



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

SOFTWARE DEFECT PREDICTION

Chiara Iurato 0341704

CONTENUTO

- 01** INTRO
- 02** OBIETTIVI
- 03** CASO DI STUDIO
- 04** METODOLOGIA
- 05** RISULTATI
- 06** CONCLUSIONI
- 07** MINACCIE ALLA VALIDITÀ
- 08** RINGRAZIAMENTI E LINKS



INTRO

Nel campo dello sviluppo del software, garantire alta qualità e minimizzare i costi sono obiettivi fondamentali. Il software evolve rapidamente e con l'aggiunta continua di nuove funzionalità, frequenti rilasci di versioni, il rischio di introdurre nuovi bug aumenta significativamente. Ma come possiamo prevenire i bug?



SOLUZIONE



Uno dei topic di ricerca più popolari nel software engineering è il “**Software Defect Prediction**” che si occupa di prevedere i futuri difetti del software e di costruire teorie empiriche della qualità del software.
Durante questa presentazione, esploreremo questi temi in dettaglio, applicando tecniche di Machine Learning per rilevare e prevenire i bug nel software.

OBIETTIVI

Costruire un dataset che aggrega informazioni sulla storia passata



Valutare le prestazioni dei modelli al variare delle tecniche utilizzate



Identificare quali tecniche aumentano l'accuratezza dei classificatori

CASO DI STUDIO

Analizzeremo le classi “buggy” per i progetti di BOOKKEEPER e ZOOKEEPER. Come facciamo a sapere se una classe è stata buggy o meno? Di quali strumenti abbiamo bisogno?



METODOLOGIA

PRIMO PASSO

Recuperiamo tramite Jira tutte le release per un dato progetto

SECONDO PASSO

Utilizziamo Git per catturare tutti i commit di una data release, se una release non contiene commit, la scartiamo

METODOLOGIA

TERZO PASSO

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



Recuperiamo tutti i ticket di tipo "buggy fixed closed" e ne estraiamo le versioni affette (AV). Per i ticket il cui il campo AV è vuoto, possiamo utilizzare proportion. Nelle prime iterazioni, adottiamo la tecnica del cold start per calcolare la mediana di P basandoci su altri progetti Apache simili. Successivamente, utilizziamo un approccio incrementale per ottenere una media delle proportion calcolate sui difetti "fixed" delle versioni precedenti.

METODOLOGIA

QUARTO PASSO

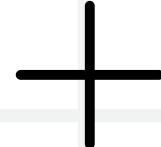
Estraiamo le classi toccate dai bug

Adesso abbiamo tutti gli elementi per costruire il dataset. Quali sono le metriche scelte?



METRICHE SCELTE

Size	Dimensione di una classe (linee di codice totali)
Number of Revision	Numero di revisioni
Number of Defect Fixes	Numero di difetti risolti
Number of Author	Numero di autori che hanno contribuito alla classe



LOC added	Linee di codice aggiunte
LOC removed	Linee di codice rimosse
LOC touched	Linee di codice toccate
Churn	$ LOC \text{ added} - LOC \text{ removed} $

NB: per ogni metrica considerare anche la media e il valore massimo

METODOLOGIA

QUINTO PASSO

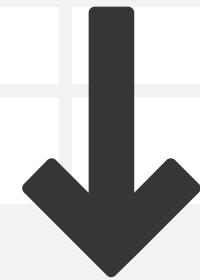
Prima di valutare il dataset dobbiamo fare attenzione al problema dello Snoring. Che cos'è?



I difetti non possono essere individuati prima che vengano risolti, il che significa che la presenza di bug dormienti può generare molti falsi negativi e peggiorare le prestazioni dei classificatori. Per evitare questo problema, è necessario non utilizzare dati troppo obsoleti o troppo recenti. Una soluzione efficace consiste nell'utilizzare le prime metà delle release di ogni progetto.

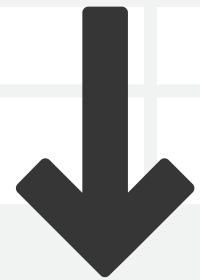


SESTO PASSO



TRAINING SET

E' il dataset utilizzato per addestrare i classificatori



TESTING SET

E' il dataset utilizzato per valutare la bontà delle predizioni fatte:

Più grande è il training set, migliore sarà il classificatore

SIZE MATTERS!



TRAINING SET

TESTING SET

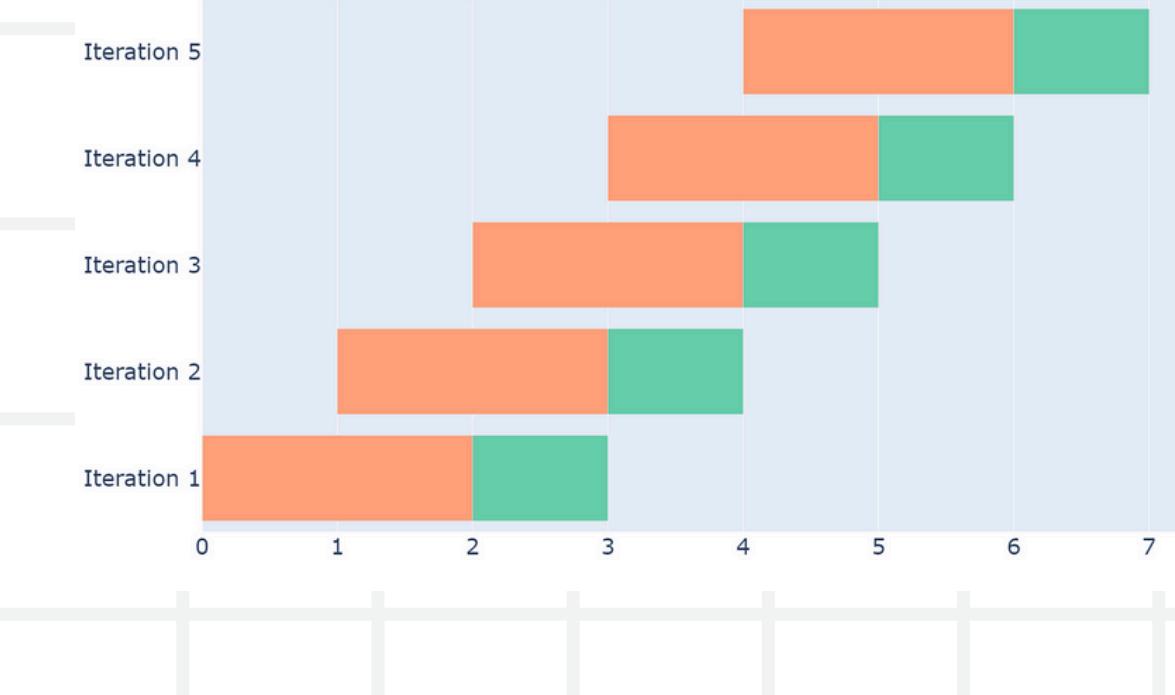
Più grande è il testing set, più accurata sarà la stima dell'errore

METODOLOGIA

SETTIMO PASSO

Per effettuare la validazione dei classificatori usiamo la tecnica del Walk Forward.

Questa tecnica risulta essere molto utile in contesti dove i dati sono “time-sensitive”. Il modello è addestrato su una finestra di dati storici e testato sulla finestra successiva, ripetendo il processo avanzando le finestre nel tempo. Questo simula la predizione futura basata su dati passati, migliorando la valutazione realistica delle prestazioni del modello.



CONFIGURAZIONE

CLASSIFICATORI

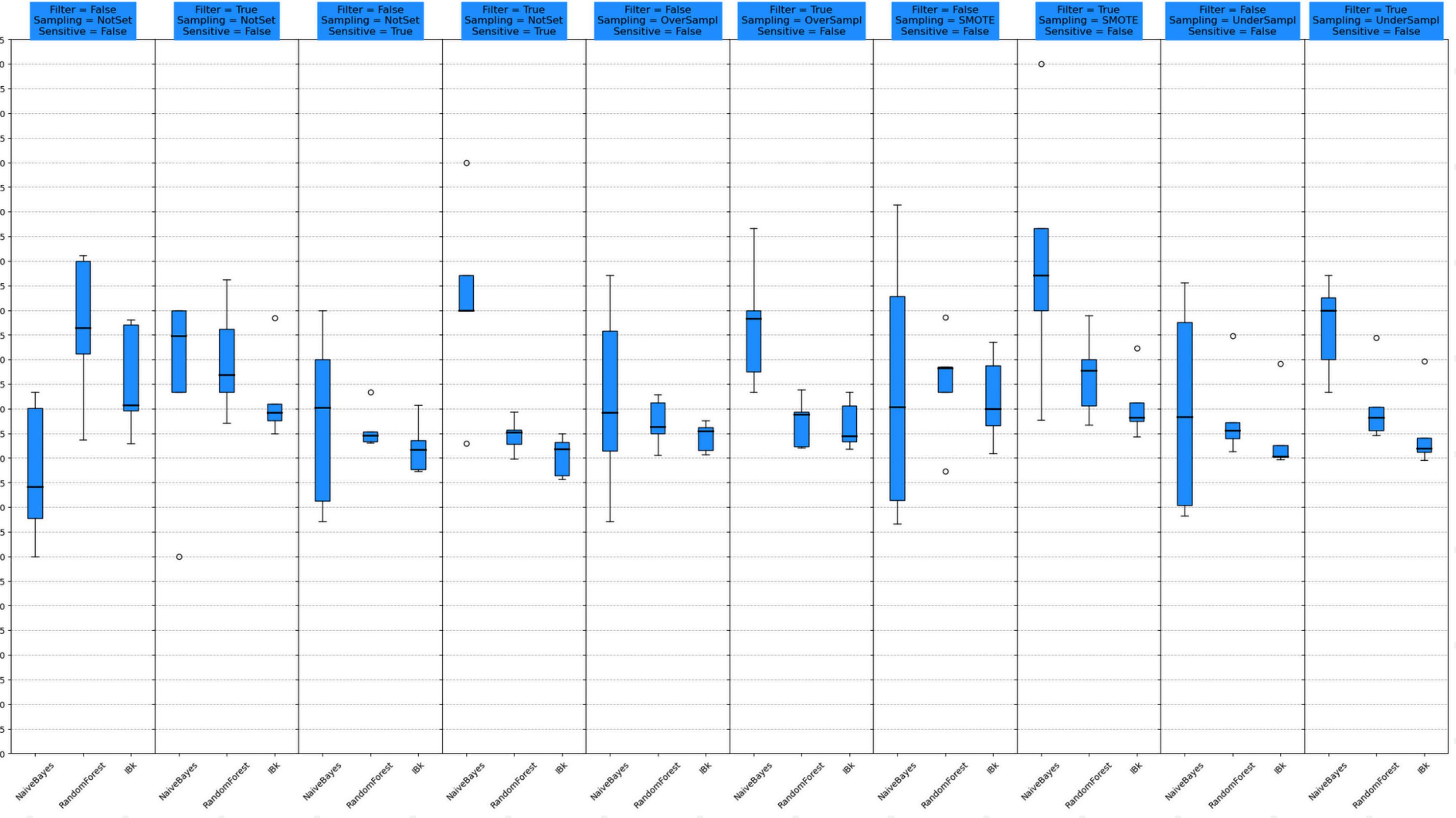
- Random Forest
- Naive Bayes
- Ibk

METRICHE

- Recall
- Precision
- Kappa
- AUC

TECNICHE

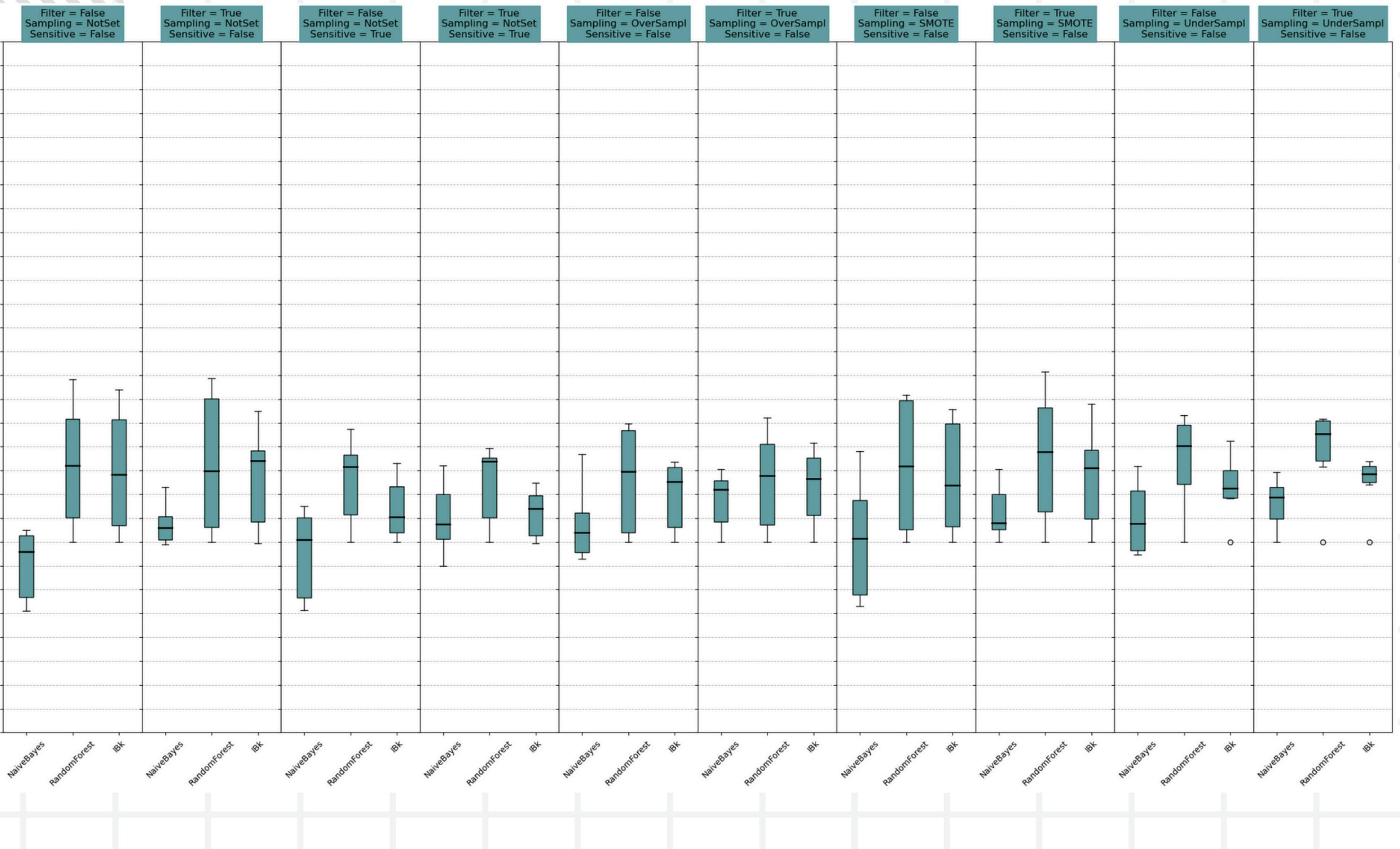
- Feature Selection tramite Best First (bi-directional)
- Balancing tramite Undersampling, Oversampling e SMOTE
- Sensitive Learning



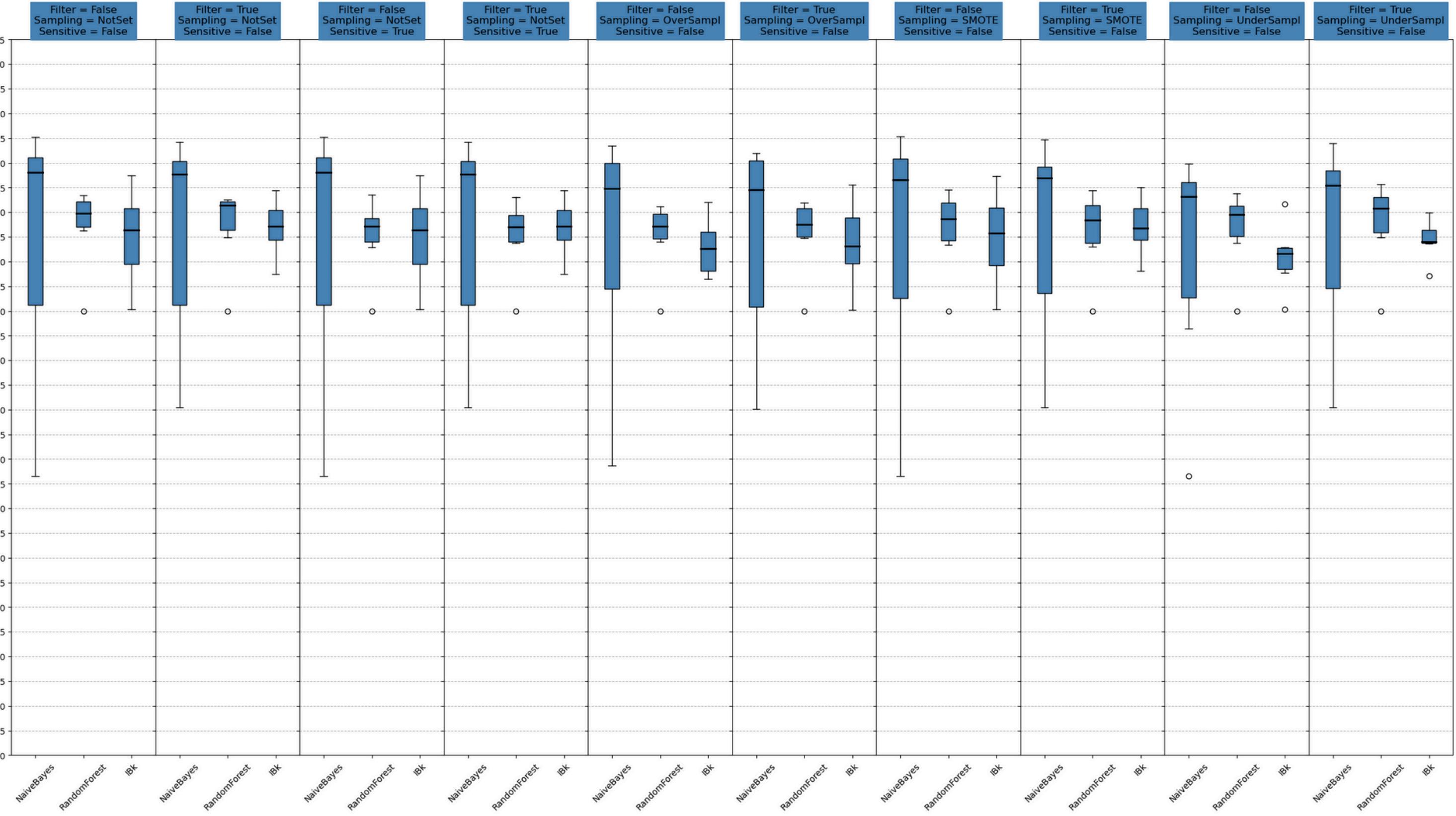
Naive Bayes beneficia notevolmente dalle tecniche di Sampling e Feature Selection, ottenendo buoni risultati soprattutto con SMOTE. Al contrario, Random Forest raggiunge un'ottima mediana anche senza preprocessamento. IBk, invece, tende a mostrare una performance inferiore, con una precisione particolarmente bassa quando combinato con la tecnica di Sensitive Learning.



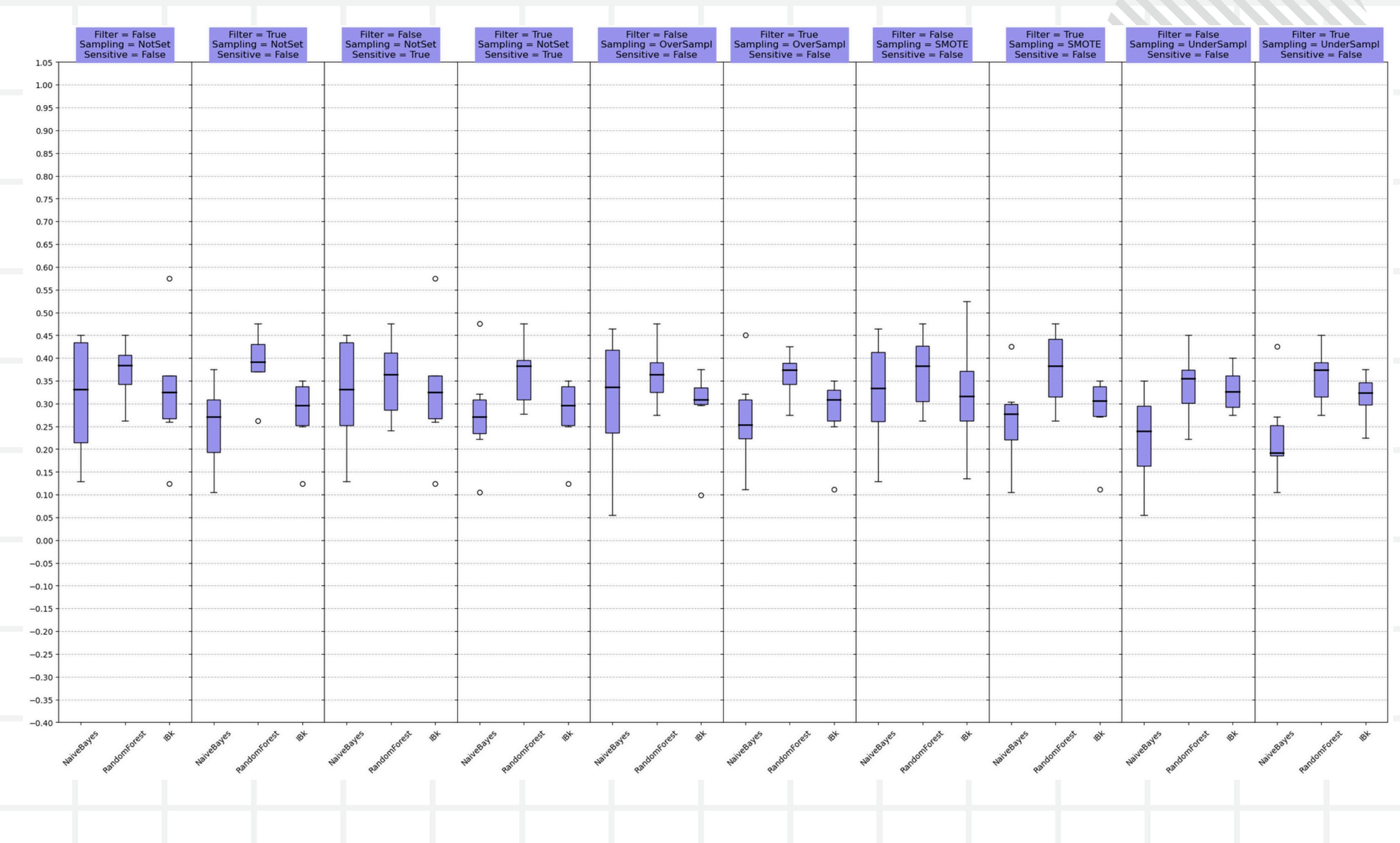
Come previsto, la recall più bassa si ottiene con Naive Bayes senza preprocessing, mentre quella più alta si raggiunge con Random Forest e IBK. Sensitive Learning ha contribuito ad aumentare la recall, ma la mediana massima si ottiene combinando le tecniche di UnderSampling e Feature Selection. In generale, Naive Bayes mostra scarse performance in tutte le configurazioni, con l'unica eccezione dell'OverSampling combinato con Feature Selection, che ha portato a un miglioramento.



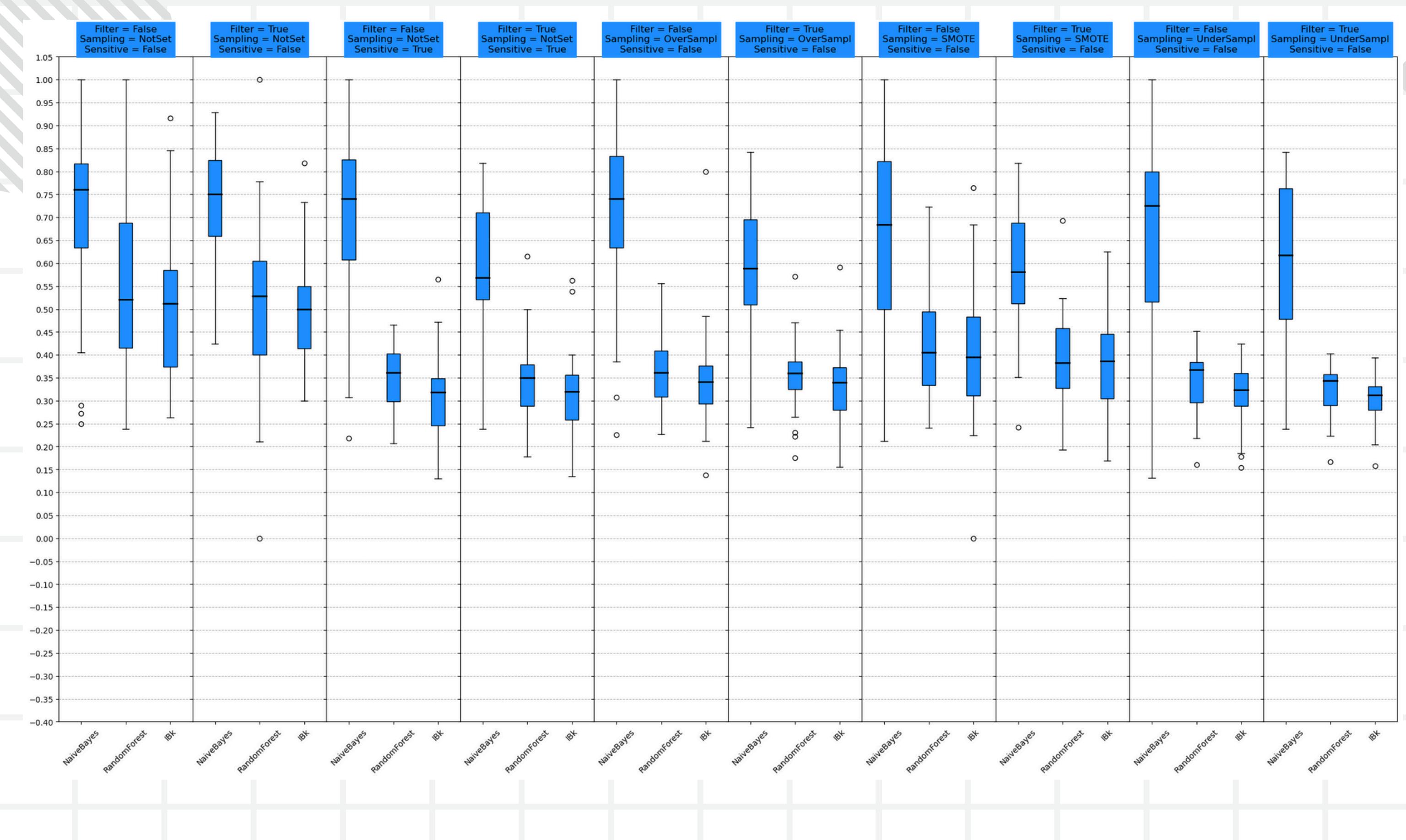
Kappa, che misura l'accuratezza di un classificatore rispetto ad uno dummy, raggiunge i valori massimi per Random Forest quando si utilizzano le tecniche di UnderSampling e Feature Selection. Al contrario, Naive Bayes mostra valori negativi sia con Sensitive Learning che con SMOTE. IBk ha le peggiori performance quando viene utilizzato con Sensitive Learning da solo, mentre ottiene risultati migliori con la combinazione di Sensitive Learning e Feature Selection.



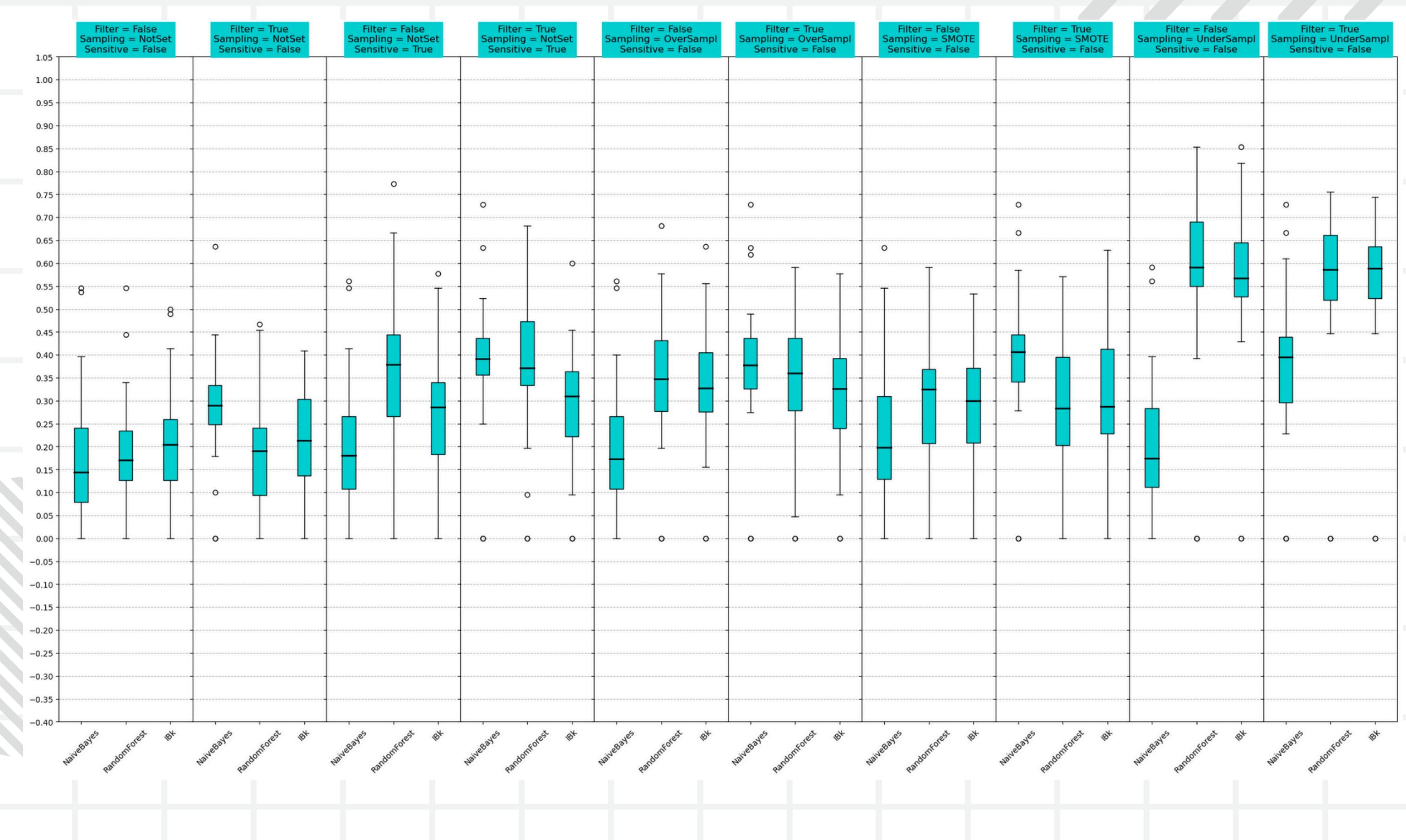
Per quanto riguarda AUC, Naive Bayes si distingue come il migliore tra i classificatori, nonostante una variazione molto elevata. Le tecniche di Sensitive Learning e Feature Selection non contribuiscono ad aumentare l'AUC, e l'UnderSampling lo peggiora ulteriormente. Random Forest, invece, mostra buoni risultati con la combinazione di UnderSampling e Feature Selection. IBk registra i valori più bassi, con performance peggiorate dall'UnderSampling e leggermente migliorate da Sensitive Learning e Feature Selection.



NPofB20 considera la percentuale delle entità difettose che possono essere identificate ispezionando il 20% delle linee di codice, ordinate secondo le probabilità predette di difetto. Random Forest tende a mostrare valori di NPofB20 più elevati in generale. La configurazione con solo Feature Selection dà la mediana più alta. Al contrario, Naive Bayes registra i valori più bassi con Feature Selection e UnderSampling. IBk raggiunge il suo picco quando usato solo con UnderSampling.



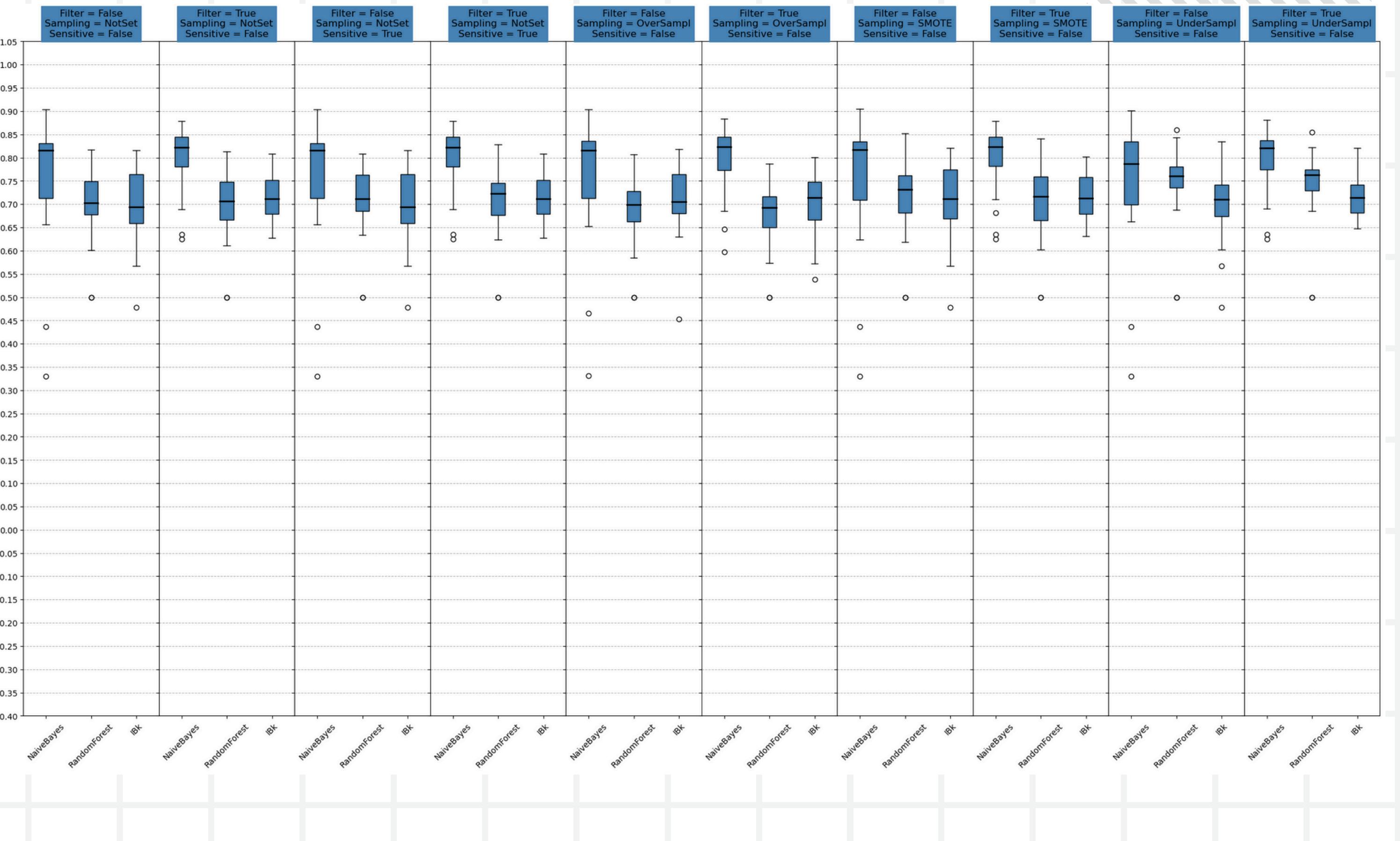
Rispetto a Bookkeeper, la precisione è significativamente più alta. Naive Bayes si distingue chiaramente come il miglior classificatore, mostrando la mediana più alta nella configurazione senza preprocessing. La tecnica di Sensitive Learning, da sola, non sembra migliorare le prestazioni degli altri classificatori, mentre la Feature Selection migliora complessivamente le prestazioni di tutti e tre i classificatori. Il classificatore meno performante risulta essere IBk, che registra il valore più basso quando combinato con Sensitive Learning



La recall più alta è ottenuta da Random Forest e IBk nelle configurazioni di UnderSampling con o senza Feature Selection. Sensitive Learning migliora le prestazioni generali di tutti e tre i classificatori, ma la configurazione ottimale è data dalla combinazione di OverSampling e Feature Selection. Naive Bayes mostra prestazioni inferiori in generale, soprattutto quando non viene applicata alcuna tecnica di preprocessing.

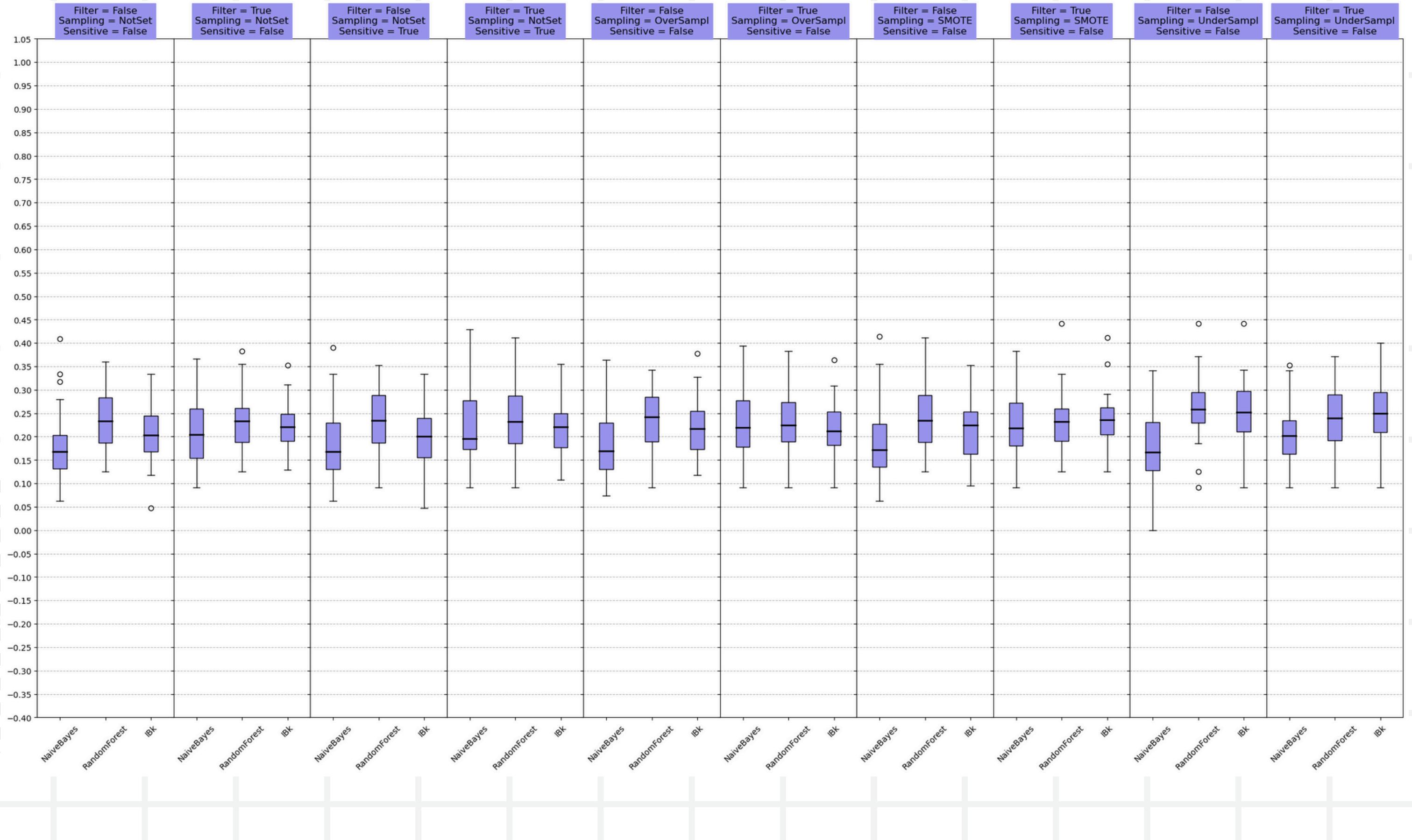


Kappa, a differenza di BooKeper, non mostra valori negativi e l'uso delle tecniche di Feature Selection e Sampling migliorano significativamente le performance di Naive Bayes. IBk mantiene performance costanti, ottenendo buoni risultati con l'UnderSampling da solo. L'uso del Sensitive Learning migliora leggermente le prestazioni di Naive Bayes e Random Forest, ma combinato con la Feature Selection si osserva un aumento significativo.



AUC mostra valori generalmente stabili con variabilità molto simile tra loro. Il classificatore che raggiunge i valori massimi è sempre Naive Bayes, seguito da Random Forest e IBk. La mediana più alta si ottiene combinando le tecniche di UnderSampling e Feature Selection. Possiamo notare anche che l'utilizzo del Sensitive Learning impatta le performance dei classificatori solo se combinato con Feature Selection.

Rispetto a BookKeeper, Naive Bayes risulta essere il peggiore, eccetto quando viene utilizzato solo con SMOTE. In generale, i valori limite sono molto simili tra tutte le configurazioni, e il classificatore che sembra comportarsi meglio è Random Forest. IBk supera di poco Random Forest quando usato con UnderSampling e Feature Selection.



CONCLUSIONI

Non esiste un classificatore meglio dell'altro, tuttavia possiamo notare che per **BookKeeper**:

- **Naive Bayes**: Dimostra un'alta precisione ma una recall molto bassa
- **Random Forest**: Beneficia particolarmente delle tecniche di UnderSampling e Feature Selection, ottenendo le migliori performance in termini di Kappa. Mantiene una performance costante e elevata con diverse configurazioni di bilanciamento delle classi
- **IBk**: Mostra performance inferiori rispetto agli altri classificatori, raggiungendo i migliori risultati di recall con UnderSampling e Feature Selection

Per **Zookeeper** invece:

- **Naive Bayes**: Si distingue per una precisione elevata, oltre a ottime performance in termini di Kappa e AUC
- **Random Forest**: Ottiene prestazioni complessivamente buone, con la recall più alta nelle configurazioni che includono UnderSampling
- **IBk**: Anche in questo caso registra la recall più alta con UnderSampling, ma presenta i valori di Kappa più bassi quando si utilizza la Feature Selection. Le sue performance complessive sono simili a quelle di Random Forest, ma inferiori.

MINACCE ALLA VALIDITÀ

Possiamo suddividere le minacce in quattro categorie principali:

- **Validità Interna:**
 - La rimozione delle release con date mancanti potrebbe escludere dati rilevanti, influenzando la completezza dell'analisi
 - Ignorare i ticket con OV uguale alla prima release estratta potrebbe introdurre un bias, eliminando bug rilevanti
 - L'eliminazione delle release senza commit associati potrebbe portare a una perdita di informazioni utili
 - L'affidamento alla data di risoluzione di JIRA senza verifiche ulteriori potrebbe introdurre errori se le informazioni di JIRA sono imprecise.
- **Validità Esterna:**
 - Utilizzare progetti Apache per il cold start potrebbe non essere rappresentativo per altri tipi di progetti software, limitando la generalizzabilità dei risultati.
- **Validità di Costrutto:**
 - Ignorare la prima iterazione nell'approccio di Walk Forward potrebbe trascurare informazioni utili
- **Validità Statistica:**
 - L'utilizzo di una soglia fissa per il cold start (=5) potrebbe non essere ottimale per tutti i dataset, influenzando l'accuratezza delle stime di proportion.
 - La mediana dei risultati dei progetti simili per il cold start potrebbe non essere rappresentativa dei dati attuali, introducendo un potenziale bias.



GRAZIE PER L'ATTENZIONE!

🔍 **GITHUB**

<https://github.com/chiaraiurato/ISW2>

🔍 **SONAR CLOUD**

https://sonarcloud.io/project/overview?id=chiaraiurato_ISW2