Machine Learning for Software Engineering

Deliverable del progetto di ISW2

Simone Tiberi (M. 0299908)
(email: simone.tiberi.98@gmail.com)

Università degli studi di Roma Tor Vergata

23 giugno 2022

Agenda

Scopo

Eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di:

- sampling (oversampling, undersampling e SMOTE)
- cost sensitive classification (sensitive threshold e learning)
- feature selection (best first)

sull'accuratezza di modelli predittivi di localizzazione di bug nel codice dei progetti Apache BookKeeper e Storm, utilizzando:

- walk forward come tecnica di valutazione;
- RandomForest, NaiveBayes e IBk come classificatori.

Per la realizzazione dello studio si è fatto uso di:

- Java, come linguaggio di programmazione;
- Github API e JIRA Rest API per la costruzione dei dataset per ciascun progetto;
- WEKA come toolkit di Machine Learning.



Progettazione: Ottenimento delle versioni rilasciate per i progetti

Per ottenere la lista delle versioni rilasciate per ciascun progetto, è stata utilizzata la RestAPI di JIRA. A partire dal JSON restituito a seguito della richiesta effettuata a:

https://issues.apache.org/jira/rest/api/2/project/<PROJ>

sono state estratti nome e data di rilascio per ciascuna versione, al fine di popolare una lista ordinata per la seconda delle due informazioni.

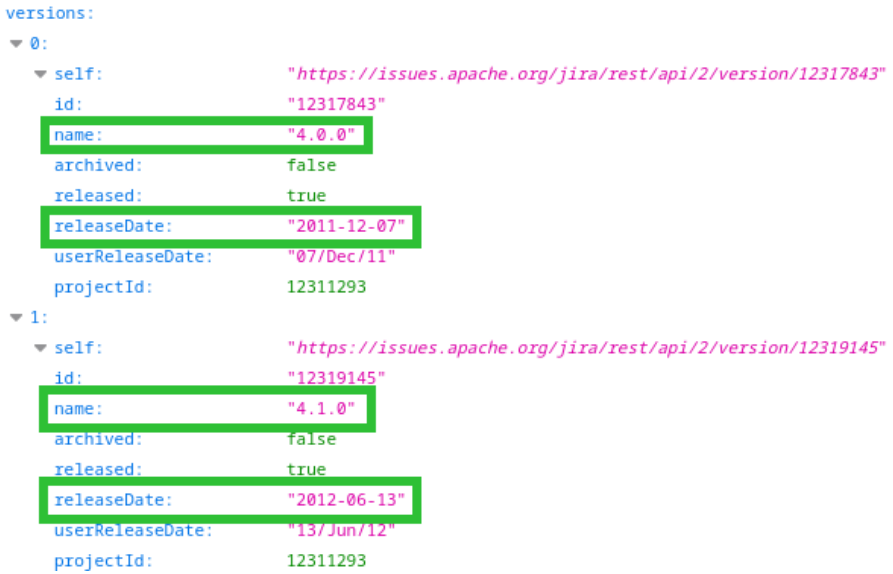


Figura: Estratto del JSON restituito da JIRA
(in verde i campi d'interesse)

Risultato

Sono state individuate:

- 14 versioni per BookKeeper;
- 29 versioni per Storm.

Progettazione: Ottenimento dei bug relativi ai progetti (1)

La lista di bugs relativi ai due progetti è stata ottenuta sfruttando l'API JSON di JIRA. Tramite JQL, è stato specificato di volere unicamente gli issue di tipo bug, la cui risoluzione sia fixed e il cui stato sia resolved o closed.

Per i bug restituiti dall'ITS:

- come FV è stata considerata l'ultima delle fixed versions restituite in ordine temporale;
- come OV è stata considerata la prima successiva alla data di creazione del ticket;
- come IV è stata considerata, la prima delle affected versions restituite in ordine temporale.

Figura: Estratto del JSON restituito da JIRA
(i colori sono stati utilizzati per associare le parti dell'immagine a quanto riportato accanto)

Progettazione: Ottenimento dei bug relativi ai progetti (2)

Una volta ottenuta la lista, è stato necessario sanificarla, rimuovendo tutti i bug tali per cui:

- la FV non corrisponde ad alcuna versione realmente esistente;
- l'ordinamento FV $\geq$ OV non è rispettato.

In questa fase i bug per cui l'IV è successiva all'OV vengono mantenuti non considerando l'injected version fornita da JIRA, al fine di impostarla a seguito tramite proportion.

Risultato

In definitiva, ai fini dello studio, stati presi in considerazione:

- 401 bugs su 435 (92.18%) per BookKeeper;
- 1029 bugs su 1157 (88.93%) per Storm.

Progettazione: Proportion

Non per tutti i bug riportati in JIRA è disponibile l'insieme delle affected versions. Questo è problematico ai fini dello studio, in quanto limita la possibilità di etichettare le classi come buggy in determinate versioni.

Per ovviare a ciò è stata utilizzata la tecnica proportion, in particolare nella sua variante incrementale. In definitiva, per ciascun bug presente nella lista ottenuta al passo precedente, in caso di assenza della IV:

1. è stato calcolato P come media dei rapporti $\frac{FV - IV}{FV - OV}$ per tutti i bug fixed nelle versioni precedenti;
2. è stata calcolata la FV come $FV - (FV - OV) \cdot P$.

Risultato

Sono state calcolate le IV, tramite incremental proportion per:

- 236 bug su 401 (58.85%) per BookKeeper;
- 406 bug su 1029 (39,46%) per Storm.

Progettazione: Ottenimento dei commit effettuati sui progetti (1)

Per l'integrazione con Git è stata utilizzata la Github REST API. L'azienda fornisce la possibilità di effettuare al più 5000 richieste autenticate all'ora (60 senza autenticazione), motivo per cui, per poter rendere l'esecuzione dell'analizzatore più efficiente, è stato elaborato un meccanismo di caching delle request effettuate.

I commit sono infatti oggetti immutabili nel tempo, per cui se una richiesta è stata già effettuata in passato non ha senso dover contattare nuovamente il server remoto, ma è possibile prelevare il JSON direttamente dalla cache locale.
Ai fini dello studio effettuato, per ciascun progetto P, sono state utilizzate due tipi di richieste differenti:

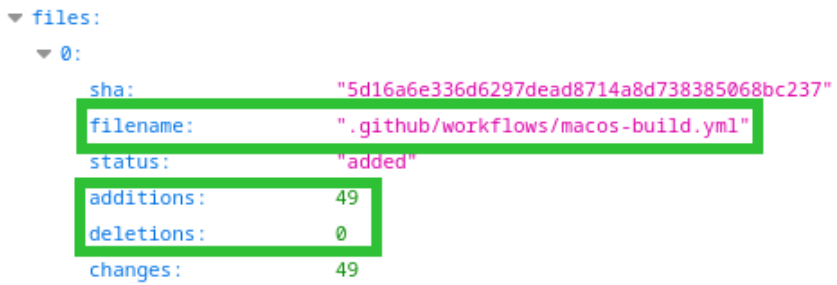
- https://api.github.com/repos/apache/<P>/commits?per_page=100&page=<X>, al variare di X, per costruire la lista di commit;
- https://api.github.com/repos/apache/<P>/commits/<sha>, al variare dello sha, per ottenere informazioni relative a ciascun commit.

Progettazione: Ottenimento dei commit effettuati sui progetti (2)

In particolare, relativamente al secondo tipo di query presentate nella slide precedente, per ogni commit sono state estratti lo SHA, l'autore, la data, il messaggio ed infine il diff.

Quest'ultimo non è altro che un array di oggetti JSON, ciascuno dei quali contenente il nome del i-esimo file toccato dal commit ed il numero di righe aggiunte e/o rimosse.

(a) Dati generali del commit



(b) Dati specifici per ciascun file toccato

Figura: Estratti dei file JSON restituiti dall'API di Github
(in verde i campi d'interesse)

Progettazione: Costruzione del dataset

L'associazione fra le informazioni raccolte da Git e JIRA è realizzata come segue:

- per i bug, tramite la key. In particolare, per ogni bug B e per ogni commit C, se il messaggio di C contiene la chiave di B, i file presenti nel diff vengono aggiunti all'insieme di quelli associati a B;
- per le versioni semplicemente tramite l'ordinamento temporale tra le date.

Risultato

Sono state individuate:

- 854 su 4077 (20.94%) istanze buggy per BookKeeper;
- 3707 su 29579 (12.53%) istanze buggy per Storm.

Una volta definiti, modellati e relazionati fra loro i vari building blocks, iterativamente è stato popolato un dataset, contenente le metriche evidenziate in blu in tabella. È opportuno sottolineare come, per limitare l'effetto dello snoring, ai fini dello studio è stata considerata unicamente la prima metà di release.

Version	Name	Size	NR	NAuth	LOC added	MAX LOC added	AVG LOC added	Churn	MAX Churn	AVG Churn	ChgSetSize	Age	Buggyness

Tabella: Header del dataset

Progettazione: WEKA

- Al termine della fase di costruzione del dataset, questo viene memorizzato in due formati differenti: CSV e ARFF. Quest'ultimo viene sfruttato come base di partenza per la valutazione oggetto dello studio tramite il toolkit WEKA.
- Ai fini dell'analisi, al variare della configurazione d'esecuzione (i.e., tecnica di selezione delle feature, di balancing, applicazione o meno di classificazioni sensibili al costo) è stata valutata l'accuratezza dei classificatori presi in esame.
- Per ogni configurazione sono state eseguite N - 1 run di walk-forward, con N numero di versioni presenti nel dataset, ed è stato popolato un CSV contenente i risultati finali dello studio, sfruttato per produrre i grafici mostrati nelle slide successive.
- Per analizzare i risultati è stato adottato un approccio incrementale ovvero, partendo dall'analisi dell'accuratezza dei classificatori senza applicare tecniche, per poi aggiungerle progressivamente, scegliendo ogni volta la configurazione migliore.
- Per confrontare le prestazione, poiché il tipo di grafico adottato è il box-plot sono state prese in considerazione la mediana, l'ampiezza della distribuzione e la posizione dei quantili principali (i.e., 1Q e 3Q).

Risultati: Solo classificatori (BOOKKEEPER)

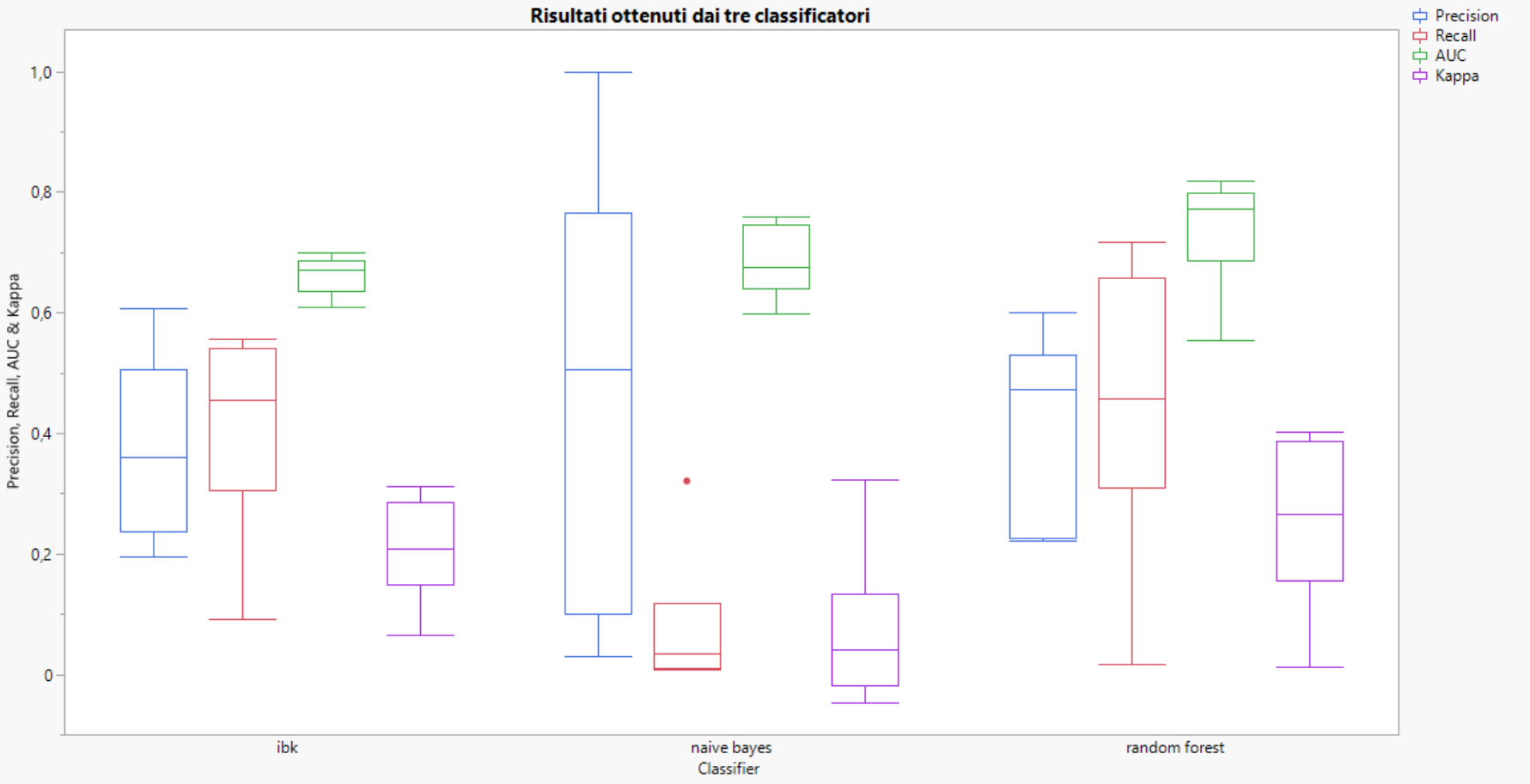


Figura: Risultati dei 3 classificatori senza applicazione di alcuna tecnica di selezione delle feature, di balancing e di classificazione sensibile al costo

Risultati: Solo classificatori (STORM)

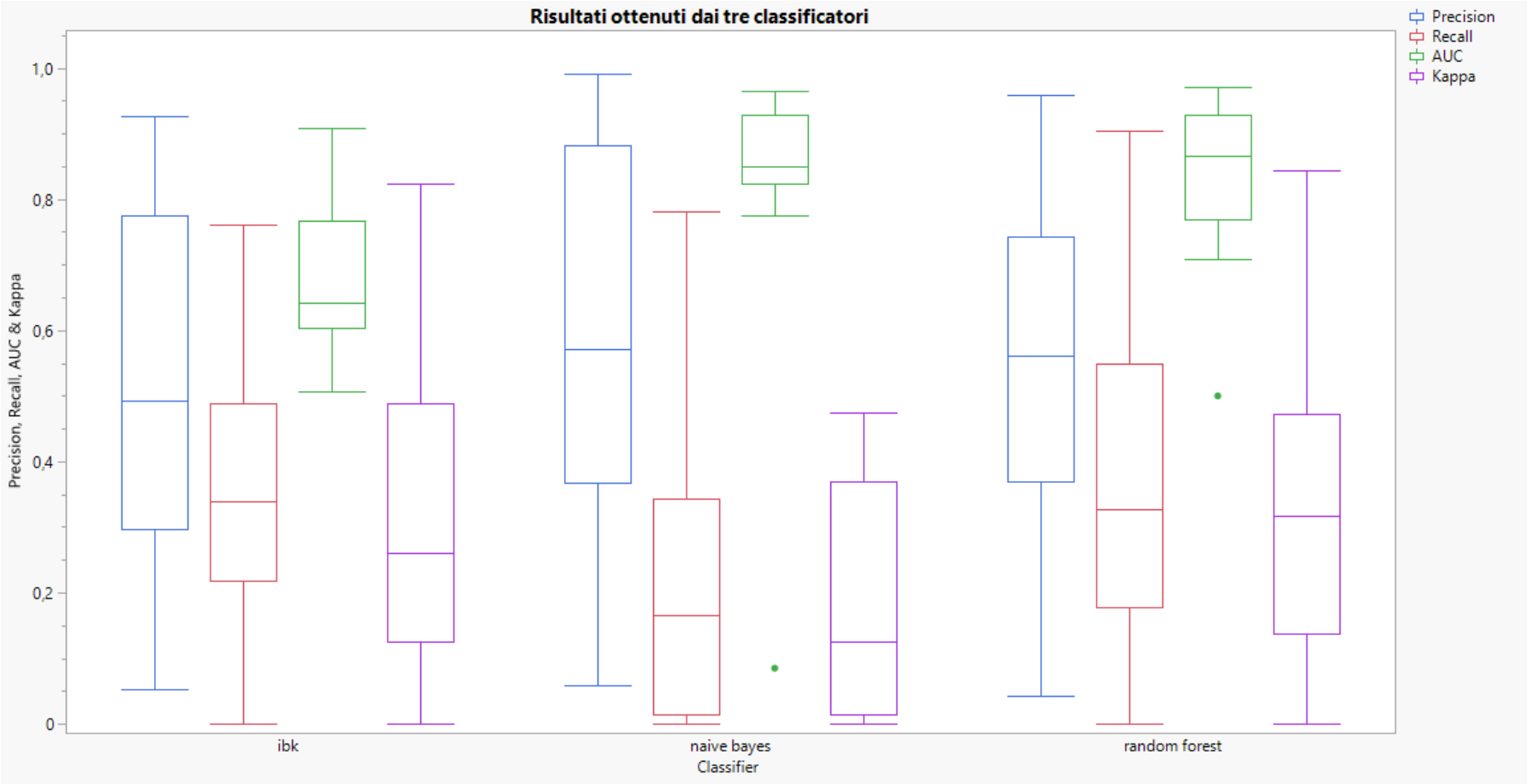


Figura: Risultati dei 3 classificatori senza applicazione di alcuna tecnica di selezione delle feature, di balancing e di classificazione sensibile al costo

Risultati: Solo classificatori (Analisi)

BookKeeper

Il grafico mostra chiaramente come Random Forest sia il miglior classificatore per i dati disponibili. Infatti, per ciascuna metrica presa in considerazione:

- Naive Bayes ha prestazioni decisamente inferiori rispetto agli altri, fatta eccezione della Precision;
- IBk, in termini di mediana e posizionamento di 3Q, ha prestazioni leggermente inferiori rispetto a Random Forest.

Storm

In questo caso, eleggere il classificatore migliore è più complesso. Di fatti, mentre tra Random Forest e IBk la scelta ricade sul primo poiché ha valori migliori per ogni metrica, il confronto con Naive Bayes non è banale da analizzare, infatti:

- Naive Bayes ha una Precision migliore rispetto a Random Forest, sia in termini di mediana che di 3Q;
- Random Forest ha AUC, Kappa e Recall migliori, quest'ultima in particolare in quanto quella di Naive Bayes è la peggiore in assoluto.

Per le fasi successive dello studio è stato deciso di adottare Random Forest, in quanto ha valori migliori per AUC e Kappa ed inoltre la differenza di Precision è decisamente più sottile rispetto a quella della Recall.

Risultati: Applicazione di feature selection (BOOKKEEPER)

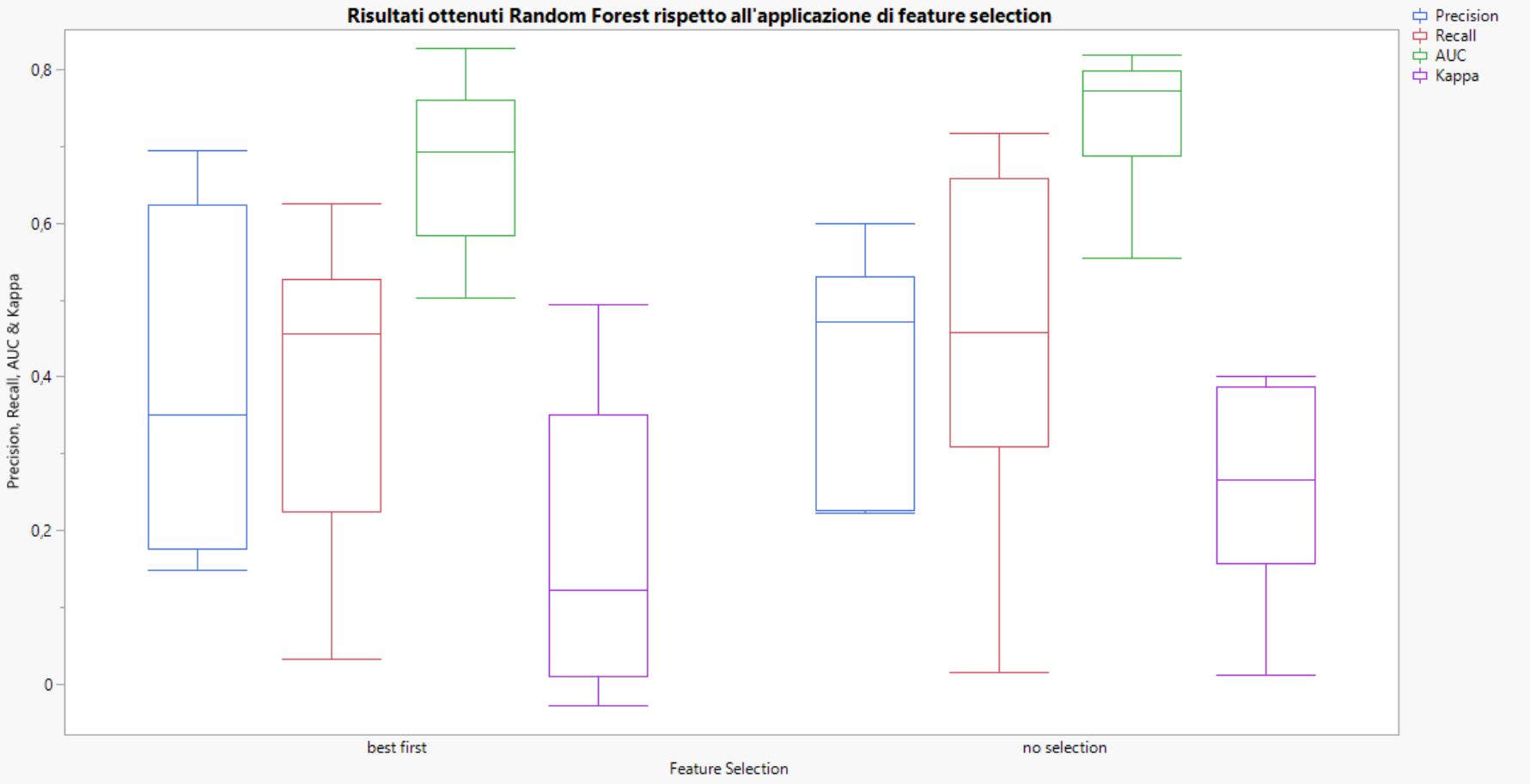


Figura: *Risultati di Random Forest al variare della tecnica di feature selection considerata, senza applicare balancing e classificazione sensibile al costo*

Risultati: Applicazione di feature selection (STORM)

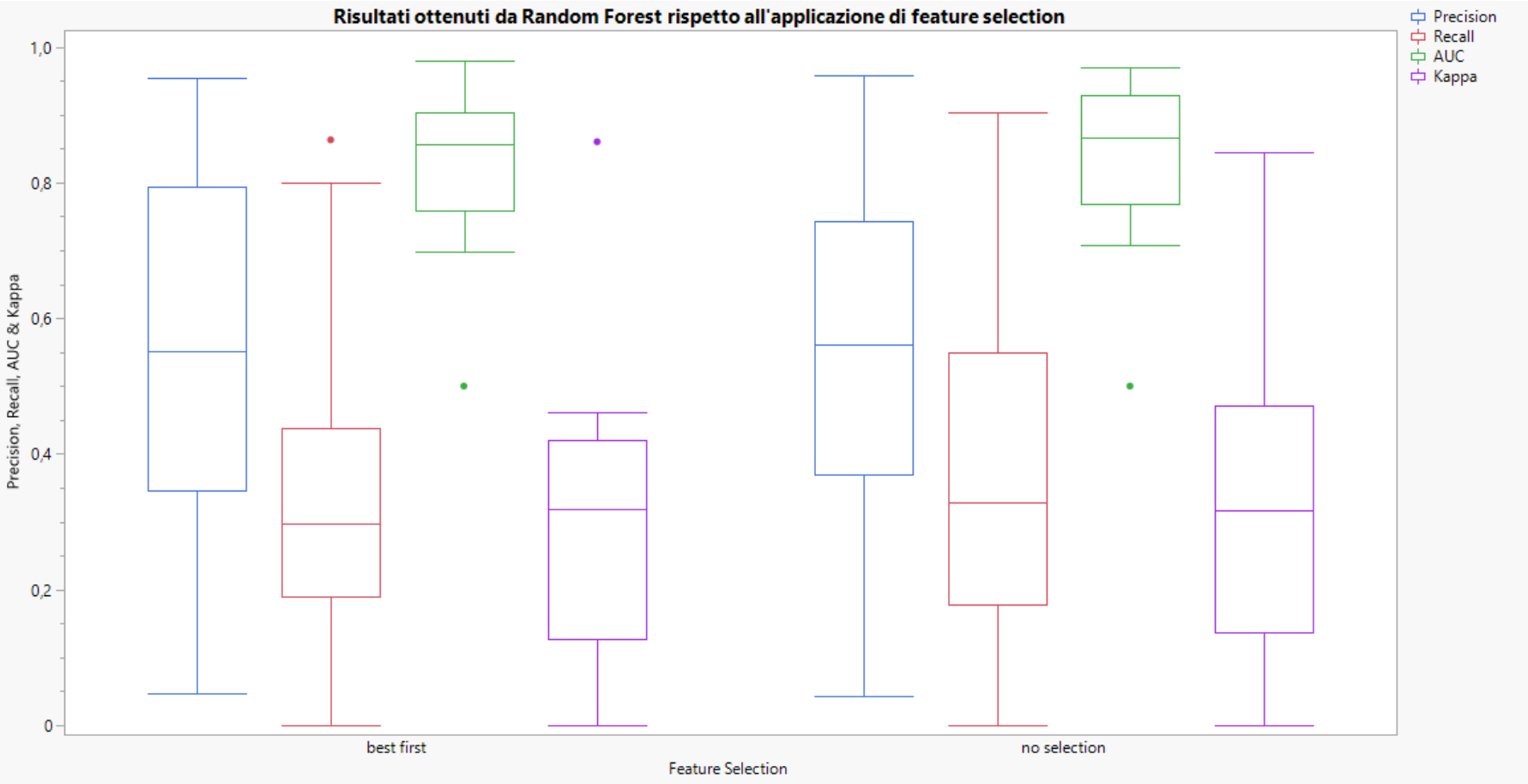


Figura: *Risultati di Random Forest al variare della tecnica di feature selection considerata, senza applicare balancing e classificazione sensibile al costo*

Risultati: Applicazione di feature selection (Analisi)

BookKeeper

Applicando la tecnica di feature selection Best First prima di eseguire la classificazione con Random Forest è possibile notare come:

- in termini di mediana si ha un peggioramento per tutte le metriche, fatta eccezione per la Recall che rimane più o meno costante;
- in termini di distribuzione si sperimenta un aumento del massimo e di 3Q per la Precision.

Nelle sucessive fasi dello studio è stato scelto di non utilizzare Best First, poiché in un campione di *piccole dimensioni* la mediana rappresenta un'informazione più significativa rispetto alla forma della distribuzione.

Storm

In questo caso, si ottengono risultati leggermente diversi rispetto a BookKeeper, infatti:

- in termini di mediana la situazione rimane pressoché inalterata per tutte e 4 le metriche considerate;
- in termini di distribuzione si sperimenta un peggioramento per tutte le metriche fatta eccezione della Precision.

La scelta anche in questo caso è ricaduta nel non utilizzare Best First ed è stata guidata principalmente dai valori osservati delle metriche AUC e Kappa.

Risultati: Applicazione del balancing (BOOKKEEPER)

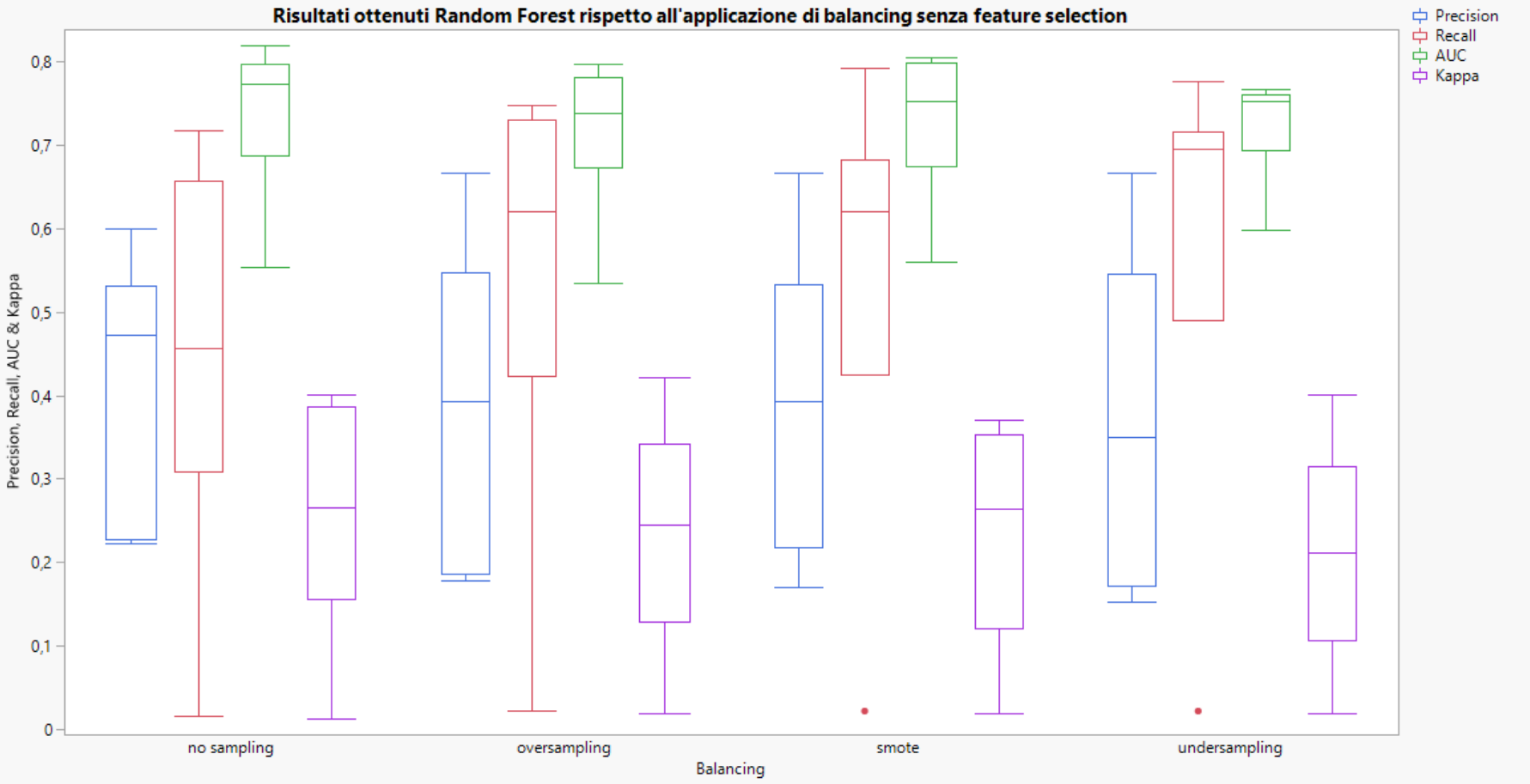


Figura: *Risultati di Random Forest senza feature selection e classificazione sensibile al costo al variare della tecnica di balancing considerata*

Risultati: Applicazione del balancing (STORM)

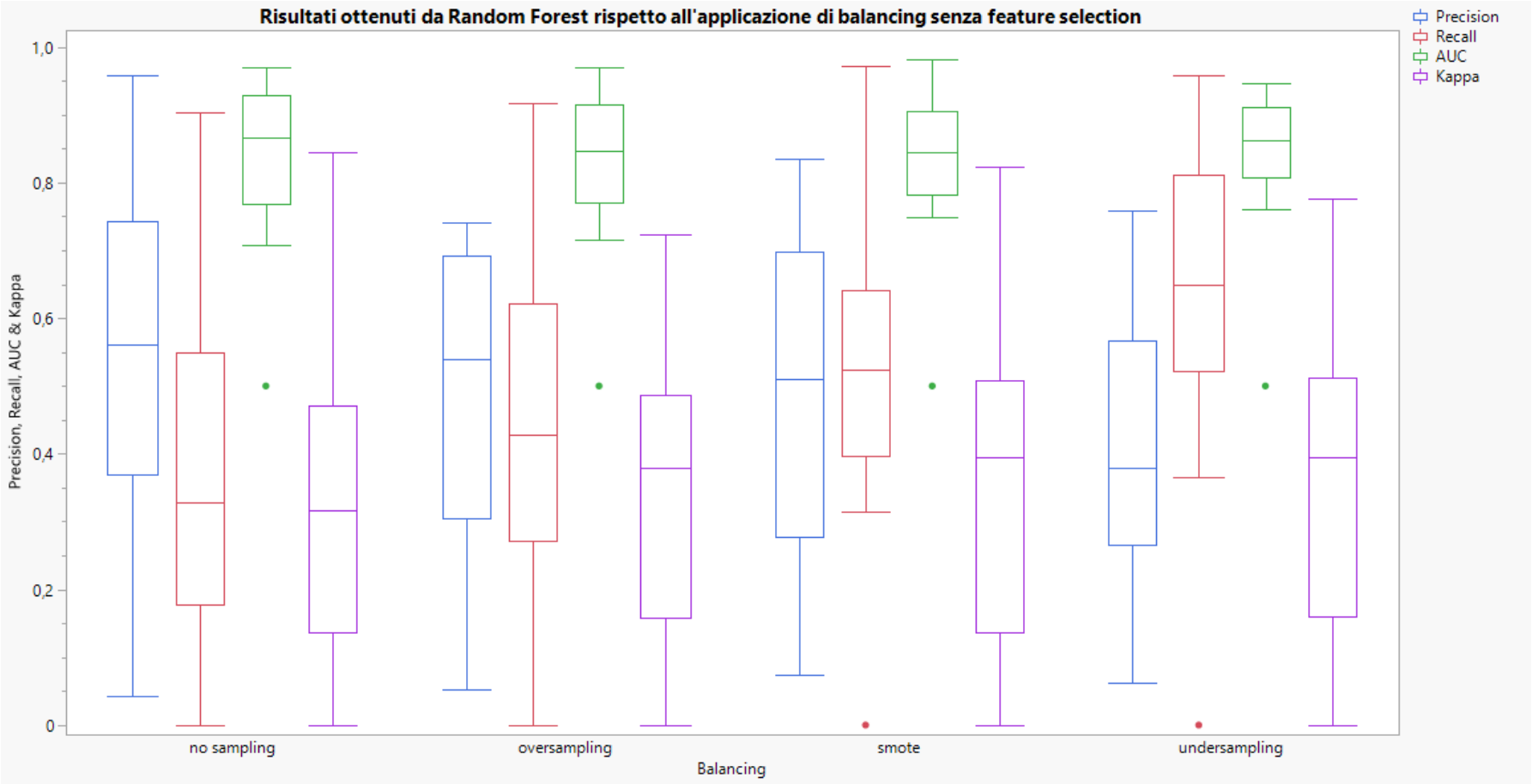


Figura: *Risultati di Random Forest senza feature selection e classificazione sensibile al costo al variare della tecnica di balancing considerata*

Risultati: Applicazione del balancing (Analisi)

Nel caso dell'applicazione del balancing al dataset, l'analisi può essere portata in parallelo sui due progetti. Osservando i box plot riportati nelle due slide precedenti infatti è possibile osservare una leggera diminuzione della Precision ed innalzamento della Recall per tutte le tecniche applicate, comportamento del tutto atteso in quanto aumentando la percentuale di istanze positive nel dataset è naturale tendere a classificare buggy. In particolare in entrambi i progetti l'effetto è maggiore se si applica undersampling per via della diminuzione del numero di istanze presenti nel training set.

Per entrambi i progetti eleggere la configurazione migliore in assoluto è impossibile. Di certo dall'analisi si evince come sia sconsigliato adottare undersampling ed inoltre:

- nel caso in cui si è interessati a massimizzare la Precision, è consigliabile non applicare alcuna tecnica di sampling;
- viceversa nel caso in cui si è interessati alla Recall è consiliabile applicare una tra oversampling e SMOTE.

L'impossibilità nell'eleggere un candidato migliore in assoluto è confermata anche dalle distribuzioni quasi identiche dei valori assunti dalla AUC per tutte e 4 le configurazioni.

Risultati: Applicazione di cost sensitive classification (BOOKKEEPER)

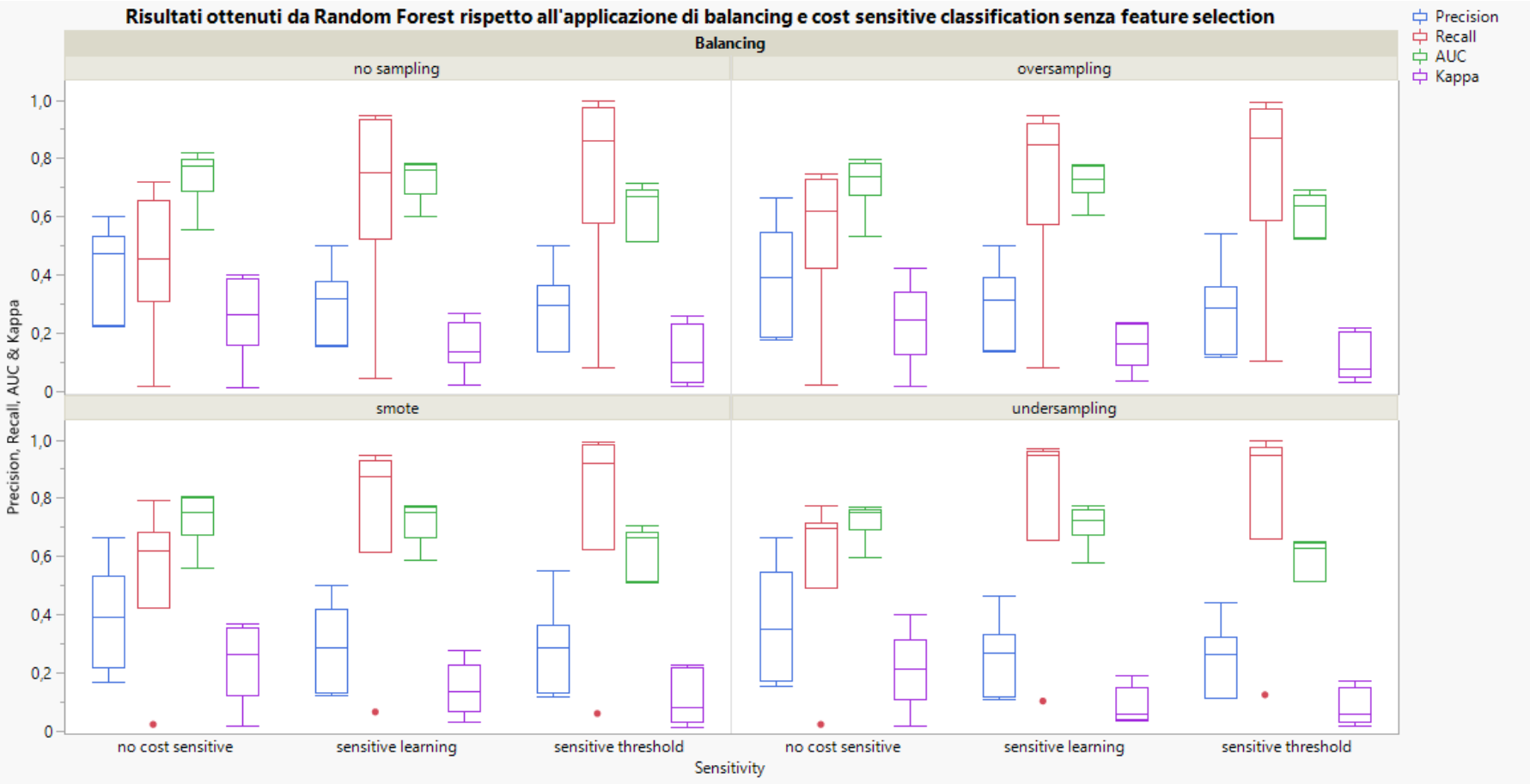


Figura: *Risultati di Random Forest senza feature selection al variare delle tecniche di balancing e classificazione sensibile al costo considerate*

Risultati: Applicazione di cost sensitive classification (STORM)

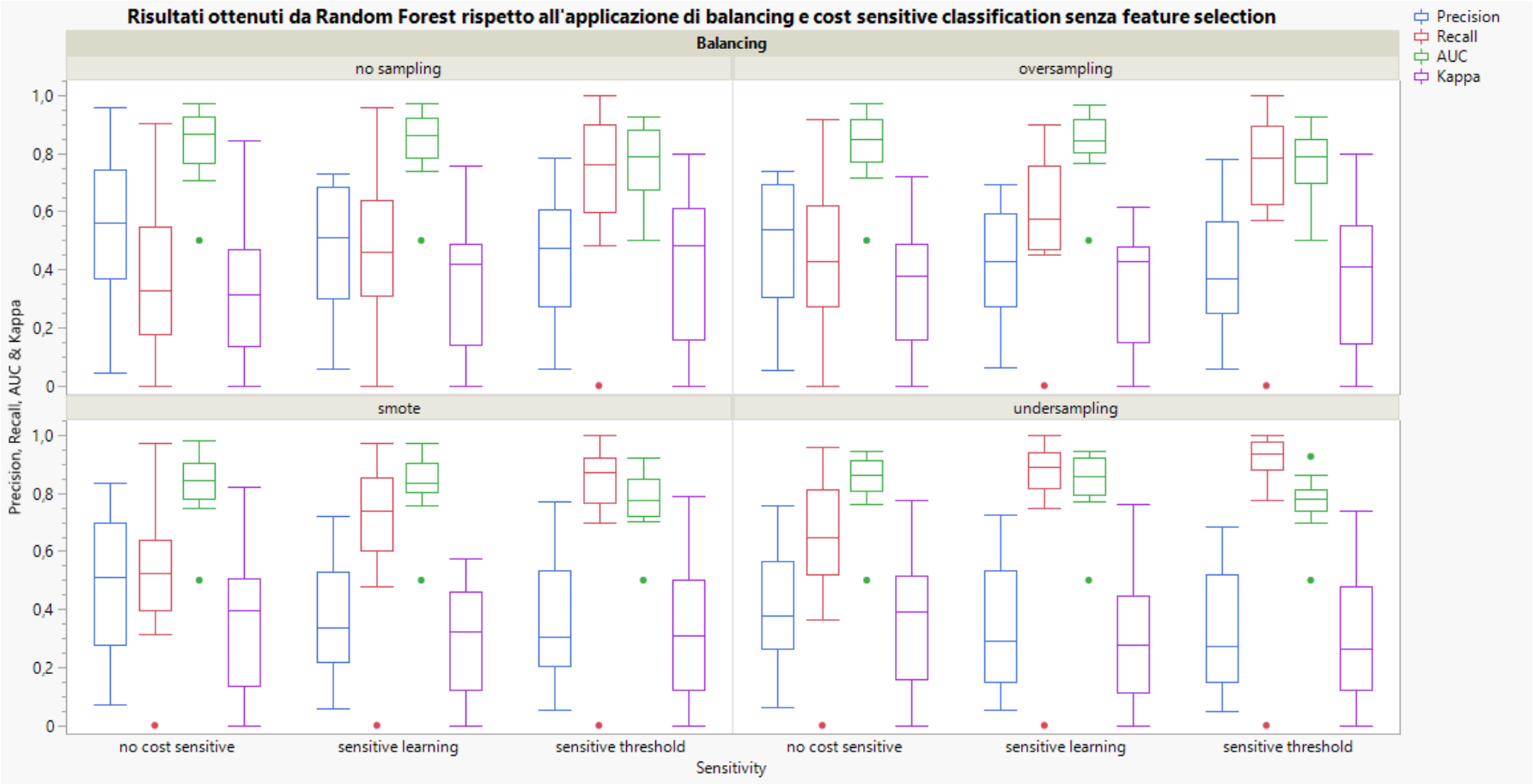


Figura: *Risultati di Random Forest senza feature selection al variare delle tecniche di balancing e classificazione sensibile al costo considerate*

Risultati: Applicazione di cost sensitive classification (Analisi)

Non avendo eletto una configurazione migliore nel caso dell'applicazione di tecniche di sampling è stato preferito prendere in considerazione per i classificatori sensibili al costo tutte le possibili combinazioni.
Per entrambi i progetti è stata adottata come matrice di costo quella riportata accanto.

CTP = 0	CFN = 10
CFP = 1	CTN = 0

Tabella: Matrice di costo (CFN = 10 · CFP)

Sia per BookKeeper che per Storm è evidente come:

- penalizzando maggiormente un FN rispetto ad un FP, aumenti la Recall e diminuisca la Precision, come è ragionevole che sia;
- adottando sensitive threshold, la variazione delle due metriche sia massima rispetto a sensitive learning;
- AUC rimane pressoché invariata applicando sensitive learning, mentre descresce leggermente nel caso della variante con threshold;
- Kappa descresce in maniera similare sia applicando sensitive learning che threshold.

Conclusioni

- Dall'analisi effettuata, come è normale che sia, è evidente non sia possibile eleggere una configurazione migliore in assoluto (no silver bullet).
- È opportuno osservare come i risultati ottenuti siano coerenti con quanto atteso dalla teoria (e.g. applicando cost sensitive classification ci si aspetta che la recall aumenti).
- Il fatto che AUC sia sempre, fatta eccezione per gli outliers, maggiore di 1/2 e Kappa sempre sopra lo 0 è sintomo che l'insieme delle feature scelte sia correlato alla buggyness e quindi i classificatori tendono a ad avere prestazioni migliori di uno *dummy*.
- Il fatto che le considerazioni fatte per BookKeeper siano compatibili con quelli di Storm indica che i risultati dello studio possano essere generalizzati ad altri progetti open source analoghi con una discreta confidenza.
- Per via del limite di slide disponibili, dell'intero albero delle configurazioni possibili è stato considerato un solo cammino, ottenuto effettuando ad ogni passo una scelta greedy.

Links

Github

https://github.com/tibwere/ISW2-deliverable-2

SonarCloud

https://sonarcloud.io/project/overview?id=tibwere_ISW2-deliverable-2