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

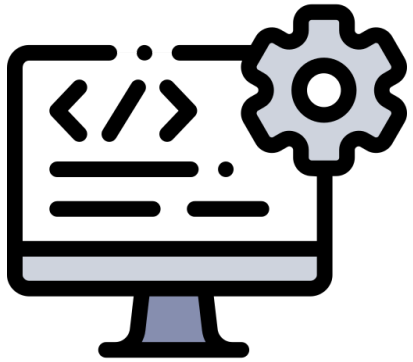
Previsione e Prevenzione dei Bug
Attraverso l'Analisi delle Metriche di Codice

Masci Francesco 0365258

Indice

Il problema della manutenibilità

La **manutenibilità** è uno dei pilastri dello sviluppo di sistemi software sostenibili. Sebbene l'attenzione si concentri spesso sulla qualità del codice, la ricerca empirica ha una lacuna nel quantificare l'impatto diretto del **refactoring** sulla riduzione dei difetti a livello di singolo metodo.



L'idea alla base del progetto è fornire uno strumento in grado di **individuare** quali metodi siano a rischio di bug, ma anche quantificare in modo tangibile quanti difetti si potrebbero **prevenire** attuando interventi mirati di refactoring.

Le domande di ricerca

Lo studio si articola attorno a due domande principali:

- **RQ1**: Quale classificatore è in grado di prevedere con maggiore accuratezza se un metodo sarà buggy in una data release?
- **RQ2**: quanti metodi buggy si potrebbero potenzialmente evitare applicando tecniche di predizione efficaci a supporto delle attività di refactoring?

Raccolta delle metriche

- Il processo di creazione del dataset è basato su un'estrazione di informazioni da due progetti open source di rilievo: **Apache BookKeeper** e **Apache OpenJPA**.
- L'obiettivo è mappare l'evoluzione di ogni singolo metodo e associarne lo stato di "buggyness" nel tempo. Per farlo, sono stati uniti i dati dei repository **Git** con le informazioni di tracciamento dei bug da **Jira**.
- Per ogni release di ciascun progetto, è stata eseguita un'analisi statica e dinamica del codice sorgente tramite Git. Sono state raccolte metriche strutturali (come LoC e Complessità Ciclomatica) e metriche storiche (come il churn) per ogni metodo in ogni commit che lo riguarda.
- È stato considerato che, qualora un metodo non fosse stato modificato in una release, le sue metriche statiche restassero invariate rispetto alla release precedente

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	Project	Package	Class	Method	Version	LOC	Statement	Cyclomati	Cognitive	MethodHi	AddedLine	MaxAddec	AvgAdded	DeletedLir	MaxDelete	AvgDelete	Churn	MaxChurn	AvgChurn	BranchPo	NestingDe	Parameter
2	BOOKKEEI	org.apach	Segment	newBuilde	4.1.1	2	2	1	0	2	5	5	5,00	0	0	0,00	5	5	5,00	0	1	1
3	BOOKKEEI	org.apach	Segment	newBuilde	4.2.0	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1
4	BOOKKEEI	org.apach	Segment	newBuilde	4.2.1	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1
5	BOOKKEEI	org.apach	Segment	newBuilde	4.2.2	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1
6	BOOKKEEI	org.apach	Segment	newBuilde	4.3.0	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1
7	BOOKKEEI	org.apach	Segment	newBuilde	4.2.3	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1
8	BOOKKEEI	org.apach	Segment	newBuilde	4.3.1	2	2	1	0	0	0	0	0,00	0	0	0,00	0	0	0,00	0	1	1

Estratto del dataset di BookKeeper

Raccolta delle metriche

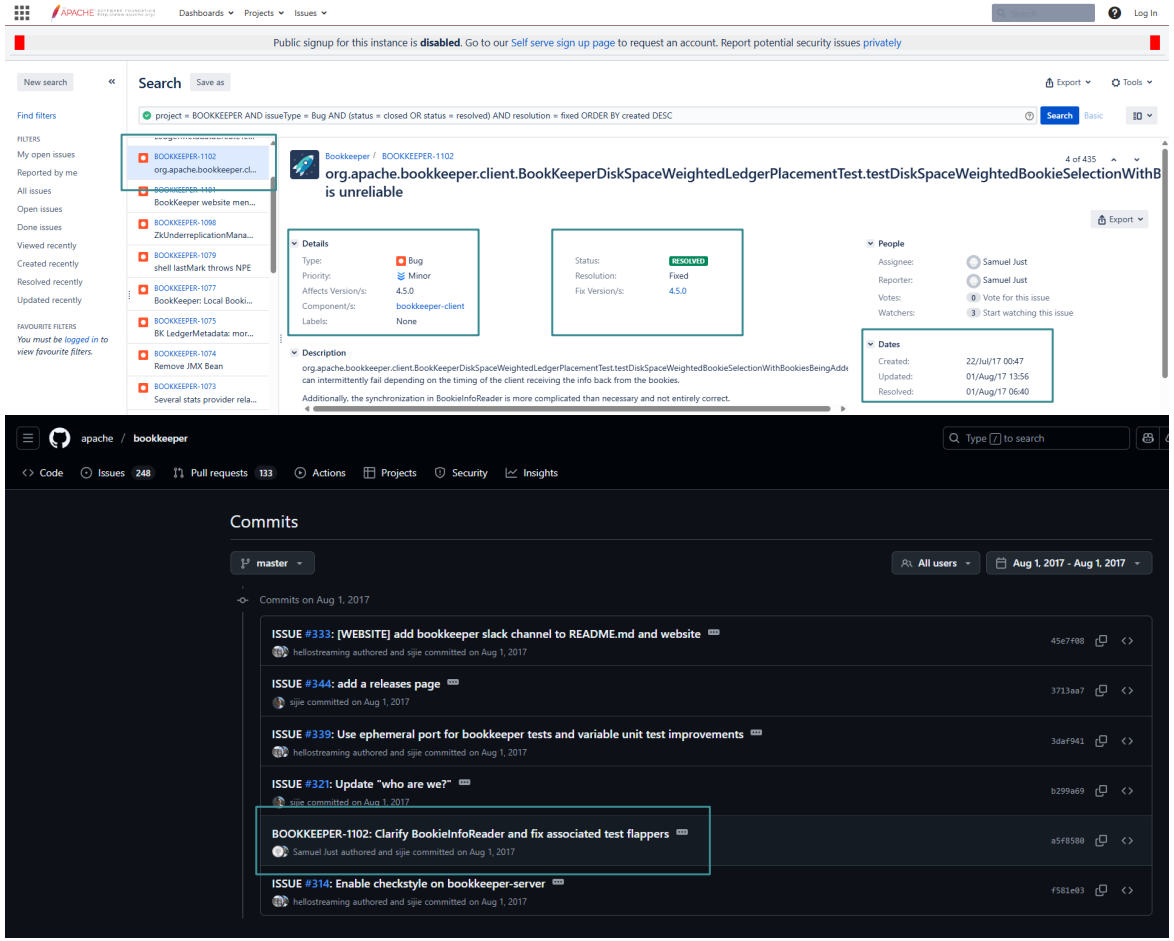
Codice	*Nome*	*Significato*
Project	Progetto	Nome del progetto software
Package	Pacchetto	Nome del package in cui il metodo è contenuto
Class	Classe	Nome della classe contenente il metodo
Method	Metodo	Nome del metodo analizzato
Version	Versione	Release software di riferimento
LOC	Lines of Code	Numero 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

Etichettatura

- Quando un commit riporta nel suo messaggio l'ID di un ticket di Jira, questo viene associato ad esso.

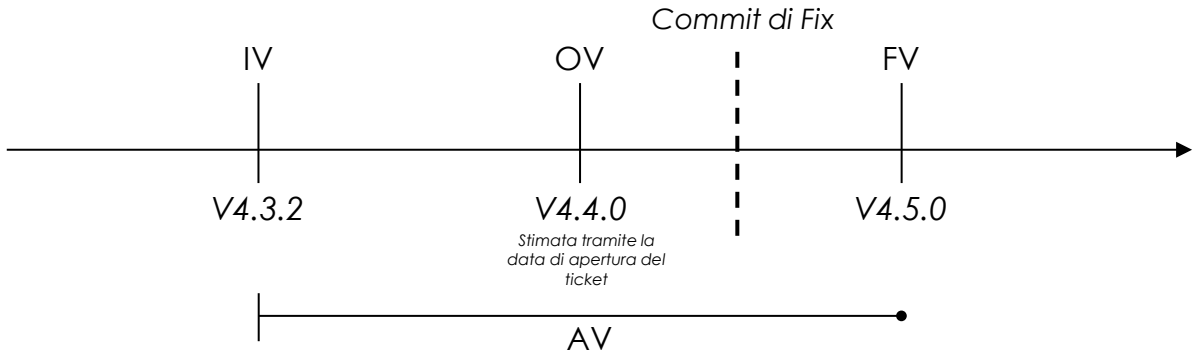
- Se il ticket è di tipo "bug " ed è stato risolto e chiuso, si assume che quel commit fosse una "fix" per il bug e i metodi modificati in esso vengono etichettati come "buggy" nelle release comprese tra *injected version* e *fixed version* del ticket in esame.

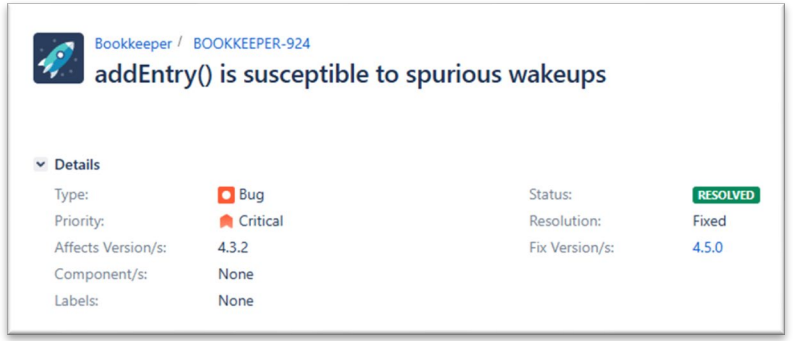
- Le informazioni non valide relative alle *affected versions* sono state **scartate** e gestite come dati mancanti, in particolare quando temporalmente successive alla *opening version* del ticket.



Moving Window Proportion

- Per superare la limitazione dei dati incompleti, abbiamo utilizzato la tecnica di inferenza statistica ***Proportion***, la quale sfrutta il comportamento storico del progetto per stimare le *affected versions (**AV**) di un ticket* tramite la sua *injected version* (**IV**), *opening version* (**OV**) e *fixed version* (**FV**).





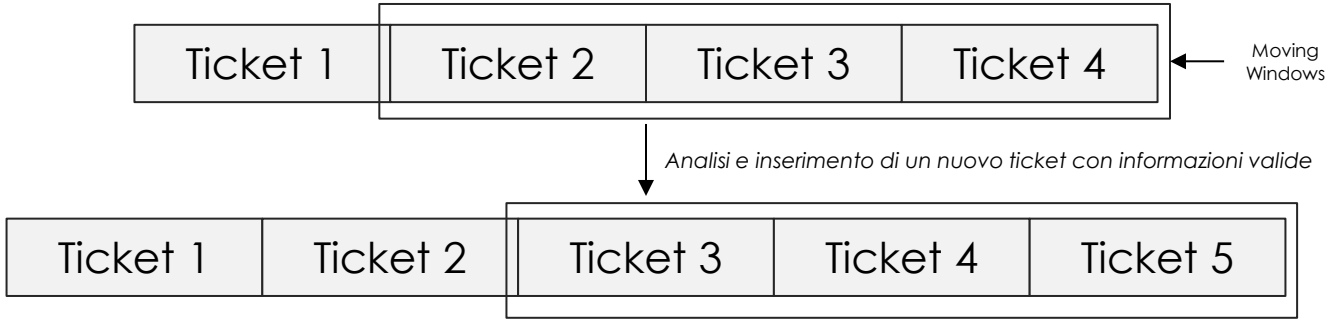
Estratto ticket da Jira

- Le *affected versions* sono tutte quelle comprese tra la IV (inclusa) e la FV (esclusa).

Moving Window Proportion

- Viene creata una finestra mobile (*moving window*) che esamina un sottoinsieme di ticket adiacenti. La dimensione della finestra è fissa, pari all'1% del totale dei ticket.

- In ogni istante, il sistema mantiene nella finestra i ticket più recenti per i quali sono note (e valide) sia la data di inject che la data di fix.



Moving Window Proportion

- Su ogni finestra, viene calcolato il ***Proportion Ratio***, ovvero la proporzione temporale tra la durata del bug (*fix date - inject date*) e la durata di risoluzione (*fix date*).

$$P = \frac{FV - IV}{FV - OV}$$

- Questo *Proportion Ratio* viene applicato ai ticket che non hanno una *inject date* esplicita per stimarla a partire dalla *fix date*.

$$IV = FV - (FV - OV) \cdot P$$

L'impatto delle metriche

- L'analisi di correlazione ha rivelato una distinzione fondamentale tra i due progetti, evidenziando che l'impatto delle metriche sulla "buggyness" dipende dal contesto del software.

BOOKKEEPER

Le metriche storiche sono risultate i predittori più influenti. La forte correlazione negativa tra la "buggyness" e metriche come *MethodHistories* e *AvgChurn* suggerisce che i metodi che subiscono modifiche frequenti tendono a maturare, diventando più stabili e meno soggetti a bug nel tempo. La storia evolutiva di un metodo è quindi un indicatore cruciale del suo stato di salute.

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

Correlazione delle feature in BookKeeper

L'impatto delle metriche

- L'analisi di correlazione ha rivelato una distinzione fondamentale tra i due progetti, evidenziando che l'impatto delle metriche sulla "buggyness" dipende dal contesto del software.

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

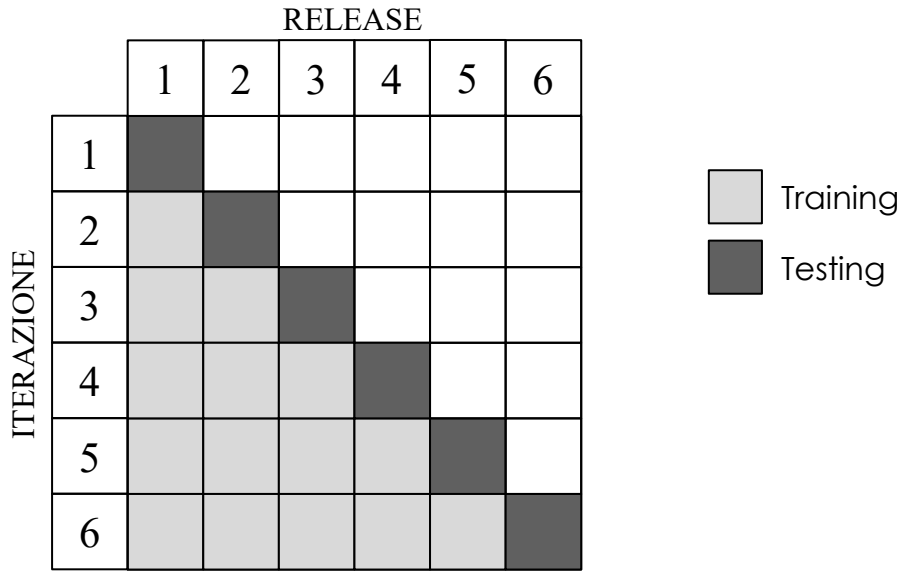
Correlazione delle feature in OpenJPA

OPENJPA

Le metriche strutturali sono risultate essere i predittori dominanti. Correlazioni positive con *Statement* e *LOC* indicano che la complessità intrinseca e le dimensioni del codice sono i principali fattori di rischio. Metodi più lunghi e complessi sono, in questo caso, intrinsecamente più propensi a contenere difetti.

Validazione Walk-Forward

- Per garantire che i nostri risultati fossero robusti e applicabili in uno scenario di sviluppo reale, è stata scelta la tecnica di validazione **walk-forward**.
- Metodi di validazione classici come il *k-fold cross-validation* non sono ideali per l'analisi di dati sequenziali come le release di un software. Utilizzare dati futuri per addestrare un modello che deve prevedere eventi passati (fenomeno noto come *data leakage*) porta a risultati non realistici e a una sovrastima delle prestazioni.
- Questo approccio simula un ambiente di produzione. Per ogni release successiva, il modello viene addestrato solo sui dati delle release precedenti.
- Questo metodo garantisce che il modello utilizzi solo informazioni disponibili in un dato momento, rendendo le previsioni applicabili in un contesto reale.

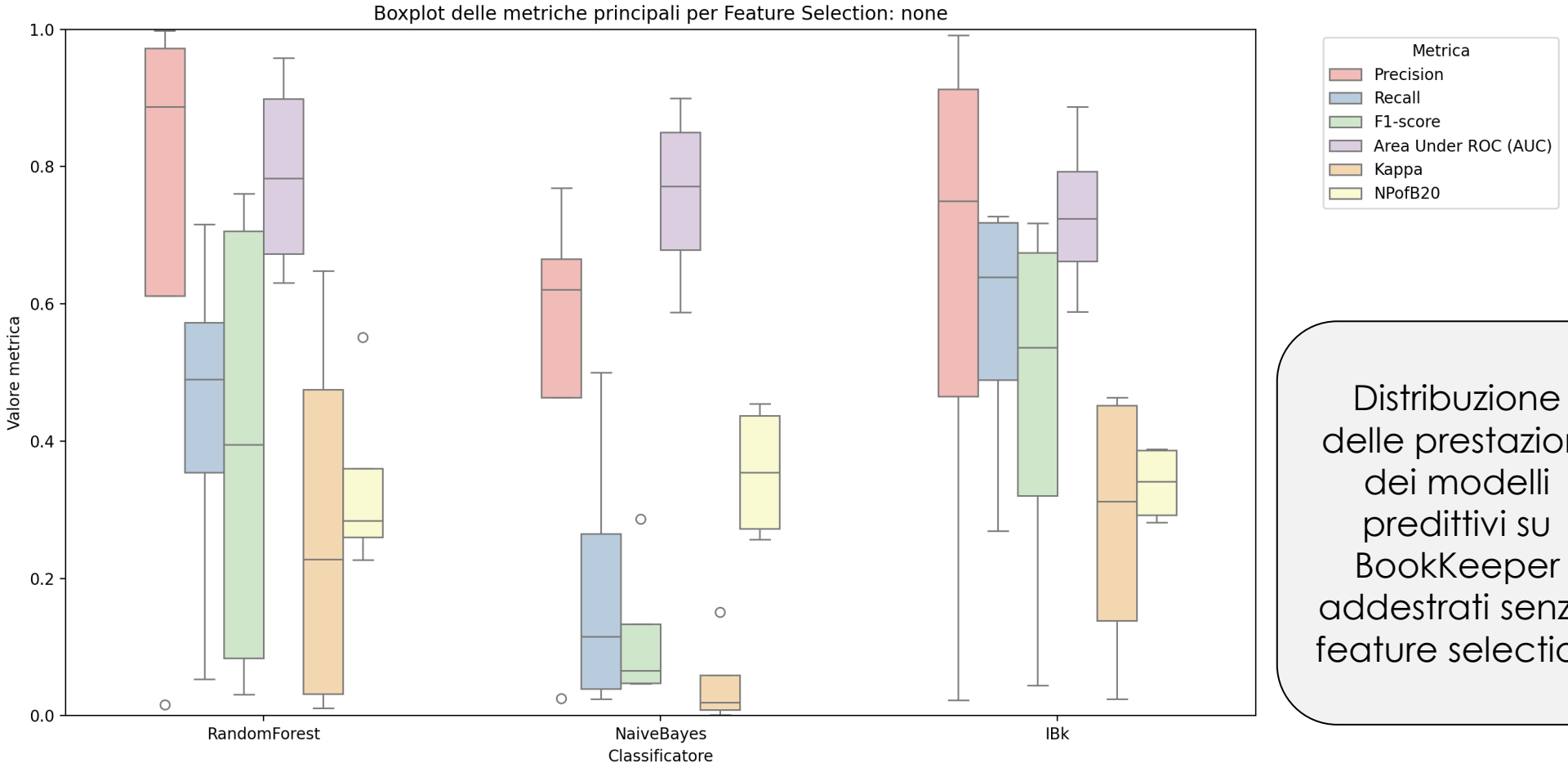


Confronto dei modelli predittivi

- Per l'addestramento dei classificatori, sono state esplorate 2 tecniche di *feature selection*: **Information Gain** e **Forward Search**. Queste tecniche ci hanno permesso di identificare il sottoinsieme di metriche più rilevanti per la previsione dei bug.
- I modelli sono stati anche addestrati sul dataset completo per confronto.

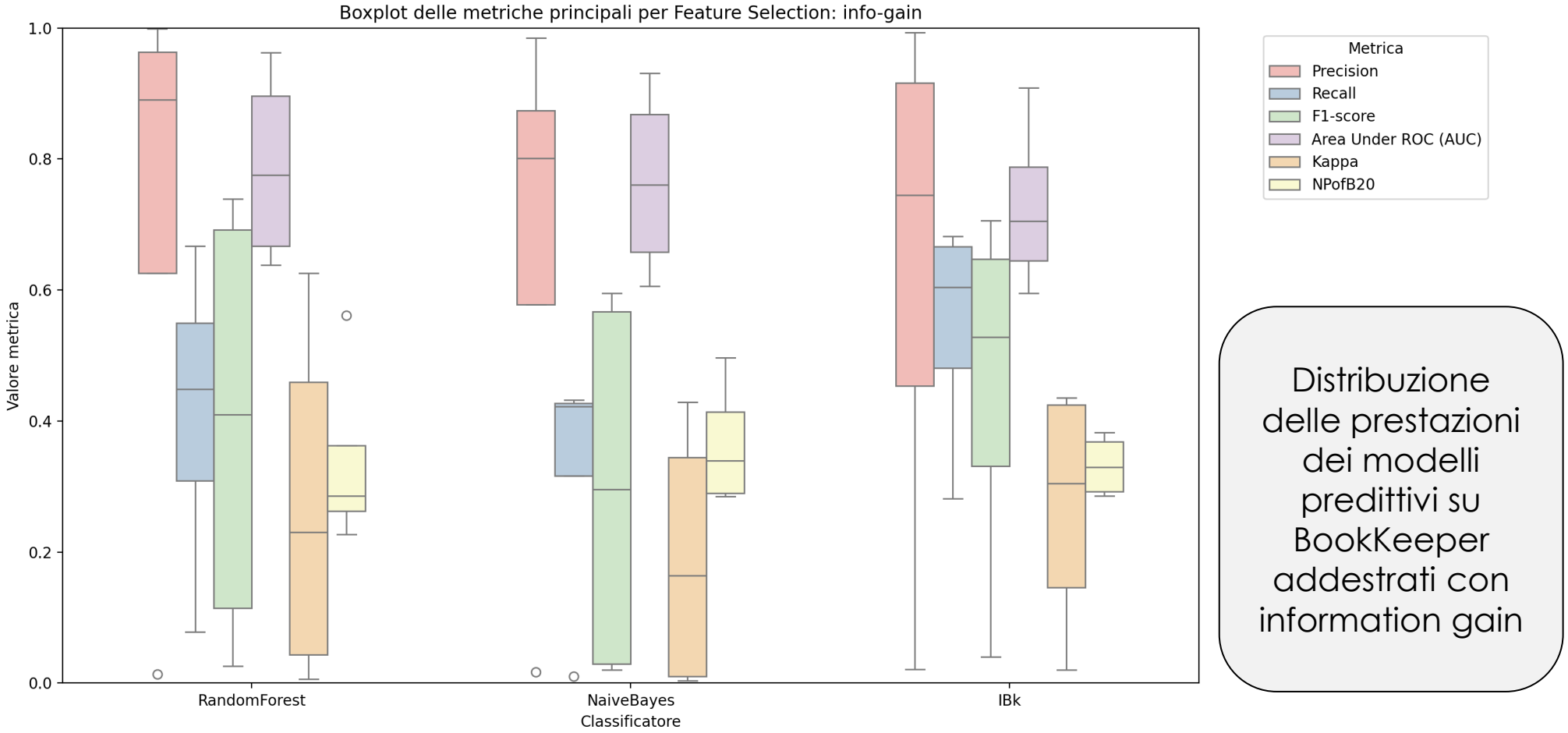
- Per identificare il classificatore più efficace nella previsione della "buggyness", abbiamo addestrato e confrontato tre modelli distinti, ciascuno con un approccio differente alla classificazione: **Naive Bayes**, **IBk (k-Nearest Neighbors)** e **Random Forest**.

Confronto dei modelli predittivi - BookKeeper



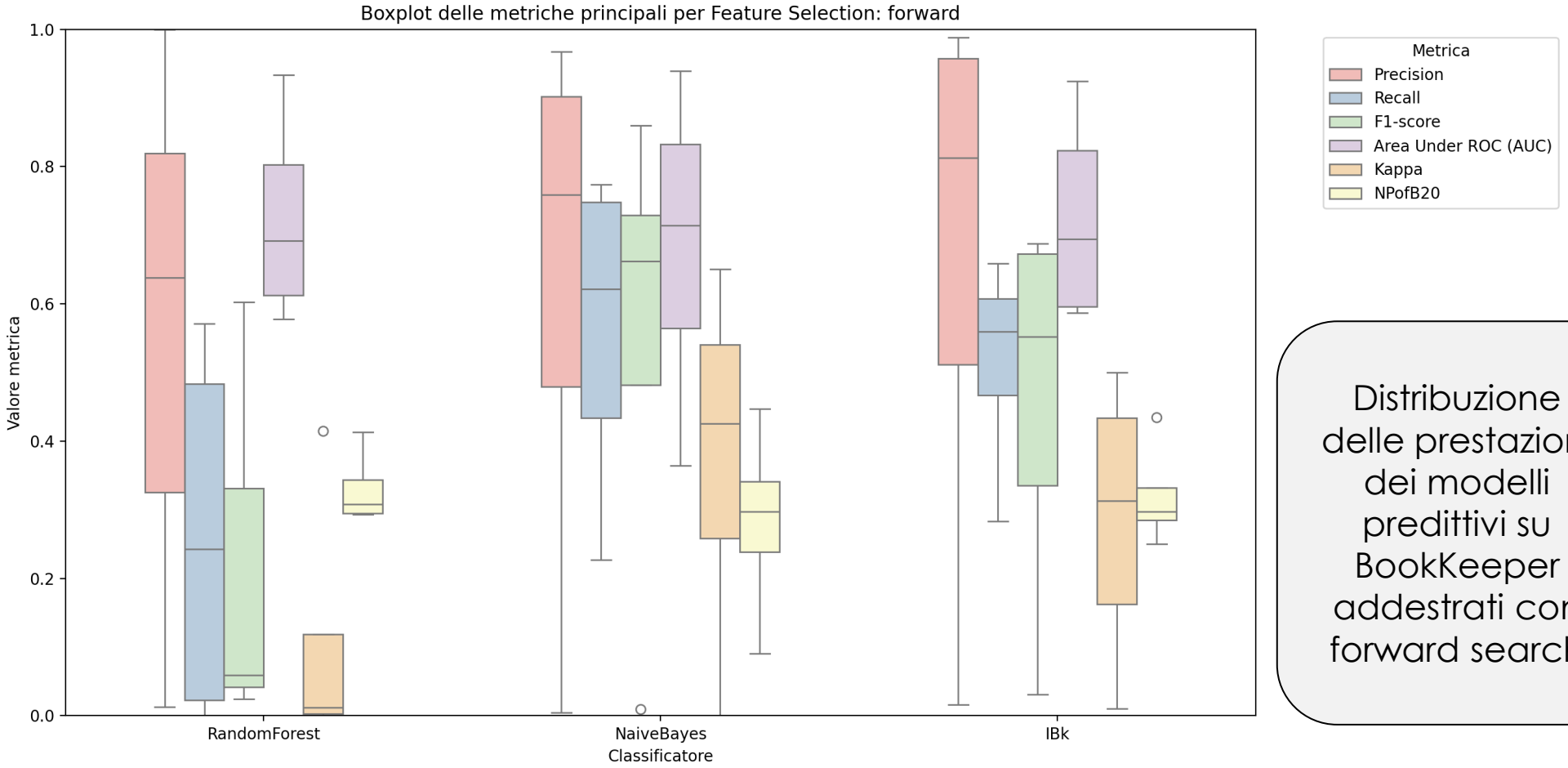
Distribuzione delle prestazioni dei modelli predittivi su BookKeeper addestrati senza feature selection

Confronto dei modelli predittivi - BookKeeper



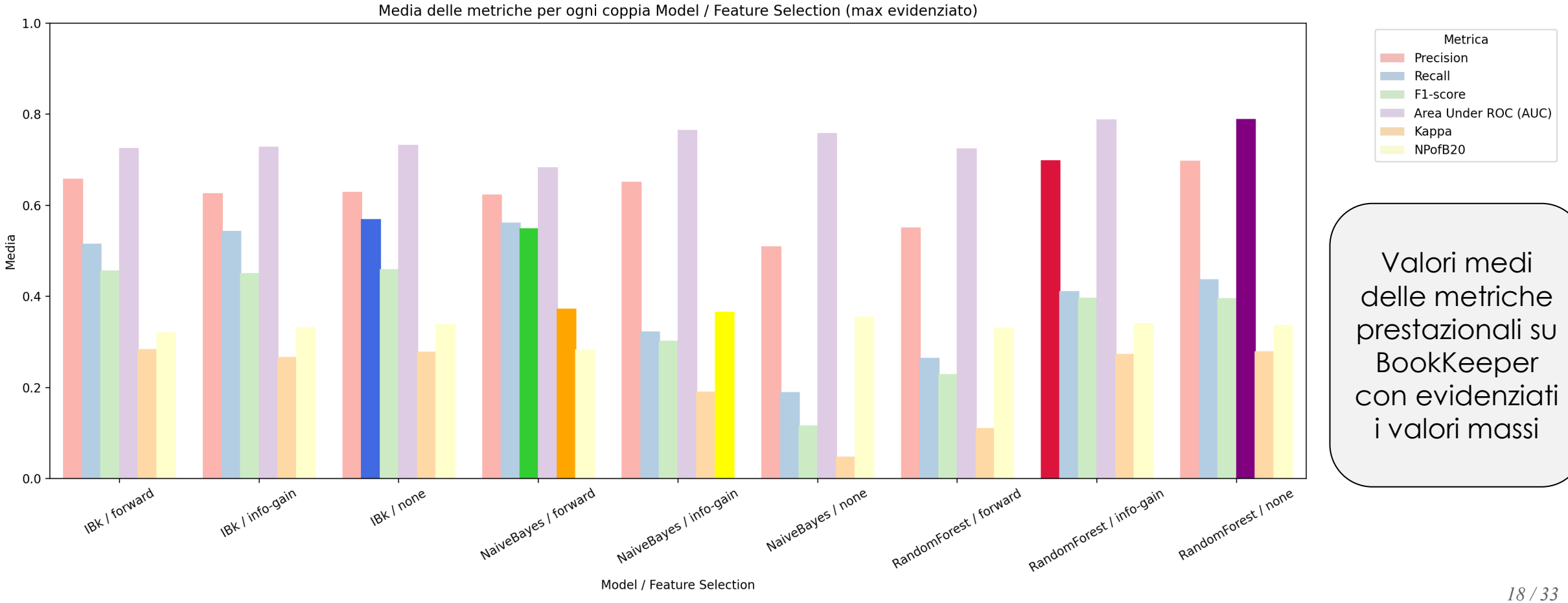
Distribuzione delle prestazioni dei modelli predittivi su BookKeeper addestrati con information gain

Confronto dei modelli predittivi - BookKeeper



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

Confronto dei modelli predittivi - BookKeeper



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

Confronto dei modelli predittivi - BookKeeper

NAIVE BAYES

Questo modello ha ottenuto i risultati complessivamente peggiori, con valori di *recall* e *precision* più bassi.

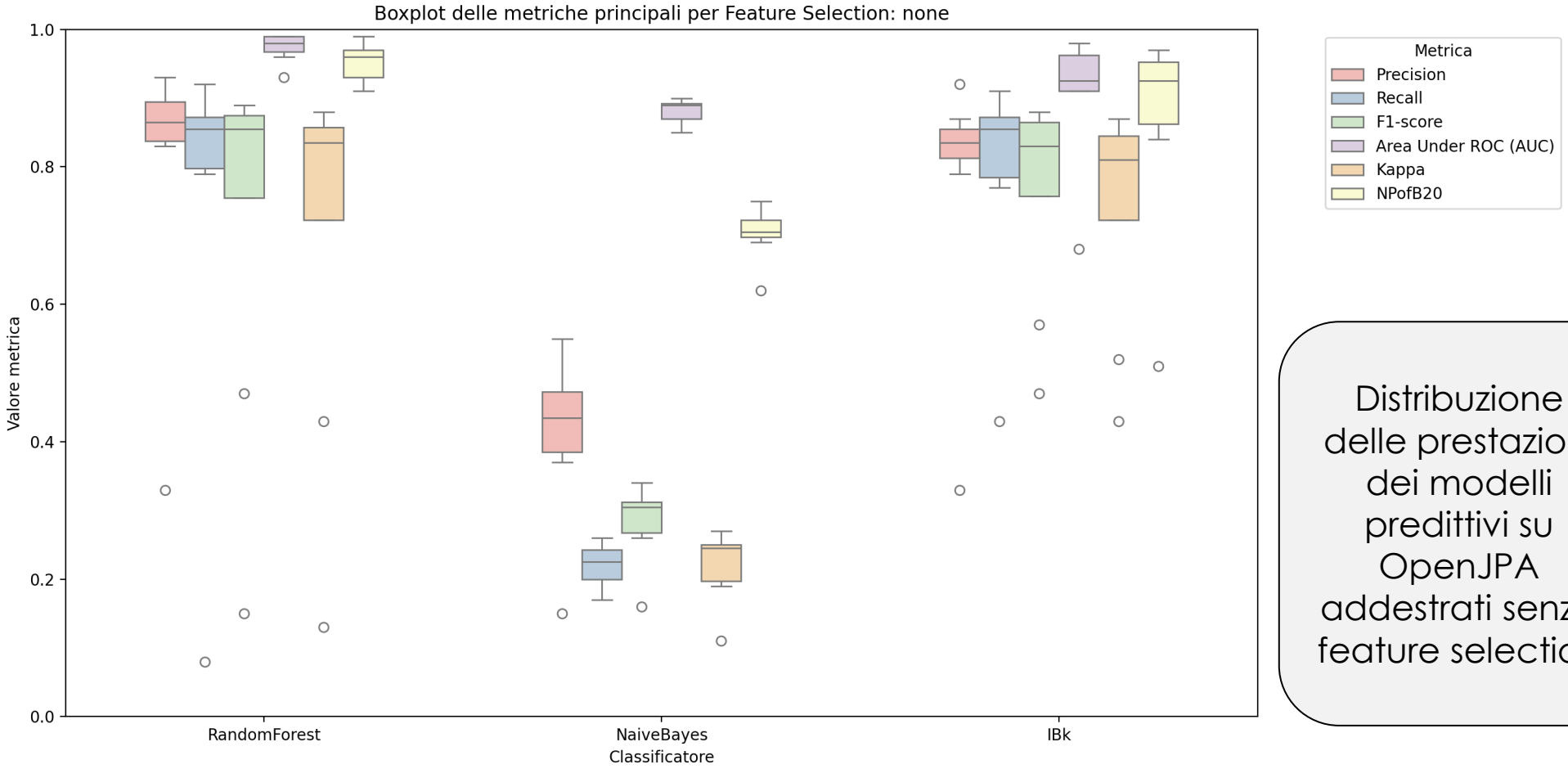
IBK

Ha mostrato risultati comparabili a *Random Forest* in alcune metriche, ma è risultato meno stabile e robusto rispetto alle variazioni dei dati.

RANDOM FOREST

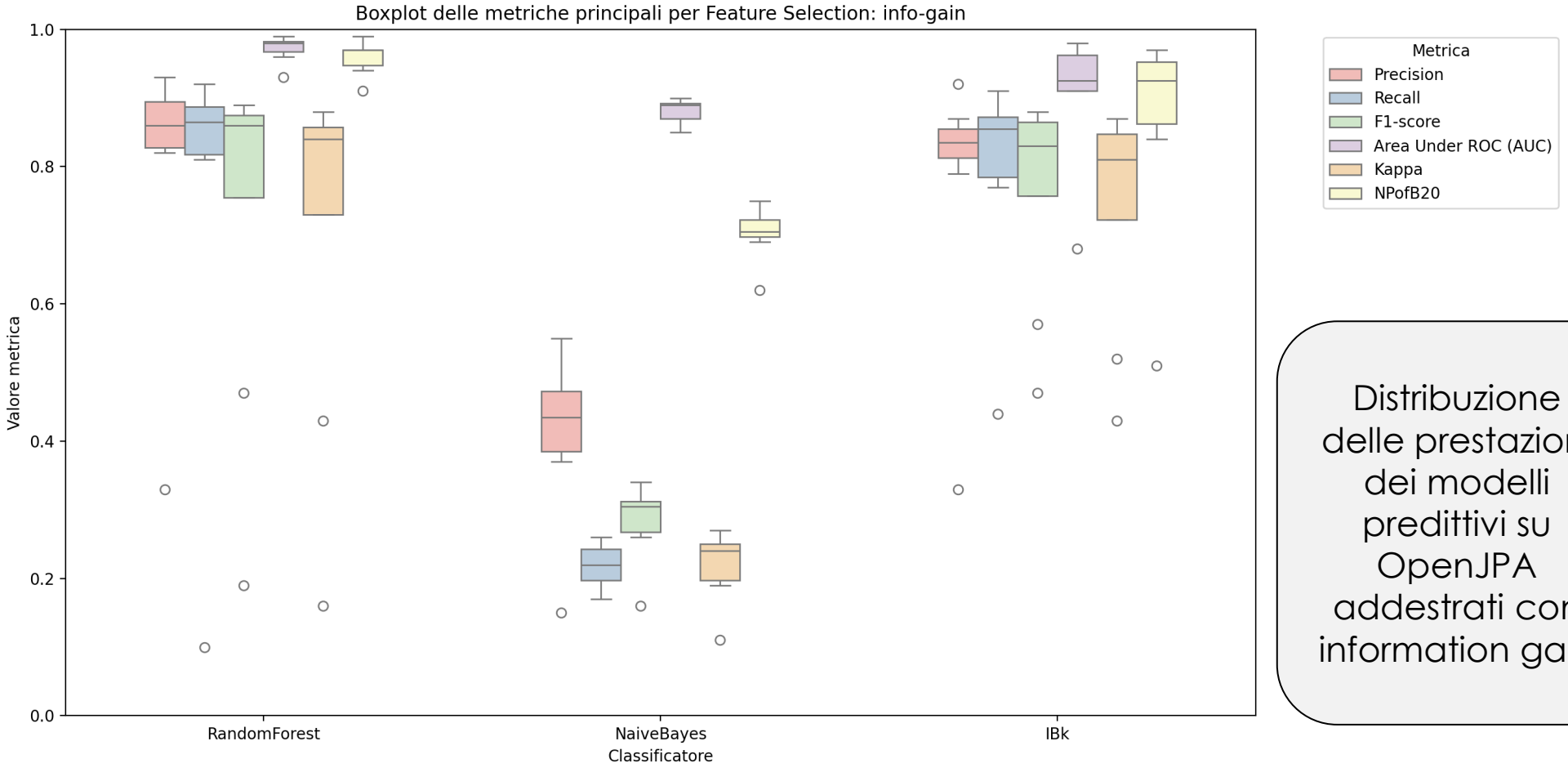
Il modello si è dimostrato il più robusto e stabile, offrendo una migliore capacità di generalizzazione.

Confronto dei modelli predittivi - OpenJPA



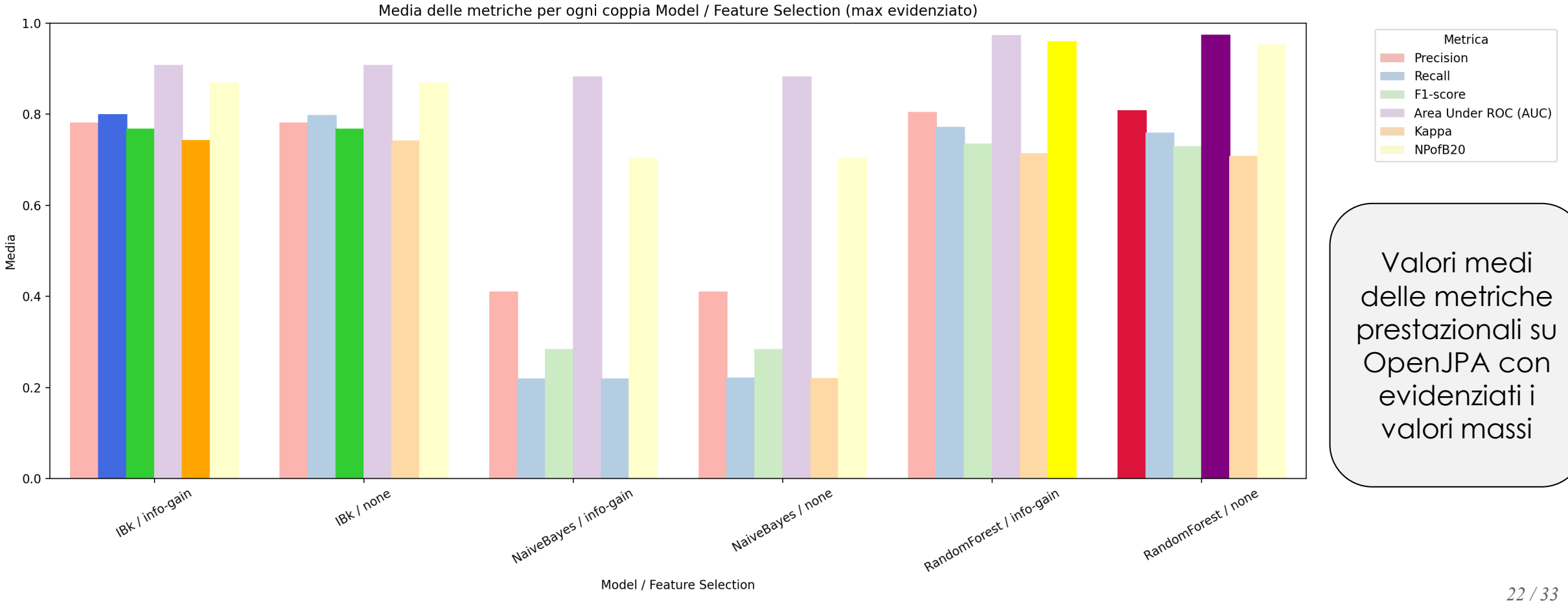
Distribuzione delle prestazioni dei modelli predittivi su OpenJPA addestrati senza feature selection

Confronto dei modelli predittivi - OpenJPA



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

Confronto dei modelli predittivi - OpenJPA



Valori medi delle metriche prestazionali su OpenJPA con evidenziati i valori massi

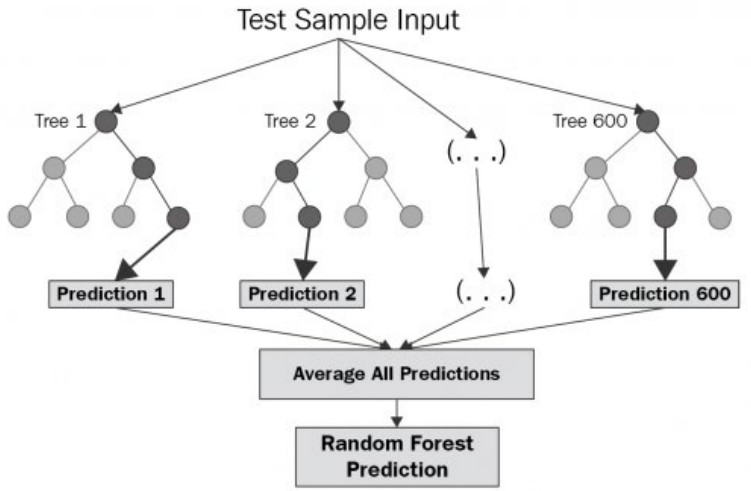
Confronto dei modelli predittivi - OpenJPA

RANDOM FOREST

Il modello Random Forest ha costantemente superato gli altri classificatori. La sua capacità di gestire le complessità del dataset ha garantito le migliori prestazioni, come dimostrato dalla distribuzione dell'AUC che evidenzia una minore variabilità e un'alta efficacia predittiva. Questo risultato si è mantenuto anche in presenza della riduzione della dimensionalità operata dalla selezione delle feature.

Confronto dei modelli predittivi

- Le tecniche di selezione delle feature hanno giocato un ruolo cruciale nell'ottimizzazione dei modelli.
- L'**Information Gain** è stato il metodo più efficace per entrambi i progetti, riducendo le feature da 22 a 13 in Bookkeeper e a 18 in OpenJPA. Questa riduzione non ha compromesso le prestazioni del modello.
- **Forward Search**: Ha prodotto modelli instabili con metriche sensibilmente inferiori rispetto all'Information Gain.
- In conclusione, i risultati confermano che **Random Forest**, combinato con la selezione delle feature tramite **Information Gain**, rappresenta il modello più equilibrato e robusto per la previsione dei bug in entrambi i contesti analizzati.



Refactoring guidato dai dati

- La selezione dei metodi da sottoporre a refactoring non è stata arbitraria, ma è stata guidata direttamente dai risultati dell'analisi di correlazione. Sono stati individuati, infatti, i metodi con le metriche più fortemente correlate alla "buggyness".

BOOKKEEPER

L'analisi di correlazione ha rivelato che la complessità ciclomatica è una delle metriche più predittive. La scelta è ricaduta sul metodo ***processPacket***, che presentava un valore eccezionalmente alto (34), rendendolo un candidato ideale per un intervento di refactoring facilmente misurabile e rappresentativo del problema.

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

Metriche dei metodi antecedenti al refactoring

Refactoring guidato dai dati

- La selezione dei metodi da sottoporre a refactoring non è stata arbitraria, ma è stata guidata direttamente dai risultati dell'analisi di correlazione. Sono stati individuati, infatti, i metodi con le metriche più fortemente correlate alla "buggyness".

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

Metriche dei metodi antecedenti al refactoring

OPENJPA

La selezione del metodo ***eval*** è stata guidata dalla forte correlazione positiva tra la "buggyness" e il numero di statement. Questo metodo, con un valore eccezionalmente alto di 175 statement, rappresenta un caso emblematico di come un codice eccessivamente lungo e complesso aumenti la probabilità di difetti, rendendolo il bersaglio perfetto per un intervento di refactoring.

Refactoring guidato dai dati

- Per entrambi i metodi è stata utilizzata la tecnica di **Extract Method**, che consiste nel suddividere un metodo troppo lungo o complesso in sottocomponenti più piccoli e gestibili, ognuno con una singola responsabilità.

BOOKKEEPER

Il metodo ***processPacket*** è stato ridotto in tre metodi più piccoli, portando a una riduzione della complessità ciclomatica del 79% (da 34 a 7).

OPENJPA

Il metodo ***eval*** è stato frammentato in quattro unità più piccole, con una riduzione del numero di statement di oltre il 90% (da 175 a 13).

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

Metriche dei metodi a seguito del refactoring

Analisi "what-if" - BookKeeper

- Per valutare l'impatto potenziale del refactoring guidato, è stata condotta un'analisi "what-if", simulando il suo effetto e studiandone i risultati.
- Nel caso di BookKeeper, l'intero dataset originale (**Dataset A**) è stato scomposto in due sottoinsiemi: un primo gruppo di 7.032 metodi che contenevano la feature "actionable" più correlata, ovvero la complessità ciclomatica (**Dataset B+**), e un secondo gruppo di 100 metodi che non presentavano tale caratteristica (**Dataset C**).
- Per simulare il refactoring sul **Dataset B+,** è stato costruito, a partire da esso, il **Dataset B** dove artificialmente è stata azzerata la feature "actionable".
- I risultati suggeriscono che, se si azzerasse la feature "actionable" da tutti i metodi in cui è presente, si potrebbero potenzialmente evitare 47 bug, equivalenti a circa lo 0,66% dei bug totali del progetto.

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

Esiti dell'analisi What-If sul progetto BookKeeper

Analisi "what-if" - OpenJPA

- Come per BookKeeper, l'intero dataset originale (**Dataset A**) è stato scomposto in due sottoinsiemi: un primo gruppo di 14.334 metodi che contenevano la feature "actionable" più correlata, ovvero il numero di statement (**Dataset B+**), e un secondo gruppo di 578 metodi che non presentavano tale caratteristica (**Dataset C**).

- Per simulare il refactoring sul **Dataset B+,** è stato costruito, a partire da esso, il **Dataset B** dove artificialmente è stata azzerata la feature "actionable".

- In questo contesto simulato, il numero di metodi buggy predetti è sceso di 486 unità. Questa diminuzione è pari al 3,25% delle previsioni iniziali sul Dataset B+ (486 su 14938). Inoltre, l'affidabilità del modello è migliorata notevolmente, passando da una sovrastima del 4,21% a una sovrastima dello 0,82% rispetto ai bug predetti sul Dataset B+.

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

Esiti dell'analisi What-If sul progetto OpenJPA

Risposte alle domande di ricerca

- Il nostro studio ha fornito una risposta chiara e concreta alle domande di ricerca, confermando che un approccio predittivo può guidare in modo efficace il processo di manutenzione del software.

- **Risposta a RQ1:** Il classificatore **Random Forest** si è dimostrato il più efficace, ottenendo le migliori prestazioni in termini di AUC e precision. La sua robustezza e capacità di gestire le complessità dei dati lo rendono lo strumento ideale per identificare in modo proattivo i metodi a rischio di bug.
- **Risposta a RQ2:** L'analisi "what-if" ha quantificato il potenziale impatto del refactoring, dimostrando che la riduzione delle metriche più fortemente correlate ai difetti può portare a una diminuzione significativa del numero di bug attesi. La simulazione ha mostrato un potenziale di prevenzione di 47 bug per BookKeeper e ben 486 bug per OpenJPA.

- Questi risultati offrono una prova tangibile del legame tra le metriche di qualità del codice e la presenza di difetti, suggerendo che un'attenzione mirata al miglioramento strutturale può portare a un notevole aumento della manutenibilità del software.

Minacce alla validità

- È importante riconoscere che la nostra analisi è soggetta a determinate limitazioni e minacce alla validità. Riconoscerle è fondamentale per comprendere il contesto dei risultati e per orientare la ricerca futura.

- **Errori di Misura delle Metriche:** Il calcolo automatico delle metriche può essere soggetto a imprecisioni. Piccoli errori nella misurazione possono propagarsi e influenzare le previsioni del modello.
- **Assunzioni nell'Analisi "What-If":** L'esperimento what-if si basa su due assunzioni semplificative. La prima è che le feature siano tra loro indipendenti, e la seconda è che sia possibile azzerare completamente la feature "actionable" tramite un refactoring. In uno scenario reale, la rimozione completa non è sempre possibile, e l'interazione tra le metriche può essere più complessa.
- **Bias nella Selezione:** La scelta delle metriche su cui intervenire per l'analisi what-if è stata basata sulle correlazioni più forti, ma può essere influenzata da un bias soggettivo. Questo potrebbe portare a ignorare altre metriche potenzialmente rilevanti.

Lavori futuri

- Nonostante i risultati positivi, la ricerca apre la strada a ulteriori sviluppi per migliorare e validare l'approccio predittivo in diversi contesti. I prossimi passi futuri potrebbero includere:
- **Analisi comparativa:** Applicare la stessa metodologia a progetti con architetture non object-oriented (come progetti funzionali o ibridi) per verificare se le correlazioni e le previsioni si mantengono valide.
- **Integrazione dei dati:** Arricchire il dataset con metriche a runtime e dati di performance per rafforzare ulteriormente la predizione.
- **Sviluppo di strumenti interattivi:** Creare tool per gli ingegneri del software che, in tempo reale, evidenzino i metodi critici durante la fase di scrittura del codice, trasformando la prevenzione dei bug in un processo integrato e continuo.

RIFERIMENTI

GitHub repository per il codice sorgente: https://github.com/F-masci/isw2-prediction

Analisi di SonarCloud: https://sonarcloud.io/project/overview?id=F-masci_isw2-prediction

GRAZIE PER L'ATTENZIONE