TOR VERGATA
UNIVERSITÀ
DEGLI STUDI
DI ROMA

Macroarea di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Ingegneria del Software II (9 CFU) A.A. 2020/2021
Docenti Proff. Davide FALESSI, Guglielmo DE ANGELIS, Giuseppe F. CALAVARO

Modulo *Machine Learning for Software Engineering*

Deliverable II

Massimo STANZIONE
Matr. 0304936

Indice

- Introduzione – Contesto
- Progettazione
  - Modellazione del problema
  - Interfacciamento con JIRA
  - Metriche per la misurazione
- Variabili
  - Versioni
  - Tickets e bugs
- Analisi ML
  - Analisi dei valori e miglioramenti per i progetti considerati
- Analisi del codice
- Riferimenti

Introduzione – Contesto

- **Obiettivo**
  Presentazione dei risultati ottenuti dall'analisi ed applicazione di tecniche di sampling, classificazioni *cost-sensitive* e tecniche di feature selection su modelli di Machine Learning di due progetti open-source.
  Nel dettaglio, si considera come le tecniche citate agiscano sui risultati dei classificatori *NaiveBayes*, *Ibk* e *RandomForest*.
- **Progetti in analisi**
  - **Apache BookKeeper™**, servizio di storage ottimizzato in scalabilità e disponibilità per carichi di lavoro real-time
  - **Apache OpenJPA**, servizio di persistenza integrabile in container Java EE ed altri framework compatibili.
- **Ambienti di sviluppo**
  - Eclipse 2020-09 su Debian 9
  - IntelliJ IDEA 2021.2.3 su Linux Mint 20.2

Progettazione – Modellazione del problema

Il problema della creazione del dataset e della successiva analisi è stato affrontato **per fasi successive**, mediante l'ausilio delle repositories dei progetti disponibili su *GitHub*, dello strumento di issue tracking *JIRA* e dello strumento di Machine Learning *Weka*.

Le fasi previste ed eseguite sono state le seguenti:

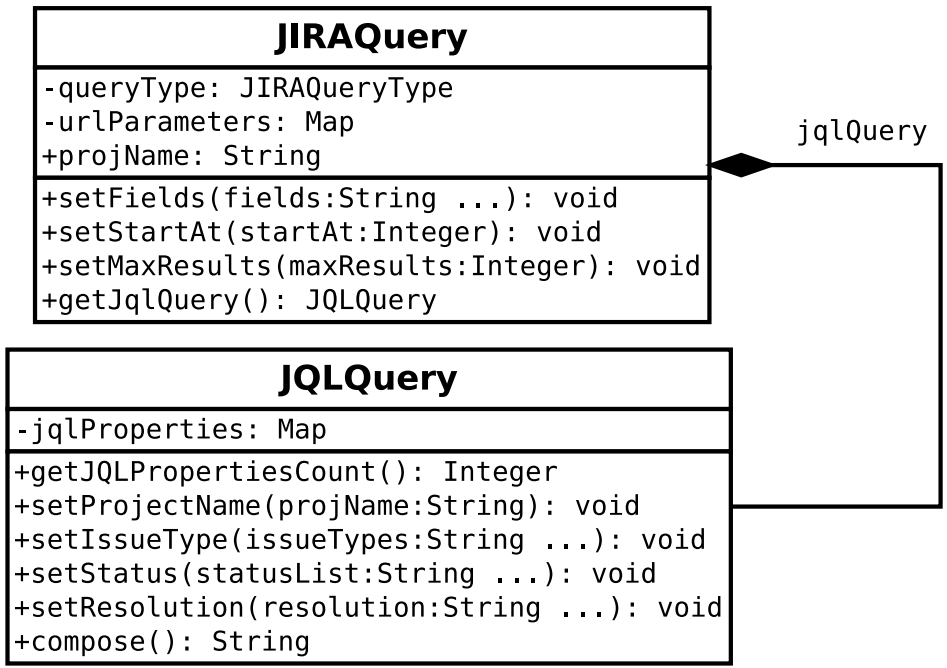
1. Generazione locale di una ***working copy*** del progetto;
2. Fetching dei **commit** del progetto;
3. Fetching delle **versioni** del progetto;
4. Analisi dei **ticket** relativi a bug di tipo *fixed*;
5. Preparazione dei **dataset** per il modello di Machine Learning;
6. **Analisi** Machine Learning

Progettazione – Interfacciamento con JIRA

Le informazioni relative ai ticket sono state prelevate dal sistema di issue tracking JIRA, mediante le API da esso esposte.

Dal punto di vista implementativo, tale operazione è stata effettuata mediante predisposizione di **due apposite classi**, `JIRAQuery` e `JQLQuery`, tramite le quali sono state manipolate le queries in linguaggio JQL (JIRA Query Language) impiegate per il prelievo delle informazioni.

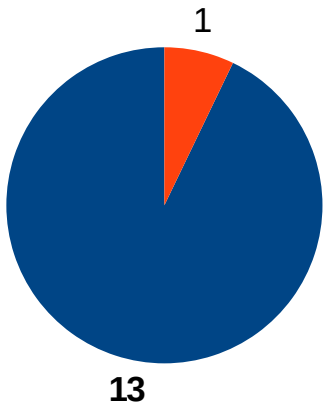
I risultati, in formato JSON, sono poi stati manipolati con l'ausilio di una classe similmente predisposta, `JSONHandler`.



Variabili – Versioni

Per ogni progetto sono state considerate le versioni disponibili, scartando quelle che ricadono in almeno una delle seguenti categorie:

- Non corrispondenza tra JIRA e GitHub;
- Stato “non finale”, es. versioni “beta”.

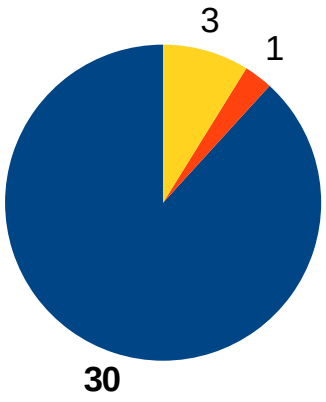


Apache BookKeeper™

Matching JIRA-GitHub

Non presenti in GitHub

Le versioni così ottenute rappresentano comunque **la maggioranza** delle versioni presenti.



Apache OpenJPA

Matching JIRA-GitHub

Non presenti in GitHub

Non finali

Variabili – Tickets e bugs

Particolare attenzione è stata posta nella ricostruzione e **computazione del ciclo di vita** dei bug.

Le informazioni sulle versioni riferite dai ticket, in particolare per le *Affected Versions*, hanno dato spesso luogo a **contraddizioni ed inconsistenze** nello svolgimento del progetto.

Per tale motivo sono state individuate ed analizzate le seguenti problematiche:

- **`AV_AFTER_FV`**: esistono versioni indicate come AV ma poste oltre la FV;
- **`AVS_NOT_CONSISTENT`**: le AV riportate non sono susseguenti tra la IV e la FV;
- **`IV_AFTER_OV`**: la IV riportata è successiva alla OV;
- **`FV_AS_AV`**: la FV è riportata anche nell'elenco delle AV;
- **`NOT_REPORTED`**: non vi è alcuna AV riportata.

Le casistiche sono documentate nella enumerazione `JIRAAffectedVersionsCheck`.

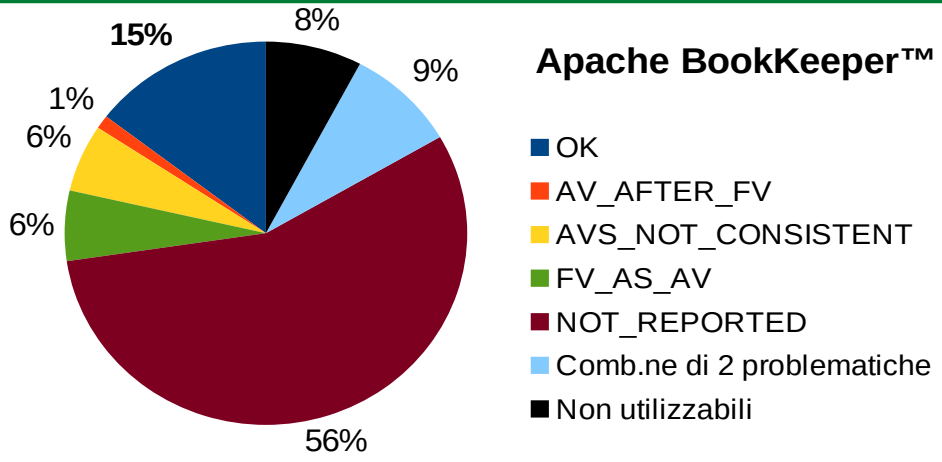
Variabili – Tickets e bugs

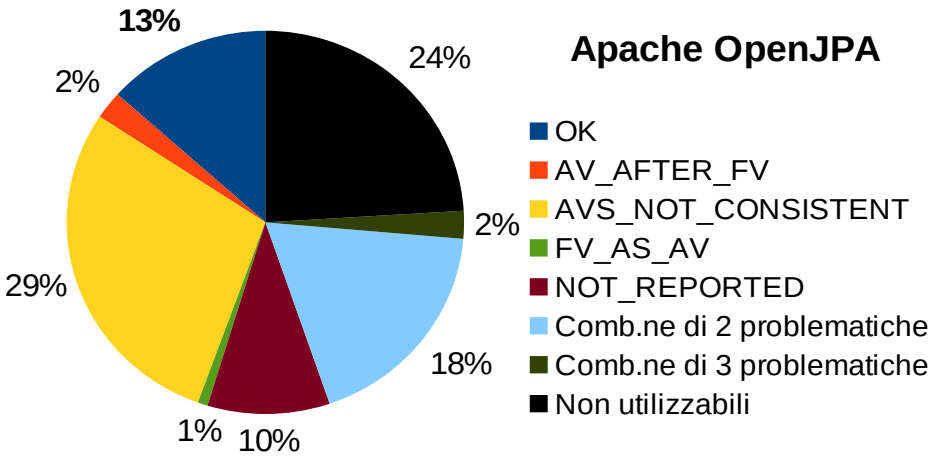
Si è notato che **percentuali molto ridotte** delle AV riportate non presentavano alcun problema tra quelli indicati.

Inoltre, `IV_AS_OV` è presente sempre in combinazione con altre problematiche.

Ove possibile si è **ricostruito** il ciclo di vita mediante predizione della IV o altre tecniche di compensazione, scartando i ticket per i quali ciò non sia stato possibile.

È stato predisposto uno **strumento di ispezione** per la visualizzazione immediata del ciclo di vita e delle problematiche, reperibile in *`graphicBugLifecycleVisualizer.csv`*.





Progettazione – Metriche per la misurazione

Al fine della creazione del dataset, si è deciso di effettuare la misurazione delle classi dei progetti in analisi secondo le metriche di seguito riportate.

La scelta delle metriche è stata effettuata in base alle tre differenti aree di azione ad esse relative, in modo da poter ottenere un modello il più possibile vario e comprensivo di ogni aspetto.

Specifiche delle versioni	Codice	Statistiche sulle variazioni del codice
• **NR** numero di commits • **NAuth** numero di autori coinvolti • **ChangesetSize** dimensione degli insiemi di modifiche	• **Size** LOC della classe • **LOC_Added** totale delle LOC aggiunte • **LOC_Touched** totale delle LOC aggiunte/rimosse	• **MaxLOCAdded** massimo numero di LOC aggiunte • **AvgLOCAdded** numero medio di LOC aggiunte

Progettazione – Analisi ML

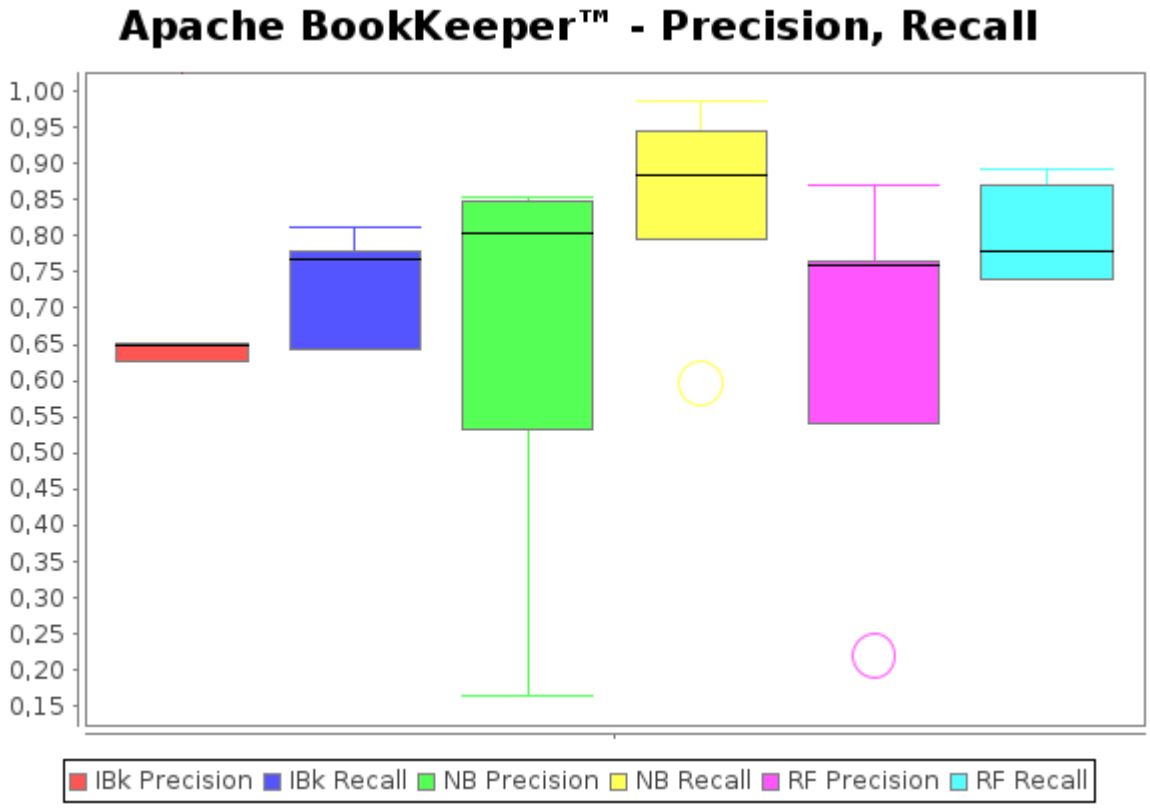
Si procede all'applicazione della tecnica *Walk Forward* sui dataset prodotti, mediante le Java API offerte dal software *Weka*, sviluppato dall'Università di Waikato (Nuova Zelanda).

- **Obiettivo**:
  Si considerano, nel dettaglio, le prestazioni dei classificatori considerati nelle metriche *Precision*, *Recall*, *AUC*, *Kappa*.
- **Implementazione**:
  A tale scopo sono state introdotte le dipendenze `weka-stable` e `SMOTE` dal pacchetto `nz.ac.waikato.cms.weka` nel file **POM Maven** allegato al progetto.
- **Problematiche**:
  Durante lo sviluppo del progetto si è notato che la versione 1.0.2 della dipendenza `SMOTE` presentava una problematica, documentata in forum dedicati, che portava a continui sollevamenti di `IllegalStateException` durante l'analisi del classificatore *RandomForest*. Tale problema è stato risolto considerando la **versione 1.0.3**, che pone soluzione al bug associato.

Apache BookKeeper™: Precision, Recall

I valori di *Precision* e *Recall* si attestano stabilmente sopra la soglia 0.5, dato anche il limitato sbilanciamento del dataset verso le classi “non-buggy”.

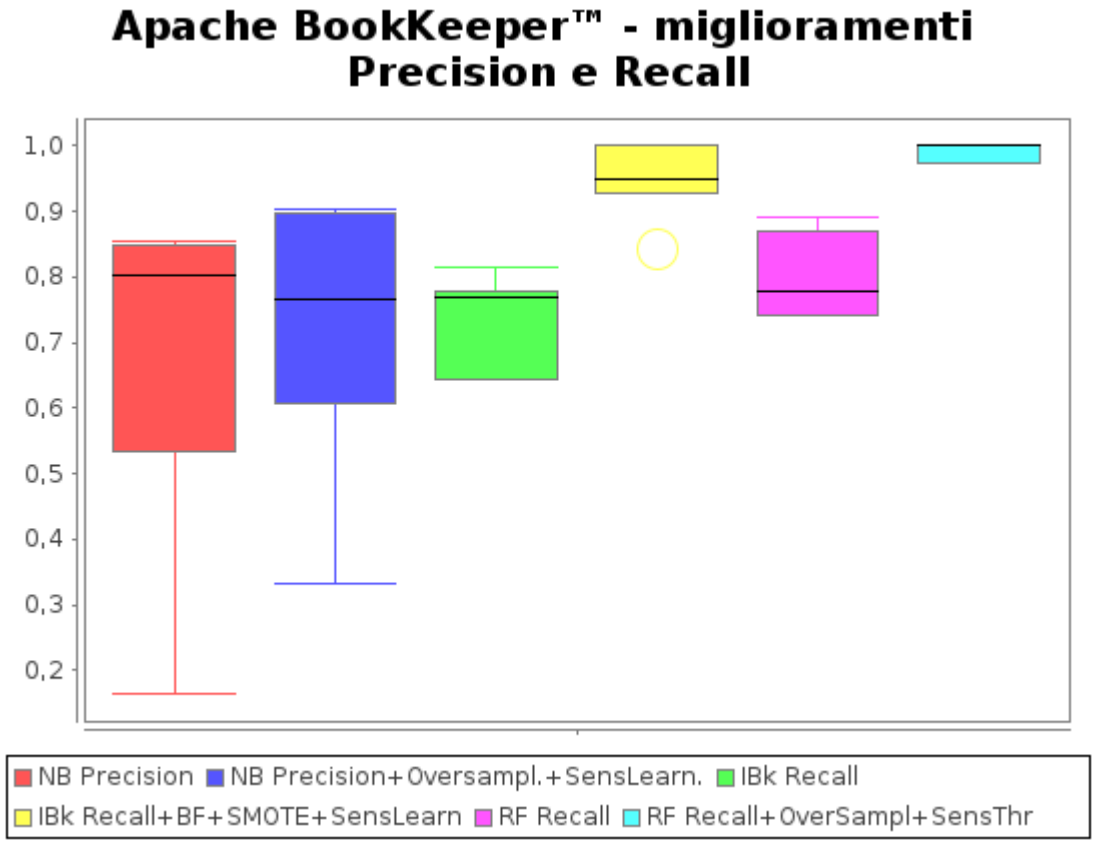
Il valore di *Recall* per il classificatore *NaiveBayes* risulta avere valori elevati, seppur in presenza di un outlier nel valore 0.56.



Apache BookKeeper™: Precision/Recall, migliorie

In presenza di tecniche di *Feature selection*, di *balancing* e in limitati casi di *cost sensitive classifying*, si ottengono miglioramenti anche importanti per le metriche in esame: il balancing, in particolare, agevola l'equilibrio tra le istanze.

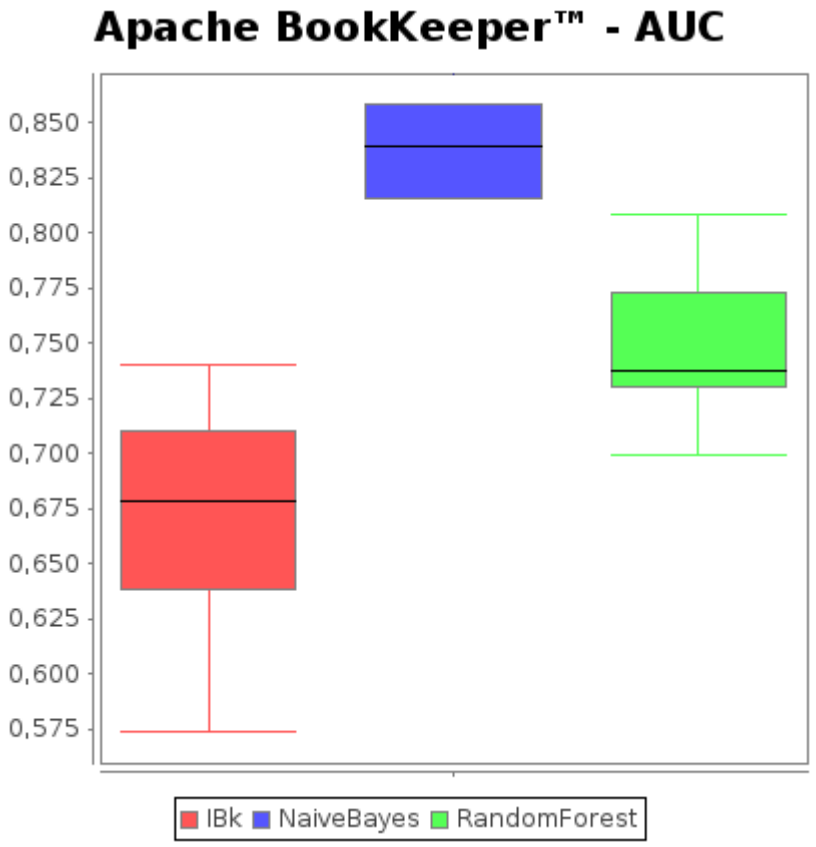
I miglioramenti sono più marcati sia per la *Recall* che per la *Precision*, in particolare per la tecnica *SMOTE*.



Apache BookKeeper™: AUC

Ad una prima analisi, i valori di *AUC* sembrano mantenersi al di sopra di quelli di un classificatore *dummy*, con un comportamento nettamente migliore per il classificatore *NaiveBayes*.

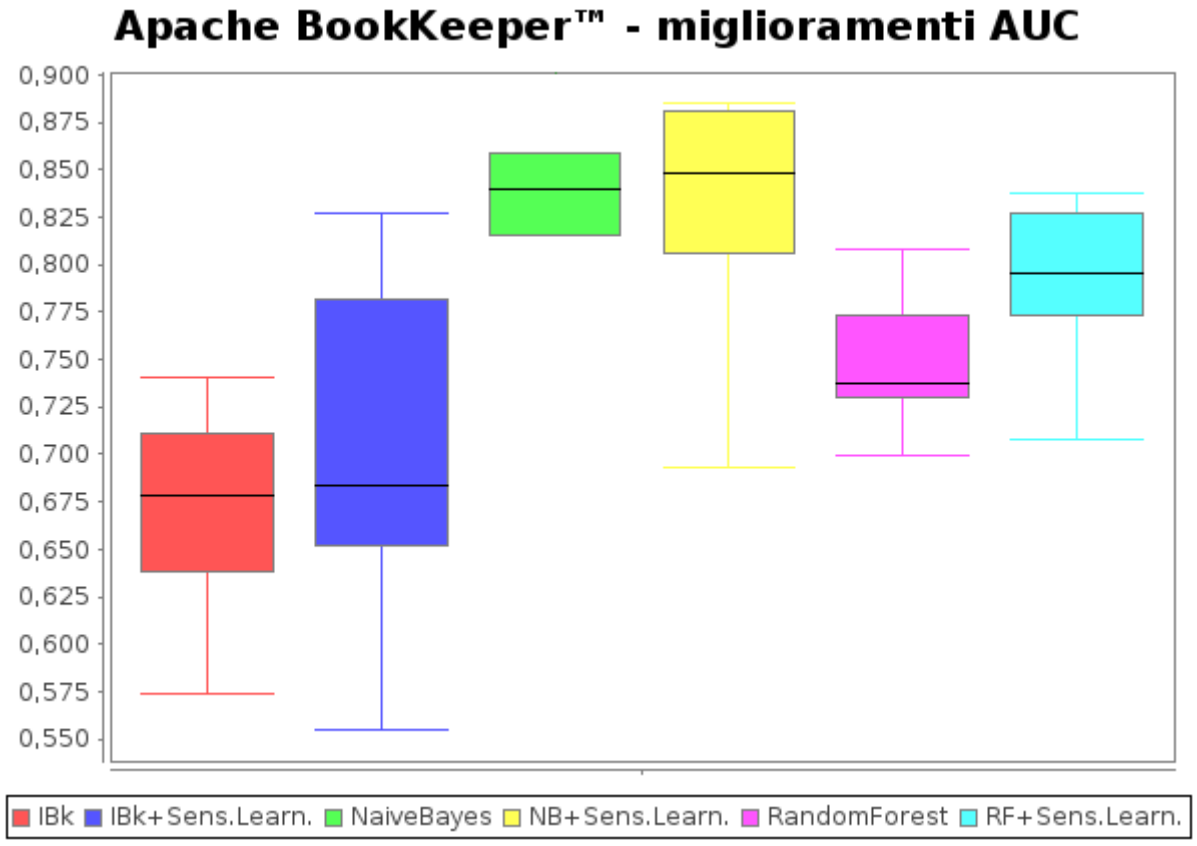
Il classificatore *IBk*, invece, in alcune iterazioni della tecnica Walk Forward, ha presentato valori prossimi a quelli di un classificatore *dummy*.



Apache BookKeeper™: AUC, migliorie

In seguito all'applicazione delle tecniche di *Feature selection*, *balancing* e *sensitivity*, non si notano particolari migliorie per la caratteristica *AUC*, eccezion fatta per *RandomForest* che vede aumentare il proprio valore mediano.

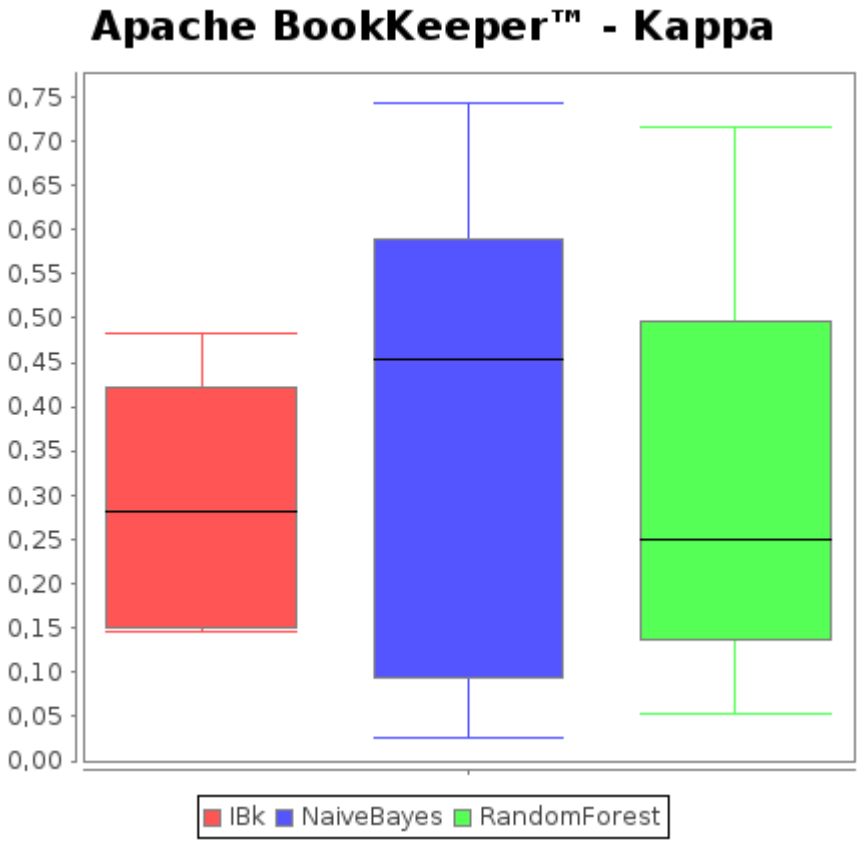
Ciò risulta essere conforme alle aspettative, in quanto *AUC* non è particolarmente sensibile alle tecniche di *cost sensitivity*.



Apache BookKeeper™: Kappa

Per quanto riguarda la metrica Kappa, si nota che ciascuno dei classificatori della terna considerata espone un comportamento sempre migliore rispetto ad un classificatore *dummy*: non sembra esistere alcuna iterazione *Walk Forward* in cui Kappa assume valori negativi.

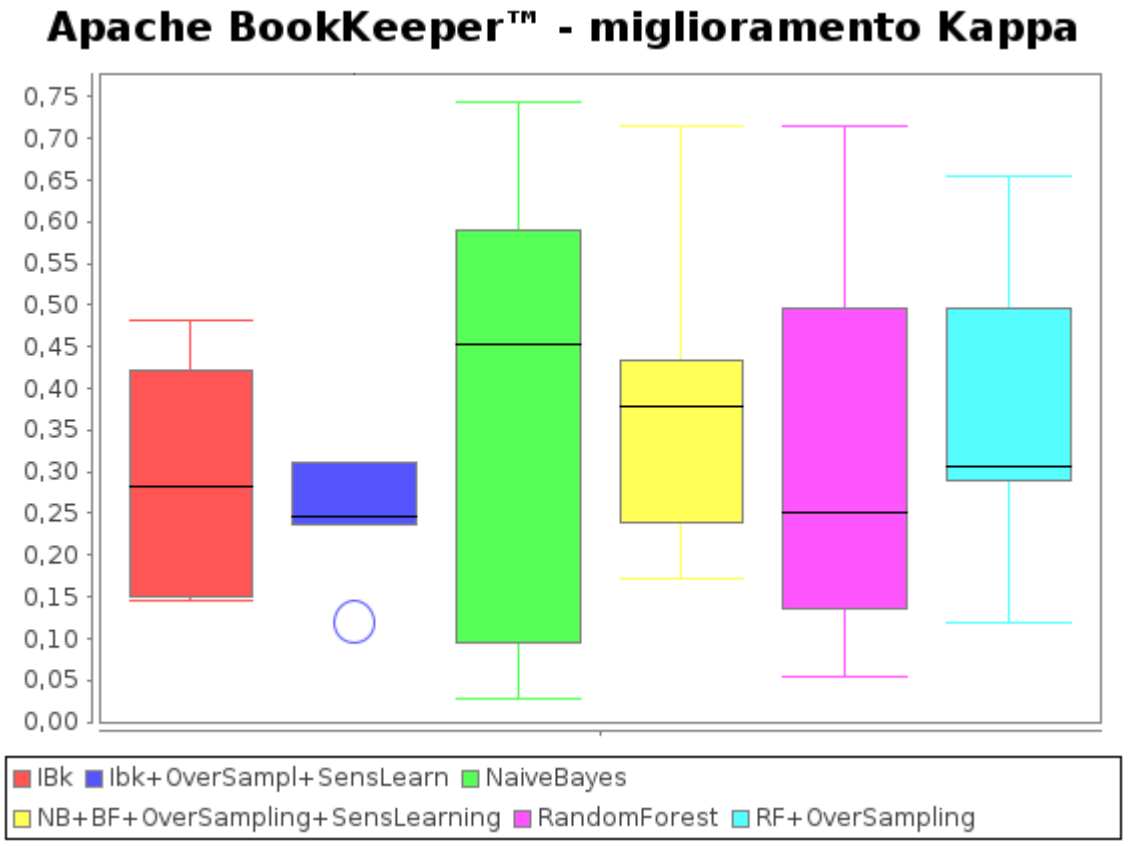
Il classificatore *NaiveBayes* si distingue nettamente dai due restanti, con un valore mediano quasi doppio.



Apache BookKeeper™: Kappa, migliorie

Dai dati sperimentali non emergono significative variazioni positive del valore della metrica in seguito all'applicazione delle tecniche in esame.

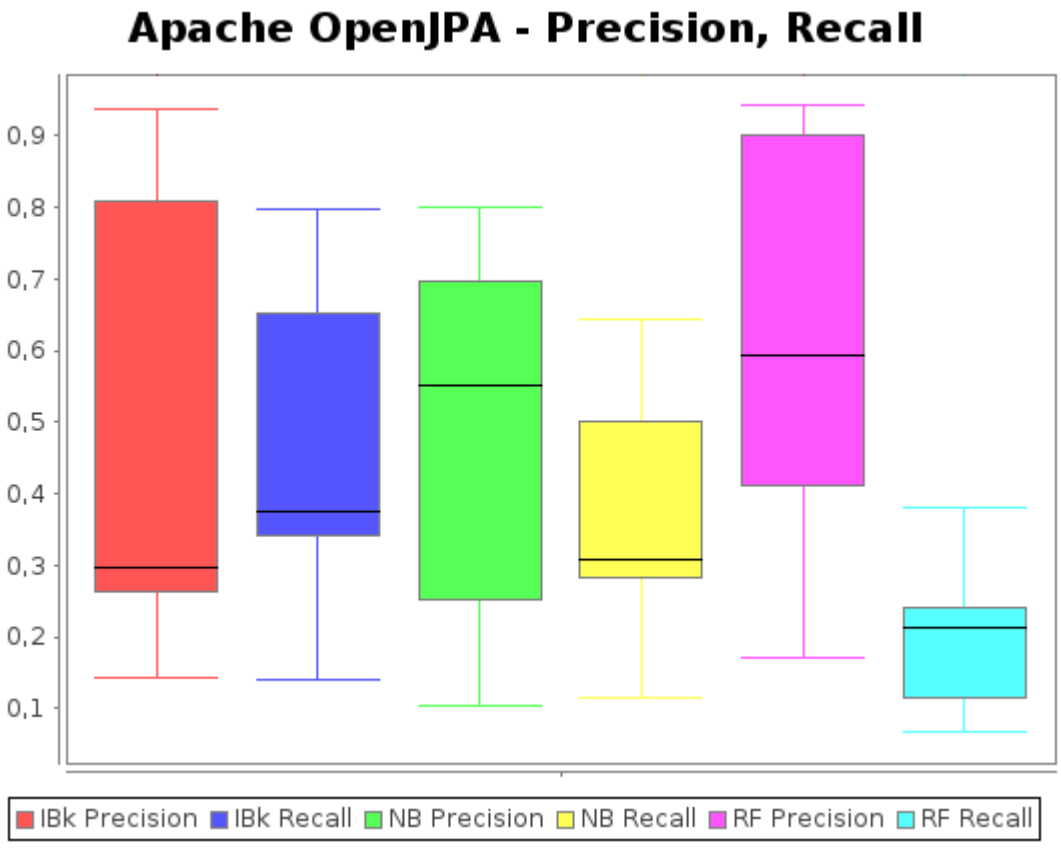
Si osserva, invece, un lieve calo delle prestazioni del classificatore *NaiveBayes* in presenza della tecnica di feature selection *BestFirst*.



Apache OpenJPA: Precision, Recall

I valori di *Precision* e *Recall* indicano difficoltà nel riconoscimento di classi “buggy”, circostanza attesa dal momento che il dataset è lievemente sblianciato verso le classi “non-buggy”.

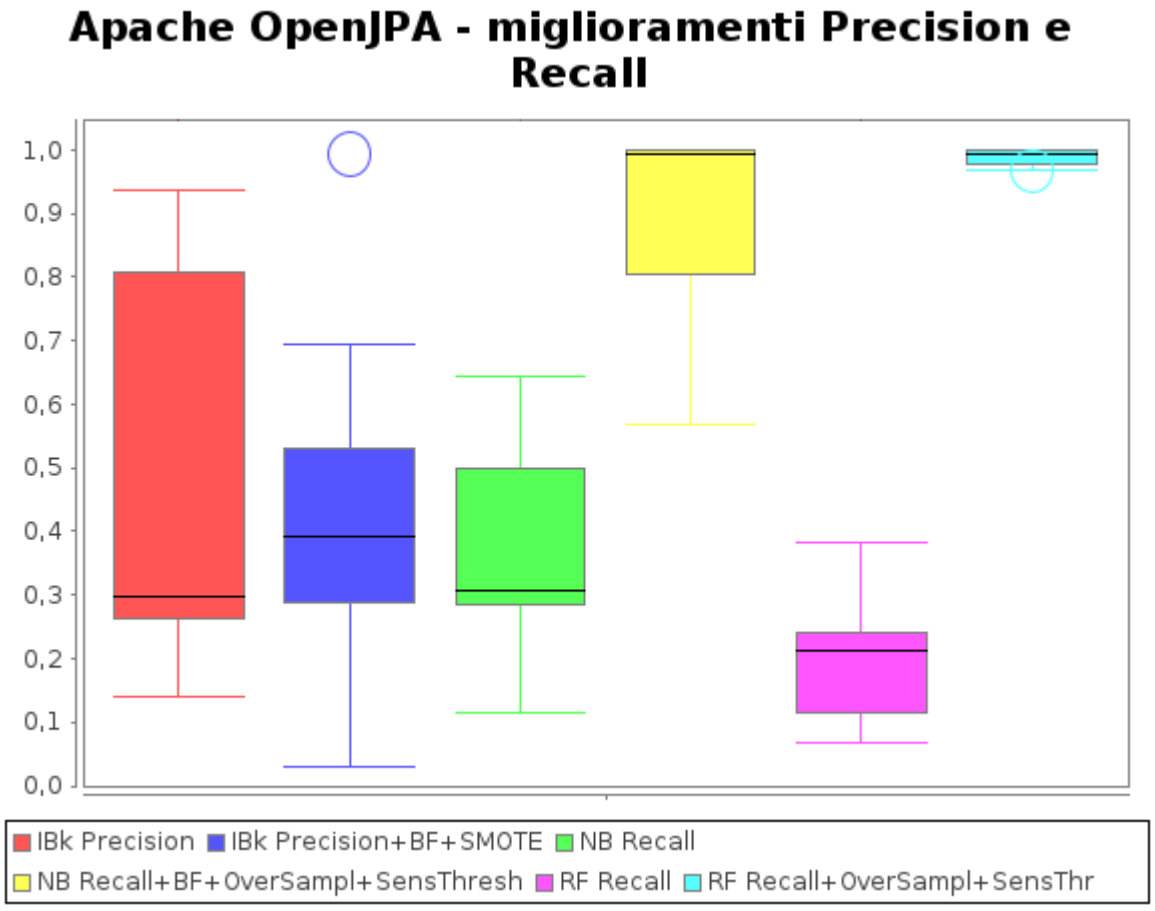
Il valore di *Recall* per il classificatore *RandomForest*, che nel progetto precedente aveva valori più accettabili, è qui la minore, con un valore mediano pari a 0.211.



Apache OpenJPA: Precision/Recall, migliorie

I miglioramenti sembrano piuttosto contenuti rispetto ai valori di *Precision*, con anche lieve flessioni come quella causata da *OverSampling* al classificatore *NaiveBayes*. Il classificatore che più trae beneficio dalle tecniche è *IBk*, soprattutto in combinazione con *BestFirst* e *SMOTE*.

La *Recall*, invece, risente positivamente dei metodi di analisi: per *RandomForest*, attenzionato nella slide precedente, è nettamente migliorata, ed il valore migliore è ottenuto con la stessa combinazione di tecniche che nel progetto precedente portava a questo risultato, visibile in legenda.

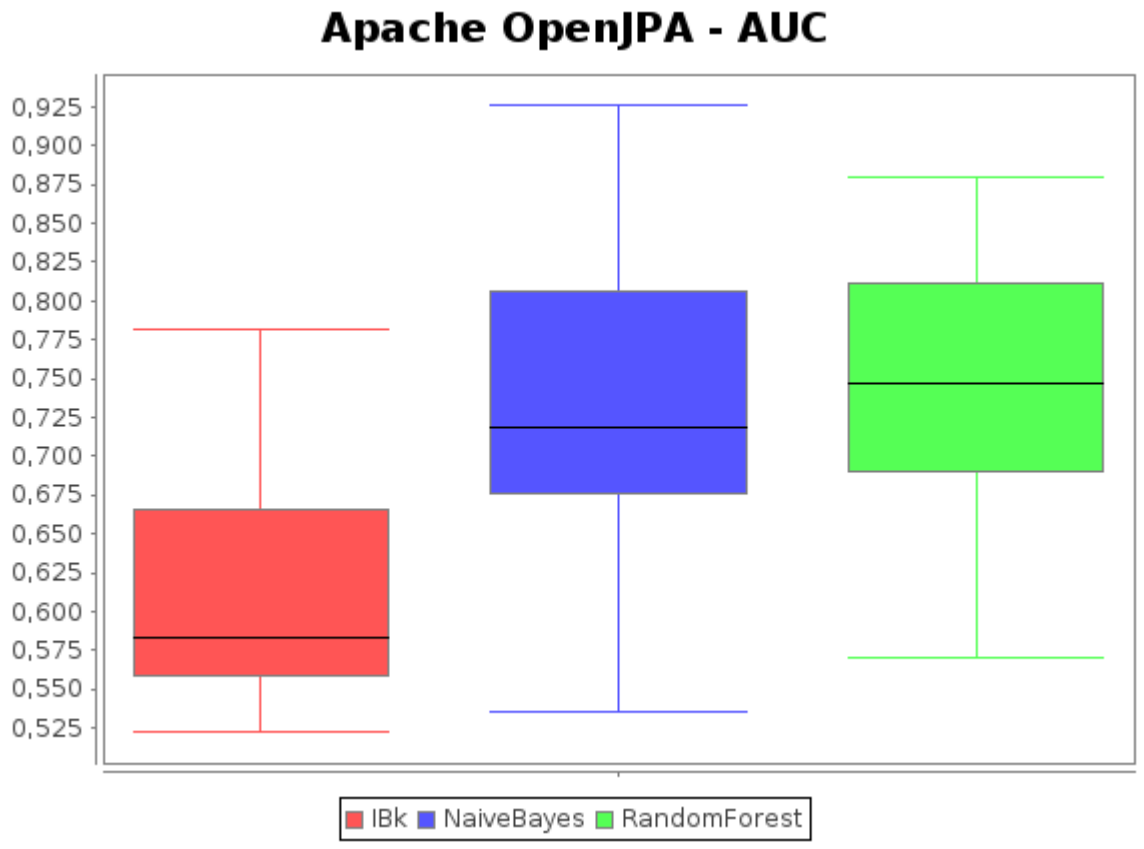


Apache OpenJPA: AUC

Similmente al progetto precedente, i valori dei tre classificatori si mantiene al di sopra della soglia del classificatore *dummy*.

Le prestazioni di *RandomForest* sono lievemente superiori, per valore mediano, a quelle di *NaiveBayes*.

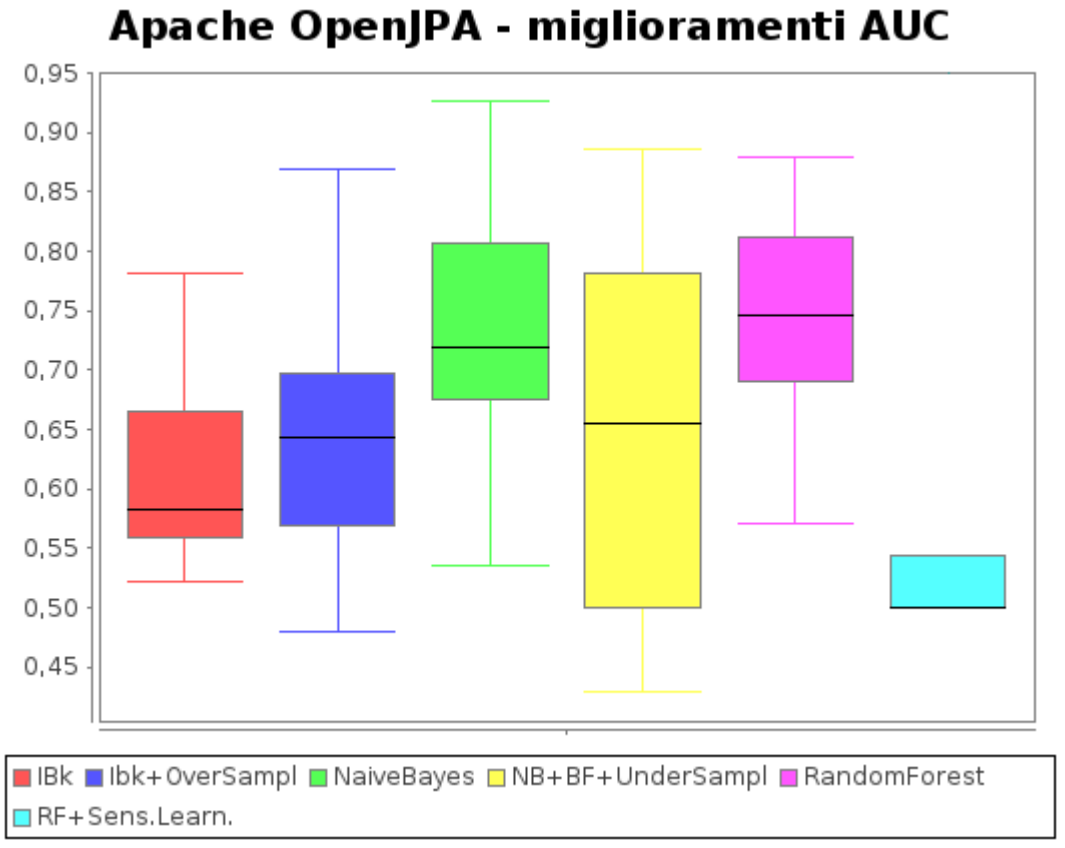
Il classificatore *IBk* registra le prestazioni peggiori.



Apache OpenJPA: AUC, migliorie

Applicazioni delle tecniche in esame al fine di migliorare i valori di *AUC* portano a variazioni tutto sommato contenute.

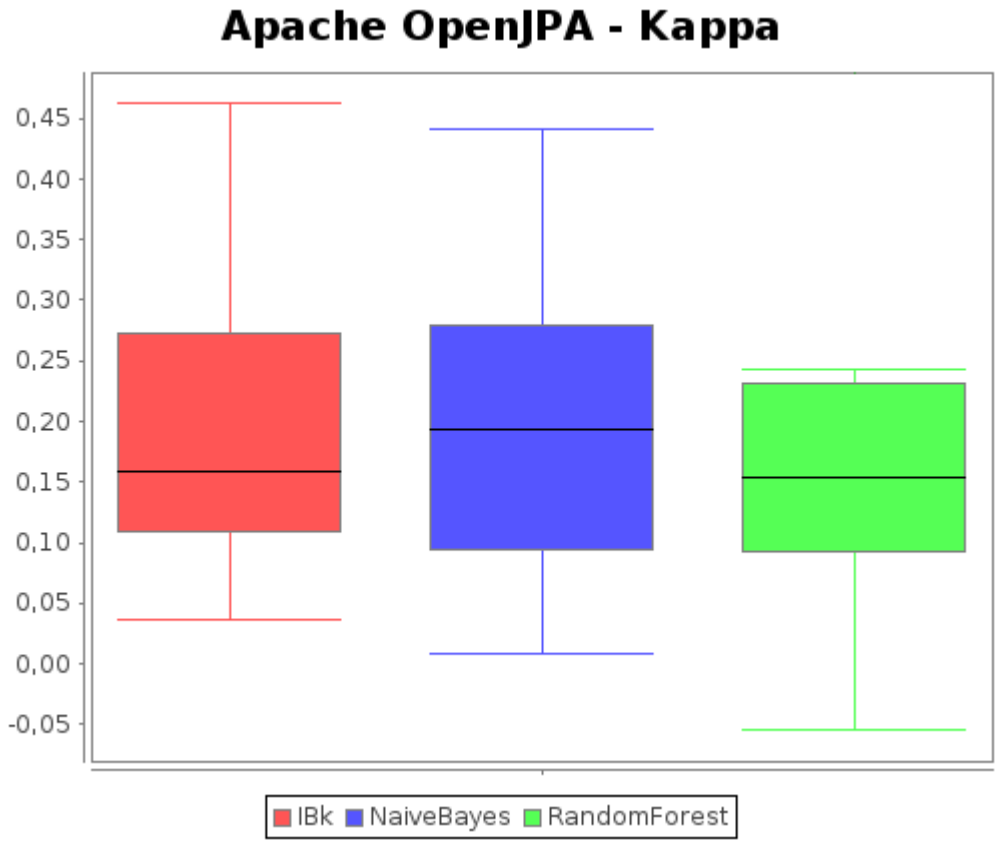
Unica eccezione è una flessione nell'applicazione della tecnica di *Featuring selection BestFirst* al classificatore *RandomForest*.



Apache OpenJPA: Kappa

I valori di *Kappa* si mantengono globalmente al di sopra del valore nullo. Le differenze tra i valori mediani sono piuttosto contenute.

Si nota una iterazione *Walk Forward* per il classificatore che, con un valore pari a -0.05, ha individuato un comportamento peggiore rispetto ad un classificatore *dummy*.

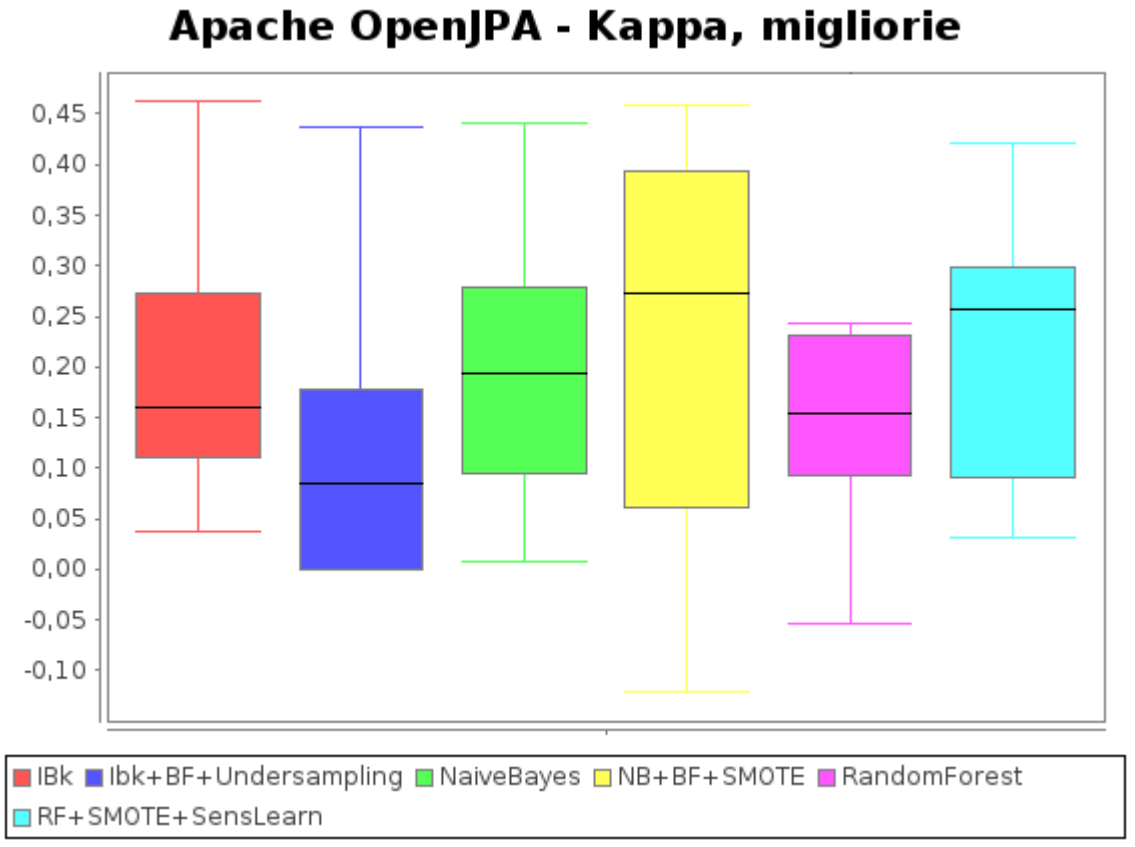


Apache OpenJPA: Kappa, migliorie

Le variazioni dei parametri in analisi, in seguito alle applicazioni delle tecniche studiate, subiscono variazioni contenute.

Nel dettaglio, si nota una flessione nell'applicazione della tecnica *BestFirst* al classificatore *IBk*, riportata in grafico in combinazione con *Undersampling*.

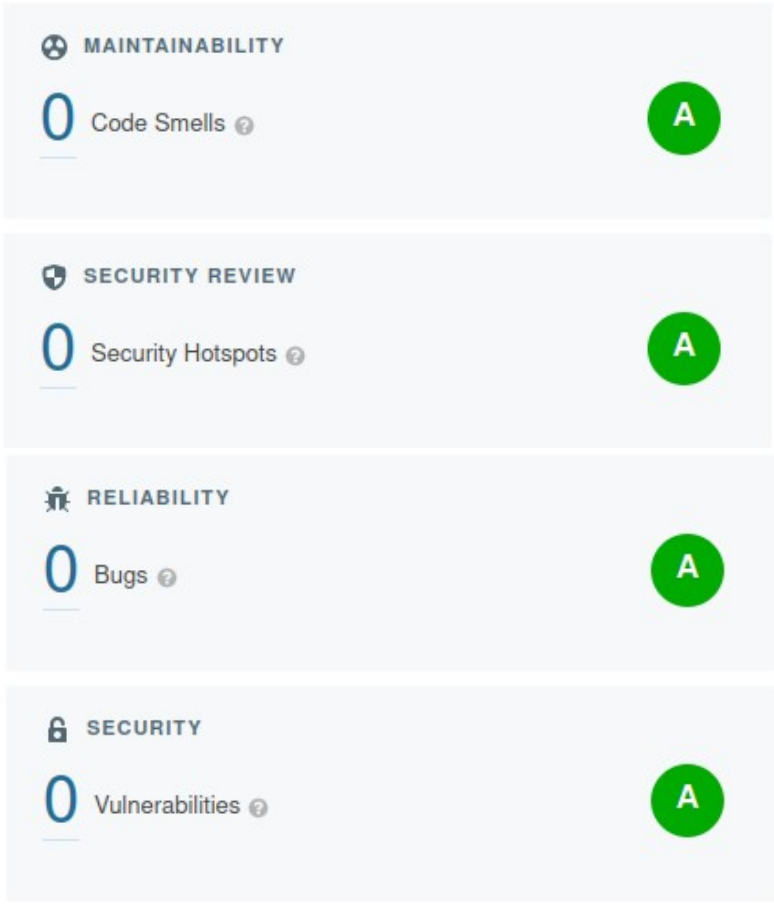
La stessa tecnica, in combinazione con *SMOTE*, porta un miglioramento in *NaiveBayes*.



Analisi del codice

Il codice è stato analizzato mediante *SonarCloud*, e l'analisi è stata **inclusa nel ciclo di build del progetto**, utilizzando *Maven* per la gestione delle dipendenze e *CircleCI* come strumento di CI/CD.

In seguito ad un processo di revisione del codice i bug, le code smells ed i problemi di sicurezza sono stati azzerati.



Riferimenti

- Sito del progetto Apache BookKeeper™:
  https://bookkeeper.apache.org
- Sito del progetto Apache OpenJPA:
  https://openjpa.apache.org
- Pagina JIRA per Apache BookKeeper™:
  https://issues.apache.org/jira/projects/BOOKKEEPER/issues
- Pagina JIRA per Apache OpenJPA:
  https://issues.apache.org/jira/projects/OPENJPA/issues
- Repository della deliverable:
  https://github.com/massimostanzione/isw2-deliverable2
- Analisi *SonarCloud*:
  https://sonarcloud.io/summary/overall?id=massimostanzione_isw2-deliverable2