Machine Learning for SE

Giulia Menichini 0298906

Indice

Obiettivi

Studio empirico finalizzato a misurare l'effetto di tecniche di sampling, cost sensitive classifier e feature selection sull'accuratezza di modelli predittivi per la localizzazione di bug nel codice sorgente di progetti OpenSource.

Sono state utilizzate le seguenti tecnologie:

Jira	GitHub	Git	Weka	Jmp	SonarCloud

I progetti analizzati sono: BOOKKEEPER ed OPENJPA

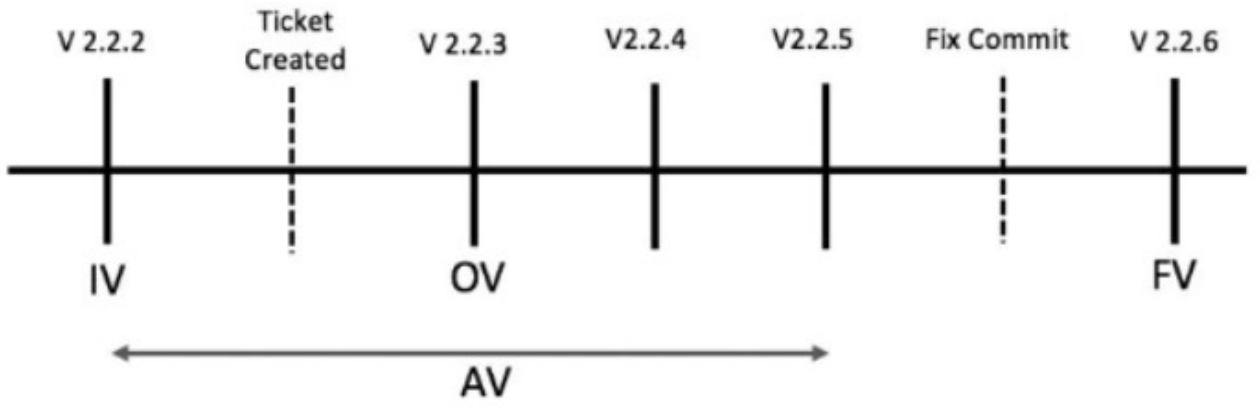
Analisi del contesto

Nell'ambito dell'ingegneria del software, è di fondamentale importanza ottimizzare la fase di testing degli artefatti software al fine di individuare tempestivamente eventuali bug. In questo contesto, l'utilizzo di tecniche di Machine Learning consente ai tester di ottenere previsioni sulla presenza di bug nelle classi, facilitando così l'identificazione e la risoluzione dei problemi nel software. Questo approccio consente ai tester di concentrare le proprie attività di testing sulle classi considerate più problematiche, consentendo una gestione più efficiente delle risorse e una migliore qualità complessiva del software.

Una classe è **buggy** se vengono fatte delle modifiche funzionali relative ad un commit che ha un commento relativo ad un ticket di tipo bug.

Nozioni Preliminari

- **Revisione** = ogni commit produce una revisione
- **Release** = revisione pubblicata, può includere nuove funzionalità, correzioni di bug, miglioramenti delle prestazioni o della sicurezza
- **Ticket** = unità di lavoro o attività che rappresenta una specifica richiesta o problema che deve essere gestito
- **IV** = Injected Version, versione in cui il bug è stato iniettato
- **OV** = Opening Version, versione in cui il ticket è stato aperto
- **FV** = Fixed Version, versione in cui il bug è stato fixato. Tale versione non presenta quindi alcun bug.
- **AV** = Affected Version, versioni affette dal bug AV = [IV,FV)



$AV \in [IV,FV)$

Raccolta dei dati

- I dati riguardanti le release sono stati raccolti utilizzando le API REST di JIRA effettuando una interrogazione mediante l'url: **«https://issues.apache.org/jira/rest/api/2/project/»** con la quale viene restituito un file json, tale file contiene l'ID della release, il nome e la data, si procede poi con l'ordinamento delle versioni cronologicamente.
- Vengono successivamente raccolte le informazioni sui ticket tramite una query con issueType=«**Bug**», status=«**closed**» o «**resolved**» e resolution = «**fixed**». Come per le release tale query restituisce un file json. Analizzando il file si ottengono quindi le informazioni sul campo «key», «versions» e «created» che indicano rispettivamente l'ID del ticket, le affected versions e la creation date. Con l'ID vengono trovati i commit effettuati per risolvere i bug (relativi al ticket). La creation date permette, effettuando un merge tra questa informazione e le informazioni sulle release, di settare la OV come la release la cui data precede la data di creazione del ticket. Se l'affected version è riportata viene settata l'IV come la prima release presente avendo cura di verificare che questa sia consistente:
  - Se OV = 1 allora IV = 1 perché IV $\leq$ 0V per definizione.
  - Se IV != 0 verifico che FV > IV e OV $\geq$ IV, se si setto AV = [IV, FV) altrimenti setto errore , IV = 0 e svuoto la lista di AV
  - Se FV = IV allora AV = 0, il bug è stato iniettato e fixato nella stessa release.
- La FV viene settata andando ad effettuare un merge delle informazioni ottenute da github (tramite git log) e le informazioni sulle Release. Si va infatti a cercare la data dell'ultimo commit e settare la FV come la release immediatamente successiva a tale data.

Proportion

Per i ticket che non hanno l'informazione sull'IV viene utilizzato il metodo **Proportion**:

- Esiste una proporzionalità nella distanza (in termini di release) tra la Opening Version e Injected Version e quella tra Fixed Version e Injected Version. Si calcola quindi il fattore di proporzionalità:

$$P = \frac{FV - IV}{FV - OV}$$

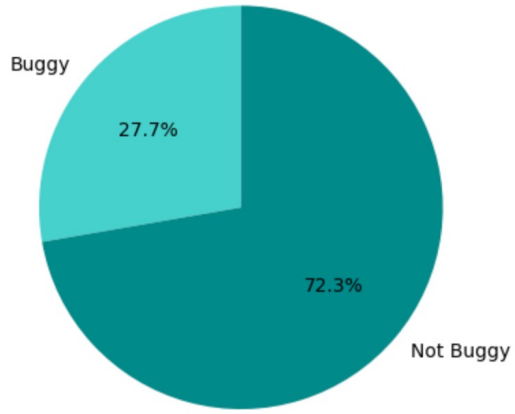
Per i ticket che non hanno IV viene calcolato il valore di **P** come la media dei **P** calcolati sul 2% dei ticket (**Moving Windows**) con IV valido e applicando la formula

$$IV = FV - (FV - OV) * P$$

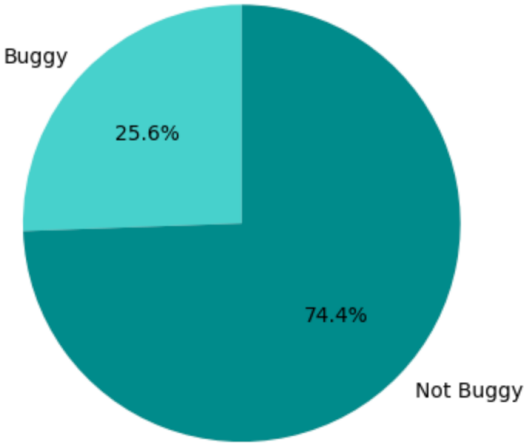
Se sono presenti ticket con FV = OV allora IV = FV.

- Una volta definito ed applicato il metodo Proportion, viene misurata la buggyness della classe in ogni release del progetto. In particolare La bugginess viene settata a **true** per tutti i file .java delle release presenti nell'intervallo [IV, FV).

BOOKKEEPER



OPENJPA



Generazione del Dataset

L'addestramento del modello è basato sull'**apprendimento supervisionato**. La macchina viene fornita di un set di dati di training che sono già stati etichettati correttamente. Successivamente, il modello viene valutato sulla sua capacità di generalizzare l'apprendimento su un set di dati di testing che non è stato ancora visto dalla macchina. In questo modo, si può valutare l'efficacia del modello nel generalizzare e applicare le conoscenze apprese durante l'addestramento a nuovi dati, consentendo di valutare l'affidabilità e le prestazioni del modello nel contesto specifico.

Per la creazione del dataset sono state acquisite tutte le informazioni ritenute importanti al fine di determinare la defectiveness di una determinata classe.

L’ultimo 50% delle release viene rimosso, poiché è stato dimostrato che questo riduce il missing rate (percentuali di classi che sono falsi negativi / tutte le classi positive) fino al 10%, le release più recenti hanno una maggiore probabilità di risultare snoring, ovvero classi di tipo bug ma la cui difettosità non è stata ancora formalizzata dal fix del bug.

Dataset

Il dataset contiene le seguenti metriche:

SIZE = Linee di codice totali

LOC_TOUCHED = somma durante le revisioni di (added+deleted) LOC

LOC_ADDED = somma operazioni di added durante le revisioni

MAX_LOC_ADDED = massimo LOC added durante le revisioni

AVG_LOC_ADDED = LOC added medio durante le revisioni

CHURN = somma durante le revisioni di (added-deleted) LOC

MAC_CHURN = valore massimo di churn

AVG_CHURN = valore medio di churn

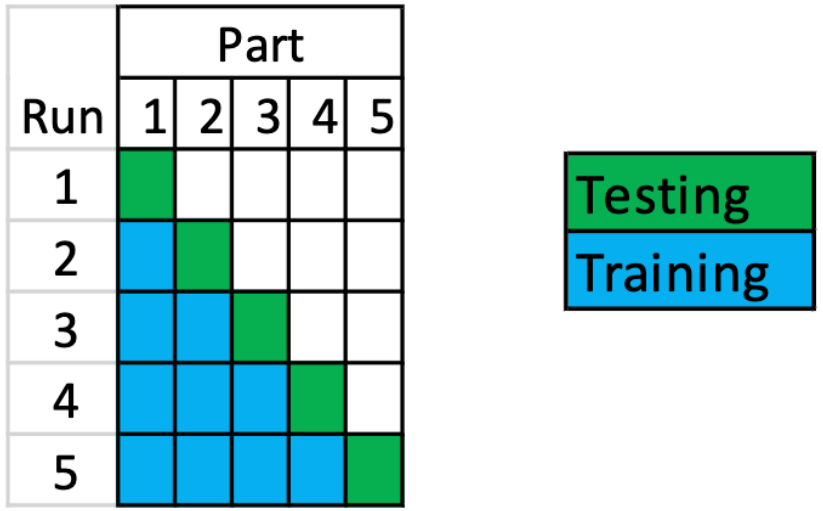
NR = numero di revisioni

NAUTH = numero di autori

Per poter analizzare i dati con Weka, il file csv creato è stato convertito in file ARFF.

Machine Learning

L'obiettivo principale di questa fase è valutare quale combinazione di classificatore e metodologia (feature selection, cost-sensitive classifier e balancing) garantisca la maggiore accuratezza in termini di metriche sul dataset appena creato. Nel contesto di questo studio, è stata adottata la tecnica "**Walk Forward**", che prevede la suddivisione dei dati storici in diverse parti ordinate, destinate alle fasi di addestramento e di test. Questo processo viene ripetuto più volte, avanzando progressivamente nel tempo e mantenendo l'ordinamento temporale reale dei dati. In questo caso specifico, ogni parte del dataset rappresenta una release.



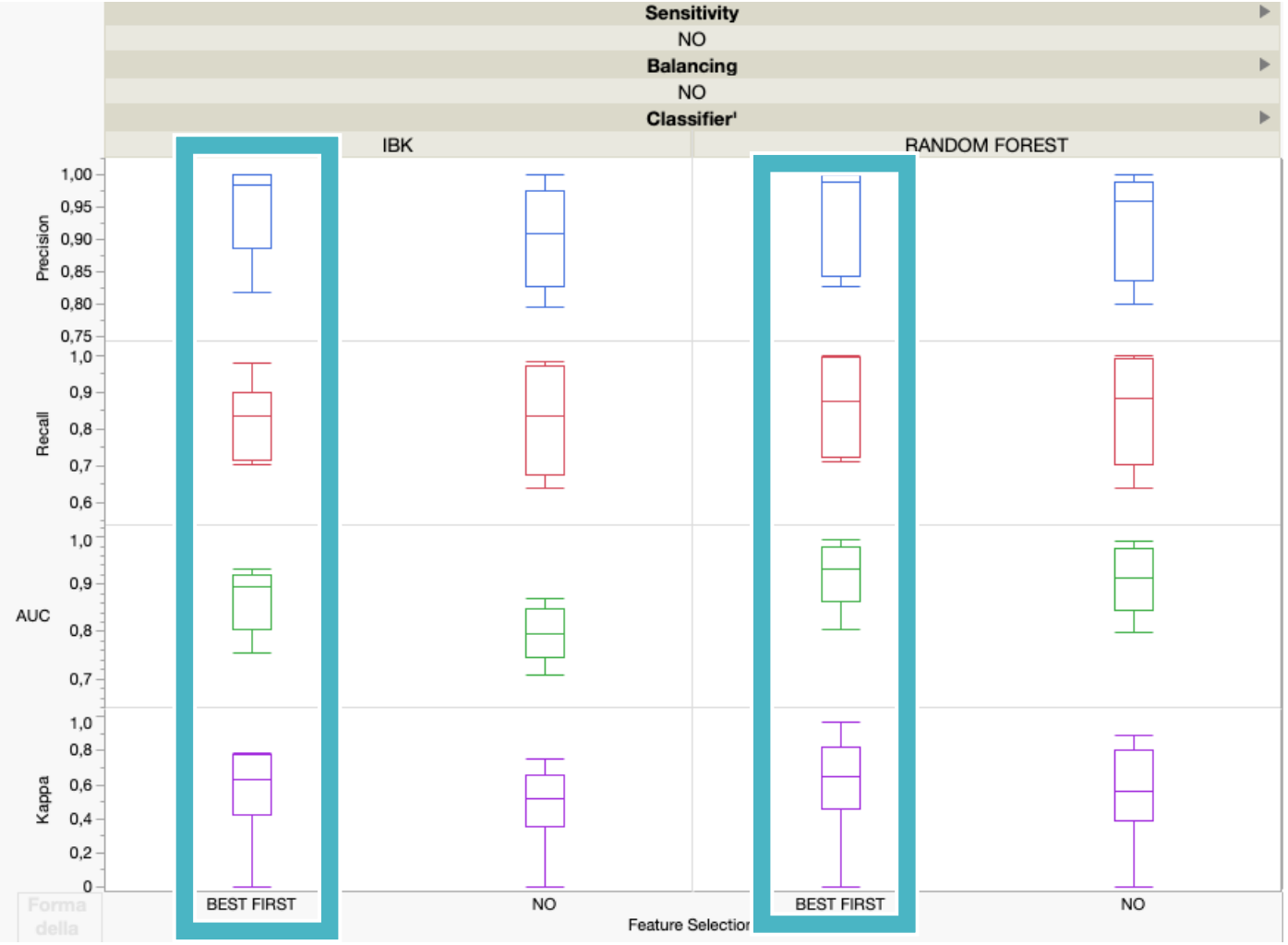
Machine Learning (2)

- I classificatori testati sono i seguenti:
  - **IBK, Naive Bayes, Random Forest**
- Le tecniche di bilanciamento evitano dataset sbilanciati verso una direzione garantendo quindi che il classificatore veda lo stesso numero di istanze per ogni classe. Sono state testate le seguenti metodologie:
  - **No sampling, Oversampling**, **Undersampling, SMOTE**
- Le tecniche di Feature Selection selezionano un sottoinsieme delle caratteristiche (o feature) più rilevanti per la previsione del modello e scartano quelle meno informative
  - **Best First, No Feature Selection**
- Le tecniche di Cost Sensitive tiene conto dei costi associati alla classificazione errata
  - **No CostSensitive, Sensitive threshold, Sensitive Learning**

```
//peso assegnato alla matrice di costo
private static CostMatrix createCostMatrix(double weightFalsePositive, double weightFalseNegative) {
    CostMatrix costMatrix = new CostMatrix(2);
    costMatrix.setCell(0, 0, 0.0);
    costMatrix.setCell(1, 0, weightFalsePositive);
    costMatrix.setCell(0, 1, weightFalseNegative);
    costMatrix.setCell(1, 1, 0.0);
    return costMatrix;
}
```

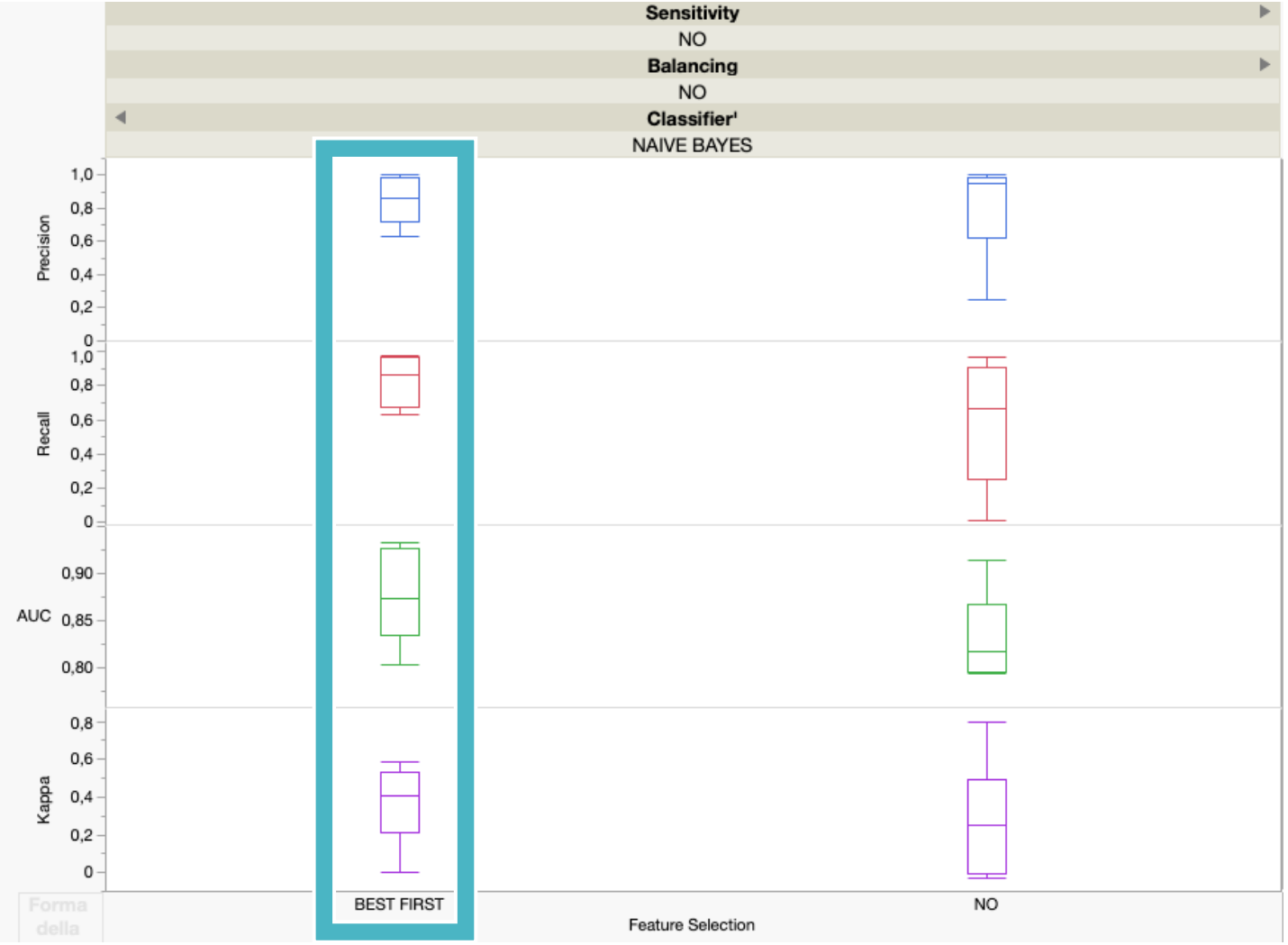
CFN = 10 * CFP

Risultati Bookkeeper con FS (IBK – RF)



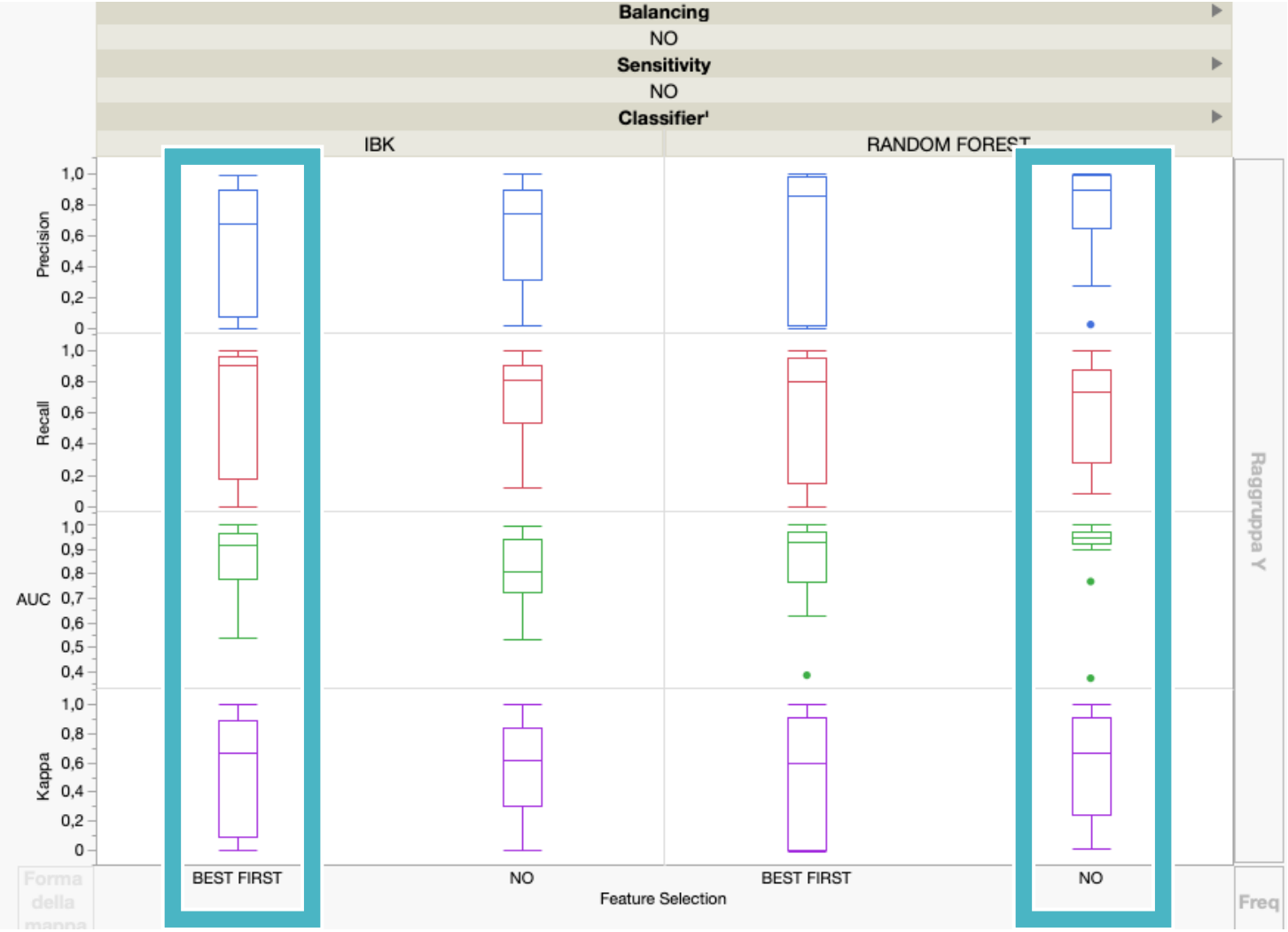
Dal grafo si evince che applicando la Feature Selection al classificatore IBK si ottengono risultati migliori in termini di mediana per Precision, AUC e Kappa la Recall rimane invece costante. Visto il miglioramento ottenuto soprattutto nella metrica Kappa e l'uguaglianza per la Recall in questo contesto viene applicata la Feature Selection. A parità di prestazioni infatti si preferisce sempre, ove possibile, ridurre dimensionalmente il dataset per garantire processi di addestramento più brevi. Discorso analogo per il classificatore Random Forest, tutte le metriche migliorano e la Recall rimane costante. Anche per questo classificatore si sceglie quindi di applicare la FS.

Risultati Bookkeeper con FS (NB)



Per il classificatore Naive Bayes con l'applicazione della FS si ottiene un miglioramento nelle metriche: Recall, AUC e Kappa. La Precision ha prestazioni leggermente migliori senza applicazione di FS. Visto il miglioramento ottenuto e il raggiungimento di una Precision comunque molto alta (<0.8) si sceglie di applicare tale tecnica al classificatore.

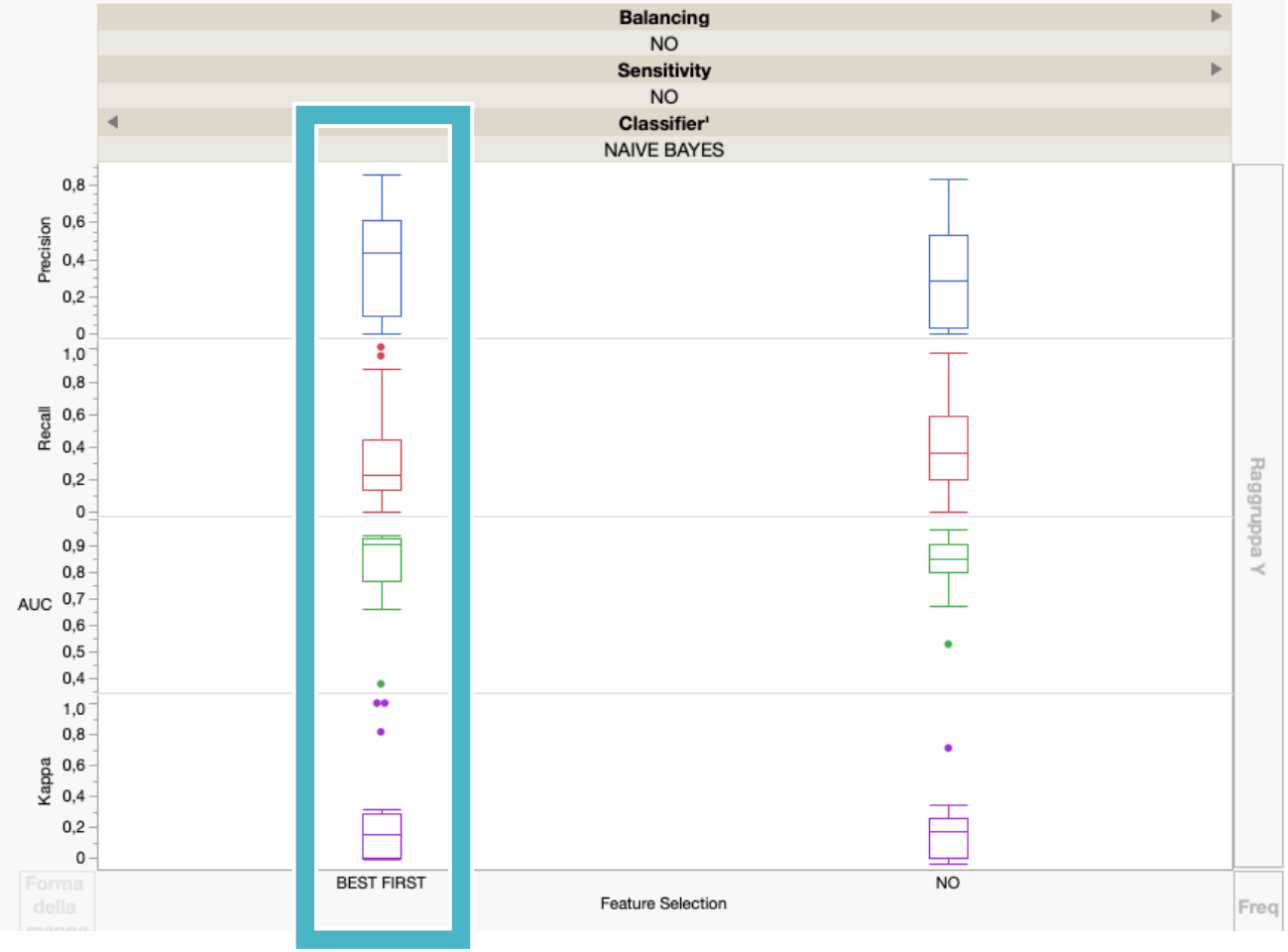
Risultati OpenJPA con FS (IBK – RF)



Per il classificatore **IBK** le metriche AUC e Recall migliorano le loro prestazioni del 10% con l'applicazione della FS. La Metrica Kappa garantisce un miglioramento del 5% mentre la Precision ha un peggioramento del 6%. Con i miglioramenti ottenuti e l'abbondante superamento della soglia 0.5 per la metrica Precision, si sceglie di applicare la FS a questo classificatore.

Discorso differente per il classificatore **Random Forest** in cui tutte le metriche, eccezion fatta per la Recall che rimane costante, peggiorano con l'applicazione della FS in particolare Kappa scende di circa il 10%. Si sceglie quindi di non applicare tale tecnica a questo classificatore.

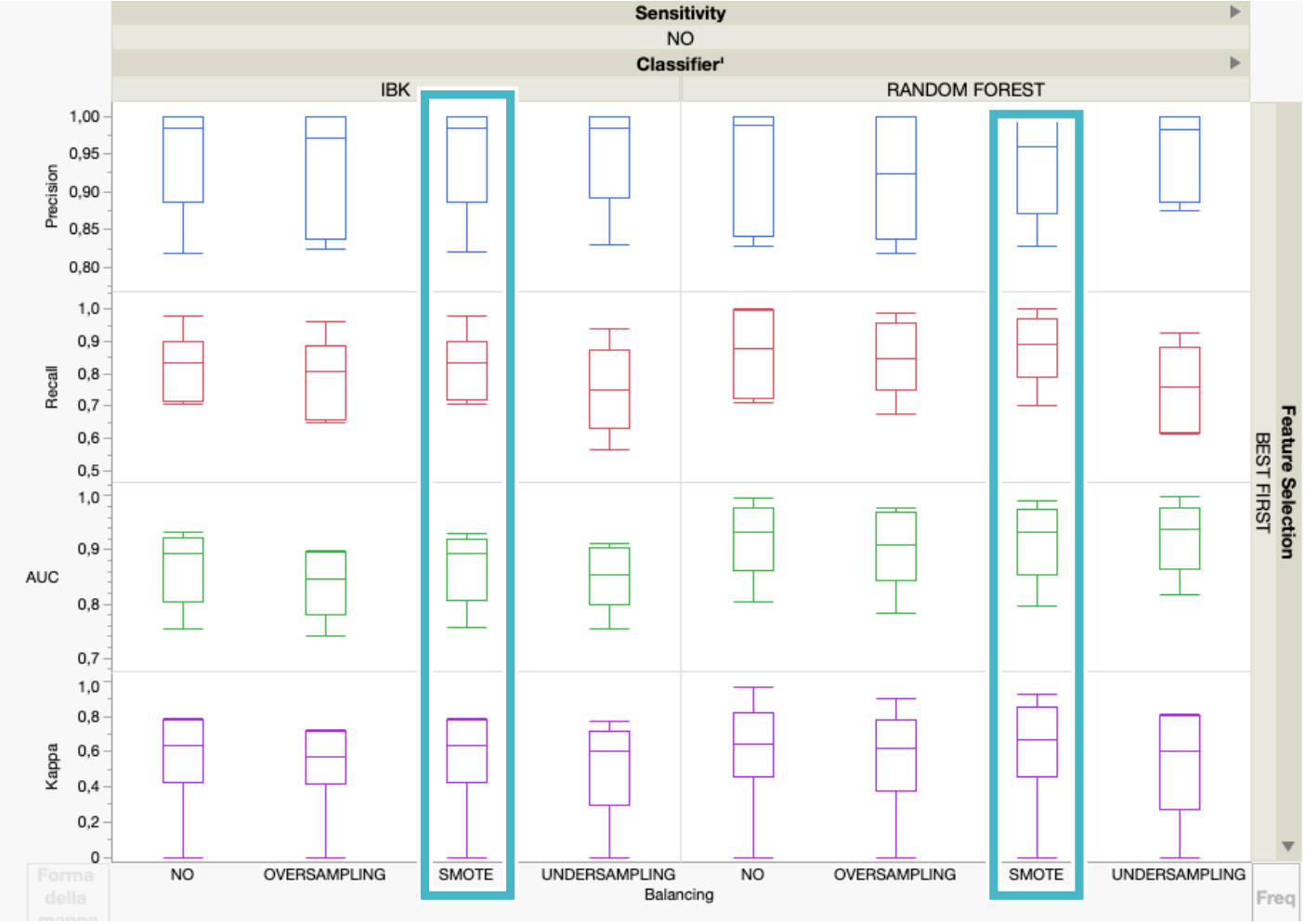
Risultati OpenJPA con FS (NB)



Naive Bayes è il classificatore che a questo punto dell'analisi registra prestazioni peggiori.

Per il classificatore Naive Bayes si ottengono prestazioni migliori in termini di AUC e Precision con l'applicazione della FS la quale però fa ottenere una Recall molto bassa (<0.5) e un valore Kappa circa costante (ha una variazione di circa 0.1%). Tuttavia senza applicare la FS la metrica che più ne risente è la Precision (< 0.3). Viste le basse prestazioni nel generale e volendo valutare le prestazioni medie si sceglie di applicare la tecnica di FS a questo classificatore.

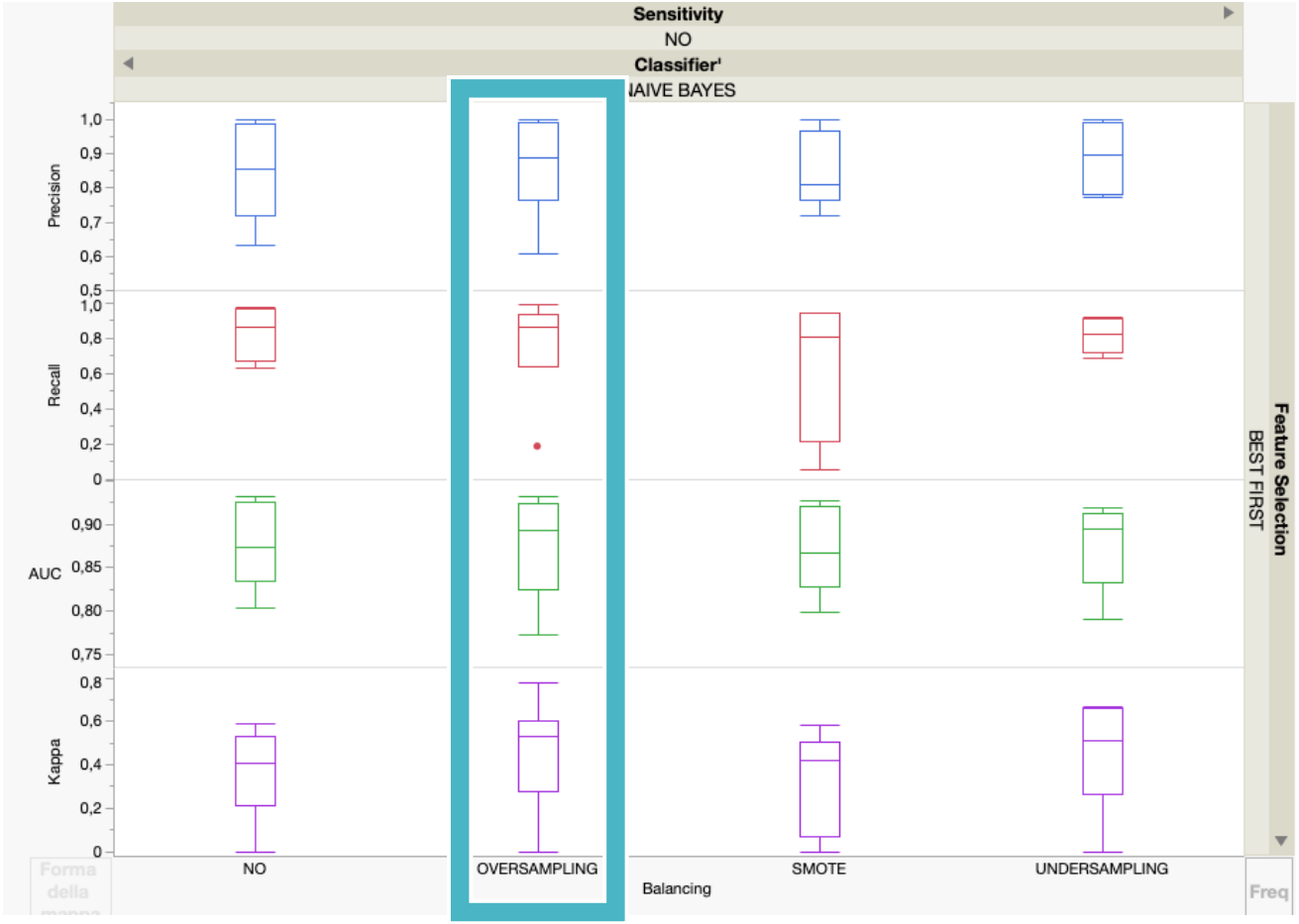
Bookkeeper – FS + Balancing (IBK – RF)



Per il classificatore IBK le quattro metriche coinvolte migliorano le proprie prestazioni con l'applicazione della tecnica SMOTE pur non discostandosi molto dal caso No Balancing. Il caso peggiore si registra con la tecnica di oversampling.

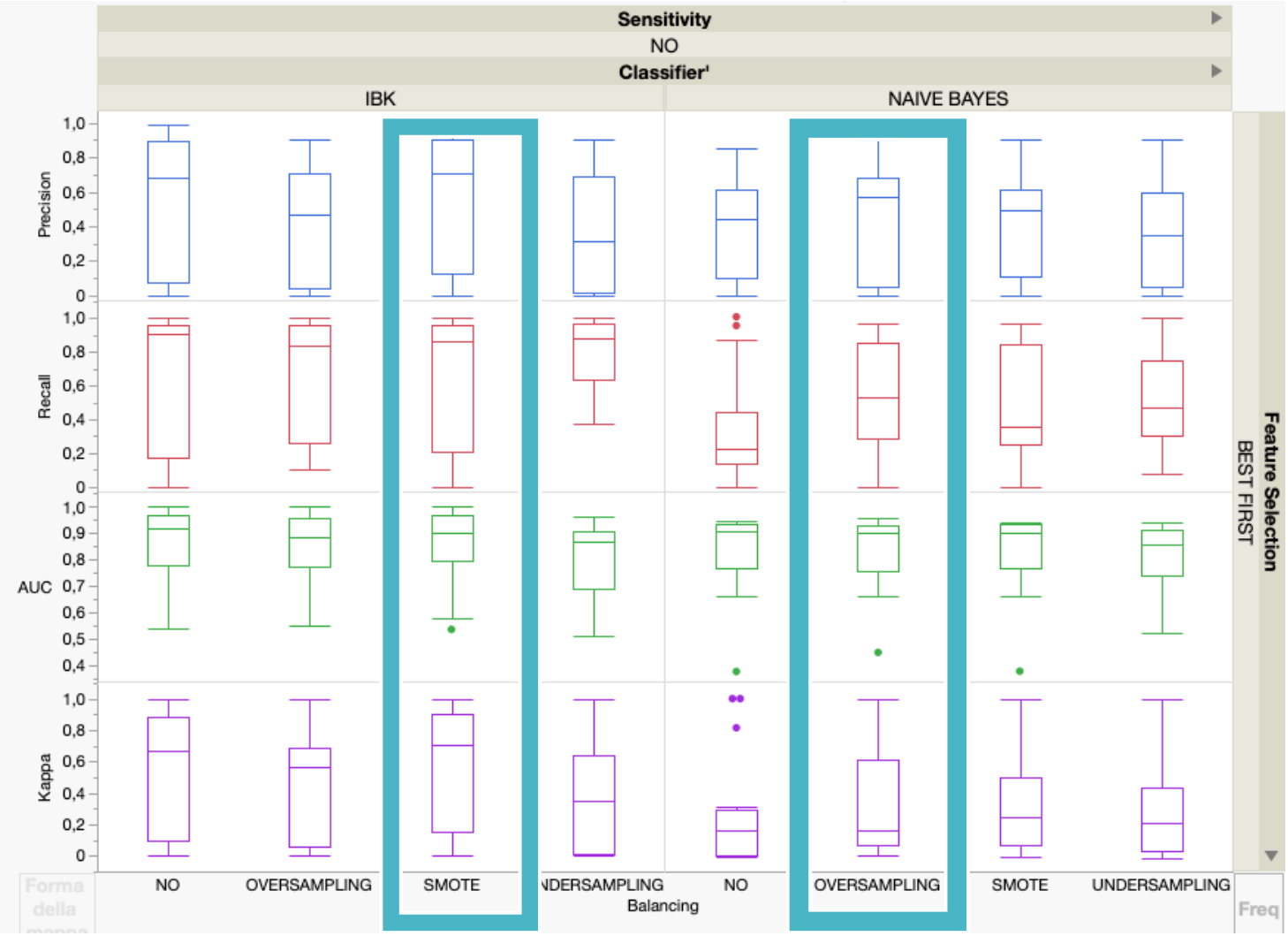
Per il classificatore Random Forest analogamente prestazioni migliori vengono raggiunte con l'applicazione della tecnica SMOTE la quale garantisce valori migliori in termini di Kappa (>0.6), Recall e AUC rimangono invece costanti rispetto al caso No Balancing. Precision risulta leggermente superiore senza applicare alcuna tecnica ma questa scelta fa perdere il 2% in Kappa e Recall.

Bookkeeper – FS + Balancing (NB)



Per il classificatore Naive Bayes valori migliori per Kappa, AUC e Precision vengono raggiunti con la tecnica di Oversampling. La Recall è maggiore nel caso No Balancing ma poiché il miglioramento di questa sola metrica è trascurabile (1%) si sceglie di applicare la tecnica di Oversampling che garantisce un miglioramento medio maggiore.

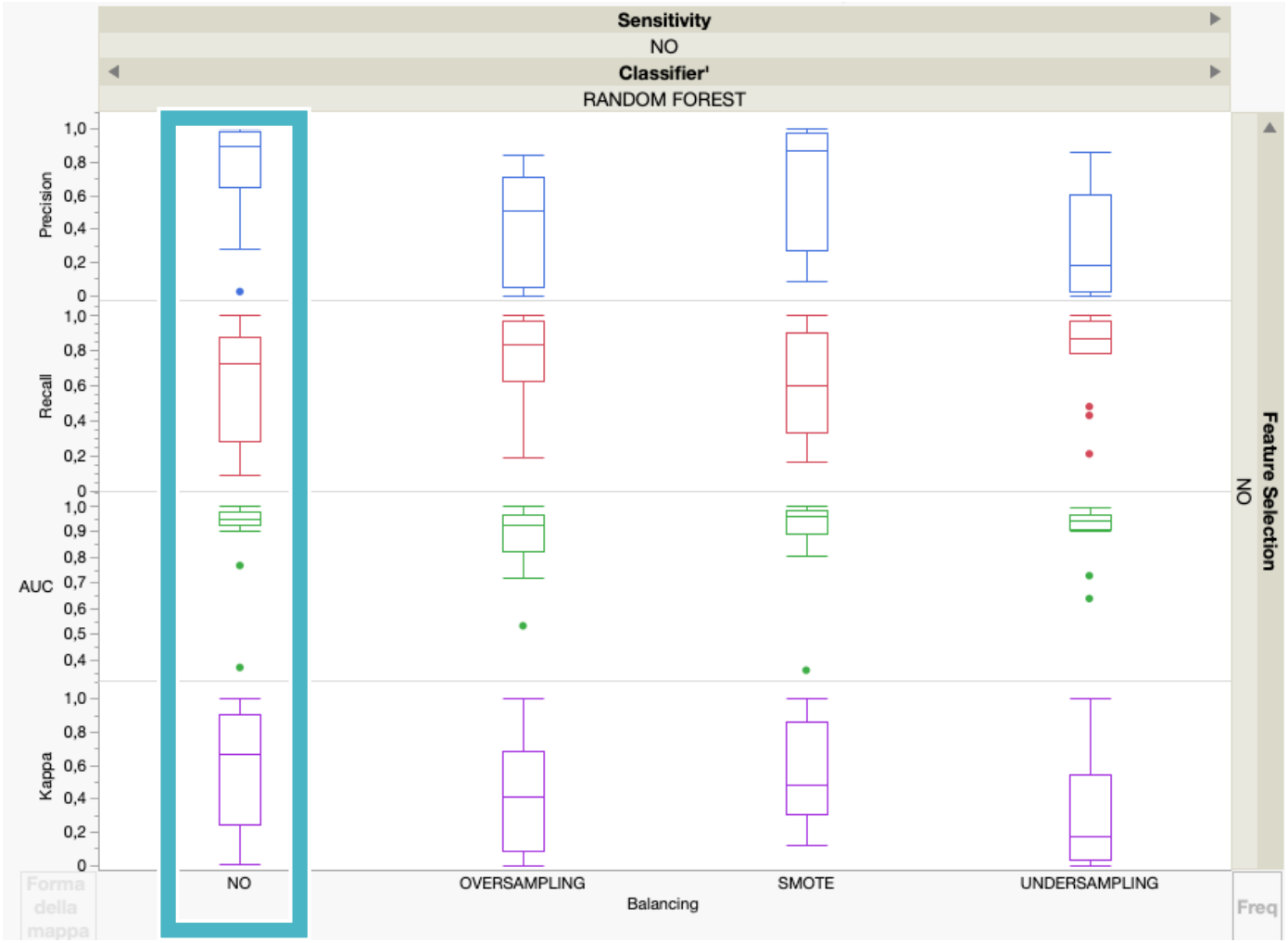
OpenJPA – FS + Balancing (IBK – NB)



Per il classificatore IBK è possibile scartare le tecniche di Undersampling ed Oversampling poiché degradano notevolmente le prestazioni. La scelta quindi ricade su No Balancing e tecnica SMOTE. Con l'applicazione della tecnica SMOTE vengono garantite Kappa e Precision maggiori, mentre si ottengono una Recall (di circa il 3 %) e AUC (di circa il 1 %) maggiori con il No Balancing. In media il comportamento è migliore con l'applicazione della tecnica SMOTE e volendo dare maggiore rilevanza alla metrica Kappa avendo le altre metriche sufficientemente alte con entrambe le metodologie, si sceglie di applicare tale tecnica.

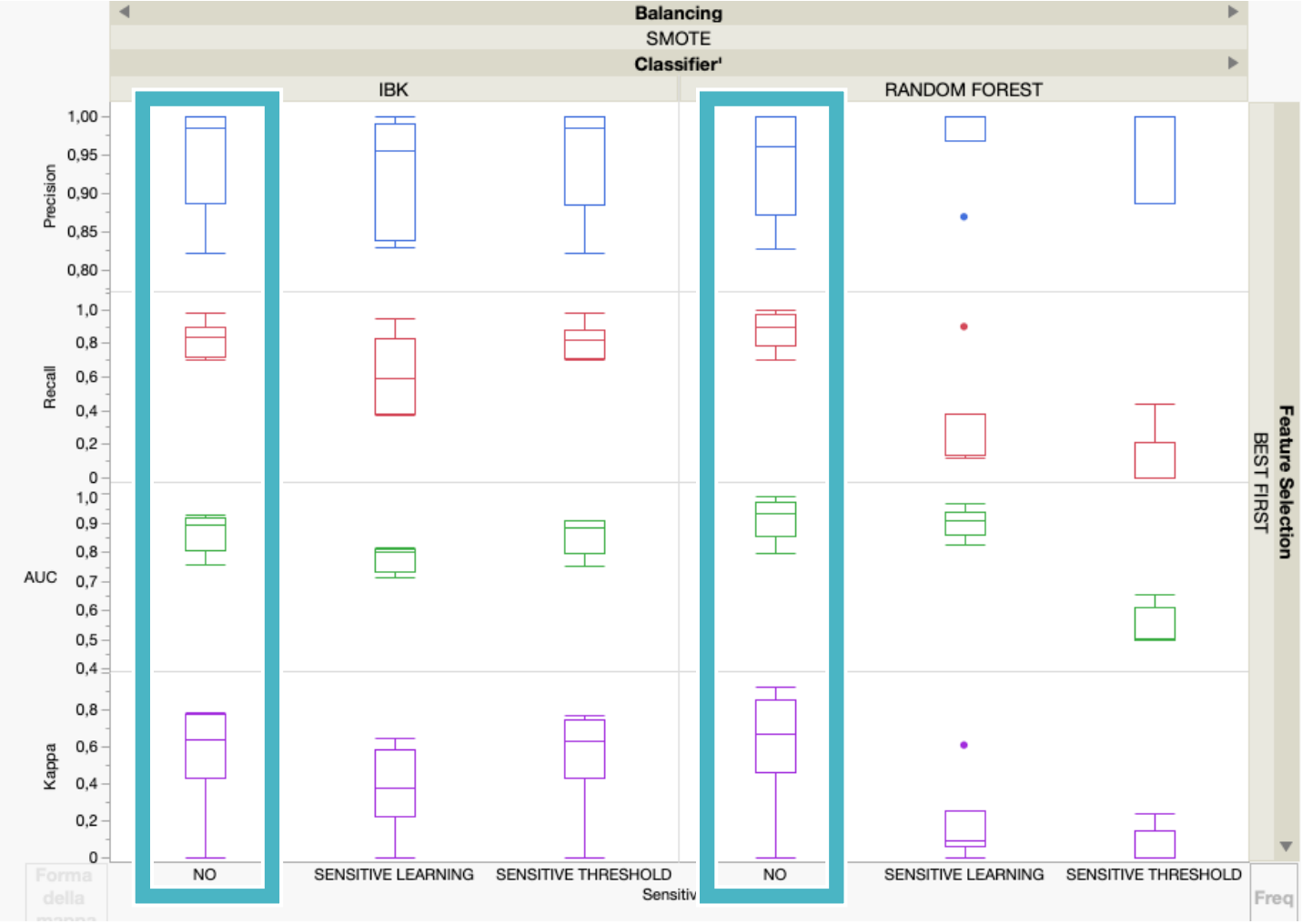
Per il classificatore Naive Bayes si sceglie di applicare la tecnica di Oversampling poiché è la sola che garantisce valori di prestazioni maggiori di 0.5 per Precision, Recall ed AUC. Valori di Kappa maggiori sono garantiti dalla tecnica SMOTE la quale fa però ottenere valori di Recall molto bassi (<0.4) si sceglie quindi di scartarla.

OpenJPA –Balancing (RF)



Per il classificatore Random Forest si sceglie di non applicare alcuna tecnica di Balancing. La non applicazione infatti permette il raggiungimento di valori maggiori per la metrica Kappa (> 0.6) e valori sufficientemente alti per le altre metriche coinvolte. In particolare la Precision risulta essere > 0.8, la Recall > 0.7 e AUC > 0.9. La tecnica che registra risultati peggiori è la tecnica di Undersampling la quale causa un notevole abbassamento della Precision (< 0.2) e di Kappa (<0.2).

Bookkeeper – FS + Sampling + CS (IBK-RF)



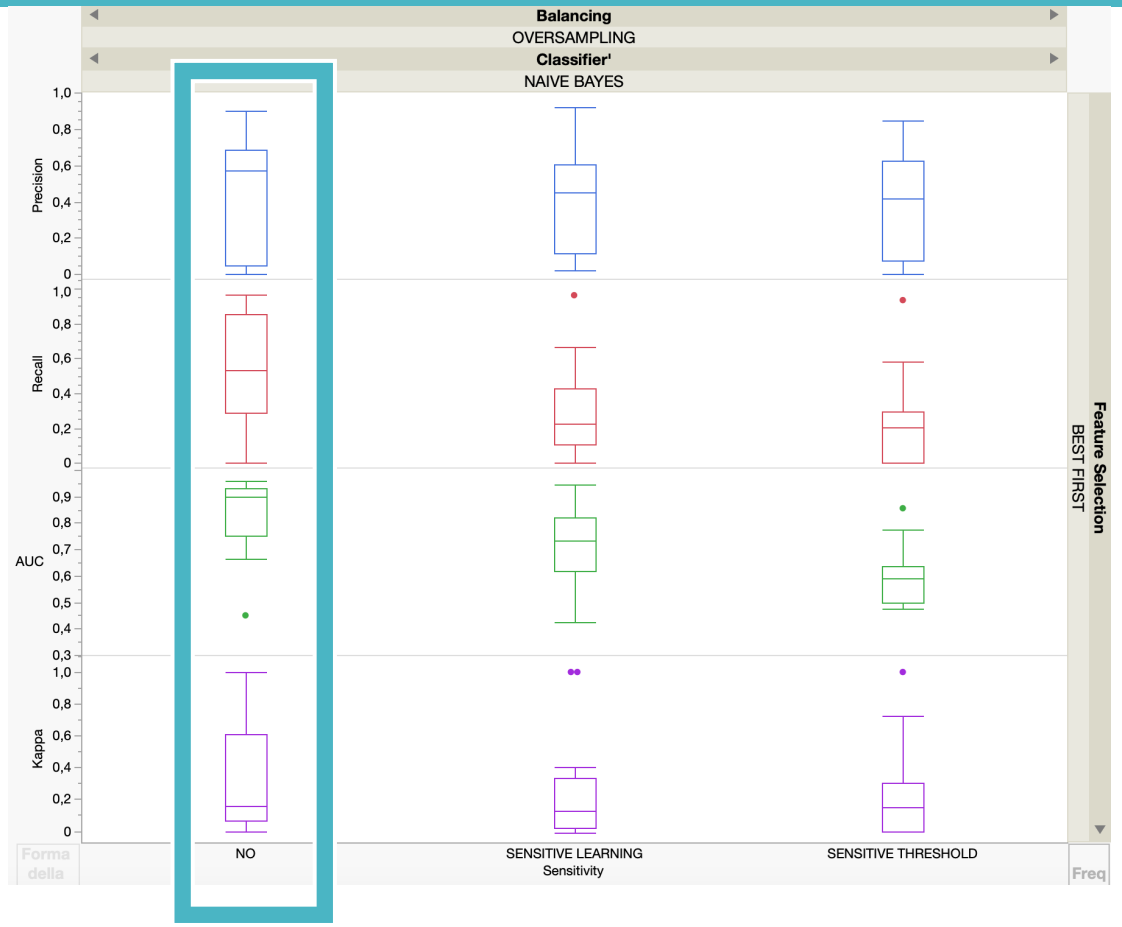
Come è possibile osservare dal grafo il cost sensitive classifier per entrambi i classificatori non migliora le prestazioni. Per il classificatore IBK infatti applicando tale tecnica si perde in Recall, AUC e Kappa. Con il classificatore Random Forest si acquista in Precision (raggiunge infatti valore massimo) ma causa un netto peggioramento della Recall (< 0.5). Per entrambi i classificatore si sceglie quindi di non utilizzarla.

Bookkeeper – FS + Sampling + CS (NB)

Discorso analogo può essere fatto per il classificatore Naive Bayes. Tutte le metriche, eccezion fatta per la Precision, peggiorano le loro prestazioni con l'applicazione del cost sensitive classifier. Si sceglie quindi di non utilizzarla.

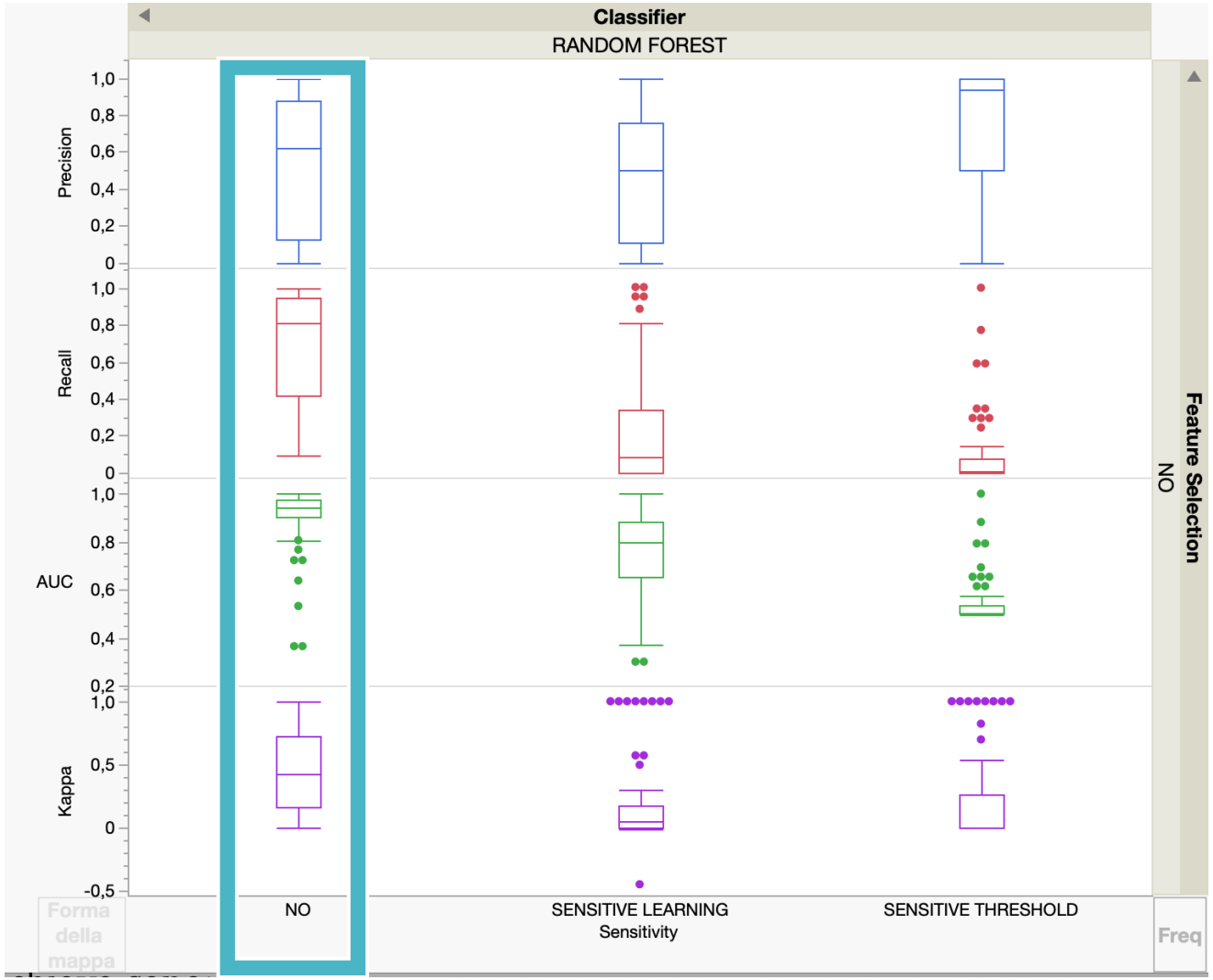
OpenJPA – Cost Sensitive (IBK - NB)

Per il classificatore IBK si decide di scartare la tecnica di Sensitive Learning la quale causa un drastico peggioramento in Recall (<0.2). Si sceglie quindi di non applicare alcuna tecnica di Cost Sensitive poiché si ottiene un miglioramento in termini di AUC, le altre metriche sono costanti rispetto al caso Sensitive Threshold



Per il classificatore Naive Bayes, come per IBK, si sceglie di non applicare la Cost Sensitive poiché l'applicazione causa il peggioramento delle quattro metriche coinvolte.

OpenJPA – Balancing + Cost Sensitive (RF)



Per il classificatore Random Forest si è deciso di non applicare Cost Sensitive Classifier poiché causa il peggioramento netto nella Recall (<<0.5) e valore Kappa prossimo a 0.

Conclusioni

Sulla base delle assunzioni precedentemente descritte (e non potendo fare una valutazione ottimale in senso assoluto) è stato possibile trarre le seguenti conclusioni :

Dataset/Classificatore	Feature Selection	Balancing	Cost-Sensitive Classifier
BOOKKEEPER (IBK)	BEST FIRST	SMOTE	NO
BOOKKEEPER(RF)	BEST FIRST	SMOTE	NO
BOOKKEEPER(NB)	BEST FIRST	OVERSAMPLING	NO
OPENJPA(IBK)	BEST FIRST	SMOTE	NO
OPENJPA(RF)	NO	NO	NO
OPENJPA(NB)	BEST FIRST	OVERSAMPLING	NO

Conclusioni(2)

- La scelta del classificatore ottimale per Bookkeeper dipende dal contesto. E' possibile escludere il classificatore Naive Bayes con il quale si ottengono prestazioni peggiori. I classificatori RF e IBK hanno prestazioni pressoché simili Il valore di Precision infatti è costante mentre Recall, AUC e Kappa risultato superiori con l'utilizzo del classificatore Random Forest che risulta essere in media il classificatore migliore per questo dataset. Si ottengono tuttavia con entrambi i classificatori buone prestazioni. Per il dataset OpenJPA Il classificatore che garantisce prestazioni in media migliori è IBK. Rispetto a NB e RF infatti permette di ottenere valori sufficientemente alti sia in Recall che in Precision (> 0.7) con il valore Kappa maggiore (> 0.5). Il classificatore peggiore risulta essere NB come accadeva per Bookkeeper con valore di Precision, Recall e Kappa inferiori.
- In tutti i casi l'AUC è superiore al valore 0.5 e kappa superiore al valore 0 indice del fatto che il classificatore riesce a lavorare meglio di un classificatore che sceglie randomicamente quale label assegnare alla classe.

Link utili

- https://sonarcloud.io/summary/overall?id=giulia97-w_ISW2--project
- https://github.com/giulia97-w/ISW2--project