# Analisi della Buggyness nei Metodi Software:

## Un Approccio Data-Driven al Refactoring

Meola Gaia 0369454

# Indice

# Introduzione

## Il problema della manutenibilità

L’incessante integrazione di funzionalità degrada la struttura interna del software. Questo accumulo di ***Debito Tecnico*** aumenta esponenzialmente la propensione ai difetti nei sistemi complessi.

La letteratura attuale si focalizza su **classi** o **moduli**.
Manca un'analisi empirica solida a livello di **singolo metodo**.


Studio sui progetti **Apache BookKeeper** e **OpenJPA** per validare l'efficacia del Machine Learning nella predizione dei bug e quantificare quanti bug si potrebbero prevenire attuando mirati interventi di **refactoring**.

# Le Domande di Ricerca

**RQ1:** Quale dei classificatori valutati risulta più efficace nell'individuare con precisione i metodi problematici nei due scenari studiati?

**RQ2:** Quanti potenziali bug si sarebbero potuti evitare applicando in tempo attività di refactoring mirate alla rimozione delle anomalie strutturali?

# Misura e Metodologia

# Dataset ed Estrazione delle Etichette

**Target:** Generazione di un dataset etichettato a livello di **metodo**.

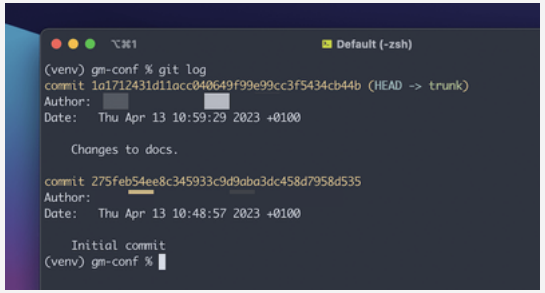

**Input**: Repository **Git** (storico codice) e **Jira** (segnalazioni bug).

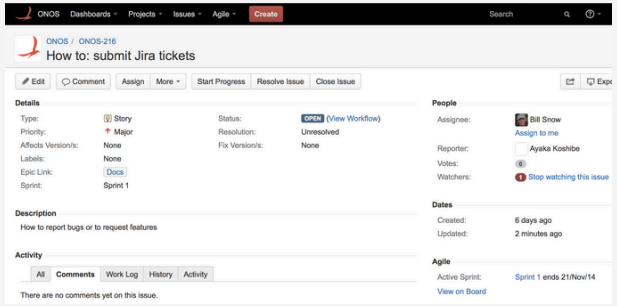
**Mitigazione Snoring**: Rimozione selettiva delle release finali per eliminare il rumore.

**Risultato**: Coinvolgimento del **33% delle release** nel dataset finale.

# Dataset ed Estrazione delle Etichette

**Associazione Ticket-Commit**: Spesso i messaggi di commit non contengono l'ID del ticket Jira (es. "BOOKKEEPER-123"), portando a una perdita di dati preziosi per il dataset. Quindi si è preferito adottare un approccio ibrido per garantire la qualità del dataset:




## Approccio Ibrido Multi-Livello

- **Linkage Diretto (Pattern Matching):** Identificazione immediata tramite Regex degli ID ticket nei messaggi di commit (es. commit.getMessage()).
- **Euristica di Recupero (Missing Link Recovery):** Per i commit senza ID, il sistema applica una validazione basata su tre vincoli di integrità:
  - **Vincolo Temporale**: Il commit deve ricadere nell'intervallo (Opening Version, Fix Version) del bug.
  - **Coincidenza di Autore**: Verifica che il commit sia stato effettuato dallo stesso sviluppatore che ha risolto il ticket su Jira.
  - **Consistenza dei File**: Il commit deve aver modificato file già confermati come parte della fix in altri commit associati al ticket.

# Dataset ed Estrazione delle Etichette

Per garantire la massima precisione, il sistema non si limita a identificare i file modificati, ma **isola i singoli metodi** difettosi attraverso un'analisi chirurgica del codice sorgente.

**Estrazione dei Diff (JGit):** Per ogni commit di fix, il sistema calcola le differenze rispetto al parent, isolando i blocchi di righe modificati .

**Mappatura dei Confini:** Grazie all'analisi statica, ogni metodo nel dataset è associato a coordinate fisiche precise [*StartLine*, *EndLine*].

**Validazione dell'Intersezione:** Un metodo viene etichettato come *buggy* solo se esiste una sovrapposizione fisica tra le righe variate nel diff e l'intervallo sintattico del metodo stesso.

**Riduzione del Rumore:** Questo approccio permette di scartare le modifiche effettuate nello stesso file ma esterne al metodo (es. refactoring di altri metodi o commenti), assicurando che solo il codice funzionalmente responsabile del bug venga marcato nel dataset.

# Dataset ed Estrazione delle Etichette

**Problematica:** L'analisi Method-Level SZZ è un'indagine chirurgica che isola il metodo colpevole nella Fixed Version.

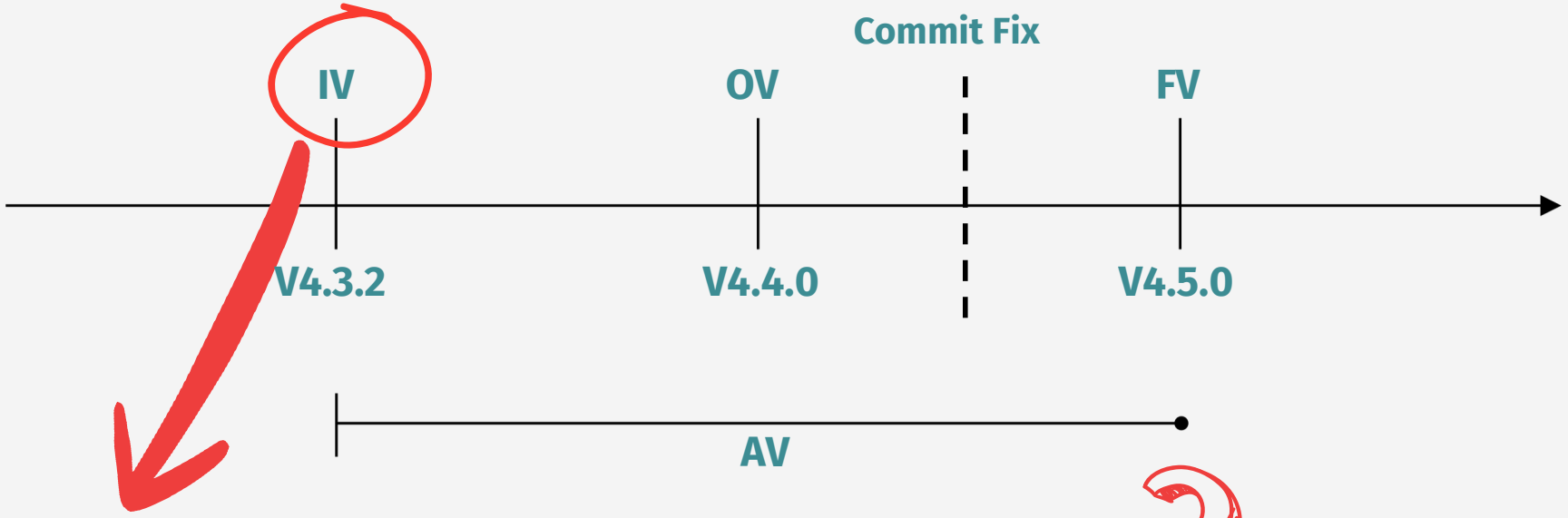
Tuttavia, questo dato descrive la soluzione, ma non l'origine.

**Domanda Chiave:**

*"In assenza di Affected Versions su Jira, come identifichiamo tutte le release in cui il metodo era già presente e difettoso?"*

# Dataset ed Estrazione delle Etichette

*<<Esisterà un numero proporzionale di versioni per trovare il difetto e per risolverlo. >>*




$$IV_{index} = FV_{index} - (FV_{index} - OV_{index}) \times P$$

# Dataset ed Estrazione delle Etichette

**Filtro Outlier**: Scarto dei ticket aventi **P > 1.5**

**Cold Start**: Utilizzo di un valore di P globale derivato da benchmark esterni

**Stima Incrementale**: Transizione Automatica a P locale al raggiungimento della soglia di **10 ticket validi**

**Labeling finale**: Propagazione dell'etichetta *buggy* per tutte le istanze dei metodi nell'intervallo [IV, FV)

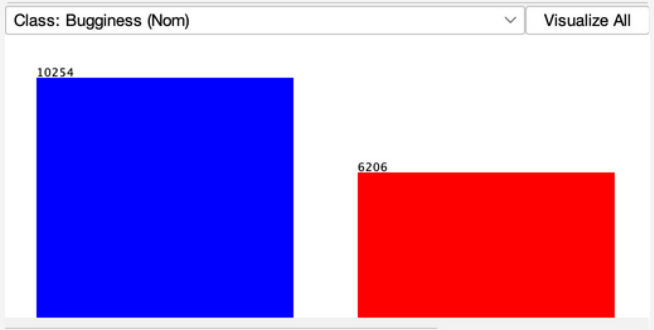
# Metriche e Definizioni

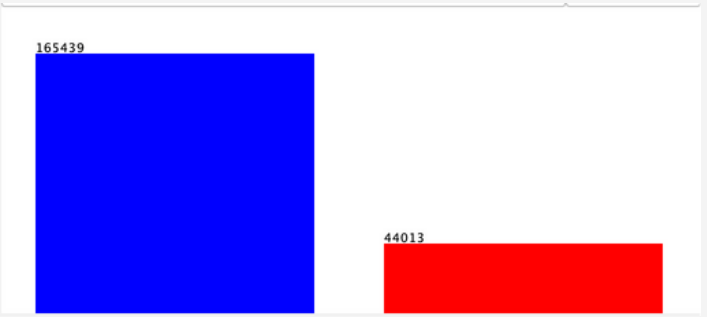
| Metrica | Definizione |
| --- | --- |
| MethodHistories | Numero totale di revisioni (commit) subite da un metodo. |
| DistinctAuthors | Numero di diversi sviluppatori che hanno modificato quel codice. |
| Churn | Indice di instabilità: quanto spesso il codice viene riscritto o rimosso. |
| StmtAdded | Quantità totale di istruzioni aggiunte nel ciclo di vita del codice. |
| StmtDeleted | Quantità totale di istruzioni rimosse nel ciclo di vita del codice. |

# Metriche e Definizioni

| Metrica | Definizione |
|---|---|
| StatementCount | Numero complessivo di istruzioni eseguibili presenti. |
| Number of Smells | Quantità di anomalie nel design che indicano potenziali bug o fragilità. |
| CyclomaticComplexity | Numero di percorsi logici indipendenti (complessità dei rami if/loop). |
| CognitiveComplexity | Grado di difficoltà che uno sviluppatore incontra nel capire il flusso del codice. |
| Nesting Depth | Livello massimo di annidamento di blocchi di codice uno dentro l'altro. |
| LocalVariableCount | Numero di variabili locali definite. |
| ReturnTypeComplexity | Grado di complessità del tipo di dato che la funzione restituisce. |
| LOC | Lines of Code: numero totale di linee che compongono il file. |
| ParameterCount | Numero di parametri in ingresso a una funzione o metodo. |

# Milestone1




| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Project | Method | ReleaseID | LOC | CyclomaticCc | CognitiveCon | Number of S | ParameterCc | NestingDept | StatementCc | LocalVariabl | ReturnTypeC | MethodHisto | StmtAdded | StmtDeleted | Churn | DistinctAuth | Bugginess |
| 2 | Bookkeeper | ../bookkeepe | 4.0.0 | 35 | 5 | 5 | 1 | 1 | 2 | 10 | 6 | 1 | 3 | 7 | 3 | 10 | 2 | Yes |
| 3 | Bookkeeper | ../bookkeepe | 4.0.0 | 17 | 2 | 1 | 0 | 1 | 1 | 8 | 2 | 1 | 5 | 35 | 1 | 36 | 1 | Yes |
| 4 | Bookkeeper | ../bookkeepe | 4.0.0 | 20 | 4 | 3 | 0 | 0 | 1 | 10 | 4 | 2 | 6 | 41 | 8 | 49 | 2 | Yes |
| 5 | Bookkeeper | ../bookkeepe | 4.0.0 | 5 | 2 | 1 | 0 | 0 | 1 | 3 | 1 | 1 | 3 | 10 | 1 | 11 | 1 | Yes |
| 6 | Bookkeeper | ../bookkeepe | 4.0.0 | 2 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 2 | 4 | 0 | 4 | 1 | Yes |
| 7 | Bookkeeper | ../bookkeepe | 4.0.0 | 9 | 3 | 3 | 1 | 2 | 2 | 3 | 2 | 1 | 3 | 13 | 0 | 13 | 1 | Yes |

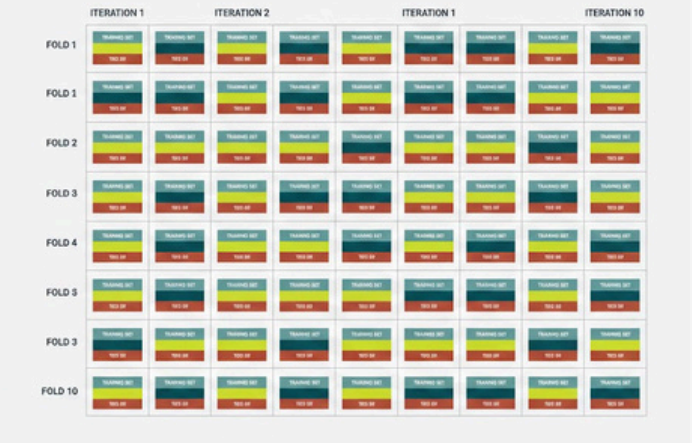
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Project | Method | ReleaseID | LOC | CyclomaticCc | CognitiveCon | Number of S | ParameterCc | NestingDept | StatementCc | LocalVariabl | ReturnTypeC | MethodHisto | StmtAdded | StmtDeleted | Churn | DistinctAuth | Bugginess |
| 2 | Openjpa | ../openjpa/o | 0.9.0 | 4 | 2 | 1 | 0 | 2 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No |
| 3 | Openjpa | ../openjpa/o | 0.9.0 | 5 | 2 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No |
| 4 | Openjpa | ../openjpa/o | 0.9.0 | 5 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No |
| 5 | Openjpa | ../openjpa/o | 0.9.0 | 5 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No |
| 6 | Openjpa | ../openjpa/o | 0.9.0 | 5 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 | 0 | 2 | 1 | No |
| 7 | Openjpa | ../openjpa/o | 0.9.0 | 5 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | No |

# Protocollo Sperimentale

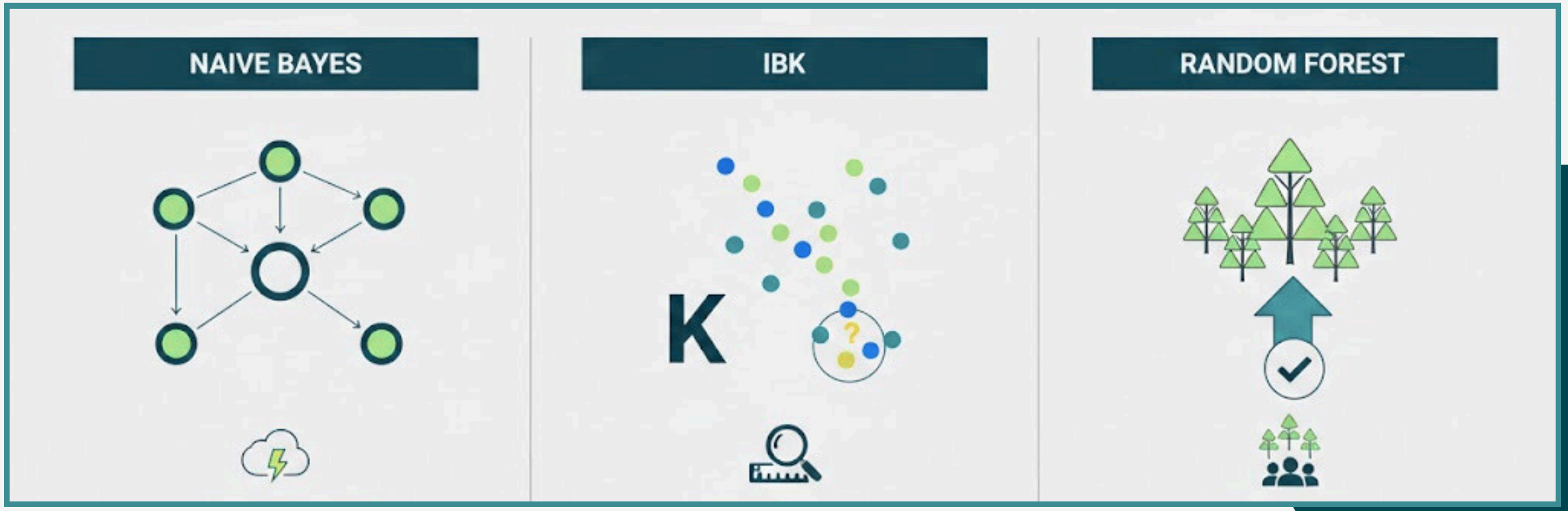
Strategia: **10-times 10-cross validation**

Scelta Metodologica: L'analisi focalizzata sul primo **33%** delle release avrebbe reso una suddivisione cronologica troppo frammentata, impedendo al modello di imparare dati significativi.

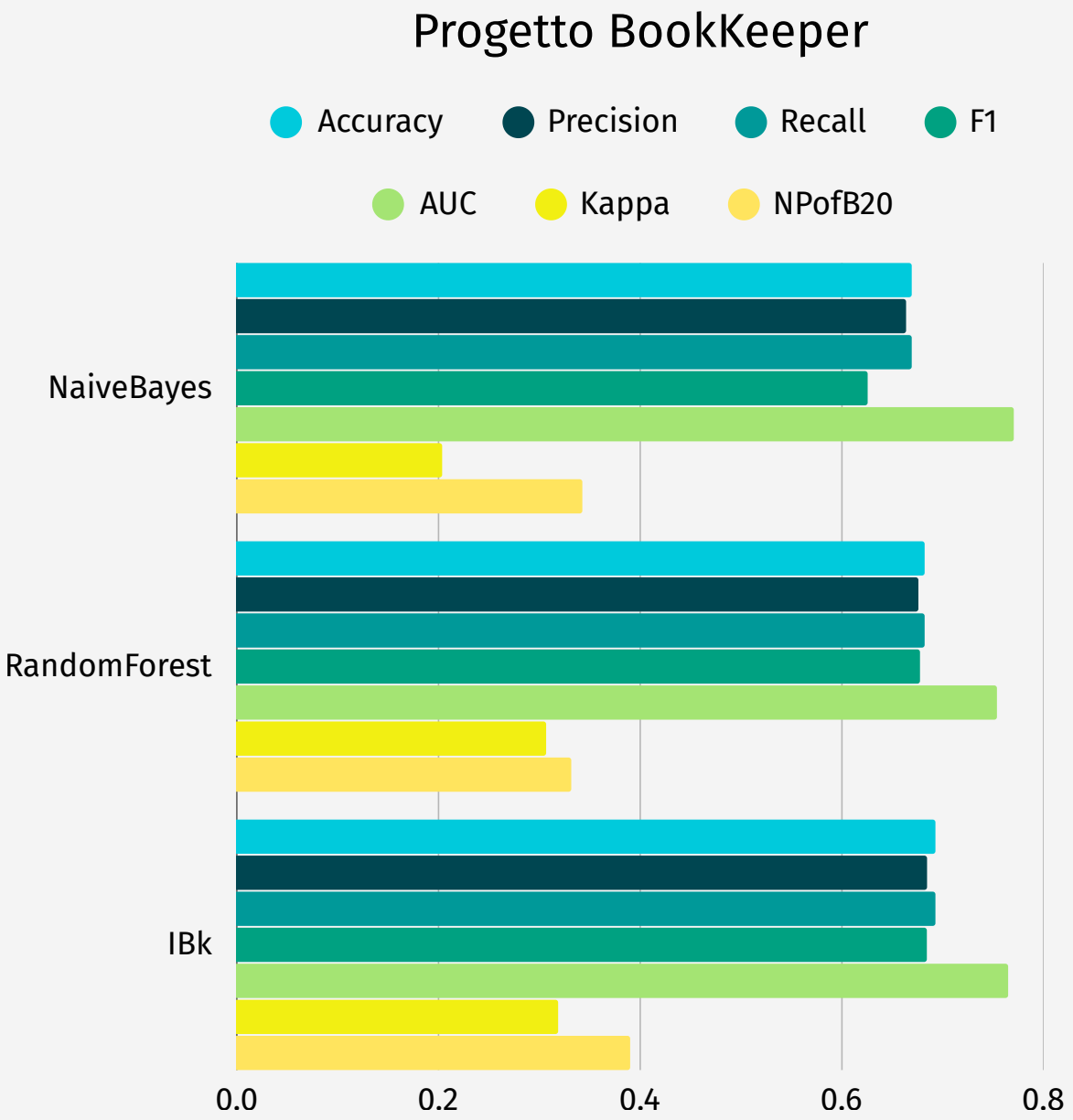
Feature Selection: Feature Selection di tipo **Filter** basata su **Info Gain** (soglia impostata a 0.1) per escludere quelle metriche con potere discriminante trascurabile rispetto alla variabile target.

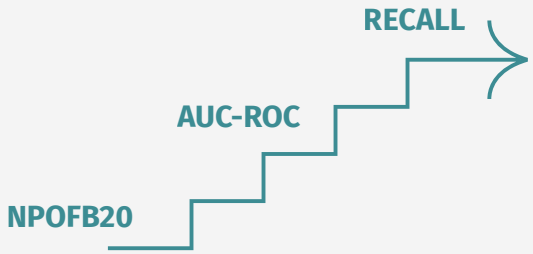



# Protocollo Sperimentale



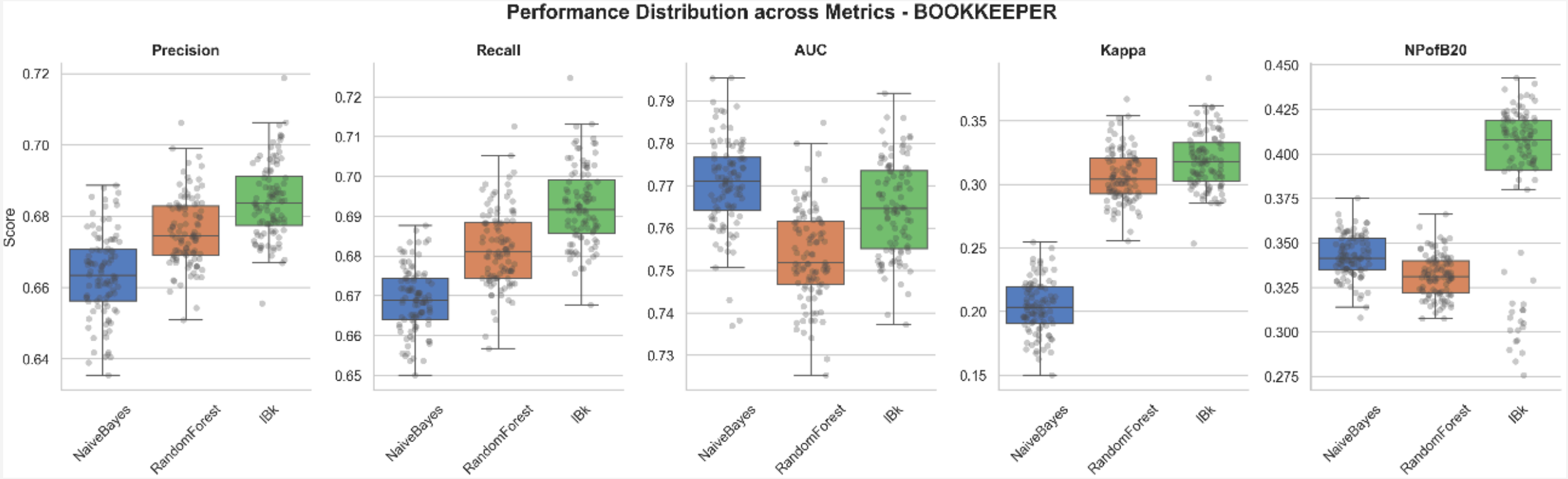
# Milestone2



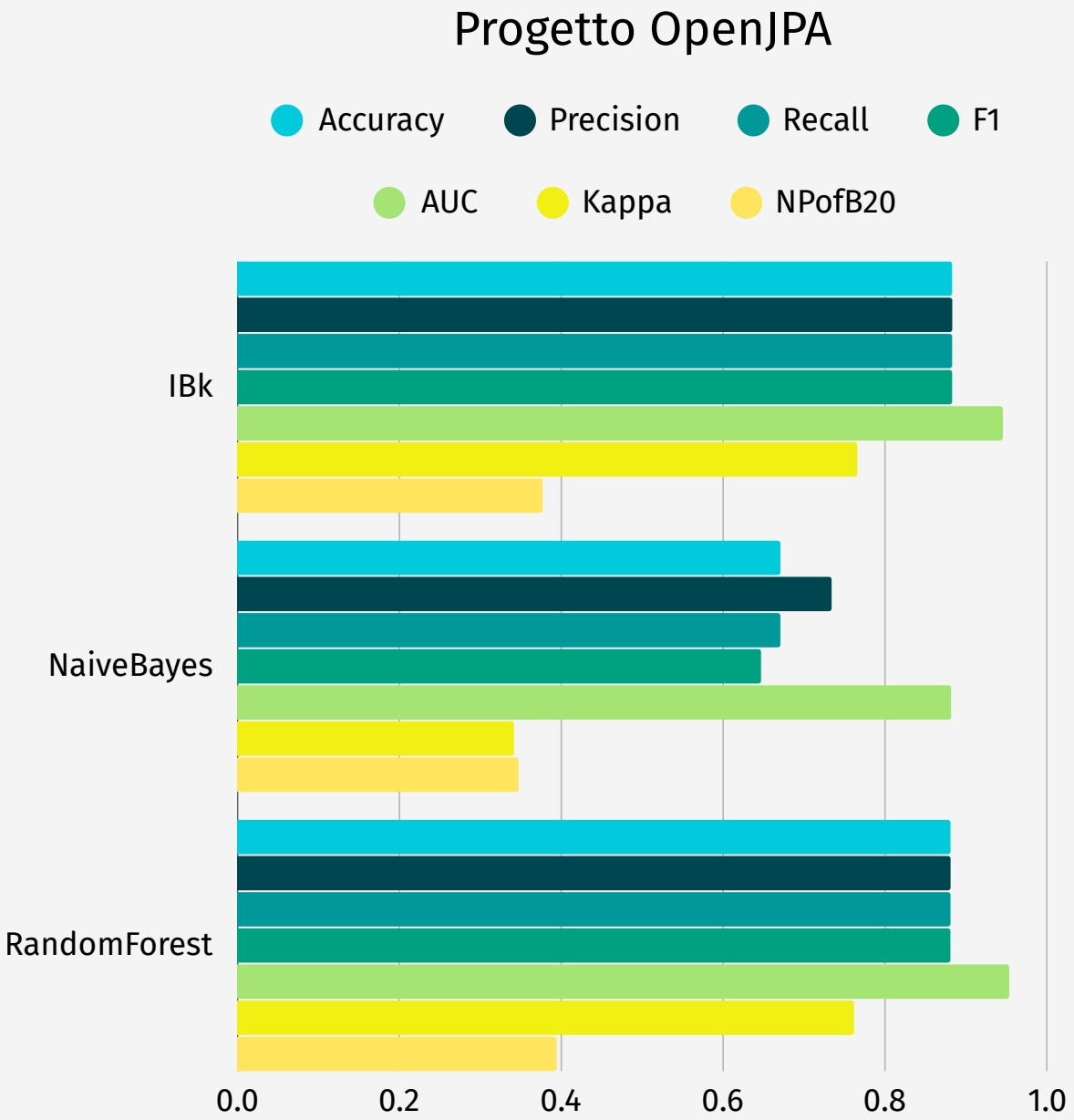



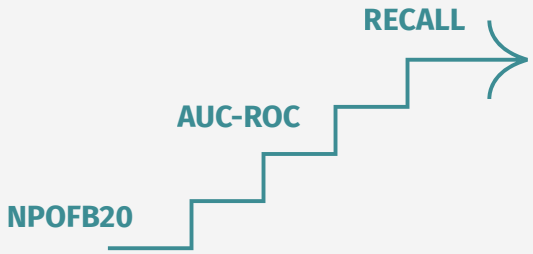

**BestClassifier - Caso Studio 1: BookKeeper**

Classificatore Selezionato: **IBk (k = 3)**


Performance Distribution across Metrics - BOOKKEEPER
Precision
Recall
AUC
Kappa
NPofB20
Score
0.72
0.70
0.68
0.66
0.64
0.71
0.69
0.67
0.65
0.79
0.78
0.77
0.76
0.75
0.74
0.73
0.35
0.30
0.25
0.20
0.15
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
NaiveBayes
RandomForest
IBk

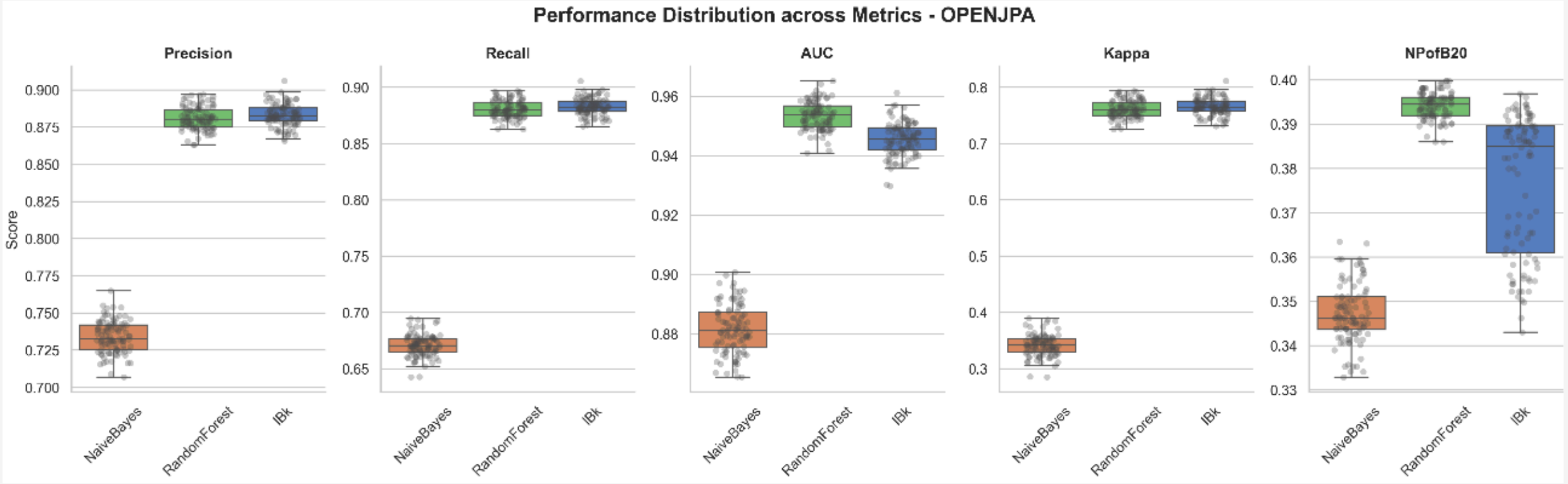
# Milestone2




## BestClassifier - Caso Studio 2: OpenJPA

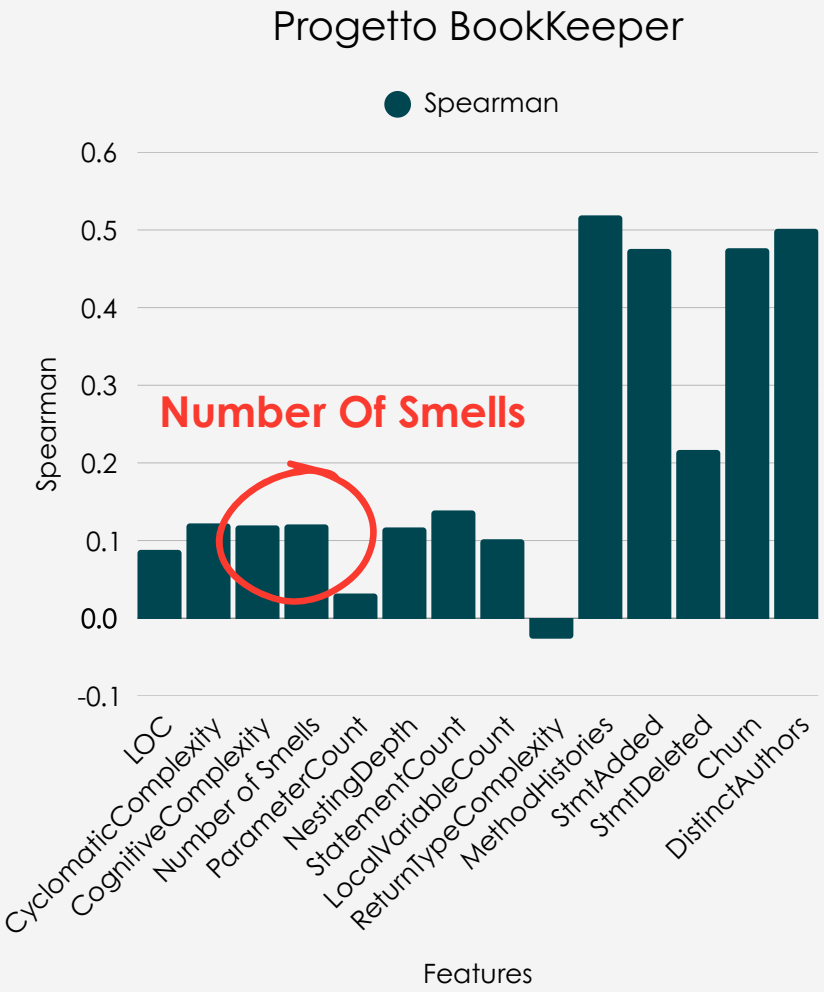
Classificatore Selezionato:
**Random Forest**

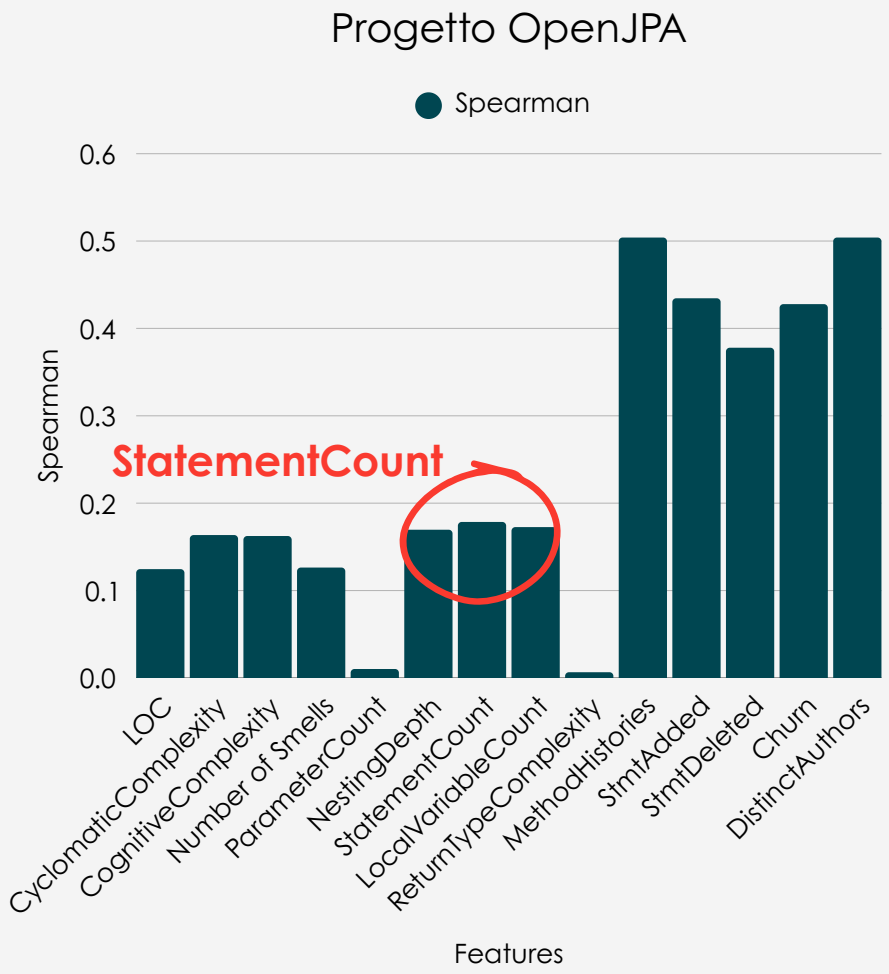

Performance Distribution across Metrics - OPENJPA
Precision
Recall
AUC
Kappa
NPofB20
Score
NaiveBayes
RandomForest
IBk

# Risultati

# Selezione dei Metodi Target

**Obiettivo:** identificare quali features hanno il maggiore impatto statistico sulla **bugginess**



# Analisi Critica dei Metodi Target

**Caso Studio 1:**

BookKeeper (v4.2.1) Metodo:
BenchReadThroughputLatency.main()

Signature: public static void main(String args) throws Exception

Ruolo Operativo: Orchestrazione del benchmark prestazionale, inclusa l'inizializzazione dell'infrastruttura ZooKeeper e la gestione del ciclo di vita dei ledger.

Analisi delle Criticità: Il metodo rappresenta un hotspot qualitativo critico caratterizzato da 8 Code Smell attivi.

**Caso Studio 2:**

OpenJPA (v2.0.0) Metodo:
FieldMetaData.copy()

Signature: public void copy(FieldMetaData field)

Ruolo Operativo: Gestione della logica di duplicazione dei metadati di persistenza tra istanze della classe.

Analisi delle Criticità: Nonostante il numero di righe contenuto (55 LOC), il metodo esibisce una densità logica anomala con uno Statement Count di 38.

# Criteri di Identificazione e Strategia di Refactoring

**Problema:** 8 Code Smells critici

**Strategia di Refactoring:**

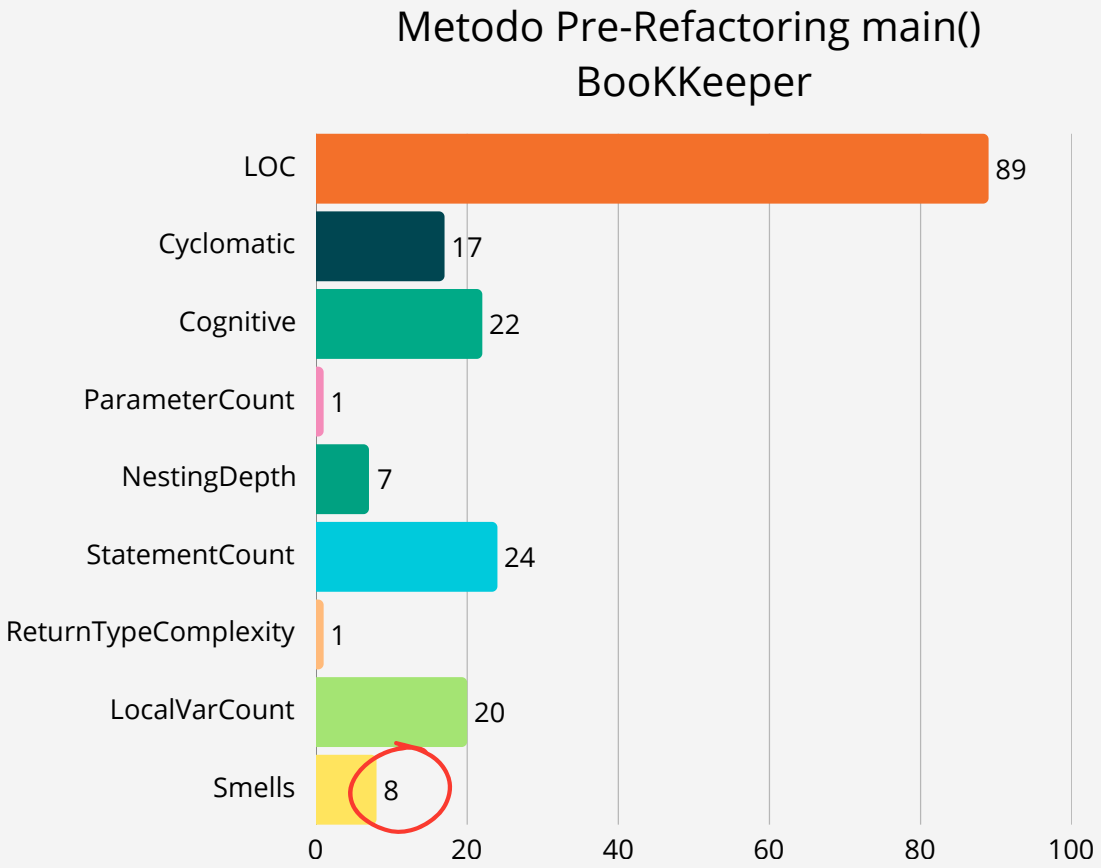
1. ***Extract Method iterativo***: Scomposizione della logica procedurale del main nei metodi specializzati getOptions, validateArgs e setLedgerParams.
2. ***Static Delegation***: Delega della logica dei Watcher ai metodi statici handleZkEvent e handleChildrenChanged.
3. ***Specializzazione dell'Exception Handling***: Sostituzione dei catch generici con blocchi mirati (KeeperException) per garantire resilienza di rete senza oscurare errori critici.
4. ***Visibility Refactoring***: Passaggio della visibilità da private a package-private per eliminare la generazione automatica di metodi sintetici (AccessorMethodGeneration).
5. ***Type Migration***: Eliminazione dell'overhead degli oggetti wrapper tramite l'uso di primitivi (parseInt/parseLong) al posto di valueOf.

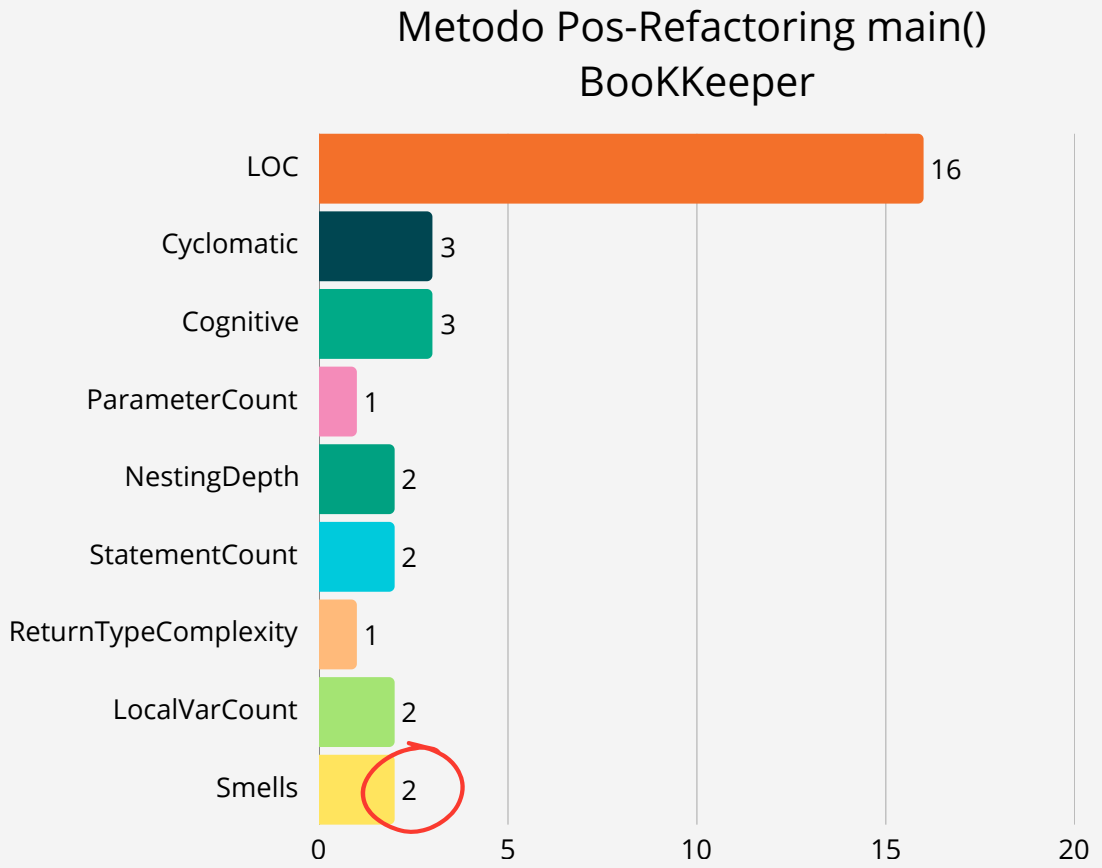
**Problema:** StatementCount elevato

**Strategia di Refactoring:**

1. ***Extract Method iterativo:*** Scomposizione in 3 unità logiche raggruppate per affinità dei dati (BasicState, FetchInfo, SequenceInfo).
2. ***Incapsulamento della Logica***: Delegata tutta la logica condizionale (if/else) ai sottometodi per "pulire" il flusso del metodo principale.

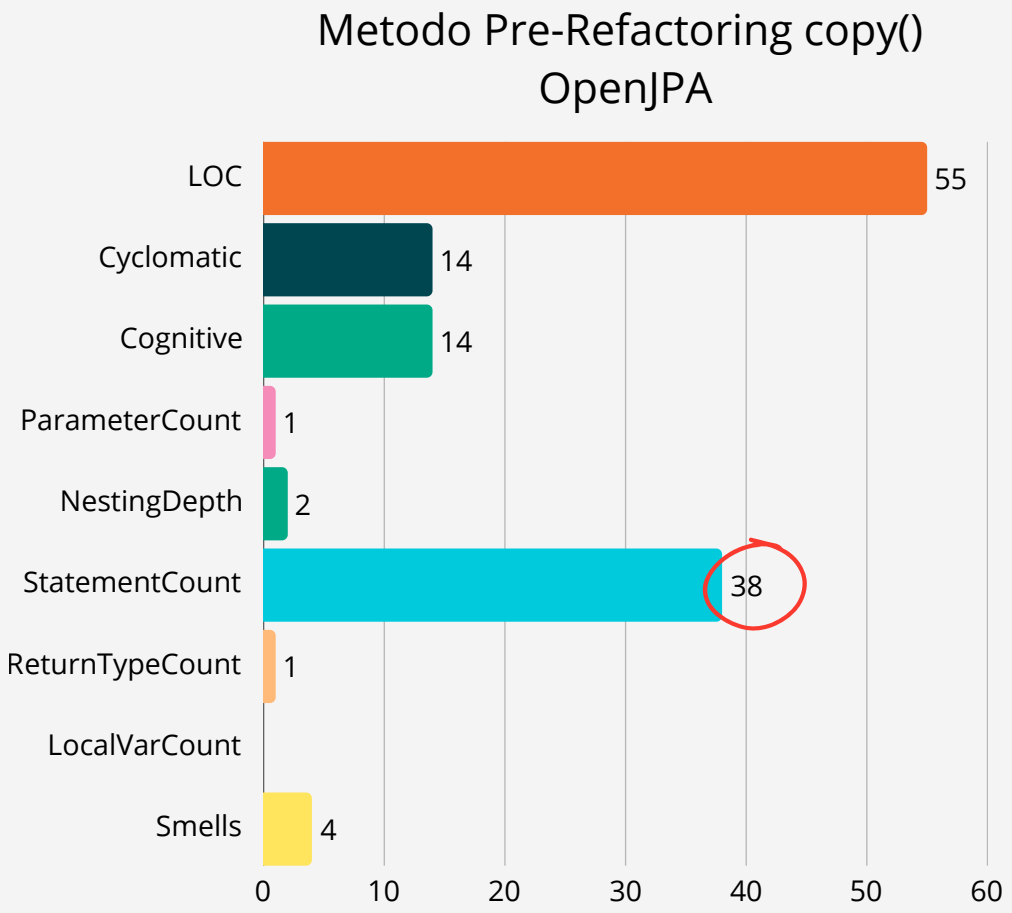
# Risultati e Validazione

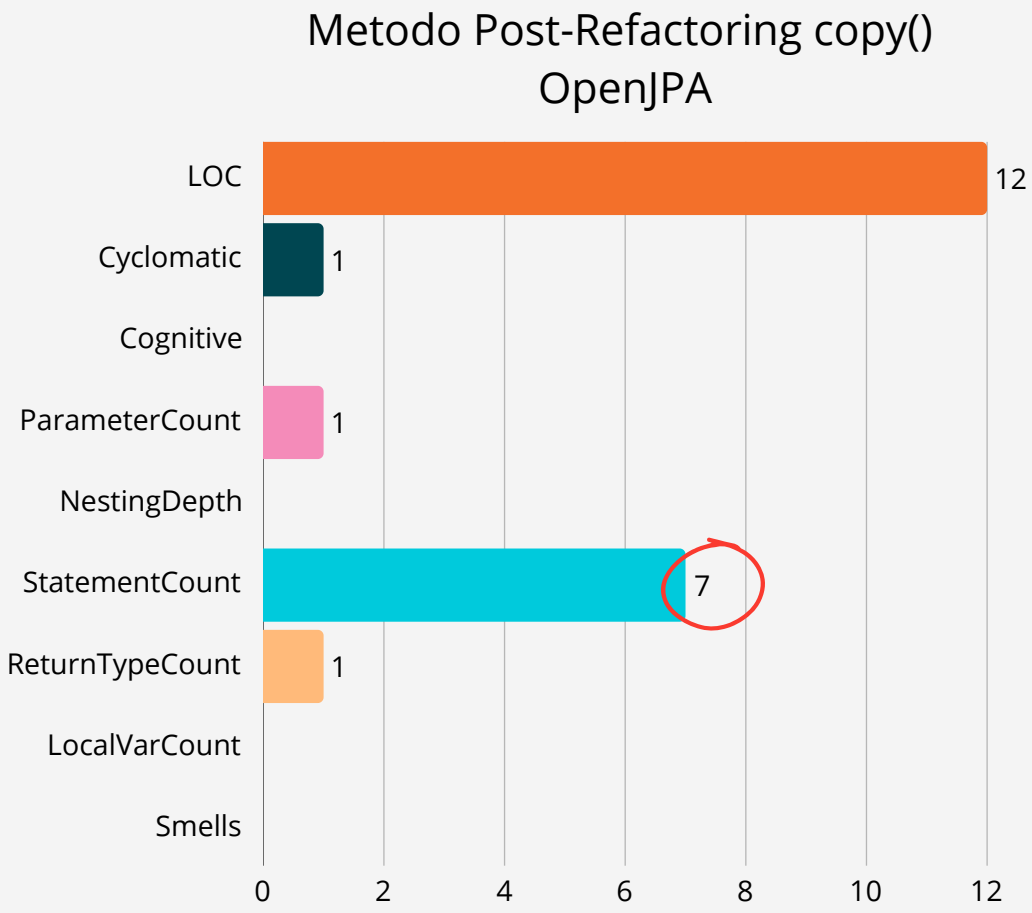



La permanenza di alcuni residui tecnici è frutto di una scelta consapevole: lo smell **Law of Demeter** persiste poiché l'ispezione delle catene di oggetti ZooKeeper è intrinseca al progetto e la sua rimozione tramite wrapper degraderebbe leggibilità e performance; parallelamente, lo smell **SignatureThrows** è stato mantenuto nel metodo main come vincolo architetturale necessario per la corretta propagazione e gestione dei fallimenti critici di sistema."

# Risultati e Validazione



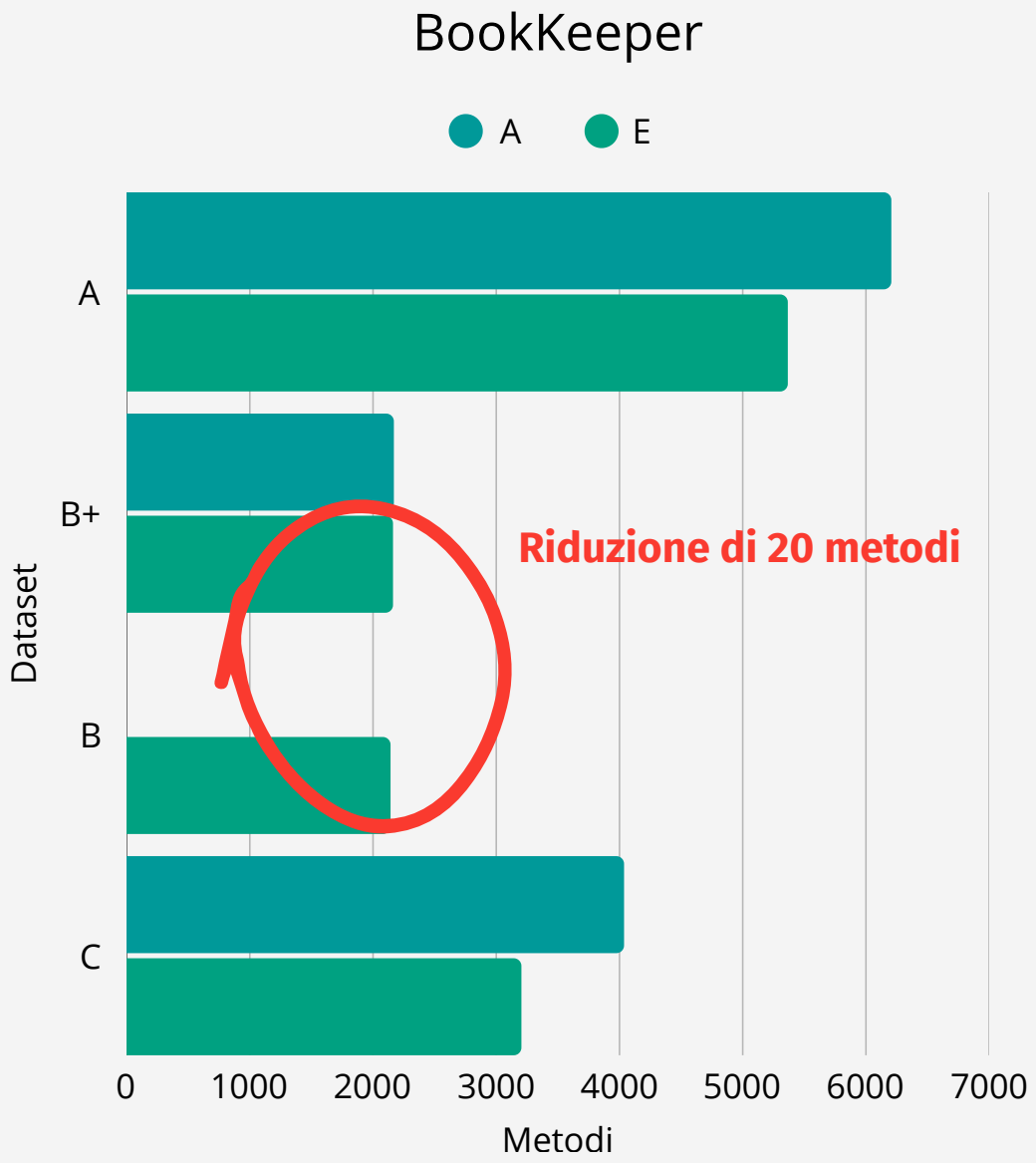
# Analisi What-IF: metodi buggy evitabili

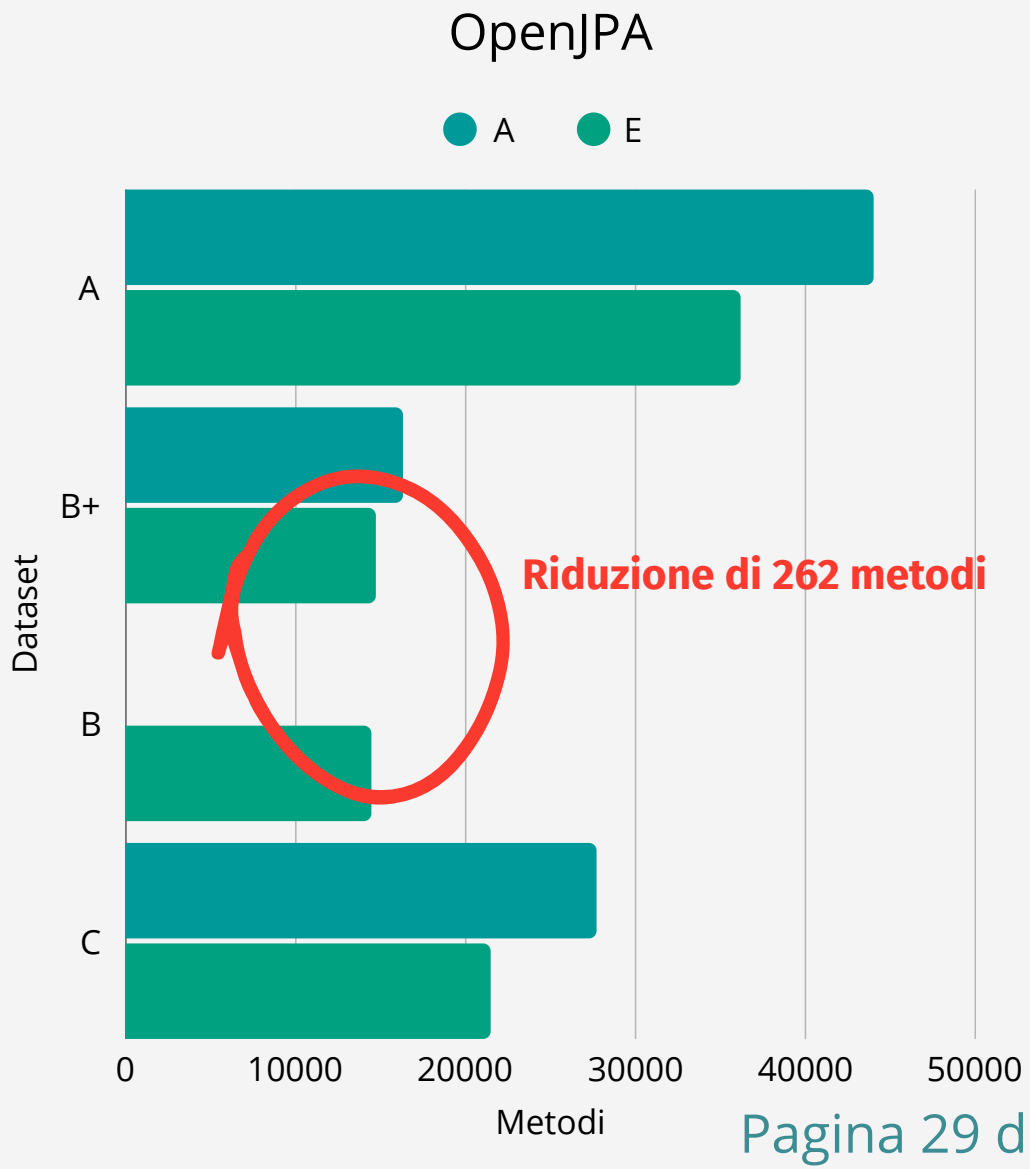
| Case Study | Risultato Post-Refactoring | Stato Finale |
|---|---|---|
| BookKeeper | L'obiettivo primario è raggiunto: il cuore del sistema è ora de-classificato come *non buggy*. l rischio non è più opaco e distribuito, ma confinato in componenti specifici (startBenchmark, handleChildrenChanged) pronti per testing mirato. | Complessità Isolata |
| OpenJPA | Abbattimento totale delle probabilità di bugginess su ogni nuovo metodo. | Validazione empirica della transizione a uno stato privo di criticità predetta. |

*La rimozione di una feature actionable innesca una reazione a catena che abbatte drasticamente tutte le metriche di complessità correlate.*

*Cosa accadrebbe se si avesse la possibilità di azzerare la feature Number of Smells?*

# Analisi What-IF: generalizzazione



# Analisi What-IF: generalizzazione

| Indicatore | Progetto: BookKeeper | Progetto: OpenJPA |
|---|---|---|
| Istanze Rimosse | **20** | **262** |
| Fattore Smell (Risk Increment) | **25,59%** | **16,40%** |
| Efficacia (rispetto a B) | 0,92% | 1,61% |
| Impatto Totale (rispetto a A) | 0,32% | 0,60% |

# Discussioni e Minacce alla Validità

# Risposte alle domande di ricerca

**RQ1: IBk** eccelle su **dataset ridotti** (**BookKeeper**), mentre **Random Forest** è superiore su **dataset ampi** (**OpenJPA**) per ranking ed efficienza.

**RQ2**: La rimozione dei **Code Smell** riduce il rischio relativo del **25,59%** (**BK**) e del **16,40%** (**OJP**), neutralizzando complessivamente **282 metodi difettosi**.

# Minacce alla Validità

1. **Interna**: Possibili errori nel tracciamento dei bug o bias manuale durante il refactoring.
2. **Costrutto**: Risultati legati alla precisione dei tool di analisi statica e dei classificatori scelti.
3. **Esterna**: Studio limitato all'ecosistema Apache e al linguaggio Java; la generalizzabilità ad altri contesti va verificata.

# Conclusioni e Sviluppi Futuri

# Conclusioni

**Impatto Qualitativo:** Il refactoring mirato dei Code Smell riduce la probabilità di bug futuri.

**Valore Operativo:** Ottimizzazione delle risorse di testing, concentrando gli interventi sulle componenti a maggior rischio.

# Sviluppi Futuri

**Ottimizzazione:** Esplorazione di tecniche wrapper per affinare la sensibilità dei modelli.

**Automazione:** Integrazione di tool per il suggerimento di refactoring smell-aware in tempo reale.

**Scalabilità**: Replicazione su contesti industriali e analisi longitudinali sulle release future.

GitHub repository per il codice sorgente:
**https://github.com/GaiaMeola/Progetto_ISW2**

Dettagli dell'analisi su SonarCloud:
**https://sonarcloud.io/project/overview?id=GaiaMeola_Progetto_ISW2**

# GRAZIE PER L’ATTENZIONE