Machine Learning per Software Engineering

Alessandro Chillotti

Università degli studi di Roma Tor Vergata

19 Giugno 2022

Indice

Introduzione (1/4)

Contesto del progetto e terminologia (1/2)

Nell'ingegneria del software esiste un tema importante che riguarda le previsioni di bug, ovvero il processo che ha l'obiettivo di identificare se artefatti software sono difettosi.
Si necessita di alcune nozioni preliminari:

- *Revisione*: ogni commit produce una revisione
- *Versione/Release*: è una revisione che viene resa pubblica ed utilizzabile
- *Ticket*: è un evento che deve essere esaminato o un elemento di lavoro che deve essere affrontato. Contiene delle informazioni importanti:
  - *Tipo*: rappresenta il tipo di lavoro che deve essere affrontato (e.g. Bug)

Introduzione (2/4)

Contesto del progetto e terminologia (2/2)

- *Injected Version* (IV): è la versione nella quale il bug è stato inserito
- *Opening Version* (OV): è la versione nella quale si è aperto il ticket relativo al bug
- *Affected Versions* (AV): sono le versioni che sono state affette dal bug
- *Fixed Version* (FV): è la versione nella quale il bug è stato risolto

Introduzione (3/4)

Problema

Come enunciato in precedenza, nell'ingegneria del software la previsione dei bug gioca un ruolo fondamentale. In particolare, nel caso di un progetto software Java:

- Una classe difettosa è una classe che presenta un bug al suo interno
- Per correggere un bug si ha bisogno di una fase di test che può essere costosa

Prevedere che una classe possa essere difettosa può permettere al tester di concentrarsi su mirate classi, senza dover spendere altri costi su classi che non sono difettose

Introduzione (4/4)

Obiettivi dello studio eseguito

Il progetto svolto è uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling, classificazioni sensibili al costo e feature selection sull'accuratezza di modelli predittivi di machine learning.
I progetti open-source utilizzati sono sviluppati dalla Apache Software Foundation:

- *Apache Bookkeeper*
- *Apache Zookeeper*

Lo scopo della presentazione è quello di presentare i risultati:

- In una prima fase verranno mostrati i risultati ottenuti al variare delle combinazioni di tecniche applicate
- In una seconda fase verrà mostrato il comportamento dei classificatori in scenari specifici

Progettazione (1/3)

Costruzione dei dataset

Per la costruzione dei dataset sono stati eseguiti i seguenti passi:

1. Combinazione dell'utilizzo delle REST api di *Jira* e il tool *JGit* per la costruzione della lista di versioni necessarie
2. Retrieve dei ticket di tipo bug e per ognuno di essi si tiene traccia dei file toccati. In particolare, nel caso in cui il ticket non avesse le AV, si è utilizzato il metodo proportion per stimare l'IV da utilizzare per definire le AV
3. Computazione della buggyness di ogni classe per ogni versione andando ad analizzare i dati ottenuti al passo 2

Caratteristiche dei dataset ottenuti

I dataset ottenuti hanno come classe minoritaria la classe buggy:

- In Bookkeeper circa il 23% di classi è buggy
- In Zookkeeper circa l'11% di classi è buggy

Progettazione (2/3)

Metriche selezionate

Le metriche selezionate per la realizzazione del dataset sono:

- *Size*: numero di LOC
- *NAuth*: numero di autori di una classe
- *Max LOC added*: massimo numero di LOC aggiunte in una release
- *Churn*: differenza fra numero di LOC aggiunte e rimosse
- *AVG churn*: media della differenza fra numero di LOC aggiunte e rimosse
- *NR*: numero di revisioni
- *LOC added*: numero di LOC aggiunte in una release
- *AVG LOC added*: numero medio di LOC aggiunte in una release
- *Max churn*: massima differenza fra numero di LOC aggiunte e rimosse
- *Age*: età in settimane di una classe

Progettazione (3/3)

Tecnica di classificazione

La tecnica di classificazione utilizzata è *walk forward*, quindi l'idea è che ad ogni run i, considerando le versioni ordinate cronologicamente, si ha che:

- il *training set* contiene le prime i - 1 versioni
- il *testing set* contiene esclusivamente la versione i-esima

I classificatori utilizzati sono *Random Forest*, *Naive Bayes* e *Ibk*. Per la loro esecuzione è stato utilizzato il tool *Weka*, in particolare le API messe a disposizione da esso.

Metriche analizzate

Le metriche di performance di cui sono stati analizzati i valori sono la *recall*, la *precision*, la *kappa* e l'*AUC*(*Area Under The Curve*).

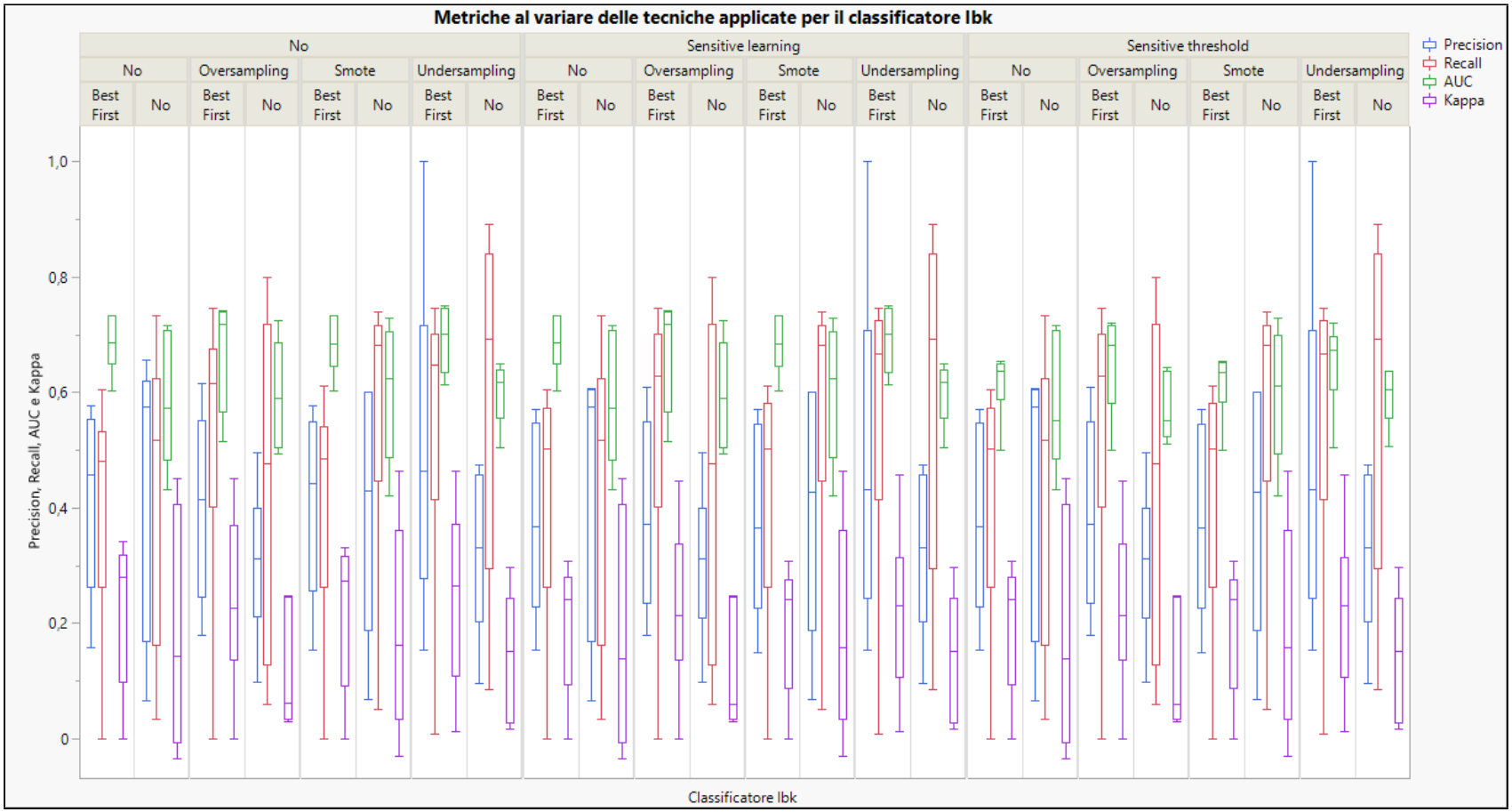
Variabili considerate

Le variabili validate empiricamente sono le seguenti:

- come feature selection, si considera *best first*
- come balancing, si considerano :
  - *oversampling*: si aumentano le istanze della classe minoritaria fino a pareggiare il numero di istanze della classe maggioritaria
  - *undersampling*: si diminuiscono le istanze della classe maggioritaria fino a pareggiare il numero di istanze della classe minoritaria
  - *SMOTE*: si creano "sinteticamente" nuove istanze per la classe minoriataria
- come cost sensitive, si considerano *sensitive threshold* e *sensitive learning* e la matrice dei costi prevede un costo dieci volte maggiore per un falso negativo rispetto al costo per un falso positivo

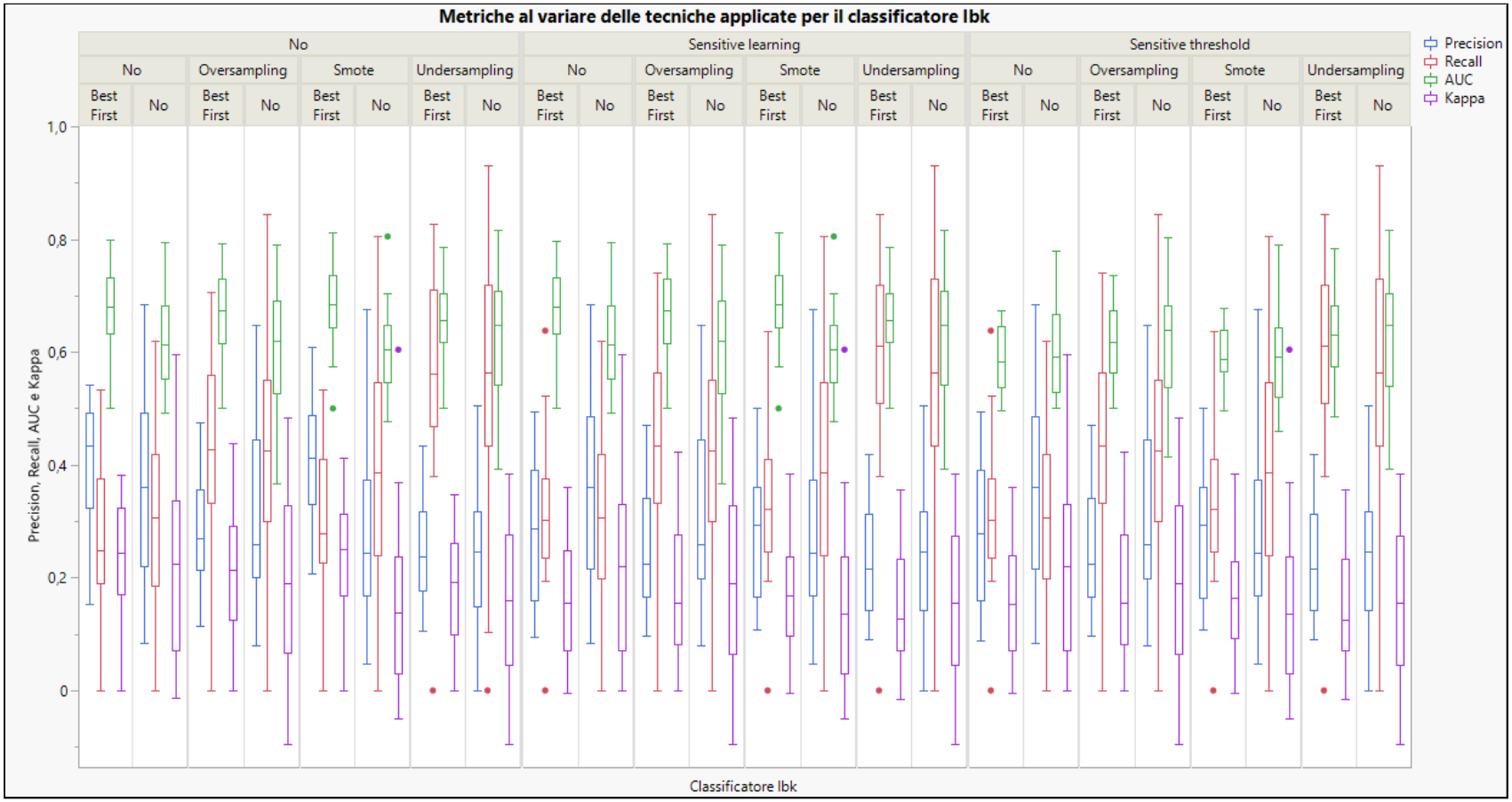
Ibk - Risultati ottenuti (1/3)

Bookkeeper - Comportamento al variare delle tecniche



Ibk - Risultati ottenuti (2/3)

Zookeeper - Comportamento al variare delle tecniche



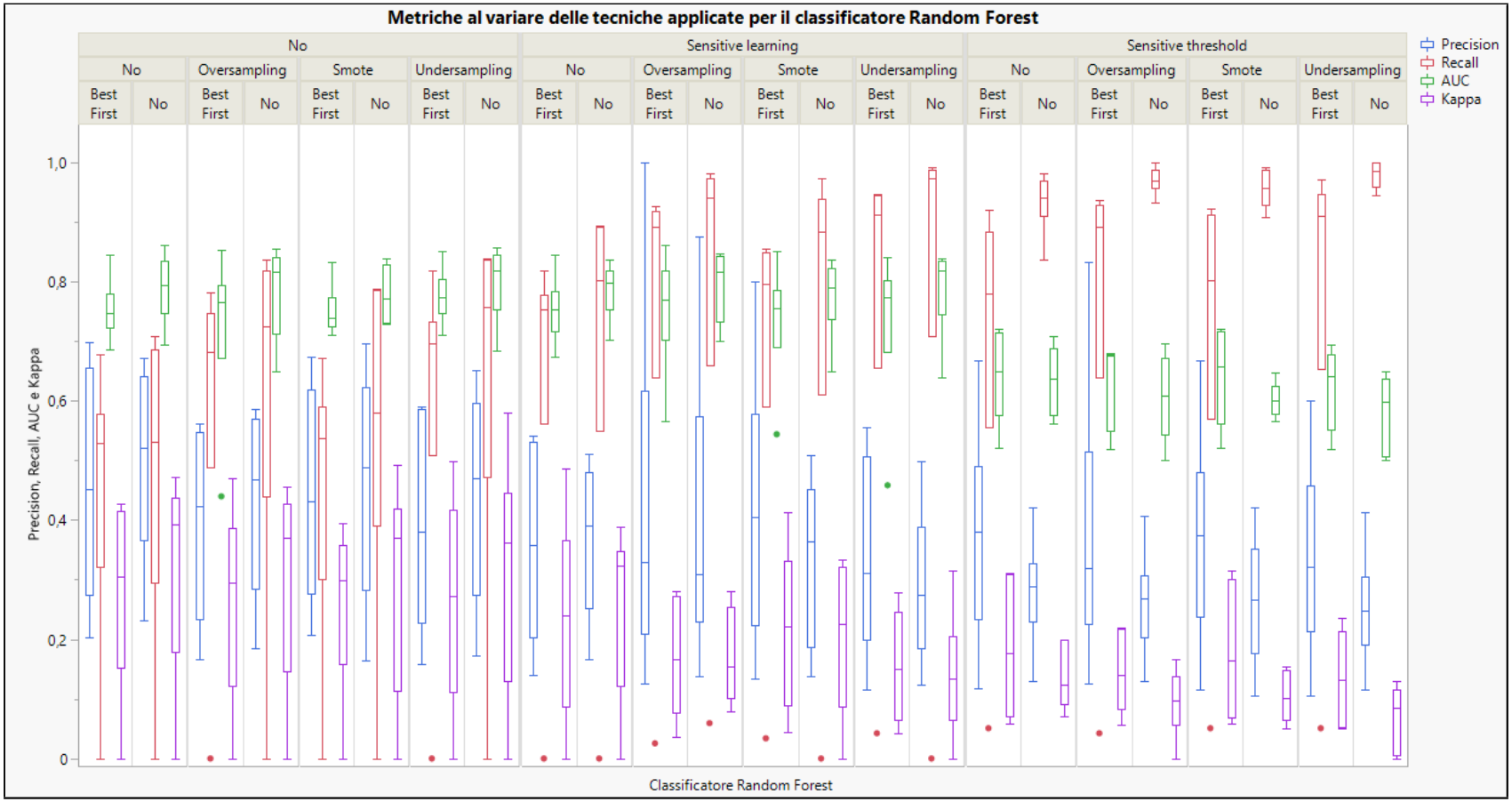
Ibk - Risultati ottenuti (3/3)

Considerazioni riguardo il classificatore Ibk

- La *recall* migliora con l'applicazione di tecniche di balancing
- La *precision* peggiora quando vengono applicate tecniche di balancing e risente anche dell'applicazione di tecniche di cost sensitive
- I risultati descritti nei primi due bullet sono attesi, in quanto un buon balancing aumenta significativamente la *recall* inficiando poco sulla *precision*
- L'*AUC* migliora con l'applicazione di feature selection, ma nel caso in cui si applica *sensitive threshold*, in Zookkeeper, questo comportamento tende a cambiare
- *Kappa* peggiora con l'applicazione di feature selection, ad eccezioni di alcuni casi in Zookkeeper ed inoltre peggiora con l'applicazione di tecniche cost sensitive

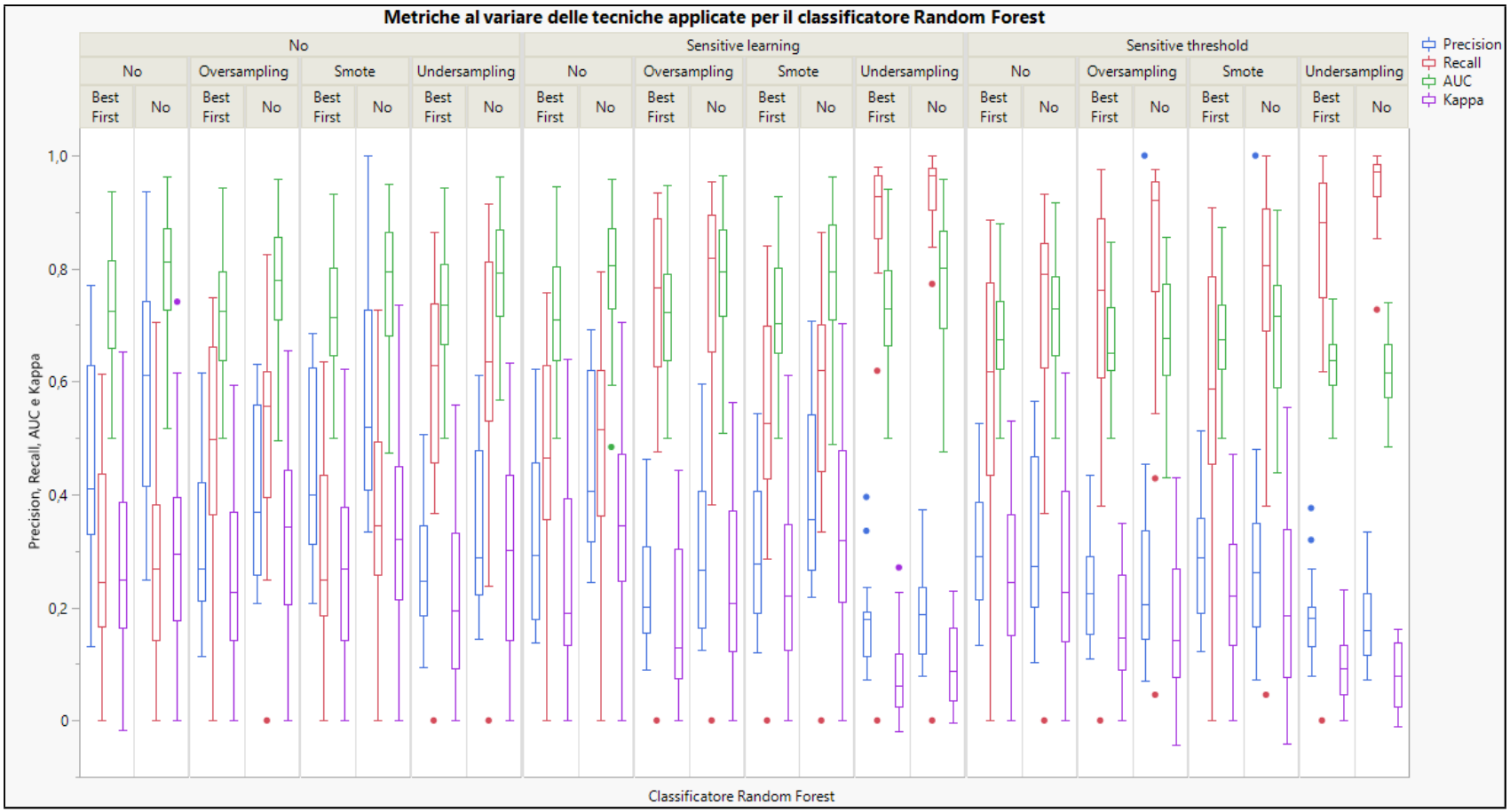
Random Forest - Risultati ottenuti (1/3)

Bookkeeper - Comportamento al variare delle tecniche



Random Forest - Risultati ottenuti (2/3)

Zookeeper - Comportamento al variare delle tecniche



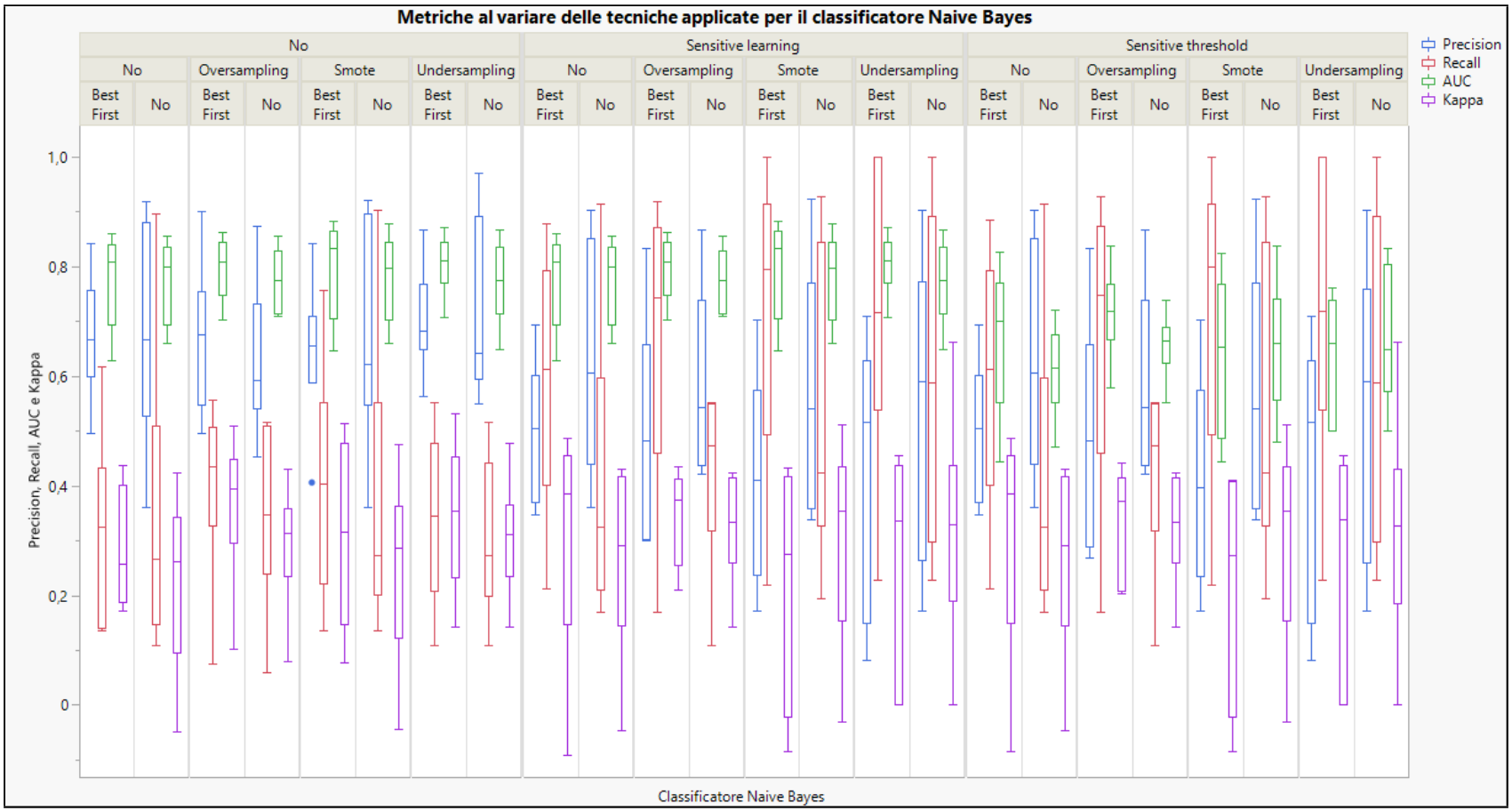
Random Forest - Risultati ottenuti (3/3)

Considerazioni riguardo il classificatore Random Forest

- La *recall* migliora nel caso in cui si applicano tecniche di balancing e tecniche di cost sensitive, in particolare si raggiungono valori alti nel caso di applicazione di *undersampling*
- D'altra parte, questo inficia sulla *precision* ed infatti il valore di precision più alto si ha quando non vengono applicate tecniche di balancing
- L'*AUC* non viene alterata significativamente nel caso in cui viene applicato *sensitive learning* rispetto al caso in cui non si applicano tecniche di cost sensitive, ma l'applicazione di *sensitive threshold* porta ad un suo peggioramento
- *Kappa* non migliora con l'applicazione di feature selection, anche se nel caso in cui si applica *sensitive threshold* questo comportamento tende ad invertirsi

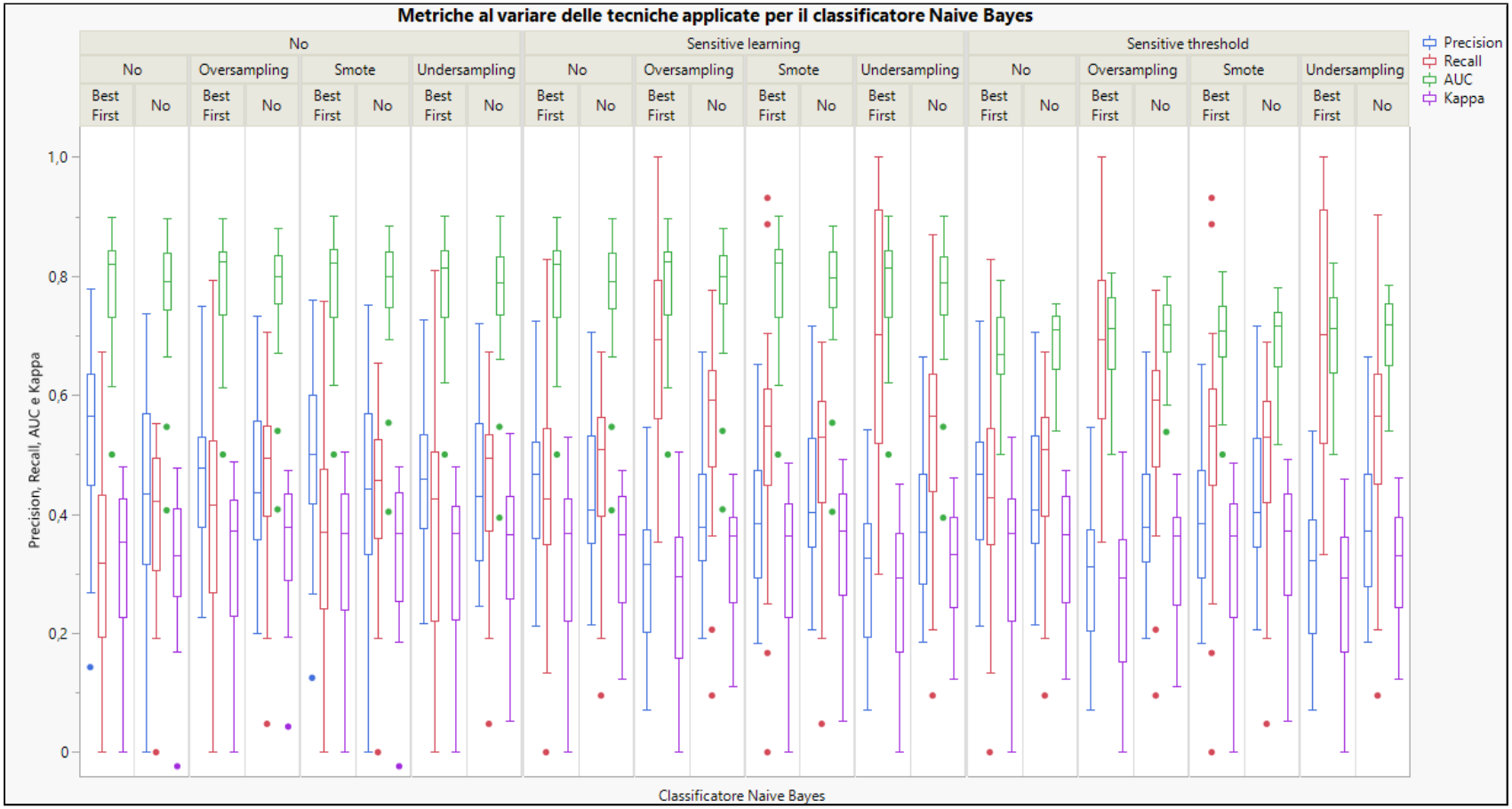
Naive Bayes - Risultati ottenuti (1/3)

Bookkeeper - Comportamento al variare delle tecniche



Naive Bayes - Risultati ottenuti (2/3)

Zookeeper - Comportamento al variare delle tecniche



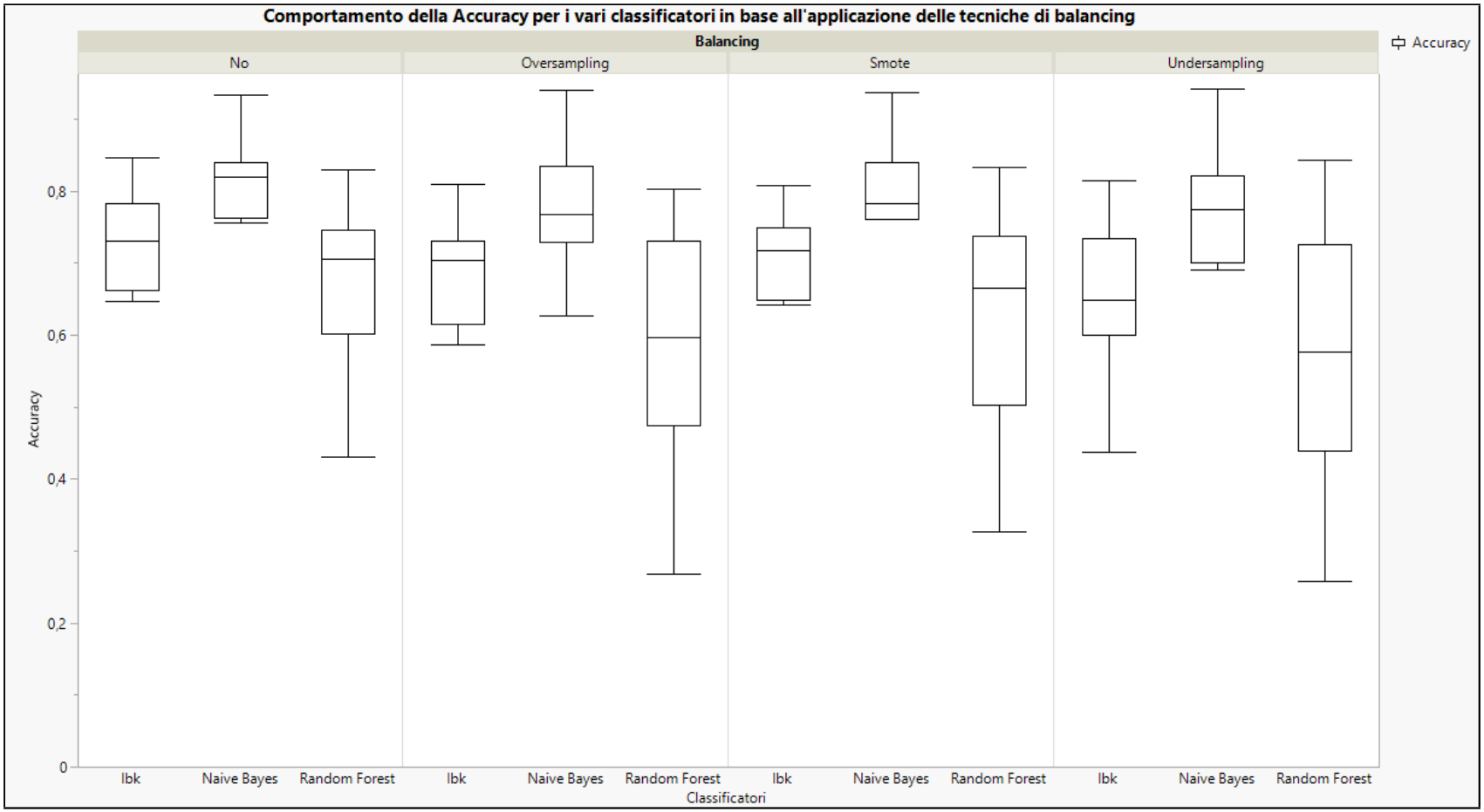
Naive Bayes - Risultati ottenuti (3/3)

Considerazioni riguardo il classificatore Naive Bayes

- La *recall* migliora con l'applicazione di tecniche di cost sensitive, feature selection e balancing
- La *precision* peggiora con l'applicazione di tecniche cost sensitive, ma in assenza di esse migliora attraverso l'applicazione di feature selection
- *AUC* mantiene valori abbastanza costanti con l'applicazione di *sensitive learning*, ma peggiora visibilmente con l'applicazione di *sensitive threshold*
- *Kappa* migliora con l'applicazione delle tecniche di balancing, tranne se in combinazione con *sensitive threshold* nel progetto Zookkeeper

Bookkeeper - Caso specifico (1/4)

Accuracy dei classificatori in base alle tecniche di sampling



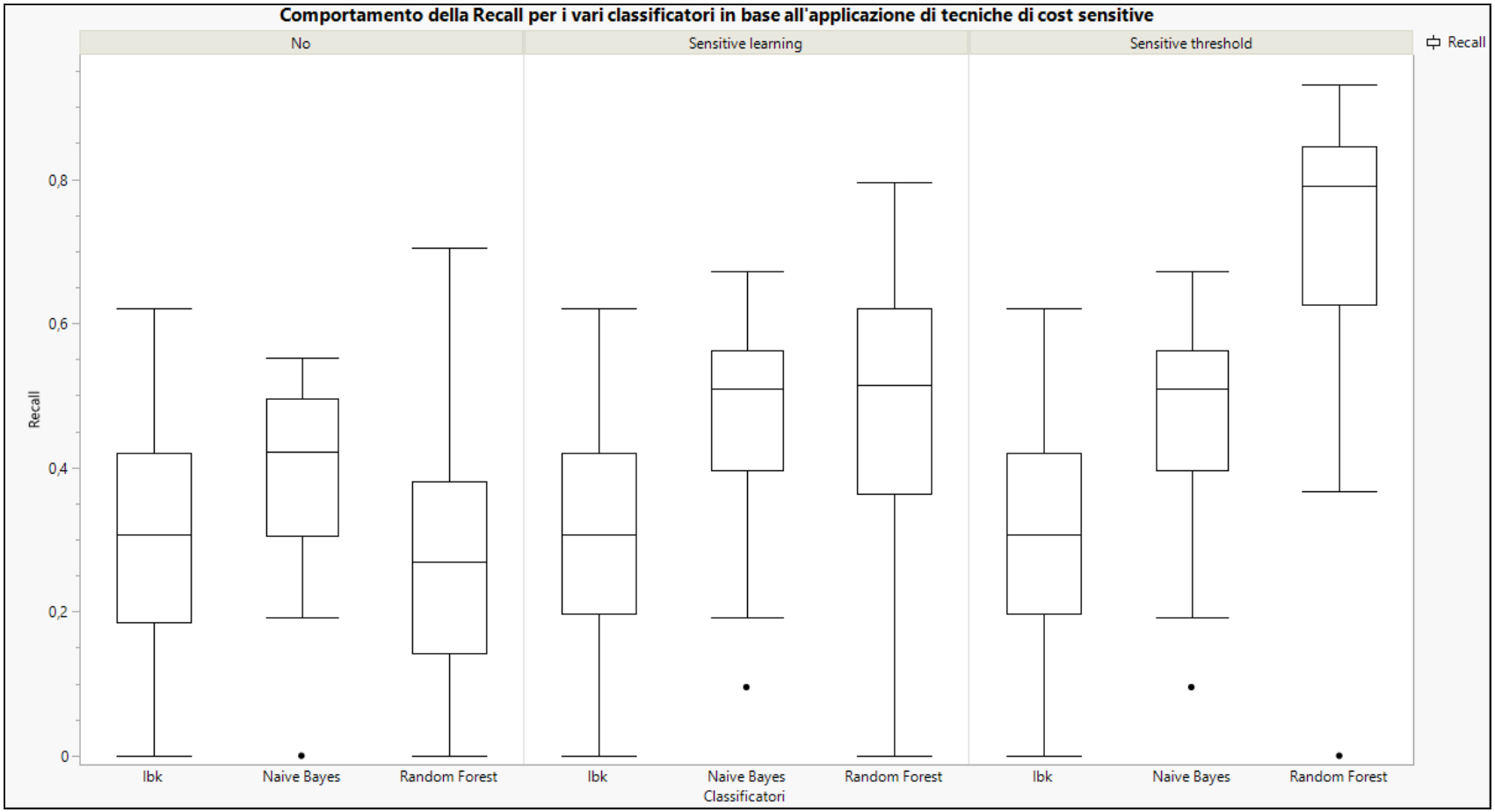
Bookkeeper - Caso specifico (2/4)

Considerazioni sulla accuracy dei classificatori

- Si può notare che nel momento in cui viene applicata una qualsiasi tecnica di balancing l'accuratezza di ogni classificatore peggiora
- Questo risultato non è una sorpresa perché nel testing set si hanno più negativi che positivi (perché nel testing set non si applica il balancing) e il classificatore restituisce più positivi nel caso in cui si applica il balancing
- Questo porta a dire che il classificatore sbaglierà tanto
  - aumentano i falsi positivi
  - diminuiscono i falsi negativi

Zookeeper - Caso specifico (3/4)

Recall dei classificatori in base alle tecniche di cost sensitive



Zookeeper - Caso specifico (4/4)

Considerazioni sulla recall dei classificatori

- Ad eccezione del classificatore Ibk in cui la *recall* rimane costante, la *recall* migliora nel caso in cui si applicano tecniche di cost sensitive
- Questo risultato non è una sopresa perché il costo per un falso negativo è dieci volte maggiore di quelle per un falso positivo e quindi l'idea di base è che il classificatore tende a classificare una classe come buggy con più facilità

Conclusioni

- La risposta, in seguito all'analisi dei risultati, è che non esiste il classificatore migliore, ma esso dipende dal contesto in cui ci si trova
- Nell'ingegneria c'è sempre un compromesso ed infatti bisogna trovare il classificatore adatto alle esigenze che si hanno
  - Ad esempio, si potrebbe volere un valore abbastanza alto di *recall* perché questo porterebbe ad avere un minor rischio di non testare una classe buggy
- Per entrambi i progetti, il classificatore *Random Forest* presenta la *recall* migliore
- *Naive Bayes* è il classificatore che presenta una *precision* più alta e *AUC* migliore

Repository di lavoro

Repository *GitHub*

La repository *GitHub* contenente il codice sorgente che ha sviluppato i risultati ottenuti è disponibile al seguente link:

`https://github.com/alessandrochillotti/isw2-deliverable-2.git`

Progetto su *SonarCloud*

L'analisi di *SonarCloud* che mostra come il codice sorgente abbia code smell pari a 0 è disponibile al seguente link:

`https://sonarcloud.io/project/overview?id=alessandrochillotti_isw2-deliverable-2`