UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Applicazione di Tecniche di Machine Learning per l'Identificazione di Vulnerabilità a Livello di Commit

RELATORE

Prof. **Fabio Palomba**

Università degli Studi di Salerno

CANDIDATO

Mohamed Amine Sarraj

Matricola: 0512109737

Anno Accademico 2020-2021

Sommario

La crescente informatizzazione della società e dei servizi, in particolare per i settori in cui le informazioni conservate, scambiate o create sono di importanza critica, ha reso la sicurezza del software una questione fondamentale per fornire protezione contro accessi indesiderati ed evitare danni irreparabili al sistema. Con l'avvento di tecniche basate sui dati, l'apprendimento automatico (Machine Learning) ha acquisito progressivamente maggiore interesse come metodo di garanzia per dei software affidabili. Nel presente lavoro si propone la realizzazione di un modello machine learning capace di predire la vulnerabilità a livello di commit e garantire la sicurezza del sistema informatico. I modelli di previsione delle vulnerabilità (VPM) sono un approccio, creato sulla base di una varietà di metriche, avente lo scopo di trovare le vulnerabilità. In primo luogo, vengono descritti gli approcci esistenti e le rispettive limitazioni; in seguito, si pone l'accento sulle metodologie che hanno assunto il ruolo di linee guida per l'intero processo di progettazione. In particolare, è stato operato un confronto sia tra le diverse classi metriche utilizzate sia tra classificatori per individuare quelli che meglio forniscono un'accurata previsione delle debolezze. Il progetto si è basato sulla realizzazione di una classificazione di debolezze a livello di commit: avendo dei database di diverse classi metriche ed usufruendo delle tecniche di features selection, per ogni classe sono state identificate le metriche fondamentali per la predizione della vulnerabilità. Inoltre, attraverso l'utilizzo di tecniche di balancing è stato possibile migliorare in modo significativo le prestazioni di dataset altamente sbilanciati, con più del 99Infine ho fatto adoperato vari tipi di classificatori, partendo da quelli di base per arrivare al techniques ensamble, voting classifiers e grid search.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Dalla nascita negli anni '50 ad oggi, l'informatica continua a fare progressi. Questi ultimi hanno un impatto diretto sulla nostra vita: infatti, grazie all'informatizzazione e alla digitalizzazione, l'informatica è diventata fondamentale sia in ambito professionale che privato.

Oggi l'informatica influisce su ogni settore, come ad esempio quello economico, sanitario, industriale; renderla sicura avrà un impatto positivo in tutti questi ambiti. Le vulnerabilità nei software sono una vera minaccia sia per le grandi società che per gli utenti privati. Attualmente, l'ampia varietà di vulnerabilità esistenti ha focalizzato l'attenzione sul miglioramento della sicurezza informatica per evitare che esse vengano usufruite per causare danni.

1.2 Motivazioni ed obiettivi

Durante la fase di programmazione di un software è possibile che si introducano delle vulnerabilità, come ad esempio un piccolo cambiamento nel codice, che possono provocare risultati indesiderati. Ad oggi sono vari i modi per verificare la presenza di vulnerabilità nel codice: il fuzzing, static code verifiers, code reviews, etc.

Dati i danni che tali vulnerabilità possono causare, il loro rilevamento il prima possibile è diventato di fondamentale importanza: a tale scopo, uno degli approcci recentemente

utilizzati si basa sul machine learning.

La necessità di una predizione istantanea della vulnerabilità ha portato alla combinazione del machine learning e della sicurezza informatica per soddisfare tale fine.

L'obiettivo principale è partire dalle famiglie di metriche e, servendoci del machine learning, rilevare le vulnerabilità ed identificarle a livello di commit utilizzando vari tipi di classificatori.

Questo tipo di approccio consente il rilevamento Just-In-Time delle vulnerabilità offrendo:

- -velocità nel raccoglimento e consegna immediata delle vulnerabilità accelerandone la correzione;
- -efficacia dei costi,in termini di tempo, risorse usate e abilità di individuare le debolezze prima del salvataggio del codice nella repository;
- -scalabilità, che rende possibile il rilevamento su piccola o grande scala.

Lo scopo ultimo del presente lavoro di tesi è confrontare diversi modelli di apprendimento e diverse famiglie di metriche per individuare quello che fornisce risultati migliori.

1.3 Metodologia e risultati

Il lavoro qui proposto parte dalla costruzione del dataset mediante il raccoglimento di commit che servono al meglio lo scopo di rilevare le vulnerabilità.

In secondo luogo, esso si è basato sulla visualizzazione dei dati e sull' organizzazione di una buona strategia; inoltre, risulta particolarmente rilevante, nel modello, scegliere le metriche che influiscono di più sulla sicurezza e che indicano le vulnerabilità. Pertanto, è stata fondamentale la verifica dell'importanza delle features per instradare quelle più importanti.

Da notare come il dataset, soffrendo di un notevole sbilanciamento, necessitasse di tecniche di balancing per la risoluzione del problema.

Infine, sono stati messi a confronto diversi classificatori, partendo da quelli di base e terminando con quelli più elaborati come gli ensemble classifiers.

1.4 Struttura della tesi

La tesi è strutturata come segue. Il secondo capitolo presenta i lavori precedenti ed alcune tecniche di rilevamento delle vulnerabilità. Il terzo capitolo si focalizza dettagliatamente sulle

tecniche usate. Il quarto capitolo mostra la validità del nostro approccio. Il quinto ed ultimo capitolo conclude la tesi offrendo un quadro generale su quanto appreso e sugli approcci futuri da considerare.

CAPITOLO 2

Stato dell'arte

In letteratura esistono molteplici tecniche di detection di vulnerabilità, categorizzabili in diversi gruppi (Dynamic analysis, Static analysis, etc). In questo capitolo saranno presentati i precedenti lavori, alcuni basati sulle tecniche sopra menzionate ed altri sul machine learning.

2.1 Approcci di Detection di Vulnerabilità

2.1.1 Dynamic Analysis

L'analisi dinamica del programma [1] è l'analisi del software che viene fatta eseguendo programmi su un processore reale o virtuale. Affinché l'analisi dinamica del programma sia efficace, il programma di destinazione deve essere eseguito con input di test sufficienti per coprire quasi tutti i possibili output.

Fuzzing

Il fuzzing[2] è una tecnica usata per individuare errori dentro un sistema. Si tratta di un security detection method che invia un input non valido in un' applicazione e, se ritorna un output imprevisto, è probabile che il software nasconda una vulnerabilità non ancora nota.

Esistono 3 tipologie di fuzzing:

- Black box Fuzzing, consistente nel cambiamento dei dati corretti di output senza avere conoscenza dell'applicazione sviluppata;

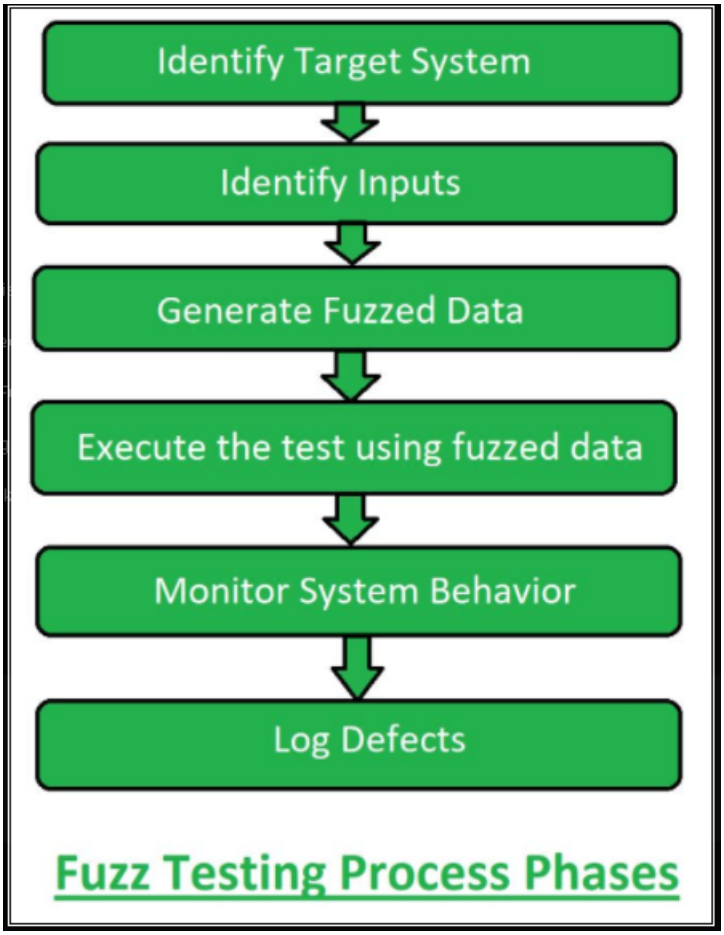

Figura 2.1: Le fasi del fuzzing.

- White box fuzzing, che prevede la generazione di un test avendo una totale conoscenza dell' applicazione e del suo funzionamento;
- Gray box fuzzing, una fusione tra la Whitebox e la Blackbox che si serve dei vantaggi di entrambe le tecniche.

Web Application Scanners

Per Web Application Scanners [3] si intendono delle applicazioni automatiche che analizzano un sistema sul web per identificarne le vulnerabilità. Una web application richiede un input all'Hyper- text Transfer Protocol request. La volatilità dell'input é la causa di tutte le vulnerabilità delle web applications .

In generale i web application scanners sono composti da quattro fasi:

Figura 2.2: mostra le quattro fasi che compongono il web application scanning.

1. Scan and crawl: raccogliere informazioni sull'applicazione Web in esecuzione, comprese le porte di rete, la versione del server Web, i moduli installati, i numeri di versione e la scansione di tutte le cartelle e i file che possono esistere sul sito Web.

2. Identify Vulnerabilities: basandoci sulle informazioni raccolte nella fase precedente, valuteremo l'esistenza di vulnerabilità Web o errori .

3. Result Analysis: rivedere le risposte dell'applicazione alle richieste Web effettuate nella fase precedente e verificare manualmente le vulnerabilità Web identificate ed eliminare i false positives.

4. Report: segnalare le vulnerabilità web identificate, inclusa la valutazione dell'impatto e le azioni consigliate per mitigarle.

Brick

Binary Run-time Integer Based Vulnerability Checker [4] ha il potere di rilevare le vulnerabilità in run time: é un approccio molto efficiente che dà risultati con una bassa percentuale di low positive e low negative.

Brick si compone di tre fasi: conversione del codice binario in una rappresentazione intermediata VEX su Valgrind ,intercettazione degli integer related statements at run-time e registrazione delle informazioni necessarie, rilevazione e localizzazione delle vulnerabilità.

2.1.2 Static Analysis

Si tratta di una tecnica difensiva e preventiva utile a rilevare le vulnerabilità nelle web applications. L'obbiettivo principale è quello di identificare le debolezze direttamente nel codice, prima del suo utilizzo effettivo nell'ambiente .

Gli approcci di Static Analysis [5] possono essere classificati in 6 famiglie: inference technique, analysis sensitivity, Analysis granularity, soundness, completeness e language .

SonarQube

SonarQube [6] è una piattaforma open source sviluppata da SonarSource per il controllo continuo della qualità del codice: l'esecuzione di revisioni automatiche con analisi statica del codice permette di rilevare bug e vulnerabilità di sicurezza in oltre 20 linguaggi di programmazione.

Find Security Bugs

Find Security Bugs [7] è un plug-in SpotBugs per i controlli di sicurezza delle applicazioni Web Java e delle applicazioni Android. Può rilevare 128 diversi tipi di vulnerabilità tra cui Command Injection, XPath Injection, SQL/HQL Injection, XXE and Cryptography weaknesses.

Find Security Bugs è uno strumento di analisi statica riferito principalemnte a Java ma utilizzato anche con i progetti Groovy, Scala e Kotlin.

2.1.3 Code Review

La Code Review [8] è il controllo sistematico del codice sorgente. Individua e corregge gli errori sfuggiti agli sviluppatori in fase di sviluppo e migliora la qualità complessiva del software oltre all'abilità dello sviluppatore stesso.

E' possibile analizzare un codice sia qualitativamente che quantitativamente.

2.2 Approcci ML per Detection di Vulnerabilità

La security inspection e testing richiede l'intervento di esperti che pensano come hackers o attackers. Sfruttando questo approccio per la ricerca delle vulnerabilità, è possibile sapere dove e cosa cercare nel sistema per rilevarle. I VPMs (Vulnerability Prediction Models), sviluppati a partire da un insieme di metriche, costituiscono un approccio molto promettente per il rilevamento delle vulnerabilità.

Dai precedenti lavori, nei quali è stata sfruttata l'intelligenza artificiale e nello specifico il machine learning, risulta che sono state utilizzate differenti famiglie di metriche per lo sviluppo dei VPMs.

L'elaborazione del VPM è formata da vari step. Partendo dalla costruzione del dataset e raccogliendo le metriche da utilizzare nel nostro modello, si passa al preprocessing nel quale prepariamo e puliamo il nostro database ed infine al modelling nel quale applicheremo su di esso i classifiers.

2.2.1 Software metrics - Broad

Questo VPM, descritto da T. Zimmermann [9], é basato sulle static code metrics e i pre- e post-releases vulnerabilities da Microsoft. È chiamato Broad e le metriche sono state raccolte da un sistema di microsoft chiamato CODEMINE.

CODEMINE fornisce 29 metriche attraverso la misurazione del churn del codice, la complessità del sistema e le features di organizzazione del team che ha sviluppato il sistema . Il classifier usato per questo VPM é il random forest.

Questo VPM dà come risultati un recall mediano di 0.2 e una precisione mediana di 0.45 .

2.2.2 Software metrics - Churn and complexity

L'approccio di Shin [10] nello sviluppo di un VPM nel rilevamento delle vulnerabilità è basato sul churn sulla complessità per potere individuare in che parte il codice presenta debolezze.

Shin ha usato il linear descriminant analisis , bayesian network modeling e random forrest per creare il VPM .

Il suo approccio ha permesso di ridurre la quantità del codice da ispezionare di 71% per mozilla e 28% per RedHat Enterprise Linux kernel.

2.2.3 Text mining

Un altro aproccio molto interessante è quello elaborato da Riccardo Scandariato, James Walden, Aram Hovsepyan e Wouter Joosen presentato nel saggio intitolato "Predicting Vulnerable Software Components via Text Mining". La tecnica, basata sul text mining [11] del codice sorgente dei componenti, ha permesso lo sviluppo di modelli di predizione capaci di rilevare quando una componente soffre di vulnerabilità. Tale approccio ha dimostrato di avere un buon rendimento sia per il recall sia per il precision.

CAPITOLO 3

Approccio ML per Detection Vulnerabilità a Livello di Commit

Il presente capitolo tratterà dell'approccio, basato sul machine learning, adottato per determinare l'esistenza delle vulnerabilità.

Il machine learning è un metodo grazie al quale un computer può imparare e adattarsi a dei dati senza l'intervento dell'uomo . É una branca dell'intelligenza artificiale che fornisce ai sistemi l'abilità di acquisire informazioni e migliorare la propria performance in modo autonomo.

Una delle cose più importanti da fare mentre si lavora su un progetto di machine learning è adottare una strategia vincente, sviluppata sulla base di una soluzione ben precisa al problema: in questo caso si tratta dell'efficienza del rilevamento delle vulnerabilità e ciò risulta possibile definendo uno scopo e un piano su come lavorare con passi ben precisi.

Tale procedimento si compone di due fasi essenziali e critiche, quali la costruzione del dataset e la costruzione della pipeline: il loro corretto sviluppo garantisce migliori risultati in modo più efficiente.

3.1 Costruzione Dataset

3.1.1 Variabili Dipendenti

Una delle particolarità del Machine Learning riguarda la potenzialità che un progetto ML ha di lavorare solo su un dataset sotto forma di tabella, nella quale si specificano quali variabili sono gli input e quali i target.

La scelta dei dati rappresenta un momento critico per il lavoro: scegliere le giuste metriche aumenta in modo significativo sia l'efficienza sia l' accuratezza del nostro algoritmo.

Per questo lavoro di tesi sono stati selezionati nove progetti JAVA le cui repository fossero pubblicamente accessibili su GITHUB, permettendone il mining. Insieme, i progetti sono composti da un totale di 56,286 commit (alla data dell'esperimento) ma, per via di limitazioni legate al calcolo, è stato fatto un campionamento casuale dei commit, per un totale di 9,091, corrispondente al 16% del totale. Sono stati scelti tali progetti contenenti vulnerabilità pubblicamente riportate nel *National Vulnerability Database* (NVD)—standard de-facto per la raccolta e la divulgazione di note vulnerabilità software descritte in termini di CVE (Common Vulnerabilities and Exposure) record. Nello specifico, abbiamo considerato tutti i CVE legati a progetti JAVA tali da avere un riferimento al *fixing commit*—il commit che ufficialmente contribuisce alla risoluzione della vulnerabilità applicandone una patch—nella repository GITHUB del progetto. La raccolta delle vulnerabilità rilevanti è avvenuta ispezionando il dump di NVD al tempo dello studio. La procedura di selezione è terminata con totale di 27 CVE appartenenti a 12 tipi diversi (Common Weakness Enumeration - CWE).

Successivamente, è stato eseguito l'algoritmo SZZ [12] per risalire, a partire dai fixing commit, ai *vulnerability-inducing commits*, ovvero commit che probabilmente sono responsabili per l'inserimento di una vulnerabilità. Nello specifico, per ogni file f_i toccato dal fixing commit c_fix, SZZ esegue il comando `git-diff` per estrarre la lista delle linee modificate in f_i rispetto al commit precedente c_{fix-1}; quindi, SZZ esegue il comando `git-blame` su tutte queste linee modificate al fine di risalire ai commit dove queste sono state modificare l'ultima volta. Tali commit saranno considerati vulnerability-inducing commit della vulnerabilità rimossa da c_fix. In sintesi, sono stati ottenuti 91 diversi vulnerability-inducing commit tra tutti i 9 progetti. I vulnerability-inducing commit, i CVE e i CWE saranno considerati le diverse *variabili dipendenti* (i.e., variabili target) che si vuole predirre tramite le diverse configurazioni di modelli di ML.

3.1.2 Variabili Indipendenti

Una volta raccolte le vulnerabilità, sono stati raccolti i dati sulle *variabili indipendenti* (i.e., feature) che saranno usate indipendentemente dalla variabile target predetta. Sono state considerate delle metriche software tali da (1) catturare aspetti rilevanti da un punto di vista della sicurezza, (2) descrivere caratteristiche legate al processo di sviluppo adottato, e (3) fossero facilmente calcolabili tramite semplice analisi statica del codice o della storia del progetto. Per queste ragioni sono state considerate due famiglie di metriche:

1. Metriche di *processo*, che riguardano aspetti legati ai commit del progetto, ad esempio la data del commit, numero di file modificati, code churn, ecc. Queste metriche sono calcolate usando uno script sviluppato ad hoc basto sulle API di PyDriller [13], una libreria per l'analisi di repository software. La descrizione di ciascuna metrica è riportata in Tabella 3.2.

2. Metriche di *sicurezza* per software Object-Oriented definite da Alshammari et al. [14]. Esse catturano aspetti legati alla facilità di accesso ai dati calcolabili a livello di classi e di progetto. Ad esempio, la metrica *Classified Attributes* (CA) misura il numero di attributi con nomi che indicano dati critici, e.g., `password` o `token`. Per il loro calcolo è stato usato un tool di nome Surface (SecURity FLAws metriCs Extractor). Le metriche definite a class-level sono state portate a livello di commit considerando tutti i file modificati dal commit ed aggregando tutti i valori ottenuti da ciascuna classe coinvolta nel commit. La descrizione di ciascuna metrica è riportata in Tabella 3.1.

3.2 Costruzione Pipeline ML

3.2.1 Data Exploration

La visualizzazione dei dati è definita come la rappresentazione grafica dei dati e delle informazioni allo scopo di comprenderli al meglio.

Il cervello umano ha infatti la capacità di elaborare meglio i dati quando sono rappresentati sotto forma di tabelle o grafici, poiché la loro visualizzazione sotto questo aspetto rende la trasmissione dei concetti più semplice e veloce.

Altra caratteristica fondamentale del data visualization riguarda la possibilità di identificare i valori i più importanti, ovvero i dati che maggiormente influiscono sui risultati, aiutandoci a capire i fattori che più influenzano il nostro modello .

Il dataset della famiglia di metriche raccolte da SURFACE è stato costruito in due tabelle diverse. La prima rappresenta le metriche e la seconda indica le variabili target .La prima tabella é composta da 15 colonne per 25114 linee, mentre la seconda ha solo 4 colonne per 116 linee. Le linee della seconda tabella rappresentano tutti i commit che hanno delle vulnerabilità.

L'obbiettivo è quello di combinare le due tabelle per formarne un' unica contenente i risultati di ogni commit: se un commit é presente nella seconda tabella allora esso presenta delle debolezze. Ciò si ottiene facendo un merge sulla colonna commitHash .

Tabella 3.1: Metriche di sicurreza calcolate dal tool SURFACE, e usate come variabili indipendenti dei modelli di ML.

Metric	Acronym	Description
CLASS-LEVEL		
Classified Attributes	CA	Number of *classified* attributes of a class, identified through pattern matching heuristics (e.g. `password`, `token`).
Classified Methods	CM	Number of *classified* methods of a class, identified through (i) pattern matching heuristics (e.g.`validatePassword`, `generateToken`) or (ii) the check of usages of classified attributes.
Classified Instance Variables Accessibility	CIVA	Ratio of non-private and non-static classified attributes out of the total number of classified attributes (CA).
Classified Class Variables Accessibility	CCVA	Ratio of non-private and static classified attributes out of the total number of classified attributes (CA).
Classified Method Accessibility	CMA	Ratio of non-private classified methods out of the total number of classified methods (CM).
Classified Methods Ratio	CMR	Ratio of the number of classified methods (CM) out of all class methods.
Classified Attribute Interactions	CAI	Sum of the number of classified methods that access each classified attribute, divided by the product of the number of classified attributes and methods (CA $\times$ CM).
PROJECT-LEVEL		
Critical Classes	CC	Number of *critical classes*, i.e., classes with at least one classified components (classified attribute or method).
Critical Classes Ratio	CCR	Ratio of the number of critical classes (CC) out of all project classes.
Critical Classes Extensibility	CCE	Ratio of non-final critical classes out of the critical classes (CC).
Classified Methods Extensibility	CME	Ratio of non-final critical methods among all classes out of the critical methods among all classes.
Critical Super Classes Ratio	CSCR	Mean of the ratios, for each class, of the number of critical super classes out of all their super classes.
Serializable Critical Classes Ratio	SCCR	Ratio of serializable critical classes out of the critical classes (CC).

Il dataset delle process metrics è rappresentato in una tabella composta da 9100 linee e 12 colonne; essa contiene sia la variabile target sia le metriche.

Il grafico 3.1 mostra in che forma sono rappresentati i nostri dati: in base al tipo si lavora in modo diverso. Possiamo vedere che la maggior parte dei nostri dati, più di ¾, sono di tipo float, gli altri invece sono di tipo object e integer.

Tutte le variabili del dataset process metrics sono di tipo integer, ad eccezione di mean_previous _changes che é di tipo float.

Uno dei problemi principali nei modelli di machine learning sono gli NA(Not Avaible) o variabili mancanti, i quali devono essere direttamente eliminati o rimpiazzati. Essi sono dovuti a varie ragioni come l' errore umano, errori dei sensori nella lettura dei dati o errori nel software .

Nel dataset delle metriche raccolte da SURFACE, tutte le variabili tranne cwe non soffrano di NA. Gli NA nel cwe non sono assolutamente un problema in questo caso, poichè nel

Tabella 3.2: Lista di metriche di processo calcolate dalle API di PyDriller, e usate come variabili indipendenti dei modelli di ML.

Metric	Description
Days After Creation	The distance in days between a commit date and the project's repository creation (i.e., the first commit) date.
Touched Files	The total number of files modified by the commit, excluding the irrelevant (test classes, documentation, build, and blob files).
Added Lines	The number of lines added in the commit among all touched files.
Deleted Lines	The number of lines removed in the commit among all touched files.
Mean of Previous Changes	The mean number of previous changes (i.e., commits) of each touched file.
Author Date	The ISO date in which the commit was made by the author.

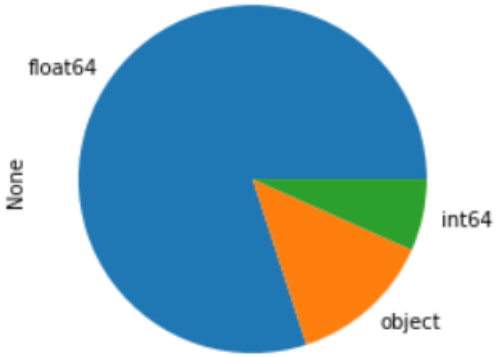

Figura 3.1: Mostra i tipi di dati della tabella di metriche raccolte da SURFACE.

processo di raccoglimento dei dati , il software inseriva NA quando il commit non rilevava vulnerabilità.

Per risolvere il problema da un punto di vista algoritmico sarà necessario sostituire gli NA nel cwe con 0 , dove 0 rappresenterà un programma sicuro senza vulnerabilità .

Il database delle process metrics non presenta nessun valore NA.

Nella variabile target risulta che vi sono 11 tipi di cwe: il più comune é NVD-CWE-noinfo, presente quando non ci sono informazioni sufficienti sul cwe per classificarlo, seguito poi da CWE-200, a causa del quale è possibile effettuare un accesso non esplicitamente autorizzato ad un'informazione sensibile; insieme essi rappresentano circa 1/2 dei cwe.

Nei valori della target si presenta il più grande problema del database: lo squilibrio nei risultati, dove più del 99% dei commit non hanno vulnerabilità. Ciò consentirà l'uso di tecniche di balancing per ottenere risultati migliori.

Il grafico 3.2 indica il range di valori di un attributo e la sua distribuzione tra la totalità

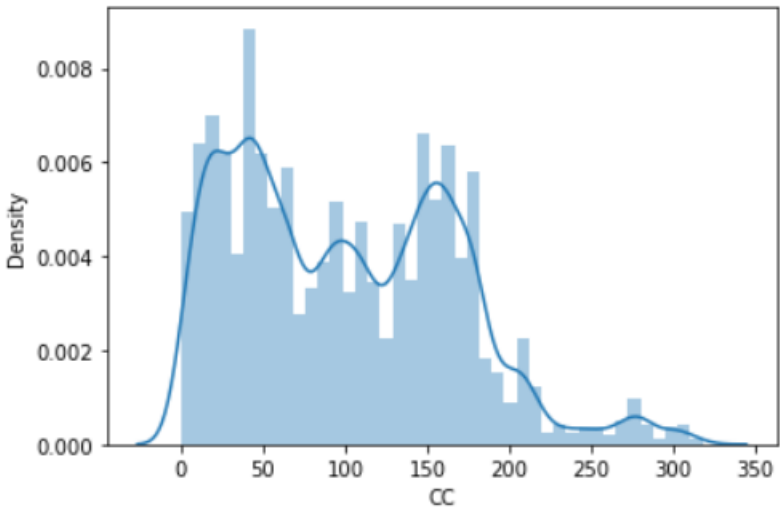

Figura 3.2: Mostra il range di valori della feature CC.

dei dati. Prendendolo ad esempio vediamo che i valori hanno un range da -20 fino a 350 con 2 picchi di notevole densità nel 50, con un valore superiore a 0.006, e un altro picco a 160 dove la densità é quasi a 0.006.

Possiamo anche notare che i valori sono quasi tutti tra 0 e 200.

Una simile distribuzione é visibile per gli altri.

3.2.2 Mutual Gain information

Nella teoria della probabilità e nella teoria dell'informazione, l'informazione mutua [15] mutua informazione (talvolta nota con il termine arcaico di transinformazione) di due variabili casuali è una quantità che misura la mutua dipendenza delle due variabili. La più comune unità di misura della mutua informazione è il bit, quando si usano i logaritmi in base 2.

Intuitivamente, l'informazione mutua misura l'informazione che X e Y condividono: essa misura quanto la conoscenza di una di queste variabili riduce la nostra incertezza riguardo all'altra. Ad esempio, se X e Y sono indipendenti, allora la conoscenza di X non dà alcuna informazione riguardo a Y e viceversa, perciò la loro mutua informazione è zero. All'altro estremo, se X e Y sono identiche allora tutte le informazioni trasmesse da X sono condivise con Y: la conoscenza di X determina il valore di Y e viceversa. Come risultato, nel caso di identità l'informazione mutua è la stessa contenuta in Y (o X) da sola, vale a dire l'entropia di Y (o X: chiaramente se X e Y sono identiche, hanno identica entropia).

Nel caso del machine learning, il mutual gain information ci permette di individuare le features più importanti e quelle che hanno l'impatto più significativo.

3.2.3 VIF

Il Variance Inflation Factor (VIF) [16] viene utilizzato per rilevare la presenza di multicollinearità. I fattori di inflazione della varianza (VIF) misurano quanto viene gonfiata la varianza dei coefficienti di regressione stimati rispetto a quando le variabili predittive non sono correlate linearmente.

Si ottiene regredendo ciascuna variabile indipendente, diciamo X sulle restanti variabili indipendenti (diciamo Y e Z) e verificando quanto di essa (di X) è spiegata da queste variabili.

$$VIF = \frac{1}{1 - R^2}$$

Attraverso il VIF, eliminiamo le variabili con il valore più alto, cioè le più correlate.

3.2.4 Modelling

Decision Tree

Nel machine learning un albero di decisione [17] è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino (path) dal nodo radice (root) al nodo foglia.

Linear regression

Nel machine learning la regressione lineare [18] è una tecnica di classificazione degli esempi di un dataset (insieme di training) per consentire alla macchina di apprendere automaticamente un modello decisionale.

Il risultato finale è una linea retta che minimizza la distanza tra gli N esempi dell'insieme di training che appartengono alla stessa categoria.

Una volta trovata, la funzione di classificazione può essere utilizzata per valutare istanze diverse dall'insieme di training. Se le coordinate (x,y) di un'istanza si avvicinano alla funzione di regressione f(x), l'istanza viene classificata con l'etichetta Z. E così via.

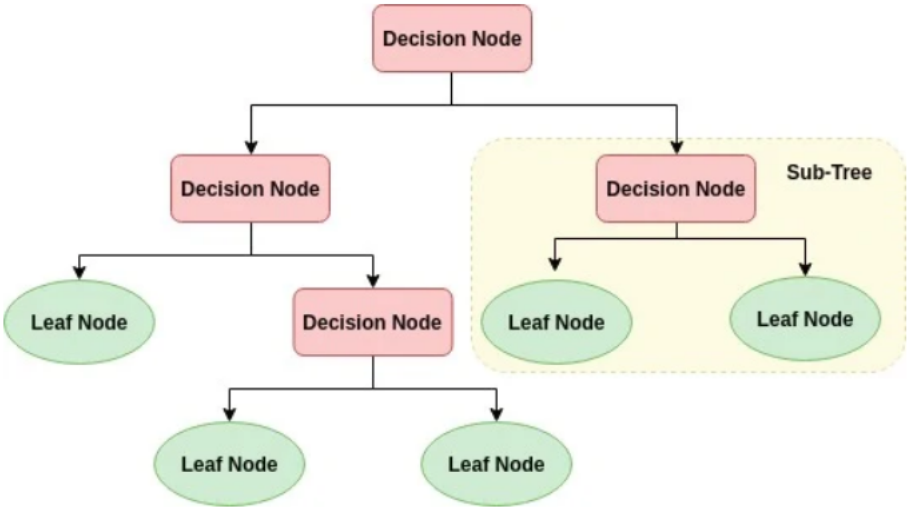

Figura 3.3: Decision Tree Diagram

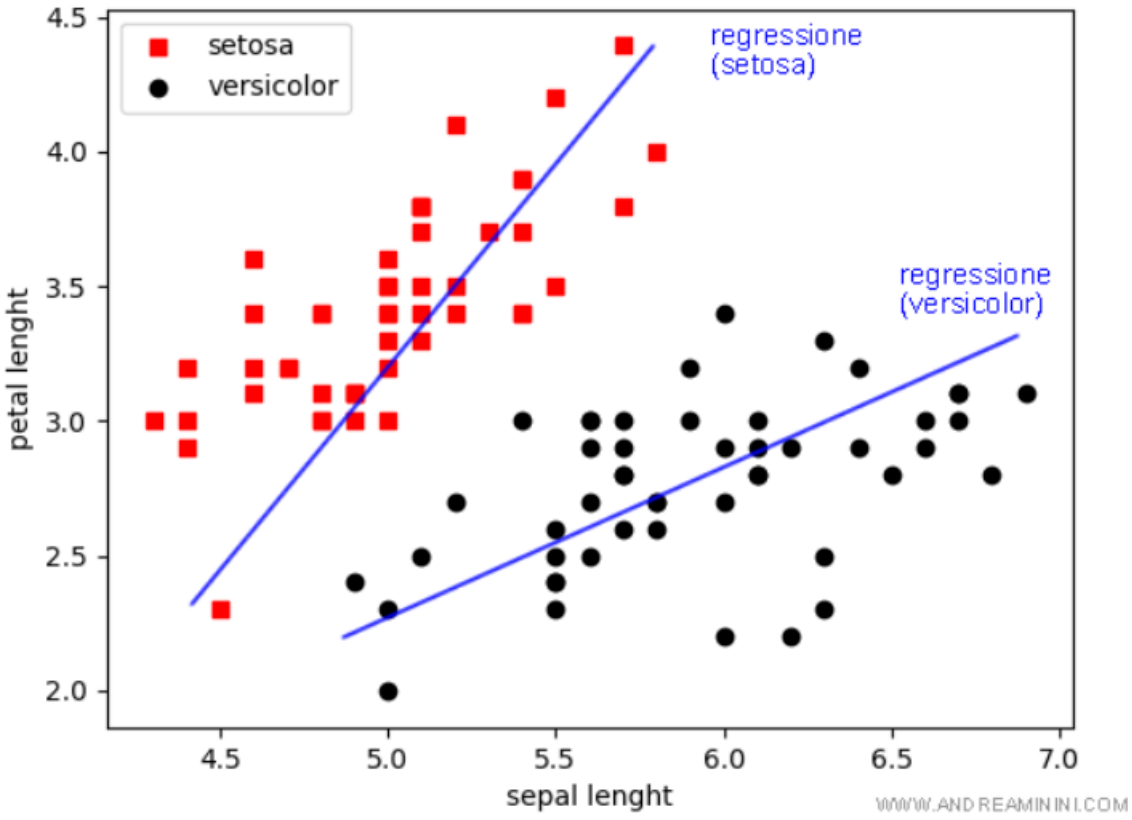

Figura 3.4: Mostra un esempio di utilizzo del linear regression.

K-nearest neighbors

Il k-nearest neighbors (k-NN) [19] è un algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato. In entrambi i casi, l'input è costituito dai k esempi di addestramento più vicini nello spazio delle funzionalità. L'output dipende dall'utilizzo di k-NN per la classificazione o la regressione:

Nella classificazione k-NN, l'output è un'appartenenza a una classe. Un oggetto è classificato da un voto di pluralità dei suoi vicini, con l'oggetto assegnato alla classe più comune tra i suoi k vicini più vicini (k è un numero intero positivo, tipicamente piccolo). Se k = 1, l'oggetto viene semplicemente assegnato alla classe di quel singolo vicino più prossimo.

Nella regressione k-NN, l'output è il valore della proprietà per l'oggetto. Questo valore è la media dei valori di k vicini più vicini.

Fase di apprendimento

Lo spazio viene partizionato in regioni in base alle posizioni e alle caratteristiche degli oggetti di apprendimento. Questo può essere considerato come l'insieme d'apprendimento per l'algoritmo, anche se esso non è esplicitamente richiesto dalle condizioni iniziali.

Calcolo della distanza

fini del calcolo della distanza gli oggetti sono rappresentati attraverso vettori di posizione in uno spazio multidimensionale. Di solito viene usata la distanza euclidea, ma anche altri tipi di distanza sono ugualmente utilizzabili, ad esempio la distanza Manhattan. Nel caso in cui si debbano manipolare stringhe e non numeri si possono usare altre distanze quali ad esempio la distanza di Hamming. L'algoritmo è sensibile alla struttura locale dei dati.

Fase di classificazione

Un punto (che rappresenta un oggetto) è assegnato alla classe C se questa è la più frequente fra i k esempi più vicini all'oggetto sotto esame, la vicinanza si misura in base alla distanza fra punti. I vicini sono presi da un insieme di oggetti per cui è nota la classificazione corretta. Nel caso della regressione per il calcolo della media (classificazione) si usa il valore della proprietà considerata.

SVM

Il Support Vector Machine (SVM) [20] è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato sia per scopi di classificazione che di regressione.

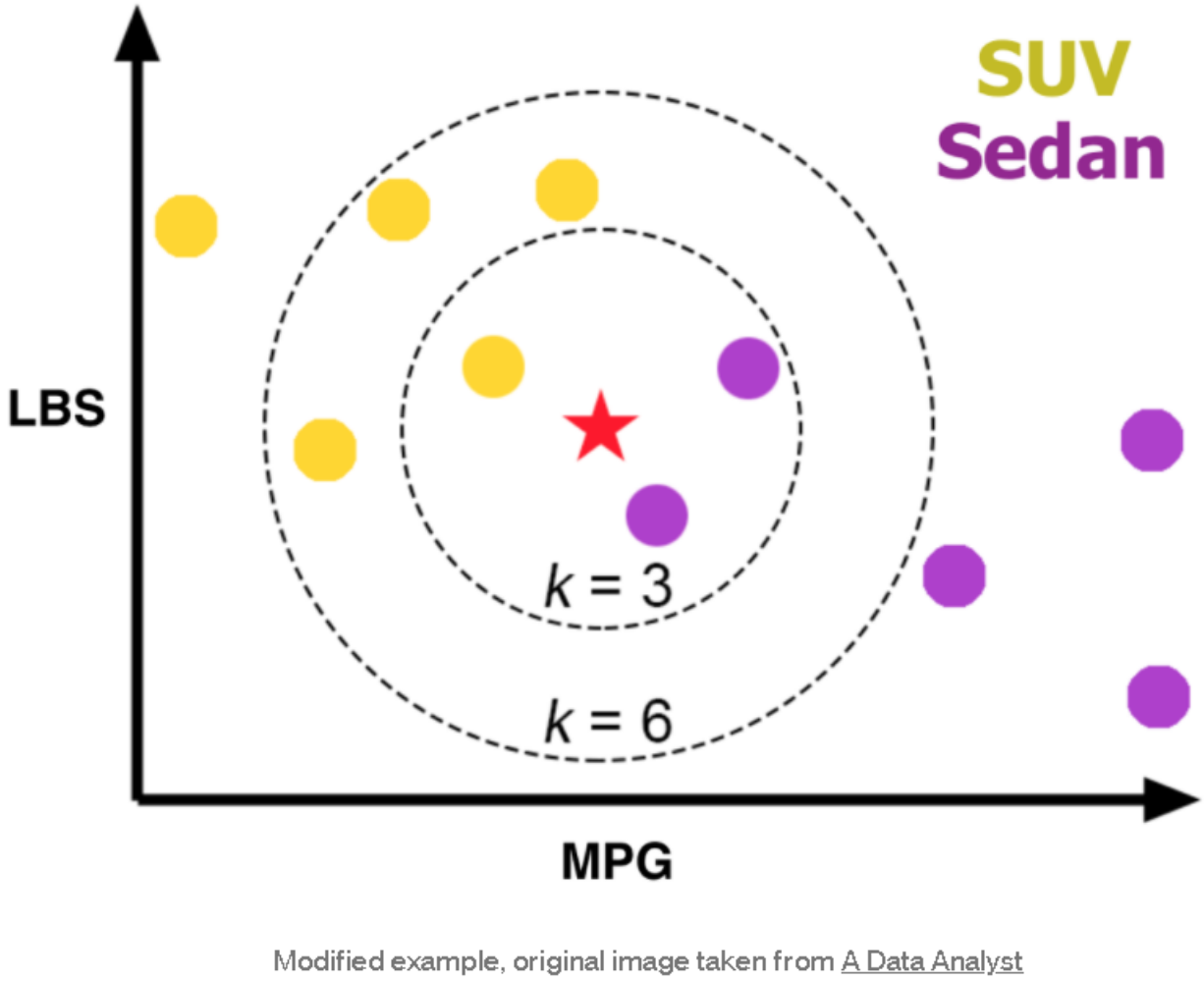

Figura 3.5: Mostra un esempio di utilizzo del KNN.

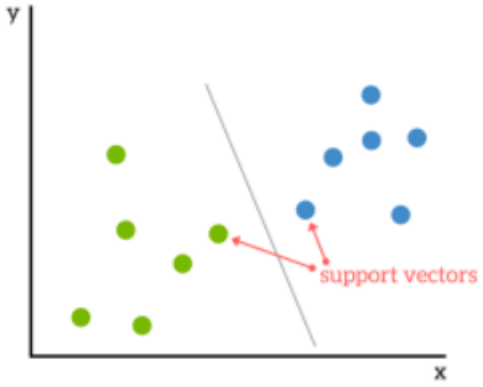

Figura 3.6: Mostra un esempio di utilizzo del SVM.

È popolare in applicazioni quali l'elaborazione del linguaggio naturale, il riconoscimento vocale e delle immagini e la computer vision.

L'algoritmo SVM ottiene la massima efficacia nei problemi di classificazione binari. Anche se viene utilizzato per problemi di classificazione multiclasse, in questo post ci concentreremo prevalentemente sulla classificazione binaria, vedendo principalmente come un algoritmo di questo tipo funziona.

Il Support Vector Machine ha l'obiettivo di identificare l'iperpiano che meglio divide i vettori di supporto in classi. Per farlo esegue i seguenti step:

Cerca un iperpiano linearmente separabile o un limite di decisione che separa i valori di una classe dall'altro. Se ne esiste più di uno, cerca quello che ha margine più alto con i vettori di supporto, per migliorare l'accuratezza del modello.

Se tale iperpiano non esiste, SVM utilizza una mappatura non lineare per trasformare i dati di allenamento in una dimensione superiore (se siamo a due dimensioni, valuterà i dati in 3 dimensioni). In questo modo, i dati di due classi possono sempre essere separati da un iperpiano, che sarà scelto per la suddivisione dei dati.

Grid Search

La ricerca sulla griglia [21] è una tecnica per l'ottimizzazione dell'iperparametro che può facilitare la creazione di un modello e valutare un modello per ogni combinazione di parametri degli algoritmi per griglia.

Ci fornisce l'insieme di iperparametri che dà il miglior punteggio.

Grid Search prende il modello o gli oggetti che preferisci addestrare e diversi valori degli iperparametri. Quindi calcola l'errore per vari valori di iperparametro, consentendo di scegliere i valori migliori.

Random Forrest

Random Forest [22] è un metodo versatile di machine learning, capace di affrontare sia compiti di classificazione che di regressione. Con le foreste casuali è anche possibile applicare metodi per la riduzione della dimensionalità, gestire dati mancanti, valori degli outlier ed altri passaggi essenziali di esplorazione dei dati, producendo buoni risultati.

Il bagging, o bootstrap aggregation, è una tecnica per ridurre la varianza di una funzione di previsione stimata. Il bagging sembra funzionare specialmente bene con procedure ad alta varianza e bassa distorsione, quali gli alberi. Per la regressione, si adatta semplicemente molte volte lo stesso albero di regressione su versioni campionate via bootstrap dei dati di training, e si calcola una media dei risultati. Per la classificazione, si adatta un "comitato" di alberi, ognuno dei queli esprime un voto per la classe prevista.

Le foreste casuali sono una modifica del metodo di bagging che costruisce una grande raccolta di alberi de-correlati, e quindi ne calcola la media. In molti problemi, le performance delle foreste casuali sono elevate; le foreste casuali sono inoltre semplici da addestrare e regolarizzare. Di conseguenza le random forest sono diventate piuttosto popolari.

Bagging

Il bagging [23] è una tecnica di machine learning che rientra nella categoria dell'Apprendimento ensemble. Nel bagging più modelli dello stesso tipo vengono addestrati su dataset diversi, ciascuno ottenuto dal dataset iniziale tramite campionamento casuale con rimpiazzo (bootstrap)[1]. Il nome bagging deriva dalla combinazione delle parole inglesi bootstrap (ovvero il campionamento casuale con rimpiazzo) e aggregation (in riferimento all'aggregazione di più modelli, tipico dell'Apprendimento ensemble).

Voting classifier

Voting classifier [24] è un modello di apprendimento automatico che si basa su un insieme di modelli e prevede un output in base alla loro più alta probabilità . Aggrega semplicemente i risultati di ciascun classificatore passato al Voting Classifier e prevede la classe di output in

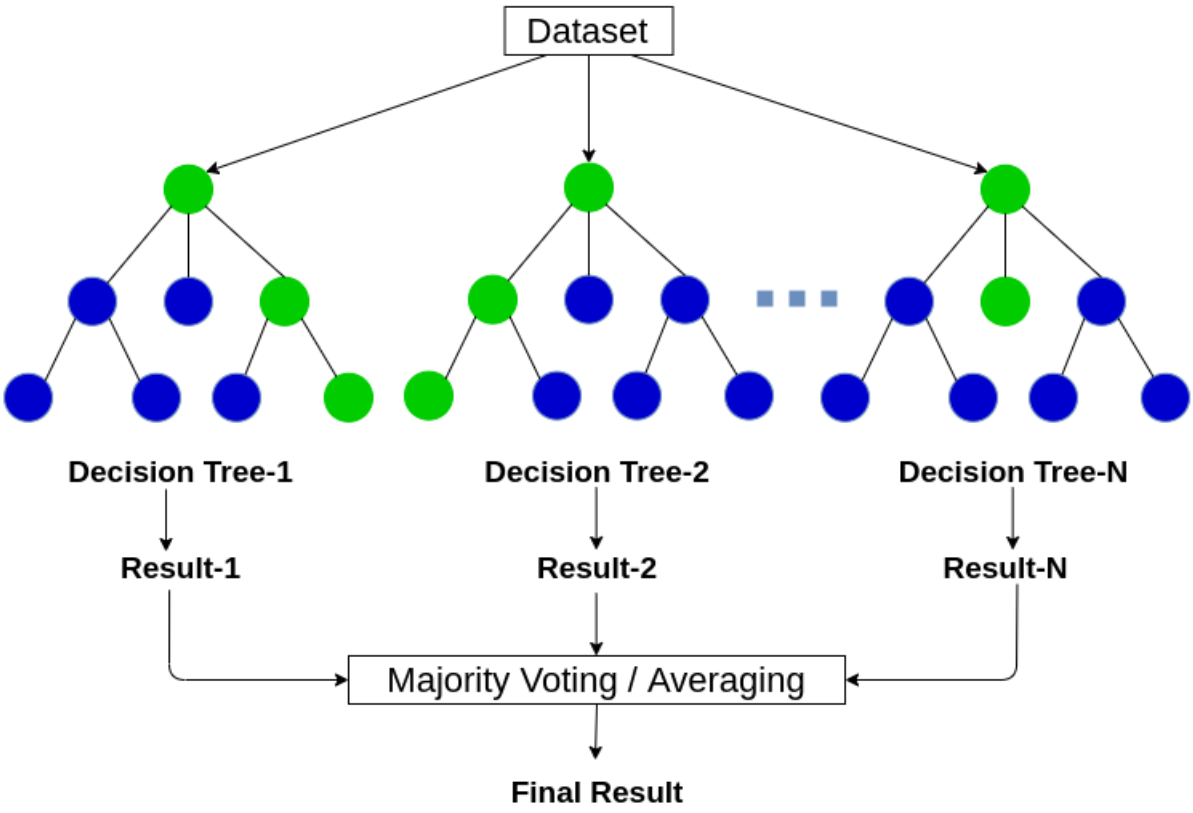

Figura 3.7: Mostra un diagramma del random forest.

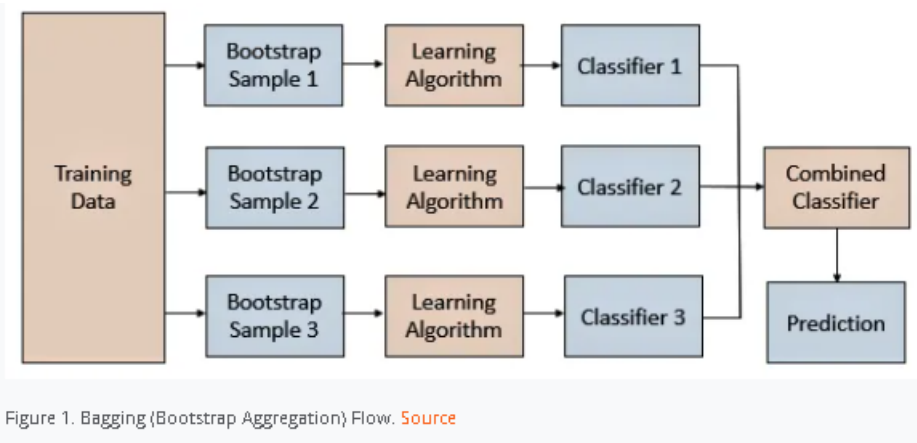

Figura 3.8: Mostra un diagramma del bagging.

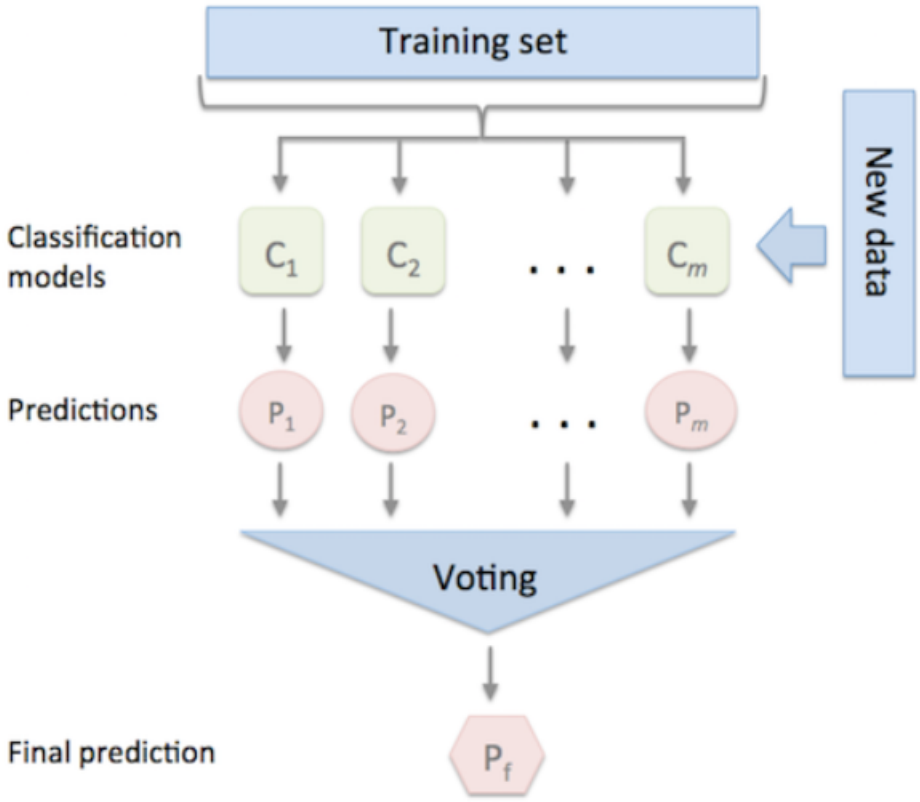

Figura 3.9: Mostra un diagramma del voting classifier.

base alla maggioranza più alta dei voti. L'idea è invece di creare modelli dedicati separati e trovare l'accuratezza per ciascuno di essi, creiamo un singolo modello che si addestra in base a questi modelli e prevede l'output in base alla loro maggioranza combinata di voti per ciascuna classe di output.

Classificatore di voto supporta due tipi di votazioni:

Hard Voting: la classe di output prevista è una classe con la maggioranza più alta di voti, ovvero la classe che ha avuto la più alta probabilità di essere prevista da ciascuno dei classificatori. Supponiamo che tre classificatori abbiano previsto la classe di output (A, A, B), quindi qui la maggioranza ha previsto A come output. Quindi A sarà la previsione finale.

Soft voting: la classe di output è la previsione basata sulla media della probabilità data a quella classe. Supponiamo che, dato un input a tre modelli, la probabilità di previsione per la classe A = (0,30, 0,47, 0,53) e B = (0,20, 0,32, 0,40). Quindi la media per la classe A è 0,4333 e B è 0,3067, il vincitore è chiaramente la classe A perché ha avuto la media di probabilità più alta da ciascun classificatore.

Balancing : OverSampling and UnderSampling

Il problema principale del nostro dataset è il suo forte sbilanciamento. Per definizione un dataset è sbilanciato quando la majority class é molto più grande della minority class. In questo database il 99% dei commit non soffre di vulnerabilità contro l'1% che ne soffre. Essendo molto sbilanciato, pone un problema nei risultati ottenuti nei vari modelli in quanto essi risultano falsificati e non rappresentano la vera accurettezza dei modelli elaborati.

Oversampling e UnderSampling [25] sono due tecniche aventi lo scopo di aggiustare la distribuzione delle classi nel dataset. Le due sono l'una l'opposto dell'altra ma condividono lo stesso obiettivo.

OverSampling techniques

OverSampling per definizione crea nuovi campioni della classe minore e ci sono 4 tecniche principali :

- Random oversampling
- SMOTE
- ADASYN
- Augmentation

UnderSampling techniques

Per definizione UnderSampling elimina campioni dalla classe di maggiorranza e ci sono 4 tecniche principalmente usate:

- Random undersampling
- Cluster
- Tomek links
- Undersampling with ensemble learning

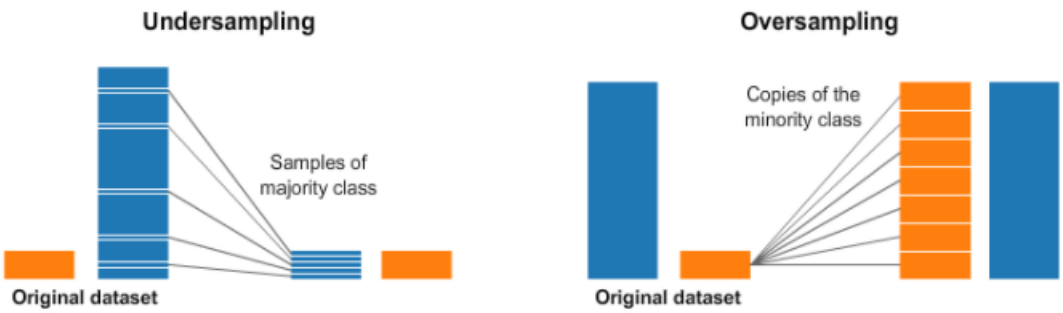

Figura 3.10: Mostra un diagramma dell'OverSampling e dell'UnderSampling.

CAPITOLO 4

Validazione

In questo capitolo , discutteremo la validazione dell'Applicazione di Tecniche di Machine Learning per l'Identificazione di Vulnerabilità a Livello di Commit ,discuteremo degli obbiettivi che vogliamo raggiungere mettendo in evidenzia quanto sono valide le techniche usate in questo scopo.

Anche in questa parte confronteremo l'efficcienza delle techniche usate tra di loro ma anche le famiglie di metriche tra di loro per individuare quelli chi prevedono il megliore risultato.

4.1 Obiettivi di validazione

La prima research question riguarda le prestazioni in generale dei nostri modelli di base su entrambe le famiglie di metriche.

- RQ1:che risultati danno i modelli di base sul nostro modello?

  Ipotizzando che le features scelte nel nostro modello influiscano sulla sua efficienza, la seconda research question é:

- RQ2:i nostri modelli sono influenzati dai metodi di features selection usati?

  Lo sbilanciamento del nostro dataset ci ha indotti a pensare che l'utilizzo di tecniche di balancing potesse avere un riscontro positivo sul modello, portandoci a formulare la terza research question:

- RQ3:Fare il balancing del dataset migliora i risultati ottenuti?

  La quarta research question riguarda gli iperparametri: abbiamo ipotizzato che il loro miglioramento potesse dare risultati più convenienti.

- RQ4:ottimizzare gli iperparametri influisce positivamente sul nostro modello?

  Per concludere, la quinta research question attiene all'efficienza degli ensemble classifiers.

- RQ5:gli ensemble classificators migliorano i nostri risultati ?

4.2 Metodologia di Analisi

Al fine di rispondere alla prima domanda RQ1 sono stati adoperati vari classificatori, tenendo in considerazione sia quelli di base quali il KNN, decision tree, linear regression e SVM sia ensemble classifiers usando il bugging, il bootstrap e il random forest.

Per trovare una soluzione alla seconda domanda, RQ2, sono stati invece utilizzati il mutual information gain per trovare le features più importanti ed il Variance Inflation Factor (VIF) per individuare quelle più correlate ed eliminarle.

Dato il forte sbilanciamento del nostro dataset, per cui il 99% di esso presenta dei commit privi di vulnerabilità, risulta fondamentale l'uso delle tecniche di balancing. A tal fine, ed anche per risolvere la terza research question, è stato preferito l'uso di Synthetic Minority Oversampling Technique (SMOTE). Nella pratica SMOTE va a "sovracampionare" gli esempi nella classe di minoranza, in modo casuale. La procedura di sovracampionamento viene ripetuta più volte fino a quando la classe di minoranza ha la stessa proporzione della classe di maggioranza.

Una risposta alla quarta domanda RQ4 si rinviene nell'impiego del GridSearchCV combinato al SVM; il GridSearchCV è una tecnica per l'ottimizzazione dell'iperparametro che può facilitare la creazione di un modello e valutarlo per ogni combinazione di parametri degli algoritmi per griglia.

Per rispondere alla quinta domanda RQ5, abbiamo usato il Random Forrest e il Bagging Classifier.

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.83	0.25	0.38	20
accuracy	//////////	///////	1.00	5027
macro avg	0.92	0.62	0.69	5027
weighted avg	1.0	1.0	1.0	5027

Tabella 4.1: Tabella chi mostra i resultati ottenuti usando Decision Tree sulle metriche computate da SURFACE.

4.3 Risultati

4.3.1 Prestazioni in generale

In questa parte presenteremo i risultati dell' applicazione dei classificatori di base su entrambe le famiglie di metriche considerate.

A tal fine, è utile definire preventivamente i risultati ottenuti. Essi sono:

Precision [26] : è l'accuratezza con cui il sistema di machine learning prevede le classi positive. E' definito come il rapporto tra i true positive e la somma dei true positive e false positive:

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

Recall [26] : è anche chiamata sensitivity o true positive rate: indica il rapporto di istanze positive correttamente individuate dal sistema di machine learning. La sua formula é :

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

F1 [26] : è conveniente fondere Precision and Recall in una sola metrica chiamata: F1 score.

F1 Score è una media armonica cioè il reciproco della media aritmetica dei reciproci.

$$\frac{TP}{TP + \frac{FN+FP}{2}}$$

Famiglia di metriche computate da SURFACE

Nelle seguenti 4 tabelle (dalla tabella 4.1 fino a 4.4) vengono presentati i risultati dell'applicazione di Decision Tree, Linear Regression, KNN e SVM.

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.00	0.00	0.00	20
accuracy	//////////	///////	1.00	5027
macro avg	0.50	0.50	0.50	5027
weighted avg	0.99	1.0	0.99	5027

Tabella 4.2: Tabella chi mostra i resultati ottenuti usando Linear Regression sulle metriche computate da SURFACE.

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.00	0.00	0.00	20
accuracy	//////////	///////	1.00	5027
macro avg	0.50	0.50	0.50	5027
weighted avg	0.99	1.0	0.99	5027

Tabella 4.3: Tabella chi mostra i resultati ottenuti usando KNN sulle metriche computate da SURFACE.

Classificatori come Linear Regression, KNN e SVM ottengono gli stessi risultati, mostrando la loro capacità di riconoscere i commit privi di vulnerabilità; tuttavia sono totalmente incapaci di riconoscere i commit che soffrono di vulnerabilità.

Per questo approccio naive il Decision Tree contenuto nella tabella 4.1 ha fornito il miglior risultato, presentando un score di 100% per il precision, il recall e il f1 quando i commit non hanno vulnerabilità, mentre per i commit che soffrono di vulnerabilità ha presentato 83% di precision, 25% di recall e 38% di f1.

Process metrics

Dalle quattro tabelle delle process metrics (dalla Tabella 4.5 fino alla 4.8) si evince che i risultati ottenuti sono molto simili dove il precision, il recall e il f1 sono tutti a 99% e sono a 0% quando i commit soffrono di vulnerabilità.

4.3.2 Impatto feature selection

Famiglia di metriche computate da SURFACE

In questa parte abbiamo usato una combinazione tra il mutual gain information e il VIF

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.00	0.00	0.00	20
accuracy	//////////	///////	1.00	5027
macro avg	0.50	0.50	0.50	5027
weighted avg	0.99	1.0	0.99	5027

Tabella 4.4: Tabella chi mostra i resultati ottenuti usando SVM sulle metriche computate da SURFACE.

////	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	1802
1.0	0.00	0.00	0.00	18
accuracy	//////////	///////	0.98	1820
macro avg	0.50	0.50	0.50	1820
weighted avg	0.98	0.98	0.98	1820

Tabella 4.5: Tabella chi mostra i resultati ottenuti usando Decision Tree sulle process metrics.

vedendo ogni volta quali metriche sono più importanti ed eliminando, grazie al VIF, quelle più correlate.

Avere tante variabili correlati tra di loro, infatti, influenza negativamente il nostro modello.

Dalla figure 4.1 possiamo notare che la metrica che influisce maggiormente sul modello é il CSCR con un valore pari a 0.0038, mentre la meno importante é il CCR.

In secondo piano il VIF é stato utilizzato per controllare quanto sono correlate le nostre metriche e si é rilevato che il dataset soffre di una forte correlazione e che la soluzione sarebbe di eliminare volta per volta le features con il VIF più alto fino ad arrivare a un massimo di VIF uguale o inferiore di 5.

Effettuando esattamente 5 iterazioni, il risultato é un dataset privato di 'CA','CCR','CCE','CMR','CME'

Applicando i classificatori di base sul nostro dataset e prendendo in conto la features selection che ha eliminato 5 metriche, i risultati ottenuti sono uguali ai precendenti, poichè da un lato il modello è capace di riconoscere al 100% i commit privi di vulnerabilità, dall'altro lato é totalmente incapace di riconoscere i commit che soffrono di vulnerabilità.

Famiglia di process metrics

Analizzando la figura 4.2 possiamo notare che solo added_lines è la metrica più influente, al contrario di deleted lines.

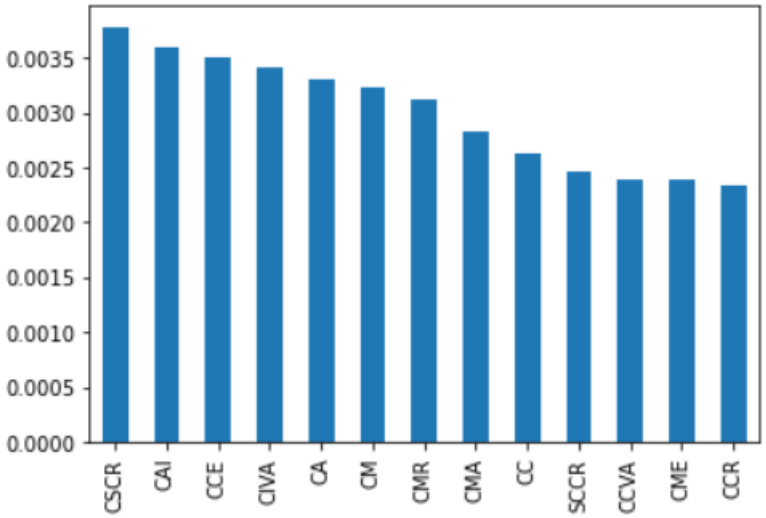

Figura 4.1: Mutual Information Gain sulle metriche computate da SURFACE, senza eliminare le variabili collegate.

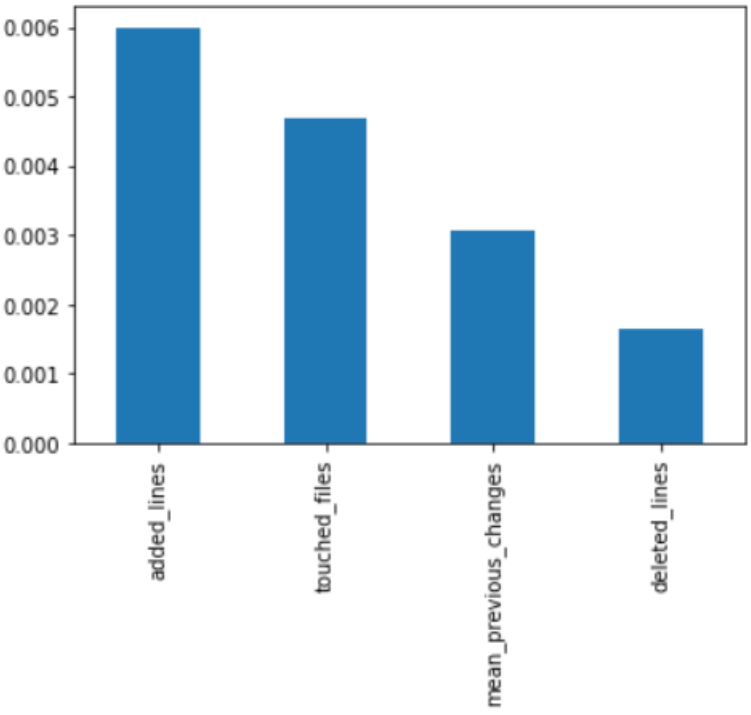

Figura 4.2: Mutual Information Gain sulle process metrics

////	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	1802
1.0	0.00	0.00	0.00	18
accuracy	//////////	///////	0.99	1820
macro avg	0.50	0.50	0.50	1820
weighted avg	0.99	0.99	0.99	1820

Tabella 4.6: Tabella chi mostra i resultati ottenuti usando Linear Regression sulle process metrics.

////	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	1802
1.0	0.00	0.00	0.00	18
accuracy	//////////	///////	0.99	1820
macro avg	0.50	0.50	0.50	1820
weighted avg	0.99	0.99	0.99	1820

Tabella 4.7: Tabella chi mostra i resultati ottenuti usando SVM sulle process metrics.

Il VIF ha dimostrato che le metriche non erano per nulla correlate tra di loro, quindi non c'era nessuna feature da eliminare.

4.3.3 Impatto del data balancing

Considerando il forte sbilanciamento tra il numero di risultati dei commit vulnerabili e non di cui il nostro dataset soffre, si è scelto di utilizzare su di esso l'OverSampling.

Famiglia di metriche computate da SURFACE

Facendo riferimento alla tabella 4.9, possiamo constatare come l'OverSampling abbia un impatto significativo sui risultati dei commit che soffrono di vulnerabilità, soprattutto a livello del recall il quale usando, ad esempio, il KNN, arriva a 70%.

Per quanto riguarda il precision e l' f1, invece, i risultati sono sempre bassi con un leggero miglioramento.

Famiglia di process metrics

La tabella 4.10 evidenzia il miglioramento dei risultati a seguito dell'uso dell' OverSampling: notiamo come il recall dei commit vulnerabili aumenti notevolmente arrivando a 78% per alcuni classificatori, mentre il precision e l'F1 non subiscono aumenti significativi.

////	precision	recall	f1-score	support
0.0	0.99	0.99	0.99	1802
1.0	0.00	0.00	0.00	18
accuracy	//////////	///////	0.99	1820
macro avg	0.50	0.50	0.50	1820
weighted avg	0.99	0.99	0.99	1820

Tabella 4.8: Tabella chi mostra i resultati ottenuti usando KNN sulle process metrics.

////	precision	recall	f1-score	support
0.0	1.00	0.91	0.95	5007
1.0	0.03	0.70	0.06	20
accuracy	//////////	///////	0.91	5027
macro avg	0.52	0.83	0.51	5027
weighted avg	1.00	0.91	0.95	5027

Tabella 4.9: Tabella chi mostra i resultati ottenuti usando KNN sulle metriche raccolte da Surface dopo usare l'OverSampling.

4.3.4 Impatto configurazione iperparametri

Per vedere l'effetto dell'ottimizzazione degli iperparametri sui nostri classificatori, ho combinato il GridSearchCV al SVM.

Famiglia di metriche computate da SURFACE

Dall'esame della tabella 4.11 si rileva che la combinazione di GridSearchCV e SVM non comporta alcun miglioramento nei risultati, i quali rimangono invariati rispetto a quelli ottenuti utilizzando esclusivamente SVM.

Famiglia di process metrics

Come nella famiglia di metriche raccolte su Surface e viste nella tabella 4.11, l'ottimizzazione degli iperparametri non fa accrescere i risultati del SVM.

4.3.5 Impatto degli ensemble classificators

Famiglia di metriche computate da SURFACE

Esaminando i risultati dei commit che soffrono di vulnerabilità contenuti nelle tabelle 4.12 e 4.13, si evince che il valore del precision e dell'f1 aumenta in modo significativo: il valore

////	precision	recall	f1-score	support
0.0	1.00	0.71	0.83	1802
1.0	0.03	0.78	0.05	18
accuracy	//////////	///////	0.71	1820
macro avg	0.51	0.75	0.44	1820
weighted avg	0.99	0.71	0.83	1820

Tabella 4.10: Tabella chi mostra i resultati ottenuti usando KNN sulle process metrics dopo usare l'OverSampling.

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.00	0.00	0.00	20
accuracy	//////////	///////	1.00	5027
macro avg	0.50	0.50	0.50	5027
weighted avg	0.99	1.00	0.99	5027

Tabella 4.11: Tabella chi mostra i resultati dell'uso del GridSearchCV sul SVM.

del precision è pari a 0.71 nel random forrest e 0.80 nel Bagging classifier, mentre quello dell' f1 é uguale a 0.37 per il random forrest e 0.32 per il Bagging classifier.

Possiamo anche notare che il riconoscimento dei commit privi di vulnerabilità é sempre altissimo, pari ad 1.00.

Possiamo anche notare che il riconoscimento dei commit chi non soffrano di vulnerabilità é sempre altissimo dove é uguale ad 1.00

Famiglia di process metrics

Per le process metrics, l'uso degli ensemble classifiers non comporta nessun miglioramento in quanto il classificatore è incapace di riconoscere i commit che soffrano di vulnerabilità.

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.71	0.25	0.37	20
accuracy	//////////	///////	1.00	5027
macro avg	0.86	0.62	0.68	5027
weighted avg	1.00	1.00	1.00	5027

Tabella 4.12: Tabella chi mostra i resultati dell'uso del Random Forrest sulle metriche raccolte da SURFACE

////	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	5007
1.0	0.80	0.20	0.32	20
accuracy	//////////	///////	1.00	5027
macro avg	0.90	0.60	0.66	5027
weighted avg	1.00	1.00	1.00	5027

Tabella 4.13: Tabella chi mostra i resultati dell'uso del Bagging classifier sulle metriche raccolte da SURFACE

CAPITOLO 5

Conclusioni

Giunti al termine del presente lavoro di tesi, possiamo trarre delle conclusioni riguardanti le tecniche affrontate nei precedenti capitoli.

In primis, dall'analisi dei vari approcci utilizzati per lo sviluppo del nostro modello sono emerse due conseguenze fondamentali: da un lato, i migliori risultati sono derivati dalla famiglia di metriche raccolte da SURFACE, per cui con alcuni classificatori si è riusciti ad avere un f1 superiore a 30; dall'altro, i process metrics si sono dimostrati totalmente incapaci di riconoscere i commit che soffrono di vulnerabilità.

In secondo luogo, confrontando i classificatori scelti nell'elaborazione del modello si è rilevato che gli ensemble classifiers e il Decision tree hanno fornito risultati superiori rispetto agli altri.

Il Decision tree ha dato il miglior risultato di tutti, avendo una precision uguale a 0.83 e un valore di f1 pari a 0.38 per i commit privi di vulnerabilità.

I valori del precision e dell'f1 dati dagli ensemble classifiers risultano altissimi rispetto agli altri: mentre i valori degli altri classificatori non vanno oltre lo 0.06, quelli degli ensamble classifiers superano lo 0.75 e 0.30.

Possiamo anche notare un risultato leggermente migliore nel random forrest.

Il problema maggiore di questo progetto si è rivelato il forte sbilanciamento del dataset, nel quale il 99% dei commit non soffre di vulnerabilità; la sfida più

grande è stata cercare di permettere al nostro modello di poter riconoscere i commit aventi delle debolezze di sicurezza.

Al fine di migliorare ulteriormente i risultati ottenuti, i futuri progetti dovranno concentrarsi maggiormente sulla creazione di un dataset più bilanciato, sulla ricerca di nuove metriche che hanno un'influenza significativa sull'efficienza del modello ed infine provare altri approcci come il deep learning.

Bibliografia

[1] Wikipedia. Dynamic program analysis. (Citato a pagina 4)

[2] pp pankaj. Software testing | fuzz testing. (Citato a pagina 4)

[3] Solvit. Web application scanning. (Citato a pagina 5)

[4] Z. Xin Bing Mao Li Xie Ping Chen, Yi Wang. Brick: A binary tool for run-time detecting and locating integer-based vulnerability. (Citato a pagina 6)

[5] Wikipedia. Static program analysis. (Citato a pagina 7)

[6] Wikipedia. Sonarqube. (Citato a pagina 7)

[7] OWASP. Owasp find security bugs. (Citato a pagina 7)

[8] Pierguido Iezzi. Code review: introduzione e spiegazione. (Citato a pagina 7)

[9] Thomas Zimmermann; Nachiappan Nagappan; Laurie Williams. Predicting vulnerable components: Software metrics vs text mining. In *2010 Third International Conference on Software Testing, Verification and Validation*, 2010. (Citato a pagina 8)

[10] Laurie Williams Member IEEE Yonghee Shin, Andrew Meneely and Jason A. Osborne. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. In *IEEE Transactions on Software Engineering*, volume 37, pages 772 – 787, 2011. (Citato a pagina 8)

[11] Jeff Stuckman Riccardo Scandariato, James Walden. Predicting vulnerable components: Software metrics vs text mining. 2014. (Citato a pagina 9)

[12] Jacek undefinedliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, page 1–5, New York, NY, USA, 2005. Association for Computing Machinery. (Citato a pagina 11)

[13] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. PyDriller: Python framework for mining software repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 908–911, New York, New York, USA, 2018. ACM Press. (Citato a pagina 12)

[14] Bandar Alshammari, Colin Fidge, and Diane Corney. Security metrics for object-oriented class designs. In *2009 Ninth International Conference on Quality Software*, pages 11–20, 2009. (Citato a pagina 12)

[15] Wikipedia. Informazione mutua. (Citato a pagina 15)

[16] Analyttica Datalab. What is the variance inflation factor (vif). (Citato a pagina 16)

[17] Wikipedia. Albero di decisione. (Citato a pagina 16)

[18] Andrea Minini. Apprendimento per regressione. (Citato a pagina 16)

[19] Wikipedia. K-nearest neighbors. (Citato a pagina 18)

[20] Lorenzo Govoni. Algoritmo support vector machine. (Citato a pagina 18)

[21] Ichi. Ottimizzazione iperparametrica del classificatore dell'albero decisionale mediante gridsearchcv. (Citato a pagina 20)

[22] Enrico Pegoraro. Random forest. (Citato a pagina 21)

[23] Wikipedia. Bagging. (Citato a pagina 21)

[24] ankushkuwar05. Ml | voting classifier using sklearn. (Citato a pagina 21)

[25] Davide Nardini. Oversampling e undersampling. (Citato a pagina 24)

[26] Andrea Provino. Precision and recall con f1 score | precisione e recupero. (Citato a pagina 28)

Ringraziamenti

Il presente elaborato conclude il mio percorso di laurea triennale, segnando la fine di un periodo pieno di difficoltà ed ostacoli e che tuttavia mi ha permesso di crescere sia come studente sia come persona.

A conclusione di questa tesi, desidero menzionare tutte le persone senza le quali questo lavoro non sarebbe stato nemmeno possibile.

Ai professori Fabio Palomba ed Emanuele Iannone va il mio più sincero grazie per la disponibilità, gentilezza e costante sostegno mostrati durante tutti questi mesi.

Ringrazio la mia famiglia che, seppur da lontano, mi ha supportato e ha sempre creduto in me.

Ringrazio la mia fidanzata e tutta la sua famiglia, continuamente disposti ad aiutarmi e sempre pronti ad includermi nella loro vita, trattandomi come uno di loro e facendomi sentire come se fossi a casa.

Infine il mio speciale ringraziamento va alla prof.ssa Filomena Ferrucci, che dal principio mi ha fornito il suo sostegno e dato la possibilità di essere uno studente dell'Università di Salerno.

Grazie infinite a tutti voi.

Amine