggeriscono un alto livello di performance del modello nella predizione dei difetti nel codice Terraform.
Inoltre, l'analisi ha rivelato che le metriche più efficaci per la predizione dei difetti in Terraform sono TextEntropy (entropia del testo), LineSourceCode (numero di linee di codice sorgente) e NumTokens (numero di token).

1.4 Struttura della tesi

Tutti i temi ed i risultati introdotti sono approfonditi all'interno del presente lavoro di tesi che è strutturato come segue:

- **Capitolo 2 - Background e lavori correlati**, questo capitolo offre una panoramica sull'Infrastructure as Code (IaC) e analizza lo stato dell'arte in merito alla predizione dei difetti nel contesto dei linguaggi di IaC. Inoltre, stabilisce le basi concettuali su cui si fonda il lavoro di tesi.
- **Capitolo 3 - Metodologia di ricerca**, tale capitolo introduce nuove metriche specifiche per Terraform e illustra le modifiche apportate al tool RADON Defuse. Infine, presenta in dettaglio lo studio empirico condotto, delineando le domande di ricerca e la strategia adottata per affrontarle.
- **Capitolo 4 - Studio Empirico Esteso sul Codice Terraform**, questo capitolo illustra lo studio empirico condotto nel contesto di Terraform.
- **Capitolo 5 - Risultati ottenuti**, questo capitolo espone e discute i risultati emersi dallo studio empirico illustrato nel capitolo precedente.

- **Capitolo 6 - Minacce alla validità**, nel quinto capitolo vengono esaminate e illustrate le possibili minacce alla validità dello studio, nonché le misure adottate per mitigarle.
- **Capitolo 7 - Conclusioni**, questo capitolo riassume l'intero lavoro di ricerca, evidenziando i risultati ottenuti, e propone possibili sviluppi futuri.

CAPITOLO 2

Background e lavori correlati

2.1 Infrastructure-as-Code

La pratica nota come Infrastructure-as-code (IaC) consente la creazione, la gestione e il provisioning delle risorse infrastrutturali, come server, database o servizi cloud, attraverso l'utilizzo di codice. Questo approccio mira a sostituire le configurazioni manuali, che spesso comportano errori umani, con l'automazione, la scalabilità e la riproducibilità fornite dal codice. Nel contesto dell'IaC, esistono diverse tipologie di linguaggi, ognuna con le proprie caratteristiche e vantaggi distinti. Le due tipologie più comuni sono i linguaggi dichiarativi e i linguaggi imperativi.
I linguaggi dichiarativi sono orientati alla descrizione dell'infrastruttura desiderata, permettendo di specificare quali debbano essere le risorse create e come esse devono essere configurate senza dettagliare le azioni necessarie per raggiungere lo stato desiderato. Al contrario, i linguaggi imperativi richiedono l'esplicitazione dettagliata delle azioni specifiche da eseguire per ottenere un determinato stato.
Esistono linguaggi appositamente progettati per la gestione delle risorse in ambienti cloud specifici, come ad esempio Amazon Web Services (AWS) o Microsoft Azure. Allo stesso modo, esistono anche linguaggi che sono focalizzati sulla configurazione delle macchine o degli ambienti di sviluppo.

All'interno di questa sezione verranno presentati due linguaggi di IaC: Terraform ed Ansible. La decisione di focalizzarsi su Terraform è stata guidata dalla sua crescente popolarità nel contesto industriale e dall'assenza di studi precedenti sulla predizione dei difetti nel linguaggio Terraform, secondo le informazioni disponibili al momento della realizzazione della presente tesi.

2.1.1 Ansible

Ansible è una piattaforma open-source che automatizza il provisioning, la gestione della configurazione, il deployment delle applicazioni, l'orchestrazione e molti altri processi IT.
L'architettura di Ansible si basa su un modello client-server, in cui un nodo controllore si connette tramite Secure Shell (SSH) a nodi controllati e invia loro comandi per raggiungere lo stato specificato nel codice di automazione.
Utilizza un linguaggio di scripting basato su YAML, un linguaggio di serializzazione leggero, leggibile dall'uomo, progettato principalmente per essere facile da leggere e modificare [3]. L'utilizzo di YAML consente di definire in modo dichiarativo le configurazioni desiderate.
Un playbook è un file composto da una serie di play. Ogni play specifica i task che devono essere eseguiti e gli host su cui eseguirli. All'interno di una play, i task sono delle unità di azione che specificano il modulo da invocare e i parametri che occorrono per eseguire l'azione desiderata. Questo consente di definire in modo preciso quali azioni devono essere eseguite su un host remoto. I moduli, a loro volta, sono le unità di codice che Ansible esegue sui nodi controllati, oltre a quelli messi a disposizione da Ansible è possibile utilizzare dei moduli personalizzati.

Lo script riportato in Fig. 2.1 rappresenta un playbook composto da una singola play di nome "Update web servers", il gruppo "webservers" è designato come destinazione delle operazioni. La sezione "tasks" definisce una serie di azioni che devono essere eseguite sulla configurazione dei nodi target. All'interno dello snippet sono presenti due task, il task denominato "Apache is at the latest version" verifica che il pacchetto Apache sia installato nella versione più recente utilizzando il modulo

"ansible.builtin.yum".

Altro componente importante all'interno di Ansible sono i ruoli. Essi permettono di raggruppare in modo logico e riutilizzabile una serie di tasks, variabili, file di configurazione e altre risorse migliorando la modularità, la riutilizzabilità e la leggibilità del codice.

```
1  name: Update web servers
2  hosts: webservers
3  tasks:
4    - name: Apache is at the latest version
5      ansible.builtin.yum:
6        name: httpd
7        state: latest
8    - name: Apache service is enabled and running
9      ansible.builtin.service:
10       name: httpd
11       state: started
12       enabled: yes
```

Figura 2.1: Esempio di script Ansible

2.1.2 Terraform

Terraform è uno strumento di provisioning e gestione delle risorse che permette agli utenti di creare, configurare e gestire l'infrastruttura necessaria per le applicazioni software. Utilizzando il codice di infrastruttura, Terraform interagisce con le API fornite dai fornitori di servizi cloud e altre piattaforme, consentendo agli utenti di creare e configurare risorse quali macchine virtuali, istanze di database, bilanciatori di carico e molto altro.

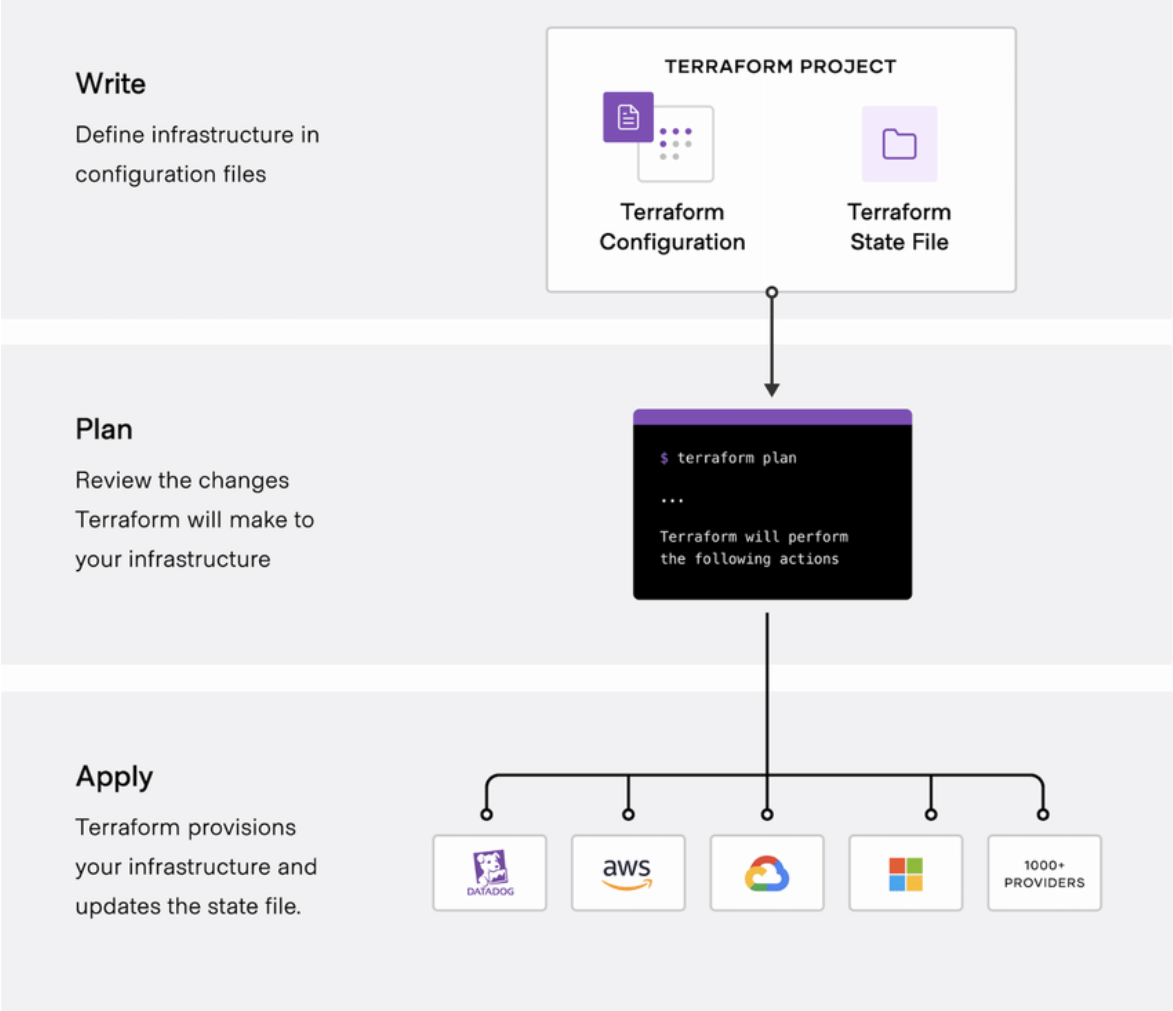

Figura 2.2: Le fasi di gestione delle risorse in Terraform

Il processo di gestione delle risorse di Terraform si struttura in tre fasi distinte, che sono eseguite in sequenza secondo l'ordine indicato e che sono illustare in figura 2.2.

1. **Write**, rappresenta la fase in cui si definisce l'infrastruttura all'interno di file di configurazione.

2. **Plan**, è la fase in cui Terraform esamina il codice sorgente scritto nella fase precedente e lo confronta con l'infrastruttura esistente, generando un piano di azione che descrive le modifiche che verranno apportate all'infrastruttura in modo da allinearla alla configurazione desiderata.

3. **Apply**, Terraform applica il piano d'azione generato nella fase precedente modificando l'infrastruttura in modo coerente e sicuro.

La configurazione, definita nel file principale "main.tf", rappresenta lo stato desiderato dell'infrastruttura da gestire. I file di configurazione in Terraform possono essere scritti sia nel formato YAML che nel linguaggio HCL (HashiCorp Configuration Language), che è un linguaggio open-source che è stato sviluppato da HashiCorp, quest'ultimo è maggiormente utilizzato e pertanto ci concentreremo su di esso all'interno del presente lavoro di tesi.
Uno dei componenti principali del linguaggio sono i **provider**, essi fungono da interfacce tra il codice Terraform e le API dei servizi cloud o altre piattaforme supportate. Consentono agli utenti di creare, configurare e gestire le risorse in diversi ambienti semplificando l'interazione con piattaforme diverse. Esistono poi le risorse, esse costituiscono gli elementi fondamentali dei blocchi di codice, offrono la possibilità al programmatore di specificare quali oggetti dell'infrastruttura si desidera creare, modificare o eliminare.

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-0c94855ba95c71c99"
  instance_type = "t2.micro"
  key_name      = "my-key-pair"
  tags = {
    Name = "example-instance"
  }
}
```

Figura 2.3: Esempio di script Terraform

Ad esempio, all'interno dello snippet illustrato in Figura 2.3, viene creata una risorsa di tipo "aws instance" interfacciandosi al provider "aws. Questa risorsa rappresenta un'istanza di macchina virtuale (EC2) all'interno del servizio AWS. È importante notare come ogni risorsa abbia i propri attributi e le proprie opzioni di configurazione.

Altro componente fondamentale di Terraform sono i moduli che consentono di organizzare il codice in unità modulari e riutilizzabili. Un modulo rappresenta una collezione di risorse correlate che possono essere gestite insieme. Questo approccio modulare semplifica la gestione di infrastrutture complesse, poiché permette di suddividere il codice in componenti più piccole. I moduli possono essere condivisi e riutilizzati all'interno di un progetto o persino tra progetti diversi, migliorando l'efficienza dello sviluppo e della manutenzione dell'infrastruttura.

2.2 Defect prediction

La predizione dei difetti software è uno dei temi di ricerca più popolari nell'ingegneria del software. Il suo obiettivo primario è quello di predire in anticipo i moduli software inclini a presentare difetti, al fine di consentire una migliore pianificazione degli sforzi per garantire la qualità del software. Tale approccio si propone di individuare in modo proattivo i punti critici all'interno del sistema software, al fine di concentrare le risorse di controllo e correzione sui componenti più suscettibili a difetti, ottimizzando così l'allocazione delle risorse impiegate per garantire la qualità del software. [4]

Molti degli studi esistenti riguardo la defect prediction nel software impiegano tecniche di machine learning. Il processo di costruzione di un modello di predizione dei difetti richiede una serie di passaggi fondamentali. Inizialmente, è necessario estrarre istanze di dati da archivi software quali ad esempio, sistemi di version control. Tali istanze possono essere classi, metodi o file. Segue poi l'estrazione di metriche che descrivono diversi aspetti del software e del processo adottato per svilupparlo. Le istanze vengono quindi etichettate come difettose o prive di difetti in base alla presenza o assenza di anomalie. A questo punto si raggruppa un insieme di queste istanze formando un training set con il quale il modello di predizione viene allenato. Una volta costruito, il modello riesce a classificare una qualsiasi istanza come difettosa o no.
Le metriche giocano un ruolo fondamentale per il corretto funzionamento del modello di predizione. All'interno del lavoro presentato da Radjenovi et al. [5] sono

illustrate alcune problematiche relative all'individuazione di metriche per la costruzione di modelli. Tra queste, le principali problematiche sono la mancanza di comparabilità delle metriche utilizzate, la limitata generalizzabilità delle metriche a contesti specifici, la disponibilità e qualità dei dati utilizzati, la scalabilità dei modelli per grandi progetti e la valutazione e confronto dei modelli. Affrontare tali sfide e colmare queste lacune rappresenta un'importante area di ricerca per migliorare l'accuratezza, e l'affidabilità pratica dei modelli di predizione dei difetti software.

Nonostante il tema della predizione dei difetti sia stato oggetto di numerosi studi e ricerche negli ultimi anni, è rilevante sottolineare che l'attenzione si è principalmente concentrata sul contesto dei linguaggi di sviluppo, mentre è stata dedicata una quantità limitata di sforzi alla predizione dei difetti nei linguaggi di Infrastructure as Code (IaC). Questa disparità di copertura evidenzia l'opportunità di esplorare e approfondire l'applicazione di tecniche di predizione dei difetti specificamente nel contesto dei linguaggi di IaC, come nel caso del presente lavoro di tesi su Terraform.

2.3 Stato dell'arte

All'interno della presente sezione verranno presentati e analizzati i risultati ottenuti da studi precedenti riguardanti la predizione dei difetti nell'ambito dell'IaC.

2.3.1 Toward a catalog of software quality metrics for IaC

Lo studio intitolato "Toward a catalog of software quality metrics for IaC" condotto da Dalla Palma, Di Nucci, Palomba e Tamburri [6] ha evidenziato che nonostante il crescente ruolo del codice di infrastruttura nella gestione delle risorse IT, le metriche di qualità tradizionalmente utilizzate nello sviluppo software potrebbero non essere completamente adattate per valutare la qualità del codice di infrastruttura.
Di conseguenza, l'obiettivo del paper è stato quello di colmare questa lacuna fornendo un catalogo di metriche specifiche per il codice di infrastruttura.
In particolare, all'interno di tale lavoro gli autori si sono concentrati sul linguaggio Ansible ed hanno per tale linguaggio introdotto 46 metriche.

Le metriche proposte possono essere suddivise in tre gruppi distinti. Il primo gruppo è costituito da otto metriche che rappresentano le metriche tradizionali utilizzate nei linguaggi orientati agli oggetti e che possono essere adattate ad Ansible e all'ambito generale dell'IaC. Il secondo gruppo consiste in quattordici metriche derivate da altre metriche precedentemente investigate in lavori riguardanti linguaggi diversi, come ad esempio il lavoro condotto da Rahman e Williams et al. [7] su Chef e Puppet. Alcune di queste metriche sono state tradotte ed applicate ad Ansible. Infine, il terzo gruppo riguarda ventiquattro metriche specificamente correlate alle pratiche consigliate e non consigliate in Ansible. Queste metriche sono state introdotte appositamente per Ansible all'interno dello studio.
La Tabella 2.1 presenta un elenco delle 46 metriche introdotte, accompagnate da informazioni dettagliate sul loro nome e sulle modalità di calcolo. Ad esempio, la metrica "NumPlays" indica il numero di Play all'interno di un Playbook Ansible. Per determinare i valori, gli autori hanno proposto delle euristiche, ad esempio contare il numero di occorrenze della parola chiave "hosts", poiché questa parola chiave è obbligatoria e viene specificata una sola volta all'interno di ciascuna Play, questa è una delle metriche che appartiene al secondo gruppo descritto precedentemente.

Il presente lavoro di tesi si basa sulle metriche precedentemente sviluppate per Ansible, cercando di estendere e adattare tali metriche per l'applicazione a Terraform. Allo stesso tempo, si è prestata attenzione alla necessità di introdurre nuove metriche specifiche per Terraform, al fine di cogliere gli aspetti peculiari di tale linguaggio che non possono essere adeguatamente rappresentati attraverso una traduzione diretta. Questo adattamento e specializzazione delle metriche per Terraform è stato motivato dalla consapevolezza che le metriche sviluppate per un determinato linguaggio o contesto potrebbero non essere direttamente trasferibili ad un altro. Ogni linguaggio o framework di automazione delle infrastrutture presenta caratteristiche uniche e specifiche che richiedono un'analisi mirata e l'identificazione di metriche rilevanti per la valutazione della qualità del codice. Di conseguenza, al fine di ottenere un modello di predizione affidabile per Terraform, si è scelto di specializzare le metriche esistenti, tenendo conto delle caratteristiche distintive di tale linguaggio.

Tabella 2.1: Riepilogo delle metriche identificate e implementate all'interno del lavoro "Toward a catalog of software quality metrics for IaC".

Misura	Tecnica di misurazione	Ambito
LinesBlank	Linee vuote totali	Generale
LinesComment	Conteggio degli statement che iniziano con #	Generale
LinesSourceCode	Numero totale di linee di codice eseguibile	Generale
NumCommands	Numero di occorrenze sintattiche di command, expect, psexec, raw, script, shell, e telnet	Generale
NumConditions	Conteggio di occorrenze negli statement when di is, in, ==, !=, >, >=, <, <=	Generale
NumDecisions	Conteggio di occorrenze negli statement when di and, or, not	Generale
NumDeprecatedKey	Conteggio di occorrenze delle keyword deprecate	Generale
NumEnsure	Conteggio di occorrenze in when di parole che matchano con la regex `"\w+.stat.\w+"`	Generale
NumFile	Conta il numero di keyword "file"	Generale
NumFileMode	Conta il numero di keyword "mode"	Generale
NumLoop	Conta il numero di loop	Generale
NumMathOperations	Conta il numero di occorrenze di +, -, /, //, %, *, **	Generale
NumParameters	Conta il numero di keyword che rappresentano un modulo	Generale
NumPaths	Conta le keyword paths, src e dest	Generale
NumRegex	Conta il numero di regexp	Generale
NumSSH	Conta il numero di keyword ssh_authorized_key	Generale
NumSuspComments	Conta il numero di commenti con TODO, FIXME, HACK, XXX, CHECKME, DOCME, TESTME, o PENDING	Generale
NumUrls	Conta il numero di url	Generale
NumUrls	Conta il numero di parole separate da uno spazio bianco	Generale
NumUserInteractions	Conta il numero di keyword "prompt"	Generale
NumVariables	Somma len(vars) nelle play	Generale
NumPlays	Conta il numero di keyword "hosts"	Playbook

Misura	Tecnica di misurazione	Ambito
NumRoles	Lunghezza della sezione roles nel playbook	Playbook
AvgPlaySize	LinesSourceCode(playbook)/NumPlays	Playbook
AvgTaskSize	LinesSourceCode(tasks)/NumTasks	Playbook
NumBlocks	Conta il numero di block	Playbook
NumBlocksErrHand	Conteggio del numero di sezioni block-rescue-always	Playbook
NumDeprModules	Conta il numero di moduli deprecati	Playbook
NumDistinctModules	Conteggio del numero di moduli non mantenuti dalla community	Playbook
NumExternalModules	Conteggio del numero di moduli mantenuti dalla community	Playbook
NumFactModules	Conteggio del numero di moduli listati nella documentazione	Playbook
NumFilters	Conta il numero di | all'interno dell'espressione {{*}}	Playbook
NumIgnoreErrors	Conta il numero di keyword ignore_errors	Playbook
NumImportPlaybook	Conta il numero di keyword import_playbook	Playbook
NumImportRole	Conta il numero di keyword import_role	Playbook
NumImportTasks	Conta il numero di keyword import_tasks	Playbook
NumInclude	Conta il numero di keyword include	Playbook
NumIncludeRole	Conta il numero di keyword include_role	Playbook
NumIncludeTasks	Conta il numero di keyword include_tasks	Playbook
NumIncludeVars	Conta il numero di keyword include_vars	Playbook
NumKeys	Conta il numero di keys nel dizionario che rappresentano un playbook o un tasks	Playbook
NumLookups	Conta il numero di lookup	Playbook
NumNameWithVar	Conta il numero di nome che hanno un match con la regexp ".*{{\w+}}.*"	Playbook
NumTasks	len(tasks) in un playbook o in un file di task	Playbook
NumUniqueNames	Conta il numero di nomi con valori unici	Playbook

2.3.2 RADON Defuse per la predizione dei difetti

RADON Defuse è un'applicazione web presentata per la prima volta da Dalla Palma, Di Nucci, Palomba e Tamburri nel paper [2] che mira a predire i difetti all'interno del codice Ansible. L'applicazione utilizza una combinazione di metriche di prodotto e di processo per sviluppare modelli accurati.
Essa è formata da quattro componenti:

- **Github IaC repositories collector**, utilizzato per collezionare repository pubbliche contenenti codice Ansible tramite le API di GitHub basate su GraphQL. Esso permette di personalizzare la query di ricerca GraphQL permettendo di ottenere informazioni come nome, descrizione, URL e directory principali, da repository che corrispondono ai criteri di selezione definiti dall'utente che sono: la data di creazione, la data dell'ultima push, il numero minimo di rilasci, stelle, follower, ed il linguaggio predominante. Le repository archiviate, replicate e forked vengono escluse.

- **Repository Scorer**, lo strumento riceve in input il percorso locale (o l'URL remoto) di una repository Git, calcola il numero di push events all'interno del default branch negli ultimi 6 mesi, il numero di release, la percentuale di script di IaC all'interno della repository, il numero di contributori, se la repository utilizza servizi di CI, la percentuale di commenti, il numero medio di commit per software, il numero medio di errori per mese, se la repository ha una licenza, ed il numero di LOC. L'insieme di queste metriche permette di selezionare repository di progetti ben ingegnarizzati e che non siano inattive.

- **IaC repository miner**, estrae gli script di IaC suscettibili a difetti e neutri da una repository. Successivamente, raccoglie un ampio insieme di metriche che comprendono metriche strutturali che sono quelle metriche introdotte all'interno del paper illustrato nella sezione precedente del documento (2.3.1), delta metrics per comprendere il cambiamento delle metriche appena citate tra due release diverse e infine delle metriche di processo, le quali considerano aspetti relativi al processo di sviluppo.

- **IaC defect predictor** , utilizza i framework Python scikit-learn e imblearn per creare una pipeline che bilancia e pre-elabora il dataset, addestra e convalida i modelli di apprendimento automatico, e li utilizza per fare previsioni su nuove istanze. Questo modulo sfrutta diverse configurazioni, come selezione delle caratteristiche, normalizzazione dei dati, bilanciamento delle classi, scelta dei classificatori e degli iperparametri, al fine di trovare la configurazione ottimale per la predizione dei difetti negli script IaC. L'uso di scikit-learn e imblearn semplifica il processo di costruzione e valutazione dei modelli, consentendo di adattarsi alle specifiche del dataset e migliorare l'accuratezza delle predizioni dei difetti.
  La Figura 2.4 illustra l'architettura di RADON Defuse. Le repository raccolte dal *Github IaC repositories collector* vengono passate in input al *Repository scorer* per selezionare le repository rilevanti. Successivamente, l'*IaC repository* miner estrae informazioni dalle repository selezionate, ottenendo un insieme di script IaC soggetti a difetti e script IaC neutri. Questi dati vengono utilizzati dall'*IaC defect predictor* per costruire e valutare i modelli di predizione dei difetti. Un elemento di rilevanza fondamentale è l'utilizzo del tool *AnsibleMetrics* all'interno del componente *IaC repository miner*. Tale strumento è responsabile del calcolo delle metriche strutturali sul codice sorgente Ansible.
  L'obiettivo principale di questa tesi è l'estensione dell'applicativo *RADON Defuse* per consentire la predizione dei difetti nel linguaggio Terraform. Tale estensione permetterà di applicare le tecniche di defect prediction a Terraform, ampliando così le funzionalità del framework esistente.
  Nel contesto del presente lavoro di tesi, è fondamentale apportare modifiche significative anche al tool *AnsibleMetrics*, al fine di creare un nuovo strumento chiamato *"TerraformMetrics"*. Questo nuovo tool sarà appositamente progettato per il calcolo delle metriche specifiche degli script Terraform. Tale strumento modellato su misura permetterà, quindi, di condurre un'analisi accurata e approfondita delle metriche sul codice sorgente Terraform e di svolgere le successive attività di predizione dei difetti in modo efficace e preciso.

Utilizzando RADON Defuse, gli autori del paper hanno condotto una serie di

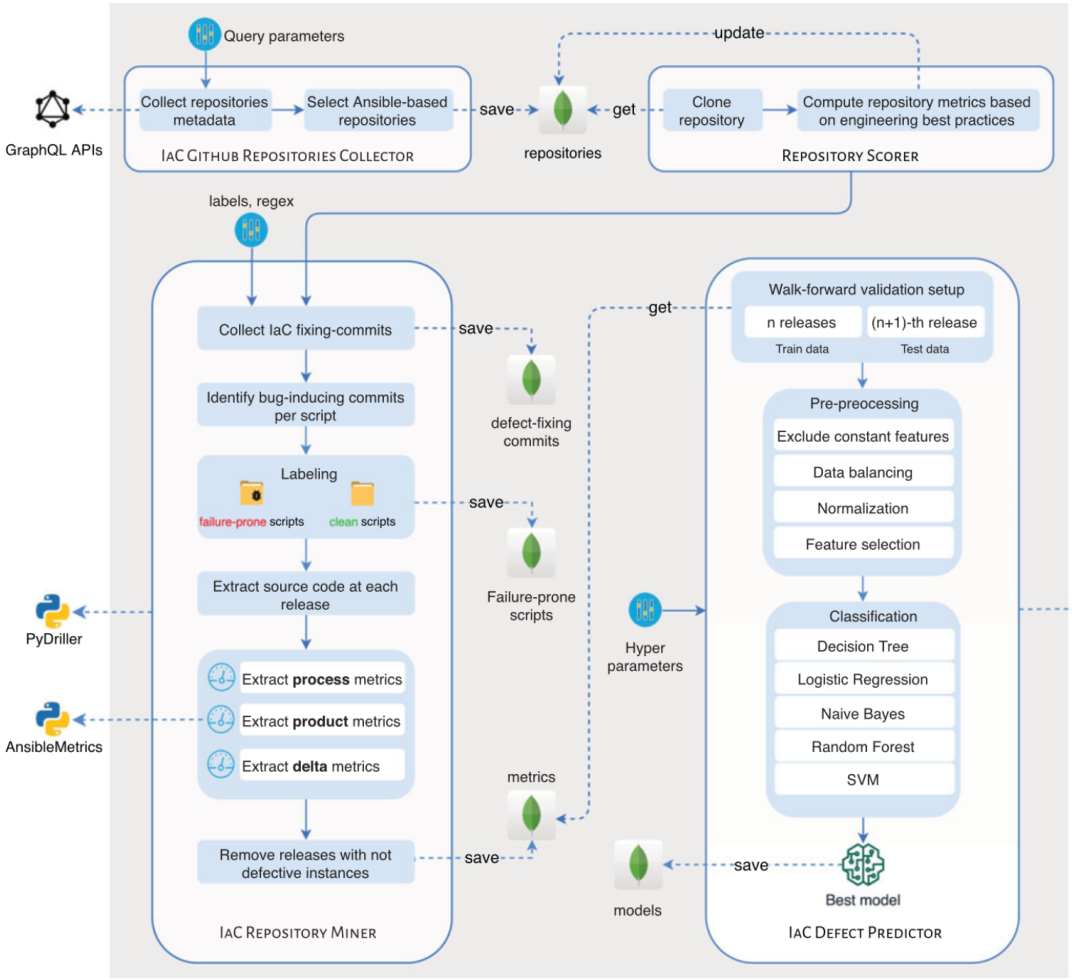

Figura 2.4: Architettura del framework RADON

studi empirici al fine di valutare l'impatto del modello di machine learning sulle prestazioni nella predizione dei difetti. È emerso che esistono differenze tra i vari modelli, e la tecnica più performante è risultata essere il *Random Forest*. Inoltre, gli esperimenti hanno dimostrato con elevata confidenza che un modello addestrato sulle metriche strutturali del codice sorgente ottiene risultati migliori rispetto a un modello addestrato su altre metriche. In particolare, è stato evidenziato che le metriche più rilevanti nel contesto del codice Ansible sono NumTokens, TextEntropy e LinesCode.

2.3.3 Predizione dei difetti in Terraform

Allo stato dell'arte, sono stati condotti numerosi studi e ricerche nel campo della predizione dei difetti per il codice tradizionale delle applicazioni, mentre sono stati effettuati meno studi specifici per l'Infrastructure-as-Code. Tuttavia, è importante sottolineare che, fino ad oggi, non sono stati effettuati specifici lavori di defect prediction per il linguaggio Terraform.
La maggior parte degli studi si è concentrata su linguaggi di scripting come Ansible, Chef e Puppet, ma non sono ancora stati condotti esperimenti mirati alla predizione dei difetti nel contesto di Terraform. Pertanto, esiste una lacuna nella letteratura riguardante la defect prediction per il codice Terraform, e questo lavoro di ricerca si propone di colmare tale vuoto estendendo e applicando *RADON Defuse* per il calcolo delle metriche e la predizione dei difetti specificamente per lo scenario di Infrastructure-as-Code basato su Terraform.

CAPITOLO 3

Estensione dell'applicativo RADON Defuse

All'interno di questo capitolo, vengono esposte in dettaglio tutte le modifiche che sono state apportate al tool *RADON Defuse* descritto nella sezione 2.3.2 del capitolo precedente. Tali modifiche sono state effettuate al fine di sviluppare modelli predittivi dei difetti per il linguaggio Terraform.

3.1 Ricerca di repositories Terraform

Il Github Repositories Collector è stato sviluppato con l'obiettivo di raccogliere repository pubbliche da GitHub, offrendo all'utente la possibilità di specificare diversi parametri di ricerca. Tra questi parametri figurano la data di creazione, la data dell'ultima push, il numero minimo di rilasci, stelle, follower e il linguaggio predominante. Per quanto concerne i linguaggi infrastrtturali, inizialmente, al momento della selezione del linguaggio, il tool consentiva esclusivamente la ricerca di repository contenenti codice Ansible. Tuttavia, sono state apportate modifiche appropriate per consentire la ricerca anche di repository pubbliche che contengono codice Terraform.

3.2 Modifiche al Repository Scorer

Il repository "scorer" nella sua versione originale è progettato per valutare la rilevanza e l'attività delle repository. Una delle metriche utilizzate per questa valutazione compare il conteggio degli eventi di "push" verificatisi nei sei mesi precedenti. Questa metrica è un indicatore dell'attività continua di una repository.
Tra le diverse metriche calcolate dallo strumento, vi è la percentuale di codice riconducibile all'Infrastructure as Code.
Il componente è stato modificato per includere, nel calcolo di questa metrica, anche i file con estensione ".tf", che sono specifici di Terraform. Questo ha permesso di ottenere una stima più accurata della presenza di codice IaC all'interno delle repository esaminate. Tali metriche sono descritte in maniera profonda all'interno del lavoro presentato da Munaiah et al. [8]

3.3 Repositories Miner e TerraformMetrics

Il componente Repositories Miner inizia il suo processo collezionando i fixing commits, che sono i commit che risolvono difetti introdotti in commit precedenti noti come bug inducing commit. Sulla base di queste informazioni, l'algoritmo assegna etichette agli script identificandoli come soggetti a difetti o neutrali. Questo processo deve essere completato per condurre la defect prediction sugli script Terraform.
Tuttavia, il processo prosegue con gli step successivi. Dopo l'estrazione del codice sorgente di ciascuno degli script, il Miner estrae metriche dagli script Ansible utilizzando il componente denominato *AnsibleMetrics* [9].
Poiché era necessario raccogliere metriche anche dagli script Terraform, è stato sviluppato un nuovo componente denominato *TerraformMetrics*. Analogamente al funzionamento del componente *AnsibleMetrics* per il codice Ansible, il nuovo componente *TerraformMetrics* estrae una serie di metriche di prodotto dal codice Terraform.
Al fine di soddisfare tale esigenza, è stato ritenuto necessario definire nuove metriche specifiche da estrarre per il linguaggio Terraform.
Le nuove metriche sono state formulate prendendo spunto da quelle precedentemen-

te definite nel lavoro di Dalla Palma et al. [6].

```
resource "aws_instance" "web_server" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"

  provisioner "remote-exec" {
    inline = [
      "echo Configuring web server...",   #Configurazione del server web
      "sudo apt update",
      "sudo apt install -y apache2",
    ]
  }

  provisioner "file" {
    source      = "local/path/to/files"
    destination = "/var/www/html/"
  }
}

resource "aws_instance" "database_server" {
  ami           = "ami-98765432"
  instance_type = "t2.micro"

  provisioner "remote-exec" {
    inline = [
      "echo Configuring database server...",
      "sudo apt update",
      "sudo apt install -y mysql-server",
    ]
  }
}
```

Figura 3.1: Un esempio di codice Terraform, LinesSourceCode=30, LinesComment=1, LinesBlank=4, NumResources=2.

```
resource "null_resource" "example" {.

  # Provisioner di tipo http
  provisioner "http" {
    request_method = "GET"
    request_url    = "https://example.com/api/data"
  }

  module {
    source = "./path/to/module"
  }.
}
```

Figura 3.2: Un esempio di codice Terraform, in cui NumDataSourceHttp=1, NumModules=1

Durante il processo di definizione, si è cercato di individuare una corrispondenza tra le componenti del linguaggio Ansible e quelle del linguaggio Terraform, al fine di tradurre in modo coerente ciascuna metrica da un linguaggio all'altro. Inoltre, al fine di coprire tutti gli aspetti del linguaggio Terraform, sono state definite appositamente nuove metriche "ad hoc". Queste metriche sono state appositamente sviluppate per essere applicate nell'ambito di Terraform, con l'obiettivo di identificare in modo accurato le feature peculiari e significative di questo linguaggio..
Complessivamente, il processo di definizione delle nuove metriche per Terraform è iniziato adattando prima le metriche già presenti per Ansible al contesto di Terraform e andando successivamente a definire delle metriche specifiche per il contesto in analisi. Si possono di conseguenza introdurre le seguenti 25 metriche:

Catalogo delle metriche per Terraform tradotte da quelle già definite per Ansible

- **LinesBlank**, **LinesComment** e **LinesSourceCode**: per quantificare rispettivamente il numero di righe vuote presenti nel codice sorgente, il numero di righe di commenti e il numero di linee di codice effettivo. Si è rilevato che queste metriche potevano essere mappate direttamente da Ansible a Terraform, poiché la loro applicazione risulta altamente generalizzabile. All'interno dell'esempio di codice in Figura 3.1 LinesSourceCode=30, LinesComment=1, LinesBlank=4.

- **NumConditions**: si contano il numero di condizioni presenti all'interno dello script. Questa metrica ha subito una leggera modifica rispetto a quella di Ansible in quanto gli operatori relazionali che esistono in Terraform sono esclusivamente i seguenti: ==, !=, >, >=, <, <=, in, contains. Tuttavia, il significato della metrica rimane lo stesso.

  Più sono presenti condizioni più complesso risulta mantenere la blueprint.

- **NumDecisions**: si contano il numero di decisioni presenti all'interno dello script. Questa metrica ha subito una leggera modifica rispetto a quella definita per Ansible in quanto gli operatori booleani in Terraform sono i seguenti &&, || e ! . Tuttavia, il significato della metrica rimane lo stesso.

  Più sono presenti decisioni più complesso risulta mantenere la blueprint.

- **NumResources**: è una metrica utilizzata per misurare il numero di risorse presenti all'interno di uno script Terraform. Tale metrica rappresenta un adattamento della metrica *NumTasks* definita per Ansible, che conta il numero di task presenti in un playbook. Questa metrica è stata applicata a Terraform considerando le analogie tra il concetto di risorsa in Terraform e quello di task in Ansible. Entrambi i concetti specificano le azioni principali che uno script deve eseguire. Nell'esempio in Figura 3.1, NumResources=2.

  Maggiore è il numero di risorse più complicato diventa manutenere la blueprint.

- **AvgResourcesSize**: continuando con l'analogia tra le task in Ansible e le risorse in Terraform, è possibile tradurre la metrica *AvgTasksSize* che determina la taglia media di una task all'interno di uno script Ansible. In questo contesto, possiamo definire la metrica *AvgResourcesSize* per Terraform, che calcola la taglia media di una risorsa all'interno di uno script Terraform. Nell'esempio in Figura 3.1, AvgResourcesSize=13.

  Più alto è questo valore più è difficile mantenere la blueprint.

- **NumRemoteExec**: un provisioner di tipo "remote exec" permette di eseguire comandi su macchine remote tramite connessione SSH. Questo meccanismo risulta essere molto simile al modulo command definito in Ansible, poiché

entrambi consentono di eseguire comandi su macchine remote dopo la loro creazione o aggiornamento. Per tale ragione, è stato possibile ricavare la metrica *NumRemoteExec* da quella precedentemente individuata per Ansible chiamata *NumCommand*. Nell'esempio in Figura 3.1, NumRemoteExec=2.

La presenza di un numero elevato di provisioner "remote exec" rende più complicato il mantenimento della blueprint.

- **NumFile**: si conta il numero di occorrenze di provisioner di tipo "file". Il provisioner di tipo file in Terraform è un meccanismo che consente di copiare file da una macchina locale o da una posizione remota sulla macchina virtuale o il nodo di destinazione definito in Terraform. Può essere utilizzato per gestire il trasferimento di file durante il provisioning delle risorse di infrastruttura. Risulta essere molto simile al modulo file presente in Ansible che permette la gestione di file, dierctory e link simbolici.
  Nell'esempio in Figura 3.1, NumFile=1.

  Un numero elevato di provisioner di tipo "file" può portare a uno script Terraform più complesso e difficile da mantenere. Diventa complicato gestire e tenere traccia di numerosi provisioner con le relative configurazioni e dipendenze.

- **NumPermissions**: si conta il numero di occorrenze del parametro file_permission all'interno del blocco provisioner "file", tale parametro viene utilizzato per impostare le autorizzazioni di accesso per un file creato o copiato su una macchina remota. Questa metrica è stata ottenuta partendo da quella chiamata *NumFileMode* definita per Ansible.

  Più alto è il valore di questa metrica più si può considerare la blueprint difficile da testare.

- **NumModules**: in Ansible, le direttive *import* e *include* sono utilizzate per consentire ai programmatori di dividere playbook di grandi dimensioni in file più piccoli. In Terraform, invece, si fa uso dei *moduli* per organizzare e riutilizzare il codice in modo modulare. Nello specifico, le metriche *NumImport* e *NumInclude* sono state tradotte in una metrica denominata *NumModules*, che conta il numero di moduli utilizzati all'interno di uno script Terraform. Nell'esempio in Figura

3.2, NumModules=1.

Un valore più alto di questa metrica indica una maggiore riusabilità del codice. Tuttavia, è importante notare che un aumento della riusabilità può comportare una maggiore complessità nella manutenzione dello script.

- **NumDataSourceHttp**: questa metrica conta il numero di data source 'http' in Terraform. I data source 'http' consentono di recuperare informazioni da una risorsa HTTP remota durante la fase di pianificazione o applicazione delle configurazioni. Questo è analogo al modulo 'uri' trattato nella metrica di Ansible denominata *NumURLs*. Nello snippet riportato in Figura 3.2, NumDataSourceHttp=1.

  Più Urls ci sono specificati all'interno dei data source, maggiore è la complessità nella manutenzione dello script.

- **NumErrHandling**: si conta il numero di funzioni try. Tale funzione consente di gestire in modo più flessibile eventuali errori o valori nulli durante l'elaborazione delle configurazioni Terraform. La metrica è stata ricavata partendo da *NumBlocksErrorHandling* di Ansible.

  Più alto è il valore di questa metrica più si può considerare la blueprint difficile da testare poichè, occorre considerare tutti i possibili scenari di errore e verificare che vengano gestiti correttamente.

- **NumTokens**: **NumRegex** e **NumPaths**, per quantificare rispettivamente il numero di token, il numero di espressioni regolari e il numero di path, presenti nel codice sorgente. Si è rilevato che anche queste metriche potevano essere mappate direttamente da Ansible a Terraform, poiché la loro applicazione risulta altamente generalizzabile.

  Più alto è il valore di queste metriche più si può ritenere la blueprint complessa.

- **NumMathOperations**: per quantificare rispettivamente il numero di operazioni matematiche all'interno dello script si conta il numero di occorrenze degli

operatori + - * / % ^.

Più alto è il valore di questa metrica più si può ritenere complesso testare e mantenere la blueprint.

- **NumVariables**: analogamente a ciò che è stato definito per Ansible è stata introdotta una metrica che conta il numero di variabili specificate all'interno di uno script di configurazione Terraform.

Aggiornare o modificare variabili in un codice con un alto numero di variabili richiede attenzione e può comportare cambiamenti in molti punti del codice. Questo può rendere la manutenzione più lunga e suscettibile a errori.

Tabella 3.1: Catalogo delle metriche per Terraform tradotte da quelle già definite per Ansible

Metrica	Tecnica per l'estrazione	Metrica Ansible
LinesBlank	Conta il numero di righe vuote.	LinesBlank (UGUALE)
LinesComment	Conta il numero di occorrenze di # e di /* .	LinesComment (UGUALE)
LinesSourceCode	Conta il numero totale di righe di codice eseguibile.	LinesSourceCode (UGUALE)
TextEntropy	Calcolato con la seguente formula $\sum_{t \in \text{tokens(script)}} p(t) \log_2(p(t))$	TextEntropy (UGUALE)
NumConditions	Conta il numero totale di occorrenze di ==, *!=*, >, >=, <, <=, *in*, *contains*.	NumConitions
NumDecisions	Conta il numero totale di occorrenze di *&&*, *||* o *!* .	NumDecisions
NumResources	Conta il numero totale di occorrenze di *resource* in uno script.	NumTasks
AvgResourcesSize	Calcola LinesSourceCode(Resources) / NumResources.	AvgTasksSize

Metrica	Tecnica per l'estrazione	Metrica Ansible
NumRemoteExec	Conta il numero di occorrenze di provisioner di tipo "remote exec".	NumCommands
NumFile	Conta il numero di occorrenze di provisioner di tipo "file".	NumFile
NumPermissions	Conta il numero di occorrenze del parametro *file_permissions* all'interno di un provisioner "file".	NumFileMode
NumModules	Conta il numero di occorrenze della keyword "module"	NumInclude NumImport
NumDataSource Http	Conta il numero di occorrenze dei data source di tipo "http".	NumURLs
NumErrHandling	Conta il numero di occorrenze dei funzioni "try".	NumBlocksError Handling
NumTokens	Conta il numero di parole separate da uno spazio bianco.	NumTokens (UGUALE)
NumRegex	Conta il numero di espressioni regolari.	NumRegex (UGUALE)
NumPaths	Conta il numero di path specificate all'interno dello script.	NumPaths (UGUALE)
NumMathOperations	Conta il numero totale di occorrenze di +, -, *, /, %, ^.	NumMathOperations
NumVariables	Conta il numero totale di occorrenze della keyword "variable".	NumVariable

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web_server" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}

data "aws_ami" "latest_ubuntu" {
  most_recent = true
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
  }
  owners = ["099720109477"]
}

output "web_server_ip" {
  value     = aws_instance.web_server.private_ip
  depends_on = [aws_instance.web_server]
}
```

Figura 3.3: Un esempio di codice Terraform, in cui NumOutputs=1, NumDataSources=1, NumProviders=1 e NumDependsOn=1.

Catalogo delle metriche introdotte appositamente per Terraform

Le metriche descritte di seguito sono state introdotte durante il lavoro di tesi attuale per affrontare tutte le peculiarità del linguaggio Terraform, poiché la mera traduzione delle metriche precedentemente sviluppate per Ansible risultava insufficiente.

- **NumOutputs**: conta il numero di *Terraform output*. Gli output definiti in uno script specificano le informazioni che vengono esposte al di fuori dello script stesso, consentendo il loro utilizzo da parte di altri moduli o ambienti. Nell'esempio in Figura 3.3, NumOutputs=1.

Un alto numero di output può complicare l'interfaccia pubblica del modulo e rendere meno chiare le informazioni esposte.

- **NumProviders**: conta il numero di *Terraform provider*. Il numero di provider utilizzati nello script indica quante piattaforme o servizi di cloud provider sono coinvolte nella configurazione. Nell'esempio in Figura 3.3, NumProviders=1.

  Un elevato numero di provider può aumentare la complessità del codice a causa delle diverse sintassi e opzioni di configurazione.

- **NumDependsOn**: conta il numero di occorrenze della keyword *depends on*. Il numero di dipendenze tra risorse indica quanto le risorse sono interconnesse tra loro. Nell'esempio in Figura 3.3, NumDependsOn=1.

  Un alto numero di dipendenze può rendere le modifiche al codice più complesse a causa delle possibili conseguenze su altre risorse.

- **NumDataSources**: Conta il numero di data sources usati. I data sources sono utilizzati per ottenere informazioni da una piattaforma o un servizio esterno. Nell'esempio in Figura 3.3, NumDataSources=1.

  Un numero elevato di data source può aumentare la complessità dello script a causa delle possibili interazioni con altre risorse.

- **NumDynamicBlock**: Conta il numero di dynamic block. I dynamic block permettono di creare blocchi di configurazione in modo dinamico all'interno di un modulo, il che può rendere il codice più flessibile e adattabile.

  Un'elevata presenza di blocchi dinamici può rendere difficile la comprensione e la manutenzione del codice.

- **NumLocals**: Conta il numero di *variabili locali*. Le variabili locali sono variabili che vengono utilizzate all'interno di un modulo o di una configurazione per

memorizzare valori intermedi o calcolati.

La creazione eccessiva di variabili locali, soprattutto quando sono denominate in modo poco chiaro o senza una reale necessità, può compromettere la leggibilità e la manutenibilità del codice. Questo fenomeno può derivare da una progettazione non ottimale delle variabili locali, rendendo il codice meno comprensibile e più complesso da gestire.

Tabella 3.2: Catalogo delle nuove metriche introdotte appositamente per Terraform

Metrica	Tecnica per l'estrazione
NumOutputs	Conta il numero di occorrenze della keyword "output".
NumProviders	Conta il numeero di occorrenze della keyword "provider".
NumDependsOn	Conta il numero di occorrenze della keyword "DependsOn".
NumDataSources	Conta il numero di occorrenze della keyword "data".
NumDynamic Block	Conta il numero di occorrenze della keyword "dynamic".
NumLocals	Conta il numero di variabili dichiarate all'interno di un blocco "local".

Oltre all'introduzione di *TerraformMetrics* per l'estrazione delle metriche dagli script Terraform, è stato necessario apportare ulteriori modifiche al componente Repositories Miner. Tali modifiche sono state implementate al fine di consentire l'estrazione esclusiva dei fixing commits dai file di script Terraform contenuti all'interno della repository selezionata per l'analisi e di scartare tutti i file che non riguardavano Terraform.

In aggiunta, all'interno della classe BaseMiner, precedentemente definita nel file base.py, sono stati identificati tre metodi che necessitavano di personalizzazione

durante l'implementazione del TerraformMiner. Tale adattamento è stato concepito con l'obiettivo di potenziare la capacità di rilevazione dei fixing commits nel contesto specifico di Terraform. Tuttavia, nel corso della presente tesi, è stato possibile sviluppare solamente due di tali metodi.

ll primo di questi metodi è stato denominato *is_data_changed*, ed è stato progettato per effettuare una verifica approfondita delle modifiche apportate da un commit specifico ai dati all'interno di una risorsa in un ambiente Terraform.

Il secondo metodo implementato è denominato *is_include_changed*, il quale si concentra sulla valutazione delle modifiche apportate ai dati all'interno di un blocco module in uno script Terraform.
Va notato che il terzo metodo originariamente previsto, il cui obiettivo era rilevare modifiche relative al modulo "service" di Ansible, non è stato implementato. Questa decisione è stata motivata dal fatto che non è stato individuato alcun componente all'interno del linguaggio Terraform che corrispondesse a un concetto analogo al modulo "service" di Ansible, il quale è principalmente utilizzato per la gestione dei servizi di sistema su host remoti. Di conseguenza, è stato ritenuto non necessario definire tale metodo all'interno della classe TerraformMiner sviluppata per l'estensione del componente Repositories Miner.

3.4 Modifiche al Defect Predictor

Il componente noto come *Defect Predictor* ha il compito di ricevere i file potenzialmente soggetti a difetti (failure prone file) e le metriche estratte per ciascuno di essi dal modulo *Repositories Miner*. La sua funzione principale consiste nell'addestrare modelli di machine learning in grado di determinare se un file in ingresso può essere considerato un failure prone file o meno.
Nella versione originale di questo strumento, veniva eseguito un processo di preelaborazione dei dati. In questa fase, la normalizzazione dei dati era condotta mediante l'utilizzo della classe *MinMaxScaler* fornita dalla libreria scikit-learn. Questo scaler è utilizzato per trasformare le features in modo che si trovino in un intervallo

specifico, di solito tra 0 e 1. Questo è utile perché assicura che tutte le feature abbiano lo stesso peso e non siano influenzate dalle differenze nelle scale dei dati.
Per la selezione delle feature, veniva impiegato l'algoritmo "SelectKBest". Un metodo di selezione delle feature che mira a identificare le migliori 2 feature utilizzando il test del chi-quadro (chi2) come metrica per selezionarle.
Successivamente, il "Defect Predictor" procedeva con l'addestramento di classificatori di tipo "Decision Tree". Questa scelta era motivata, probabilmente, dalla velocità di addestramento dei modelli basati su alberi decisionali, che tendono ad essere più efficienti in termini computazionali rispetto ad alcuni altri algoritmi di machine learning.

All'interno del prsente lavoro si è deciso di lasciare inalterata la parte di normalizzazione dei dati.
Per la parte di feature selection invece, si è scelto di utilizzare la *Recursive Feature Elimination with Cross Validation*, una tecnica di selezione delle feature in cui vengono addestrati modelli machine learning con diverse combinazioni di feature e valutati tramite cross-validation. Le feature meno informative vengono eliminate gradualmente fino a ottenere un set ottimale per la costruzione del modello. La ragione principale di questa scelta risiede nella natura più avanzata di RFECV nella selezione delle feature. RFECV non si limita a utilizzare metriche statistiche per valutare la rilevanza delle feature, riuscendo a catturare relazioni complesse tra le feature e la variabile target.
In conclusione, è stato scelto di utilizzare il classificatore "RandomForest". Tale decisione è stata basata sulla ricerca condotta da Dalla Palma et al. [2], che ha dimostrato l'efficacia dei classificatori di tipo RandomForest nella predizione dei difetti nel contesto di Terraform. Pertanto, questa scelta è stata fatta come punto di partenza per la presente indagine.

Al termine della fase di addestramento, il sistema archivia i risultati ottenuti dall'addestramento del modello in un'apposita istanza di un database in Firestore. Questi risultati includono le feature selezionate durante il processo di addestramento e le prestazioni del modello valutate durante il processo di validazione. Tale conservazione dei dati è fondamentale per mantenere un registro accurato delle informazioni

cruciali relative all'addestramento del modello.
In aggiunta, il modello stesso viene preservato in un file nel formato joblib. Questa operazione di persistenza consente di archiviare il modello in modo tale da poterlo recuperare e utilizzare in futuro senza la necessità di un ulteriore ciclo di addestramento. Tale approccio è strategico per ottimizzare l'efficienza e la scalabilità delle operazioni di machine learning, consentendo un rapido accesso al modello già addestrato per scopi applicativi successivi.
Questa procedura di registrazione dei risultati e di persistenza del modello garantisce un'adeguata tracciabilità delle attività di addestramento, facilita la condivisione e la riproduzione dei risultati, nonché consente un utilizzo efficiente delle risorse di calcolo e tempo.

CAPITOLO 4

Studio Empirico Esteso sul Codice Terraform

In questo capitolo viene presentato lo studio empirico condotto utilizzando l'applicativo *RADON Defuse*, precedentemente modificato come illustrato all'interno del capitolo 3 di questo documento. Lo scopo di questo studio è valutare l'influenza delle metriche di prodotto specificamente introdotte per Terraform nella predizione di script di Infrastructure as Code (IaC) che potrebbero essere suscettibili di fallimenti.

L'obiettivo principale di questo studio è condurre un'analisi approfondita del tool *RADON Defuse*, previamente modificato, all'interno del contesto della "Within-Project Defect Prediction." L'obiettivo fondamentale di tale analisi è valutare l'efficacia di questo strumento nella rilevazione di script Terraform potenzialmente difettosi e, contemporaneamente, identificare le caratteristiche distintive che possono essere associate a tali script. Il *punto di vista* è quello dei ricercatori che desiderano condurre un'analisi sperimentale per valutare l'efficacia della predizione dei difetti applicata a Terraform e che potrebbero essere interessati a repliche o ulteriori approfondimenti sull'argomento.

4.1 Le repository prese in esame

Per il presente studio, è stata presa la decisione di focalizzarsi su Terraform, in considerazione della sua crescente adozione nell'ambito aziendale e della mancanza di ricerche pregresse che avessero esaminato la predizione dei difetti specificamente nel contesto di Terraform.

Attraverso l'impiego della nuova versione del "Repositories Collector," sono state estratte tutte le repository pubbliche create a partire dal 1° Giugno 2016. Questa data è stata selezionata in quanto coincide con l'introduzione del supporto per Microsoft Azure all'interno del framework Terraform. Si evidenzia che Terraform ha fatto la sua prima apparizione nel Luglio del 2014, e nel 2015 è stato esteso con il supporto per Amazon Web Services (AWS). Pertanto, il 1° Giugno 2016 è stato considerato come un indicatore significativo della maturità del linguaggio Terraform nel contesto delle infrastrutture cloud.
Questa scelta metodologica mira a garantire che le repository prese in considerazione rappresentino un contesto stabile e maturo per l'analisi dei difetti, tenendo conto delle evoluzioni significative nell'ecosistema di Terraform.
Inoltre, al fine di garantire la rilevanza e la recente attività delle repository considerate, è stato stabilito come criterio essenziale che ciascuna di esse dovesse avere registrato almeno un "push" effettuato nel "main branch" nei sei mesi precedenti all'inclusione nell'analisi. Tale restrizione temporale è stata introdotta per assicurare che le repository considerate fossero attivamente mantenute e utilizzate nel periodo più recente, aumentando così la pertinenza dei dati raccolti per il nostro studio.

Complessivamente, il numero di repository raccolte mediante l'utilizzo del "Repository Collector," il quale è stato ampiamente delineato nel capitolo 2 e le cui modifiche sono state dettagliate nella sezione 3.1, è risultato pari a 1.140. Tuttavia, è importante sottolineare che, poiché si trattava di repository pubbliche, vi era la possibilità di includere progetti privi di contenuto o progetti personali sviluppati da individui meno esperti nell'uso di Terraform, con conseguente ridotta affidabilità qualitativa.

Pertanto, è stato necessario applicare ulteriori filtri servendosi del *Repository Scorer* alle repository raccolte al fine di selezionare esclusivamente quelle che si riferivano a progetti caratterizzati da un grado di ingegnerizzazione più elevato e da una maggiore affidabilità qualitativa. Tutte le repository che non soddisfacevano i requisiti delineati nella Tabella 4.1 sono state escluse dalla selezione. I criteri di selezione delle repository, come delineati nella Tabella 4.1, sono stati adottati dallo studio condotto da Dalla Palma [2]. Tali criteri sono stati oggetto di un'attenta analisi e sono stati ulteriormente raffinati per il presente lavoro di ricerca, il quale ha coinvolto un numero significativamente maggiore di progetti, consentendo l'applicazione di filtri più rigorosi.

Tabella 4.1: Filtri applicati per la selezione delle repository servendosi del *Repository Scorer*.

Nome	Descrizione del criterio
Release	La repository deve avere almeno 2 release
Percentuale di IaC	La repository deve avere almeno il 10% di codice Terraform
Numero di contributori	La repository deve avere almeno 2 contributori
Continuous Integration	La repository deve avere dei servizi di continuous integration
Percentuale di commenti	La repository deve avere una percentuale di commenti di almeno lo 0.5%
Linee di codice	La repository deve avere almeno 200 linee di codice di IaC
Licenza	La repository deve avere una licenza
Frequenza dei commit	La repository deve avere una media del numero di commit al mese almeno pari a 2.0.

Ciascun criterio di selezione è stato definito sulla base di ragionamenti specifici. Ad esempio, il requisito che richiede che il numero di "Release" sia maggiore di due è fondamentale poiché il processo di addestramento del modello richiede almeno due release di riferimento, come verrà dettagliato più approfonditamente in seguito

all'interno del presente capitolo.
La "Percentuale di IaC" è un criterio significativo, in quanto la presenza di una quantità considerevole di codice di Infrastructure as Code (IaC) è essenziale per garantire la validità della repository nell'ambito dello studio condotto.
Altri criteri, come il "Numero di Contributori," la presenza di "Continuous Integration," la "Percentuale di Commenti," e la disponibilità di una "Licenza" sono tutti indicatori di un elevato grado di ingegnerizzazione della repository. La presenza di una licenza testimonia anche la serietà del progetto e la sua natura non ludica.
Infine, "Linee di Codice" è un criterio di selezione importante, poiché si mira ad analizzare repository di dimensioni adeguate per condurre un'analisi accurata. Questi criteri di selezione sono stati attentamente ponderati e applicati per garantire la validità e l'affidabilità dei dati raccolti nel contesto di questo studio di ricerca.

Dopo l'applicazione di tutti i filtri, il numero rimanente di repository è **203**, mentre sono state scartate 937 repository. La Tabella 4.2 illustra il numero di repository scartate da ciascun criterio selezionato. È importante notare che la somma delle repository scartate da ciascun criterio è maggiore del totale delle repository scartate, poiché una repository può essere scartata da più criteri.

Tabella 4.2: Numero e percentuale di progetti scartati per ognuno dei criteri

Contributori	Percentuale commit	Percentuale commenti	Freq. commit	Linee Codice	Licenza	CI
83 (9%)	214 (23%)	35 (4%)	581 (63%)	18 (2%)	370 (40%)	180 (20%)

Come evidenziato nella Tabella 4.2, più della metà delle repository è stata scartata in quanto non soddisfaceva il criterio di avere almeno 2 commit al mese in media. Questo criterio è stato scelto con attenzione per garantire che le repository selezionate avessero un livello adeguato di attività e sviluppo nel tempo.
L'esclusione di queste repository non è stata una decisione arbitraria, ma è stata basata su considerazioni empiriche e teoriche. Un numero insufficiente di commit al

mese può indicare una scarsa attività o interesse nel progetto, rendendo le repository meno rappresentative del contesto di sviluppo reale. Inoltre, la presenza di un numero adeguato di commit è fondamentale per la corretta applicazione dei modelli di predizione dei difetti, in quanto tali modelli richiedono dati temporali significativi per effettuare previsioni accurate.

Sulle rimanenti 203 repository selezionate, è stato applicato il Repository Miner con l'obiettivo di estrarre i cosiddetti 'fixing commits' da ciascuna di esse. Una volta completata l'estrazione, è stata condotta un'attenta revisione di tali fixing commits al fine di identificare e rimuovere manualmente quelli che erano stati erroneamente identificati come tali ma che in realtà non rappresentavano correzioni di difetti nel codice.
Questa fase di revisione è stata necessaria per evitare falsi positivi nel nostro dataset e garantire che solo i fixing commits autentici fossero inclusi nell'analisi successiva. L'accuratezza di questa fase di revisione è stata fondamentale per garantire l'affidabilità dei dati utilizzati nella nostra ricerca. L'eliminazione dei falsi positivi ha contribuito a fornire un dataset più accurato e rappresentativo delle correzioni effettive apportate alle repository, consentendo così analisi più precise e informative nei capitoli successivi del nostro studio.

Al termine della fase di revisione manuale, 23 repository sono state escluse a causa dell'assenza completa di fixing commits, mentre altre 114 sono state escluse in quanto presentavano un numero di fixing commit inferiore a 8. Di conseguenza, le 66 repository rimanenti sono state selezionate come oggetto dello studio.
Questa decisione è stata presa poiché, con un tale numero di fixing commit insufficiente, sarebbe risultato impraticabile sviluppare un modello all'interno del progetto. In queste circostanze, persino l'implementazione della tecnica di bilanciamento dati più avanzata avrebbe incontrato difficoltà nel generare istanze artificiali in grado di rappresentare adeguatamente la classe di minoranza.

Tabella 4.3: Statistiche sul numero di fixing commits per le 66 repository rimaste.

Media	Mediana	Deviazione Standard	Min	Massimo
31.67	16.00	34.03	8.00	143.00

La tabella 4.3 presenta un'analisi delle statistiche relative al numero di "fixing commits" nell'insieme delle 66 repository rimanenti. Questi dati sono presentati per offrire una visione generale delle tendenze nei fixing commits all'interno di questo insieme di repository.
La media del numero di fixing commits è di 31.67, con un valore mediano di 16.00. Questi valori suggeriscono che le repository presentano un numero considerevole di fixing commits, con una distribuzione che si concentra attorno alla mediana.
La deviazione standard di 34.03 indica una notevole variabilità nei dati, con alcune repository che registrano un numero significativamente più alto di fixing commits rispetto alla media.
Il valore minimo di fixing commits è 8.00, mentre il massimo è 143.00, evidenziando la gamma di variazione all'interno di questo gruppo di repository.

In conclusione, la presente indagine è stata condotta esclusivamente sul campione costituito da 66 repository selezionate al termine del processo di selezione precedentemente descritto. Tutti i risultati e le analisi presentate in seguito si riferiranno esclusivamente a questo campione di repository selezionato.

4.2 Le domande di ricerca

L'obiettivo principale di questa ricerca consiste nell'effettuare una valutazione delle prestazioni del classificatore Random Forest nell'identificazione dei file suscettibili a difetti. Inoltre, uno degli aspetti di rilevanza prioritaria di questo studio è la determinazione delle metriche di prodotto presentate nello studio che dimostrano la maggiore efficacia nella predizione dei difetti all'interno degli script Terraform. Quin-

di le domande di ricerca che si propone di rispondere il nostro studio sono le seguenti:

RQ1. *Come si comporta il classificatore Random Forest nella predizione dei difetti all'interno degli script Terraform e quali sono le sue prestazioni in questo contesto?*

RQ2. *Tra le metriche di prodotto introdotte all'interno di questo studio, quali dimostrano la maggiore efficacia nella predizione dei difetti all'interno degli script Terraform?*

Nel corso di questa ricerca, le domande di cui sopra saranno affrontate mediante l'analisi dei risultati derivati dall'addestramento di 66 modelli distinti, ognuno dei quali è stato creato per una delle repository selezionate, oggetto dello studio. Questi modelli sono stati addestrati utilizzando il Defect Predictor modificato, come precedentemente descritto nel capitolo dedicato alle metodologie.
L'algoritmo Random Forest utilizzato nella componente applicativa del Defect Predictor è noto per la sua elevata precisione nella classificazione e per la sua capacità di gestire dati affetti da valori mancanti o contaminati da rumore. Inoltre, uno studio precedente condotto da Dalla Palma et al. [2] ha evidenziato che il Random Forest ha ottenuto le migliori performance nella predizione dei difetti nel contesto di Ansible. Pertanto, in questa ricerca abbiamo deciso di concentrarci esclusivamente sull'analisi delle prestazioni del Random Forest nel contesto di Terraform e di verificarne la validità e l'efficacia, senza eseguire un confronto diretto con altri algoritmi come Support Vector Machine.
L'obiettivo principale è quindi quello di valutare se il Random Forest conferma le sue prestazioni di successo nel contesto di Terraform, come precedentemente osservato in un contesto simile con Ansible.

Per rispondere alla prima domanda è necessario stabilire un criterio per valutare le performance dei modelli addestrati. Per farlo è stato deciso di utilizzare le seguenti due misure:

1. **Area Under the Curve - Precision and Recall (AUC-PR)**, tale misura è suggerita, all'interno del lavoro di Ali Khan e Ali Rana et al. [10], come misura più

appropriata come parametro di valutazione per i modelli di predizione dei difetti, dato che i dati sui difetti sono sbilanciati. Inoltre, questa misura è già stata impiegata nell'ambito dello studio condotto da Della Palma, di conseguenza la sua adozione agevola la possibilità di effettuare un confronto diretto tra i risultati ottenuti all'interno del contesto di Ansible.

2. **Matthews Correlation Coefficient (MCC)**, il paper presentato da Jingxiu Yao et al. [11] suggerisce di considerare l'uso del coefficiente di correlazione di Matthews (MCC) anziché l'F1 come metrica di prestazione in contesti di predizione dei difetti del software. Questo perchè l'F1 può essere influenzato in modo significativo dalla distribuzione delle classi e potrebbe non essere la metrica più affidabile quando si hanno dati sbilanciati. Inoltre tale metrica è stata già utilizzata all'interno del già citato lavoro di Dalla Palma.

Per poter rispondere alle domande di ricerca sono stati creati 66 modelli. Durante il processo di addestramento dei modelli, è stata impiegata la tecnica di selezione delle feature denominata Recursive Feature Elimination with Cross-Validation (RFECV). Questa tecnica ha consentito di identificare un sottoinsieme ottimale di feature, riducendo la complessità dei modelli e mantenendo solo le feature più informative.

La validazione dei modelli è stata eseguita utilizzando una metodologia di *"**walk-forward**"*, che ha suddiviso i dati in parti ordinate in modo da evitare l'uso di dati passati nel test-set rispetto a quelli contenuti nel train-set. Nel contesto specifico di questo studio, è stata implementata una variante nota come *"**walk-forward release**"*, in cui le parti ordinate corrispondevano alle diverse release del software. In ciascuna iterazione di questa metodologia, i dati relativi alla release corrente venivano assegnati al test set, mentre tutti i dati appartenenti alle release precedenti costituivano il train set. La Figura 4.1 mostra l'*n-esima* iterazione del processo sopracitato, è possibile notare che tutti i dati della release numero n ed i precenti vengono utilizzati come dati di train mentre i dati dell'*(n+1)-esima* release sono utilizzati come dati di test.

Figura 4.1: Esempio di iterazione numero n all'interno del processo di walk forward release cross validation.

Inoltre, per trovare la migliore configurazione dei parametri del classificatore Random Forest, è stata eseguita una ricerca casuale dei parametri utilizzando RandomizedSearchCV di scikit-learn. Questa procedura ha permesso di esplorare diverse combinazioni di parametri in modo casuale, al fine di individuare la configurazione ottimale per ciascun modello.

I risultati di questa fase includono il miglior estimatore, un rapporto dettagliato delle prestazioni e l'elenco delle feature selezionate. È importante sottolineare che i modelli ottenuti in questa fase sono stati addestrati utilizzando il classificatore Random Forest. Questa configurazione di addestramento ha costituito la base per l'analisi e la valutazione dei 66 modelli di predizione dei difetti del software, fornendo una solida fondazione per le conclusioni e le raccomandazioni tratte dalla ricerca.

4.3 Metodologia per raccolta e analisi dei dati

All'interno di questa sezione, verrà dettagliatamente illustrata la metodologia utilizzata per raccogliere e analizzare i dati al fine di fornire risposte esaustive alle due domande di ricerca.

4.3.1 Metodologia RQ1

Dopo aver completato l'addestramento dei 66 modelli RADON Defuse genera un rapporto completo delle prestazioni di ciascun modello. Questi dati sono

successivamente archiviati in modo persistente su un'istanza di database Firestore. Questo consente un'analisi dettagliata delle prestazioni in un momento successivo all'addestramento, garantendo la conservazione e l'accessibilità dei dati relativi alle prestazioni dei modelli.
In seguito, sono stati raccolti in modo sistematico tutti i dati necessari per rispondere alle domande di ricerca. In particolare, per ciascuno dei 66 modelli, sono state estratte e registrate le seguenti metriche chiave: l'Area Under the Curve - Precision and Recall (AUC-PR), il Matthews Correlation Coefficient (MCC), la Precision e la Recall. Queste metriche sono state accuratamente organizzate e strutturate in un file CSV dedicato. Successivamente, è stata condotta l'analisi statistica dei dati, al fine di ottenere una comprensione dettagliata delle prestazioni dei modelli.

4.3.2 Metodologia RQ2

Il rapporto ottenuto al termine della fase di addestramento documenta anche l'intero processo di addestramento.
È importante sottolineare che nell'ambito di tale processo è stato utilizzato l'algoritmo di selezione delle feature RFECV, il quale ha identificato le feature più rilevanti per ciascun modello di machine learning.
Tale processo ha comportato la selezione automatica delle feauture più importanti, eliminando quelle meno rilevanti attraverso un processo di eliminazione iterativo basato sulla cross-validation.
Il rapporto risultante è stato archiviato in modo persistente su un'istanza del database Firestore e contiene un elenco dettagliato delle feature identificate come le più rilevanti per ciascun modello addestrato.
In particolare, è stato registrato il numero di volte in cui ciascuna metrica è stata identificata come rilevante in ognuno dei modelli analizzati.

CAPITOLO 5

Risultati ottenuti

Il capitolo corrente presenta i risultati derivanti dall'approfondito studio empirico delineato nel capitolo precedente. In particolare, questo capitolo analizza i risultati derivati dall'addestramento di 66 modelli distinti, uno per ciascuna delle repository oggetto dell'indagine. L'analisi condotta è finalizzata a rispondere in maniera esaustiva alle specifiche domande di ricerca precedentemente delineate all'interno della sezione 4.2.

5.1 RQ1 - Prestazioni del Random Forest

All'interno della sezione seguente, vengono esaminate le prestazioni del classificatore Random Forest nell'ambito della predizione dei difetti nel contesto specifico di Terraform. L'obiettivo principale di questa analisi è fornire una risposta esauriente alla seguente domanda di ricerca:

RQ1. *Come si comporta il classificatore Random Forest nella predizione dei difetti all'interno degli script Terraform e quali sono le sue prestazioni in questo contesto?*

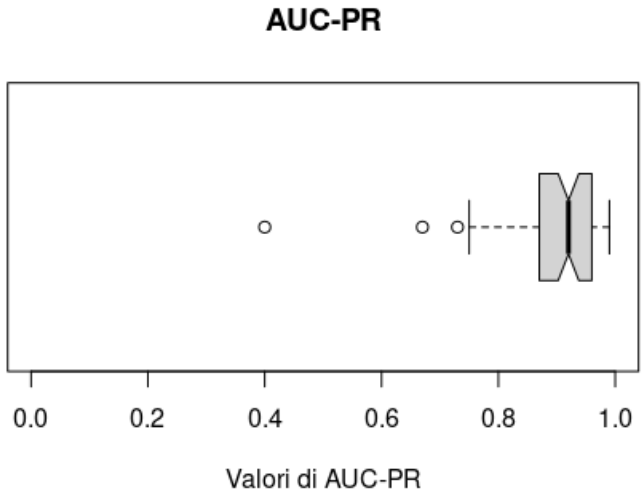

Figura 5.1: Area Under the Curve - Precision and Recall (AUC-PR) dei modelli addestrati.

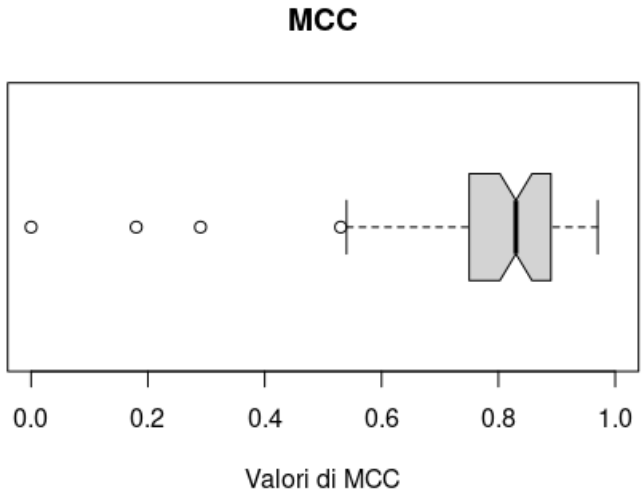

Figura 5.2: Matthews Correlation Coefficient dei modelli addestrati.

5.1.1 Risultati

In Figura 5.1, vengono presentate le prestazioni dei modelli misurate in termini di Area Under the Curve - Precision and Recall (AUC-PR). L'analisi del boxplot evidenzia che il 50% centrale dei dati si colloca in un intervallo che va da 0.87 a 0.96. Tale intervallo rappresenta la fascia di prestazioni tipica della maggior parte dei modelli inclusi nello studio.

L'AUC-PR è una misura di prestazione comunemente utilizzata per valutare la capacità di un modello di classificazione nel riconoscere correttamente i casi positivi in un set di dati in cui la classe positiva è in minoranza. Nel contesto dell'analisi condotta, è importante notare che i valori dell'AUC-PR si avvicinano al valore massimo possibile di 1. Questo valore massimo indica una capacità di separare in modo perfetto i casi positivi dai falsi positivi. Di conseguenza, è possibile concludere che i modelli addestrati dimostrano una notevole capacità discriminante e sono in grado di effettuare una distinzione efficace tra le istanze positive e i falsi positivi.
Nel box plot rappresentato in Figura 5.2, vengono presentate le prestazioni dei modelli misurate in termini di Matthews Correlation Coefficient (MCC). L'analisi del boxplot evidenzia che il 50% centrale dei dati si colloca in un intervallo compreso tra 0.75 e 0.89. Questo intervallo rappresenta la fascia di prestazioni tipica della maggior parte dei modelli inclusi nello studio.
Il Matthews Correlation Coefficient (MCC) è una misura di accuratezza di un modello di classificazione, con valori compresi tra -1 e +1. Valori positivi di MCC indicano una buona capacità del modello di separare le classi, mentre valori negativi indicano un peggioramento rispetto a una predizione casuale. Osservando il box plot in Figura, si può concludere che i modelli analizzati sono altamente efficaci nel distinguere tra casi positivi e negativi nel problema di classificazione. È degno di nota sottolineare il fatto che le prestazioni del modello Random Forest all'interno di questo studio risultino essere confrontabili e simili a quanto riscontrato nello studio condotto da Dalla Palma et al. [2]
In tale studio, il Random Forest aveva dimostrato di essere il metodo di apprendimento con le prestazioni superiori rispetto ad altri metodi nella predizione dei difetti nel contesto di Ansible. Di conseguenza, è possibile concludere che il Random Forest ha confermato la sua efficacia anche nel contesto specifico di Terraform.

5.2 RQ2 - Migliori metriche per la predizione

All'interno della sezione seguente, vengono esaminati invece i risultati della feauture selection tramite RFECV nel contesto della defect prediction in Terraform. L'obiettivo principale di questa analisi è fornire una risposta esauriente alla seguente domanda di ricerca:

RQ2. *Tra le metriche di prodotto introdotte all'interno di questo studio, quali dimostrano la maggiore efficacia nella predizione dei difetti all'interno degli script Terraform?*

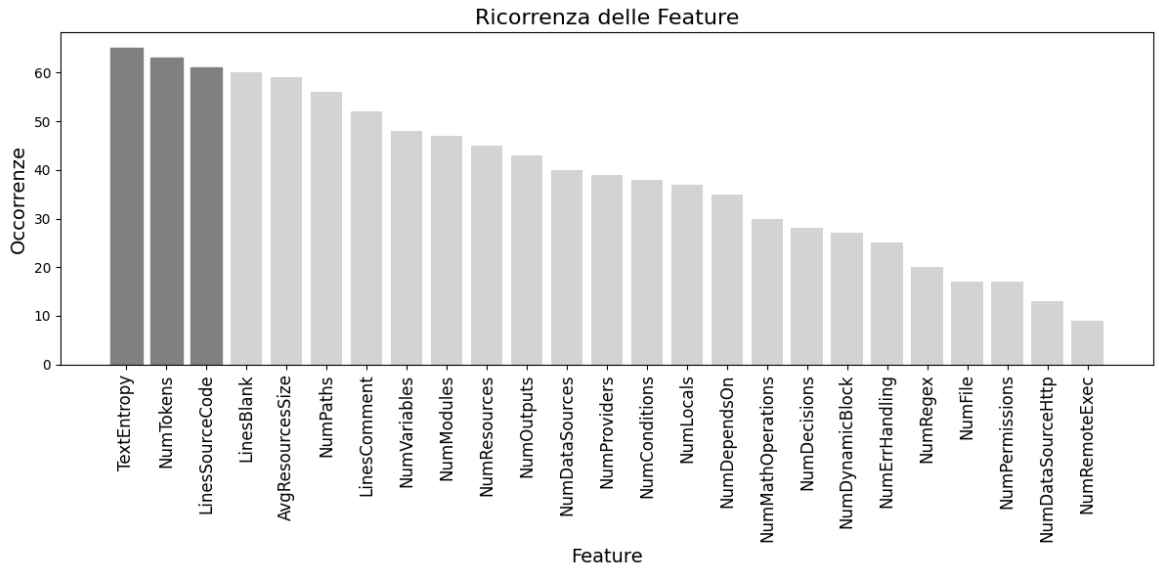

Figura 5.3: Numero di occorrenze per ciascuna metrica.

5.2.1 Risultati

La Figura 5.3 illustra il numero di volte in cui ciascuna metrica è stata selezionata come rilevante dal processo RFECV. È degno di nota evidenziare che tutte le metriche sono state selezionate almeno una volta.
Tuttavia, è fondamentale sottolineare che alcune metriche sono state selezionate molto più frequentemente rispetto ad altre. In particolare, le metriche più frequentemente selezionate sono, in ordine di importanza, *TextEntropy*, *NUmTokens*, e *LinesSourceCode*, che appaiono come rilevanti in un numero significativamente maggiore di casi rispetto alle altre metriche. Al contrario, la metrica *NumRemoteExec* è stata selezionata il

minor numero di volte, con un totale di soli 9 casi. È opportuno effettuare un'analisi comparativa tra i risultati di questo studio e quelli ottenuti da Dalla Palma et al. [2]. Nello studio condotto da Dalla Palma, le metriche più rilevanti sono state identificate come NumTokens, LinesCode (corrispondente a LinesSourceCode nel contesto di questo studio) e TextEntropy, tutte occupando il primo posto con lo stesso numero di occorrenze. È interessante notare che, anche nel presente studio, queste tre metriche sono emerse come le più rilevanti, seppur non condivendo il primo posto in ex aequo. Infatti, come illustrato in Figura 5.3 la metrica *TextEntropy* è risultata essere la più efficace essendo stata selezionata 65 volte seguita da *NumTokens* con 63 selezioni e *LinesSourceCode* con 61 selezioni.

Questi risultati evidenziano la robustezza e la coerenza di entrambi gli studi, sottolineando l'importanza di tali metriche per la predizione dei difetti sia nel contesto di Ansible che nel contesto di Terraform.

Di conseguenza, riguardo alla domanda di ricerca affrontata in questa sezione, è emerso che tra le metriche di prodotto introdotte all'interno di questo studio, quelle che dimostrano la maggiore efficacia nella predizione dei difetti all'interno degli script analizzati sono *TextEntropy*, *NumTokens* e *LinesSourceCode*. Una spiegazione dettagliata di queste metriche è fornita nella sezione 3.3 del presente documento.

CAPITOLO 6

Minacce alla validità

Questo capitolo espone le minacce alla validità dello studio e descrive le strategie adottate per mitigarle. Poiché il presente studio rappresenta un'estensione dello studio condotto da Dalla Palma et al. [2], alcune delle minacce all'validità identificate in tale studio vengono ereditate e discusse in questo capitolo.

6.1 Minacce alla validità di costrutto

Le minacce alla validità di costrutto si concentrano sulla relazione tra la teoria sottostante e le osservazioni effettuate. Nel contesto del nostro studio, le seguenti minacce sono state identificate:

Script failure prone non identificati correttamente. Poiché il tool RADON Defuse è utilizzato per identificare i fixing commits effettua un'analisi dei messaggi associati ad ogni commit e siccome questa strategia di ricerca di informazioni sui difetti potrebbe essere influenzata da bias. Per mitigare questo potenziale bias nei dati relativi ai difetti, si è deciso di mantenere una dimensione del campione considerevole, basandosi sulle conclusioni dello studio condotto da Rahaman et al. [12]. In particolare, l'aumento delle dimensioni del campione è stato raccomandato come un metodo per

ridurre il rischio di bias nei dati. Nel nostro caso, abbiamo analizzato un notevole numero di fixing commits, che ammonta a circa 2500, il che costituisce una quantità significativa e contribuisce a minimizzare il possibile bias. Inoltre, per garantire una maggiore precisione nell'identificazione dei fixing commits, tutti i commit sono stati sottoposti a un'ispezione manuale. Questo processo aveva l'obiettivo di eliminare eventuali casi in cui i commit fossero stati erroneamente identificati come fixing commits quando in realtà non lo erano.
L'analisi dei messaggi dei commit ha comportato anche una riduzione significativa del numero di file inizialmente considerati non soggetti a difetti ma che, in realtà, presentavano delle problematiche.

Errore nella computazione delle metriche introdotte. Un errore nel processo di estrazione delle metriche dagli script potrebbe avere un impatto significativo sui risultati relativi all'importanza delle metriche stesse.
Per garantire l'integrità di questo processo di estrazione dei valori delle metriche, è stato adottato un approccio automatizzato mediante l'utilizzo del tool Terraform-Metrics. È fondamentale notare che è stato condotto un rigoroso processo di test su TerraformMetrics, garantendo così che non vi fossero errori nel calcolo delle metriche.

6.2 Minacce alla validità interna

Le minacce alla validità interna si concentrano sulla relazione tra trattamento e risultato, sono dovute a confounding factors. Nel contesto del nostro studio, le seguenti minacce sono state identificate:

Bassa capacità predittiva delle metrice introdotte. È possibile che le metriche introdotte non dispongano intrinsecamente di una capacità predittiva significativa, ma piuttosto che i risultati positivi ottenuti siano il risultato di variabili che, in qualche modo, presentano correlazioni con tali metriche. È importante notare che molte delle metriche introdotte sono state adattate da metriche già esistenti utilizzate in altri linguaggi sia di programmazione che infrastrutturali, le quali sono state oggetto di studi pregressi. Per quanto riguarda le metriche specificamente progettate

per Terraform e non adattate da altri contesti, sono state sviluppate cercando di conformarsi alle best practice riconosciute. Questo approccio è stato adottato per mitigare la minaccia di bassa capacità predittiva delle metriche.

6.3 Minacce alla validità esterna

Le minacce alla validità esterna sono concerni alla generalizzazione dei risultati, all'interno del nostro studio possono derivare dall'aver scelto un campione di progetti non rappresentativo dell'intera popolazione dei progetti Terraform.
Effettivamente, le principali problematiche dal punto di vista della validità esterna includono quanto segue:

Selezione di sole repository pubbliche. L'utilizzo di repository pubbliche come fonte primaria di dati storici sui commit presenta innegabili vantaggi, quali l'ampia accessibilità dei dati e l'evitare ostacoli legati a questioni di riservatezza o autorizzazioni speciali. Inoltre, la scelta di repository pubbliche agevola la replicabilità degli esperimenti, promuovendo la trasparenza e la verifica dei risultati ottenuti. Tuttavia, è importante riconoscere che questa approccio può comportare diversi problemi. Uno di questi problemi è la difficoltà nel generalizzare i risultati ottenuti. Le repository pubbliche possono presentare diversità e potrebbero non rappresentare al meglio l'intera popolazione dei progetti software Terraform. Alcune di esse potrebbero non essere adeguatamente ingegnerizzate o potrebbero mancare di collaborazione tra gli sviluppatori. Per mitigare questa minaccia all'interno di questo studio, è stata condotta l'estrazione di un elevato numero di repository pubbliche, e sono stati applicati criteri rigorosi per la selezione di quelle da includere nello studio. Questi criteri si sono concentrati sulla scelta di repository ben strutturate e ingegnerizzate, caratterizzate da una chiara evidenza di collaborazione tra gli sviluppatori. In questo modo, si è cercato di affrontare la sfida della rappresentatività dei dati e della generalizzazione dei risultati all'interno del contesto specifico di Terraform.

6.4 Minacce alla validità delle conclusioni

La validità delle conclusioni è determinata da questioni che incidono sulla capacità di tracciare una conclusione corretta. Nel contesto del nostro studio, le seguenti minacce sono state identificate:

Affidabilità delle misure. Misure inaffidabili o soggettive potrebbero minare la validità delle conclusioni. All'interno del presente studio per valutare le prestazioni di un modello sono state utilizzate due misure che sono l'AUC-PR e il MCC. L'AUC-PR è suggerita, all'interno del lavoro di Ali Khan e Ali Rana et al. [10], come misura più appropriata come parametro di valutazione per i modelli di predizione dei difetti, dato che i dati sui difetti sono sbilanciati. Cosa simile vale per MCC che è oggetto di numerosi studio come ad esempio quello di J. Yao et al. [11] che suggerisce di preferirla all'F1 nei contesti di defect prediction. Inoltre tali metriche sono comunemente usate negli studi riguardo la predizione dei difetti [13] [14]. Queste misure sono oggettive e consentono un confronto accurato delle prestazioni dei modelli.
Per valutare l'importanza delle feature, è stato adottato un approccio basato sul conteggio delle volte in cui ciascuna caratteristica è stata selezionata durante il processo di selezione delle feature condotto mediante RFECV. Questo metodo fa uso dell'apprendimento automatico per identificare le feature più rilevanti all'interno di un modello. Il suo funzionamento prevede l'eliminazione iterativa delle feature meno influenti fino a ottenere un sottoinsieme ottimale, il quale migliora le prestazioni complessive del modello. Pertanto, tale metodo di confronto può essere considerato valido.

CAPITOLO 7

Conclusioni

Nel contesto del presente lavoro di tesi, sono state introdotte nuove metriche di prodotto per la predizione dei difetti nel linguaggio di programmazione Terraform, e il tool Radon Defuse è stato esteso per supportare la predizione dei difetti all'interno dei progetti Terraform. L'obiettivo principale era quello di utilizzare il tool esteso e le metriche di prodotto introdotte per condurre uno studio empirico.
Nel corso di questo studio empirico, sono stati selezionati con attenzione 66 progetti pubblici basati su Terraform. Per ciascun progetto, il tool esteso è stato utilizzato per addestrare modelli di classificazione Random Forest. L'obiettivo di questo sforzo era duplice: innanzitutto, valutare le prestazioni del classificatore Random Forest nel contesto specifico di Terraform; in secondo luogo, esaminare l'efficacia delle nuove metriche di prodotto nell'identificare potenziali difetti in Terraform.

I risultati ottenuti da questo studio indicano che il Random Forest ha confermato la sua efficacia anche nel contesto specifico di Terraform. In particolare, è stato osservato un punteggio medio di AUC-PR (Area Under the Precision-Recall Curve) di circa 0.90 e un punteggio medio di MCC (Matthews Correlation Coefficient) di circa 0.80, suggerendo un buon livello di prestazioni del modello nella predizione dei difetti nel codice Terraform. Inoltre, è interessante notare che le prestazioni del

Random Forest in questo studio sono risultate paragonabili allo studio precedente condotto nel contesto di Ansible da Dalla Palma et al. [2]. Questo suggerisce che il Random Forest rimane un modello efficace per la predizione dei difetti, indipendentemente dal linguaggio o dal contesto specifico.

Questo studio ha anche contribuito a identificare quali tra le nuove metriche di prodotto introdotte possono essere efficaci nella predizione dei difetti in Terraform. È stato osservato che tra le metriche di prodotto introdotte, alcune si sono dimostrate particolarmente efficaci nella predizione dei difetti nei codici sorgente analizzati. In particolare, le metriche che hanno dimostrato di possedere una maggiore capacità predittiva sono le seguenti:

- TextEntropy: si riferisce all'entropia del testo presente nel codice sorgente.
- NumTokens: conta il numero di parole separate da uno spazio bianco nel codice sorgente.
- LinesSourceCode: conta il numero di linee di codice eseguibile presenti nel file sorgente.

L'analisi delle prestazioni di queste metriche nel contesto dello studio ha rivelato che sono in grado di fornire informazioni significative per la predizione dei difetti nei codici sorgente esaminati.

Come prospettive di sviluppo future, si intende ampliare questa ricerca per includere anche altri linguaggi di Infrastructure as Code (IaC) diversi da Terraform e Ansible. L'obiettivo principale di questa estensione è consolidare ulteriormente i risultati ottenuti finora e cercare di generalizzarli sull'intero panorama dell'IaC. In partiolare potrebbe essere interessante esplorare i servizi specifici offerti da un fornitore di servizi cloud, come ad esempio AWS CloudFormation o i modelli Azure Resource Manager (ARM).

Inoltre, un'altra possibile direzione di ricerca consiste nell'eseguire uno studio simile a quello attuale, ma con un focus specifico sull'esplorazione di tecniche di apprendimento semi-supervisionato e non supervisionato. Tale approccio consentirebbe di

valutare le prestazioni e i risultati ottenuti da queste tecniche e confrontarli con quelli derivanti dall'apprendimento supervisionato.

Bibliografia

[1] S. Jourdan and P. Pomès, *Infrastructure as Code (IAC) Cookbook*. Packt Publishing Ltd, 2017. (Citato a pagina 1)

[2] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri, "Within-project defect prediction of infrastructure-as-code using product and process metrics," *IEEE Transactions on Software Engineering*, vol. 48, no. 6, pp. 2086–2104, 2021. (Citato alle pagine 2, 15, 32, 36, 40, 46, 48, 49 e 54)

[3] M. Eriksson and V. Hallberg, "Comparison between json and yaml for data serialization," *The School of Computer Science and Engineering Royal Institute of Technology*, pp. 1–25, 2011. (Citato a pagina 6)

[4] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *Iet Software*, vol. 12, no. 3, pp. 161–175, 2018. (Citato a pagina 10)

[5] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and software technology*, vol. 55, no. 8, pp. 1397–1418, 2013. (Citato a pagina 10)

[6] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri, "Toward a catalog of software quality metrics for infrastructure code," *Journal of Systems and Software*, vol. 170, p. 110726, 2020. (Citato alle pagine 11 e 21)

[7] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 164–175. (Citato a pagina 12)

[8] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating github for engineered software projects," *Empirical Software Engineering*, vol. 22, no. 6, pp. 3219–3253, 2017. (Citato a pagina 20)

[9] S. Dalla Palma, D. Di Nucci, and D. A. Tamburri, "Ansiblemetrics: A python library for measuring infrastructure-as-code blueprints in ansible," *SoftwareX*, vol. 12, p. 100633, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352711020303460 (Citato a pagina 20)

[10] S. A. Khan and Z. Ali Rana, "Evaluating performance of software defect prediction models using area under precision-recall curve (auc-pr)," in *2019 2nd International Conference on Advancements in Computational Sciences (ICACS)*, 2019, pp. 1–6. (Citato alle pagine 40 e 52)

[11] J. Yao and M. Shepperd, "Assessing software defection prediction performance: Why using the matthews correlation coefficient matters," in *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, 2020, pp. 120–129. (Citato alle pagine 41 e 52)

[12] F. Rahman, D. Posnett, I. Herraiz, and P. Devanbu, "Sample size vs. bias in defect prediction," in *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, 2013, pp. 147–157. (Citato a pagina 49)

[13] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 321–332. (Citato a pagina 52)

[14] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2012. (Citato a pagina 52)

Ringraziamenti

Questa tesi rappresenta la conclusione di un lungo percorso universitario, un viaggio che mi ha offerto l'opportunità di acquisire una vasta conoscenza e di crescere come persona. Un percorso che ha avuto inizio con un ragazzo determinato, alla ricerca di conferme delle proprie capacità durante i primi esami universitari.
Nel corso di questi cinque anni, non solo ho trovato le conferme che cercavo, ma ho anche affrontato periodi difficili, superandoli con costanza e dedizione che sono sempre solito avere nelle cose che faccio con passione.
Quello di oggi rappresenta per me un grande traguardo e, allo stesso tempo, un inizio ricco di speranze. Auguro a me stesso di iniziare a raccogliere i frutti del duro lavoro.

Desidero iniziare col ringraziare le persone che hanno direttamente contribuito a questo lavoro ovvero il professore Dario Di Nucci e la dottoressa Valeria Pontillo, per la disponibilità e la precisione fornitami durante l'intero processo di stesura di questo elaborato.

Un ringraziamento speciale va ai miei genitori, sia per il sostegno morale che per quello economico, che hanno reso possibile raggiungere questo traguardo che è un po' anche vostro. Vi ringrazio di cuore per avermi insegnato, tra tante altre cose, il valore e l'importanza dello studio e per aver compiuto numerosi sacrifici per aiutarmi

a raggiungere questo successo. Vi voglio bene.
Desidero ringraziare mio fratello, la persona che meglio di chiunque altro conosce tutto ciò che ho vissuto in questi cinque anni. Ti auguro un percorso universitario altrettanto gratificante e felice come il mio.
Un ringraziamento affettuoso alle mie nonne per l'amore infinito che mi hanno sempre donato e per tutte le volte in cui mi hanno fatto i complimenti quando ottenevo buoni voti.
Un ringraziamento ai miei zii, i miei cuginetti e il resto della famiglia, vi voglio bene.
Un pensiero speciale va a mio nonno Gerardo, la persona che avrebbe desiderato essere qui oggi più di chiunque altro. Sei sempre stata una persona rumorosa e irrequieta ed è in giorni come questo che la tua assenza fa ancora più rumore. Spero di averti reso orgolgioso.

Un ringraziamento speciale va ad Angela, il regalo più prezioso che questo percorso magistrale mi ha donato. Sei stata la mia spalla, il mio confessionale, senza di te non sarebbe stato lo stesso. Grazie per aver ascoltato e compreso tutti i miei sfoghi, grazie per aver gioito più di me ad ogni mio esame. Lo studio ci ha portato via del tempo insieme, quanti giorni senza vederci... ma sono sicuro che tutti questi sacrifici ci ripagheranno. Ti amo.

Ringrazio di cuore tutti i miei amici Luca, Antonello, Teresa, Raffaele, Paola, Daniela, Martina per tutti i momenti felici e spensierati passati in vostra compagnia, se sono arrivato fin qui è anche grazie alle esperienze vissute insieme a voi.

Infine, desidero esprimere la mia gratitudine a tutte le splendide persone che ho incontrato all'università e che sono poi diventate miei amici; siete davvero tanti. Grazie a voi, l'università non è stata solo un luogo per seguire le lezioni, ma anche un posto in cui ho condiviso un'infinità di momenti divertenti e di risate.

Grazie a tutti.

Questa tesi ha contribuito a piantare un albero in Kenya tramite il progetto Treedom.

`https://www.treedom.net/it/user/sesalab/event/sesa-random-forest`