UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Progettazione e sviluppo di un modello per la raccomandazione dei revisori del codice

RELATORE

Prof. Fabio Palomba

Università degli Studi di Salerno

CANDIDATO

Francesco Amato

Matricola: 0512109658

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

Give a man a purpose and the ability to achieve it and he will crawl over broken glass with a smile.

Abstract

Nell'attuale panorama aziendale orientato allo sviluppo del software, le grandi organizzazioni stanno sempre più riconoscendo il valore insostituibile della Code Reviewer Recommendation come pilastro fondamentale per garantire la qualità, la sicurezza e l'efficienza dei propri prodotti software. Questa tesi offre un'approfondita panoramica del mondo delle revisioni del codice, delineando i complessi passaggi coinvolti nel processo di revisione e introducendo gli strumenti utilizzati per la sua implementazione. All'interno di questa ricerca, verrà utilizzato un classificatore per sviluppare un modello di raccomandazione. Il rendimento di questo modello sarà accuratamente analizzato e comparato con i modelli esistenti, allo scopo di valutare l'efficacia e la validità delle raccomandazioni proposte.

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Contesto applicativo

Nell'odierna industria del software, caratterizzata da un costante aumento nella complessità dei progetti e dalla crescente importanza di prodotti software affidabili e sicuri, le revisioni del codice sono diventate un elemento cruciale per garantire la qualità e la sicurezza del software. Le grandi organizzazioni, dai giganti tecnologici alle imprese tradizionali, investono considerevoli risorse nella revisione del codice per identificare e risolvere potenziali problemi, migliorare le prestazioni e garantire la conformità ai requisiti. Tuttavia, il processo di revisione del codice presenta sfide uniche. La mole di codice da esaminare può essere massiccia, e identificare il revisore più adatto per una specifica revisione può risultare complesso. Le organizzazioni spesso mirano a massimizzare l'uso delle competenze e dell'esperienza dei loro sviluppatori, cercando di assegnare loro revisioni del codice rilevanti per le loro specializzazioni. Qui entra in gioco il concetto di Code Reviewers Recommendation, che elimina la necessità di un'assegnazione manuale dei revisori del codice, risparmiando tempo, risorse ed evitando possibili errori.

1.2 Motivazioni e Obiettivi

La pratica della Code Reviewers Recommendation sta rapidamente acquisendo importanza nel contesto attuale, poiché mira a ottimizzare sia il tempo impiegato nelle revisioni del codice che i relativi costi. Le ragioni alla base della realizzazione di questo lavoro derivano dalla necessità di raccogliere e analizzare le principali tecniche sviluppate finora nell'ambito della CRR, insieme ai risultati ottenuti. Questo studio fornisce una solida base per future evoluzioni e per chiunque desideri esplorare l'implementazione di modelli di CRR, contribuendo così a un ulteriore avanzamento di questa importante pratica nell'ambito dello sviluppo del software.

1.3 Risultati ottenuti

In questo studio, è stato impiegato un modello RandomForest per la Code Reviewers Recommendation. I risultati ottenuti non mirano a introdurre nuove prospettive nello studio della CRR, ma piuttosto a fornire un punto di riferimento. Tali risultati sono stati confrontati con due modelli preesistenti: RevFinder [24] e NaiveBayes [16], dimostrando un significativo miglioramento delle prestazioni rispetto a RevFinder, con un aumento del 33.70% nella top-2 accuracy e del 29.13% nel Mean Reciprocal Rank. Tuttavia, confrontando il modello RandomForest con NaiveBayes, il modello presentato ha mostrato prestazioni leggermente inferiori.

1.4 Struttura della tesi

La struttura del presente lavoro è articolata in cinque capitoli. Nel primo capitolo, viene delineato il contesto e la motivazione che ha portato all'implementazione di un sistema di Code Reviewers Recommendation. Nel secondo capitolo, si introduce il concetto di Code Review, esaminando il processo di revisione del codice e analizzando gli strumenti comunemente utilizzati per condurre queste revisioni. Inoltre, vengono esposti i CRR attualmente in uso e le relative limitazioni. Nel terzo capitolo, viene presentato il modello sviluppato all'interno di questo lavoro, fornendo dettagli sui dati impiegati e sulle metodologie utilizzate per la sua implementazione. Il quarto

capitolo è dedicato all'analisi dei risultati ottenuti dal modello. Infine, nel quinto capitolo, vengono delineate le possibili direzioni per gli sviluppi futuri e le aree di ricerca che ne possono beneficiare.

CAPITOLO 2

Background e stato dell'arte

In questo capitolo, esaminiamo in dettaglio il processo di revisione del codice, fornendo una definizione chiara e approfondita e analizzando attentamente le sue fasi chiave. In particolare, dedichiamo una sezione significativa all'importante aspetto della raccomandazione del revisore del codice. Esploriamo in modo dettagliato i modelli attualmente adottati, evidenziandone i vantaggi, ma senza trascurare le sfide associate a questo processo fondamentale nello sviluppo del software.

2.1 Code Review

La revisione del codice è un processo di esame manuale delle modifiche apportate al codice sorgente. Il suo obiettivo primario è rilevare in modo tempestivo eventuali difetti, migliorare la qualità del codice, condividere conoscenze e accrescere la consapevolezza del team, oltre a promuovere la responsabilità condivisa sul codice [6, 21]. Nel corso della revisione del codice, il revisore esamina attentamente il codice sorgente, valutando la sua logica, organizzazione, struttura, leggibilità e la conformità alle linee guida di codifica stabilite dal team, come descritto in [19]. Inoltre, il revisore può effettuare ricerche per individuare potenziali problemi legati alle prestazioni, vulnerabilità di sicurezza o lacune nella copertura dei test. Solitamente, durante

questo processo, vengono impiegati strumenti specifici per la revisione del codice, come le piattaforme di gestione del codice sorgente o i sistemi di controllo delle versioni, per agevolare l'analisi e i commenti relativi al codice.

2.1.1 Processo di Code Review

Il processo di code review ha avuto origine nel 1976 da M.E. Fagan [8] , il quale ha sviluppato un approccio altamente strutturato basato su revisioni manuali di gruppo. Nel corso degli anni, questo processo ha subito un'evoluzione significativa. Attualmente, il processo di code review si basa su strumenti digitali, asincroni e soprattutto informali. Nell'ultimo decennio, sono stati condotti numerosi studi sull'argomento, ma non esiste ancora uno standard definito. Le pratiche di code review variano notevolmente a seconda dell'azienda e persino del progetto. Nella Figura 2.1 è illustrato un grafico esemplificativo che mostra il processo di code review. Inizialmente, un **Autore** crea una patch, che viene poi sottoposta a revisione da parte di un **Committer** (che può anche essere l'Autore stesso). Durante questa fase di revisione, i **Revisori** valutano il codice; in genere, ogni collaboratore è incoraggiato a partecipare alla revisione delle patch. Tuttavia, solo i Revisori con l'autorità di Verifica o Approvazione hanno il potere di determinare se la patch sarà accettata o respinta [12]. Una volta che la patch è approvata, verrà unita al repository da chi l'ha inviata.

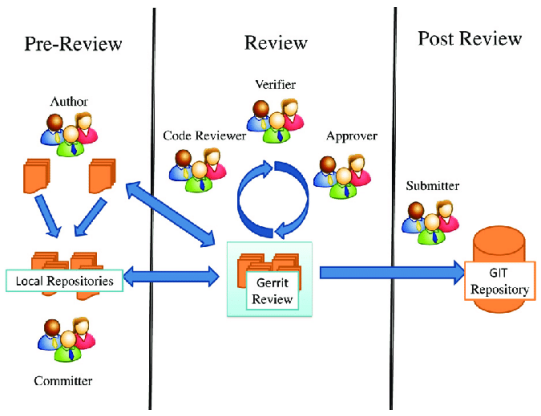

Figura 2.1: Grafico esemplificativo del processo di code review. Fonte [12]

1

2.1.2 Obiettivi della Code Review

La code review ha come principale obiettivo l'individuazione di errori e bug prima che essi vengano incorporati nel sistema. Durante questo processo, i revisori possono suggerire alternative più efficienti o proporre soluzioni migliori per risolvere determinati problemi, offrendo così un'opportunità di apprendimento reciproco tra i membri del team e promuove l'adozione di buone pratiche di sviluppo del software. La collaborazione tra i membri del team è infatti incentivata anche dalla code review, i revisori offrono critiche utili sul codice durante il processo di revisione, aiutando gli sviluppatori ad imparare come migliorare il loro lavoro, questo meccanismo di scambio basato sul dialogo contribuisce alla crescita e allo sviluppo della comunità professionale. Mantenere la coerenza del codice del progetto rappresenta un obiettivo fondamentale. Questo significa che il codice deve rispettare le convenzioni di denominazione, gli stili di programmazione e gli standard di codifica stabiliti dal team. La revisione del codice, offre un'opportunità preziosa per mantenere questa coerenza e individuare eventuali incongruenze o violazioni di tali linee guida. Infine, la revisione del codice può anche aiutare a individuare potenziali problemi di prestazioni. I revisori potrebbero individuare aree del codice che potrebbero essere ottimizzate o offrire opzioni migliori durante la valutazione. Ciò semplifica il compito di garantire che il codice sia scritto in modo efficiente e privo di difetti che potrebbero compromettere le prestazioni globali del software.

2.1.3 Elementi che incidono sulla Code Review

La revisione del codice è un processo critico nell'ambito dello sviluppo del software ed è influenzata da diversi elementi che ne determinano la qualità e l'efficacia complessiva. Prima di esaminare in dettaglio questi aspetti, è importante definire alcuni termini che saranno utilizzati successivamente:

Reviewer experience	La competenza e il livello di abilità dei revisori, che influisce sulla qualità e sulla completezza della revisione.
Patch size (LOC)	La dimensione delle modifiche apportate al codice sorgente misurata in linee di codice (LOC)
Code chunks	Le porzioni di codice modificate o aggiunte in una patch.
Number of modified files	Numero di file sorgente che sono stati modificati in una patch.
Patch writer experience	Esperienza dell'autore della patch
Review queue	La coda delle revisioni che rappresenta il numero di patch in attesa di essere revisionate.
Number of people in the discussion	Numero di individui coinvolti nella discussione durante la Code Review.
Review response time	periodo trascorso tra la sottomissione di una patch e la fornitura di un feedback da parte dei revisori.
The length of the discussion	durata della discussione che riguarda il tempo trascorso nell'analisi e nella risoluzione dei feedback durante la revisione.

Tabella 2.1: Elenco delle caratteristiche considerate durante la code review. [22]

2.2 Strumenti per la Code Review

La code review rappresenta una tappa essenziale nell'ambito dello sviluppo del software, ed è resa ancora più efficace grazie all'utilizzo di specifici strumenti e tecnologie. Esistono diverse categorie di strumenti per la code review, ciascuna con le proprie funzionalità e vantaggi [15]. Tra i più comuni vi sono gli strumenti per la revisione del codice, come Gerrit [3], Github [11] e Review Board [5] che permettono agli sviluppatori di esaminare, discutere e approvare modifiche al codice in modo collaborativo. L'integrazione di questi strumenti nell'ambiente di sviluppo, come gli IDE o le piattaforme di gestione del codice sorgente, semplifica notevolmente il flusso di lavoro, consentendo agli sviluppatori di eseguire revisioni direttamente nell'ambiente in cui lavorano. Inoltre, alcuni strumenti offrono funzionalità di automazione e controllo di qualità, rilevando automaticamente problemi comuni nel codice e contribuendo a migliorarne la qualità generale. La possibilità di commentare e discutere il codice durante la revisione agevola la collaborazione e la comunicazione all'interno del team. Nei paragrafi successivi, esploreremo come avviene il processo di code review utilizzando Gerrit come piattaforma per la code review.

2.2.1 Gerrit

Gerrit [3] è una piattaforma open-source di code review basata su Git [10]. È stata progettata specificamente per agevolare e migliorare il processo di revisione del codice collaborativo, ed è ampiamente utilizzata in progetti open-source e team di sviluppo distribuiti. La Figura 2.2 descrive il processo di code review utilizzando Gerrit come strumento per la code review. Quando Gerrit è configurato come repository centrale , tutte le modifiche al codice vengono inoltrate al Pending Changes per essere riviste e discusse da altri. È possibile inviare una modifica alla code-base una volta che un numero sufficiente di revisori l'ha accettata. Gerrit registra le note e i commenti fatti su ogni modifica, oltre all'archiviazione delle modifiche in sospeso, ciò consente di esaminare le modifiche quando si è più comodi o quando il dialogo su una modifica non può avvenire di persona. Inoltre, le note e i commenti offrono una cronologia di ogni modifica (cosa è stato cambiato, perché è stato cambiato e chi ha revisionato la modifica). Tuttavia, è importante notare che l'adozione di strumenti

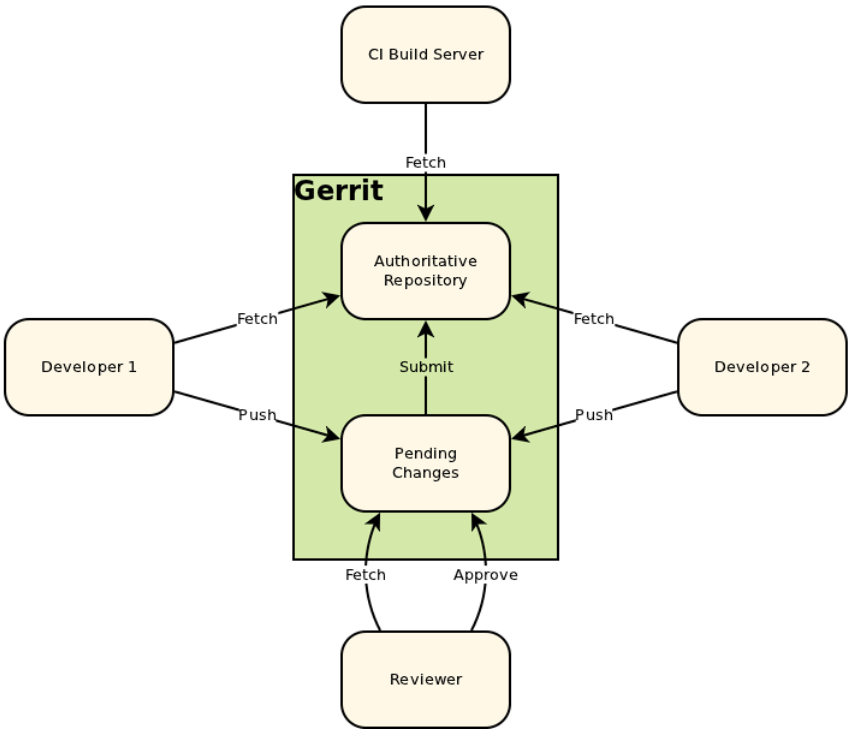

Figura 2.2: Code Review tramite Gerrit. Fonte [4]

. [2]

di code review non risolve automaticamente tutti i problemi associati al processo di revisione del codice [15]. È essenziale avere linee guida chiare e ben definite per la code review, nonché una cultura aziendale che promuova l'importanza della revisione del codice come parte integrante del ciclo di sviluppo del software.

2.3 Code Reviewers Recommendation

Durante il processo di revisione del codice, una delle fasi più cruciali consiste nella selezione dei revisori incaricati di esaminare le modifiche apportate. In un contesto tipico di sviluppo software, questa assegnazione dei revisori avviene manualmente, spesso per mezzo dei responsabili di team, che attribuiscono i compiti di revisione alle modifiche in arrivo, come ad esempio le richieste di pull. Tuttavia, questa metodologia manuale può comportare due potenziali sfide significative. In primo luogo, essendo un processo manuale, può verificarsi un ritardo nell'elaborazione delle modifiche proposte nel codice sorgente. Ciò significa che potrebbe esserci un intervallo di tempo tra il momento in cui viene apportata una modifica e quello in cui essa viene

effettivamente esaminata dai revisori. Questo ritardo può impattare negativamente sul progresso del progetto e sulla tempestiva integrazione di nuove funzionalità o correzioni nel codice. In secondo luogo, la selezione manuale dei revisori potrebbe non sempre garantire la selezione del revisore più idoneo per una particolare modifica. Ciascun revisore ha le proprie aree di competenza e le proprie conoscenze specifiche del progetto. Una selezione inappropriata potrebbe comportare richieste di revisione respinte o revisioni del codice poco efficaci, poiché il revisore potrebbe non disporre delle conoscenze necessarie per valutare accuratamente le modifiche proposte [18]. Secondo l'analisi condotta da Thongtanunam [24], la selezione errata dei revisori durante il processo di revisione può comportare un considerevole allungamento della sua durata. Inoltre, Thongtanunam evidenzia che, in un tipico progetto software, tra il 4% e il 30% delle revisioni presentano problemi di assegnazione dei revisori. Queste revisioni richiedono notevolmente più tempo, fino a 12 giorni in più, per approvare una modifica al codice. Pertanto, diventa cruciale l'utilizzo di strumenti di raccomandazione dei revisori del codice nell'ambito dello sviluppo software al fine di accelerare il processo di revisione. Per evitare tali situazioni, la comunità accademica e il settore industriale hanno sviluppato diverse proposte di modelli di **raccomandazione dei revisori del codice** (CRR) [27]. L'obiettivo comune di tali ricerche è individuare e proporre in modo automatico il revisore più idoneo per un insieme specifico di modifiche apportate al codice [27]. Questi modelli di raccomandazione tengono conto di vari fattori, come l'esperienza del revisore, le competenze specifiche, la disponibilità e la sua precedente esperienza di revisione. L'uso di questi modelli di raccomandazione può aiutare a migliorare la selezione dei revisori, riducendo la latenza nella revisione del codice e migliorando l'efficienza e l'efficacia del processo di Code Review. Nella Tabella 2.2 vengono delineate le peculiarità di ciascun CRR. Tale tabella offre una sintesi delle caratteristiche più comuni riscontrate nei sistemi di raccomandazione destinati ai revisori del codice [16]. Come si può notare dalla tabella, ogni algoritmo si specializza in una specifica caratteristica, conferendo a ciascuno un profilo unico e riconoscibile. Nelle prossime sezioni, approfondiremo in dettaglio vari algoritmi di raccomandazione in uso oggi, esaminandone le peculiarità e le specificità. Ciò ci permetterà di acquisire una comprensione completa delle loro applicazioni nel contesto delle revisioni del codice.

Feature	Paper				Total
	ReviewBot	RevFinder	CoreDevRec	Comm. Net.	
Percorso dei file	**X**	**V**	**V**	**X**	2
Interazioni sociali	**X**	**X**	**V**	**V**	2
Cronologia delle modifiche	**V**	**X**	**X**	**X**	1
Attività dei revisori	**X**	**X**	**V**	**X**	1
Total	1	1	3	1	

Tabella 2.2: Sintesi delle caratteristiche utilizzate nei comuni CRR V=usato,X=non usato [16].

2.3.1 RevFinder

RevFinder è un modello di Code-Reviewers Recommendation proposto da Thongtanunam et al. [24]. Questo approccio, noto come "File-Location based", si basa sull'assunzione che *"i file situati in percorsi simili sarebbero gestiti e revisionati da revisori con esperienze simili"* [24]. Per implementare tale logica, RevFinder calcola un punteggio di compatibilità tra i percorsi dei file revisionati in precedenza mediante operazioni di similarità tra stringhe. Per ogni nuova richiesta di pull, RevFinder genera una lista di revisori, costituita da revisori che hanno revisionato almeno una pull-request in passato. Successivamente, calcola il punteggio di compatibilità, che rappresenta il valore medio delle somiglianze tra ciascun percorso modificato in passato e il percorso della nuova richiesta di pull. Questa lista viene ordinata in base a tali punteggi e il revisore con il punteggio più alto viene assegnato alla nuova richiesta di pull [14].

2.3.2 Comment Network-based Algorithm

Il modello Comment Network-based, introdotto da Yue et al. [26], basa la sua logica sulle interazioni sociali tra sviluppatori e collaboratori all'interno di un progetto. Questo approccio parte dal presupposto che le relazioni all'interno di un team di sviluppo possano emergere attraverso l'analisi dei commenti presenti nelle richieste di pull e nelle conversazioni tra gli sviluppatori. In sostanza, le discussioni e i commenti tra i membri del team possono rivelare chi ha interessi e competenze simili a quelli dell'autore della richiesta di pull. Gli sviluppatori che mostrano affinità con l'autore

della richiesta sono considerati come candidati ideali per la revisione del codice. L'aspetto distintivo di questo approccio è che tiene conto degli interessi comuni tra gli sviluppatori, ma in modo specifico per ciascun progetto. Ciò significa che viene creato un Comment Network dedicato per ogni progetto, che cattura in modo preciso e dettagliato le dinamiche sociali e gli interessi unici di ogni contesto di sviluppo. La sfida principale legata a questo modello riguarda sicuramente l'accessibilità dei dati contenenti messaggi e commenti tra sviluppatori e membri del team. I dataset attualmente disponibili potrebbero non essere adeguati per condurre un'analisi dettagliata. Di conseguenza, è necessario effettuare operazioni di data-mining ad-hoc per raccogliere informazioni specifiche e pertinenti utili a supportare questo modello. Questo processo richiede una ricerca mirata e un'attenta estrazione dei dati per assicurare la qualità e la completezza delle informazioni utilizzate nell'analisi. Un'altra sfida significativa è legata alla veridicità dei commenti, poiché non sempre riflettono accuratamente la realtà delle dinamiche sociali nel team di sviluppo. Inoltre, la natura soggettiva dei commenti potrebbe complicare l'identificazione precisa delle relazioni e degli interessi condivisi tra gli sviluppatori. Questi fattori pongono limiti alla validità e all'affidabilità del Comment Network-based model, richiedendo un'attenta valutazione dei dati e delle informazioni ottenute attraverso questa metodologia.

2.3.3 Support Vector Machines

Le Support Vector Machines (SVM) sono un tipo di algoritmo di apprendimento supervisionato utilizzato principalmente per problemi di classificazione. L'obiettivo principale di una SVM è trovare l'iperpiano (*una sottodimensione di uno spazio n-dimensionale*) che meglio separa i punti di dati delle diverse classi in modo che ci sia il massimo margine tra le classi, visibile nella Figura 2.3. Con **Iperpiano di separazione** si intende una superficie decisionale che divide lo spazio in due regioni, una per ciascuna classe. L'obiettivo è trovare l'iperpiano che massimizza il margine tra le classi. Il margine è la distanza tra l'iperpiano e i punti di dati più vicini delle classi. Questi punti di dati più vicini sono chiamati "support vector." I **Vettori di supporto** sono i punti di dati più vicini all'iperpiano di separazione e sono cruciali per la determinazione del margine. Nel contesto specifico della CRR , le SVM hanno

trovato applicazione in CoreDevRec [14], un modello path-based che utilizza pull storiche di diversi dataset open-source e informazioni legate alle somiglianze tra i percorsi dei file, relazioni e attività tra i revisori. Nonostante le Support Vector

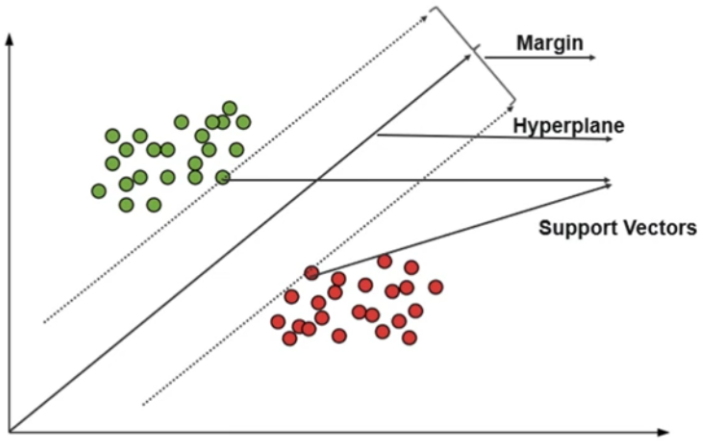

Figura 2.3: Support vector machine [25]

3

Machines siano riconosciute per la loro capacità nell'affrontare problemi complessi, è importante notare che l'addestramento di una SVM può richiedere un tempo significativo quando si lavora con set di dati di grandi dimensioni. In aggiunta, le SVM sono comunemente impiegate in problemi di classificazione binaria, ossia situazioni in cui si tratta di classificare in base a due categorie. Nel contesto della CRR, dove non si hanno soltanto due revisori ma diversi, questa caratteristica costituisce una sfida significativa. Per ovviare a questa problematica esiste l'approccio One-vs-Rest (1vsRest) che permette di affrontare la molteplicità dei revisori. Con l'approccio 1vsRest, invece di trattare tutti i revisori come una singola classe, viene addestrato un classificatore separato per ciascun revisore, considerandolo come una classe distintiva e gli altri revisori come una classe combinata. Questo metodo, sebbene risolva il problema della classificazione binaria, può notevolmente rallentare il tempo di esecuzione.

2.3.4 Naive Bayes Classification

L'algoritmo di classificazione Naive Bayes rappresenta una tecnica diffusa nell'ambito dell'apprendimento automatico, impiegata principalmente per la categoriz-

zazione di testi e dati categorici. Questo modello è stato ampiamente utilizzato nello studio condotto da [16, 17]. Questa tipologia di classificazione si basa sul teorema di Bayes, una tecnica statistica che calcola la probabilità di un evento basandosi sulla probabilità di eventi precedenti correlati e sull'assunzione "naive" (ingenua) di indipendenza condizionale tra le caratteristiche [9]. Il **Teorema di Bayes** si basa sulla relazione [20]:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- P(A | B) detta anche *likelihood function*, rappresenta la probabilità di osservare A dato che B si è già verificato.
- P(B | A) detta anche *posterior probability*, rappresenta la probabilità di osservare B dato che A si è già verificato.
- P(A) e P(B) vengono dette *prior probabilities*, rappresentano la probabilità di osservare A e B indipendentemente dall'altro.

Nella classificazione Naive Bayes, abbiamo una variabile da classificare C, che rappresenta la classe di appartenenza di un esempio E, e un insieme di attributi $(x1, x2, .., xn)$ dove xi rappresenta il valore dell'attributo Xi per quell'esempio. L'obiettivo della classificazione Naive Bayes è determinare quale classe:

$$c \in C$$

è la più probabile data la combinazione di attributi (x1,x2,..,xn) In altre parole, vogliamo trovare la classe che massimizza la probabilità condizionata:

$$P(c|x1, x2, .., xn)$$

La formula del teorema di Bayes, che abbiamo menzionato precedentemente, è fondamentale per calcolare questa probabilità condizionata. Questa formula ci dice che la probabilità che un esempio sia di classe c dato il suo vettore di attributi

$$E = (x1, x2, .., xn)$$

è proporzionale al prodotto di tre componenti:

- P(c) rappresenta la probabilità a priori della classe c, ossia quanto è probabile che un esempio sia di classe c indipendentemente dai suoi attributi.

- P(E) rappresenta la probabilità dell'intero vettore di attributi E indipendentemente dalla classe. È il prodotto delle probabilità a priori di tutti i valori indipendenti degli attributi dell'esempio E

- P(E | c) rappresenta la probabilità condizionata che l'esempio E abbia i valori specifici di attributo x1,x2,..,xn dato che è di classe c. Questa parte è calcolata utilizzando l'assunzione "naive" di indipendenza condizionale tra gli attributi.

Per determinare la classe più probabile, calcoliamo $P(c|x1, x2, .., xn)$ per ogni classe c possibile e selezioniamo quella con la probabilità più alta. Nel contesto della CRR, questo approccio può essere utilizzato per calcolare la probabilità che un revisore sia il candidato migliore per una specifica revisione del codice, basandosi sulle caratteristiche della revisione e le competenze del revisore. In conclusione, l'impiego della classificazione Naive Bayes nelle CRR rappresenta un equilibrio tra semplicità e accuratezza. Nonostante possa produrre risultati soddisfacenti in molte situazioni, è cruciale valutare attentamente i suoi pregi e difetti. L'approccio Naive Bayes si distingue per la sua semplicità sia dal punto di vista concettuale che computazionale. Questa semplicità facilita l'implementazione rapida e l'analisi efficiente dei dati. Inoltre, anche con un limitato numero di dati storici, Naive Bayes è in grado di generare risultati accettabili. Tuttavia, il principale limite di Naive Bayes risiede nell'assunzione di indipendenza condizionale tra le caratteristiche, che potrebbe non sempre riflettere realisticamente la complessità del problema. In scenari dove interazioni complesse tra le caratteristiche sono presenti, Naive Bayes potrebbe non essere in grado di catturare appieno la complessità del problema. Come detto in precedenza questo modello viene ampiamente utilizzato negli studi di [16, 17], nei capitoli successivi verranno valutati i risultati ottenuti tramite questo modello e comparati con i risultati ottenuti dal modello presentato da questo studio.

2.4 Limitazioni stato dell'arte

Le limitazioni riguardanti i CRR sono evidenti e variano principalmente in due aspetti chiave. In primo luogo, come già discusso nei paragrafi precedenti, l'operazione di revisione del codice ha una lunga storia, ma solo di recente si è assistito a

un'accresciuta attenzione verso l'applicazione dell'intelligenza artificiale in questo ambito. Pertanto, le CRR rappresentano una pratica ancora in fase di sviluppo e adozione, e questa novità comporta alcune limitazioni. Innanzitutto, la principale sfida riguarda la disponibilità limitata di dati di addestramento. Poiché le CRR si basano su algoritmi di machine learning, necessitano di una vasta mole di dati per poter fornire raccomandazioni accurate. Tuttavia, poiché questa pratica è relativamente nuova, il numero di dati di revisione del codice etichettati disponibili è limitato. Questo ridotto set di dati può influenzare negativamente la capacità degli algoritmi di apprendere efficacemente i modelli di revisione del codice, portando a raccomandazioni meno precise e affidabili. le CRR inoltre, sono fortemente dipendenti dalla qualità dei dati di input, e questa è un'ulteriore limitazione. La precisione delle raccomandazioni dipende dalla qualità delle informazioni presenti nel codice sorgente, nei commenti e nelle descrizioni delle modifiche. Se questi dati sono incompleti, ambigui o poco chiari, le CRR potrebbero produrre raccomandazioni errate o poco utili. Oltre a ciò, va sottolineato che l'efficacia delle CRR può variare notevolmente a seconda del linguaggio di programmazione, del contesto del progetto e delle preferenze individuali dei revisori. Quindi, è importante considerare che non esiste una soluzione "universale" di CRR che funzioni perfettamente per tutti i casi. Va sottolineato inoltre, che le CRR possono non essere adatte per tutti i tipi di progetti o applicazioni. Alcuni progetti possono richiedere una revisione del codice molto rigorosa e dettagliata, mentre altri possono essere più orientati alla velocità di sviluppo e potrebbero accettare un livello inferiore di revisione. Di conseguenza, le CRR potrebbero non essere la scelta ideale per tutti i contesti e dovrebbero essere utilizzate con discernimento. In conclusione, mentre le Code Reviewer Recommendation rappresentano una promettente evoluzione nel campo delle revisioni del codice, è importante riconoscere le limitazioni attuali, tra cui la disponibilità limitata di dati di addestramento e la dipendenza dalla qualità dei dati di input. Con il passare del tempo e con l'accumulo di più dati e ricerche, ci si aspetta che queste limitazioni possano essere superate, consentendo alle CRR di diventare una pratica sempre più efficace e diffusa nell'ambito delle revisioni del codice.

CAPITOLO 3

Implementazione

Il seguente capitolo è dedicato alla fase di implementazione del modello di raccomandazione dei revisori del codice. Inizieremo fornendo un'analisi approfondita dell'acquisizione dei dataset, delineando le fonti e i metodi utilizzati per raccogliere dati di rilevanza cruciale per l'addestramento del modello. Successivamente, condurremo una discussione sulle operazioni di pre-elaborazione dei dati. Queste operazioni comprenderanno processi di pulizia, normalizzazione e ingegneria delle caratteristiche, contribuendo così a garantire la qualità dei dati. Infine, effettueremo un analisi sul modello di machine leaning adottato e ci concentreremo sulla sua implementazione, questa sezione offrirà una visione completa dell'aspetto operativo del nostro progetto, mettendo in evidenza le strategie e le procedure utilizzate.

3.1 Raccolta dei dati

Per l'addestramento del modello di raccomandazione sono stati impiegati i dataset utilizzati da Jakub Lipcˇak et al [16]. Esso ha infatti condotto una serie di operazioni di data mining utilizzando le API messe a disposizione da GitHub [11] e Gerrit [3], riuscendo a raccogliere oltre 293.377 pull request provenienti da 51 progetti open-source. Nella Tabella 3.1 sono riportate informazioni cruciali sui dataset impie-

gati. È evidente che, a differenza dei dataset utilizzati in studi precedenti, i numeri relativi alle richieste di pull e ai revisori sono notevolmente più alti. Questa disparità è significativa, considerando che molte ricerche precedenti hanno dovuto affrontare la limitazione di utilizzare dataset con un basso numero di richieste di pull e di revisori [16]. Si è deciso di concentrare l'addestramento del modello su un numero

Repository	# Projects	pull requests				reviewers				owners			
		avg	max	min	tot	avg	max	min	tot	avg	max	min	tot
GitHub	37	4,323	29,807	431	159,953	583	1,949	85	21,599	1,061	2,970	313	39,267
Gerrit	14	9,530	31,582	344	133,424	204	651	30	4,558	325	766	27	4,558
Overall	**51**	**5,752**	**31,582**	**344**	**293,377**	**479**	**1,949**	**30**	**24,461**	**859**	**1,061**	**27**	**43,825**

Tabella 3.1: Resoconto dimensioni repositories GitHub e Gerrit. Fonte [16].

ristretto di dataset, specificamente Android, Qt e Openstack. Questa decisione è stata presa per una singola ragione: questi tre progetti sono stati oggetto di valutazioni approfondite in studi precedenti. Utilizzando questi progetti, si è stato in grado di confrontare le metriche del modello con quelle risultanti dagli altri modelli che hanno utilizzato gli stessi dataset. Questo approccio consente di valutare l'efficacia del modello in un contesto comparativo, fornendo così una prospettiva significativa sulle prestazioni del modello di raccomandazione. La Tabella 3.2 presenta dettagliatamente le informazioni sui dataset che abbiamo selezionato per questa ricerca. Questi dati rappresentano non solo numeri e date, ma anche il risultato di anni di sviluppo collaborativo in progetti open source di vasta portata.

#	Project	From	To	Pull Reqs	Reviewers	Owners	Subprj	Files
1	Android	2008-10-24	2012-01-26	5029	93	346	111	26,768
2	Qt	2011-05-17	2012-05-25	23665	200	444	57	77,767
3	Openstack	2011-07-18	2012-05-30	6545	82	324	35	11,409

Tabella 3.2: Resoconto dimensioni dataset utilizzati. Fonte [16].

3.1.1 Struttura dei Dataset

Ciascun dataset è stato archiviato in un file JSON. Ogni file JSON rappresenta l'insieme di pull-request di un progetto specifico, e all'interno di ciascun file, le pull-request sono organizzate in una lista. Nell'esempio visibile nella Figura 3.1, è possibile esaminare un frammento di una pull-request proveniente dal progetto Android.


```
{
  "subProject": "platform/external/wpa_supplicant_8",
  "changeId": "Ife5b96f61422921a3b3830da16227cca8154e2d3",
  "changeNumber": 31152,
  "timestamp": 1326851709000,
  "reviewers": [
    {
      "accountId": "1003981",
      "email": "wangying@android.com",
      "name": "Ying Wang",
      "avatar": "https://lh3.googleusercontent.com/photo.jpg"
    }
  ],
  "owner": {
    "accountId": "1010187",
    "email": "parthan@gmail.com",
    "name": "Partha Narasimhan",
    "avatar": "https://lh6.googleusercontent.com/photo.jpg"
  },
  "filePaths": [
    {
      "location": "wpa_supplicant/ApkBuilderTask.java"
    }
  ]
},
```


Figura 3.1: Esempio dataset

1

Ciascuna pull-request contiene dati cruciali, quali il sotto progetto, il changeId e il changeNumber, che sono unici per ogni pull-request, e il timestamp che indica la data in cui è stata effettuata la pull-request. Ogni pull-request include un elenco di reviewId, identificando gli sviluppatori coinvolti nella revisione, insieme a una lista di file modificati. Inoltre, ogni pull-request è associata a un ownerId, che identifica lo sviluppatore che ha presentato la richiesta di revisione del codice. Dettagli come l'email o l'avatar dei revisori e del proprietario non sono stati considerati; parleremo di questi aspetti nei paragrafi successivi 3.2.

3.2 Data Preprocessing

Il preprocessing rappresenta una delle fasi cruciali durante l'addestramento di un modello, poiché incide significativamente sulla qualità dei dati ottenuti. Nel contesto dei dataset utilizzati, sebbene avessero una struttura di base simile, è stata necessaria un'approfondita fase di Data Cleaning per affrontare dati sporchi e inconsistenze. Attraverso la Feature Extraction, sono state estratte informazioni rilevanti per l'attuale contesto, catturando dettagli cruciali tra revisori e richieste di revisione. Infine, la Feature Selection ha consentito di mantenere solo le variabili più significative, semplificando il modello e migliorandone la precisione. Approfondiremo questi aspetti nelle prossime sottosezioni 3.2.1, 3.2.2 e 3.2.3.

3.2.1 Data Cleaning

Il primo passo fondamentale è stato convertire ogni dataset da file JSON a file CSV. Questa conversione è stata eseguita per semplificare la gestione dei dati, rendendo possibile lavorare con maggiore agilità e precisione. I file CSV presentano una struttura dati tabellare che semplifica operazioni come il filtraggio, l'ordinamento e l'aggregazione dei dati, offrendo un'organizzazione lineare che agevola notevolmente l'analisi. Durante questa fase, sono stati inoltre apportati miglioramenti significativi all'interpretazione delle informazioni temporali. Si è infatti deciso di convertire le date di revisione dal formato Timestamp a un formato Date più comprensibile, offrendo una chiara prospettiva cronologica. Questa conversione ha permesso di analizzare in modo più accurato le tempistiche delle revisioni, consentendo una visione dettagliata delle dinamiche temporali all'interno dei dataset. In aggiunta, sono state effettuate operazioni mirate sui revisori e sul percorso dei file. Come precedentemente menzionato, si è automaticamente escluso dalla raccomandazione il proprietario della revisione se già incluso nella lista dei revisori, risparmiando risorse in raccomandazioni superflue. Infine, sono state rimosse le revisioni relative a file considerati inutili (*readme, makefile, gitignore..*). Queste operazioni di pulizia hanno contribuito a mantenere i dataset focalizzati sulle informazioni rilevanti, ottimizzando così il processo di raccomandazione.

3.2.2 Feature Extraction

Nella fase di feature extraction sono stati presi in considerazione due tipi di caratteristiche: quelle relative al percorso dei file revisionati e quelle riguardanti le attività dei revisori nel progetto.

Feature relative ai path

In un precedente studio condotto da Thongtanunam et al. [24], si è osservato come i file situati in percorsi simili vengono revisionati da revisori con esperienza simile. Sulla base di questa osservazione, si è scelto di adottare, come nello studio di [14], l'approccio TF-IDF per estrarre le similarità tra i path dei file revisionati. A differenza dello studio di Thongtanunam, che si basava su una tecnica di confronto tra stringhe, è stata quindi utilizzata questa tecnica di machine learning per valutare la similarità dei percorsi dei file. La TF-IDF (*Term Frequency-Inverse Document Frequency*) è una metodologia di machine learning che misura l'importanza di un termine all'interno di un insieme di documenti, nel nostro contesto specifico, essa valuta l'importanza di un termine all'interno dei percorsi dei file che hanno subito una revisione. Ciò avviene estraendo, per ciascun percorso, sotto-stringhe consecutive. Per esemplificare, consideriamo il percorso "src/com/android/ApkBuilderTask.java". In questo caso, verranno estratte quattro sotto-stringhe dall'inizio del percorso: "src", "src/com", "src/com/android", "src/com/android/ApkBuilderTask.java". Queste sotto-stringhe costituiranno quattro feature distinte il cui peso è calcolato nel seguente modo:

$$peso_{x,y} = tf_{x,y}(log(\frac{NP}{df_y}) + 1)$$

Dove x rappresenta la sotto-stringa nella pull-request y, $tf_{x,y}$ è il numero di volte in cui la sotto-stringa x appare nella pull-request y, df_y è il numero di pull-request che contengono la sotto-stringa x e NP è il numero totale di pull-request nel dataset [14]. Questa formula assegna un peso alle sotto-stringhe basato sulla loro frequenza nella specifica pull-request e sulla loro rarità nel dataset. Nella Tabella 3.3, viene illustrato un esempio di applicazione della tecnica TF-IDF su due percorsi: "src/com/android/ApkBuilderTask.java" e "src/com/android/LocalSettings.java". Come evidenziato, le prime tre sotto-stringhe hanno lo stesso valore in entrambi

i percorsi, poiché entrambi i percorsi contengono quelle specifiche sotto-stringhe. Al contrario, le due sotto-stringhe successive differiscono nei loro valori, perché "ApkBuilderTask.java" appare solo nella prima pull-request e "LocalSettings.java" appare solo nella seconda pull-request. Pertanto, hanno una frequenza diversa ($tf_{x,y}$) e quindi valori TF-IDF diversi nella tabella.

	src	src/com	src/com/android	src/com/android/ApkBuilderTask.java	src/com/android/LocalSettings.java
0	0.448321	0.448321	0.448321	0.630099	0.000000
1	0.448321	0.448321	0.448321	0.000000	0.630099

Tabella 3.3: Esempio di esecuzione della TF-IDF

Feature relative alle attività

Sono state individuate non solo le caratteristiche associate ai percorsi dei file soggetti a revisione, ma anche altre peculiarità rilevanti legate alle attività dei revisori del codice. Poiché essi ricoprono un ruolo di notevole responsabilità nel processo di revisione, è stato essenziale tener conto delle loro attività. Oltre ad estrarre delle feature uniche per i dataset utilizzati sono state prese in considerazione alcune feature precedentemente utilizzate nello studio condotto da [14]. *Evaluate Pulls* riflette il numero di revisioni effettuate in precedenza dai revisori ed è stata utilizzata per valutare la produttività di tali revisori. Allo stesso modo, *First Time* rappresenta l'intervallo di tempo tra la prima revisione effettuata dal revisore e l'inizio del set di test. *Total Pulls* costituisce un indicatore significativo, impiegato esclusivamente per valutare la scalabilità del modello, considerando il numero totale di richieste di pull nel progetto. *RevByOwner* indica il numero di pull-request revisionate da un utente specifico per un proprietario specifico della pull-request, mentre *FileByOwner* rappresenta il numero di file revisionati per un proprietario specifico della pull-request. Si é deciso di escludere le feature legate al fattore temporale a causa delle restrizioni presenti nei dataset utilizzati. I dataset utilizzati non includevano informazioni sulla data di revisione o sulla data di chiusura della revisione, rendendo impossibile l'integrazione di tali variabili nel nostro modello.

3.2.3 Feature Selection

Nella fase di Feature Selection, sono stati dedicati sforzi considerevoli per identificare le feature più rilevanti per il problema. Questa fase è stata cruciale nel processo di costruzione del modello, in quanto ha aiutato a ridurre la complessità del modello stesso e a migliorare le prestazioni predittive. Per identificare le feature più informative e indipendenti tra di loro, é stato considerato il coefficiente di correlazione di Spearman, con una soglia di threshold pari a 0.75 oltre la quale le feature vengono eliminate. Questo criterio permette di escludere le feature che mostravano una correlazione molto forte, evitando così la ridondanza di informazioni all'interno del dataset. Le feature con una correlazione inferiore a questa soglia sono state considerate come potenziali candidati per il nostro modello. E' inoltre stata calcolata la feature importance, eliminando quelle con un valore di importanza pari a 0. Questo approccio ha consentito di individuare non solo le feature altamente correlate, ma anche quelle che non contribuivano significativamente alla previsione del modello.

3.3 Classificatore utilizzato

Il modello selezionato per affrontare il problema della raccomandazione dei revisori è il **RandomForest**. Questo algoritmo, parte della famiglia degli alberi decisionali, è noto per la sua capacità di gestire sia dati numerici che categorici e per la sua flessibilità nel gestire grandi insiemi di dati.

3.3.1 DecisionTree

Come già accennato, RandomForest è un membro della famiglia degli Alberi Decisionali. Un Albero Decisionale è un algoritmo di apprendimento supervisionato non parametrico impiegato sia per compiti di classificazione che di regressione. Dalla Figura 3.2 possiamo osservare come la sua struttura è organizzata come un albero gerarchico, comprendente un nodo radice, rami, nodi interni e nodi foglia. Il nodo radice rappresenta l'inizio del processo decisionale, mentre i rami si diramano nei nodi interni, ciascuno dei quali rappresenta una decisione basata su una specifica caratteristica. Infine, i nodi foglia rappresentano le previsioni finali del modello, che

possono essere etichette di classe per problemi di classificazione o valori numerici per problemi di regressione [2].

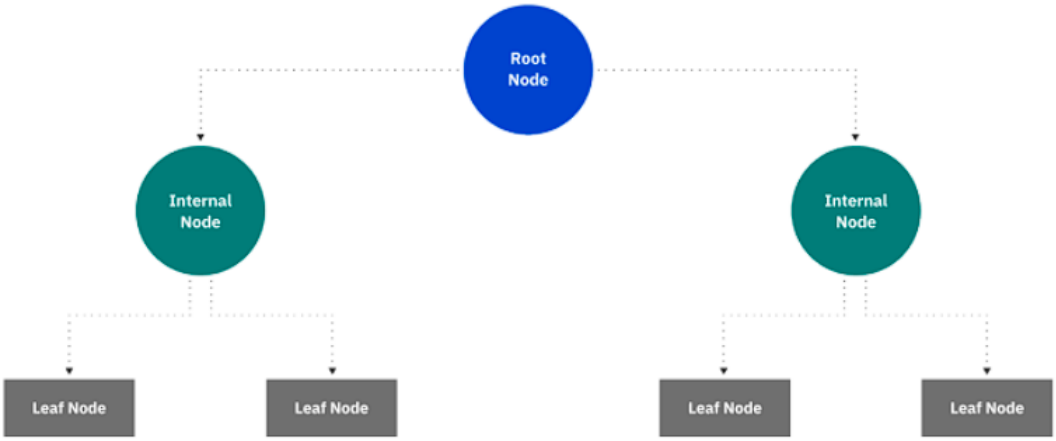

Figura 3.2: Albero decisionale. Fonte [2].

2

Gli alberi decisionali si basano sul principio del *Divid et Impera*, questo approccio è alla base del loro funzionamento: l'algoritmo divide il problema complesso in decisioni più piccole e gestibili. Ogni nodo interno rappresenta una divisione dei dati in base a una specifica caratteristica, riducendo così il problema originale in sotto-problemi più semplici. Questi sotto-problemi vengono poi ulteriormente divisi nei nodi successivi, seguendo lo stesso principio, fino a quando si raggiungono i nodi foglia. La problematica fondamentale legata agli alberi decisionali è la loro incapacità di affrontare efficacemente problemi complessi. All'aumentare delle dimensioni dell'albero, si verifica spesso una suddivisione eccessiva dei dati, noto come **overfitting**. Questo fenomeno comporta una frammentazione eccessiva dei dati, compromettendo l'accuratezza delle previsioni. Per prevenire tale problema, gli alberi decisionali preferiscono strutture concise e mirano a mantenerle il più semplici possibile [2].

3.3.2 RandomForest

Dato l'elevato grado di complessità del problema affrontato, non era possibile gestirlo efficacemente mediante un Decisione Tree. Pertanto, è stato adottato l'approccio per mezzo del RandomForest. Questa scelta è derivata dalla capacità di

RandomForest di gestire in modo ottimale il problema dell'overfitting, offrendo una soluzione più robusta e accurata. L'algoritmo RandomForest opera combinando i risultati di diverse strutture ad albero decisionali per produrre un unico output, come in Figura 3.3. Questo approccio si basa sul concetto di casualità delle caratteristiche (*o bagging delle caratteristiche*) [1]. La casualità delle caratteristiche permette di migliorare sia la diversità che l'accuratezza degli alberi decisionali all'interno di un modello Random Forest. Questo avviene selezionando casualmente solo un sottoinsieme delle caratteristiche disponibili per ciascun albero decisionale. Di conseguenza ogni albero è addestrato su un insieme diverso di caratteristiche, contribuendo così a migliorare l'efficacia e la robustezza complessiva del modello.

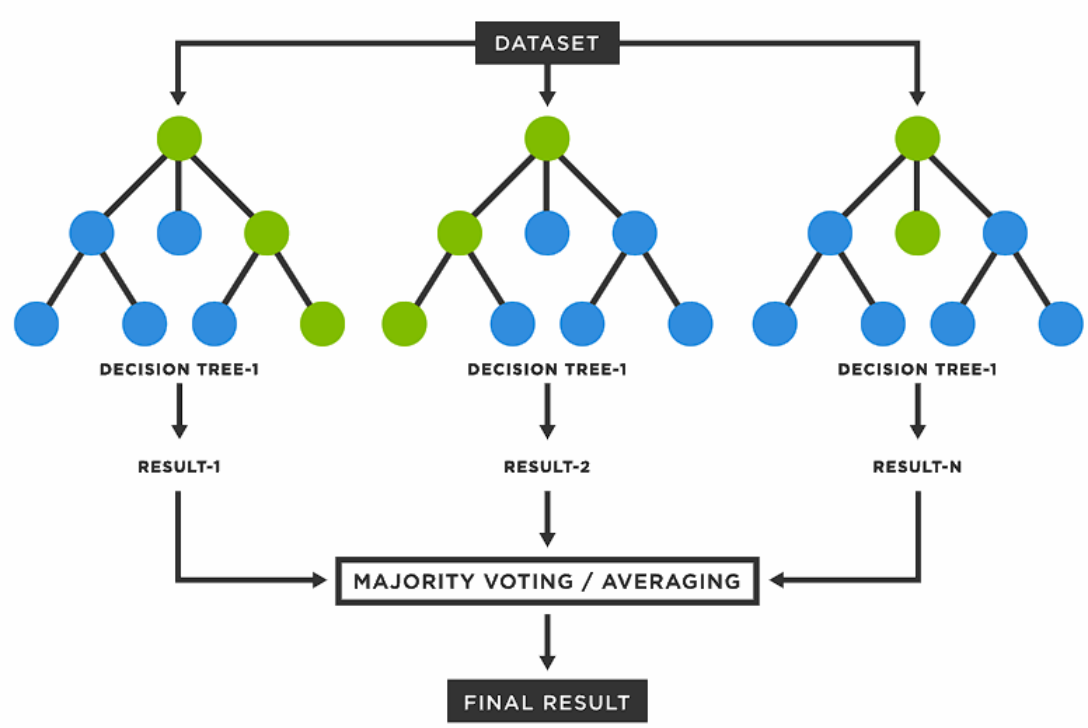

Figura 3.3: RandomForest. Fonte [1].

3

3.4 Fold-Validation

Come detto nei capitoli precedenti i dataset che sono stati considerati sono quelli utilizzati in [16], per questo motivo è stato deciso di adottare la stessa metodologia di cross-validation, ovvero l'11-fold validation, ispirata dagli studi di Bettenburg et

al [7]. e Jeong et al [13]. In questo approccio, i dataset sono stati suddivisi in 11 fold della stessa dimensione e ordinati in ordine cronologico, come in Figura 3.4.

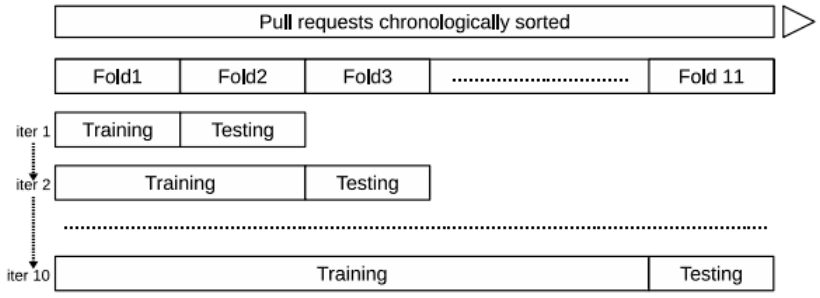

Figura 3.4: 11-fold validation. Fonte [16].

4

CAPITOLO 4

Analisi dei risultati

Nel capitolo dedicato all'Analisi dei Risultati, viene esaminato il modello implementato in confronto con algoritmi già esistenti, come RevFinder [24] e l'approccio tramite Naive Bayes [16, 17]. Verranno analizzate a fondo le prestazioni del nostro modello rispetto a questi benchmark, evidenziando le differenze chiave e le eventuali migliorie apportate.

4.1 Metriche di valutazione dei CRR

Durante la valutazione del modello implementato, sono state adoperate due metriche statistiche cruciali frequentemente utilizzate nei problemi di raccomandazione: la Mean Reciprocal Rank (MRR) e la Top-k Accuracy.

4.1.1 Mean Reciprocal Rank

La Mean Reciprocal Rank (MRR) è una metrica ampiamente usata per valutare la qualità dei sistemi di raccomandazione. MRR restituisce un valore che riflette la posizione media del revisore nella lista generata dall'algoritmo di raccomandazione dei revisori [16]. Un MRR prossimo a 1 indica che il sistema di raccomandazione è in

grado di collocare spesso il revisore corretto in cima alla lista di raccomandazioni, indicando un'elevata prestazione. Al contrario, un punteggio vicino a 0 suggerisce che il sistema di raccomandazione ha difficoltà a posizionare regolarmente il revisore corretto, indicando prestazioni scadenti. L'MRR viene calcolato mediante la seguente formula:

$$MRR = \frac{1}{|N|} \sum_{i=1}^{N} \frac{1}{rank(i)}$$

Dove N corrisponde al numero totale di revisioni, $rank(i)$ corrisponde al rank del primo revisore corretto suggerito per la i-esima revisione.

4.1.2 Top-k Accuracy

La Top-k Accuracy è un indicatore cruciale che valuta l'efficacia del modello nel processo di classificazione delle osservazioni. Questa metrica mira a determinare se il revisore corretto è incluso tra le prime k raccomandazioni fornite dal sistema di raccomandazione, senza tener conto dell'ordine esatto di queste raccomandazioni. Per calcolare la Top-k Accuracy, il sistema di raccomandazione genera una lista di k potenziali revisori per ciascuna review del codice, seguendo le linee guida descritte in riferimenti precedenti [15, 24, 23]. Successivamente, si effettua un controllo per verificare se il revisore corretto è effettivamente incluso tra questi k revisori suggeriti. È rilevante notare che i valori di k utilizzati per calcolare la Top-k Accuracy corrispondono a quelli suggeriti dalle ricerche precedentemente menzionate, ovvero (1, 3, 5, 10). La Top-k Accuracy viene calcolata tramite la seguente formula:

$$top - kAccuracy = \frac{R}{|N|}$$

Dove R rappresenta il numero di raccomandazioni corrette tra i primi k revisori suggeriti, mentre N rappresenta il numero totale di revisioni nel dataset di valutazione.

4.2 Risultati ottenuti

In questo paragrafo vengono presentati i risultati ottenuti attraverso il confronto tra le prestazioni del modello proposto, RevFinder e l'approccio Naive Bayes. Le

Tabelle 4.1 e 4.2 mostrano l'accuratezza dei due modelli, evidenziando i risultati conseguiti.

Dataset	Top-1	Top-3	Top-5	Top-10	MRR
Android	47.29%	72.49%	81.17%	89.50%	0.617
OpenStack	37.49%	65.61%	77.04%	87.66%	0.544
Qt	18.21%	33.21%	40.65%	51.55%	0.297
Average	34.33%	57.10%	66.29%	76.24%	0.486

Tabella 4.1: Risultati RevFinder. Fonte [17, 16].

Dataset	Top-1	Top-3	Top-5	Top-10	MRR
Android	53.65%	80.78%	86.89%	91.79%	0.679
OpenStack	39.93%	70.90%	80.60%	89.69%	0.574
Qt	37.37%	65.67%	75.33%	84.30%	0.540
Average	43.65%	72.45%	80.94%	88.59%	0.597

Tabella 4.2: Risultati Naive Bayes. Fonte [17, 16].

La tabella 4.3 rappresenta i risultati ottenuti tramite il modello proposto:

Dataset	Top-1	Top-3	Top-5	Top-10	MRR
Android	51.93%	78.52%	86.67%	94.11%	0.672
OpenStack	45.06%	73.88%	83.45%	90.68%	0.623
Qt	30.64%	59.85%	70.26%	81.63%	0.487
Average	42.54%	70.75%	80.12%	88.80%	0.594

Tabella 4.3: Risultati modello proposto

Confronto delle soluzioni

Per valutare l'efficacia del modello proposto rispetto alle due soluzioni già esistenti, è stata adottata la seguente formula di valutazione [17]:

$$Miglioramento = \frac{Res1 - Res2}{Res2} \times 100$$

Questa formula calcola il miglioramento percentuale del modello proposto rispetto alle soluzioni preesistenti, dove Res1 rappresenta il risultato ottenuto dal modello proposto e Res2 indica il risultato delle soluzioni esistenti. Un miglioramento positivo indica un aumento delle prestazioni del modello proposto rispetto alle soluzioni esistenti. Le Tabelle 4.4 e 4.5 mostrano i miglioramenti ottenuti tramite il modello proposto.

Dataset	Top-1	Top-3	Top-5	Top-10	MRR
Android	9.81%	8.31%	6.77%	5.15%	8.91%
OpenStack	20.19%	12.60%	8.32%	3.44%	14.52%
Qt	68.25%	80.21%	72.84%	58.35%	63.97%
Average	32.75%	33.70%	29.31%	22.31%	29.13%

Tabella 4.4: Miglioramenti di RandomForest rispetto a RevFinder

Dataset	Top-1	Top-3	Top-5	Top-10	MRR
Android	-3.20%	-2.79%	0.25%	2.52%	-1.03%
OpenStack	12.84%	4.20%	3.53%	1.10%	8.53%
Qt	-18.09%	-8.86%	-6.73%	-3.13%	-9.81%
Average	-2.81%	-2.48%	-0.98%	0.16%	-0.77%

Tabella 4.5: Miglioramenti di RandomForest rispetto a Naive Bayes

Dalla Tabella 4.4, è evidente che l'approccio basato su RandomForest ha ottenuto una precisione significativamente superiore rispetto all'algoritmo RevFinder. La top-2 accuracy è migliorata del 33.70% e l'MRR medio ha visto un incremento del 29.13%. Questo significativo miglioramento può essere attribuito alla mancanza di considerazione da parte di RevFinder dei revisori passati e all'uso di operazioni di similarità tra stringhe per confrontare i percorsi dei file revisionati. Inoltre, va notato che nel contesto di questo studio è stato utilizzato un insieme di caratteristiche (feature-set) molto più ampio rispetto a quello impiegato da RevFinder. Come discusso nei capitoli precedenti (3.2.2, 3.2.3), sono state incorporate sia caratteristiche relative ai percorsi dei file revisionati sia caratteristiche legate alle attività dei revisori. Queste ultime non sono state considerate in RevFinder, il che ha contribuito in modo significativo al

notevole miglioramento delle prestazioni osservato nel modello proposto. La Tabella 4.5 mostra i risultati del confronto tra l'approccio Naive e il modello proposto. A differenza del confronto con RevFinder, in questo caso si osserva che RandomForest è leggermente meno efficiente. La differenza nella top-5 accuracy è solamente del -0.98% con un MRR medio del -0.77%. Questa differenza nelle metriche è dovuta al fatto che l'approccio proposto da [16, 17] tiene conto anche dei nomi dei progetti. Nel suo studio infatti valuta l'importanza di tutte le feature utilizzate, evidenziando come il nome dei progetti rappresenti una delle caratteristiche con maggiore impatto sull'accuracy. Va notato che nel caso del progetto Openstack si è verificato un leggero miglioramento sia nell'accuracy che nell'MRR. Questo miglioramento è presumibilmente attribuibile all'alta correlazione tra gli owner e i revisori all'interno di tale progetto [16]. Tale associazione stretta tra chi ha creato il progetto e chi lo ha revisionato ha contribuito in modo sostanziale all'aumento dell'accuratezza complessiva e del tasso di recupero medio delle informazioni. I risultati emersi da questa ricerca aprono nuove prospettive e stimolano ulteriori indagini nel campo della raccomandazione dei revisori nel processo di revisione del codice. Esistono interessanti opportunità di studio volte a esplorare ulteriori fattori che potrebbero influire sulle prestazioni del modello proposto. Ad esempio, sarebbe utile esaminare l'impatto delle caratteristiche legate al linguaggio utilizzato nel codice oggetto di revisione. L'analisi di queste specifiche caratteristiche potrebbe rivelare pattern e correlazioni inaspettate, offrendo così nuovi spunti per ottimizzare le raccomandazioni dei revisori. Inoltre, un'altra area di interesse potrebbe concentrarsi sul background lavorativo dei singoli revisori coinvolti nel processo di revisione. Esplorare le esperienze passate, le competenze specifiche e le preferenze dei revisori potrebbe contribuire a sviluppare un modello più personalizzato e adattabile alle esigenze individuali dei team di sviluppo. Questo approccio potrebbe non solo migliorare l'accuratezza delle raccomandazioni, ma anche favorire un ambiente di lavoro più efficiente e collaborativo, dove le competenze specifiche dei revisori vengono sfruttate in modo ottimale.

CAPITOLO 5

Conclusioni e sviluppi futuri

In questo studio, è stata condotta un'analisi approfondita nel campo della Code Review e dei Code Reviewer Recommendation, dimostrando la loro fondamentale importanza nel contesto del processo di sviluppo del software. Durante questa ricerca, sono state esplorate diverse strategie, spaziando dalle tradizionali euristiche basate sull'esperienza all'implementazione avanzata di algoritmi di machine learning. Ciascuna di queste metodologie ha presentato vantaggi e limitazioni specifiche, contribuendo in modo significativo alla comprensione della complessità intrinseca del problema. È stato sviluppato un sistema di Code Reviewer Recommendation basato sull'algoritmo RandomForest, che è stato successivamente addestrato su tre dataset di progetti open-source, comprendenti un totale di 35.239 pull-request. I risultati ottenuti sono stati confrontati con quelli di due modelli preesistenti. L'analisi dei risultati ha evidenziato un notevole miglioramento delle prestazioni rispetto a RevFinder: la top-2 accuracy è aumentata del 33.70% e il Mean Reciprocal Rank medio è risultato superiore del 29.13%. Tuttavia, se confrontato con l'approccio basato su Naive Bayes, il modello proposto mostra prestazioni leggermente inferiori. L'obiettivo di questa studio non era solo quello di progettare e sviluppare un modello di CRR, ma di sottolineare l'importanza di considerare vari fattori e approcci nella progettazione di sistemi di raccomandazione dei revisori, offrendo spunti preziosi

per ricerche future nel campo del Code Reviewers Recommendation. In conclusione, il modello proposto ha conseguito risultati soddisfacenti, rivelando il suo potenziale per futuri miglioramenti. Nei prossimi studi, ci concentreremo sull'ottimizzazione approfondita del modello esistente. Questo processo comprenderà l'esplorazione di nuove feature che potrebbero incrementare ulteriormente la precisione delle raccomandazioni, come le caratteristiche linguistiche del codice oggetto di revisione o il background professionale dei revisori, come già menzionato in precedenza. Parallelamente, intendiamo condurre un'analisi dettagliata degli iperparametri del modello RandomForest, con l'obiettivo di individuare configurazioni ottimali che massimizzino l'efficacia del sistema. Allo stesso tempo, prevediamo di esplorare nuovi approcci e tecniche nell'ambito dell'apprendimento automatico e dell'intelligenza artificiale. Questa ricerca si concentrerà sulla valutazione dell'applicabilità di algoritmi più avanzati, finalizzati a risolvere sfide specifiche nel contesto della CRR.

Bibliografia

[1] Cos'è Random Forest? | IBM. (Citato alle pagine iii e 25)

[2] Cos'è un Albero Decisionale | IBM. (Citato alle pagine iii e 24)

[3] Gerrit code review | gerrit code review. (Citato alle pagine 8 e 17)

[4] How gerrit works. (Citato alle pagine iii e 9)

[5] Take the pain out of code review | review board, 2019. (Citato a pagina 8)

[6] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *ICSE*, pages 712–721. IEEE Computer Society, 2013. (Citato a pagina 4)

[7] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and 3 Sunghun Kim. Duplicate bug reports considered harmful ... really? In *2008 IEEE International Conference on Software Maintenance*, pages 337–345, 2008. (Citato a pagina 26)

[8] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976. (Citato a pagina 5)

[9] M. Frąckiewicz. Algoritmi di Machine Learning: dagli alberi decisionali alle reti neurali profonde. https://ts2.space/it/algoritmi-di-machine-learning-dagli-alberi-decisionali-alle-reti-neurali-profonde/, may 14 2023. (Citato a pagina 14)

[10] Git. Git, 2019. (Citato a pagina 8)

[11] GitHub. Github, 2023. (Citato alle pagine 8 e 17)

[12] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, A. E. Camargo Cruz, Kenji Fujiwara, and Hajimu Iida. Who does what during a code review? datasets of oss peer review repositories. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 49–52, 2013. (Citato alle pagine iii e 5)

[13] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, and Kwangkeun Yi. Improving code review by predicting reviewers and acceptance of patches. 01 2009. (Citato a pagina 26)

[14] Jing Jiang, Jia-Huan He, and Xue-Yuan Chen. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology*, 30:998–1016, 09 2015. (Citato alle pagine 11, 13, 21 e 22)

[15] Ruiyin Li, Peng Liang, and Paris Avgeriou. Code reviewer recommendation for architecture violations: An exploratory study. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, page 42–51, New York, NY, USA, 2023. Association for Computing Machinery. (Citato alle pagine 8, 9 e 28)

[16] Jakub Lipcak and Bruno Rossi. A large-scale study on source code reviewer recommendation. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 378–387, 2018. (Citato alle pagine iii, iv, 2, 10, 11, 14, 15, 17, 18, 25, 26, 27, 29 e 31)

[17] Jakub Lipčák. Optimal recommendations for source code reviews, 2017. (Citato alle pagine iv, 14, 15, 27, 29 e 31)

[18] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. Code reviewing in the trenches: Challenges and best practices. *IEEE Software*, 35(4):34–42, 2018. (Citato a pagina 10)

[19] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. The impact of code review coverage and code review participation on software

quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 192–201, New York, NY, USA, 2014. Association for Computing Machinery. (Citato a pagina 4)

[20] Ba Olshausen. Bayesian probability theory. *The Redwood Center for Theoretical ...*, pages 1–6, 01 2004. (Citato a pagina 14)

[21] Peter C. Rigby and Christian Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 202–212, New York, NY, USA, 2013. ACM. (Citato a pagina 4)

[22] Shaykh Siddique. The factors of code reviewing process to ensure software quality, 07 2021. (Citato alle pagine iv e 7)

[23] Patanamon Thongtanunam, Raula Gaikovina Kula, Ana Erika Camargo Cruz, Norihiro Yoshida, and Hajimu Iida. Improving code review effectiveness through reviewer recommendations. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE 2014, page 119–122, New York, NY, USA, 2014. Association for Computing Machinery. (Citato a pagina 28)

[24] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150, 2015. (Citato alle pagine 2, 10, 11, 21, 27 e 28)

[25] M. Waseem. A Quick Guide To Learn Support Vector Machine In Python. https://www.edureka.co/blog/support-vector-machine-in-python/, aug 8 2023. (Citato alle pagine iii e 13)

[26] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74, 01 2016. (Citato a pagina 11)

[27] H. Alperen Çetin, Emre Doğan, and Eray Tüzün. A review of code reviewer recommendation studies: Challenges and future directions. *Science of Computer Programming*, 208:102652, 2021. (Citato a pagina 10)

Ringraziamenti

Desidero esprimere la mia profonda gratitudine alle persone speciali che hanno reso possibile il completamento di questa tesi.

In primo luogo, ringrazio la mia famiglia, mio padre e mia nonna, per il loro costante sostegno e per i sacrifici che hanno fatto per permettermi di realizzare i miei sogni.

A Davide ed Alfonso, due fratelli, presenti nella mia vita fin dai tempi delle superiori, con voi ho condiviso un bellissimo periodo della mia vita, entrambi siete stati sempre pronti a sostenere le mie scelte e a sopportare nelle mie stupidaggini.

A Caiazzo, grazie per le serate trascorse a piazza Europa, dove abbiamo condiviso chiacchiere, risate e momenti di amicizia. Le partite di Padel che abbiamo giocato insieme sono diventate un'importante pausa dalla routine accademica, riempiendo le nostre giornate di energia positiva e divertimento

A Tony e Paky, ci siamo conosciuti durante il triennio e subito si è creato un forte legame. Il vostro supporto e la vostra amicizia sono stati fondamentali, in particolare Tony, che ha pazientemente risposto alle mie innumerevoli domande, come lui stesso chiama, "abbonate". Senza di voi, probabilmente non avrei mai nemmeno iniziato questo percorso o lo avrei abbandonato nei primi mesi.

Ringrazio il gruppo Napalm, con cui ho condiviso non solo giornate di studio e risate, ma anche momenti di puro divertimento. La vostra compagnia ha reso il mio percorso accademico non solo educativo, ma anche incredibilmente divertente.

Ringrazio tutti voi per le serate indimenticabili trascorse su Discord, giocando e scherzando.