Analisi della Buggyness nei Metodi Software: Un Approccio Data-Driven al Refactoring

Francesco Masci
Studente del Corso di Laurea Magistrale in Ingegneria Informatica
Università degli Studi di Roma "Tor Vergata"
Matricola 0365258, A.A. 2024/2025
francesco.masci.02@students.uniroma2.eu

ABSTRACT - In questo progetto di ingegneria del software viene realizzato un dataset a partire da due progetti open source, con l'obiettivo non solo di raccogliere informazioni sui metodi presenti in diverse release, ma soprattutto di analizzare l'impatto del refactoring sulla qualità del software. Il dataset comprende annotazioni dettagliate sullo stato buggy o non buggy dei metodi, insieme a numerose metriche statiche e dinamiche. Su questa base vengono addestrati e valutati diversi modelli di machine learning per prevedere, all'interno di una specifica release, se un metodo sia incline a presentare bug. L'analisi mira a comprendere se interventi di refactoring, guidati da metriche strutturali, possano contribuire a ridurre la probabilità di introdurre difetti, migliorando così la manutenibilità e la robustezza del codice. I risultati ottenuti mettono in evidenza la possibilità di identificare con buona accuratezza i metodi problematici, offrendo un valido supporto alle decisioni in fase di testing e manutenzione. Il progetto dimostra come approcci data-driven possano favorire strategie di sviluppo più efficaci, orientate alla prevenzione dei bug prima ancora della loro manifestazione.

I. Introduzione

La manutenibilità del software rappresenta una delle sfide più rilevanti nello sviluppo di sistemi complessi e in continua evoluzione. Il codice sorgente, infatti, subisce modifiche frequenti per correggere errori, aggiungere funzionalità o adattarsi a nuovi requisiti, e tali interventi possono influire significativamente sulla qualità complessiva del software. In particolare, la presenza di difetti nei metodi software è un indicatore critico di rischio che può compromettere la stabilità e l'affidabilità del sistema, oltre a incrementare i costi e i tempi di manutenzione. Per questi motivi, comprendere e prevedere la propensione ai difetti a livello di singolo metodo diventa un elemento chiave per supportare processi di sviluppo più efficaci e orientati alla qualità.

Nonostante l'ampio interesse verso la qualità del codice, esiste una carenza di evidenze empiriche solide riguardo all'impatto della riduzione dei cosiddetti "code smells" e altre metriche di qualità sul livello di buggyness dei singoli metodi nelle release software. Gran parte della letteratura si concentra sull'analisi a livello di classi o moduli, mentre la granularità metodologica, cruciale per interventi mirati, è ancora poco esplorata. Questa lacuna limita la possibilità di definire strategie preventive e di ottimizzare le attività di testing focalizzandosi sulle unità di codice più a rischio.

Il presente studio si inserisce in questo contesto con l'obiettivo di colmare tale divario attraverso un'analisi empirica basata su due progetti open source. Viene creato un dataset dettagliato contenente informazioni sulle modifiche e sullo stato buggy dei metodi in più release, da cui vengono estratte caratteristiche significative per la costruzione di modelli predittivi.

Le domande di ricerca fondamentali a cui si vuole rispondere sono:

- quale classificatore è in grado di prevedere con maggiore accuratezza se un metodo sarà buggy in una data release?
- quanti metodi buggy si potrebbero potenzialmente evitare applicando tecniche di predizione efficaci a supporto delle attività di refactoring?

La struttura del lavoro è organizzata come segue: nella Sezione 2 vengono descritti il dataset, le metriche adottate e il protocollo sperimentale utilizzato per l'analisi e la scelta del predittore; nella Sezione 3 sono presentati e discussi i risultati ottenuti; la Sezione 4 approfondisce le implicazioni dei risultati, evidenziando le principali minacce alla validità dello studio; infine, nella Sezione 5 vengono tratte le conclusioni e delineate possibili direzioni per sviluppi futuri.

Il codice utilizzato è disponibile pubblicamente nel repository GitHub [1], così come l'analisi della qualità del codice tramite SonarCloud [2].

II. Misurazioni e metodologia

a) Creazione del dataset

I progetti utilizzati per la costruzione del dataset sono BookKeeper e OpenJPA, entrambi caratterizzati da un numero significativo di release, ottenute tramite il sistema di tracciamento Jira. Per ciascuna release sono stati raccolti tutti i metodi presenti, insieme ad una collezione di metriche statiche e dinamiche. L'assegnazione dell'etichetta "buggy" è avvenuta collegando i commit ai ticket Jira relativi a bug, identificando i metodi coinvolti nelle modifiche. Nei casi in cui i ticket non riportavano esplicitamente le versioni affette, è stata applicata la tecnica Proportion.

Questa tecnica si basa sull'idea che i bug in un progetto software si comportano in modo simile nel tempo, e che la distribuzione delle affected versions e delle fixed versions

per i ticket completi può essere utilizzata per stimare la stessa distribuzione per i ticket incompleti. Per fare ciò, viene utilizzata una moving window di dimensione pari all'1% del totale dei ticket ordinati temporalmente. Questa finestra scorre lungo la sequenza dei ticket e, per ciascun punto, calcola la proporzione tra versioni introduttive e versioni di fix note, estendendo tale proporzione anche ai ticket incompleti all'interno della finestra.

È importante sottolineare che sono state scartate le informazioni sulle affected versions di Jira quando queste erano successive alla versione di apertura del ticket. In questo modo, è stato possibile inferire in modo più completo la distribuzione dei difetti e associare correttamente lo stato buggy ai metodi coinvolti nelle diverse release.

Il dataset finale integra quindi informazioni su metodi, release e un'ampia gamma di metriche relative alla struttura del codice e alla sua evoluzione nel tempo.

b) Metriche utilizzate

Nel progetto sono state utilizzate diverse metriche, mostrate in Tabella 7, per descrivere ogni metodo all'interno di una specifica release, includendo sia metriche strutturali, come linee di codice (LOC), numero di statement, complessità ciclomatica, complessità cognitiva, branch points e nesting depth, sia metriche storiche, come il numero di modifiche e il churn, le linee aggiunte e rimosse sia il valore complessivo, medio e massimo. Queste informazioni sono state estratte a livello di metodo sfruttando strumenti basati su JavaParser, garantendo così un'elevata precisione e granularità.

L'analisi delle correlazioni statistiche tra ciascuna metrica e la variabile target Buggy, condotta utilizzando i coefficienti di Pearson e Spearman, ha messo in evidenza una marcata differenza nella natura dei due progetti.

In BookKeeper, i cui valori sono riportati in Tabella 5, le metriche storiche si sono dimostrate più rilevanti rispetto a quelle strutturali. In particolare, si osservano forti correlazioni negative con la presenza di bug per metriche come MethodHistories (Spearman pari a -0.20) e AvgChurn (Spearman pari a -0.162). Questo suggerisce che metodi frequentemente modificati tendono a stabilizzarsi nel tempo, risultando meno soggetti a introdurre bug, come mostrato anche dalla correlazione di Version (Spearman pari a 0.26).

Nel caso di OpenJPA, i cui valori sono riportati in Tabella 6, invece, le metriche strutturali risultano predominanti. Si osservano, ad esempio, correlazioni positive tra la presenza di bug e Statement (Spearman pari a 0.253) e LOC (Spearman pari a 0.253). Ciò indica che la complessità interna del codice gioca un ruolo cruciale nella determinazione della qualità, e che metodi più lunghi, profondi o complessi sono maggiormente associati a difetti.

c) Protocollo sperimentale

Il protocollo sperimentale adottato si basa su uno schema temporale walk-forward, in cui ogni release viene predetta utilizzando esclusivamente i dati delle versioni precedenti. Questa scelta non solo evita fenomeni di data leakage, ma riflette uno scenario realistico di applicazione pratica, in cui un modello predittivo viene impiegato per supportare decisioni su nuove versioni del software.

L'efficacia dei modelli è stata valutata attraverso diverse metriche. L'AUC (Area Under the Curve) misura la capacità del classificatore di distinguere correttamente metodi buggy da quelli non buggy, mentre la Kappa valuta la qualità della classificazione rispetto a una scelta casuale. La precision rappresenta la proporzione di metodi realmente buggy tra quelli predetti come tali, e il recall indica la proporzione di metodi buggy effettivamente identificati. Dalla loro combinazione deriva l'F1-score, che bilancia precision e recall in un unico valore sintetico. L'accuracy esprime invece la proporzione complessiva di classificazioni corrette, sebbene possa risultare meno informativa in presenza di dataset sbilanciati. Infine, la NPofB20 indica quanti difetti reali vengono individuati considerando solo il 20% del codice classificato come più rischioso, misurando quindi l'utilità del modello nel guidare le ispezioni.

Per quanto riguarda i modelli predittivi, sono stati valutati Naive Bayes, Random Forest e IBk (k-Nearest Neighbors).

Per la selezione delle feature è stato utilizzato l'information gain con soglia pari a 0.01, ottenendo una riduzione consistente della dimensionalità: da 22 a 13 feature in BookKeeper e da 22 a 18 in OpenJPA. Tale riduzione non ha compromesso le prestazioni complessive: i modelli Random Forest, ad esempio, con feature selection hanno mantenuto valori di AUC e F1-score comparabili a quelli con tutte le feature, con benefici in termini di semplificazione del modello e riduzione della complessità computazionale.

Figura 1 - Distribuzione dell'AUC per BookKeeper tramite Box Plot. Nella prima colonna sono mostrati i risultati per Random Forest, nella seconda colonna quelli di Naive Bayes e infine nell'ultima colonna quelli di IBK

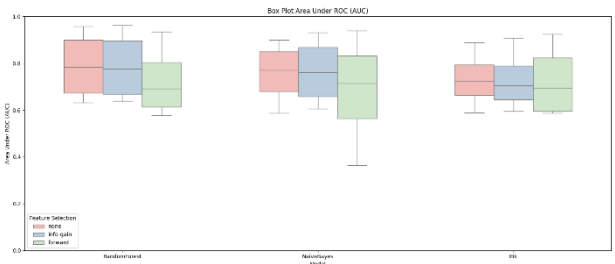
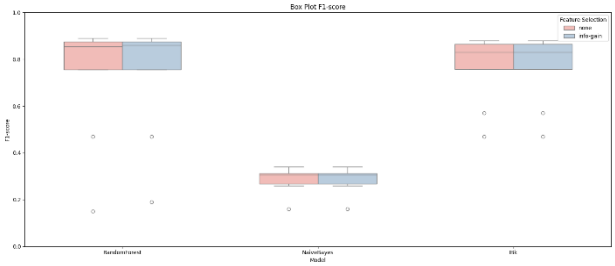


Figura 2 - Distribuzione di F1 per OpenJPA tramite Box Plot. Nella prima colonna sono mostrati i risultati per Random Forest, nella seconda colonna quelli di Naive Bayes e infine nell'ultima colonna quelli di IBK



I risultati, riportati in Figura 3, Figura 4, Figura 5, Figura 6, Figura 7, Figura 8, Figura 9, Figura 10 e Figura

11 e in forma numerica in Tabella 8 e Tabella 9, mostrano inoltre come la variabilità rimanga contenuta, confermando la robustezza dell'approccio.

Sono state testate anche strategie alternative di selezione, come la forward search e la backward elimination. Tuttavia, la backward elimination si è rivelata impraticabile per l'elevato costo computazionale, mentre la forward search, pur riducendo ulteriormente il numero di feature, ha prodotto modelli instabili, con metriche sensibilmente inferiori a quelle dell'information gain, come evidenziato nei box plot di precision e recall.

Naive Bayes ha mostrato complessivamente i risultati peggiori, con recall e precision più bassi rispetto agli altri due modelli, rendendolo meno affidabile per la predizione della buggyness. IBk ha ottenuto risultati comparabili, ma non è stato scelto come modello finale poiché la Random Forest si è dimostrata più robusta e stabile rispetto alle variazioni dei dati e offre migliori capacità di generalizzazione, risultando più adatta per applicazioni predittive su dataset complessi.

In sintesi, dunque, i risultati indicano che Random Forest rappresenta il modello più equilibrato e robusto, capace di garantire un buon compromesso tra accuratezza, stabilità e interpretabilità, soprattutto se combinato con la selezione delle feature tramite information gain.

d) Assunzioni

Nell'analisi si è assunta l'indipendenza tra le coppie metodo–release, trattandole come unità distinte, anche se in realtà possono esistere dipendenze temporali tra versioni consecutive. La scelta di limitare la sperimentazione a tre classificatori noti è stata motivata dall'obiettivo di focalizzarsi su modelli ampiamente studiati. È stato considerato che, qualora un metodo non fosse stato modificato in una release, le sue metriche statiche (come LOC e numero di statement) restassero invariate rispetto alla release precedente. Il refactoring manuale rilevante è stato eseguito da un singolo sviluppatore; pertanto, cambiamenti stilistici minori e non funzionali sono stati ignorati nel processo di analisi. Poiché non sono disponibili strumenti di analisi statica specifici a livello di metodo, sono stati implementati estrattori di metriche personalizzati basati sul parsing del codice sorgente con l'ausilio di JavaParser. Questa soluzione ha consentito di calcolare in modo affidabile le metriche di complessità e struttura a livello metodologico, riducendo potenziali errori di attribuzione e garantendo coerenza nell'estrazione delle feature. Tutti gli esperimenti sono stati eseguiti con seed fisso, garantendo la piena riproducibilità dei risultati.

III. Risultati

Questo capitolo presenta i risultati dell'analisi dei progetti BookKeeper e OpenJPA, con l'obiettivo di individuare i metodi candidati al refactoring attraverso lo studio della correlazione tra caratteristiche del codice e presenza di difetti, e di valutare l'efficacia di tale scelta.

a) Selezione razionale del metodo target

L'analisi della correlazione tra le feature raccolte e la bugginess ha guidato la selezione delle feature actionable da usare come criterio prioritario di refactoring. Per il progetto BookKeeper, la metrica con maggiore correlazione assoluta è risultata essere la complessità ciclomatica ($\rho = 0.059$ Spearman). Sebbene di valore basso, è stata comunque scelta in quanto metrica interpretabile e direttamente mitigabile tramite refactoring. È stato quindi selezionato il metodo con massima complessità ciclomatica nella release più recente, ovvero processPacket nella classe BookieServer, versione 4.2.1, con un valore di 34.

Per OpenJPA, invece, la metrica con la più alta correlazione è risultata il numero di statement ($\rho = 0.253$ Spearman). Anche in questo caso, si tratta di una caratteristica significativa e correggibile. È stato dunque selezionato il metodo eval della classe JPQLExpressionBuilder, versione 1.2.0, con 175 statement.

Questi due metodi risultano buoni candidati al refactoring: processPacket è al centro del flusso di comunicazione tra client e bookie in BookKeeper, per cui ha alta criticità architetturale e impatto sul core del sistema; eval in OpenJPA rappresenta la logica di parsing e valutazione delle query JPQL, ed è pertanto centrale per l'intero sistema di persistenza.

b) Descrizione del metodo selezionato

Il metodo processPacket in BookieServer si occupa di gestire pacchetti di richiesta provenienti dal client, instradandoli a seconda del tipo (read, write, addEntry). Il metodo contiene 140 righe di codice e 118 statement, con una complessità ciclomatica pari a 34 e una profondità di annidamento di 10. Le sue caratteristiche suggeriscono elevata ramificazione logica, confermata dalla presenza di 33 branch points e una complessità cognitiva di 98. Non è presente una storia di modifiche rilevante (1 modifica nella release corrente) e le operazioni di churn sono nulle, indicando una recente stabilità. Tuttavia, la struttura attuale suggerisce code smells come "Long Method", "God Method" e "Feature Envy".

Per OpenJPA, il metodo eval della classe JPQLExpressionBuilder ha 261 linee di codice, 175 statement, una complessità ciclomatica di 82 e una cognitiva di 171. Il metodo non ha subito modifiche recenti ma presenta una struttura fortemente complessa, probabilmente associata a un singolo costrutto monolitico per la valutazione delle query. Anche in questo caso si osservano code smells simili, soprattutto "Long Method" e "Complex Conditional".

Questi dati confermano la complessità strutturale dei due metodi selezionati e giustificano la scelta di intervenire su di essi.

Tabella 1 - Metriche dei metodi antecedenti al refactoring

Project	BOOKKEEPER	OPENJPA
Package	org.apache.BookKeeper.proto	org.apache.openjpa.kernel.jpql
Class	BookieServer	JPQLExpressionBuilder

Method	processPacket	eval
Version	4.2.1	1.2.0
LOC	140	261
Statement	118	175
Cyclomatic	34	82
Cognitive	98	171
MethodHistories	1	0
AddedLines	5	0
MaxAddedLines	5	0
AvgAddedLines	5	0
DeletedLines	0	0
MaxDeletedLines	0	0
AvgDeletedLines	0	0
Churn	5	0
MaxChurn	5	0
AvgChurn	5	0
BranchPoints	33	81
NestingDepth	10	5
ParametersCount	2	1
Buggy	false	false

c) Criteri per identificare i target del refactoring

L'individuazione dei target di refactoring è stata guidata dalla necessità di affrontare code smells quali metodi troppo lunghi e complessi, che indicavano l'opportunità di applicare la tecnica di Extract Method. Questa scelta ha permesso di frammentare il codice monolitico in componenti più piccoli e semanticamente coerenti, riducendo così la complessità e facilitando la manutenzione.

La regola di prioritizzazione ha previsto la riduzione della metrica principale — complessità ciclomatica per processPacket e numero di statement per eval — senza aumentare le altre metriche correlate positivamente alla bugginess, come la complessità cognitiva, i branch points o la profondità di annidamento. Questa strategia ha assicurato un miglioramento complessivo della qualità del codice, applicando l'estrazione di metodi solo quando si poteva garantire un beneficio misurabile e sostenibile.

d) Risultati del refactoring

Il refactoring ha interessato entrambi i metodi con l'estrazione di sottocomponenti funzionali, portando a una drastica riduzione delle metriche critiche. Per processPacket, il metodo originario di 140 righe, 118 statement e complessità ciclomatica 34 è stato suddiviso in tre metodi. Il metodo principale è ora ridotto a 30 righe con complessità ciclomatica pari a 7, mentre i metodi handleReadEntry e handleAddEntry contano rispettivamente 78 e 23 righe. Per eval, originariamente con 261 righe e 175 statement, il refactoring ha prodotto quattro metodi più piccoli, con il principale ridotto a 37 righe e 13 statement, mentre la complessità ciclomatica è passata da 82 a 24. Le metriche mostrano nel complesso una riduzione significativa, con la complessità ciclomatica di processPacket diminuita di circa il 79% e il numero di statement di eval ridotto di oltre il 90%.

La compilazione e i test unitari hanno confermato la corretta funzionalità del codice dopo il refactoring. Nel caso di OpenJPA, durante la fase di testing sono stati rilevati alcuni errori, tuttavia si è verificato che tali errori erano già presenti prima del refactoring e non coinvolgevano il metodo oggetto dell'intervento. Probabilmente queste problematiche sono dovute a incompatibilità tra le versioni di Java utilizzate, dato che il progetto è strutturato per Java 5, una versione ormai obsoleta e difficilmente installabile nei moderni ambienti di sviluppo.

Tabella 2 - Metriche dei metodi a seguito del refactoring

Project	BOOKKEEPER	OPENJPA
Package	org.apache.BookKeeper.proto	org.apache.openjpa.kernel.jpql
Class	BookieServer	JPQLExpressionBuilder
Method	processPacket	eval
Version	4.2.1	1.2.0
LOC	30	37
Statement	25	13
Cyclomatic	7	24
Cognitive	14	49
MethodHistories	1	1
AddedLines	41	23
MaxAddedLines	41	23
AvgAddedLines	41	23
DeletedLines	43	29
MaxDeletedLines	43	29
AvgDeletedLines	43	29
Churn	84	52
MaxChurn	84	52
AvgChurn	84	52
BranchPoints	6	23
NestingDepth	3	2
ParametersCount	2	1
Buggy	false	false

e) Analisi What-if dei Metodi buggy evitabili

Per valutare l'impatto di interventi mirati sui metodi difettosi, è stata condotta una simulazione what-if con i modelli precedentemente addestrati. L'esperimento consiste nell'azzerare le feature actionable individuate tramite l'analisi di correlazione (ad esempio l'elevata complessità ciclomatica), così da stimare quanti difetti potrebbero essere evitati. In questo modo è possibile quantificare il potenziale miglioramento della manutenibilità del software in termini di riduzione dei difetti previsti dal modello.

Nel caso di BookKeeper, come mostrato nella Tabella 3 il dataset completo (A) contiene 7132 metodi effettivamente buggy, e il classificatore ne predice altrettanti. Il sottoinsieme B+ comprende i 7032 metodi in cui la feature actionable è presente; su questo insieme il modello predice esattamente 7032 metodi buggy. Applicando la simulazione what-if, ovvero ponendo a zero la feature actionable (dataset B) nel dataset B+, il numero di metodi predetti come buggy si riduce a 6985. Ciò significa che 47 metodi non sarebbero più classificati come buggy in presenza della modifica, corrispondenti al 0.67% di riduzione su B+ e al 0.66% rispetto all'intero dataset A. In termini pratici, questa simulazione suggerisce che, se la feature actionable venisse rimossa da tutti i metodi in cui è

presente, si potrebbero potenzialmente evitare 47 metodi buggy secondo la previsione del modello.

Tabella 3 - Esiti dell'analisi What-If sul progetto BookKeeper

Dataset	Presenti	Predetti
A	7132	7132
B+	7032	7032
C	100	100
B	-	6985

Nel caso di OpenJPA, i risultati dell'analisi what-if in Tabella 4 evidenziano un impatto più marcato. Il dataset A contiene 14.912 metodi effettivamente buggy, mentre il modello ne predice 15.511, con una sovrastima pari a 599 metodi, cioè un errore relativo del +4.02%. Il sottoinsieme B+, che rappresenta i 14.334 metodi in cui è presente almeno una feature actionable, contiene 14.938 metodi predetti come buggy. Dopo aver azzerato le feature actionable (dataset B) nel dataset B+, il numero di metodi buggy predetti scende a 14.452, con una riduzione di 486 metodi, pari al 3.25% rispetto alle previsioni iniziali su B+.

Questa riduzione si riflette anche rispetto al dataset A: i 486 metodi evitati equivalgono al 3.26% dei bug effettivi presenti in A (486 su 14.912) e al 3.13% dei bug predetti su A (486 su 15.511). Inoltre, confrontando i risultati su B con quelli su B+, si nota che su 14.334 metodi reali buggy, la previsione si riduce da 14.938 a 14.452, il che implica che il modello passa da sovrastimare di 604 metodi (+4.21%) a sovrastimare di solo 118 metodi (+0.82%). Questo dimostra che l'azzeramento della feature actionable, pur non cambiando i dati reali, riduce significativamente la tendenza del modello a segnalare falsi positivi. Complessivamente, se fosse possibile rimuovere proattivamente tale feature tramite refactoring, si potrebbero potenzialmente evitare fino a 486 metodi buggy, contribuendo sia alla riduzione del numero assoluto di metodi problematici che al miglioramento dell'affidabilità delle previsioni del classificatore.

Tabella 4 - Esiti dell'analisi What-If sul progetto OpenJPA

Dataset	Presenti	Predetti	Errore
A	14912	15511	+599 (+4.02%)
B+	14334	14938	+604 (+4.21%)
C	578	573	-5 (-0.86%)
B	-	14452	-

Questa analisi si basa sull'assunzione di indipendenza condizionata, ovvero si assume che la modifica delle feature actionable non alteri in modo significativo la distribuzione delle altre variabili. In pratica, si ipotizza che la riduzione della complessità non comporti effetti collaterali negativi su altri aspetti del metodo. Sebbene tale assunzione sia idealizzata, permette di ottenere una stima conservativa del potenziale beneficio: i risultati indicano che, con semplici refactoring mirati alla riduzione di specifici fattori strutturali, sarebbe possibile evitare fino a 486 metodi buggy in OpenJPA e 47 in BookKeeper secondo le previsioni del classificatore.

IV. Discussioni e minacce

I risultati ottenuti permettono di rispondere in modo diretto alle due domande di ricerca.

Per la RQ1, il classificatore più efficace è risultato essere Random Forest, che ha ottenuto le migliori prestazioni in termini di AUC e precision, dimostrandosi il più adatto a identificare metodi buggy sia in BookKeeper che in OpenJPA. Il suo comportamento stabile e la capacità di gestire feature rumore confermano l'efficacia di modelli ensemble in contesti ad alta dimensionalità.

Per quanto riguarda la RQ2, l'analisi what-if mostra che riducendo a zero le feature più correlate ai difetti si otterrebbe una riduzione attesa di 47 metodi buggy in BookKeeper e 486 in OpenJPA. Questo indica un miglioramento più marcato della manutenibilità per quest'ultimo progetto, dove le correlazioni tra qualità interna e presenza di bug sono più forti. BookKeeper, al contrario, presenta una struttura meno sensibile alle metriche di qualità, suggerendo una maggiore robustezza o una diversa origine dei difetti.

L'analisi è soggetta a minacce alla validità interna, tra cui errori nella misura automatica delle metriche e assunzioni semplificative nell'esperimento what-if, che presume indipendenza tra le feature e la possibilità di portare a 0 qualsiasi feature actionable. Inoltre, la selezione delle metriche modificabili può essere influenzata da bias soggettivi. Nonostante questi limiti, i risultati forniscono evidenza concreta del legame tra metriche riducibili e difettosità, sostenendo l'uso di approcci predittivi per guidare il refactoring.

V. Conclusioni e lavori futuri

Questo studio ha esplorato la possibilità di identificare metodi buggy all'interno di due progetti Java open source attraverso l'analisi di metriche statiche e la costruzione di modelli predittivi. Il risultato principale è che il classificatore Random Forest ha mostrato la maggiore efficacia, e che la presenza di feature correlabili ha un impatto significativo sulla probabilità di difettosità. Questi risultati suggeriscono che un'attenzione mirata alla riduzione delle feature problematiche può portare a un miglioramento concreto della manutenibilità del software.

Prossimi passi futuri potrebbero includere:

- analisi comparativa su progetti con architetture non object-oriented (es. funzionali o ibridi),
- integrazione delle metriche con dati runtime e di performance per rafforzare la predizione,
- sviluppo di strumenti interattivi per ingegneri del software che evidenzino metodi critici in tempo reale durante la codifica.

Riferimenti

[1] F. Masci, «GitHub repository containing the project code,» 2025. [Online]. Available: https://github.com/F-masci/buggyness-predictor.

[2] F. Masci, «SonarCloud analysis of the project code,» 2025. [Online]. Available: https://sonarcloud.io/project/overview?id=F-masci_isw2-prediction.

[3] B. Vandehei, D. A. Da Costa e D. Falessi, «Leveraging the Defects Life Cycle to Label Affected Versions,» *ACM Transactions on Software Engineering and Methodology,* vol. 30, p. 35, 2021.

[4] D. Falessi, S. Mesiano Laureani, J. Çarka, M. Esposito e D. A. Da Costa, «Enhancing the defectiveness prediction of methods and classes via JIT,» *Empirical Software Engineering,* vol. 28, n. 37, p. 43, 2023.

[5] J. Çarka, D. Falessi e M. Esposito, «On effort-awaremetrics for defect prediction,» *Empirical Software Engineering,* vol. 27, n. 152, p. 38, 2022.

[6] D. Falessi, L. Narayana, J. F. Thai e B. Turhan, «Preserving Order of Data When Validating Defect Prediction Models,» p. 20.

[7] D. Falessi, B. Russo e M. Kathleen, «What if I had no smells?,» *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement,* p. 17, 2017.

Appendice

Tabella 5 – Correlazione delle feature in BookKeeper

Attributo	Pearson	Spearman
Version	0,275376	0,259299
Cyclomatic	0,038954	0,059774
LOC	0,026111	0,049495
Statement	0,023831	0,049024
NestingDepth	0,008952	0,040925
BranchPoints	0,029815	0,034353
Cognitive	0,023181	0,032887
Project	0	0
ParametersCount	0,024358	-0,03285
MaxDeletedLines	0,015392	-0,04067
DeletedLines	0,01182	-0,04114
AvgDeletedLines	0,041126	-0,04506
Package	0,142781	-0,05193
Class	0,035863	-0,05622
Method	0,013903	-0,07425
AvgAddedLines	0,127379	-0,15491
AddedLines	0,019363	-0,15898
MaxChurn	0,010166	-0,1595
MaxAddedLines	0,027326	-0,15985
Churn	0,009939	-0,16039
AvgChurn	0,129992	-0,16192
MethodHistories	0,066044	-0,20177

Tabella 6 – Correlazione delle feature in OpenJPA

Attributo	Pearson	Spearman
Statement	0,246323	0,253107
LOC	0,240339	0,252977
NestingDepth	0,25947	0,240619
BranchPoints	0,21581	0,232049
Cognitive	0,175891	0,231096
Cyclomatic	0,216127	0,228938
MethodHistories	0,178689	0,193495
AvgAddedLines	0,08858	0,122974

ParametersCount	0,116932	0,120317
MaxDeletedLines	0,081415	0,120098
DeletedLines	0,078781	0,119689
MaxChurn	0,138471	0,112985
Churn	0,117077	0,112401
AddedLines	0,12447	0,109376
MaxAddedLines	0,139279	0,109153
AvgChurn	0,089277	0,106131
AvgDeletedLines	0,113773	0,071031
Package	0,04389	0,068831
Method	0,00839	0,045839
Project	0	0
Class	0,018911	-0,01528
Version	0,017677	-0,05573

Tabella 7 - Metriche utilizzate per la costruzione del dataset

Codice	**Nome**	**Significato**
Project	Progetto	Nome del progetto software
Package	Pacchetto	Nome del package in cui il metodo è contenuto
Class	Classe	Nome della classe contenente il metodo
Method	Metodo	Nome del metodo analizzato
Version	Versione	Release software di riferimento
LOC	Lines of Code	Numero di linee di codice nel metodo
Statement	Statement	Numero di istruzioni eseguibili nel metodo
Cyclomatic	Complessità Ciclomatica	Misura della complessità logica del metodo (numero di percorsi indipendenti)
Cognitive	Complessità Cognitiva	Stima della difficoltà cognitiva nella comprensione del metodo
MethodHistories	Storico Metodo	Numero di modifiche storiche subite dal metodo
AddedLines	Righe Aggiunte	Numero di righe di codice aggiunte nel metodo rispetto alla release precedente
MaxAddedLines	Max Righe Aggiunte	Massimo numero di righe aggiunte in una singola modifica
AvgAddedLines	Media Righe Aggiunte	Media delle righe aggiunte per modifica
DeletedLines	Righe Eliminate	Numero di righe di codice rimosse dal metodo
MaxDeletedLines	Max Righe Eliminate	Massimo numero di righe eliminate in una singola modifica
AvgDeletedLines	Media Righe Eliminate	Media delle righe eliminate per modifica
Churn	Churn	Somma di righe aggiunte e eliminate (modifiche totali)
MaxChurn	Max Churn	Massimo churn registrato in una singola modifica
AvgChurn	Media Churn	Media del churn per modifica
BranchPoints	Punti di Branch	Numero di punti di decisione (if, switch, etc.) nel metodo
NestingDepth	Profondità di Annidamento	Massimo livello di annidamento di strutture di controllo
ParametersCount	Numero di Parametri	Numero di parametri formali del metodo

Tabella 8 - Prestazioni dei modelli predittivi su BookKeeper

Model	Feature Selection	Features Number	Fold	Precision	Recall	F1-score	AUC	Kappa	Accuracy	NPofB20
RandomForest	none	22	1	0.81	0.72	0.76	0.88	0.65	0.85	0.55
RandomForest	none	22	2	0.02	0.45	0.03	0.69	0.01	0.70	0.23
RandomForest	none	22	3	0.96	0.05	0.10	0.63	0.04	0.42	0.27
RandomForest	none	22	4	1.00	0.53	0.69	0.96	0.42	0.68	0.30
NaiveBayes	none	22	1	0.61	0.19	0.29	0.83	0.15	0.68	0.43
NaiveBayes	none	22	2	0.03	0.50	0.05	0.71	0.03	0.79	0.45
NaiveBayes	none	22	3	0.63	0.02	0.05	0.59	0.00	0.39	0.26
NaiveBayes	none	22	4	0.77	0.04	0.08	0.90	0.01	0.35	0.28
IBk	none	22	1	0.61	0.72	0.66	0.76	0.46	0.75	0.39
IBk	none	22	2	0.02	0.73	0.04	0.69	0.02	0.67	0.39
IBk	none	22	3	0.89	0.27	0.41	0.59	0.18	0.53	0.28
IBk	none	22	4	0.99	0.56	0.72	0.89	0.45	0.70	0.30
RandomForest	info-gain	13	1	0.83	0.67	0.74	0.87	0.63	0.84	0.56
RandomForest	info-gain	13	2	0.01	0.39	0.03	0.68	0.01	0.70	0.23
RandomForest	info-gain	13	3	0.95	0.08	0.14	0.64	0.06	0.43	0.27
RandomForest	info-gain	13	4	1.00	0.51	0.68	0.96	0.40	0.67	0.30

NaiveBayes	info-gain	13	1	0.84	0.42	0.56	0.85	0.43	0.77	0.50
NaiveBayes	info-gain	13	2	0.02	0.43	0.03	0.68	0.01	0.73	0.39
NaiveBayes	info-gain	13	3	0.76	0.01	0.02	0.61	0.00	0.39	0.29
NaiveBayes	info-gain	13	4	0.98	0.43	0.59	0.93	0.32	0.61	0.29
IBk	info-gain	13	1	0.60	0.66	0.63	0.75	0.42	0.73	0.38
IBk	info-gain	13	2	0.02	0.68	0.04	0.66	0.02	0.66	0.36
IBk	info-gain	13	3	0.89	0.28	0.43	0.59	0.19	0.54	0.29
IBk	info-gain	13	4	0.99	0.55	0.71	0.91	0.44	0.69	0.30
RandomForest	forward	5	1	0.64	0.57	0.60	0.76	0.41	0.74	0.41
RandomForest	forward	5	2	0.01	0.45	0.02	0.58	0.00	0.61	0.30
RandomForest	forward	5	3	0.00	0.00	0.00	0.62	0.00	0.38	0.32
RandomForest	forward	5	4	1.00	0.03	0.06	0.93	0.02	0.35	0.29
NaiveBayes	forward	4	1	0.64	0.74	0.69	0.80	0.50	0.77	0.45
NaiveBayes	forward	4	2	0.00	0.23	0.01	0.36	-0.01	0.49	0.09
NaiveBayes	forward	4	3	0.88	0.50	0.64	0.63	0.35	0.65	0.31
NaiveBayes	forward	4	4	0.97	0.77	0.86	0.94	0.65	0.83	0.29
IBk	forward	5	1	0.68	0.66	0.67	0.79	0.50	0.78	0.43
IBk	forward	5	2	0.02	0.59	0.03	0.60	0.01	0.60	0.25
IBk	forward	5	3	0.95	0.28	0.44	0.59	0.21	0.55	0.30
IBk	forward	5	4	0.99	0.53	0.69	0.92	0.41	0.68	0.30

Tabella 9 - Prestazioni dei modelli predittivi su OpenJPA

Model	Feature Selection	Features Number	Fold	Precision	Recall	F1-score	AUC	Kappa	Accuracy	NPofB20
RandomForest	none	22	1	0.89	0.08	0.15	0.96	0.13	0.86	0.91
RandomForest	none	22	2	0.86	0.86	0.86	0.98	0.84	0.96	0.97
RandomForest	none	22	3	0.91	0.79	0.85	0.98	0.82	0.96	0.93
RandomForest	none	22	4	0.83	0.92	0.87	0.99	0.85	0.97	0.99
RandomForest	none	22	5	0.93	0.85	0.89	0.99	0.88	0.97	0.97
RandomForest	none	22	6	0.84	0.86	0.85	0.97	0.83	0.96	0.93
RandomForest	none	22	7	0.87	0.91	0.89	0.99	0.88	0.97	0.96
RandomForest	none	22	8	0.33	0.8	0.47	0.93	0.43	0.91	0.96
NaiveBayes	none	22	1	0.55	0.22	0.31	0.87	0.25	0.86	0.62
NaiveBayes	none	22	2	0.48	0.26	0.34	0.9	0.27	0.87	0.72
NaiveBayes	none	22	3	0.47	0.25	0.32	0.89	0.25	0.85	0.7
NaiveBayes	none	22	4	0.43	0.24	0.31	0.9	0.25	0.87	0.73
NaiveBayes	none	22	5	0.44	0.23	0.3	0.89	0.24	0.87	0.71
NaiveBayes	none	22	6	0.37	0.2	0.26	0.87	0.19	0.87	0.69
NaiveBayes	none	22	7	0.39	0.2	0.27	0.89	0.2	0.87	0.75
NaiveBayes	none	22	8	0.15	0.17	0.16	0.85	0.11	0.91	0.7
IBk	none	22	1	0.84	0.43	0.57	0.68	0.52	0.9	0.51
IBk	none	22	2	0.79	0.86	0.82	0.91	0.8	0.95	0.87
IBk	none	22	3	0.87	0.77	0.82	0.91	0.79	0.95	0.84
IBk	none	22	4	0.82	0.91	0.86	0.97	0.84	0.97	0.97
IBk	none	22	5	0.92	0.85	0.88	0.96	0.86	0.97	0.92
IBk	none	22	6	0.83	0.86	0.84	0.94	0.82	0.96	0.93

IBk	none	22	7	0.85	0.91	0.88	0.98	0.87	0.97	0.96
IBk	none	22	8	0.33	0.79	0.47	0.91	0.43	0.91	0.95
RandomForest	info-gain	18	1	0.91	0.1	0.19	0.96	0.16	0.86	0.91
RandomForest	info-gain	18	2	0.85	0.88	0.87	0.98	0.85	0.96	0.97
RandomForest	info-gain	18	3	0.89	0.81	0.85	0.98	0.83	0.96	0.95
RandomForest	info-gain	18	4	0.82	0.92	0.87	0.99	0.85	0.97	0.99
RandomForest	info-gain	18	5	0.93	0.86	0.89	0.99	0.88	0.97	0.97
RandomForest	info-gain	18	6	0.83	0.87	0.85	0.97	0.83	0.96	0.94
RandomForest	info-gain	18	7	0.87	0.91	0.89	0.98	0.88	0.97	0.97
RandomForest	info-gain	18	8	0.33	0.82	0.47	0.93	0.43	0.91	0.97
NaiveBayes	info-gain	18	1	0.55	0.22	0.31	0.87	0.25	0.86	0.62
NaiveBayes	info-gain	18	2	0.48	0.26	0.34	0.9	0.27	0.87	0.72
NaiveBayes	info-gain	18	3	0.47	0.25	0.32	0.89	0.25	0.85	0.7
NaiveBayes	info-gain	18	4	0.43	0.24	0.31	0.9	0.25	0.87	0.73
NaiveBayes	info-gain	18	5	0.44	0.22	0.3	0.89	0.23	0.87	0.71
NaiveBayes	info-gain	18	6	0.37	0.19	0.26	0.87	0.19	0.87	0.69
NaiveBayes	info-gain	18	7	0.39	0.2	0.27	0.89	0.2	0.87	0.75
NaiveBayes	info-gain	18	8	0.15	0.17	0.16	0.85	0.11	0.91	0.7
IBk	info-gain	18	1	0.84	0.44	0.57	0.68	0.52	0.9	0.51
IBk	info-gain	18	2	0.79	0.86	0.82	0.91	0.8	0.95	0.87
IBk	info-gain	18	3	0.87	0.77	0.82	0.91	0.79	0.95	0.84
IBk	info-gain	18	4	0.82	0.91	0.86	0.97	0.84	0.97	0.97
IBk	info-gain	18	5	0.92	0.85	0.88	0.96	0.87	0.97	0.92
IBk	info-gain	18	6	0.83	0.86	0.84	0.94	0.82	0.96	0.93
IBk	info-gain	18	7	0.85	0.91	0.88	0.98	0.87	0.97	0.96
IBk	info-gain	18	8	0.33	0.79	0.47	0.91	0.43	0.91	0.95

Figura 3 - Distribuzione delle prestazioni dei modelli predittivi su BookKeeper

Figura 4 - Distribuzione delle prestazioni dei modelli predittivi su BookKeeper addestrati senza featuire selection

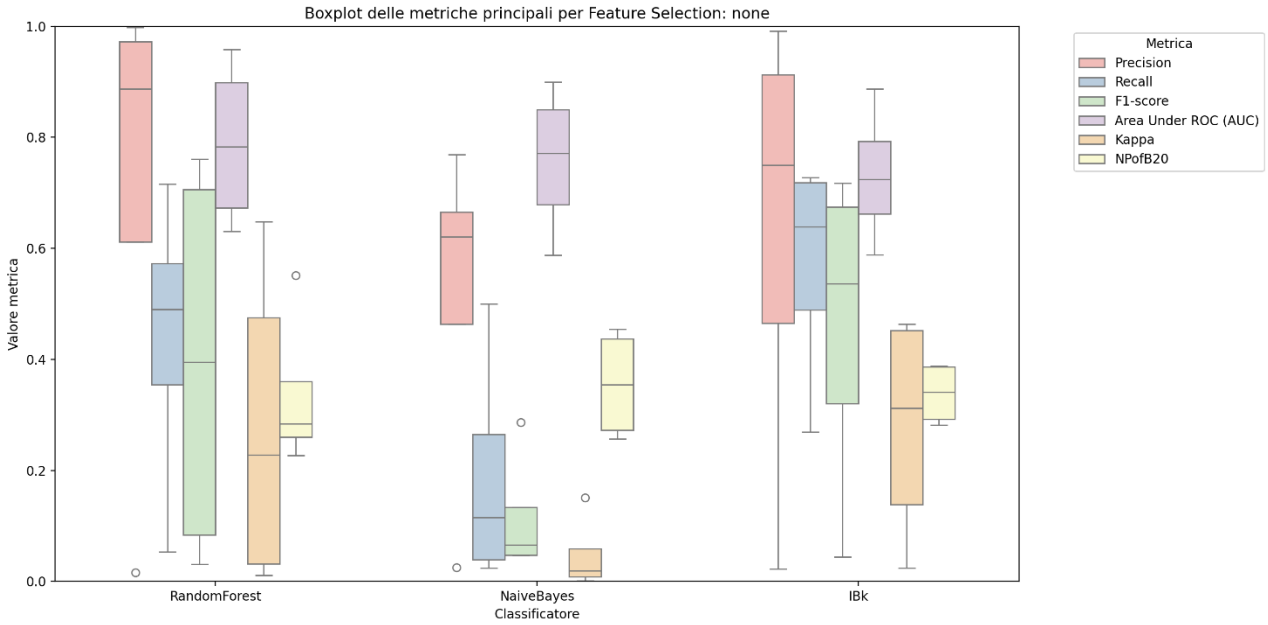


Figura 5 - Distribuzione delle prestazioni dei modelli predittivi su BookKeeper addestrati con information gain

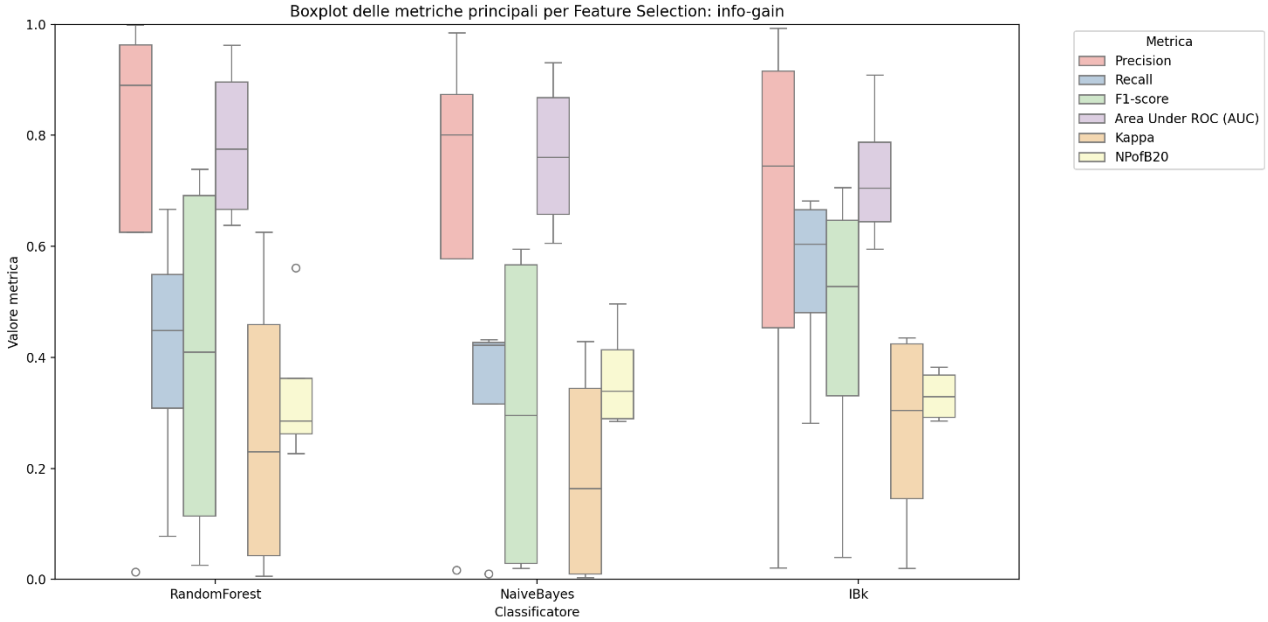


Figura 6 - Distribuzione delle prestazioni dei modelli predittivi su BookKeeper addestrati con forward search

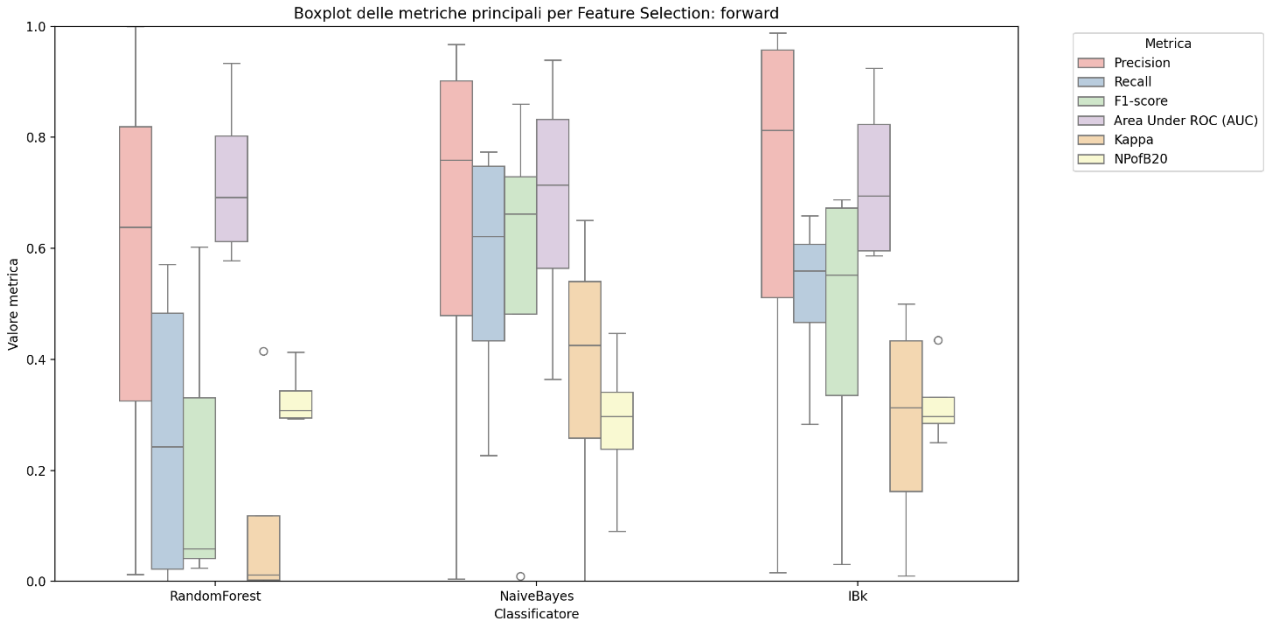


Figura 7 - Valori medi delle metriche prestazionali su BookKeeper con evidenziati i valori massi

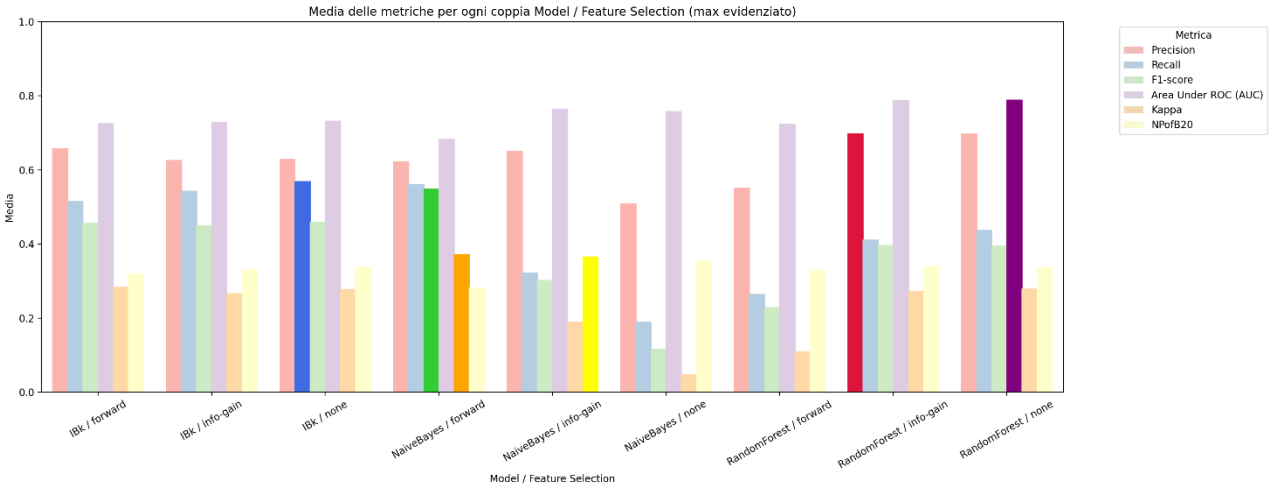


Figura 8 - Distribuzione delle prestazioni dei modelli predittivi su OpenJPA

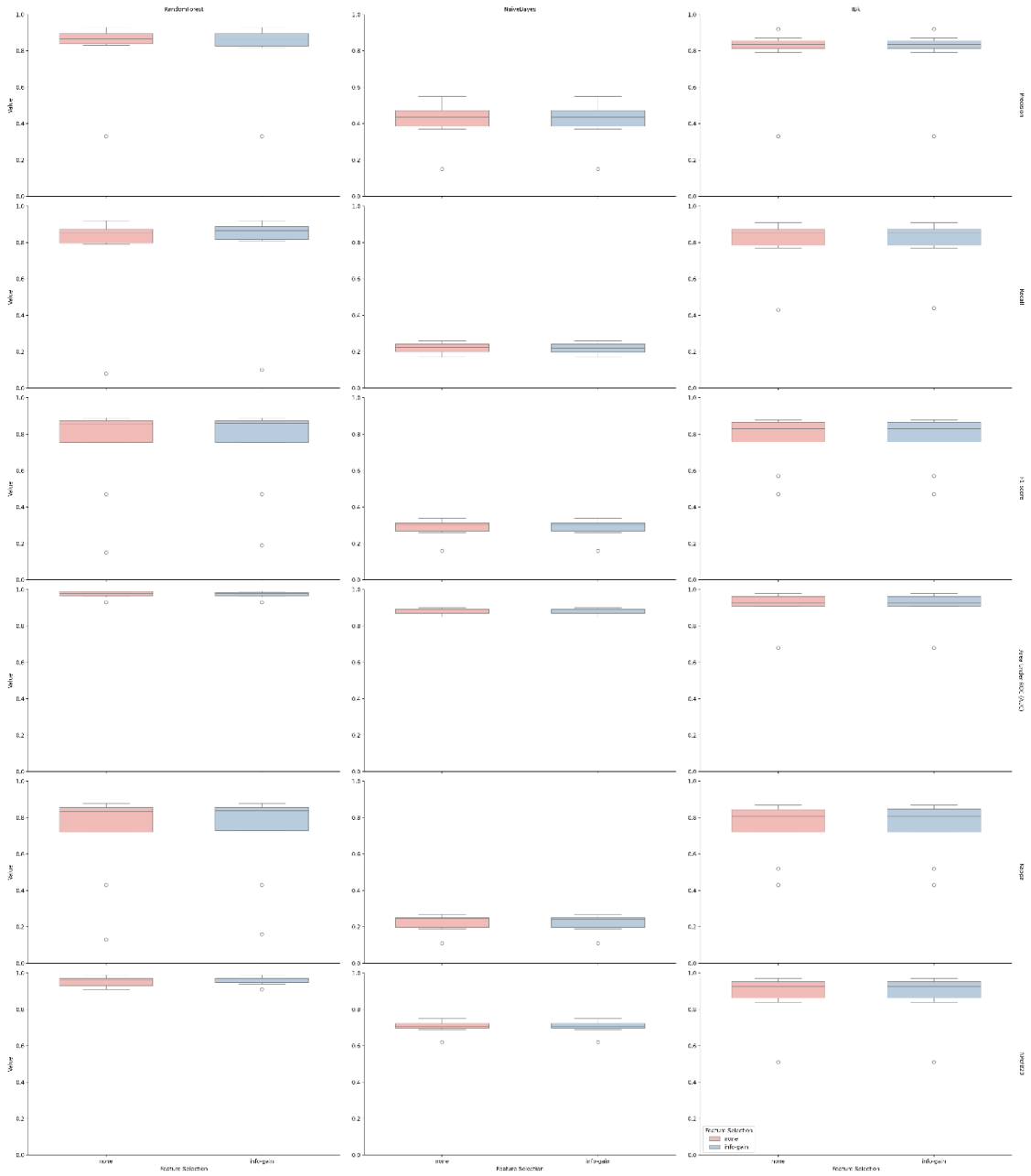


Figura 9 - Distribuzione delle prestazioni dei modelli predittivi su OpenJPA addestrati senza feature selection

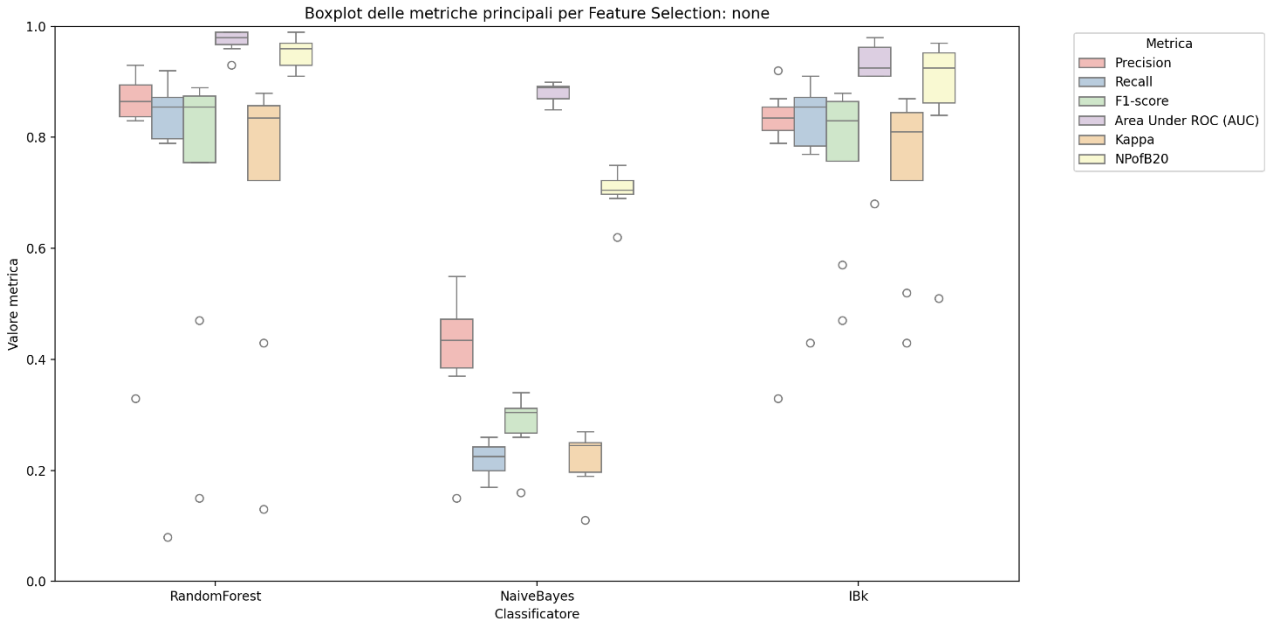


Figura 10 - Distribuzione delle prestazioni dei modelli predittivi su OpenJPA addestrati con information gain

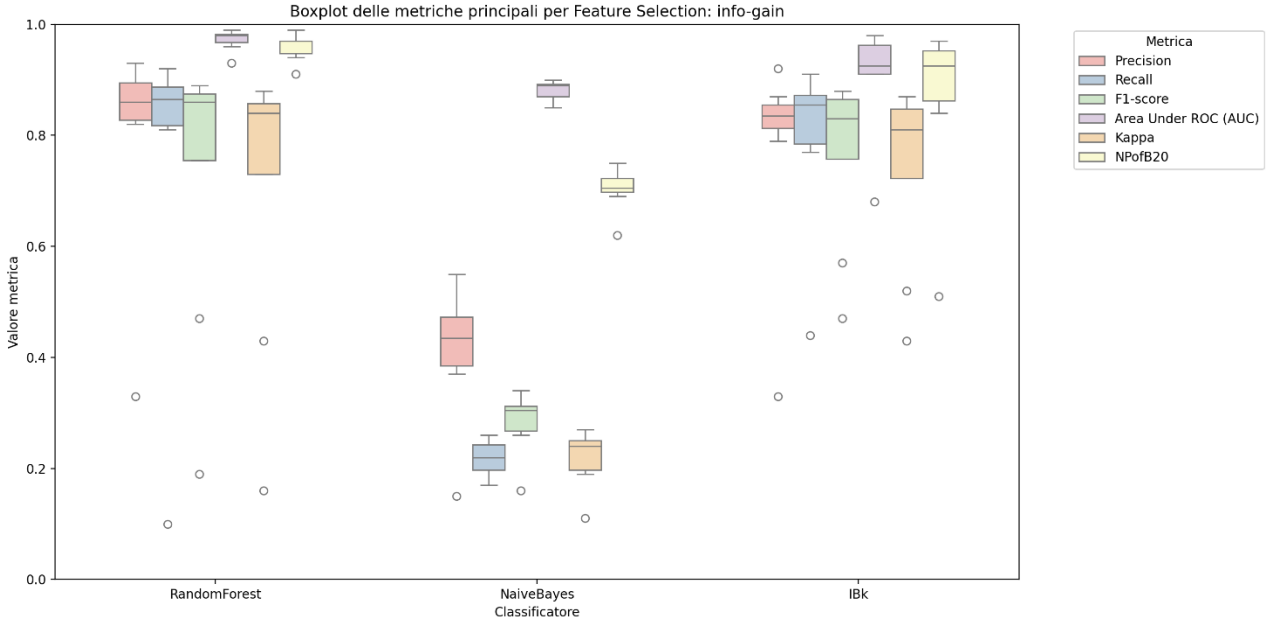


Figura 11 - Valori medi delle metriche prestazionali su OpenJPA con evidenziati i valori massimi