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

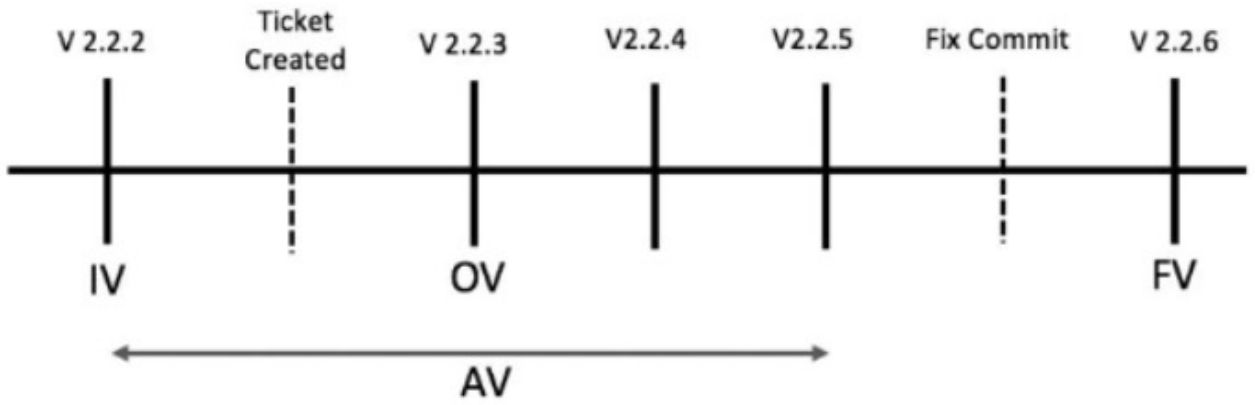
Analisi del contesto

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)



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:

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

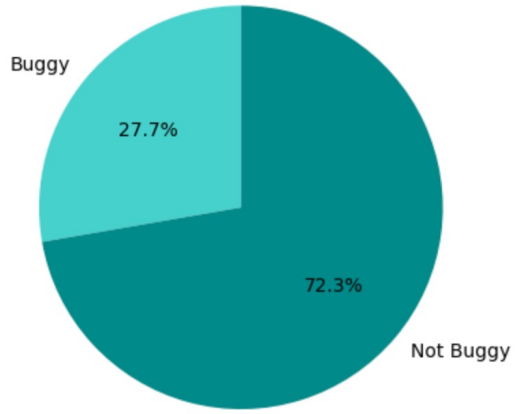
Per i ticket che non hanno IV viene calcolato il valore di **P** come la media dei **P** calcolati sul 2% dei ticket (**Moving Windows**) con IV valido e applicando la formula

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

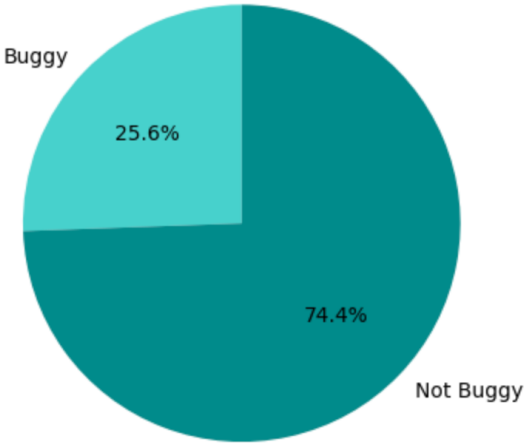
Se sono presenti ticket con FV = OV allora IV = FV.

- 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

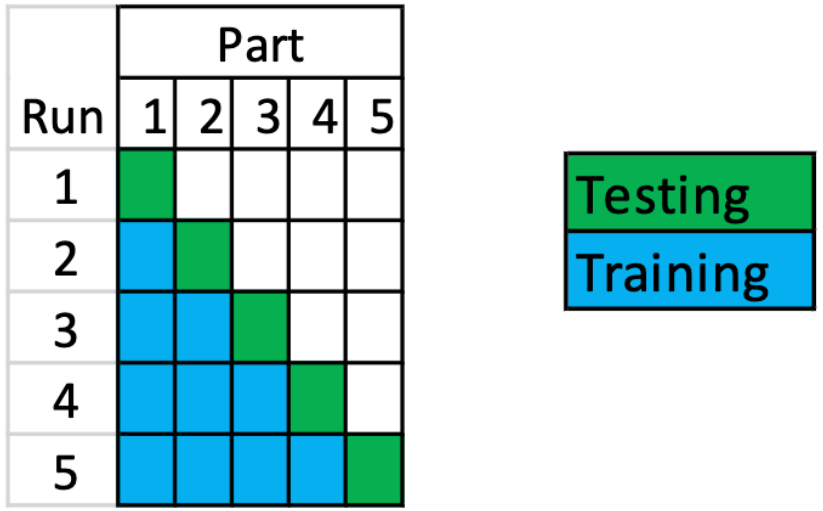
NR = numero di revisioni

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.



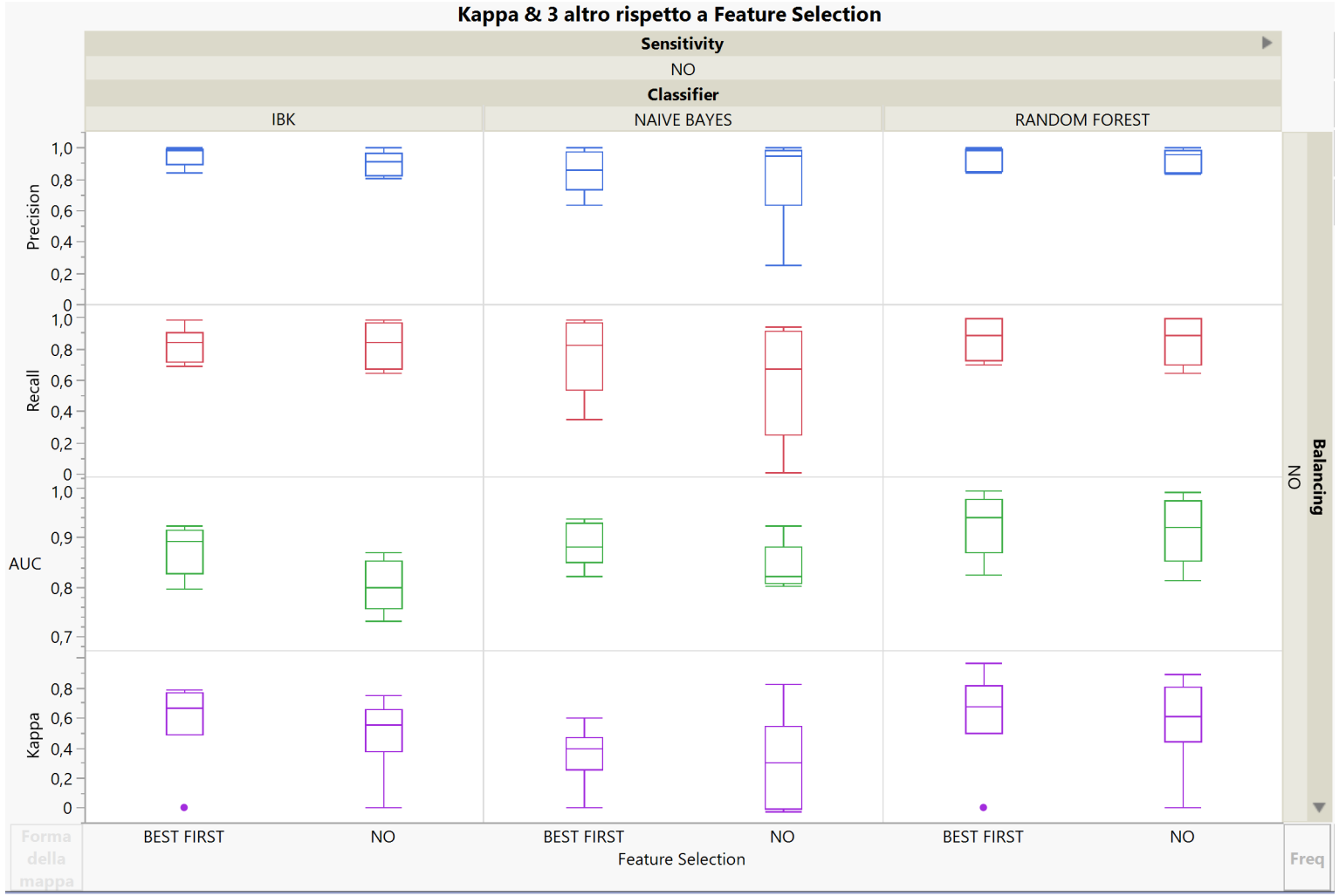
Machine Learning (2)

- 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;
}
```

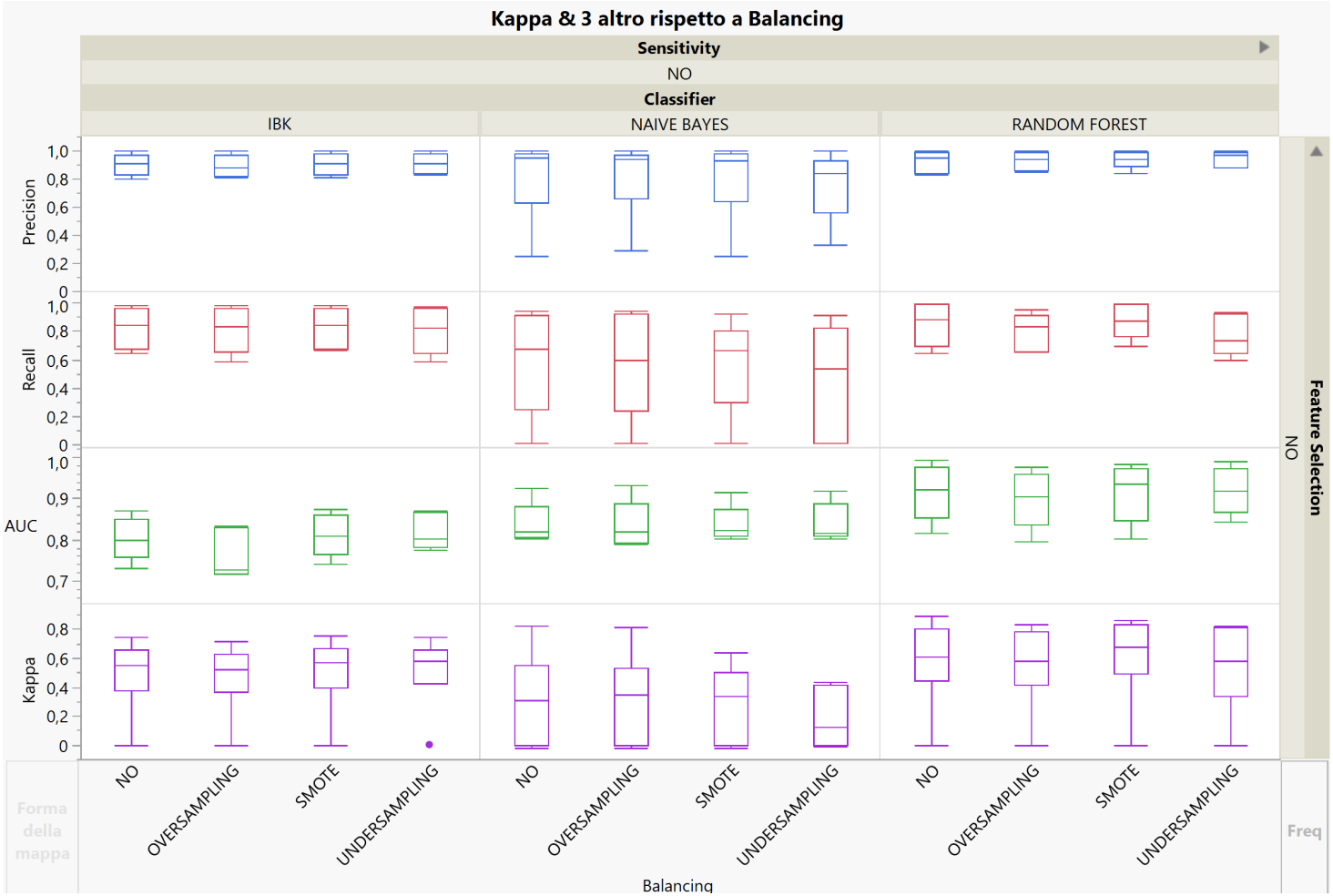
CFN = 10 * CFP

Risultati Bookkeeper con FS



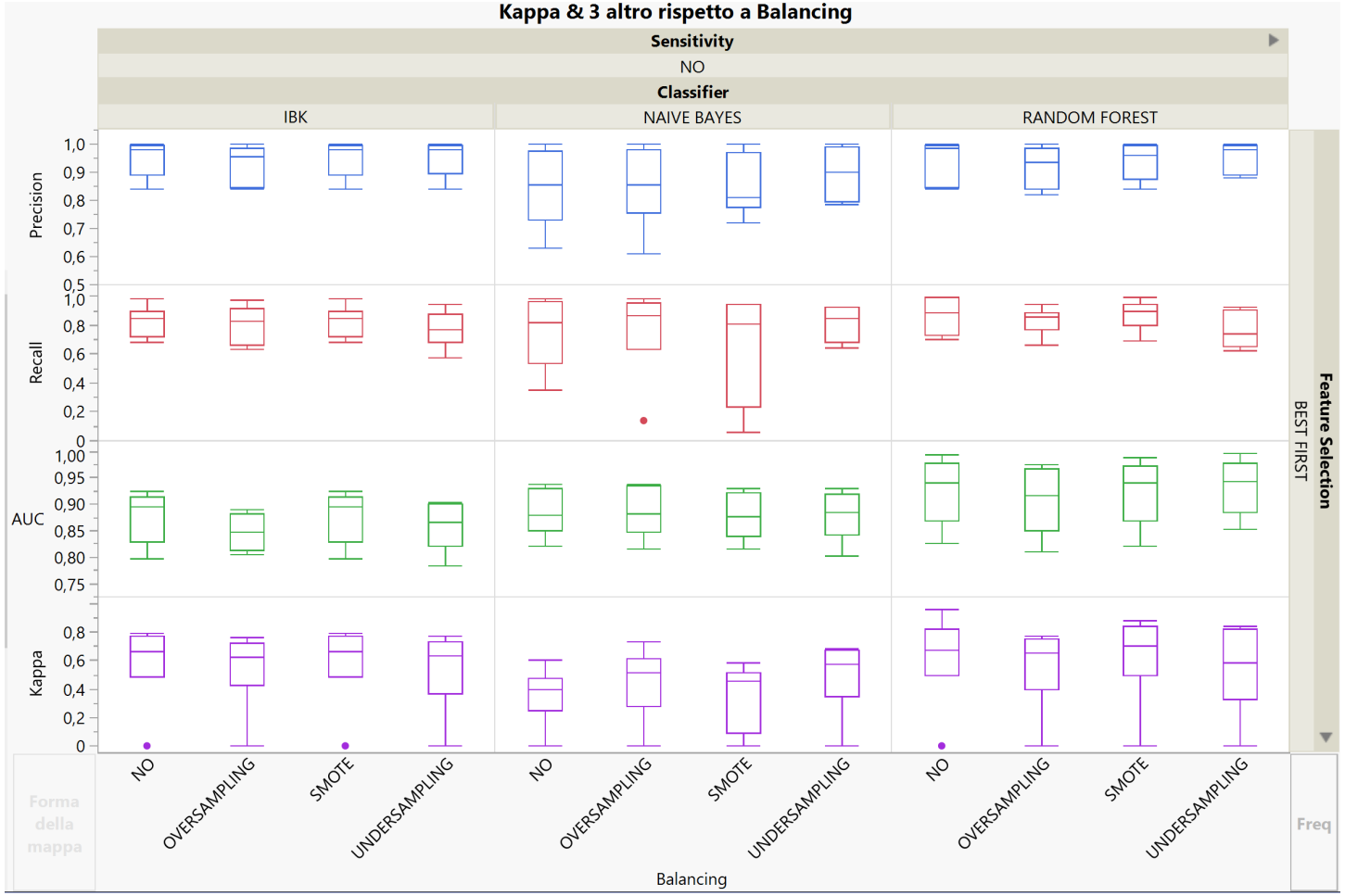
Dall'analisi del grafo si evince che l'applicazione della Feature Selection ai tre classificatori produce risultati migliori in termini di mediana per le metriche Precision, AUC e Kappa. La Recall migliora nel classificatore NB e rimane costante nei classificatori IBK e RF. Considerando il miglioramento ottenuto nelle quattro metriche per NB e il miglioramento in tre metriche per IBK e RF, insieme alla parità di Recall tra IBK e RF, si decide di applicare la Feature Selection in questo contesto. A parità di prestazioni infatti si preferisce sempre, ove possibile, ridurre dimensionalmente il dataset per garantire processi di addestramento più brevi.

Risultati Bookkeeper con Balancing



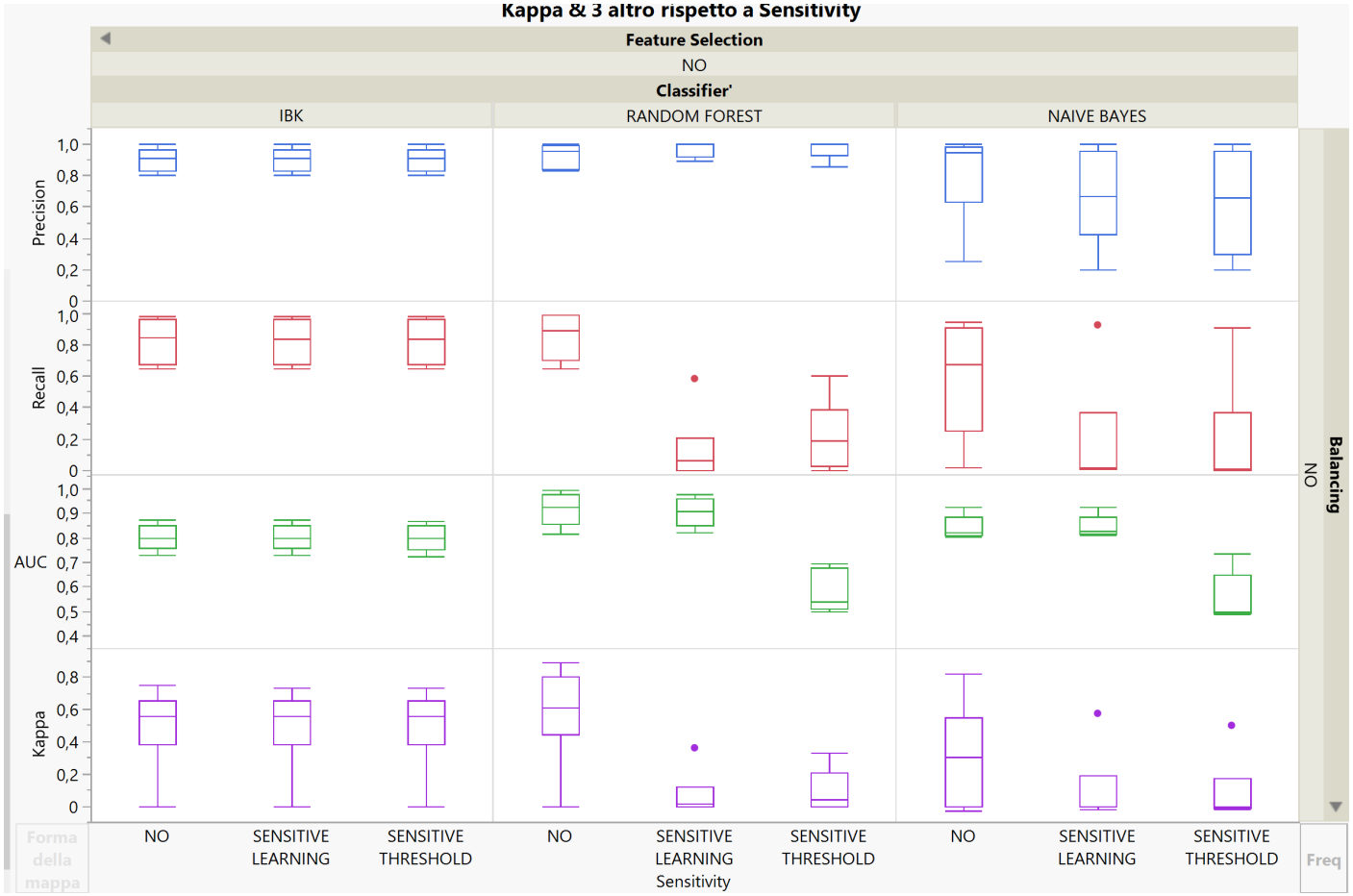
Per i classificatori IBK e RF, l'applicazione della tecnica SMOTE tende a migliorare in media le metriche coinvolte, in particolare rispetto al caso No balancing garantisce prestazioni migliori per AUC e Kappa. Per il classificatore NB, la tecnica di oversampling garantisce una crescita della metrica Kappa, non si registrano tuttavia variazioni significative. In media, si ottiene un miglioramento di circa l'1% nelle performance.

Risultati Bookkeeper con FS + Balancing



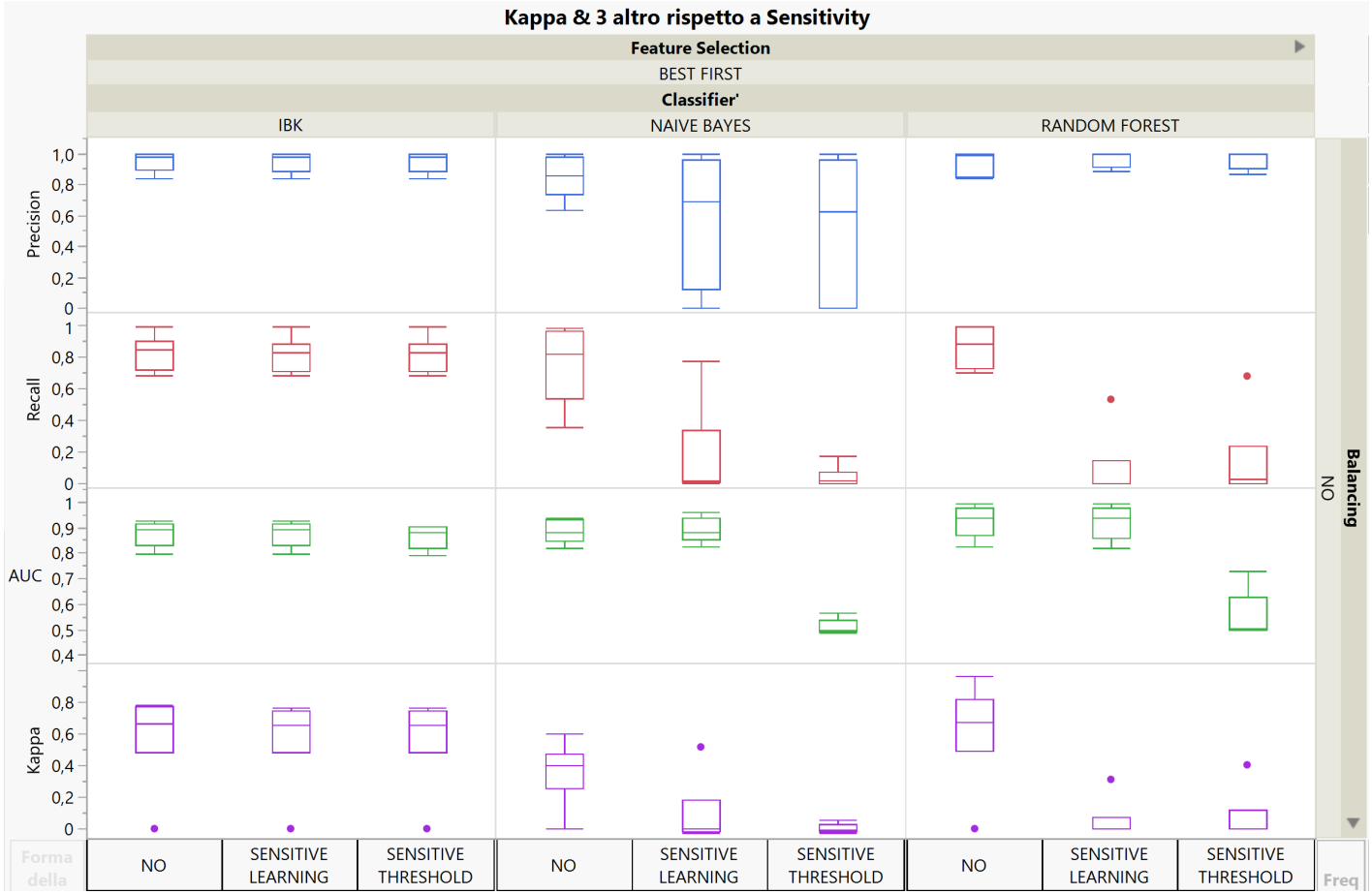
Tale grafo conferma per il classificatore IBK il miglioramento dei risultati con l'applicazione della tecnica SMOTE anche in presenta della FS tuttavia rispetto al caso precedente non si discosta molto dal caso No balancing indice del fatto che con l'applicazione della FS non è strettamente necessaria, per questo classificatore, l'applicazione del balancing riuscendo a raggiungere le stesse prestazioni con la sola applicazione della FS. Per il classificatore NB la tecnica che migliora le prestazioni combinata con la FS è la tecnica di Undersampling con la quale si garantiscono prestazioni migliori rispetto al caso solo balancing (oversampling) per Recall, AUC e Kappa. Per il classificatore RF viene confermata la bontà della tecnica SMOTE combinata con la tecnica di FS. Rispetto al caso solo Balancing si registra infatti un miglioramento per tutte le 4 metriche coinvolte

Risultati Bookkeeper con Cost Sensitive



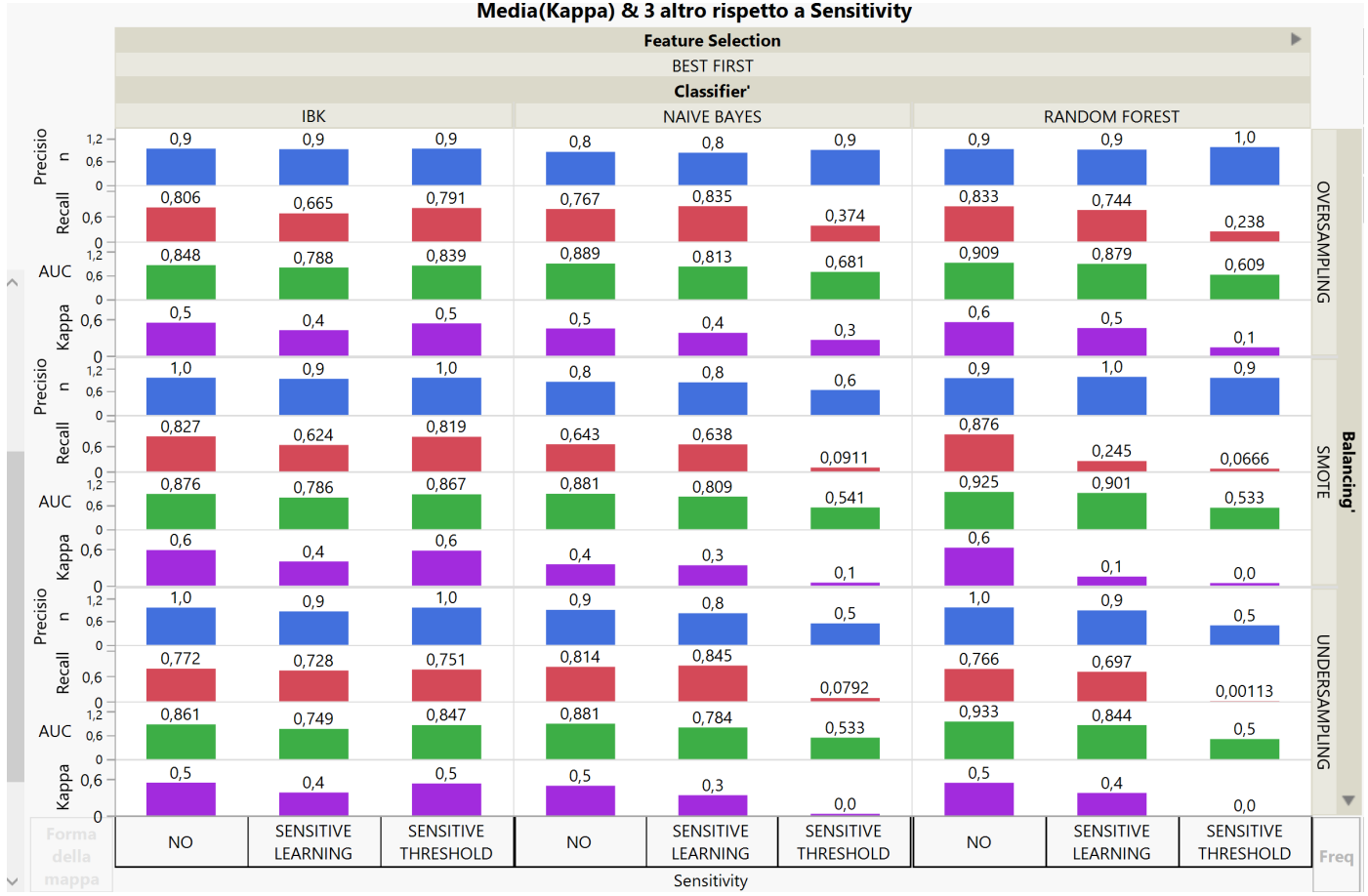
Dal grafico è possibile osservare il chiaro peggioramento delle prestazioni con l'applicazione del Cost Sensitive Classifier per tutti i classificatori testati. In particolare Random Forest e Naive Bayes registrano una significativa diminuzione in Recall e Kappa con l'applicazione di Sensitive Learning e Sensitive Threshold.

Risultati Bookkeeper con Cost Sensitive + FS



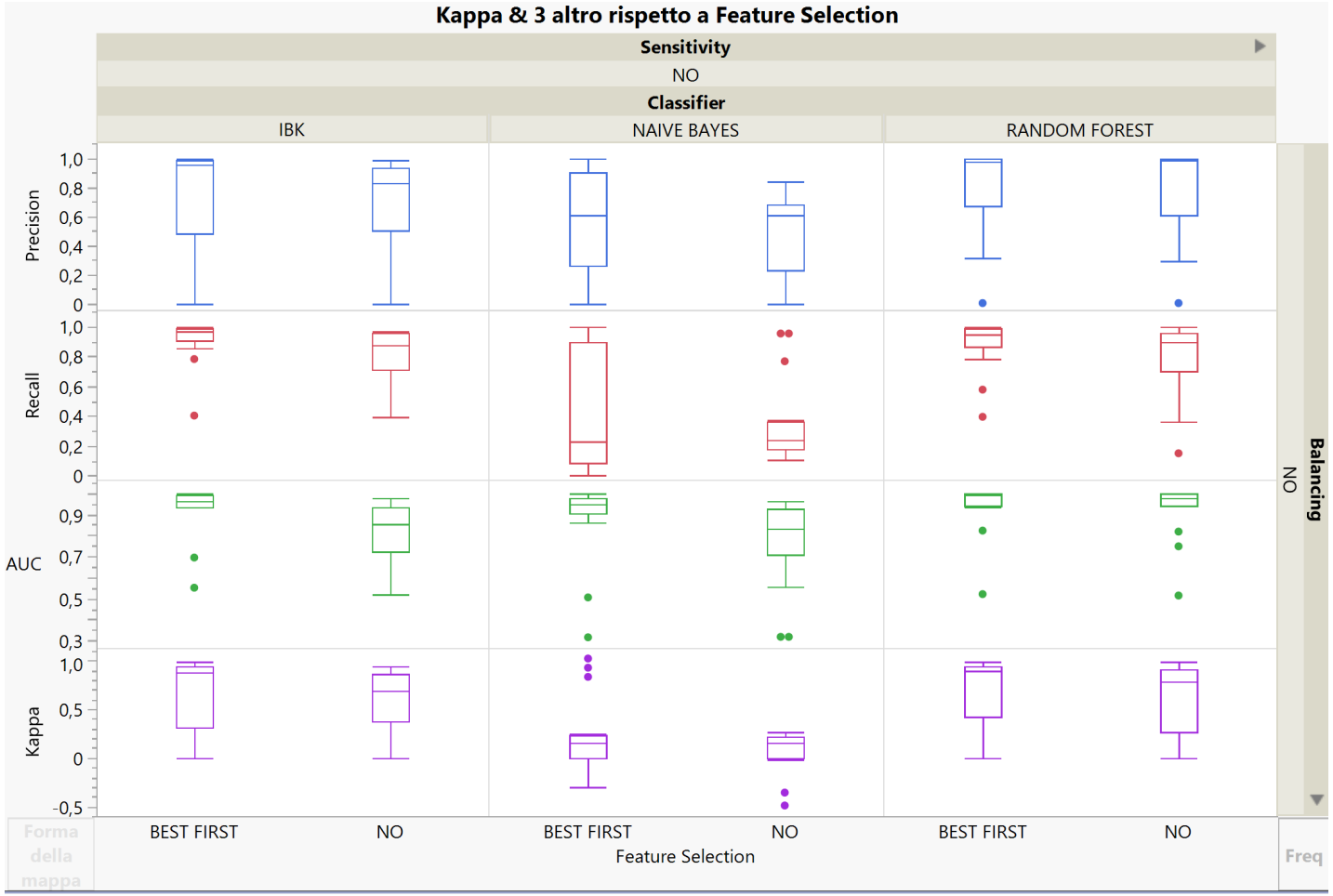
Per i tre classificatori testati le prestazioni non migliorano rispetto al caso precedente pur avendo applicato la FS in combinazione con il Cost Sensitive Classifier. La non applicazione di questa tecnica continua a registrare prestazioni significativamente superiori.

Risultati Bookkeeper con Cost Sensitive + FS + Bal



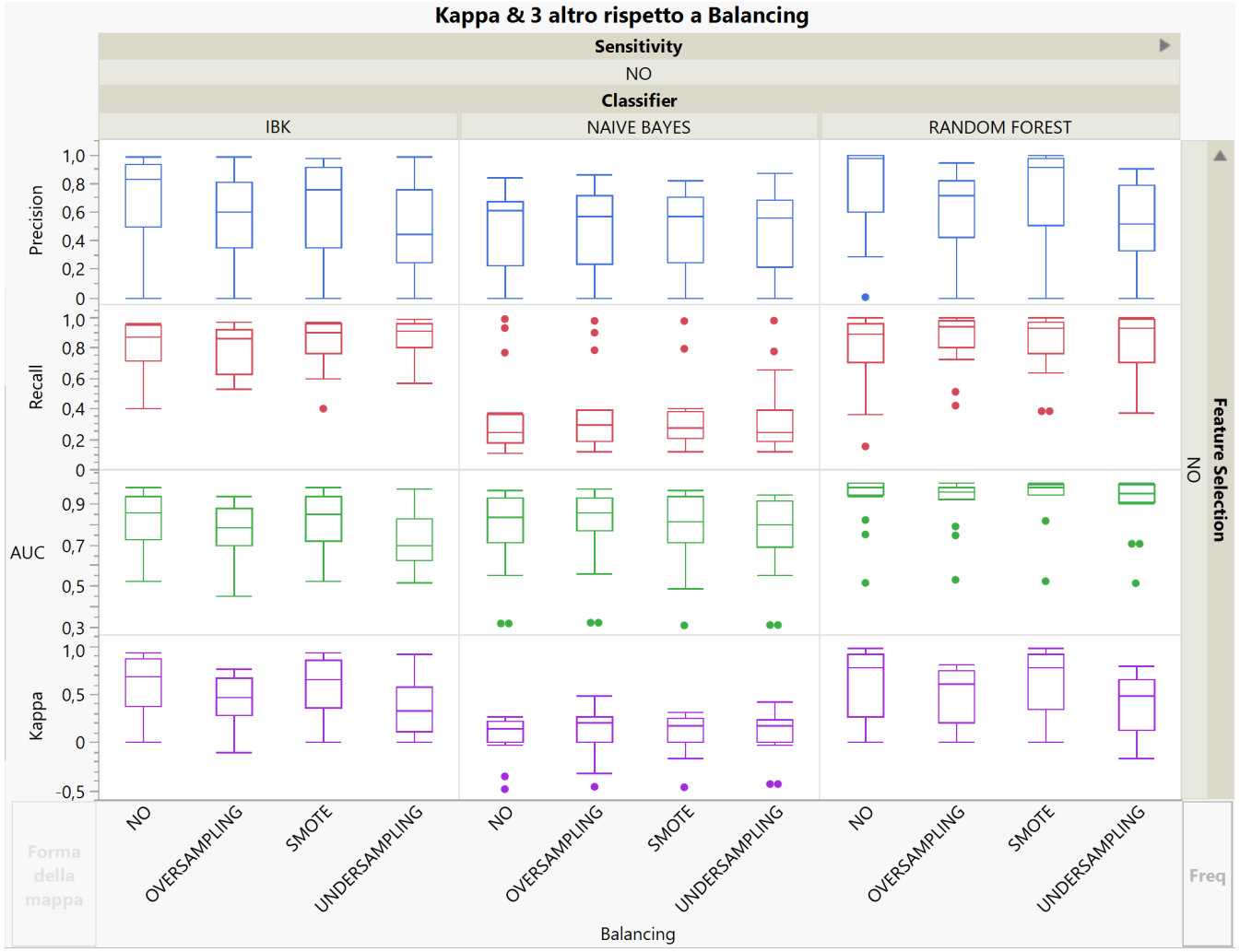
La media delle prestazioni tende a migliorare per i classificatori NB e RF (rispetto al caso precedente senza FS e Balancing) con l'applicazione del Sensitive Learning in combinazione con FS e Balancing mentre le prestazioni rimangono significativamente basse con l'applicazione del sensitive threshold anche in presenza di FS e balancing. Tuttavia prestazioni migliori continuano ad essere raggiunte senza l'applicazione della Cost Sensitive Classifier. Il classificatore IBK è l'unico che gestisce meglio tale tecnica con valori pressoché costanti per il caso No Sensitive e Sensitive Threshold.

Risultati OpenJPA con FS



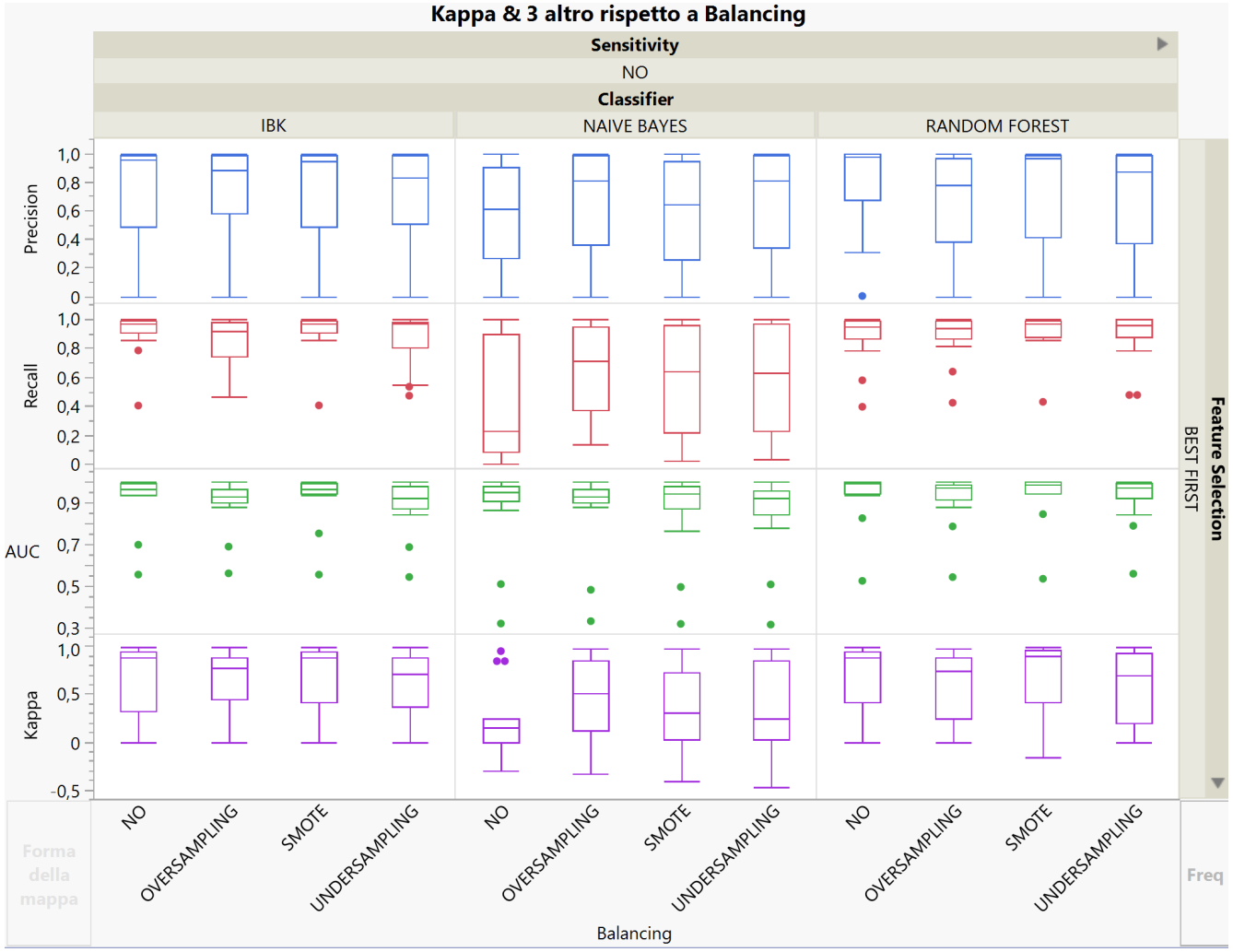
Dal grafico è possibile notare il miglioramento che si ottiene con l'applicazione della Feature Selection per i classificatori IBK e RF. Tutte le metriche infatti registrano prestazioni migliore. Il classificare NB guadagna in AUC mentre Precision, Recall e Kappa rimangono pressoché costanti. A parità di prestazioni si sceglie sempre, ove possibile, l'applicazione della FS.

Risultati OpenJPA con Balancing



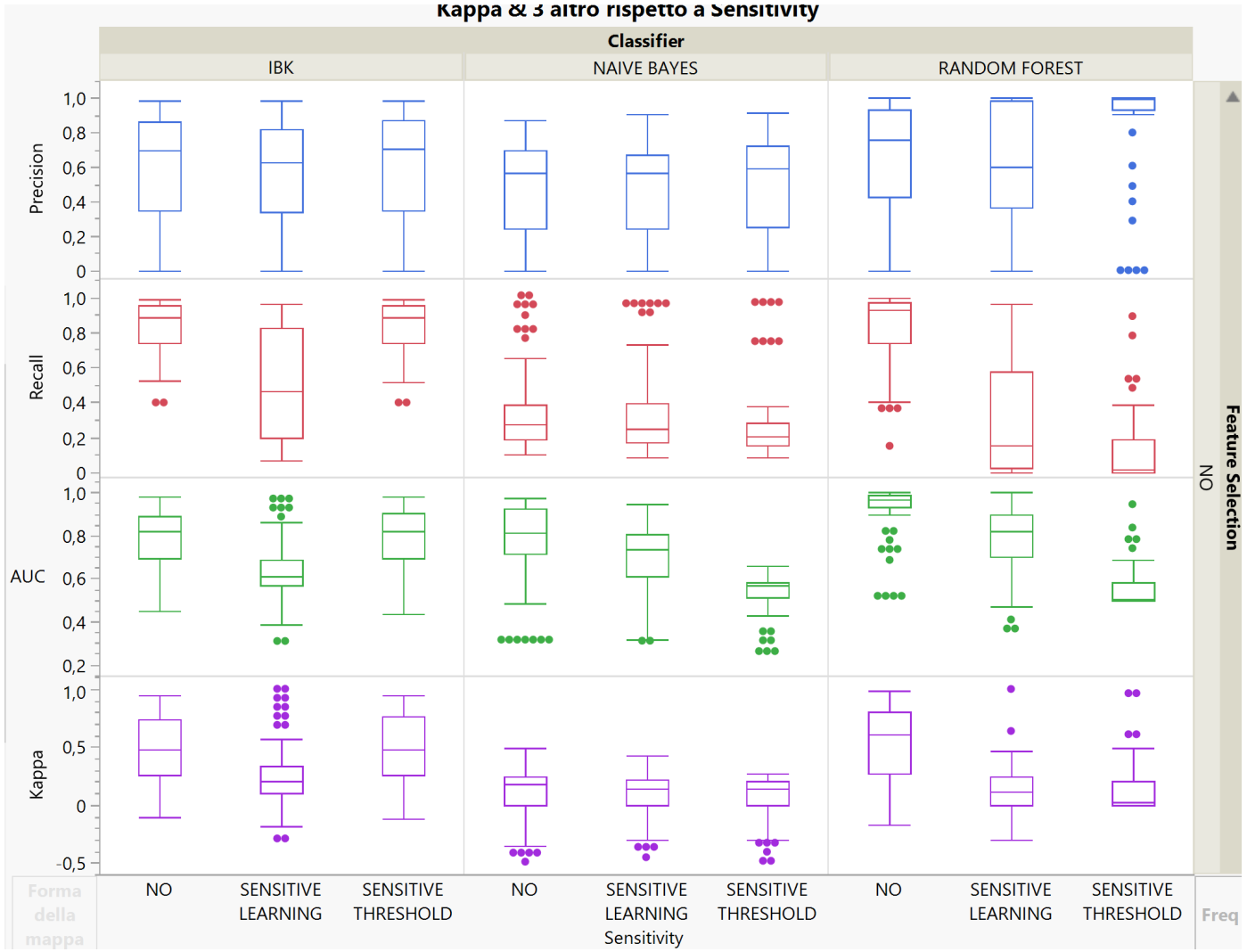
Per il classificatore IBK nessuna tecnica di balancing sembra migliorare in media le prestazioni. La non applicazione di alcuna tecnica fa raggiungere valori maggiori per Kappa, Precision ed AUC. La Recall invece migliora (del 3%) con l'applicazione della tecnica di SMOTE la quale non si discosta molto dal no balancing in termini di AUC e valore Kappa. Per il classificatore NB è possibile fare un discorso analogo, prestazioni in media migliori vengono raggiunge senza alcuna tecnica di balancing, tuttavia la tecnica SMOTE garantisce un miglioramento (1%) per Kappa e Recall la quale rimane comunque molto bassa (<0.5 peggiore quindi di un classificatore 0-R). Discorso analogo per il classificatore Random Forest il quale registra un comportamento in media migliore senza alcuna tecnica di Balancing non discostandosi molto dalle prestazioni ottenute con la tecnica SMOTE con Kappa ed AUC pressoché costanti.

Risultati OpenJPA con FS + Balancing



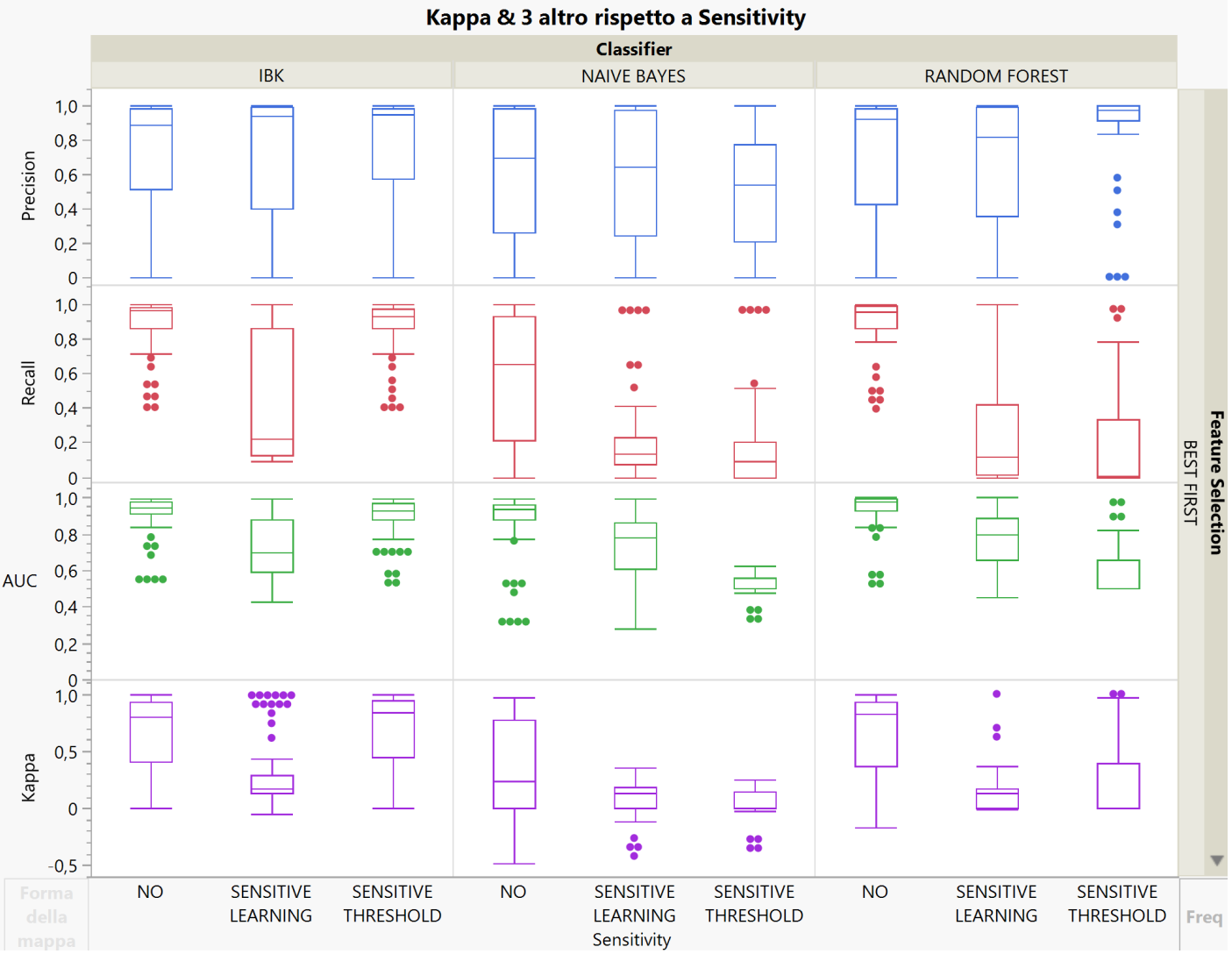
Per il classificatore IBK nonostante l'applicazione del balancing e della FS, continuano a valere le precedenti assunzioni. Il classificatore registra infatti prestazioni migliori con la sola applicazione della FS, con la tecnica SMOTE che continua a non discostarsi molto dal caso No Balancing. Il classificare NB migliora notevolmente le prestazioni, soprattutto in termini di Recall, con l'applicazione di tecniche di Balancing combinate con la FS. In particolare prestazioni maggiori vengono raggiunte con le tecniche Oversampling + Best First. La Recall infatti, precedentemente inferiore al valore 0.5, supera soglia 0.7 con un netto miglioramento anche in termini di Kappa. Il Classificatore Random Forest migliora le prestazioni con l'applicazione della tecnica SMOTE in combinazione con la Feature Selection migliorando significativamente Kappa (+10%).

Risultati OpenJPA con Cost Sensitive



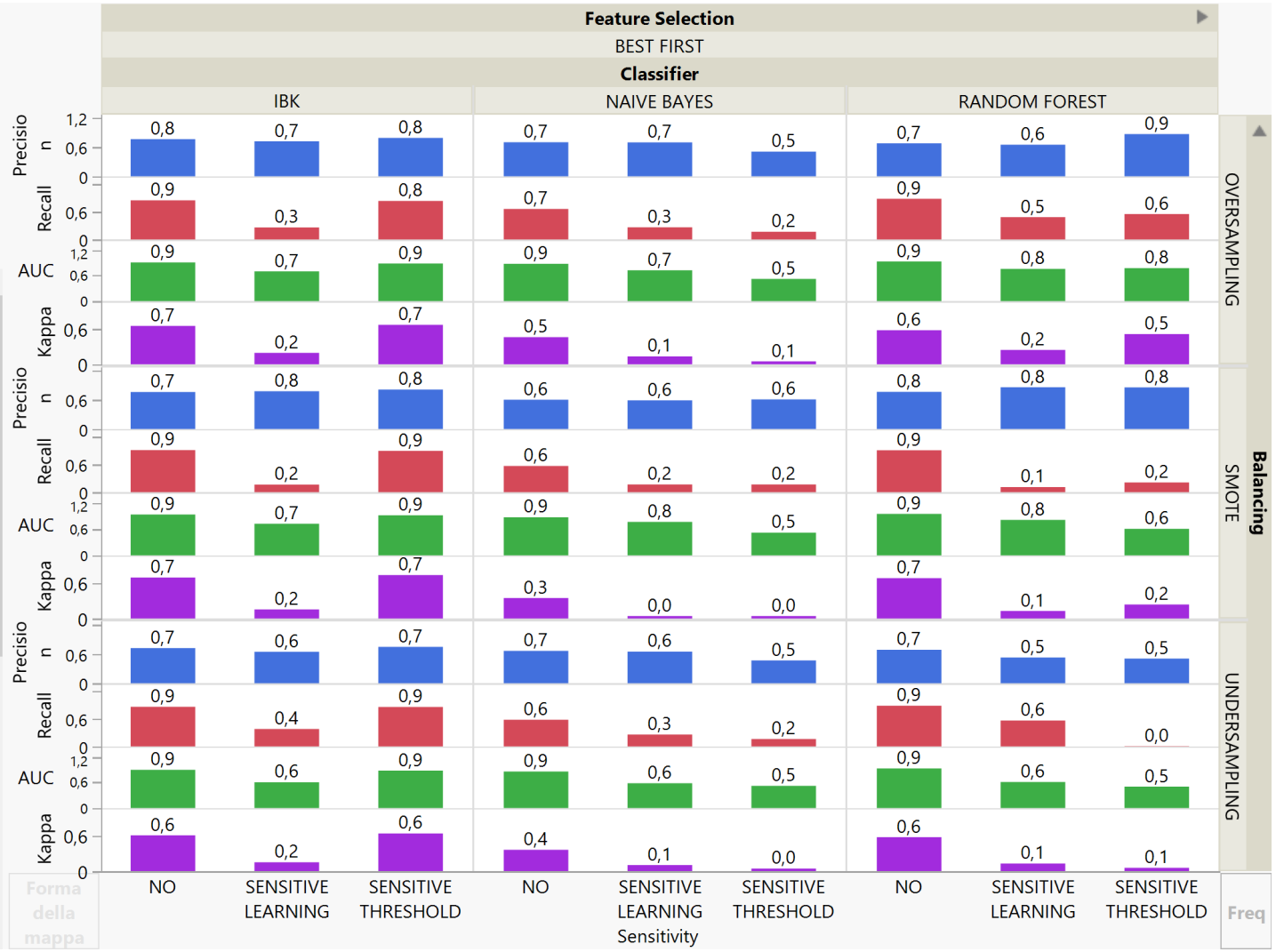
Per il classificatore IBK non si registrano miglioramenti con la sola applicazione del Cost Sensitive Classifier. Le metriche rimangono costanti nei casi No cost Sensitive e sensitive Threshold mentre peggiorano notevolmente, soprattutto Kappa, AUC e Recall con l'applicazione del Sensitive Learning. Il classificatore NB mantiene costanti Precision, Recall e Kappa ma AUC degrada notevolmente in particolare con il sensitive threshold. Per il classificatore RF tutte le metriche peggiorano con l'applicazione del Cost Sensitive Classifier nello specifico la Recall raggiunge valori molto bassi, inferiori prestazionalmente a quelli di un classificatore 0-R.

Risultati OpenJPA con Cost Sensitive + FS



Come per il Dataset Bookkeeper la FS non migliora le prestazioni se combinata con il Cost Sensitive Classifier. Anche in questo caso infatti prestazioni migliori vengono raggiunte senza alcuna applicazione di CS. IBK anche in questo caso è il classificatore che meglio reagisce all'applicazione di tale tecnica mantenendo le metriche pressoché costanti con il Sensitive Threshold rispetto al caso No Sensitive. NB e RF registrano prestazioni molto basse, inferiori in molti casi a quelle di un classificatore O-R.

Risultati OpenJPA con Cost Sensitive + FS + Bal



Applicando tecniche di FS, Balancing e CS al classificatore IBK vengono raggiunte prestazioni migliori rispetto al caso precedentemente descritto con l'applicazione della Best First in combinazione con Balancing SMOTE e Sensitive Threshold le quali garantiscono un miglioramento in termini di Precision rispetto al caso No Sensitive ma prestazioni analoghe rispetto al caso solo FS. Per i Classificatori RF e NB continuano a valere le considerazioni precedentemente fatte, l'applicazione del CS anche in combinazione con altre tecniche di Balancing e FS causa un netto peggioramento delle prestazioni comportando in molti casi prestazioni peggiori di un classificatore che sceglie randomicamente a quale classe assegnare l'istanza.

Conclusioni

Sulla base delle assunzioni precedentemente descritte (e non potendo definire una configurazione 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	NO
BOOKKEEPER(RF)	BEST FIRST	SMOTE	NO
BOOKKEEPER(NB)	BEST FIRST	UNDERSAMPLING	NO
OPENJPA(IBK)	BEST FIRST	NO	NO
OPENJPA(RF)	BEST FIRST	SMOTE	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: IBK (con le configurazioni descritte) garantisce Precision massima (variazione 1% rispetto al classificatore RF) mentre Recall ed AUC risultano maggiori con RF (variazione minima), Kappa resta invece costante. Si ottengono quindi con entrambi i classificatori buone prestazioni. Discorso analogo per il Dataset OpenJPA è possibile infatti scartare il classificatore NB, con RF e IBK i quali garantiscono prestazioni pressoché analoghe utilizzando le configurazioni precedentemente descritte.
- 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