MASSIMO BUNIY - 0350022

MACHINE LEARNING PER IL SOFTWARE ENGINEERING

A.A. 2023-2024

AGENDA

1. Introduzione
2. Obiettivi
3. Dataset
4. Machine Learning
5. Risultati
6. Links

INTRODUZIONE

Nell’Ingegneria del Software è di fondamentale importanza svolgere un’**attività di testing** per riuscire a far emergere malfunzionamenti nel software.

Tuttavia, è irragionevole anche solo pensare di andare a fare testing esaustivo per tutti i possibili scenari di un’applicazione.

Per questo motivo è di fondamentale importanza riuscire ad indirizzare correttamente l'attività di testing su porzioni di codice che si ritengono maggiormente critiche.

COME FARE CIÒ?

Esperto Machine Learning

OBIETTIVI

MACHINE LEANING

L’obiettivo principale è quello di rendere il processo di testing più **efficiente** e **mirato**.

Nel contesto del progetto, l’obiettivo è quello di applicare **diversi modelli** di Machine Learning per due progetti Apache per ottenere **predizioni** sulla bugginess delle classi, analizzare le **prestazioni** dei diversi classificatori e **identificare** il classificatore ideale per i casi di analizi.

APACHE
BOOKKEEPER

APACHE
STORM

DATASET

COSTRUZIONE DEL DATASET

Un modello di Machine Learning si addestra su un dataset. Questo è un insieme di dati che il modello utilizzata per imparare.

Essendo i progetti considerati open source riusciamo a recuperare i dati del progetto.

Esistono molti tool che, durante il ciclo di vita di un programma, permettono di mantenere informazioni sul loro sviluppo. I due che andremo ad utilizzare noi per il recupero dei dati sono:

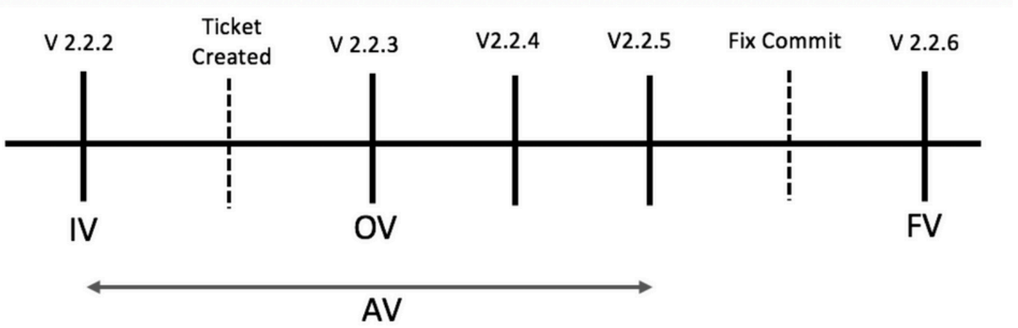
sistema per il tracciamento delle issue che raccoglie i ticket e le versioni di un software

sistema di controllo delle versioni che raccoglie i commit

DATASET

COSTRUZIONE DEL DATASET

Ciascun bug segue un ciclo di vita.



Le informazioni che riusciamo a prendere da **Jira** sono sempre la Opening Version e la Fixed Version di un ticket. D'altra parte, non tutte le issue aperte sono contrassegnate con una Injected Version.

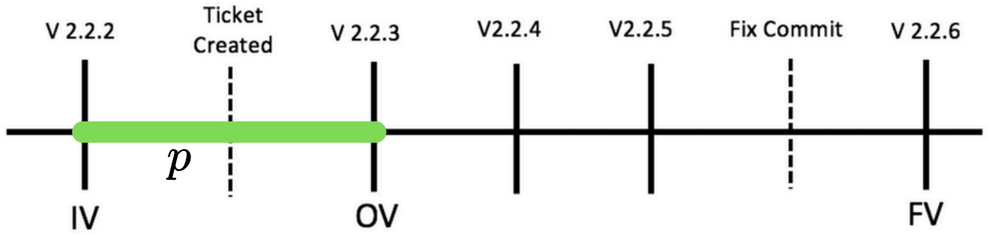
DATASET

COSTRUZIONE DEL DATASET

Non avento tutte i ticket segnati con una Injected Version, nel progetto si è fatto utilizzo della **Proportion** come tecnica di labeling per assegnarla ai ticket privi.

$$p = \frac{FV - IV}{FV - OV}, \quad IV = max\{1, FV - (FV - OV) \cdot p\}$$

$$\text{se } FV = OV \rightarrow p = FV - OV$$



Per questo motivo nel progetto si è fatto utilizzo della **Proportion** per assegnare una Injected Version ai ticket privi. Per computare "*p*" sono state utilizzate due varianti di proportion:

- **Cold Start**: nel caso i ticket precedenti al ticket analizzato siano meno di un valore di treshold pari a 5, si calcola "*p*" sfruttanto gli altri progetti Apache
- **Increment**

DATASET

METRICHE

I ticket di Jira ed i commit di git sono legati, poiché ad ogni ticket è assegnato un commit. Ricostruito il legame ho calcolato le metriche, ossia gli attributi che il modello di Machine Learning andrà ad utilizzare per fare le sue predizioni.

- Size (LOC): numero di linee di codice
- LOC touched: somma delle righe modificate in una release
- NR: numero di revisioni in una singola release
- NFix: numero di revisioni che erano di tipo fix
- NAuth: il numero di autori
- LOC Added: il numero di righe aggiunte dalle revisioni nella singola release
- Max LOC Added: numero massimo di righe aggiunte in una singola revisione
- Average LOC Added: media delle righe aggiunte dalle revisioni relative alla singola release
- Churn: somma delle righe di codice aggiunte meno le linee di codice rimosse
- Max Churn: valore massimo del churn in una singola revisione
- Average Churn: media dei churn sulle revisioni relative alla release

DATASET

BUGGYNBESS

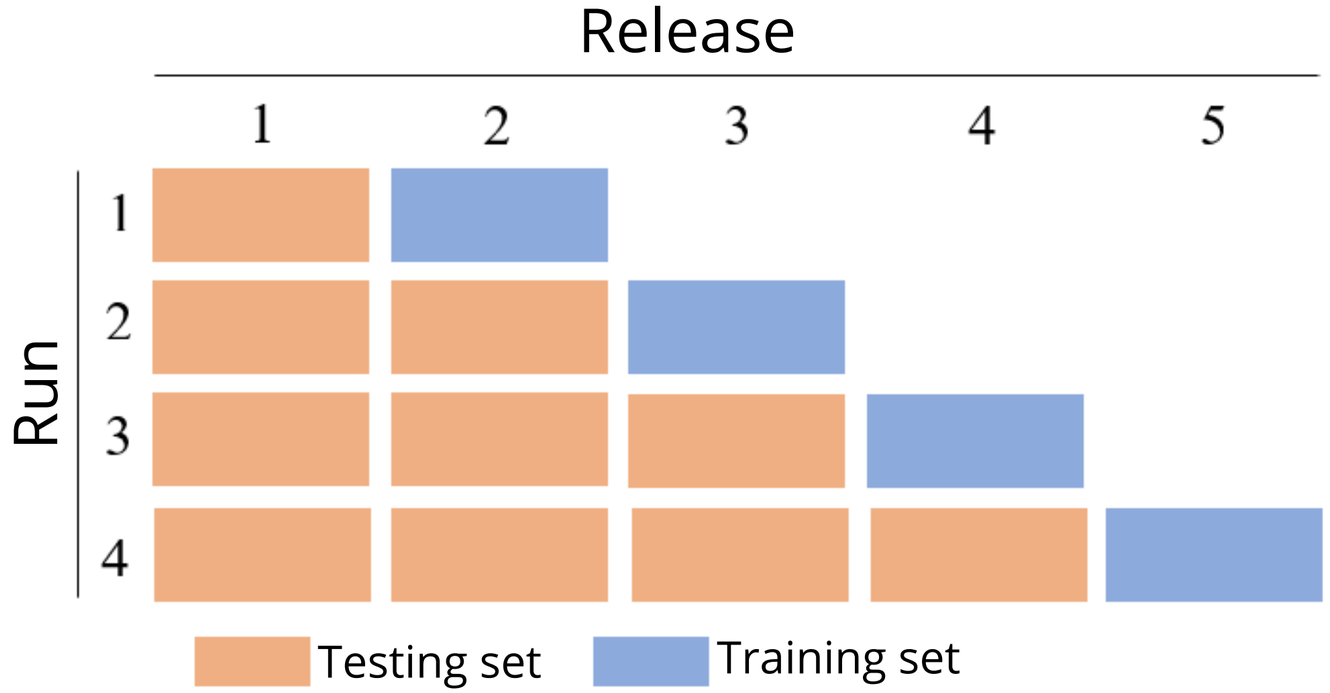
In fine nel dataset occorre andare ad indicare se la singola istanza era buggy o meno. Per fare questo si sono presi tutti i commit di tipo “bug” con risoluzione “fixed” e stato “close” o “resolved” da Jira e, per ogni classe Java modificata da un commit, si verifica se questo è associato ad uno di questi ticket. In caso di riscontro positivo, si etichetta la classe come buggy.

	Release ID	Filepath	Size	LOC Touched	NR	NFix	NAuth	LOC Added	Max LOC Added	Average LOC Added	Churn	Max Churn	Average Churn	Bugginess
1	1	bookkeeper-be…	137	386	9	1	7	262	137	29.11111111111111	138	137	15.333333333333334	no
2	1	bookkeeper-be…	244	1092	18	2	10	697	294	38.72222222222222	302	252	16.77777777777778	no
3	1	bookkeeper-se…	1026	9918	165	23	35	5024	545	30.44848484848485	130	545	0.7878787878787878	yes
4	1	bookkeeper-se…	92	485	19	2	13	428	84	22.526315789473685	371	82	19.526315789473685	yes
5	1	bookkeeper-se…	180	776	21	0	13	537	168	25.571428571428573	298	168	14.19047619047619	no
6	1	bookkeeper-se…	471	5776	88	12	28	3541	643	40.23863636363637	1306	487	14.840909090909092	yes

MACHINE LEARNING

WALK FORWARD

Ottenuto quindi il dataset complessivo, occorre suddividerli in traning set e testing set. Per costruire i set si utilizza come tecnica di valutazione il **Walk Forward**, tecnica di tipo **time-series**.



MACHINE LEARNING

SNORING

Come studiato, è possibile che un bug venga scoperto e corretto solamente dopo diverse release. Questo quindi causerà nel complesso problemi di snoring.

Per andare a cercare di mitigare questo aspetto, si è andati a rimuovere metà delle release a partire dalle più recenti. Così facendo il testing set sarà quello che subirà meno l'effetto dello snoring.

Tuttavia, ciò non è vero per le istanze del **training set**. Calcolando il valore di buggyness solamente con i ticket passati, l'effetto dello **snoring** è molto presente.

MACHINE LEARNING

WEKA

Per l’addestramento di è fatto utilizzo delle API di Weka.

I modelli che si sono andati a confrontare sono:

Random Forest **Naive Bayes** **IBk**

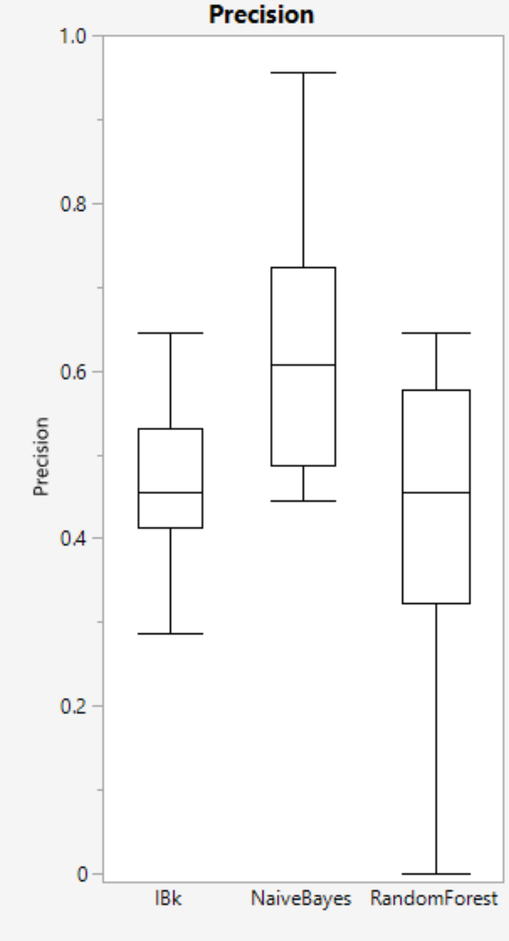
Come feature selection: **Best First**
Come tecnica di balancing: **SMOTE**
Come tecnica di cost sentive: **Sensitive Learning** con la matrice dei costi

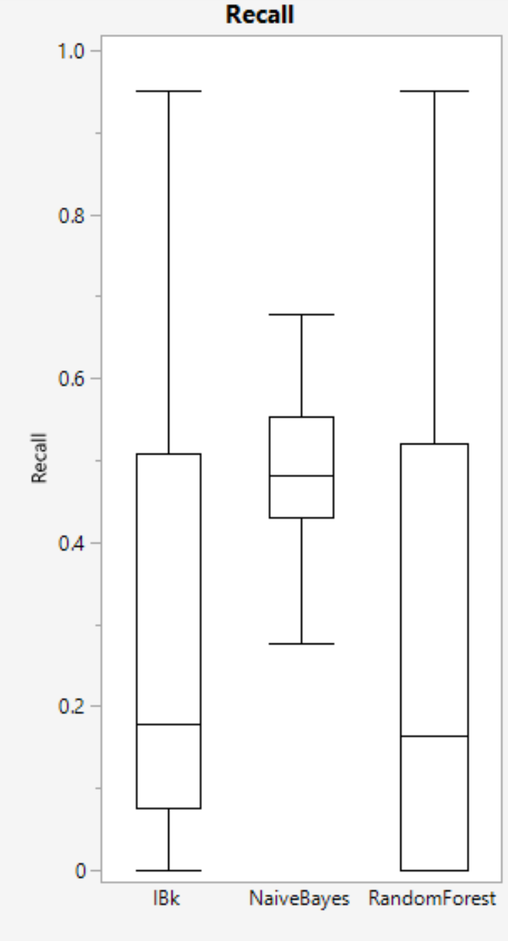
In totale sono state confrontate sei combinazioni:

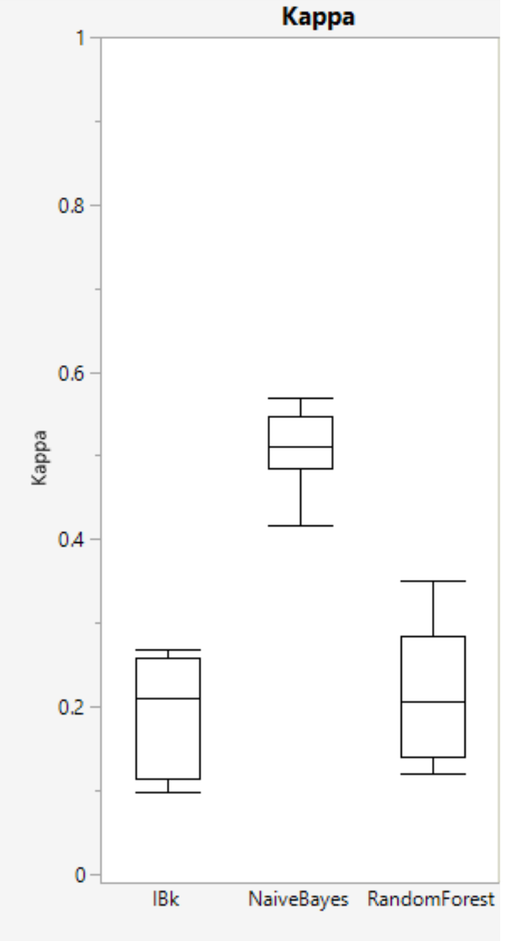
- addestrato il classificatore senza nulla
- solo feature selection
- solo sampling
- solo cost sensitive
- feature selection e sampling
- feature selection e cost sensitive

RISULTATI

BOOKKEEPER - NO FEATURE SELECTION, NO SAMPLING, NO COST SENSITIVE



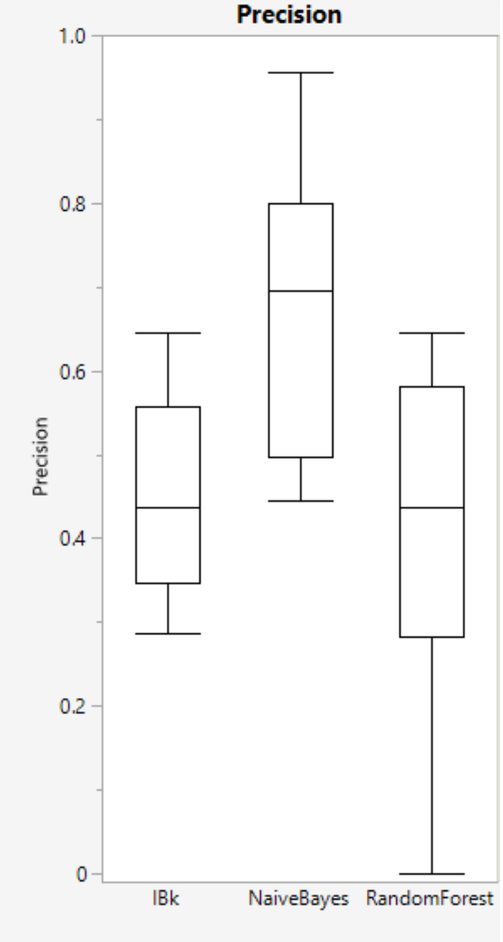


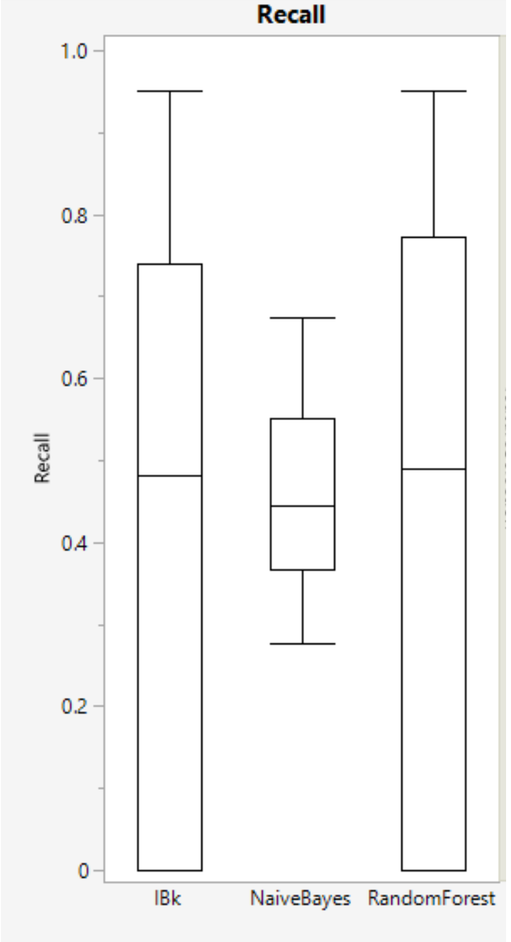


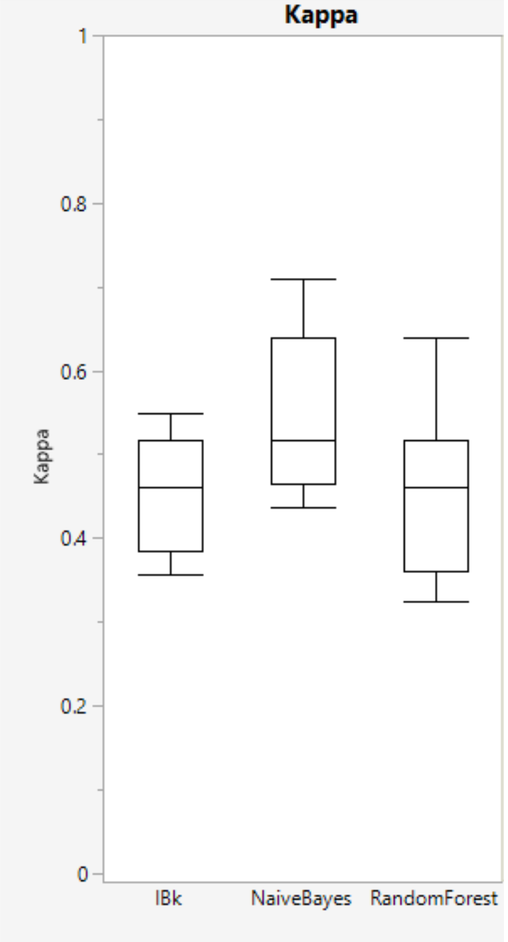
Si può vedere come nel complesso il classificatore Naive Bayes si comporti meglio rispetto agli altri due.

RISULTATI

BOOKKEEPER - FEATURE SELECTION, NO SAMPLING, NO COST SENSITIVE







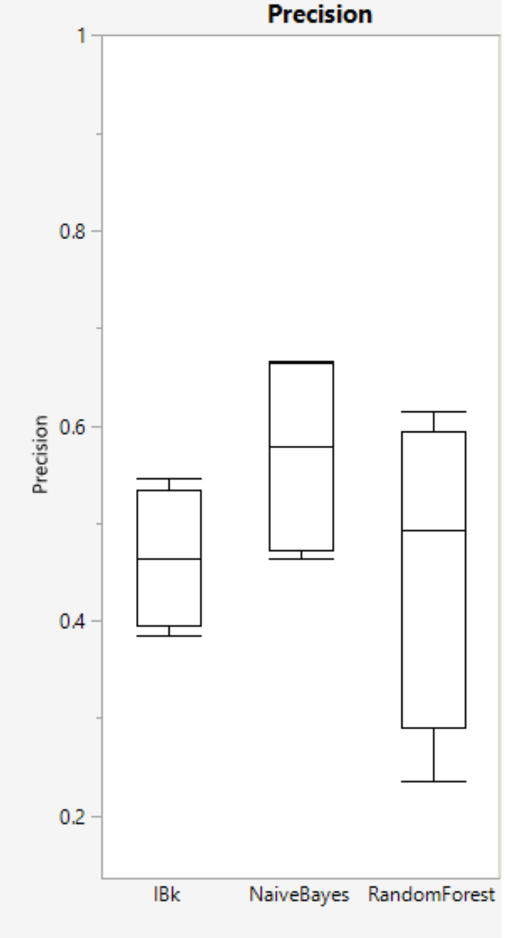
La precision è aumentata solamente per Naive Bayes e diminiutia per gli altri.

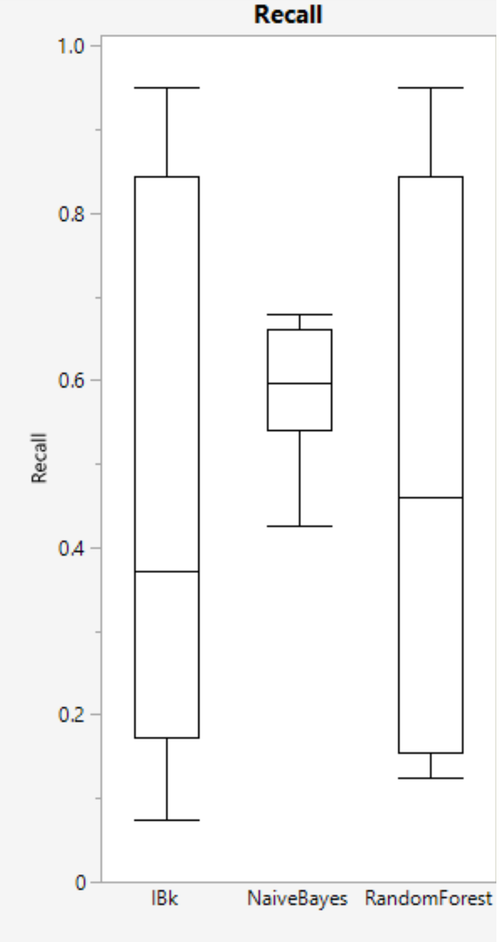
La recall invece è aumentata solamente per il Naive Bayes diminuendo di molto per IBk e Random Forest

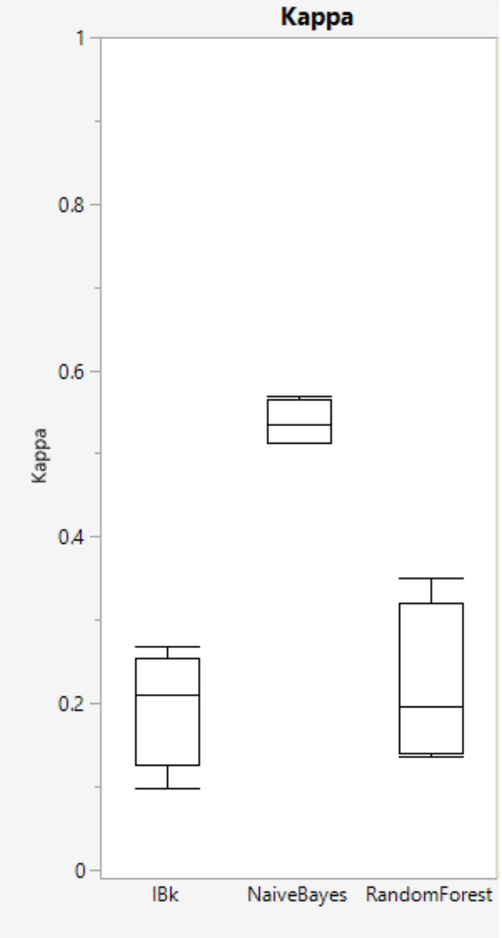
Kappa invece ha subito un incremento in generale, in particolare molto evidente per IBk e Random Forest.

RISULTATI

BOOKKEEPER - NO FEATURE SELECTION, SAMPLING, NO COST SENSITIVE







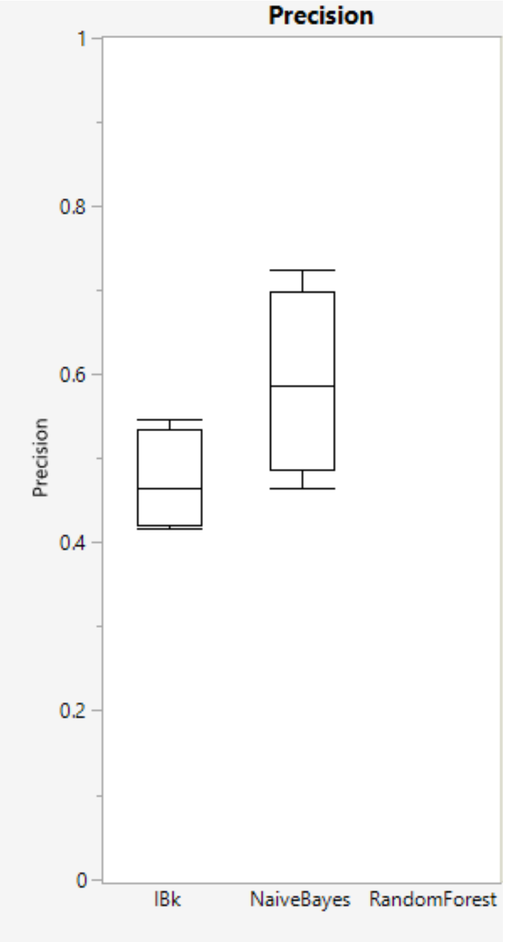
La precision ha subito un decremento in IBk e in Naive Bayes mentre per Random Forest ha subito un leggero miglioramento.

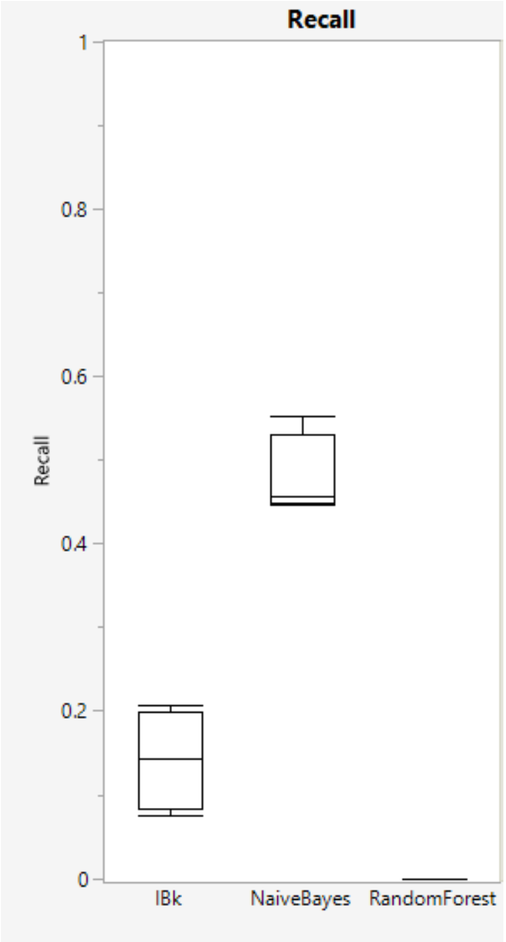
La recall in generale ha subito invece un enorme incremento.

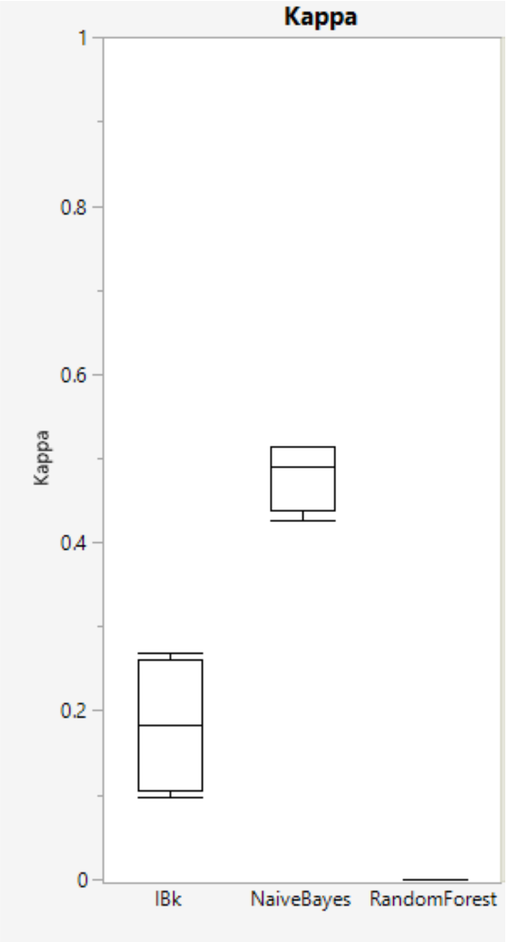
Kappa invece ha subito un miglioramento per Naive Bayes, ma restando pressoché identico per gli altri due.

RISULTATI

BOOKKEEPER - NO FEATURE SELECTION, NO SAMPLING, COST SENSITIVE







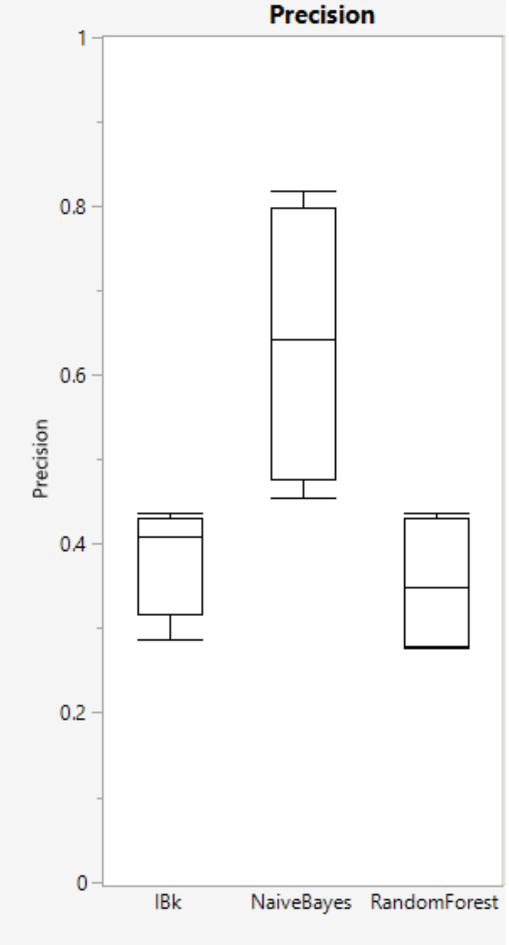
Inanzitutto occorre specificare come per il classificatore RandomForest, l'aggiunta del cost sensitive porta un errore nella sua classificazione.

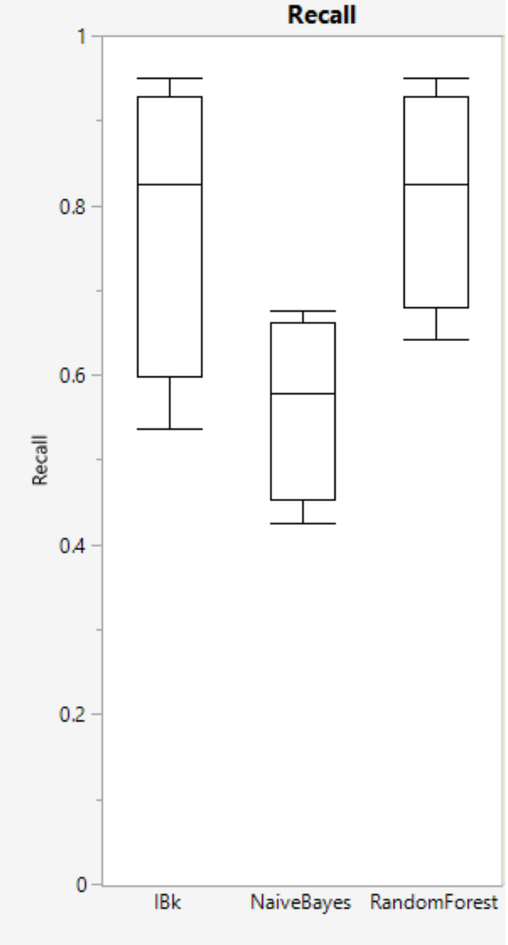
La precision risulta essere migliorata di poco solo per Naive Bayes e rimasta uguale per IBk.

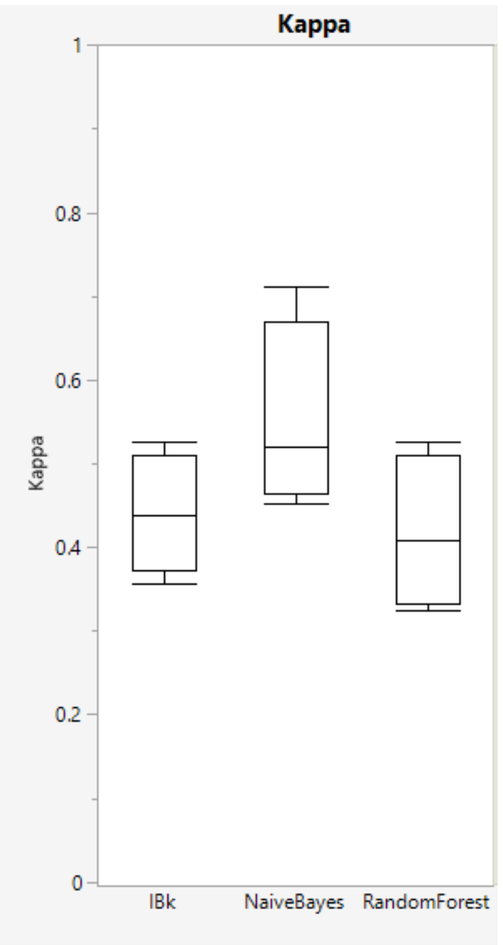
Per recall e kappa si ha un peggioramento sia per IBk ma anche per Naive Bayes

RISULTATI

BOOKKEEPER - FEATURE SELECTION, SAMPLING, NO COST SENSITIVE







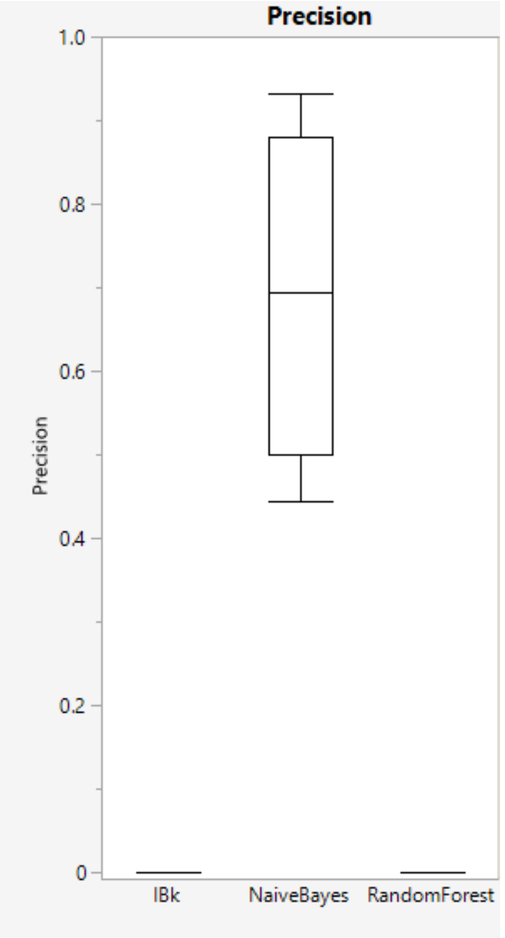
Rispetto al classificatore dei default.

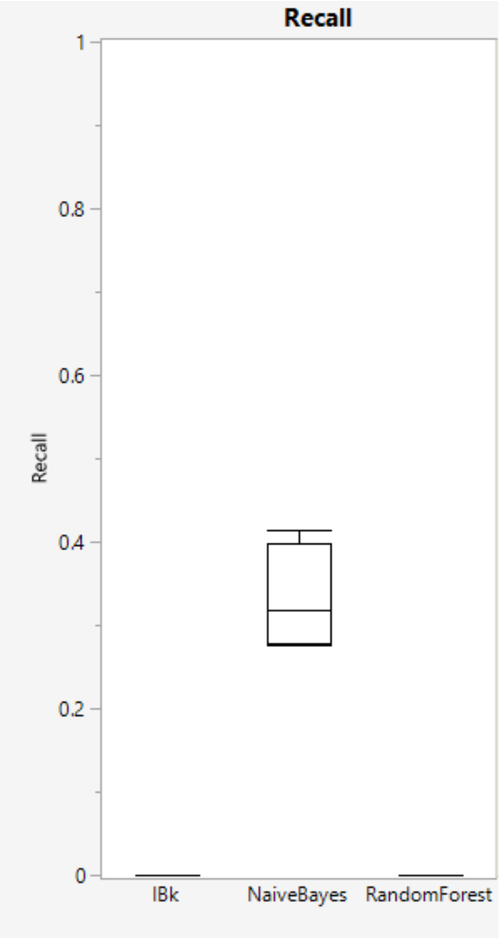
La precision è migliorata solamente per Naive Bayes, e diminuita per IBk e Random forest.

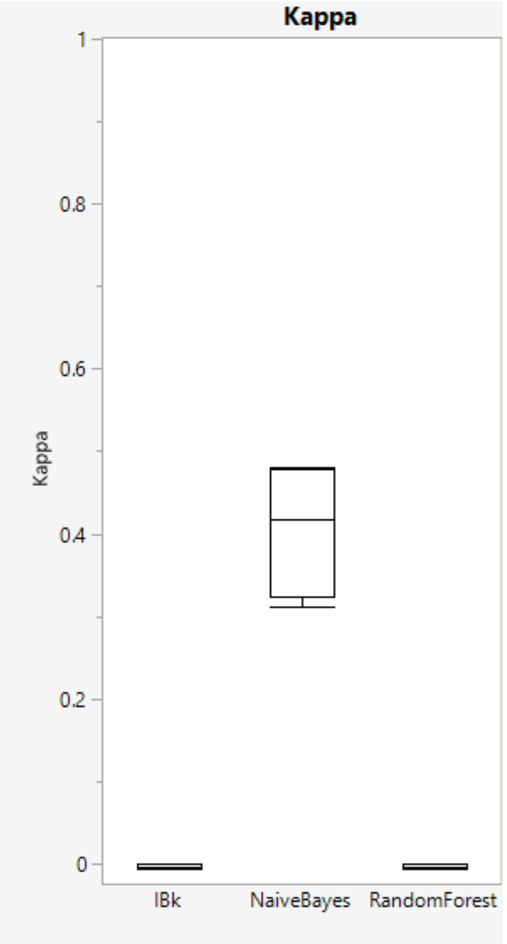
Sia la recall che kappa invece hanno subito un incremento generale, in particolare soprendono IBk e Random Forest.

RISULTATI

BOOKKEEPER - FEATURE SELECTION, NO SAMPLING, COST SENSITIVE







Come si vede dai grafici, i classificatori IBk e Random forest hanno avuto problemi andando a fare feature selection e applicando la matrice dei costi.

Valutando invece solamente Naive Bayes, la precision ha avuto un grosso miglioramento, mentre sia recall che kappa hanno subito un peggioramento evidente.

RISULTATI

BOOKKEEPER

Si è visto come il classificatore migliore per il progetto Bookkeeper tra quelli analizzati risulta essere **Naive Bayes** al quale si applicano sia **feature selection** tramite BestFirst e **sampling** con SMOTE.

Questa combinazione infatti ha uno dei valori di **recall più** alti mantenendo un livello di **precision elevato**. Ho considerato per l'analisi in particolare recall poiché un falso negativo ha un peso molto maggiore rispetto ad un falso positivo.
Nel nostro caso un falso negativo indicherebbe una classe buggy non etichettata come tale, comportando l'entrata in produzione del bug.

Non ho scelto il classificatore con sampling e feature, anche se aveva il valore di recall più alto, poiché la sua precisione era nettamente inferiore. Tenendo in considerazione i falsi positivi vanno ad incrementare il numero complessivo di classi da dover testare, portando quindi ad un incremento del costo per la fase di testing non indifferente.

RISULTATI

BOOKKEEPER

Se andassimo ad utilizzare altre metriche per la scelta del classificato cambierebbe il risultato?

Se andassimo a utilizzare come metrica NPofB20, considerando le medie rispetto al classificatore, avremmo che:

- IBk: 48,87%
- Naive Bayes: 55,69%
- Random Forest: 58,31%

Se invece vado a fare la media di NPofB20 solo sul classificato da me evidenziato dall'analisi tramite i grafici, otterrei una percentuale di 50,24%.

RISULTATI

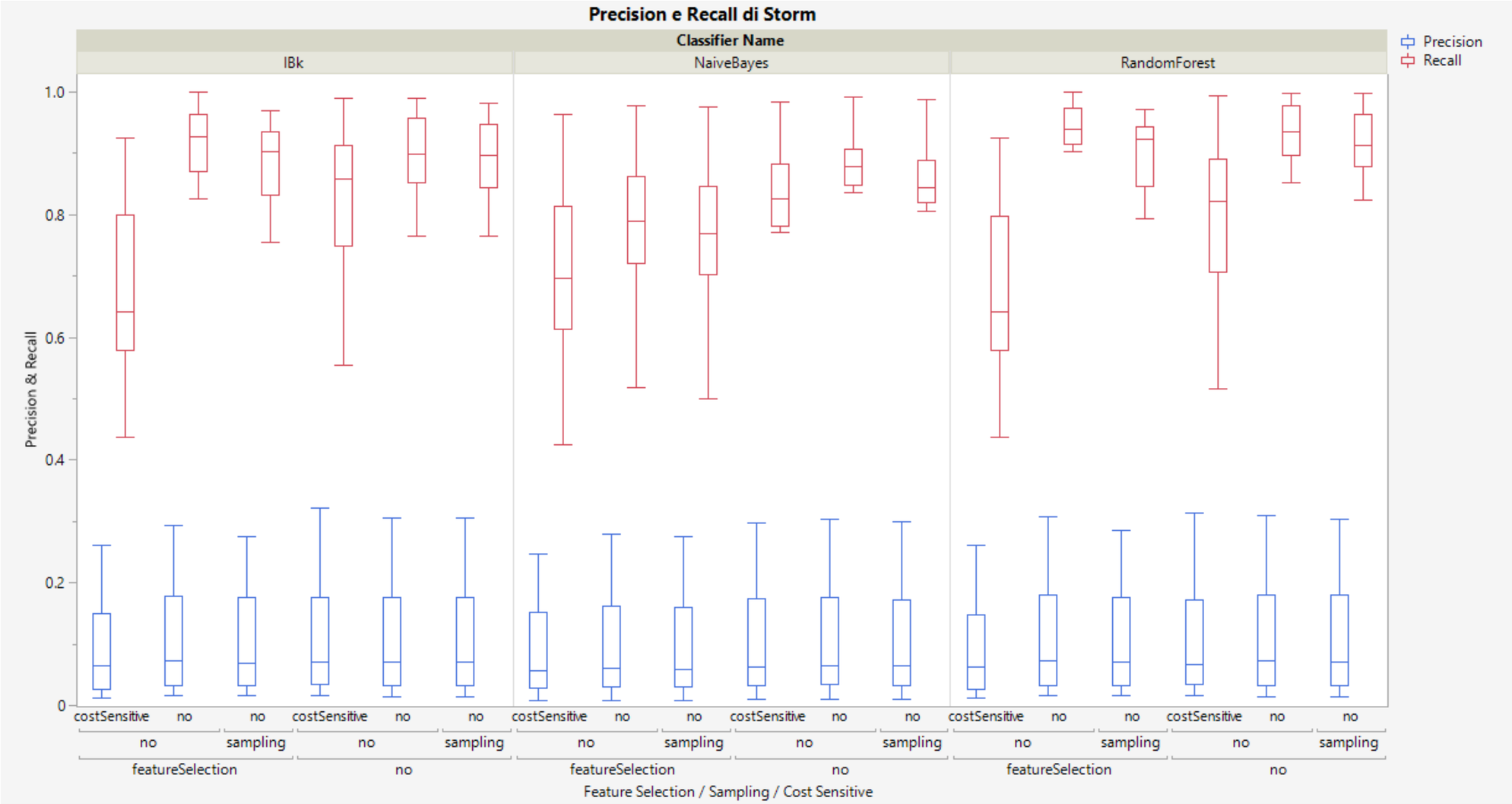
STORM

Prima di procede con l’analisi è doveroso specificare il motivo dell’assenza della metrica kappa. Infatti questa risulta essere negativa per tutti quanti i classificato e tutte quante le istanze del training set ad eccezione dell’ultima istanza, quella che considera complessivamente tutto il training set, senza walk forward.

Per questa ragione ho escluso la sua valutazione.

RISULTATI

BOOKKEEPER - FEATURE SELECTION, NO SAMPLING, NO COST SENSITIVE



RISULTATI

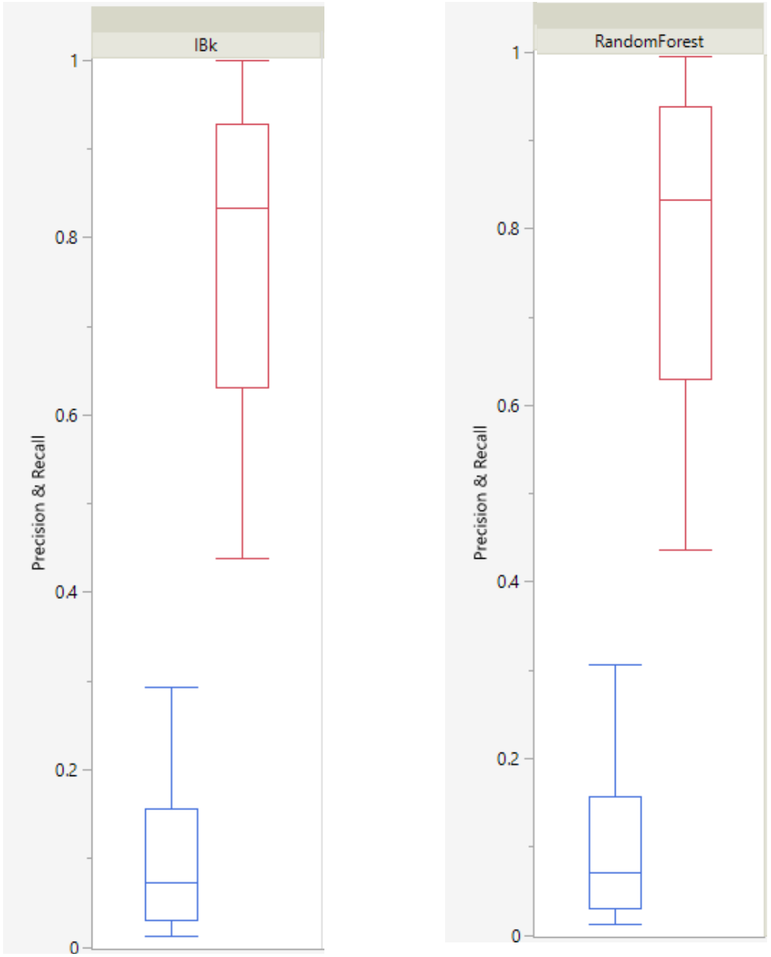
STORM

Dalla tabella si legge che Naive Bayes rispetto a IBk e Random Forest risulta avere prestazioni generalmente peggiori.

Per valutare quale classificatore utilizzare tra IBk e Random Forest sono andato a vedere il migliore di entrambi maggiormente nel dettaglio.

RISULTATI

STORM



Per entrambi sono andato a selezionare il classificatore che aveva impostato solamente il feature selection

Dal grafico la differenza è impecettibile, per cui sono dovuto ricorrere a leggere i valori effettivi. Se guardiamo numericamente la mediana di IBk è superiore di uno percentile della mediana di Random Forest rispetto a precision. Tuttavia rispetto a recall la mediana di Random Forest è superiore di due punti percentile a quella di IBk.

RISULTATI

STORM

Se andassimo ad utilizzare altre metriche per la scelta del classificato cambierebbe il risultato?

Se andassimo a utilizzare come metrica NPofB20, considerando le medie rispetto al classificatore, avremmo che:

- IBk: 37,4%
- Naive Bayes: 37,65%
- Random Forest: 36,75%

Se invece vado a fare la media di NPofB20 sui due classificatori da me evidenziati avrei come percentuale:

- IBk con solo feature selection: 37,73%
- Random Forest con solo feature selection: 37,43%

LINKS

https://github.com/Ralisin/isw2

https://sonarcloud.io/project/overview?id=Ralisin_CodeMetricsISW2