



ML FOR SOFTWARE ENGINEERING

Luca Saverio Esposito 0334321



AGENDA

- **Introduzione al contesto, obiettivi del progetto.**
- **Metodologia:**
 1. Individuazione classi buggy
 2. Costruzione del dataset, metriche considerate
 3. Valutazione dei classificatori
- **Risultati ottenuti, conclusioni.**
- **Link al repository GitHub, link SonarCloud.**

INTRODUZIONE

CONTESTO

Qualsiasi progetto software che si rispetti prevede un'attività di testing con lo scopo di individuare bug nel software.

Questa attività risulta esser onerosa e spesso complessa. Ridurre i costi ed efficientare il processo sono obbiettivi cardine di tutte le aziende.

Spesso la problematica principale è individuare quali porzioni del progetto e quante risorse assegnargli per il testing.

INTRODUZIONE

CONTESTO(2)

L'idea alla base dello studio è quella di sfruttare le informazioni del passato riguardo le classi caratterizzate da bug per predire quali classi nel futuro potranno averne.

Il tutto è realizzato tramite strumenti di Machine Learning utilizzati con un approccio di tipo black-box.



STEP PRELIMINARI

Dati due progetti open-source di Apache (BookKeeper e Tajo), realizzare un software in grado di creare un dataset completando i seguenti passi:

- individuare le classi che sono state buggy tra le varie release
- calcolare metriche del software associate alle classi
- realizzare un file arff contenente classi buggy e metriche associate

OBIETTIVO

Una volta ottenuto il dataset, utilizzare le API di Weka per stabilire quale tra i tre classificatori considerati ovvero: **Naive Bayes, Random Forest** e **IBk** effettua le predizioni migliori.

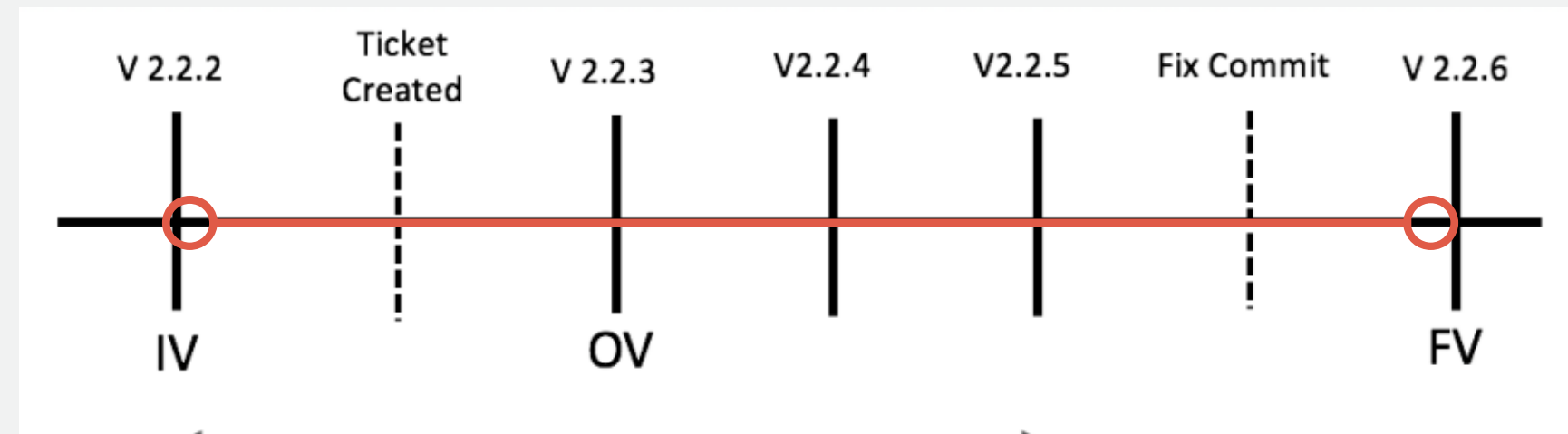
I vari classificatori saranno utilizzati insieme a tecniche di **feature selection, sampling, cost sensitivity** anche in combinazione tra loro.



METODOLOGIA

INDIVIDUARE COPPIE (CLASSE,RELEASE) BUGGY

- **IV** è la release in cui è stato introdotto il bug.
- **OV** la release in cui il bug è stato rivelato a seguito di una failure
- **FV** è la release in cui il bug è stato eliminato



Le classi risultano difettose dall'IV (inclusa) alla FV (esclusa)

METODOLOGIA

INDIVIDUARE COPPIE (CLASSE, RELEASE) BUGGY(2)

Le informazioni su quali siano OV e FV sono ottenute tramite Jira. Tuttavia, la IV non è sempre presente, perciò è necessario ottenerla in altro modo.

Il **Proportion** è una tecnica che permette di stimare l'injected version dei bug utilizzando una costante di proporzionalità p calcolata sui bug per cui IV è nota.

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

$$IV = FV - (FV - OV) \times P$$

METODOLOGIA

INDIVIDUARE COPPIE (CLASSE, RELEASE) BUGGY(3)

Il valore di P è utilizzato per calcolare l'injected version laddove non è presente.

Esistono diverse varianti di proportion, la scelta è ricaduta su ColdStart.

L'approccio consiste nel calcolare la P del progetto studiato utilizzando le P medie di altri progetti. Affinché la tecnica abbia senso: P_i devono essere simili.

- Per ogni progetto viene calcolata **P_i** ovvero **la media** delle P dei vari bug
- Viene presa **la mediana** tra le P_i e viene utilizzata come **$P_ColdStart$**

Per mantenere l'omogeneità tra i valori è stato scartato il P_i di Storm che risultava troppo grande rispetto a quelli di Avro, OpenJPA, Zookeeper e Syncope.

METODOLOGIA

COSTRUZIONE DEL DATASET

Individuate le injected version di tutti i bug ed eseguito il labeling ovvero l'associazione tra classe buggy e release in cui lo è stata; lo step successivo è quello del calcolo le metriche necessarie per la predizione.

Con l'intento di verificare empiricamente la validità del concetto **less is more**, sono state scelte svariate metriche ridondanti tra loro, attendendosi dei risultati decisamente migliori nella predizione a seguito dell' utilizzo di feature selection.

METODOLOGIA

METRICHE CONSIDERATE

NOME	DESCRIZIONE
LOC	Numero di linee di codice.
LOC_ADDED	Somma delle linee di codice aggiunte tra le varie revisioni.
MAX_LOC_ADDED	Numero max di linee di codice aggiunte in una singola revisione.
AVG_LOC_ADDED	Media delle linee di codice aggiunte sulle revisioni.
LOC_DELETED	Somma delle linee di codice rimosse tra le varie revisioni.
MAX_LOC_DELETED	Numero max di linee di codice rimosse in una singola revisione.
AVG_LOC_DELETED	Media delle linee di codice rimosse sulle revisioni.
CHURN	Somma tra le revisioni di una release di LOC_ADDED - LOC_DELETED
MAX_CHURN	Valore massimo del churn in una singola revisione.
AVG_CHURN	Media dei churn sulle revisioni relative alla release
FIXED_DEFECTS	Numero di difetti fixati.
NUMBER_OF_COMMITS	Numero di commits.
NUMBER_OF_AUTHORS	Numero di autori.
NUMBER_OF_REVISION	Numero di revisioni.

METODOLOGIA

VALUTAZIONE DEI CLASSIFICATORI

Tenendo in mente che l'obiettivo è quello di stabilire quale combinazione classificatore/tecnica di utilizzo ha ottenuto i risultati migliori, è necessario utilizzare una **tecnica di valutazione**.

La tecnica usata è **Walk Forward**

Run	Part				
	1	2	3	4	5
1	Training	Testing	Testing	Testing	Testing
2	Training	Training	Testing	Testing	Testing
3	Training	Training	Training	Testing	Testing
4	Training	Training	Training	Training	Testing
5	Training	Training	Training	Training	Training

Testing
Training

METODOLOGIA

VALUTAZIONE DEI CLASSIFICATORI(2)

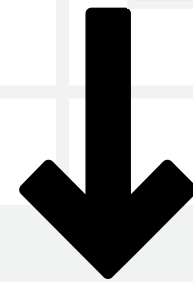
Walk forward è una tecnica di validazione di tipo **time-series**, ciò significa che è indispensabile tener conto dell'ordine temporale dei dati.

Il dataset viene **diviso in parti**, per esempio, **per ogni release**. Le parti vengono ordinate cronologicamente e in ogni run tutti i dati antecedenti rispetto a quelli da predire, ossia il test set, vengono usati come training set.

La prima iterazione, con solo testing set, non è stata effettuata. Inoltre, il processo di walk forward si ferma alla **prima metà** delle release scartando quelle più recenti. Questo per **evitare** di avere fenomeni di **snoring** nel testing set.

METODOLOGIA

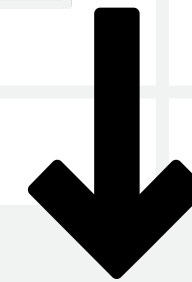
VALUTAZIONE DEI CLASSIFICATORI(3)



TRAINING SET

- E' affetto da snoring.
- Non può sfruttare dati del "futuro".

Viene ricalcolata la buggyness delle classi del set ad ogni iterazione i del walk forward con i ticket disponibili fino alla versione i -esima.



TESTING SET

- Non è affetto da snoring.
- Deve esser il più possibile fedele alla realtà.

Per il labelling del testing set sono stati usati tutti a ticket disposizione, cercando di limitare il più possibile fenomeni di snoring.

METODOLOGIA

CLASSIFICATORI E TECNICHE DI UTILIZZO

CLASSIFICATORI

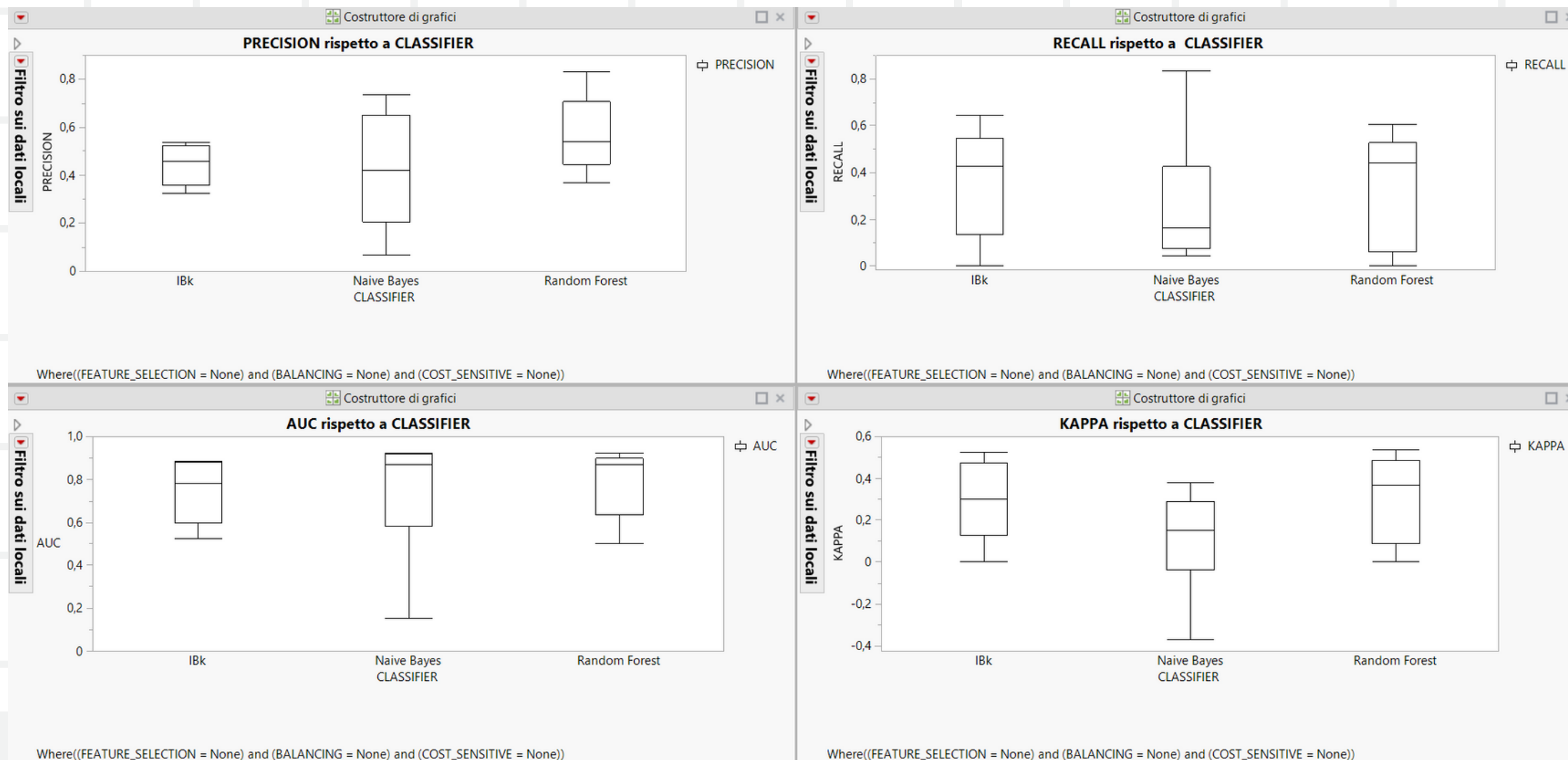
- Naive Bayes
- Random Forest
- IBk

TECNICHE

- Nessun filtro
- Feature selection (FS): Best First
- FS + sampling
- FS + sensitive learning
 $CFN = 10 * CFP$

RISULTATI

BOOKKEEPER SENZA FILTRI SUI CLASSIFICATORI

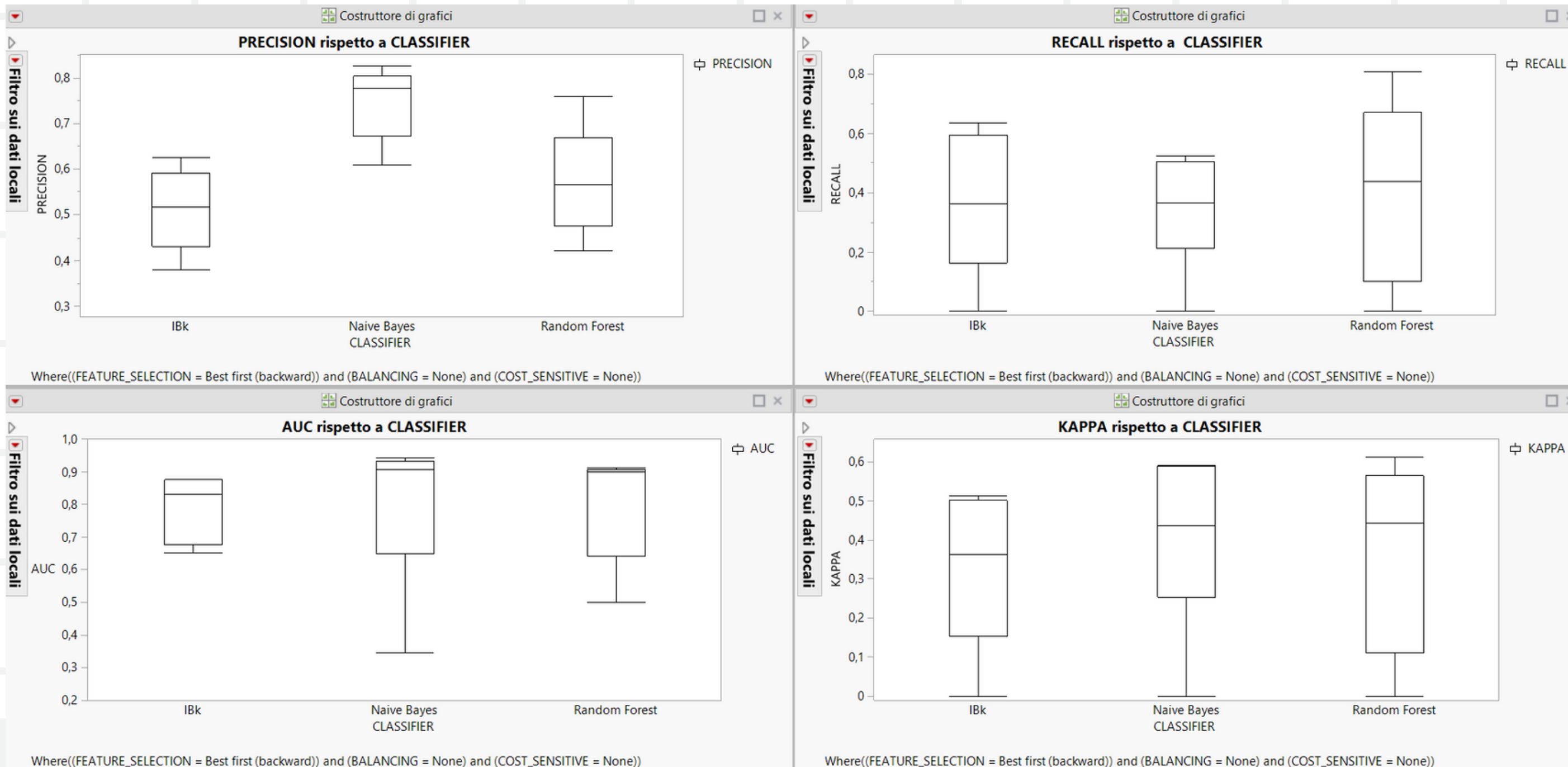


Random Forest
in media ottiene
risultati migliori

RISULTATI

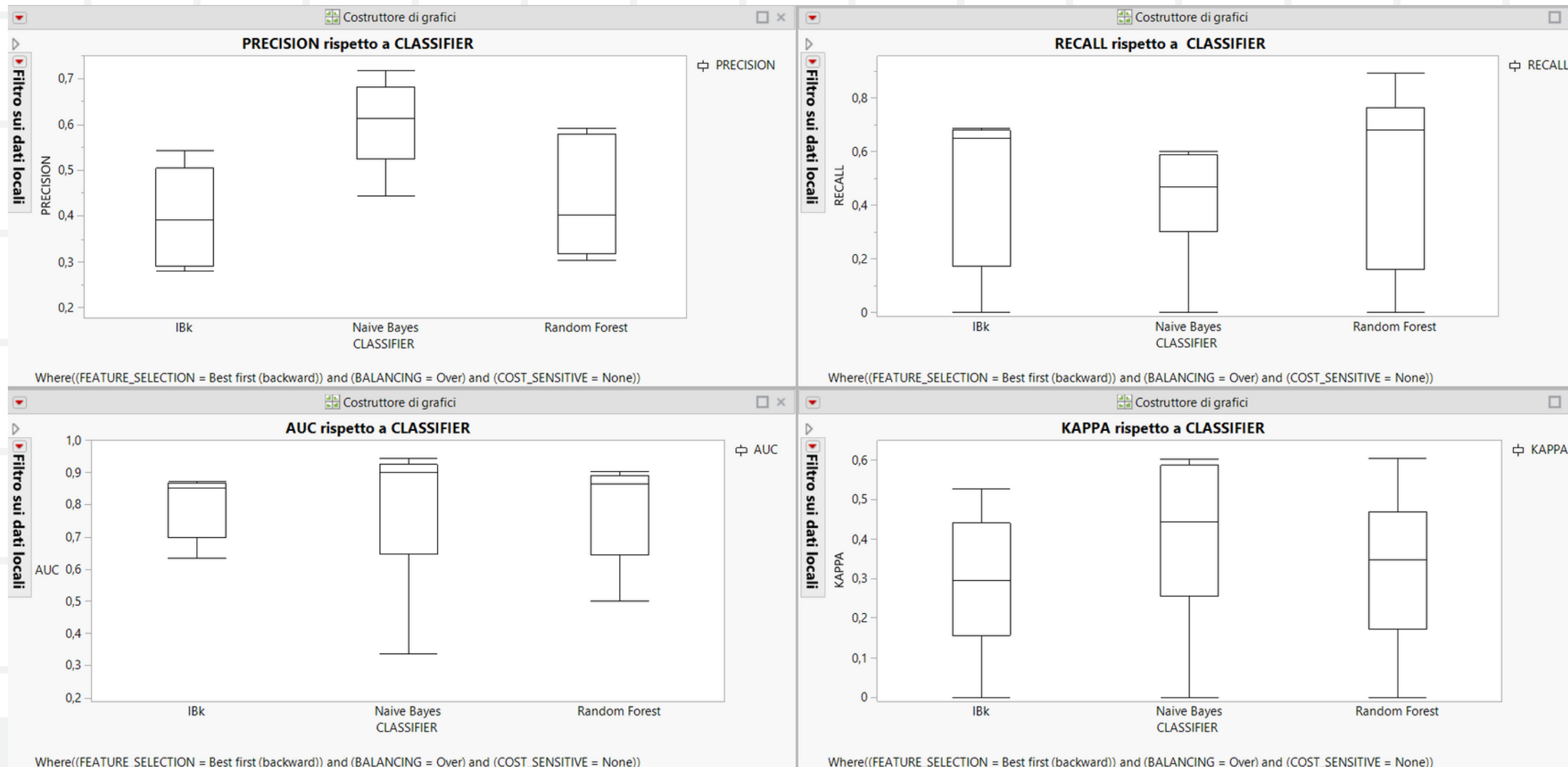
BOOKKEEPER FEATURE SELECTION (BEST FIRST)

Naive Bayes
raggiunge
ottimi valori di
precision.
Random
Forest, invece
prevale ancora
per quanto
riguarda la
recall.



RISULTATI

BOOKKEEPER FS E OVERSAMPLING



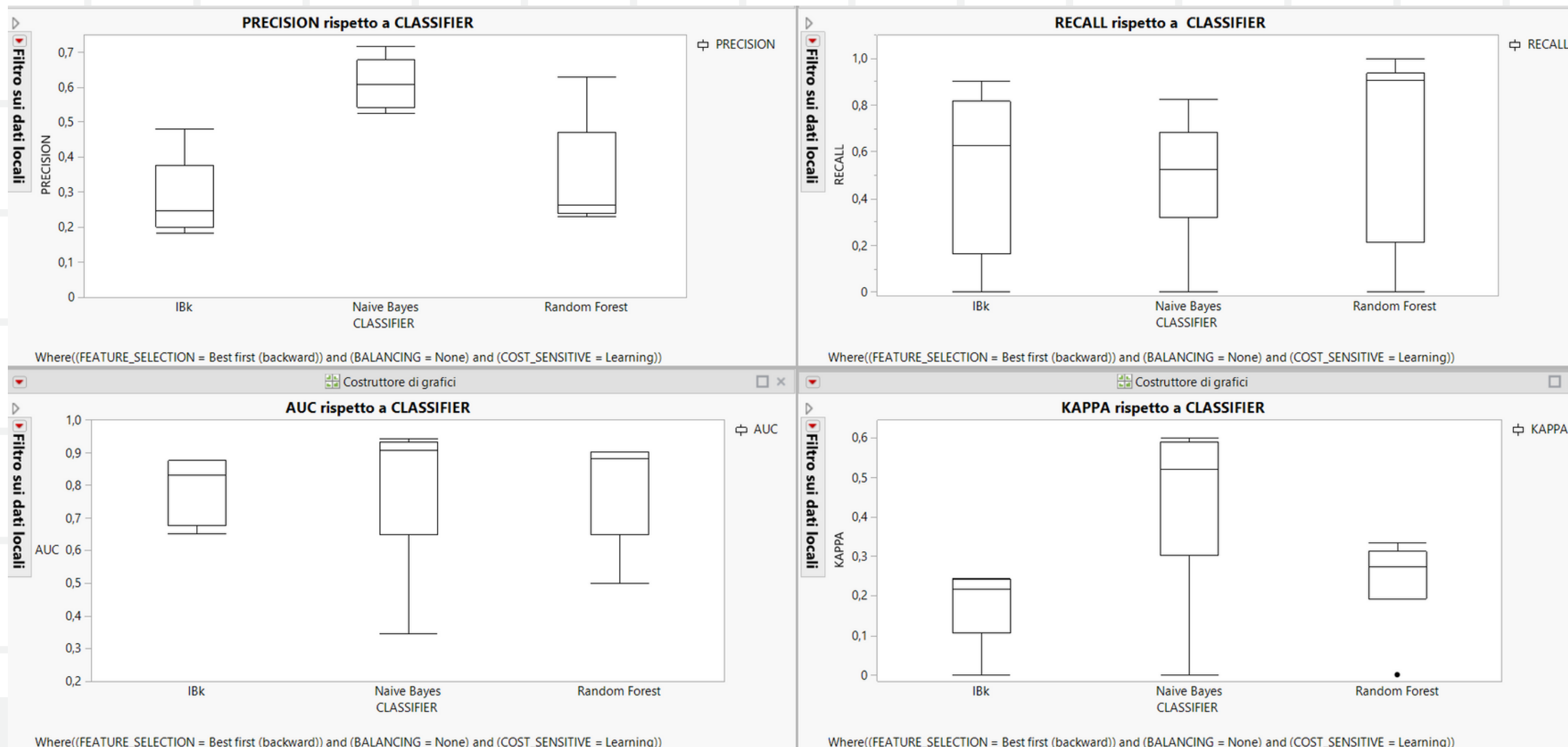
Non c'è un
classificatore
dominante
sugli altri.
**Random
Forest** mostra
un incremento
della recall a
discapito della
precision

RISULTATI

BOOKKEEPER FS E SENSITIVE LEARNING



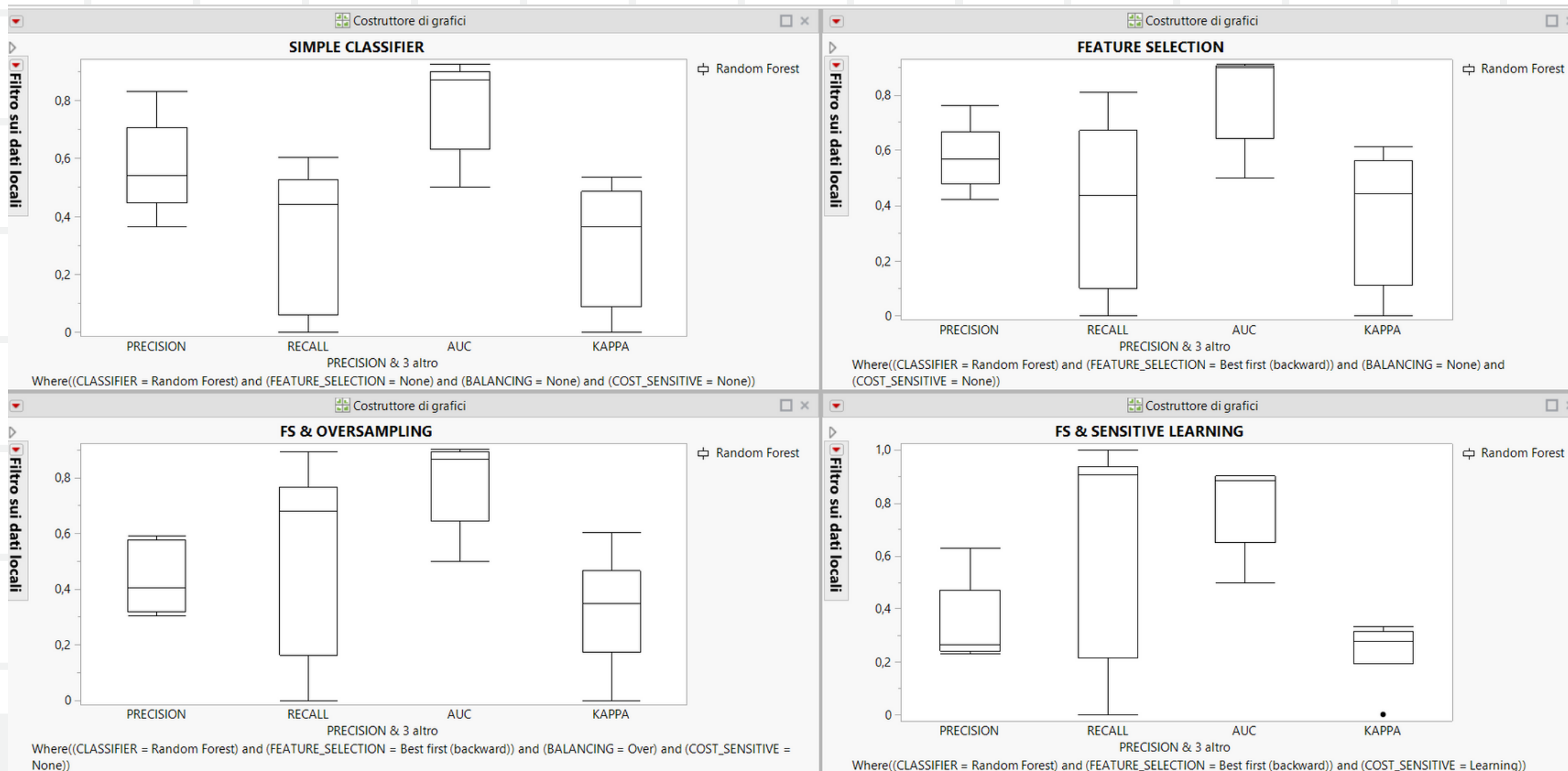
Random Forest è quello che reagisce meglio al sensitive learning ottenendo valori molto alti di recall.



RISULTATI

BOOKKEEPER, RANDOM FOREST A CONFRONTO.

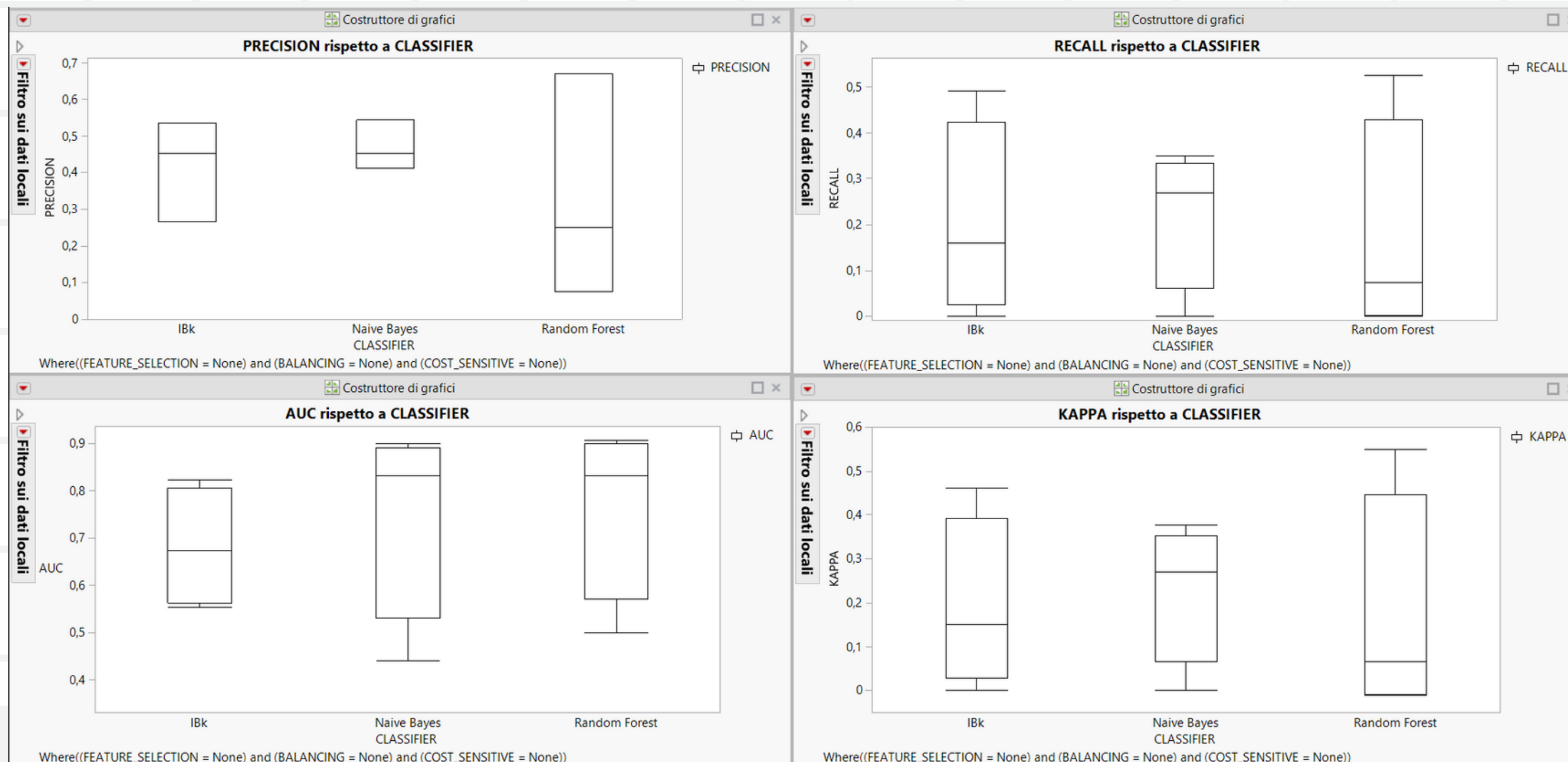
La configurazione più interessante è quella che fa uso di **sensitive learning**. Rispetto al classificatore di base raddoppia la recall media perdendo solo la metà della precision.



RISULTATI

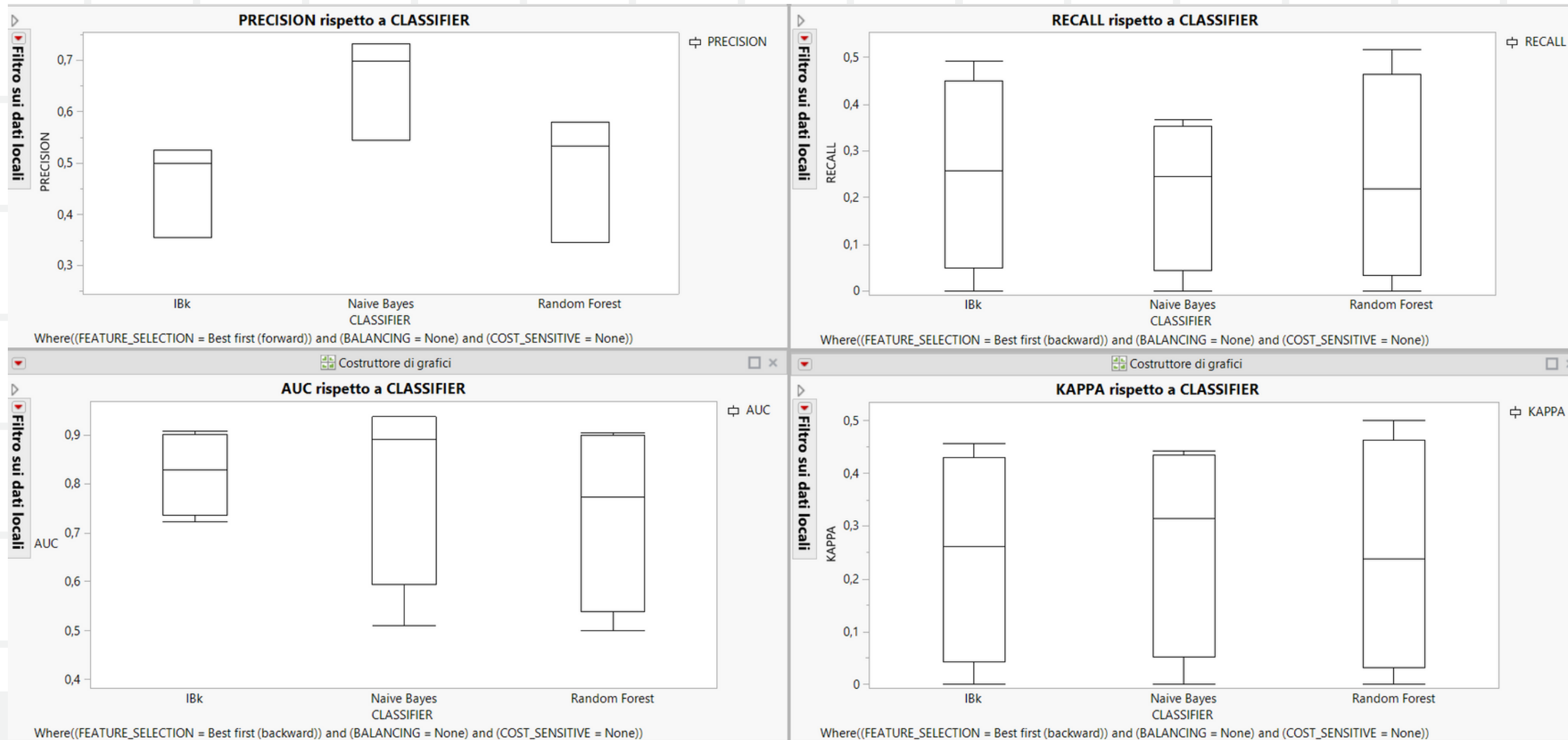
TAJO SENZA FILTRI SUI CLASSIFICATORI

Per quanto riguarda **Tajo**. Guardando i valori medi ottenuti dai classificatori base, **Naive Bayes** ha ottenuto i risultati migliori.



RISULTATI

TAJO FEATURE SELECTION (BEST FIRST)

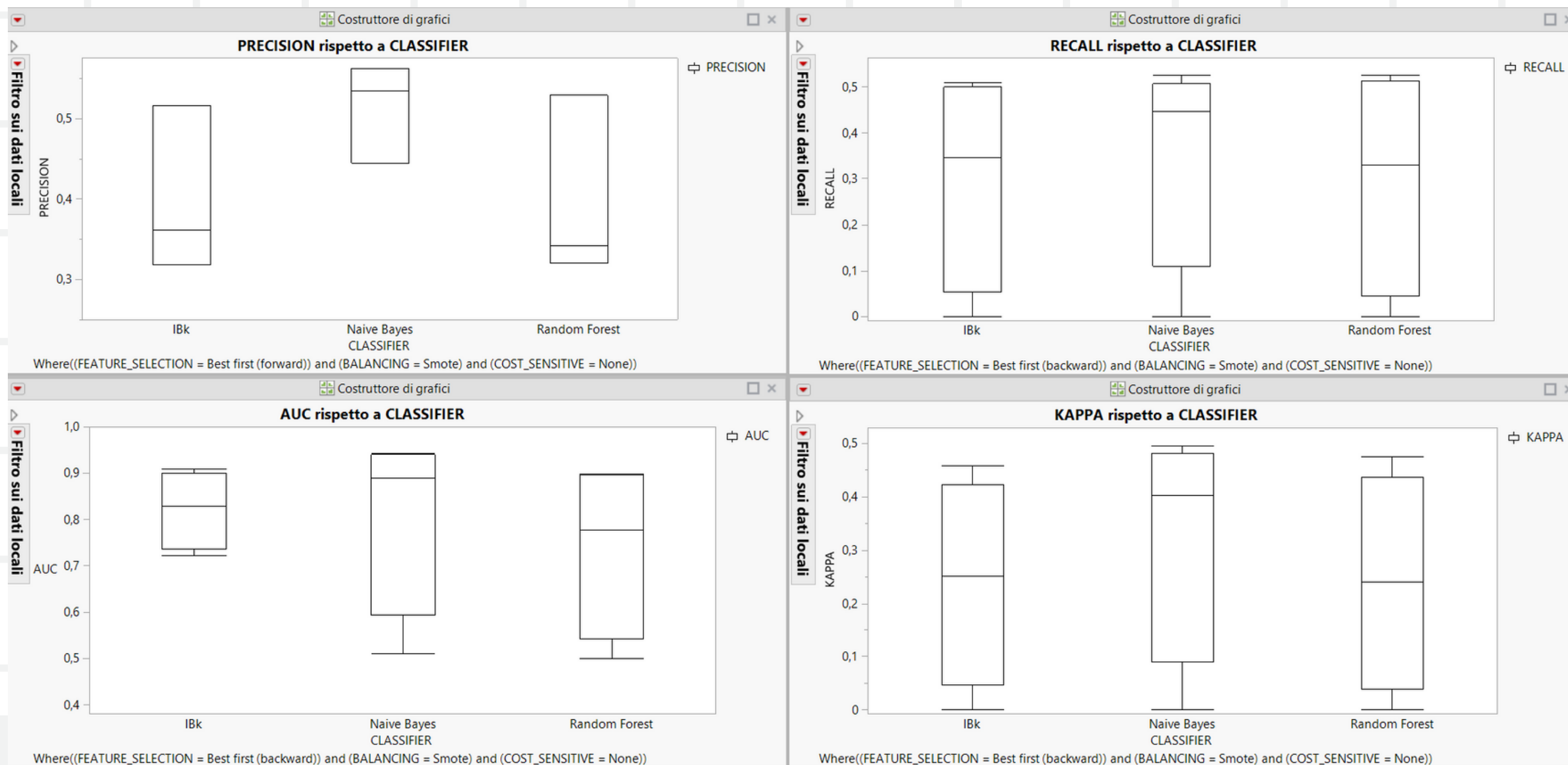


Guardando i valori medi, **Naive Bayes** con best first ha ottenuto i risultati migliori.

RISULTATI

TAJO FS E SMOTE

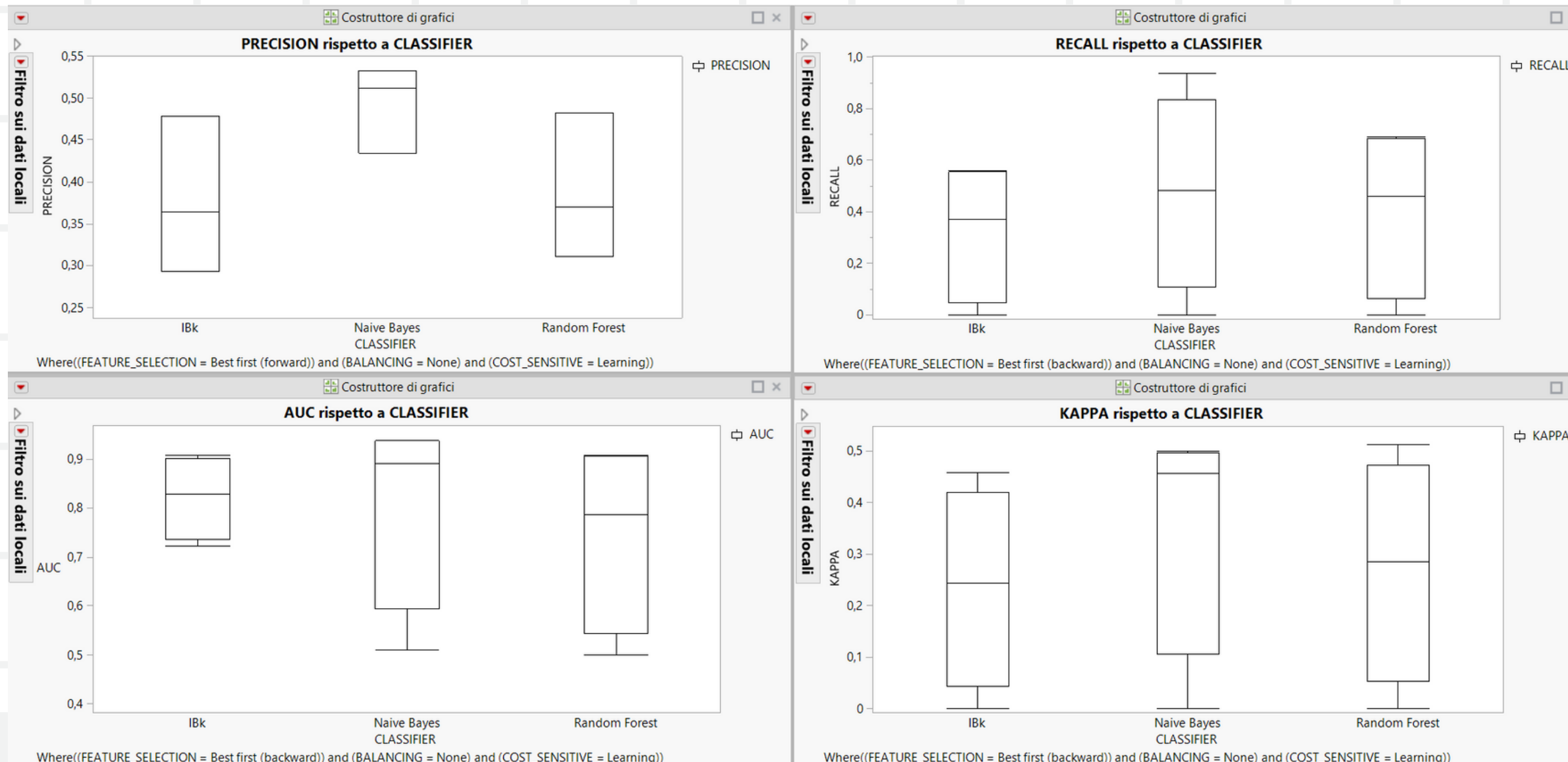
Applicando
smote come
tecnica di
sampling **Naive
Bayes** risulta
dominante
rispetto gli altri
classificatori.



RISULTATI

TAJO FS E SENSITIVE LEARNING

Ancora **Naive Bayes** risulta dominante rispetto gli altri classificatori. Tuttavia la configurazione non mostra un aumento così sostanzioso della recall.






CONCLUSIONI

BOOKKEEPER

Per il progetto BookKeeper, **Random Forest** si è rivelato il classificatore più sensibile ai filtri aggiunti. Ha raggiunto valori di recall via via più grandi, fino all'apice nella configurazione con feature selection e sensitive learning. Comportamento desiderato visto la maggiore gravità di un **falso negativo** quando si parla di bug.





CONCLUSIONI

TAJO

Per il progetto Tajo, **Naive Bayes** è stato il classificatore dominante in quasi tutte le configurazioni, tuttavia non è riuscito a raggiungere valori medi di recall superiori allo 0.5, mostrando quindi una difficoltà maggiore nell'evitare di predire falsi negativi.

LINK



🔍 **GITHUB**

<https://github.com/0334321-LSE/ISW2MLforSE>

🔍 **SONAR CLOUD**

https://sonarcloud.io/project/overview?id=0334321-LSE_ISW2InformationRetrieval

🔍 **ASSUNZIONI**

<https://github.com/0334321-LSE/ISW2MLforSE/blob/master/Assunzioni.txt>