Machine Learning for Software Engineering

Alessandro Lioi, 0333693

Università di Roma Tor Vergata

A.A. 2022/2023

Indice

Contesto

Nell'ingegneria del software, per garantire una qualità del prodotto che si sta sviluppando, è necessaria una fase di testing.

Poichè la fase di testing può risultare molto costosa, è necessario incentrare lo sviluppo dei test solo su alcune classi del sistema

Problema: come individuare queste classi che verosimilmente contengono bug?

Obiettivo

L'obiettivo dello studio è quindi quello di rendere l'attività di testing efficiente ed efficace su un sottoinsieme di classi opportunamente scelti sui progetti **Avro** e **BookKeeper**

Fasi del Progetto

1. Individuare le classi che sono state *buggy* nel passato
2. Calcolare delle metriche su queste classi per costruire un dataset
3. Costruzione del dataset, tramite la tecnica *walk-forward*
4. Utilizzare, in maniera black-box, vari classificatori per determinare quale offre le performance e le predizioni migliori

Terminologia

È necessario introdurre alcuni concetti fondamentali:

- *Revisione*: corrisponde ai commit sul version control system
- *Release* (o *Versione*): revisione che viene considerata stabile
- *Ticket*: creato su un software di bug tracking, rappresenta un lavoro da svolgere; caratterizzato dal tipo
- *Injected Version* (IV): revisione in cui il bug è stato inserito
- *Opening Version* (OV): revisione in cui si è aperto il ticket relativo
- *Fixed Version* (FV): revisione in cui il ticket è stato chiuso (bug risolto)
- *Affected Versions* (AV): insieme di revisioni in cui il bug è presente

1. Individuazione classi Buggy

Per individuare le classi buggy, si usano le informazioni presenti in:

- Git: version control system (raccolta revisioni e release)
- Jira: issue tracking system (raccolta ticket e versioni del ciclo di vita di un bug)

I ticket di Jira considerati sono stati quelli *risolti* con tipo uguale a *Bug*

Dai ticket sono state considerate le OV e FV (sempre presenti) e, qualora fossero disponibili, le AV (dalle quali è possibile ottenere la IV)

1. Caso AV assente

Nel caso in cui le AV non fossero presenti su Jira, è stata applicata la tecnica di **proportion**: $P = \frac{\text{FV} - \text{IV}}{\text{FV} - \text{OV}} \rightarrow \text{IV} = \text{FV} - (\text{FV} - \text{OV}) * P$

In particolare è stata applicata la variante **incremental**: si usa la media tra il proportion dei ticket, considerati fino alla versione precedente

Nel caso in cui la media fosse calcolata su una versione con meno di 5 ticket, è stato usato il valore di **proportion cold-start** (mediana della media di P su altri progetti simili)

I progetti simili considerati fanno sempre parte dell'Apache Software Foundation

Una volta ottenuti i valori di IV, OV e FV, è stato effettuato un mapping tra le versioni di Jira e le revisioni di Git, attraverso due filtri:

- *semantico*: si considera la revisione che corrisponde ad uno dei seguenti pattern:
  - `BookKeeper _version name_ release`: per BookKeeper
  - `Tag _version name_`: per Avro
- *data*: nel caso in cui il primo filtro fallisse, si considera l'ultima revisione del giorno di rilascio (o il successivo) indicato su Jira

Ottenute le revisioni, è stato possibile avere una lista delle classi *toccate*

Size	Linee di codice
LoC Touched	Linee di codice aggiunte e eliminate
Average LoC Added	Media LoC aggiunte tra due release
Max LoC Added	Massimo LoC aggiunte tra due release
Churn	Differenza tra linee aggiunte e eliminate
Average Churn	Churn medio tra due release
Max Churn	Churn massimo tra due release
NR	Numero di revisioni tra due release
NFix	Numero di ticket risolti tra due release
NAuth	Numero di Autori

Una volta calcolate le metriche, è stato possibile costruire i dataset necessari secondo la tecnica di validazione **time-series** *walk forward*

Considerando una generica release k

- training set: contiene le metriche fino alla versione $k-1$
- testing set (*oracolo*): contiene le metriche della versione k, sfruttando tutti i dati disponibili

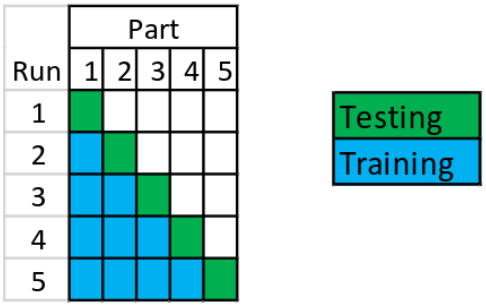


Figure 1: Esempio Walk-Forward

I classificatori utilizzati, tramite lo strumento **Weka**, sono **Naive Bayes**, **Random Forest**, **IBk**

Le metriche di performance analizzate sono:

- Recall: quanti dei positivi è stato possibile classificare
- Precision: quante volte è stato possibile classificare correttamente una istanza come positiva
- Kappa: quante volte si è stati più accurati di un classificatore *dummy*
- AUC (*area under the curve*): probabiltià che una istanza positiva, scelta casualmente sia classificata sopra una istanza negativa scelta casualmente

4. Tecniche aggiuntive

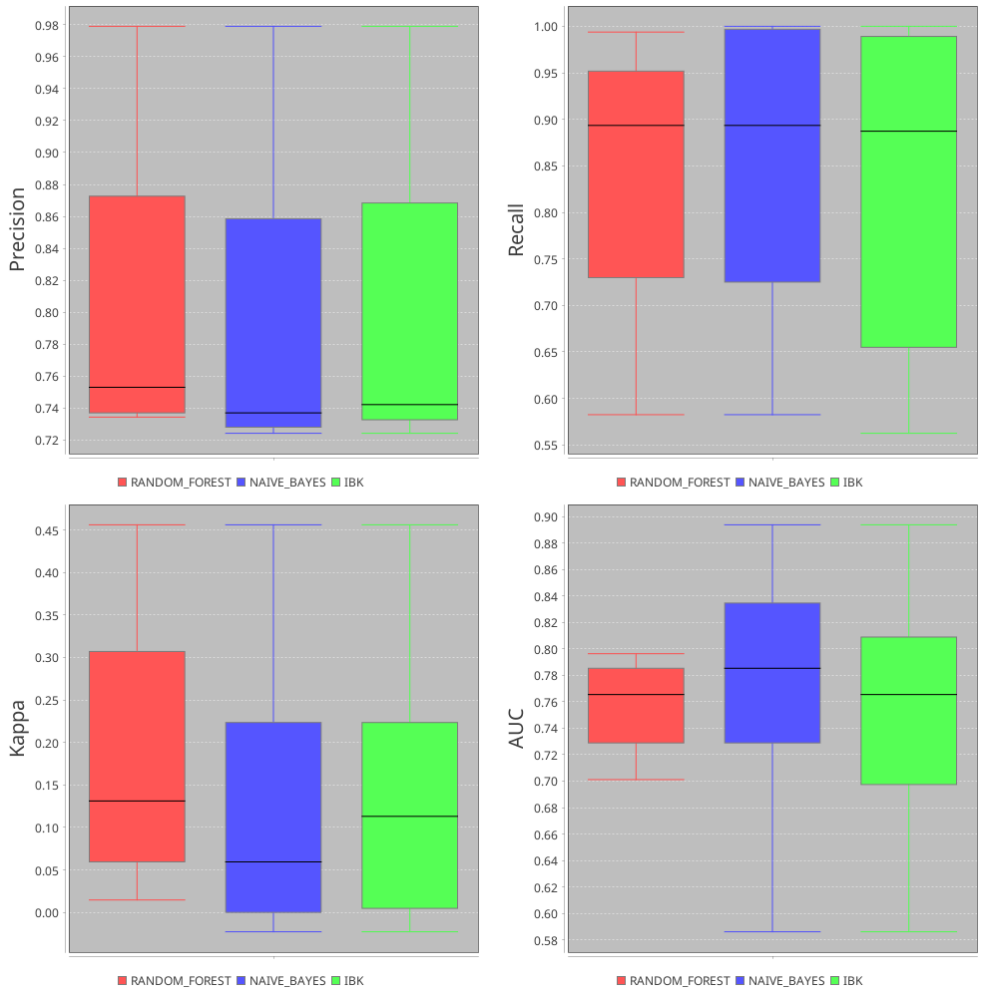
Sono state implementate le tecniche aggiuntive:

- Feature Selection: *best first*
- Balancing: *SMOTE*
- Cost Sensitive Classififer: *Sensitive Threshold*

Utilizzate nel seguente modo:

- No Filter (no feature selection, no balancing, no cost sensitive)
- Feature Selection Best First
- Feature Selection Best First + SMOTE Balancing
- Feature Selection Best First + Sensitive Threshold

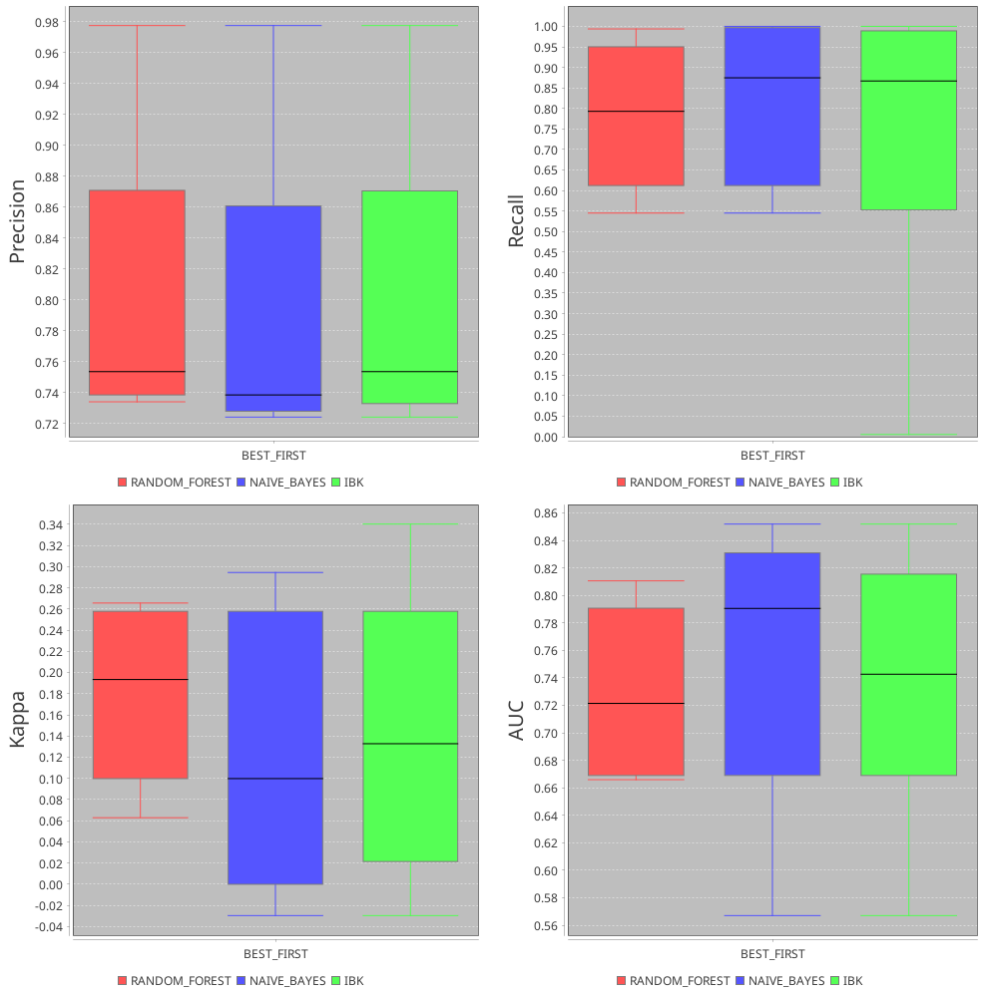
BookKeeper - no filter



Considerazioni

Non risulta esserci un classificatore migliore o peggiore in assoluto

BookKeeper - feature selection

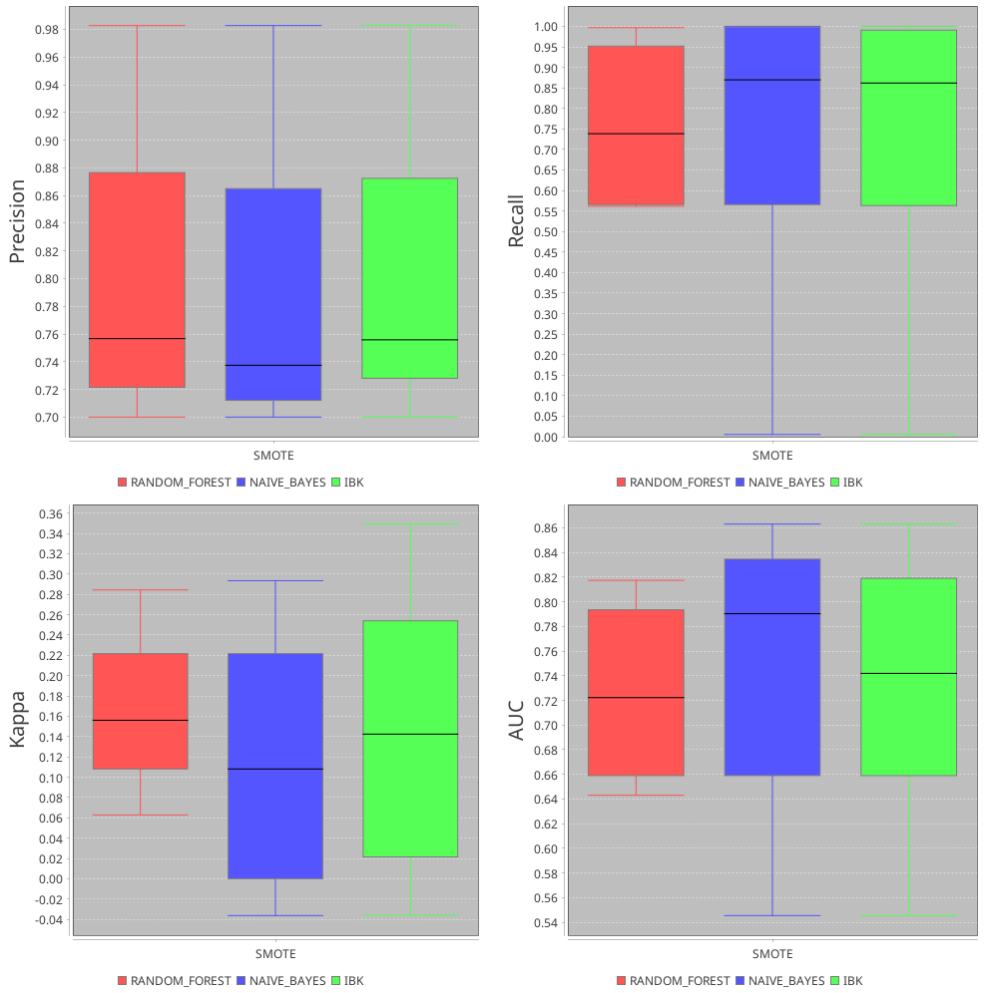


Considerazioni

Anche in questo caso nessun classificatore è migliore in assoluto

Naive Bayes risulta migliore considerando 3 metriche su 4

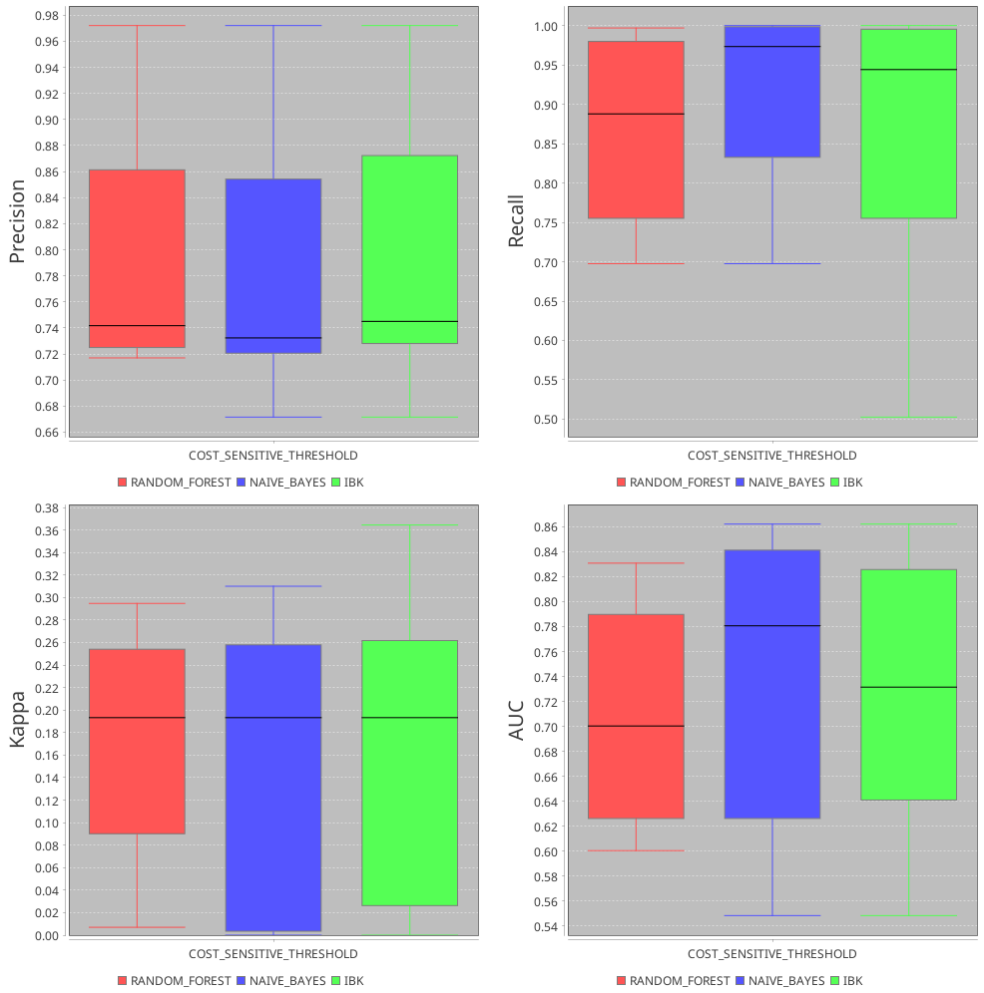
BookKeeper - FS + SMOTE



Considerazioni

In questo caso, ogni classificatore risulta migliore degli altri su una singola metrica

BookKeeper - FS + Cost Sensitive

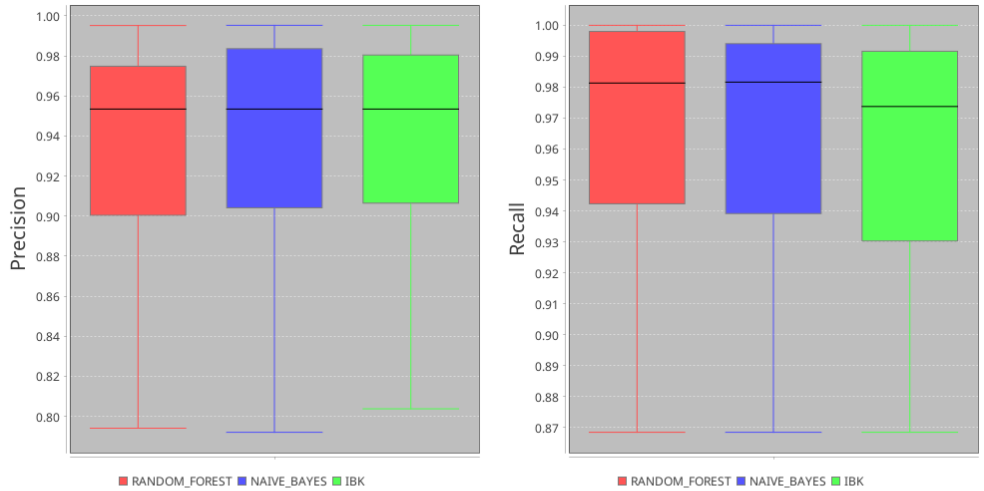


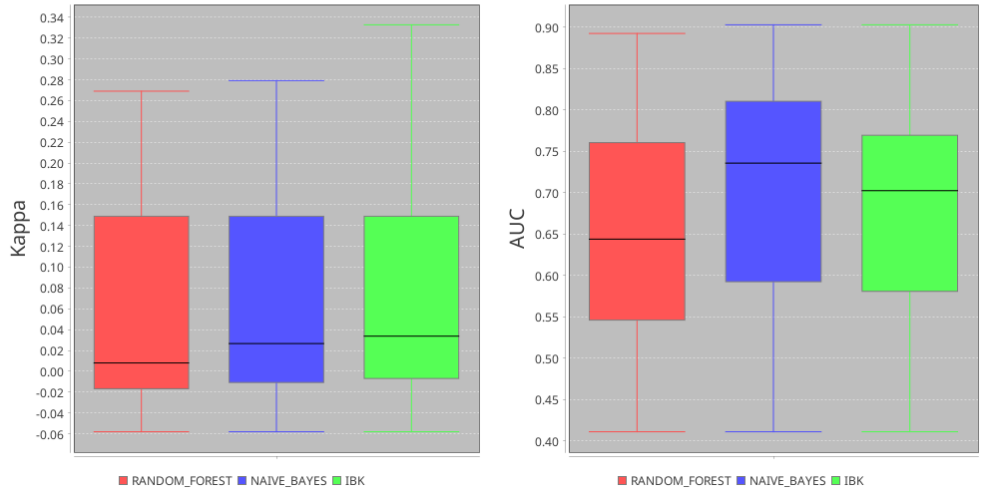
Considerazioni

Anche in questo caso nessun classificatore è migliore in assoluto

Ibk risulta migliore considerando 3 metriche su 4

Avro - no filter

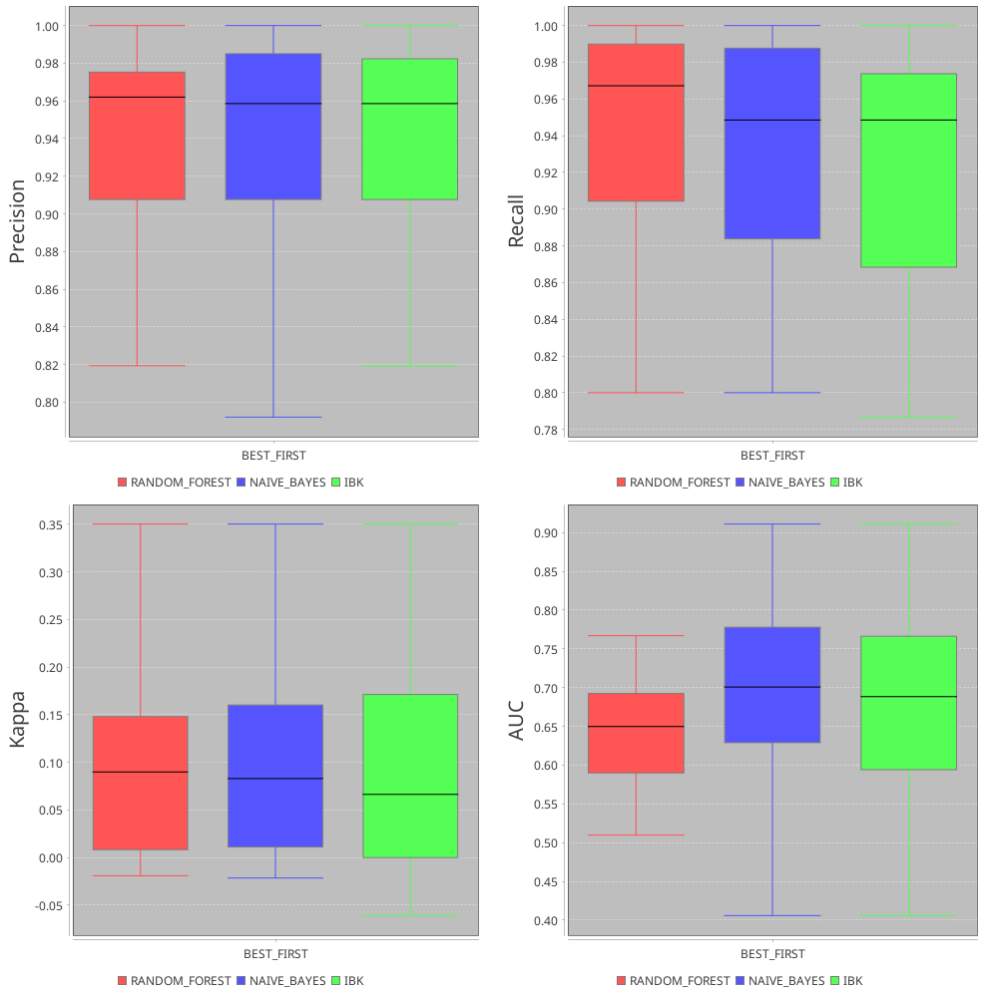




Considerazioni

Random Forest e Naive Bayes risultano essere i classificatori migliori

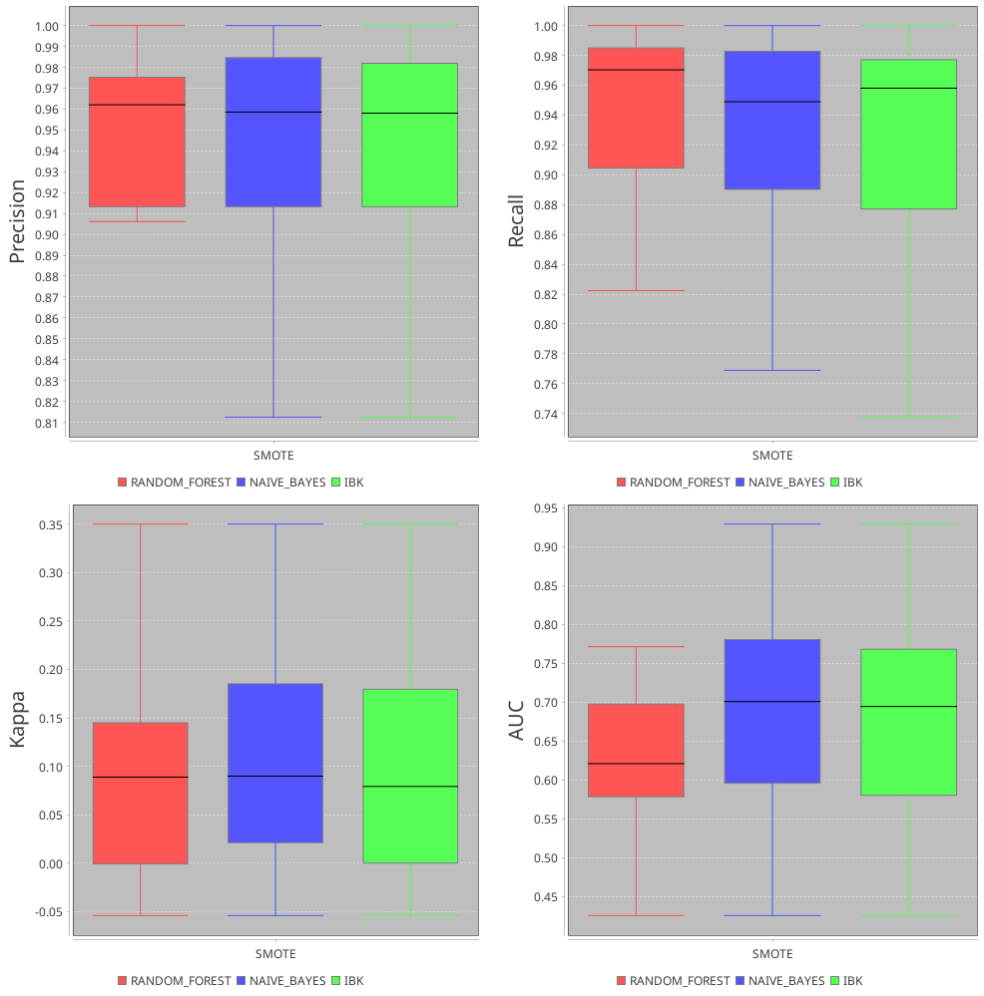
Avro - feature selection



Considerazioni

Nessun classificatore è migliore in assoluto

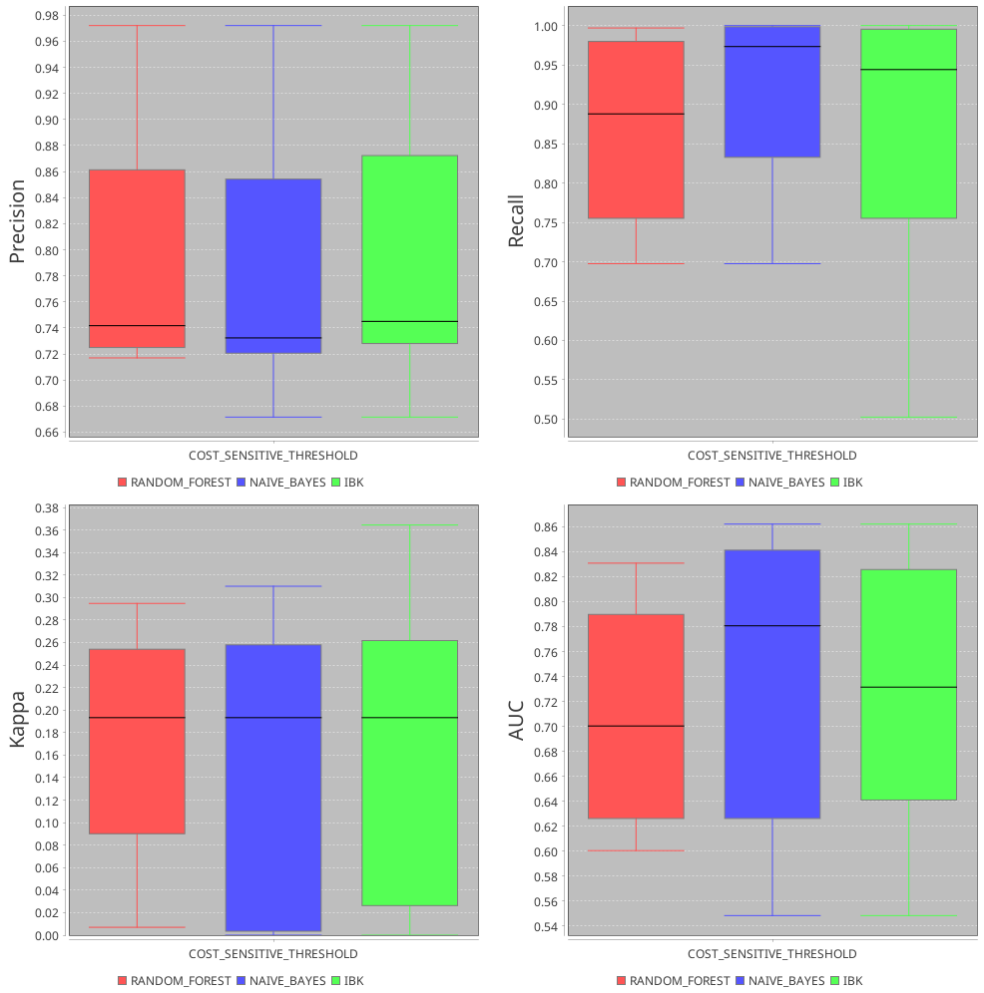
Avro - FS + SMOTE



Considerazioni

Naive Bayes risulta essere il migliore

Avro - FS + Cost Sensitive



Considerazioni

Ibk e Naive Bayes risultano essere i migliori

In generale non esiste un classificatore, o una combinazione tra le tecniche migliore delle altre. Non è quindi possibile stabilire cosa converebbe utilizzare sui singoli progetti o su altri progetti simili

Nel caso di BookKeeper, il solo utilizzo di feature selection best first porta ad una maggiore recall e un mantenimento degli stessi valori sulle altre metriche. L'aggiunta di balancing o cost sensitive threshold, porta ad un peggioramento generale

Nel caso di Avro, utilizzare feature selection best first con cost sensitive threshold, ha portato ad un miglioramento generale delle performance su tutti i classificatori

Problematiche

La scelta dei progetti usati per *proportion* potrebbe falsare i risultati, poichè risulta difficile individuare progetti simili

La metodologia di mapping tra versione di Jira e release di Git in alcuni casi non è sempre stata corretta (in particolare su Avro tra la release su Git e la creazione della nuova versione su Jira, sono state create nuove revisioni)

Repository

Repository contenente codice sorgente e artefatti usati per l'analisi (`csv`, `arff`, grafici)

https://github.com/lioia/isw2_project

SonarCloud

Analisi SonarCloud, con il risultato di 0 code smells

https://sonarcloud.io/project/overview?id=lioia_isw2_project