

MACHINE LEARNING FOR SOFTWARE ENGINEERING

ANDREA DE FILIPPIS - MATRICOLA: 0333703

CONTENUTO

01

CONTESTO

02

SCOPO E OBIETTIVI

03

SCELTE ED ASSUNZIONI

04

METRICHE CONSIDERATE

05

METODI DI VALUTAZIONE

06

RISULTATI E VALUTAZIONE

CONTESTO

Al giorno d'oggi, ogni progetto nell'ambito dell'Ingegneria del Software richiede attività di software testing. Tuttavia, ciò risulta sempre più dispendioso a livello di risorse.



È necessario individuare un sottoinsieme di classi da testare. Come stabilire quali classi hanno più probabilità di avere malfunzionamenti?

CONTESTO



Usando il Machine learning!

Il Machine learning ci permette di capire quali sono le principali caratteristiche che rendono le classi buggy analizzando la storia passata.

In questo modo, con l'uso di particolari algoritmi, detti classificatori, è possibile predire quali classi contengono difetti e concentrarsi soltanto su di loro.

SCOPO E OBIETTIVI

Scopo

Il nostro scopo era quello di analizzare due progetti di Apache Software Foundation, nello specifico BookKeeper e Syncopé, e misurare l'accuratezza di tre classificatori usando diverse tecniche per migliorare la qualità delle predizioni. Infine, stabilire come e quanto le tecniche utilizzate hanno impattato sulle predizioni.

Classificatori

- Random Forest
- Naive Bayes
- IBk

Tecniche

- Feature selection
- Sampling
- Cost sensitive

SCOPO E OBIETTIVI

Obiettivo n° 1

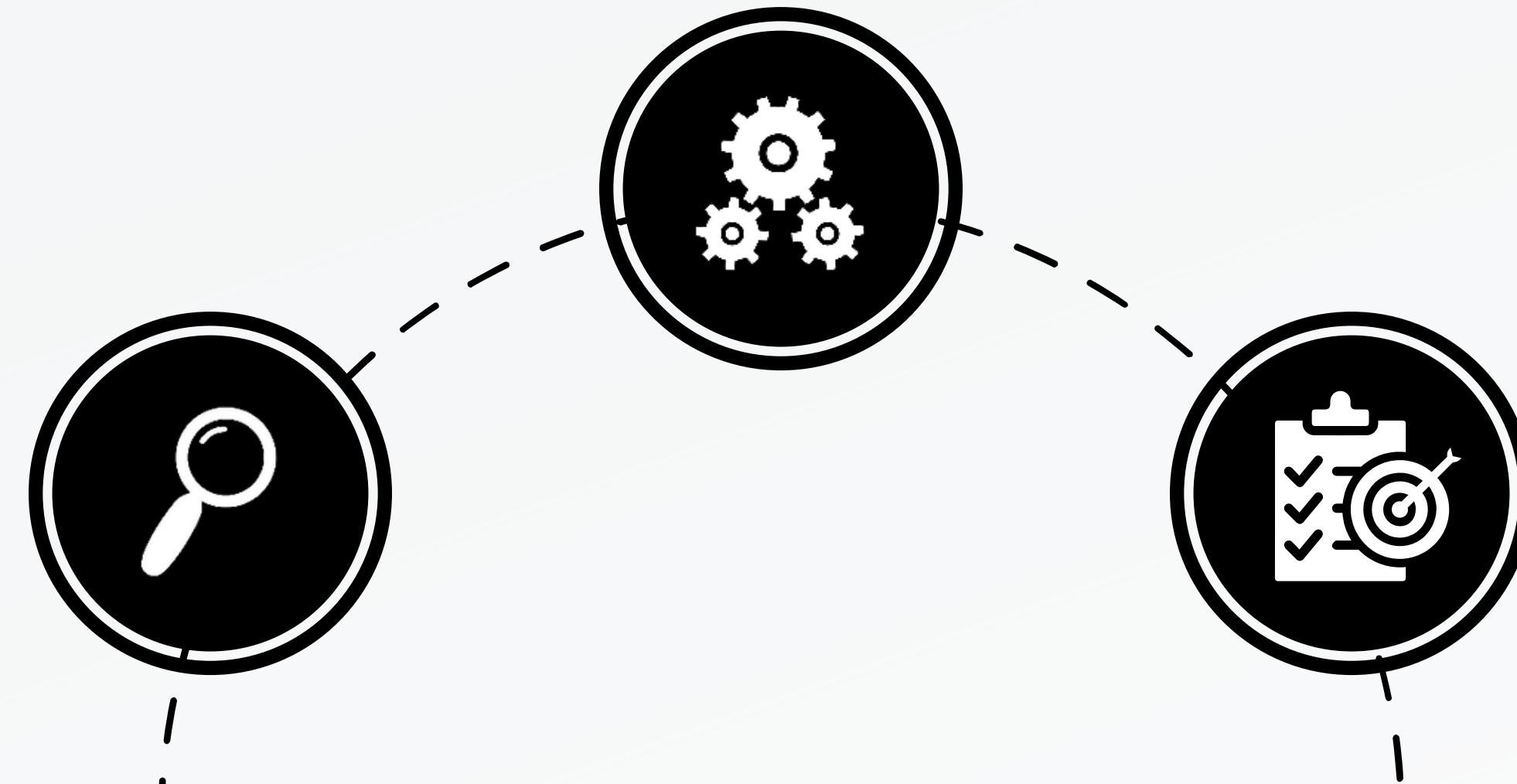
Raccolta
d'informazioni
sulla storia
passata.

Obiettivo n° 2

Costruzione del dataset e
scelta delle metriche da
considerare.

Obiettivo n° 3

Analisi e
valutazione dei
classificatori.



SCELTE ED ASSUNZIONI

Una delle prime considerazioni da fare riguarda come stabilire per quante versioni le classi sono state buggy: una classe contenente un bug è considerata buggy nelle release che vanno dalla injected version (inclusa) fino alla fixed version (esclusa).



SCELTE ED ASSUNZIONI

Tutti i bug riscontrati dai due progetti sono registrati su Jira, un software specializzato nella gestione di ticket. In Jira è sempre possibile accedere ad OV e FV, ma ciò non vale per l'IV. Infatti, l'istante di rilevazione e di eliminazione del bug sono sempre noti, ma il momento dell'introduzione no. Cosa fare in tali casi?

Proportion

Proportion è una tecnica utilizzata per stimare l'IV di un bug. Infatti, è stato notato che il tempo che intercorre tra IV e OV è spesso proporzionale a quello tra OV e FV.

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

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

In particolare, in questo progetto è stato usato incremental proportion, ovvero una variante che calcola p come la media delle p di tutti i bug precedenti a quello considerato.

SCELTE ED ASSUNZIONI

Cold start

Nel caso in cui i bug con IV disponibile fossero meno di 5, si procede con il cold start, che consiste nell'utilizzare i bug di altri progetti di Apache per il calcolo di proportion.

Snoring

Un problema da non sottovalutare: un bug, dopo essere stato introdotto, può rimanere "dormiente" per molto tempo. Ciò comporta che le ultime release sono piene di bug dormienti e che, perciò, i relativi dataset sulle classi buggy non sono affidabili.

Soluzione: buttare la seconda metà delle release. In questo modo possiamo avere un alto grado di affidabilità sui dati.

METRICHE CONSIDERATE

Metrica	Descrizione
Size	Numero di linee di codice della classe.
LOC_added	Numero di linee di codice aggiunte alla classe durante le revisioni di questa release.
AVG_LOC_added	Numero medio di linee di codice aggiunte alla classe.
LOC_deleted	Numero di linee di codice rimosse alla classe durante le revisioni di questa release.
AVG_LOC_deleted	Numero medio di linee di codice rimosse alla classe.
Churn	Somma dei moduli delle differenze tra le LOC aggiunte e quelle rimosse in una classe nella release considerata.
AVG_CHURN	Valore medio del churn.
FIXED_DEFECTS	Numero di difetti risolti in una classe durante quella release.
NUMBER_OF_COMMITS	Numero di commit all'interno della release.
NUMBER_OF_AUTHORS	Numero di sviluppatori che hanno toccato quella classe.

METODI DI VALUTAZIONE

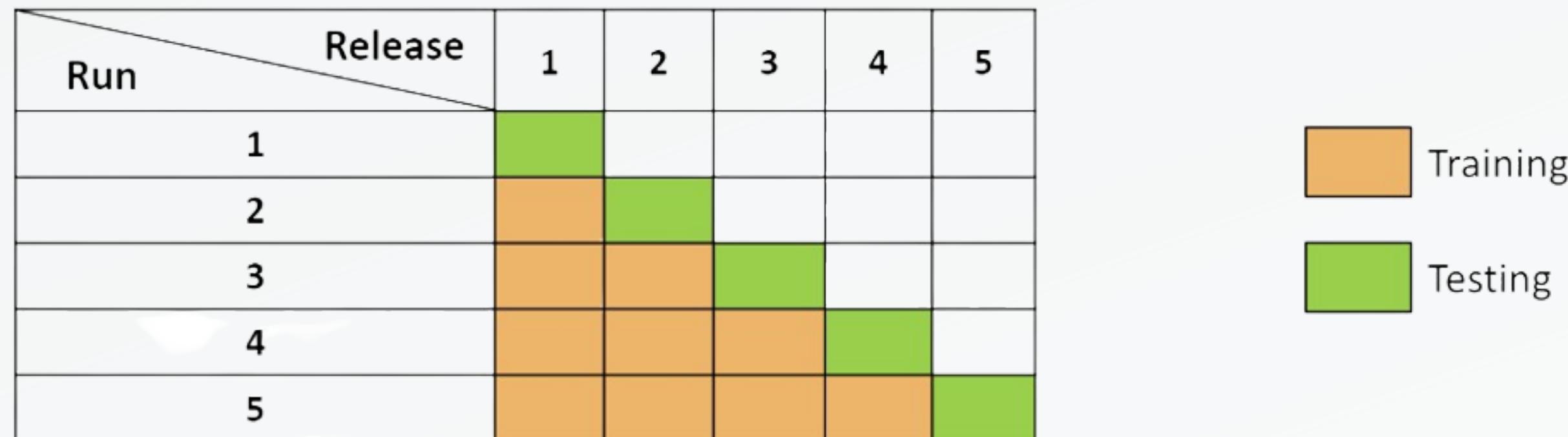
Durante l'utilizzo di un classificatore, il dataset viene diviso in **training set** e **testing set**. Il primo rappresenta il set con cui viene allenato il classificatore, mentre il secondo rappresenta il set su cui deve fare le sue predizioni e verificare i risultati.

Training set: deve essere fedele alla situazione che stiamo valutando, quindi non può usare dati futuri ed è affetto da snoring.

Testing set: deve essere fedele alla situazione reale, quindi deve usare anche dati futuri e non è affetto da snoring.

METODI DI VALUTAZIONE

Per la valutazione dei classificatori è stata usata la tecnica basata su time series, **Walk Forward**. La tecnica si basa sul concetto per cui è insensato usare dati futuri per predire quelli attuali, perché non si avrebbe una valutazione corretta dei classificatori.



RISULTATI

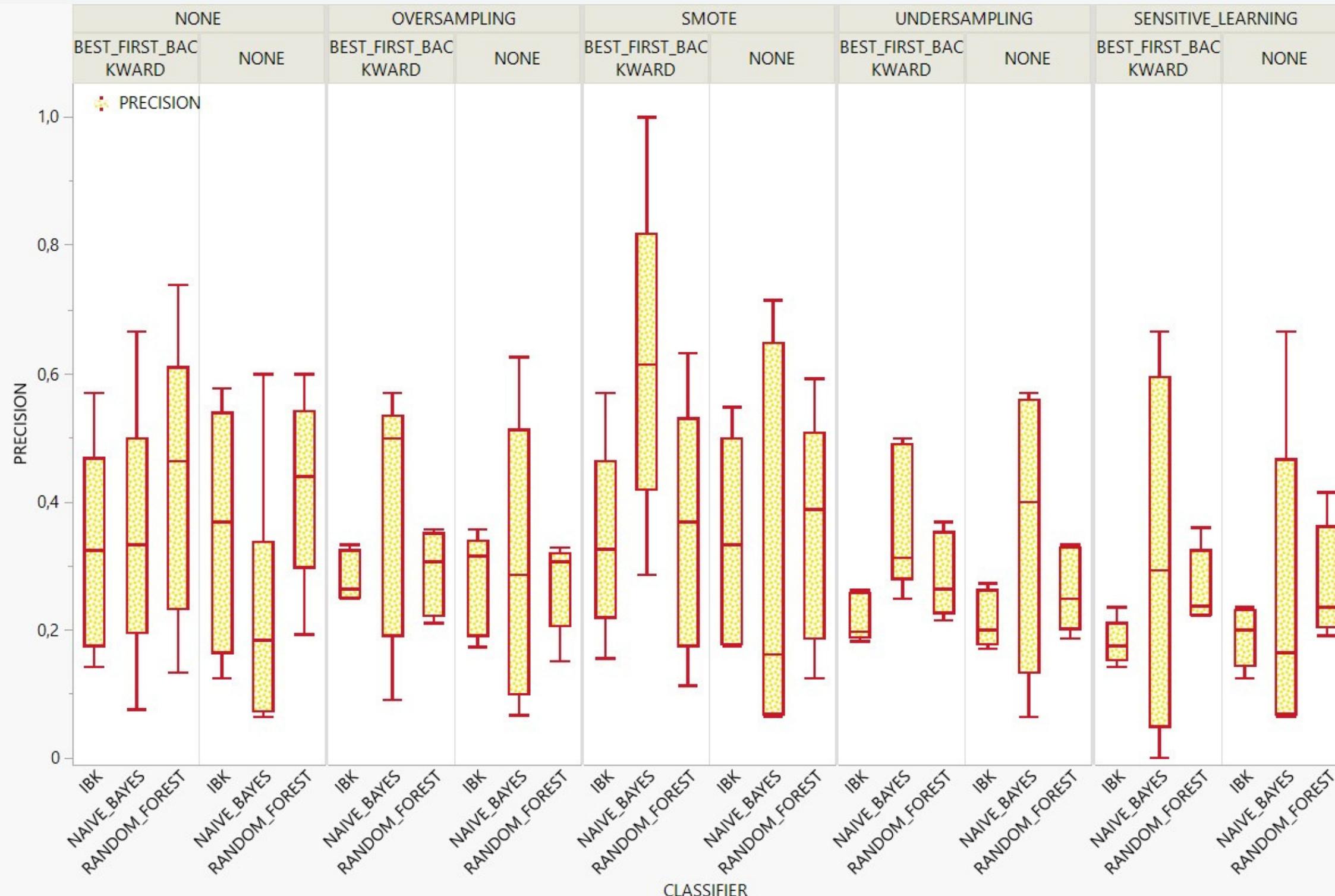
I seguenti risultati paragonano i valori **precision**, **recall**, **KAPPA** e **AUC** risultanti dai classificatori usando diverse tecniche. In questo modo, le differenze vengono rese più evidenti, soprattutto quelle relative alle tecniche usate.

Ricordiamo che i classificatori usati sono **Random Forest**, **Naive Bayes** e **IBk**, mentre per ognuna delle tecniche citate in precedenza sono state prese alcune tipologie:

- **Feature selection:** Best first backward
- **Sampling:** undersampling, oversampling e SMOTE
- **Cost sensitive:** Sensitive learning

RISULTATI - BOOKKEEPER

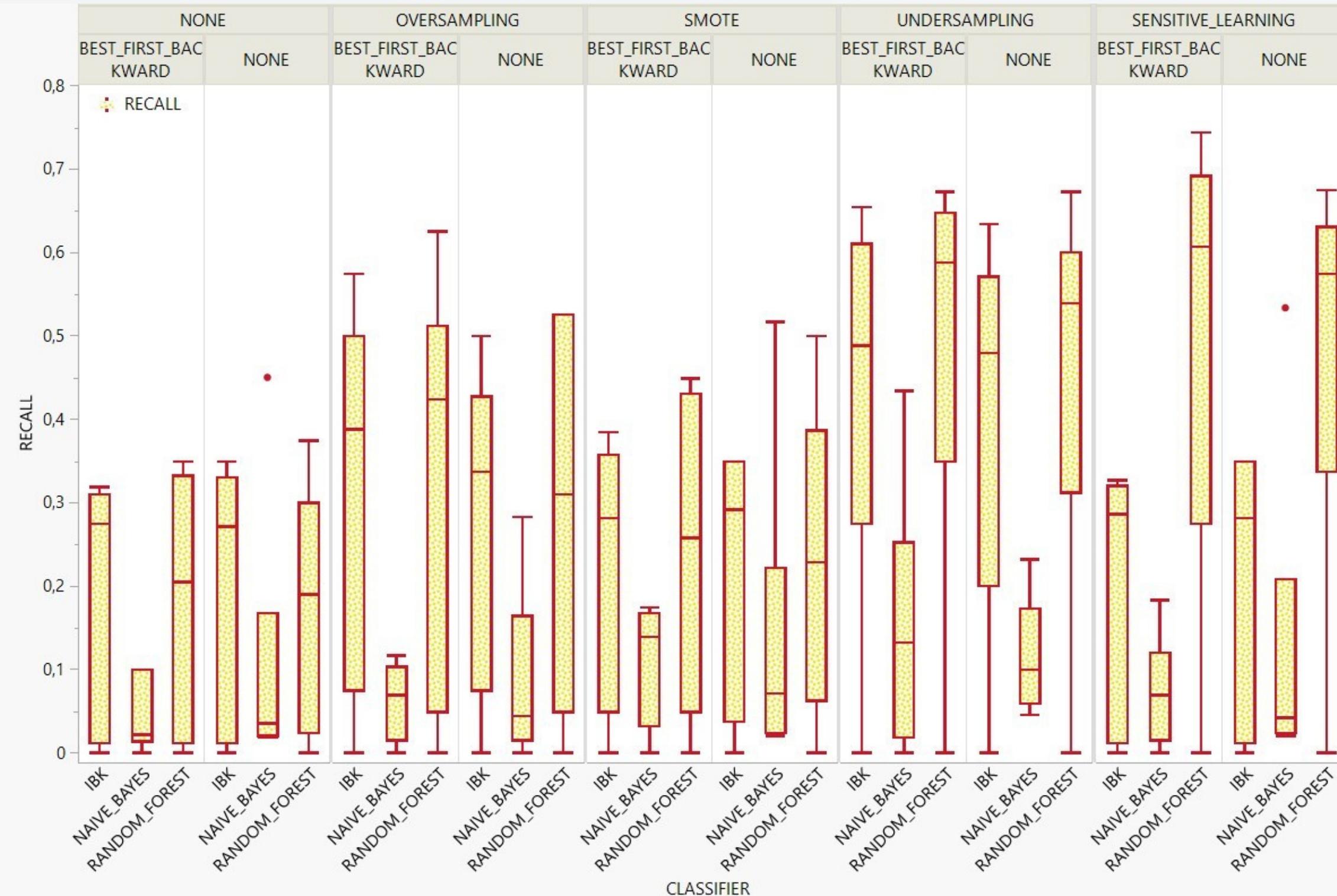
PRECISION



I risultati riguardanti la precision mettono in mostra che le tecniche di sampling utilizzate abbassano il valore finale. Questo accade perché il sampling ha l'obiettivo di aumentare in modo fittizio le istanze considerate buggy. Lo stesso vale per il sensitive learning. Le uniche eccezioni sembrano rappresentate da SMOTE, che aumenta i valori di precision, e Naive Bayes, che risulta spesso il migliore.

RISULTATI - BOOKKEEPER

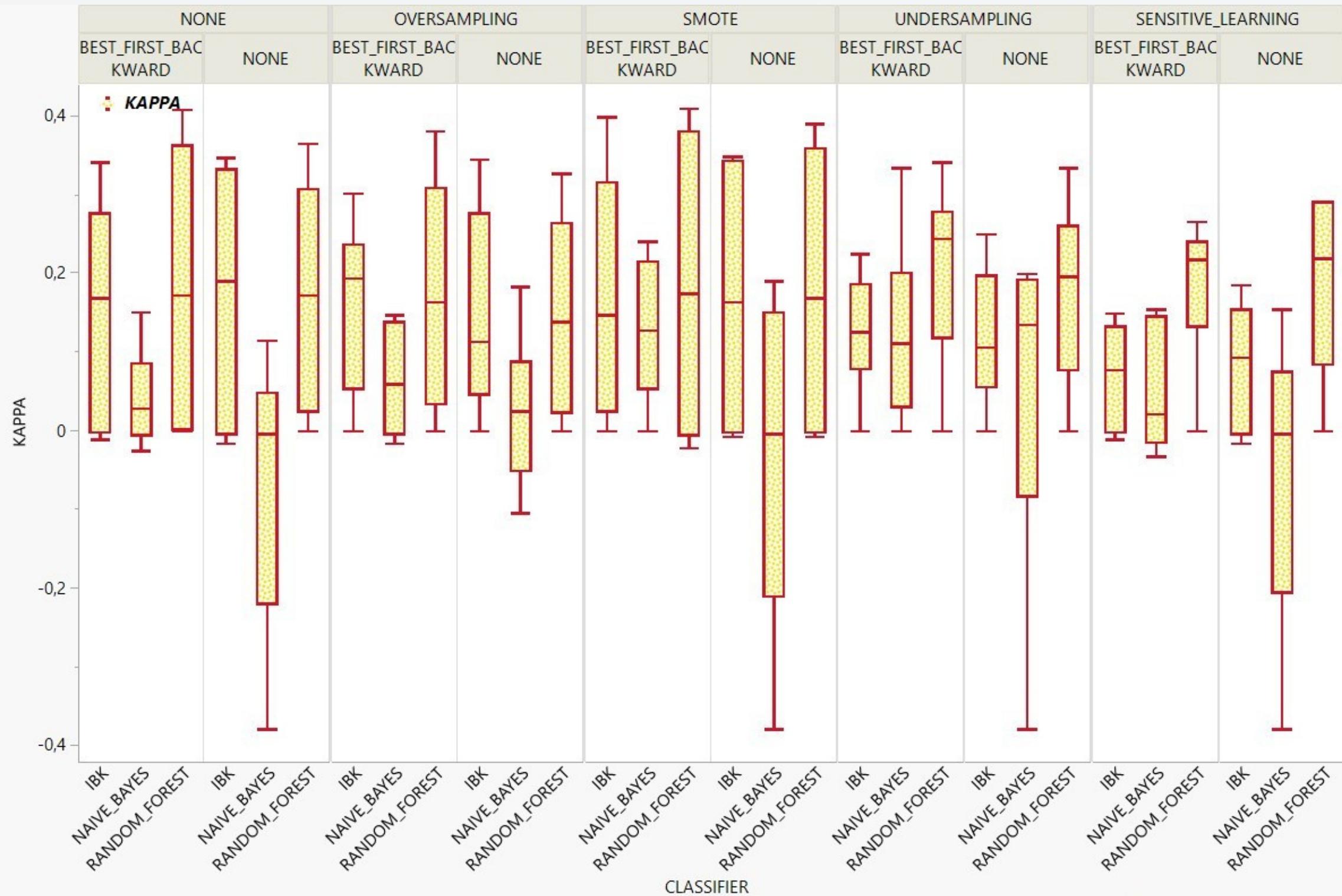
RECALL



Contrariamente a prima, sulla recall sono ben evidenti i miglioramenti fatti dalle tecniche di sampling e cost sensitive. In questo caso SMOTE è tra i peggiori e questo spiega la precision così alta. Mediamente la tecnica migliore sembrerebbe l'Undersampling, ma anche il Sensitive learning con Random Forest ha dato ottimi risultati.

RISULTATI - BOOKKEEPER

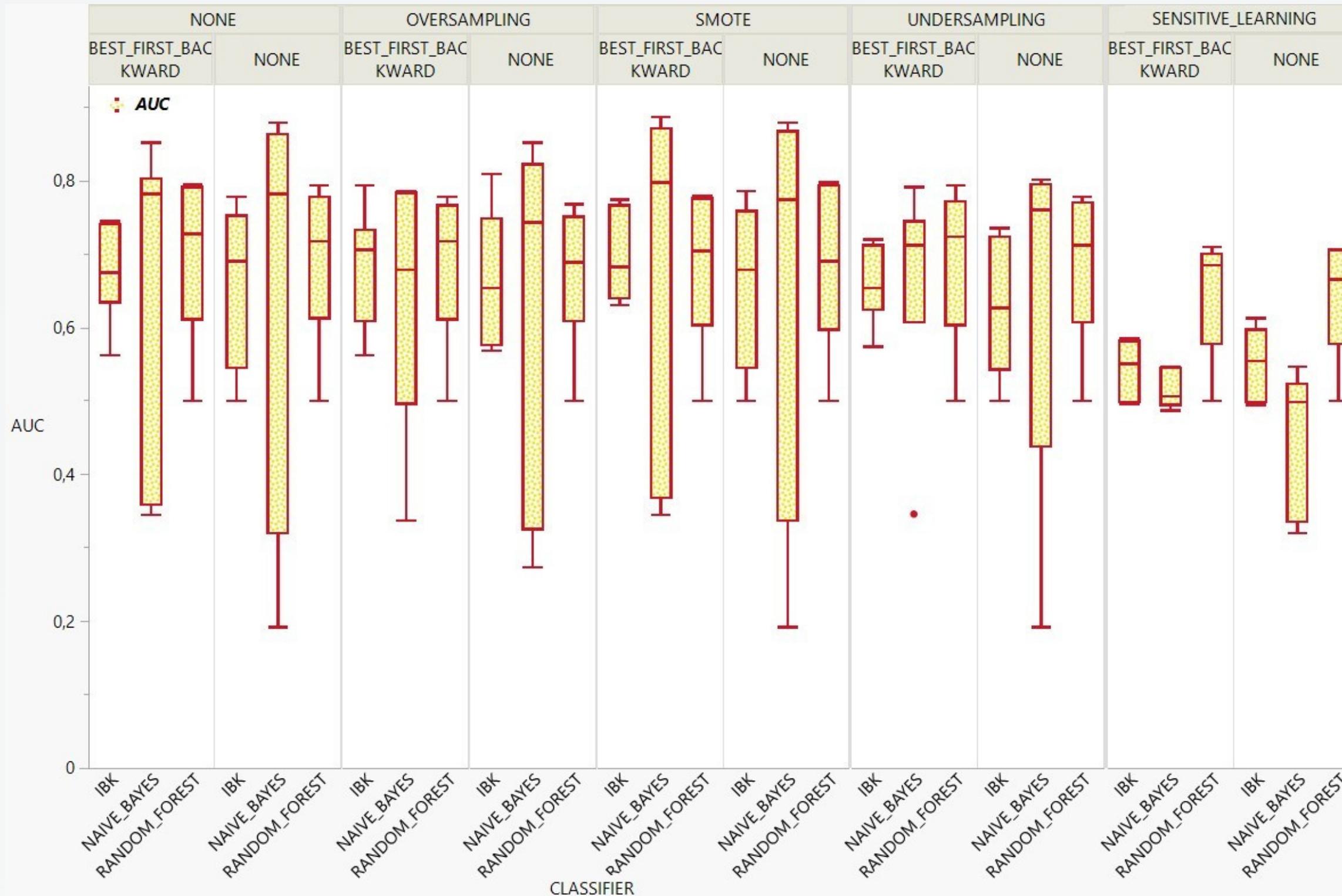
KAPPA



Riguardo ai valori della KAPPA, Naive Bayes si dimostra, come per la recall, il peggiore. Dall'altro lato abbiamo, invece, Random Forest che risulta quasi sempre il migliore. Infine, per le tecniche SMOTE e il non utilizzo di nessun sampling sembrerebbero le migliori scelte.

RISULTATI - BOOKKEEPER

AUC



Infine, l'AUC non mostra un vincitore, ma è evidente che il sensitive learning presenta i peggiori risultati. Il classificatore con i risultati più alti è sicuramente Naive Bayes, ma presenta anche valori molto contrastanti. Infatti, come si può vedere dal grafico, i suoi valori sono molto instabili anche se la sua mediana rimane piuttosto alta.

VALUTAZIONE DEI RISULTATI

Nelle varie iterazioni le metriche più scelte dalla feature selection sono state il Churn e il numero di commit. Effettivamente, una classe toccata tanto o spesso ha più probabilità di avere dei difetti.

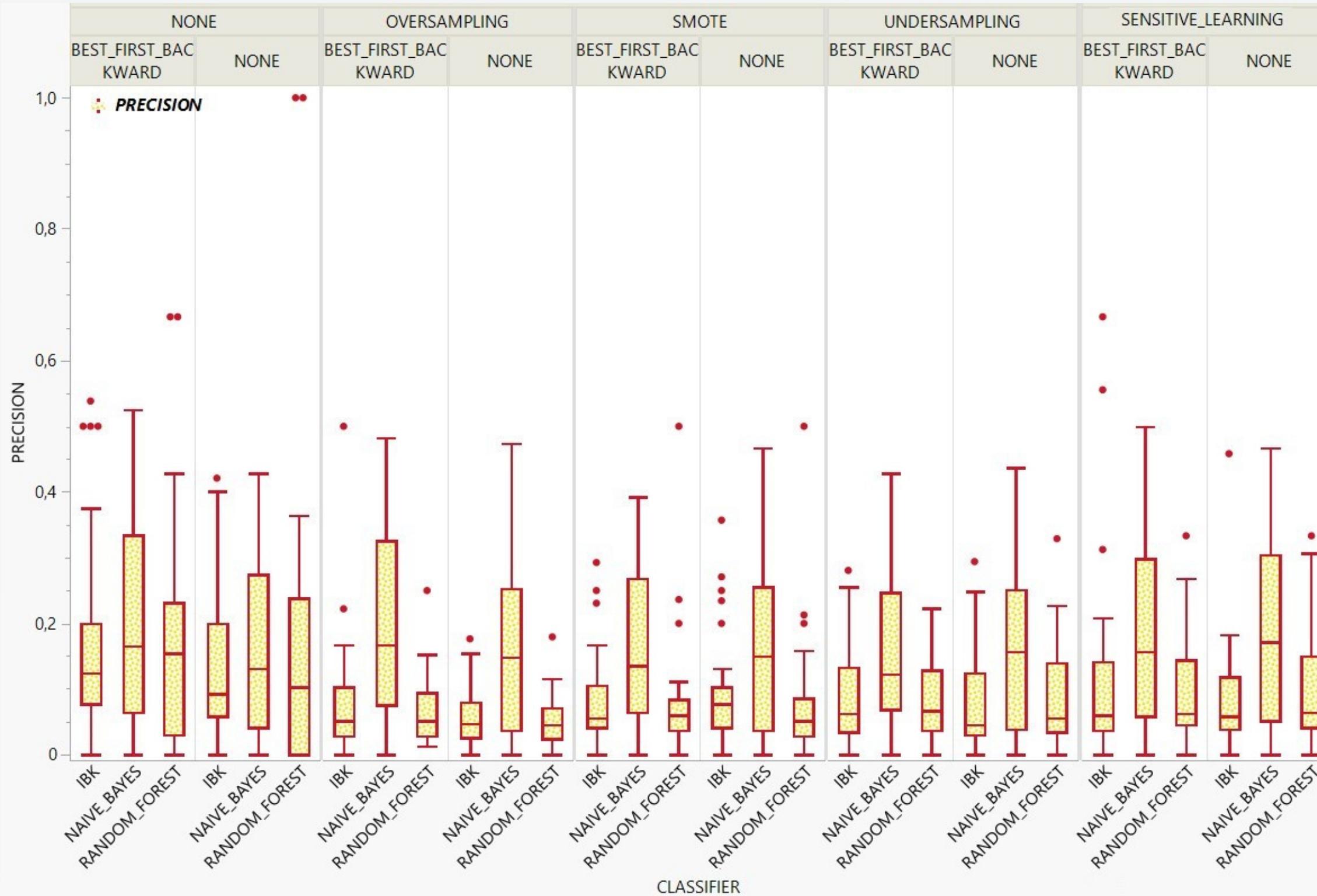
Come era prevedibile, non c'è un classificatore dominante. Naive Bayes ha sì ottimi valori di precision, ma anche pessimi valori di recall. Inoltre, presenta la peggiore KAPPA tra tutti i classificatori.

La tecnica migliore sembra essere il Sensitive learning che riesce a non abbassare troppo la precision e a raggiungere le migliori recall quando usato insieme a Random Forest.

Concludendo, dovendo scegliere un classificatore e una tecnica, la coppia migliore sembra essere formata da Random Forest e Sensitive learning.

RISULTATI - SCOPE

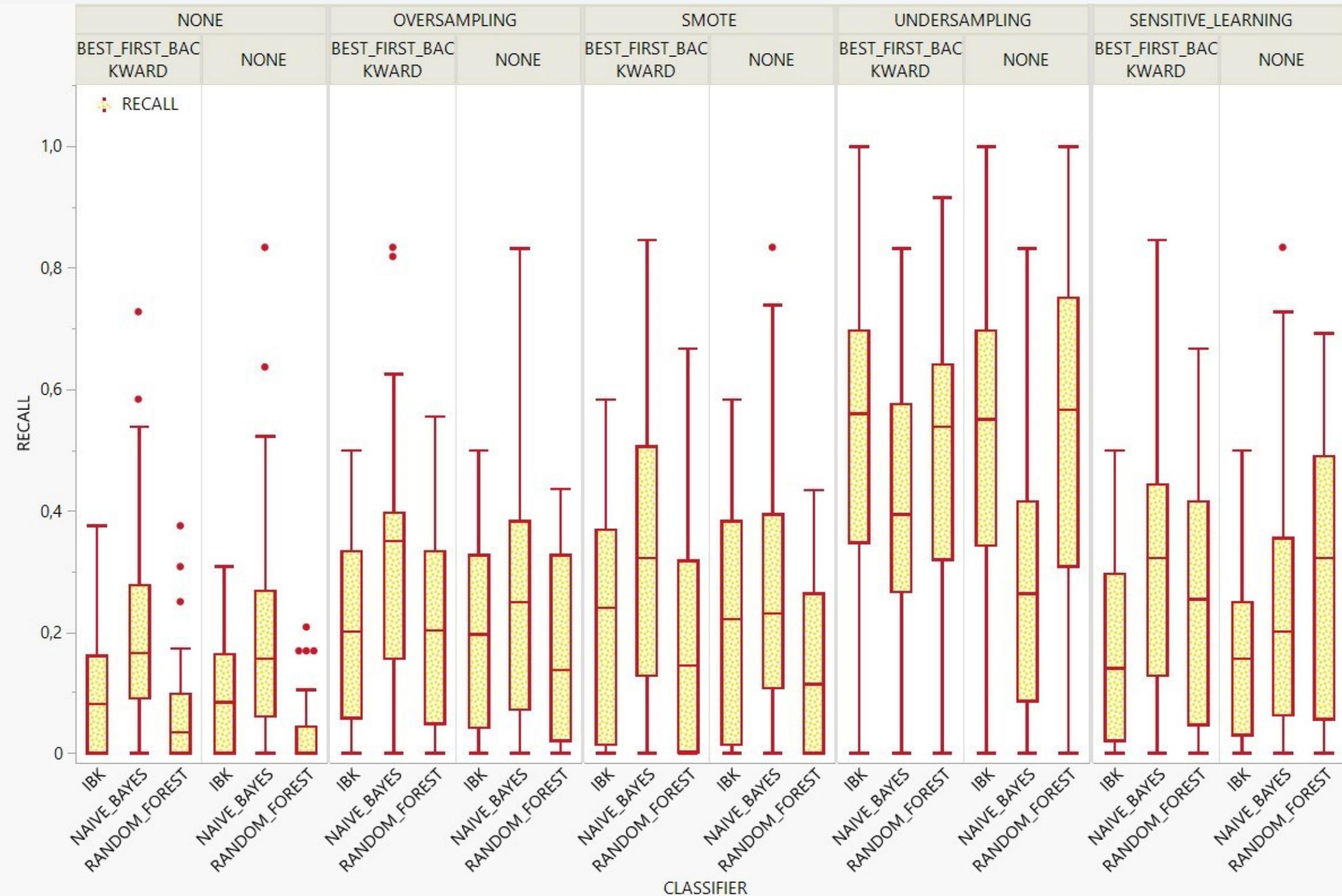
PRECISION



Innanzitutto, è evidente che, rispetto a BookKeeper, in Scope sono presenti molti più valori outliers. Nessuna tecnica si distingue particolarmente; infatti, come prima, tutte le tecniche di sampling e cost sensitive diminuiscono la precision, anche se con un impatto minore.

RISULTATI - SYNCOPE

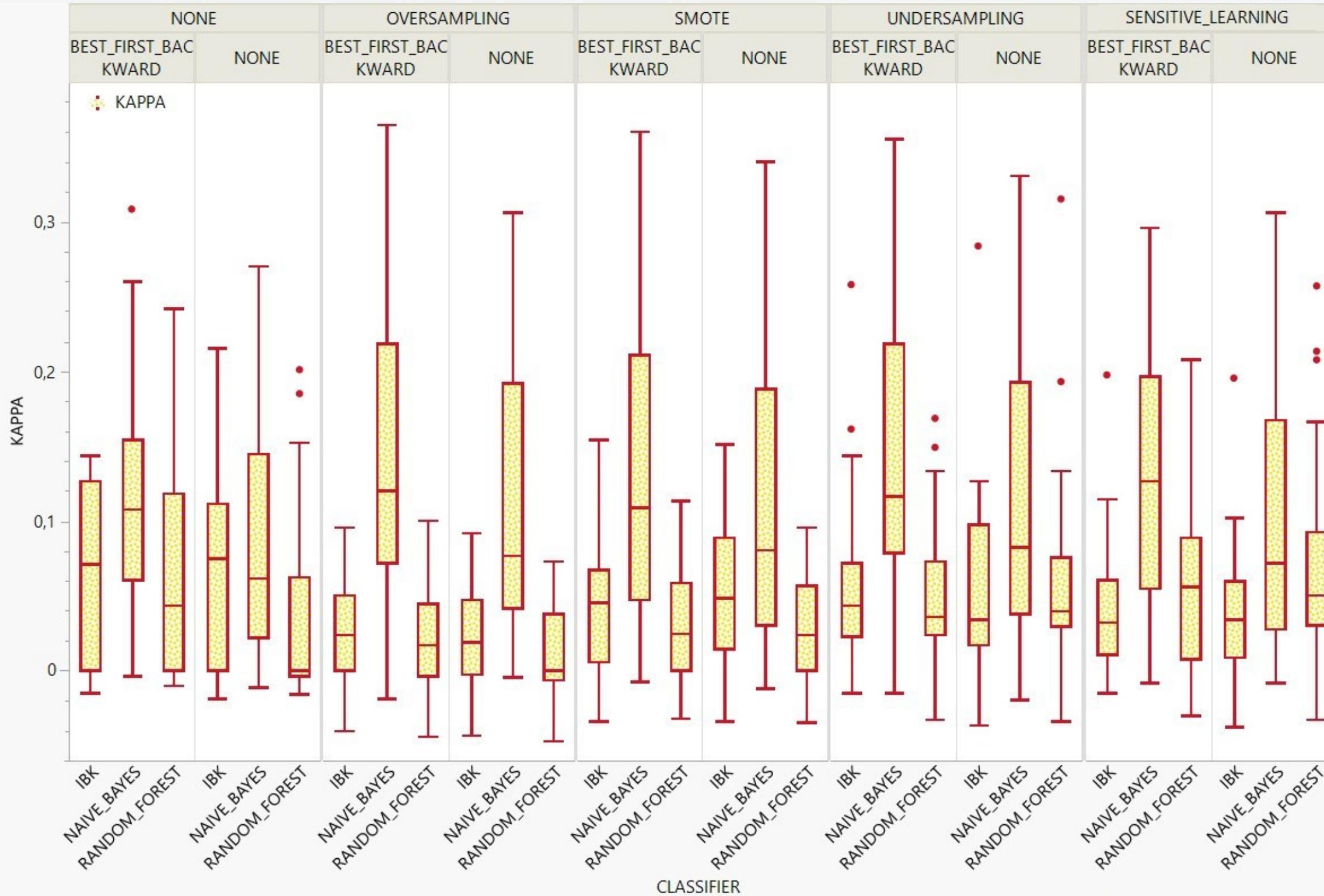
RECALL



La recall mostra molta meno stazionarietà. La tecnica migliore in assoluto è Undersampling con Random Forest e IBk. Al di fuori di Undersampling, il miglior classificatore risulta essere Naive Bayes.

RISULTATI - SYNCOPE

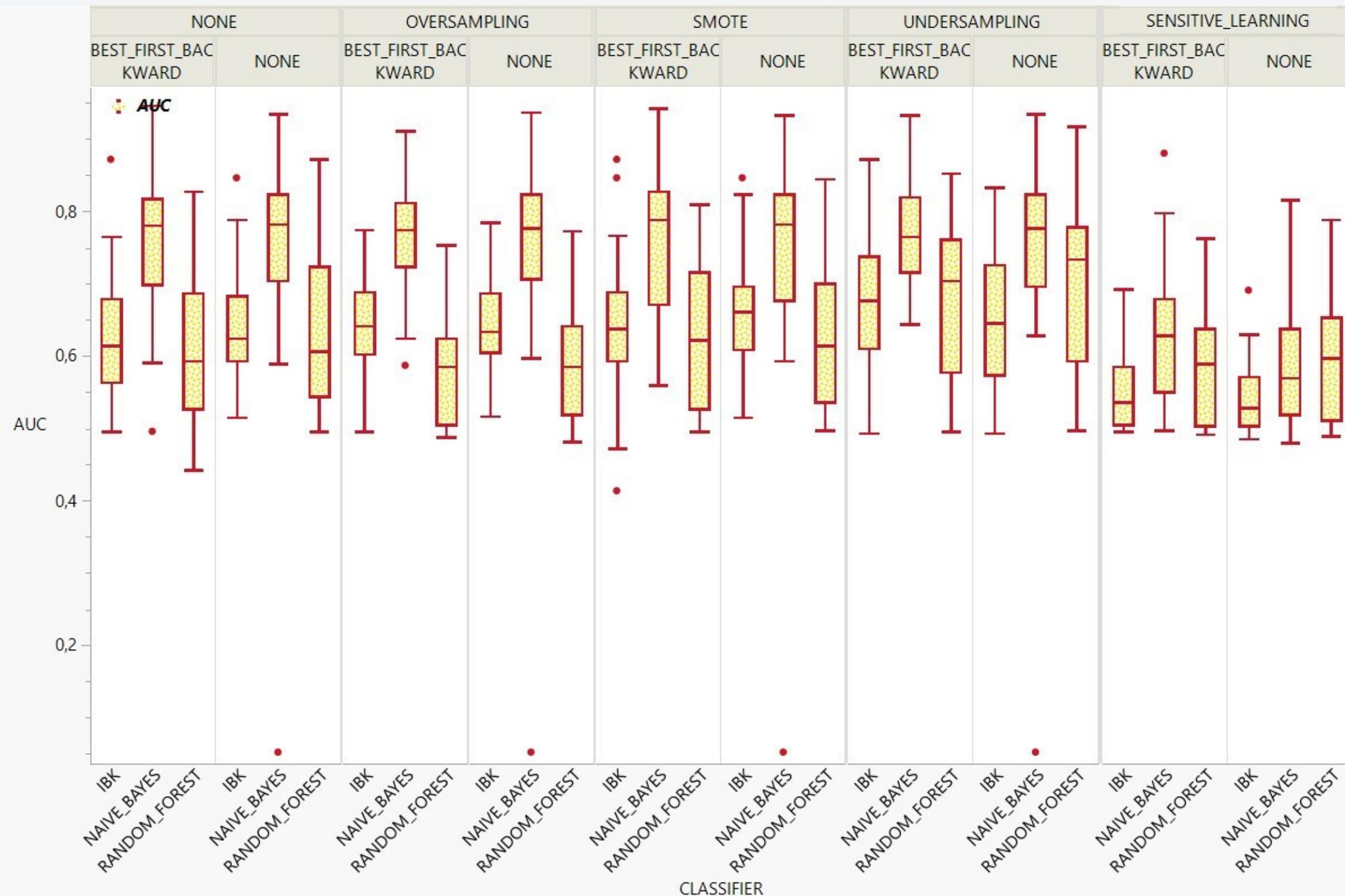
KAPPA



La KAPPA mostra ottimi valori con Naive Bayes e qualsiasi altra tecnica. Risulta, infatti, sempre il migliore. Per quanto riguarda le tecniche, SMOTE, Oversampling e undersampling hanno risultati molto simili e sono i migliori.

RISULTATI - SCOPE

AUC



Anche la AUC, mostra Naive Bayes come il classificatore migliore, mentre la tecnica con risultati peggiori è sensitive learning.

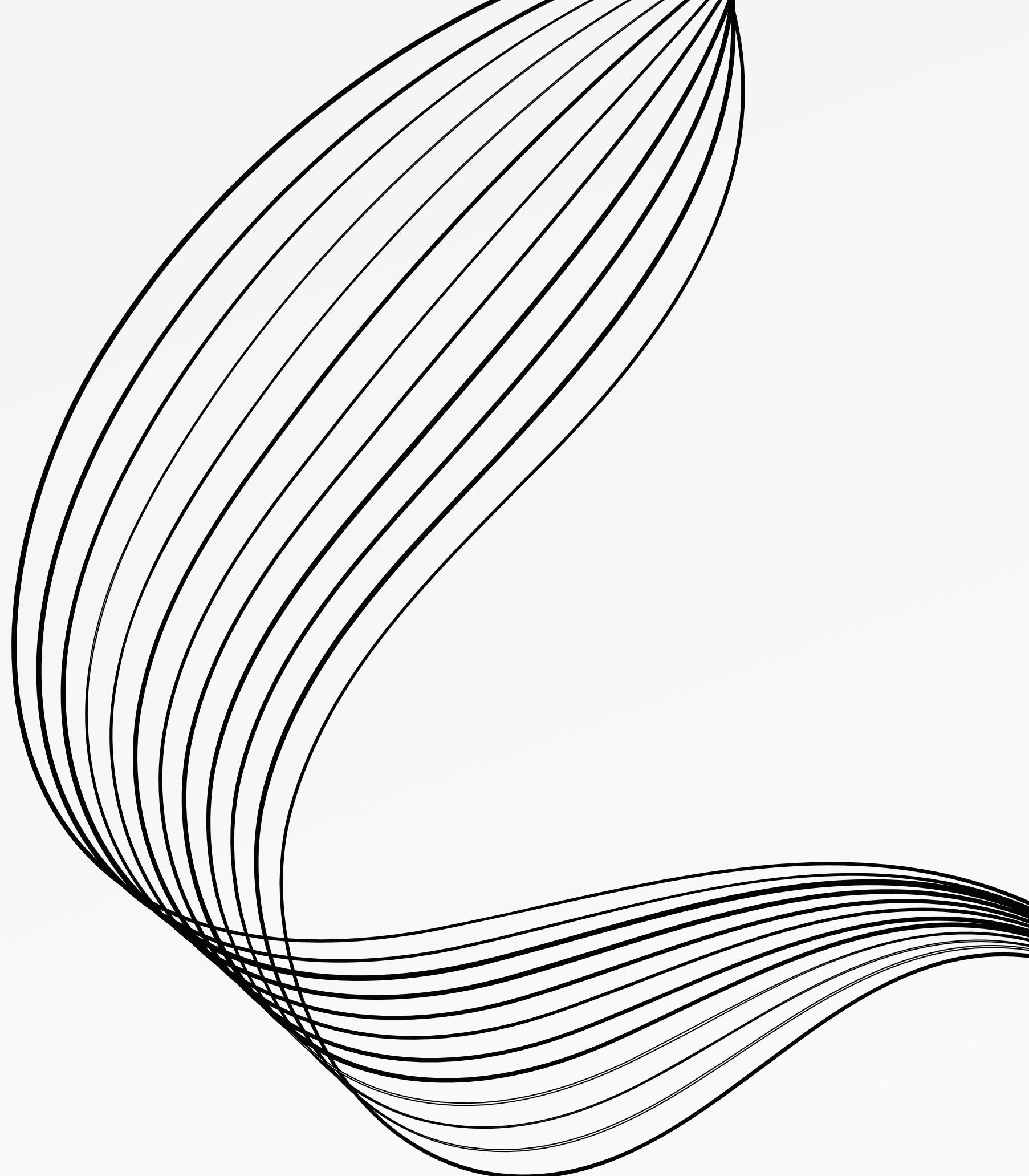
VALUTAZIONE DEI RISULTATI

In questo progetto, Naive Bayes è risultato spesso dominante sugli altri due classificatori, avendo sia una precision che una recall migliori.

Anche in questo caso, l'utilizzo della feature selection ha talvolta migliorato leggermente i risultati.

La combinazione migliore risulta essere Naive Bayes con Undersampling. Inoltre, l'utilizzo della feature selection permette di diminuire le metriche utilizzate senza influire sui risultati.

**GRAZIE PER
L'ATTENZIONE**



LINKS

Link GitHub: <https://github.com/andreaxdf/SoftwareInfoRetriever>

Link SonarCloud: https://sonarcloud.io/summary/overall?id=andreaxdf_SoftwareInfoRetriever