Machine Learning for Software Engineering

Simone Bauco – 0324495 - Università di Roma Tor Vergata

a.a. 2022/2023

Agenda

- Introduzione
  - Contesto e obiettivo dello studio
- Metodologia
  - Progettazione e costruzione del dataset
  - Metriche considerate
  - Metodologia di valutazione dei classificatori
- Risultati della misurazione
- Analisi dei risultati
- Link utili

Contesto (1)

- La fase di testing di un progetto software ha l'obiettivo di scoprire eventuali **bug** del software, e correggerli.
- In generale, richiede la progettazione e implementazione di casi di test per ogni componente (e.g. classe) del sistema.
- **Problema**: come ridurre l'effort legato alla fase di testing, senza compromettere la sua efficacia?
- **Soluzione**: individuare le classi che *verosimilmente* contengono dei bug
  - Come?

Contesto (2)

- Per predire quali classi contengono probabilmente dei bug, si possono utilizzare meccanismi di **Machine Learning (ML)**.
- Ogni paradigma di ML ha bisogno di essere «allenato» su dei dati, per imparare come effettuare una determinata operazione (e.g. predizione) su dati simili.
- Quali dati utilizzare per allenare un modello di ML adatto al nostro scopo?
  - Possiamo utilizzare progetti già esistenti, che siano simili al progetto target
  - Per questi progetti, esiste una suddivisione in *release*
  - Per ogni release, è possibile conoscere quali classi hanno manifestato la presenza di bug
- Allenando un *classificatore* con questi dati, otteniamo un sistema in grado di stimare, con una certa accuratezza, quali classi contengono dei bug.

Obiettivo

- Valutare le prestazioni di 3 classificatori esistenti:
  - Naive Bayes
  - Random Forest
  - IBk
- **Task**: predizione della ***buggyness*** delle classi.
- I dati per allenare (*training set*) e per valutare (*testing set*) i classificatori verranno estratti dai progetti open-source Apache BookKeeper e Apache Storm.
- Verrà valutato l'impatto sulle prestazioni dei classificatori di tre tecniche:
  - Feature Selection
  - Sampling
  - Cost-Sensitive Classification

Metodologia: estrazione dei dati e individuazione delle classi buggy (1)

- Come ottenere dati relativi alla *buggyness* delle classi?
  - I due progetti considerati utilizzano il sistema di *issue tracking* **Jira**, che riporta informazioni riguardanti tutti i bug scoperti nel sistema e la loro risoluzione in dei **ticket**.
- Ogni bug ha un ciclo di vita

Injected Version (IV)
release in cui il bug è stato introdotto nel sistema

Opening Version (OV)
release in cui è stato aperto un ticket relativo al bug (i.e., in cui è stato scoperto il bug)

Fixed Version (FV)
release in cui è stato risolto il bug

- Le classi in cui è presente il bug sono *buggy* dalla release IV alla release FV.
- L'insieme di release che va da IV ad OV è detto **Affected Versions (AV)**

Metodologia: estrazione dei dati e individuazione delle classi buggy (2)

- Ogni classe che è associata ad un ticket, deve essere **etichettata** come buggy dall'introduzione del bug (IV) fino al momento in cui il bug viene risolto (FV).
- I valori di OV e di FV sono sempre riportati dai ticket Jira, dal momento che indicano la creazione e la risoluzione del ticket stesso.
- Problema: non tutti i ticket Jira contengono l'indicazione relativa alla IV, che è però necessaria al labeling delle classi.

Metodologia: estrazione dei dati e individuazione delle classi buggy (3)

- Soluzione: si può assumere che esista una proporzionalità fissa tra l'intervallo di tempo (IV, FV) e l'intervallo di tempo (OV, FV).
- La tecnica che calcola il valore di IV, a partire da questa proporzionalità, è detta **Proportion**:
  - Viene definita una costante di proporzionalità p:

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

    per tutti i ticket per cui IV è nota.
  - Per tutti gli altri ticket, si calcola il valore di IV nel seguente modo:

$$IV = FV - (FV - OV) \cdot p$$

Metodologia: estrazione dei dati e individuazione delle classi buggy (4)

- Esistono diverse varianti per il calcolo di p
- Per questo progetto, è stata scelta la variante **cold-start**:
  - Viene calcolato, per un certo numero k di progetti simili, un valore $p_i, \forall i = 1, \dots, k$, che rappresenta il valore *medio* di proportion per ogni progetto considerato (ottenuto dalla formula precedente)
  - Il valore di p utilizzato nel progetto di interesse è la mediana di tutti i p_k.

- Nel nostro caso, i progetti considerati per il calcolo di p_k sono alcuni progetti sviluppati dalla stessa Apache:
  - Avro, OpenJPA, Zookeeper, Syncope, Tajo, Accumulo, Kafka (se il progetto di interesse è BookKeeper, viene incluso anche Storm nel calcolo di p, e viceversa).

Metodologia: estrazione dei dati e individuazione delle classi buggy (4)

- Una volta ottenuto il valore di IV per ogni ticket, è possibile effettuare il labeling delle classi, attraverso l'analisi dei commit associati ai ticket.
- Per ogni ticket:
  - Vengono individuati i commit associati al ticket
  - Vengono individuate le classi modificate in ogni commit associato al ticket
  - Ciascuna delle classi modificate è buggy se appartiene ad una release che è compresa nell'intervallo [IV, FV)

Assunzioni

- Nell'implementazione del meccanismo di calcolo del valore di proportion e di labeling delle classi, sono state effettuate alcune assunzioni:
  - Per evitare il fenomeno dello *snoring* (bug non ancora scoperti), è stata scartata la seconda metà delle release, dal momento che è molto probabile che tutti i bug presenti nella prima metà siano stati scoperti e, spesso, risolti.
  - Per quanto riguarda i ticket che riportano il valore di AV, sono stati considerati solamente quelli con AV consistente (i.e., IV (prima release di AV) $\leq$ OV)
  - Nel calcolo di proportion, se OV = FV, è stato considerato FV – OV = 1, per due motivi:
    - Evitare denominatori pari a 0;
    - Per la assunzione realistica che, anche se un bug è stato scoperto e risolto nella stessa release, non è detto che sia stato introdotto nel sistema nella stessa release, ma si può assumere che sia stato introdotto *almeno* nella release precedente.

Metodologia: costruzione del dataset

- Oltre alla buggyness, sono state considerate altre metriche (i.e. caratteristiche misurabili della classe), al fine di generare buoni dataset per il training dei classificatori.
- L'introduzione di metriche aggiuntive mira, infatti, a fornire più informazioni significative riguardo la classe ai classificatori, che saranno più accurati.
- Il dataset così ottenuto contiene, come **feature** (o **attributi**), informazioni riguardanti la classe (i.e., nome, release di appartenenza), tutte le metriche aggiuntive e il valore di buggyness, che sarà il valore da predire.
- Nota: tutte le metriche aggiuntive si riferiscono alla release di appartenenza della classe.

Metriche considerate

Nome	Descrizione
Size	Dimensione (in LOC) della classe
NAuth	Numero di autori (persone che hanno modificato la classe)
Fan-out	Numero di altre classi dalle quali la classe dipende
NR	Numero di commit associati alla classe
LOC added	Somma delle LOC aggiunte alla classe in tutti i commit associati
Max LOC added	Massimo numero di LOC aggiunte alla classe in un commit associato
Average LOC added	Numero medio di LOC aggiunte alla classe nei commit associati
Churn	Somma, per tutti i commit associati, di $\lvert LOC\ added\ -\ LOC\ deleted \rvert$
Max Churn	Massimo valore di Churn tra tutti i commit associati
Average Churn	Valore medio di Churn tra tutti i commit associati
Time span	Lunghezza dell'intervallo di tempo che va dal primo commit associato alla classe fino all'ultimo

Metodologia: valutazione dei classificatori (1)

- Per la valutazione dei classificatori, la tecnica scelta è **walk-forward**:

Run \ Release	1	2	3	4	5
1					
2					
3					
4					
5					

La valutazione finale si ottiene dalla media tra tutte le iterazioni

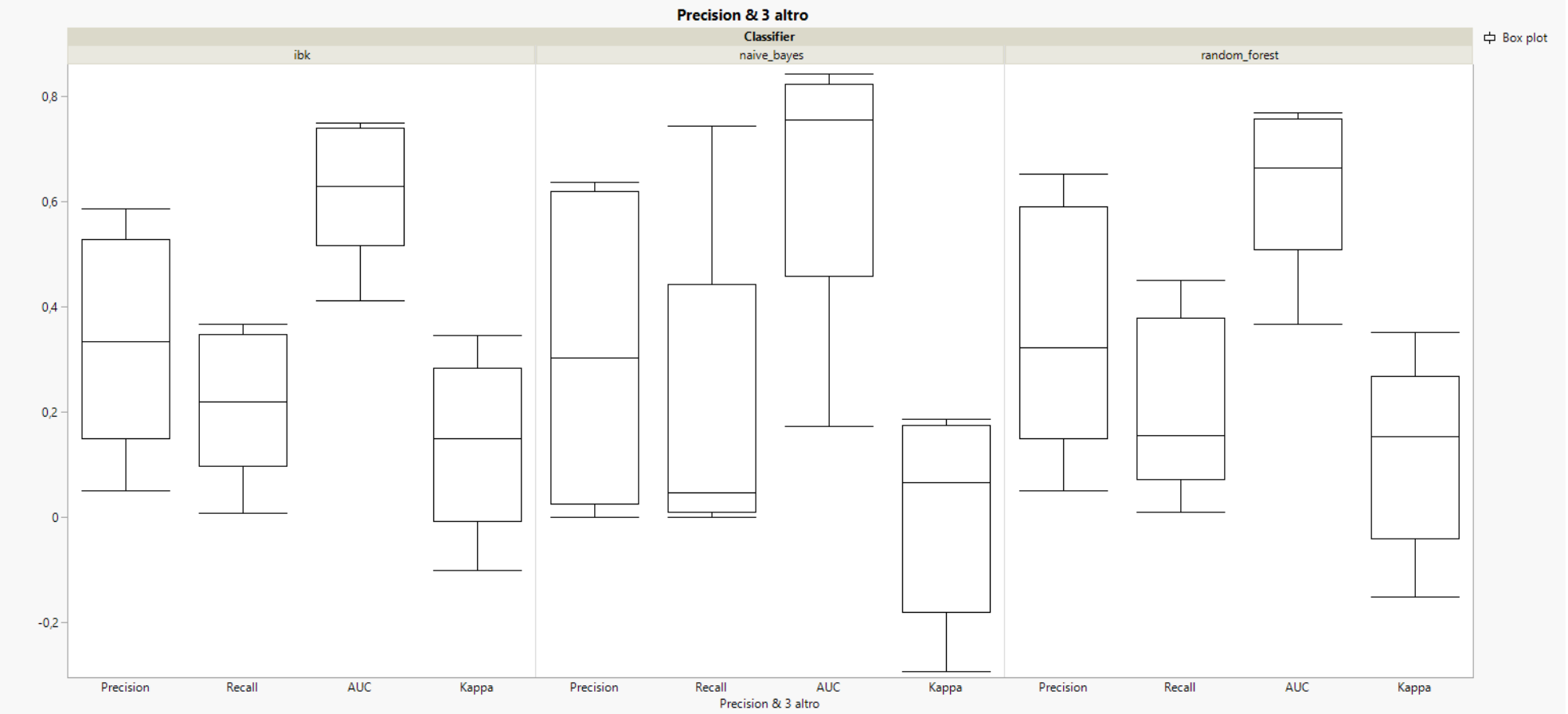
Metodologia: valutazione dei classificatori (2)

- Walk-forward è una tecnica di validazione ***time-series***, tiene conto perciò dell'ordine temporale dei dati.
- Perciò, il training set viene ottenuto considerando solo i dati disponibili fino alla release considerata:
  - e.g. considerando l'iterazione 3, il training set - formato dai dati relativi alle release 1 e 2 – viene ottenuto considerando i dati disponibili fino alla release 2, ignorando tutti i dati successivi.
- Ciò riguarda, in particolare, il labeling delle classi.
- Di conseguenza, il training set presenta snoring, dal momento che alcuni bug saranno stati scoperti solo in release successive.
- Il testing set, invece, è ottenuto considerando tutti i dati a disposizione, quindi non presenta snoring.

Classificatori e tecniche utilizzate

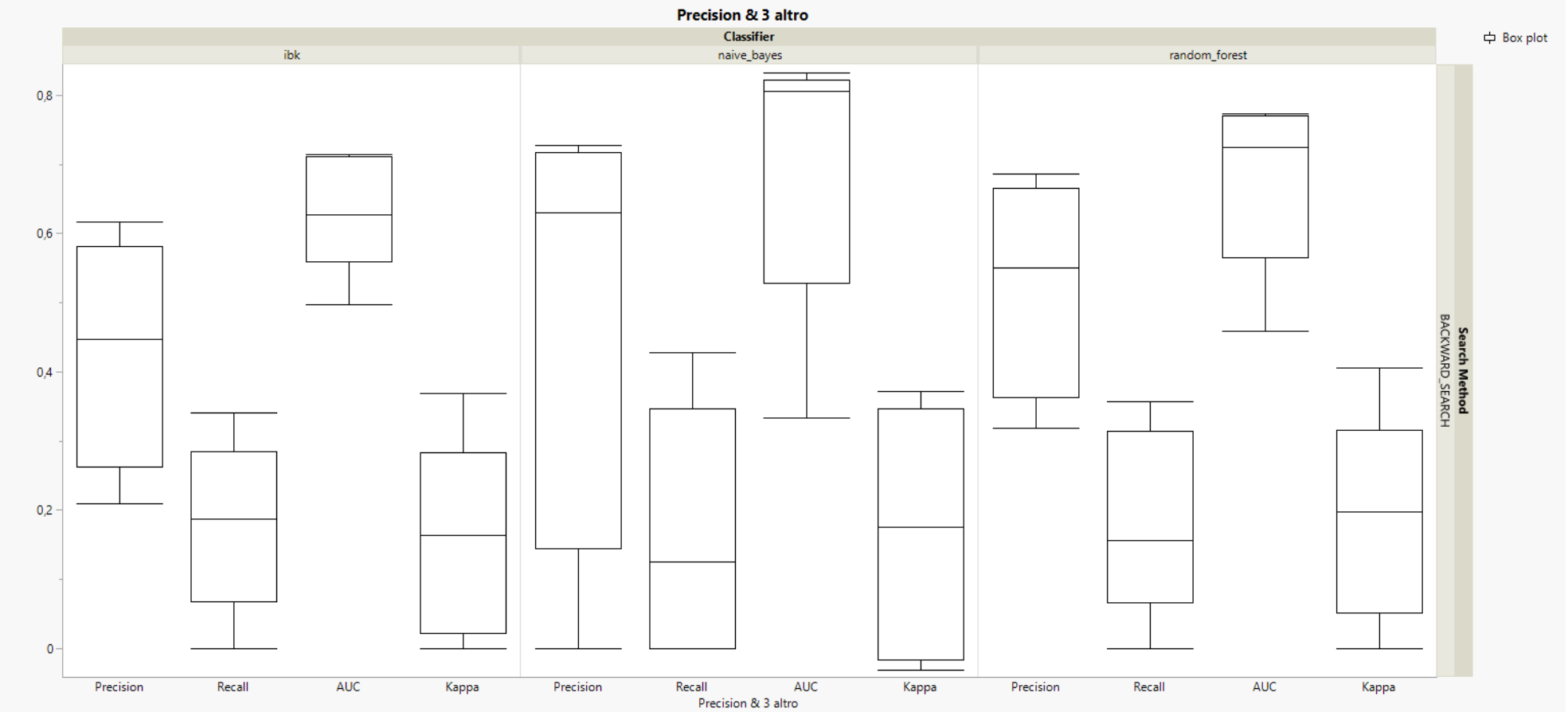
- Come già visto, i classificatori utilizzati sono Naive Bayes, Random Forest e IBk, utilizzati con e senza Feature Selection, Sampling e Cost-Sensitive Classification.
- Verranno comparate diverse configurazioni, tra cui:
  - Senza nessun filtro
  - Con Feature Selection (best first: forward search, backward search)
  - Con Feature Selection (best first, forward search) + Sampling (oversampling)
  - Con Feature Selection (best first, forward search) + Sampling (undersampling)
  - Con Feature Selection (best first, forward search) + Cost-Sensitive Classification (CFN = 10*CFP)
- Le varie configurazioni sono ripetute per ognuno dei classificatori.

Risultati: BookKeeper, senza filtri



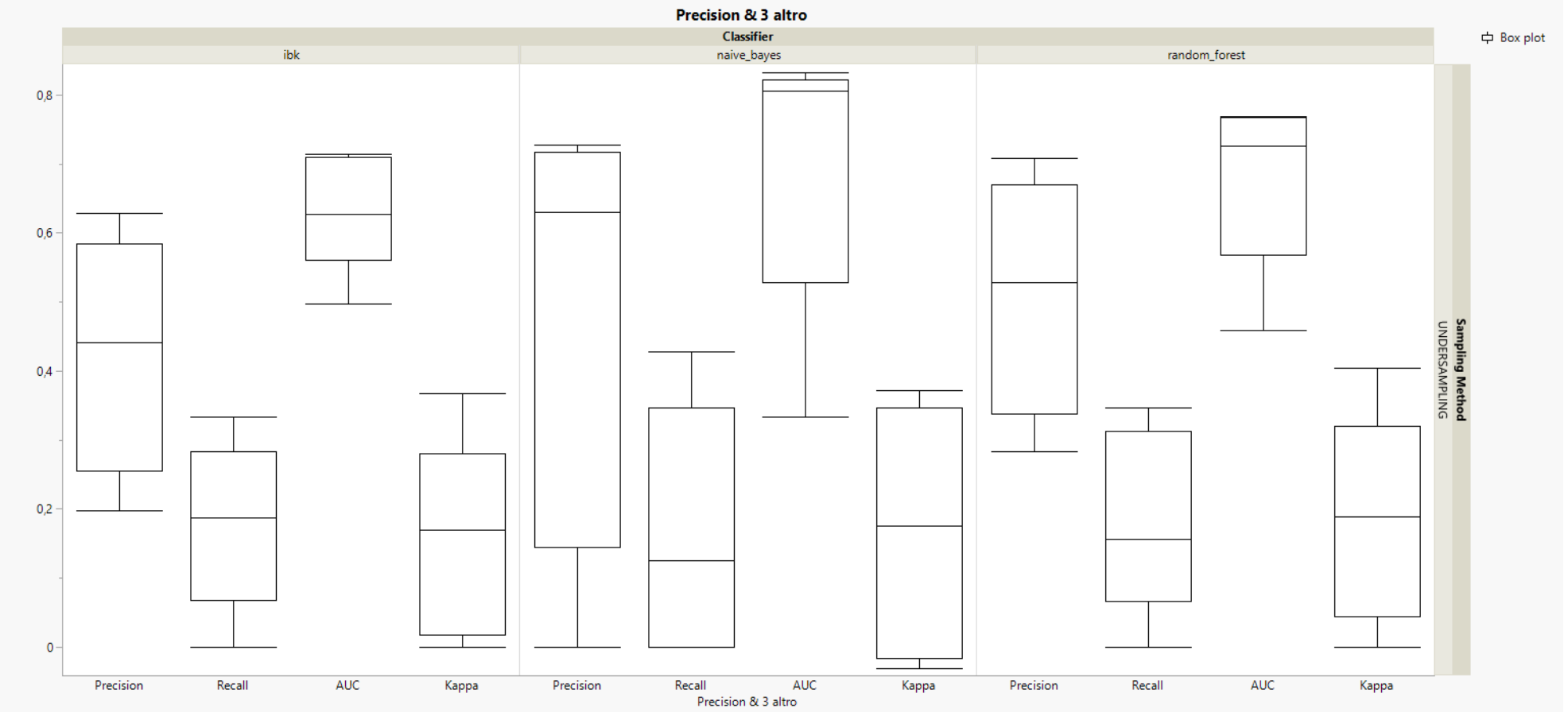
In media, il classificatore dominante, sia per precision che per recall, è IBk.

Risultati: BookKeeper, con FS (best first backward search)



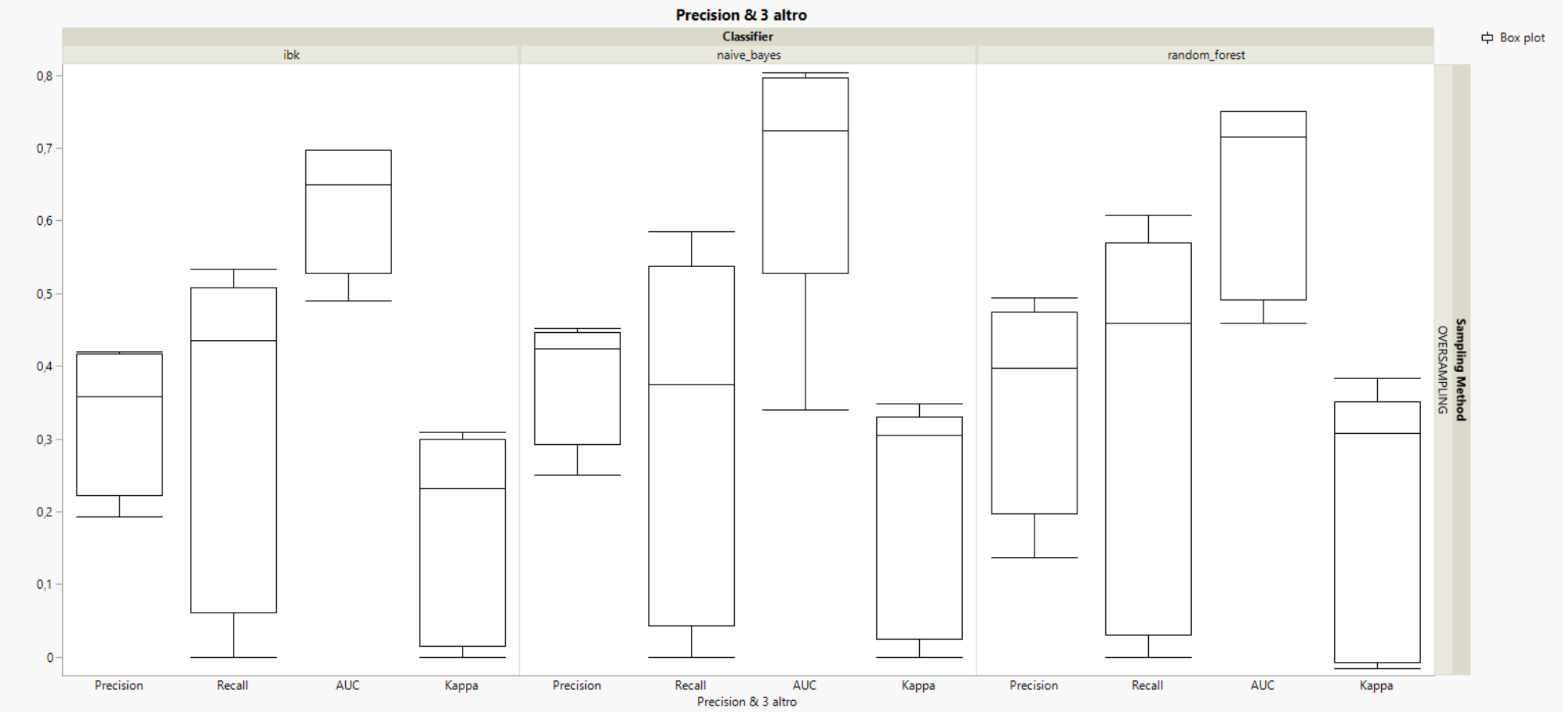
In media, il classificatore dominante per precision è Naive Bayes, mentre per recall è IBk.

Risultati: BookKeeper, con FS (best first forward search) + undersampling



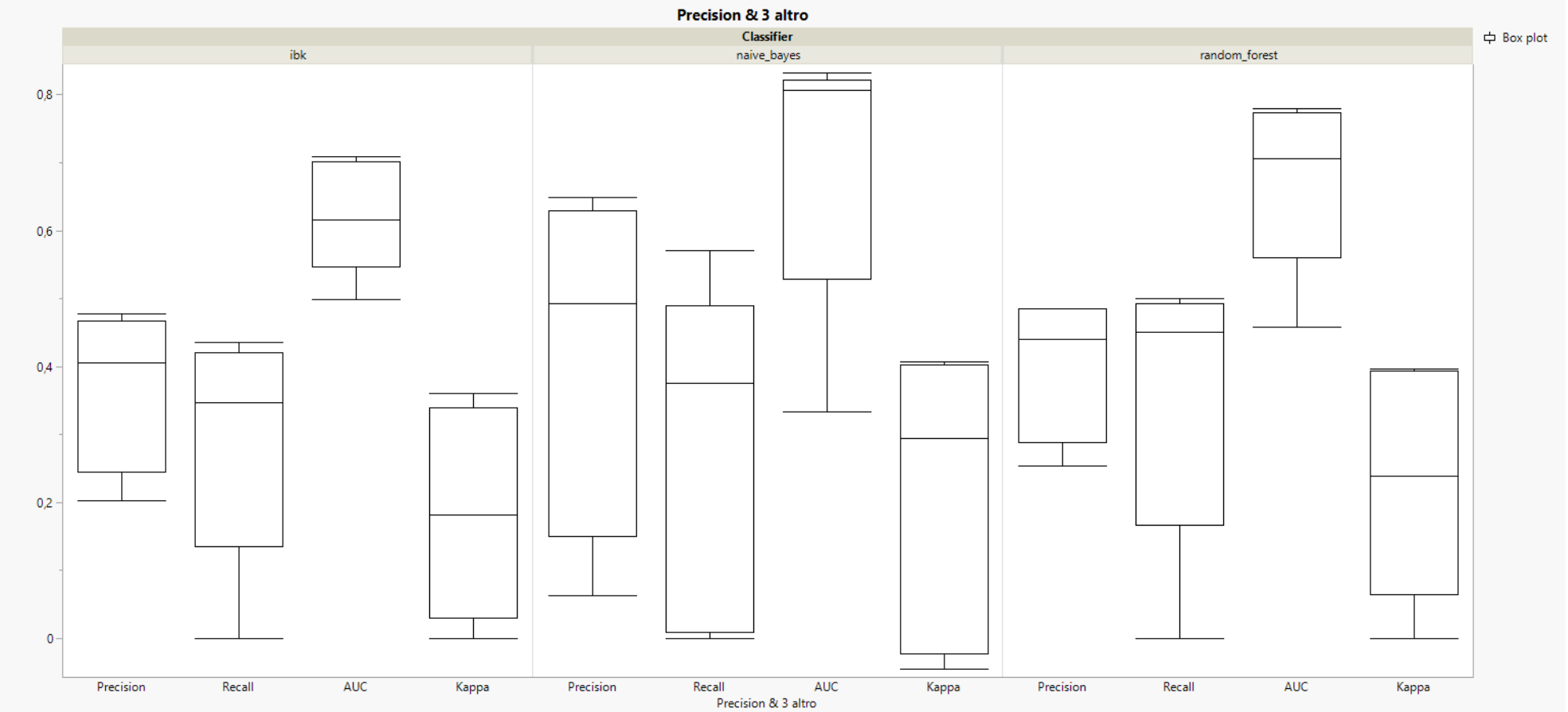
In media, il classificatore dominante per precision è Naive Bayes, mentre per recall è IBk.

Risultati: BookKeeper, con FS (best first forward search) + oversampling



In media, il classificatore dominante per precision è Naive Bayes, mentre per recall è Random Forest.

Risultati: BookKeeper, con FS (best first forward search) + CSC (CFN = 10*CFP)

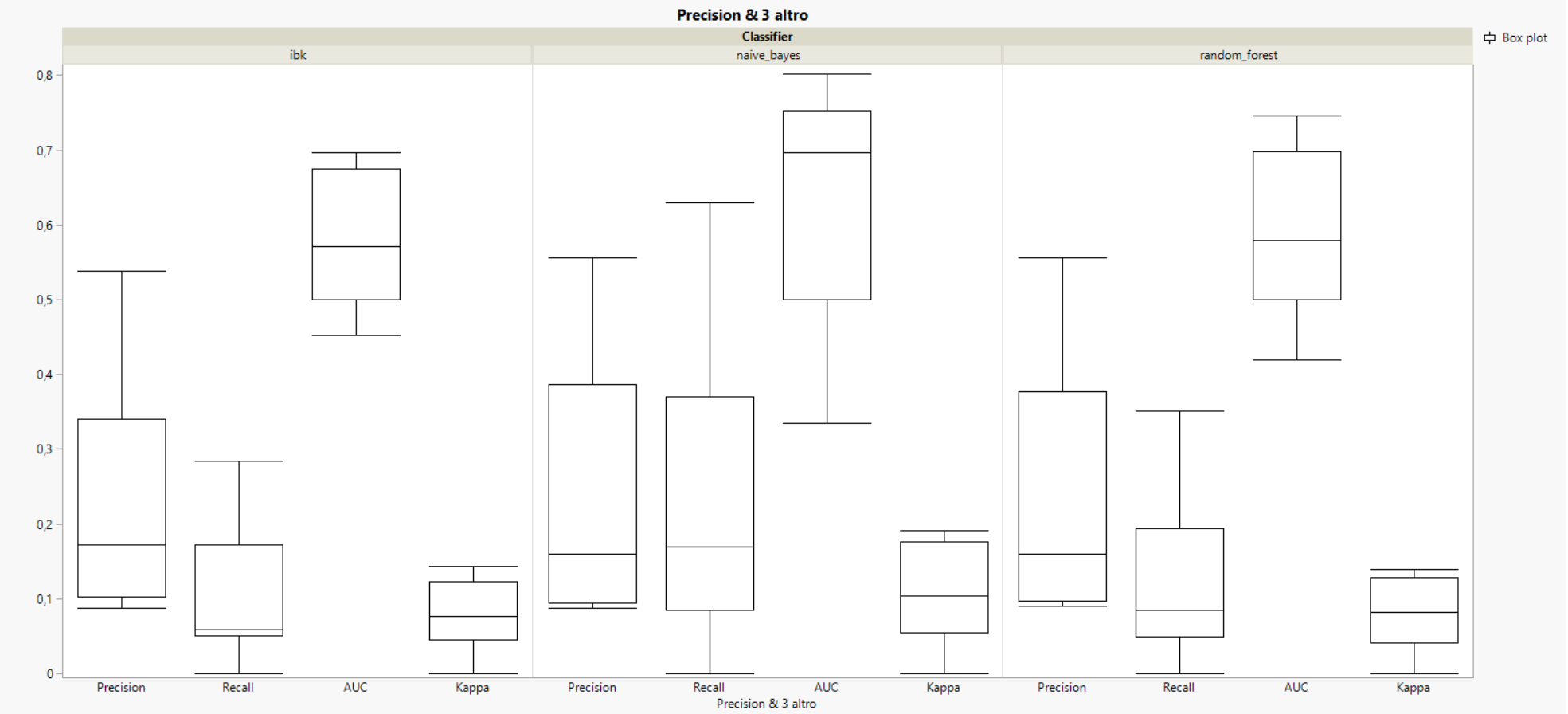


In media, il classificatore dominante per precision è Naive Bayes, mentre per recall è Random Forest.

Analisi dei risultati: BookKeeper

- Consideriamo il valore mediano massimo di ognuna delle 4 metriche di valutazione:
  - Precision: Naive Bayes + FS (backward search) e Naive Bayes + FS + undersampling
  - Recall: Random Forest + FS (forward search) + oversampling
  - AUC: Naive Bayes + FS (backward search) ma anche con undersampling e CSC
  - Kappa: Random Forest + FS (forward search) + oversampling
- Non esiste dunque un classificatore dominante rispetto agli altri.
- In questo caso, è di interesse massimizzare la recall (i.e., minimizzare i falsi negativi), quindi una buona configurazione è Random Forest con FS (forward search) e oversampling, per cui si ha il valore mediano massimo di recall e kappa, ma anche buoni valori per le altre due metriche.

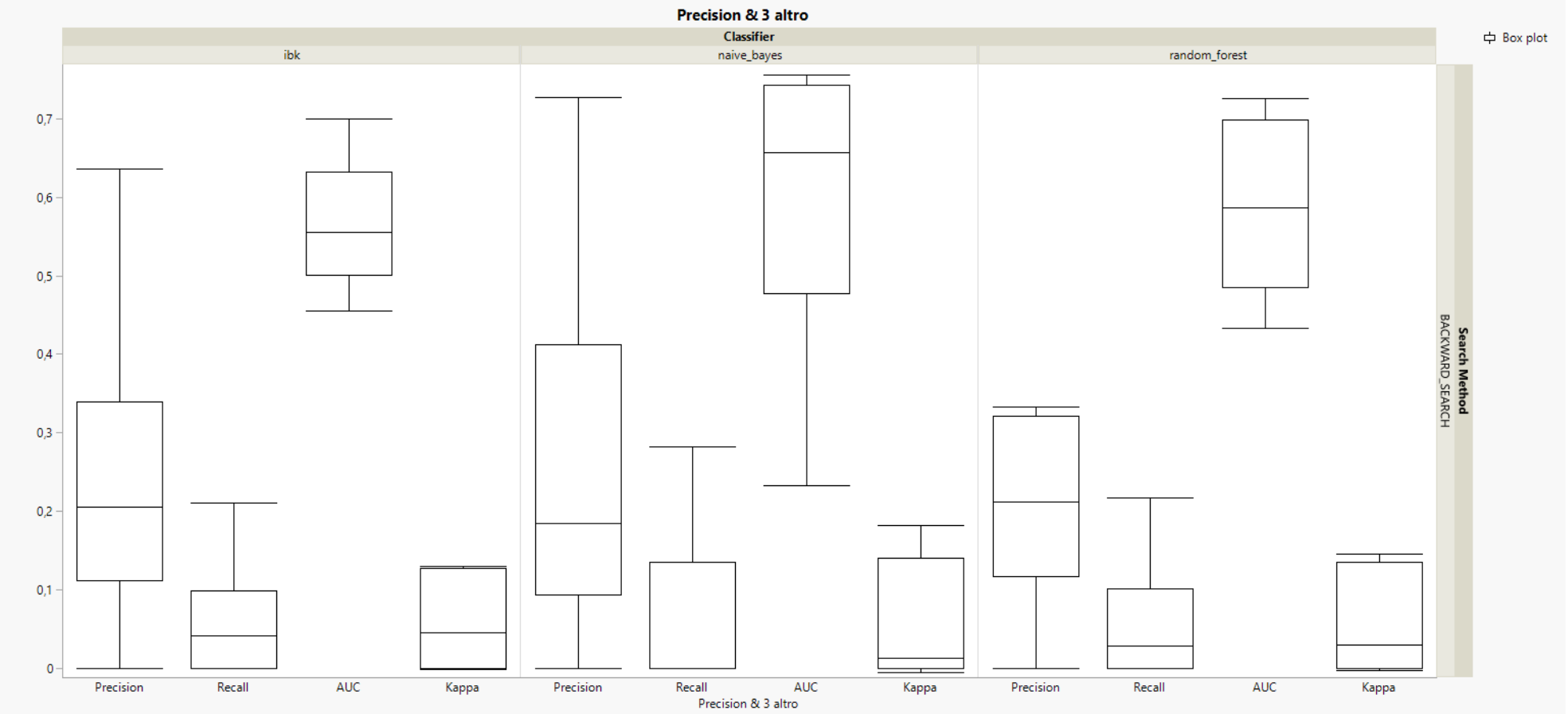
Risultati: Storm, senza filtri



I classificatori sono circa equivalenti per la precision, mentre Naive Bayes è dominante per recall

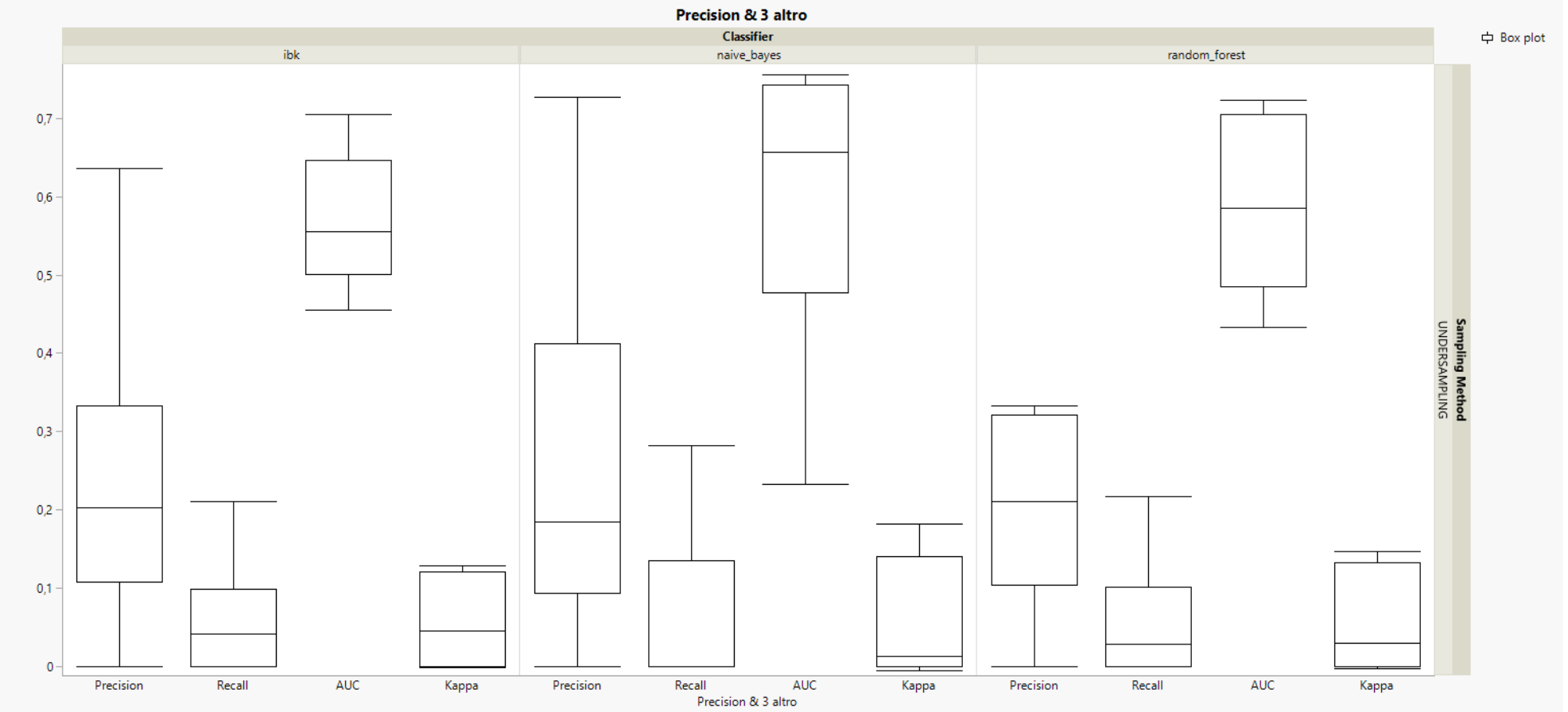
Risultati: Storm, con FS (best first, forward search)

Nota: Forward search porta agli stessi risultati



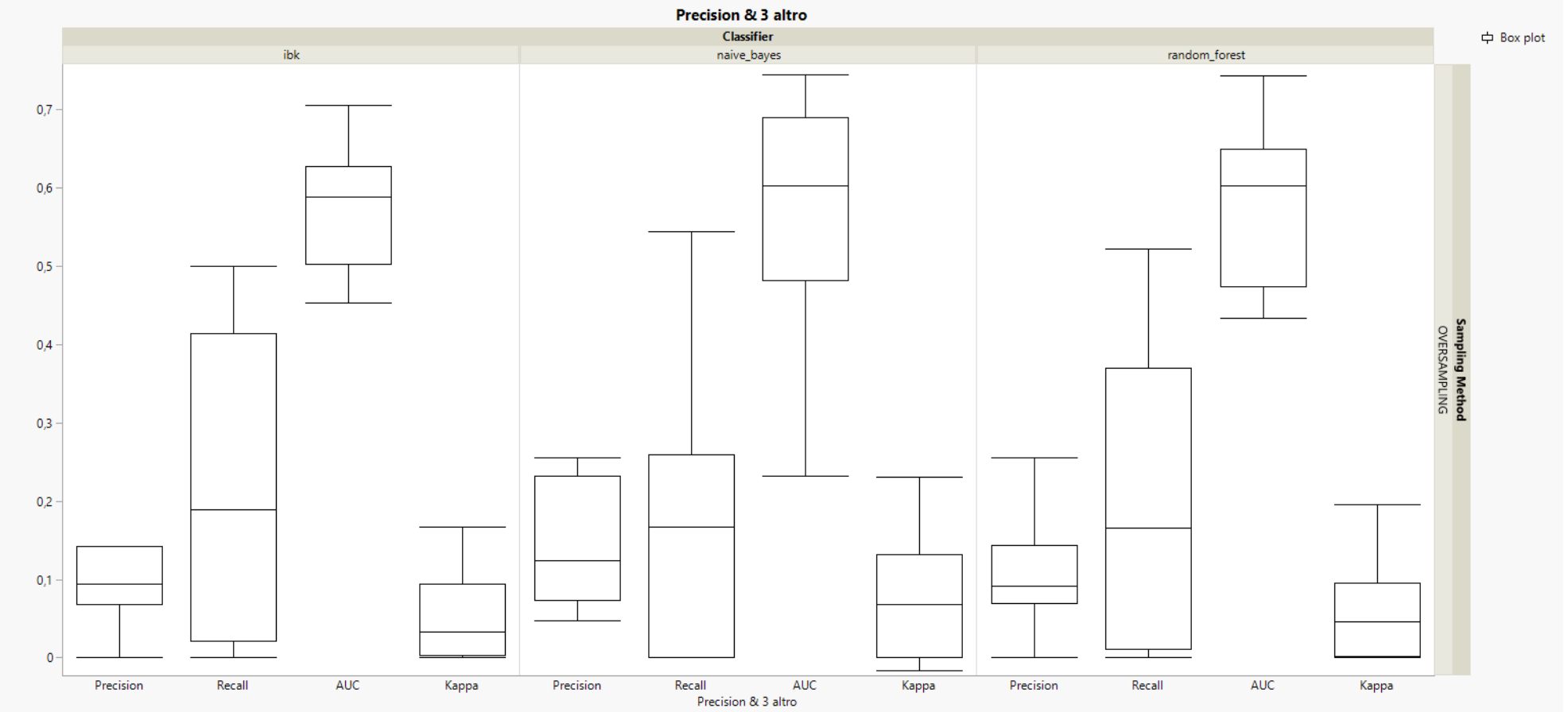
I classificatori sono circa equivalenti per Precision. Naive Bayes è dominante per recall.

Risultati: Storm, con FS (best first, forward search) + undersampling



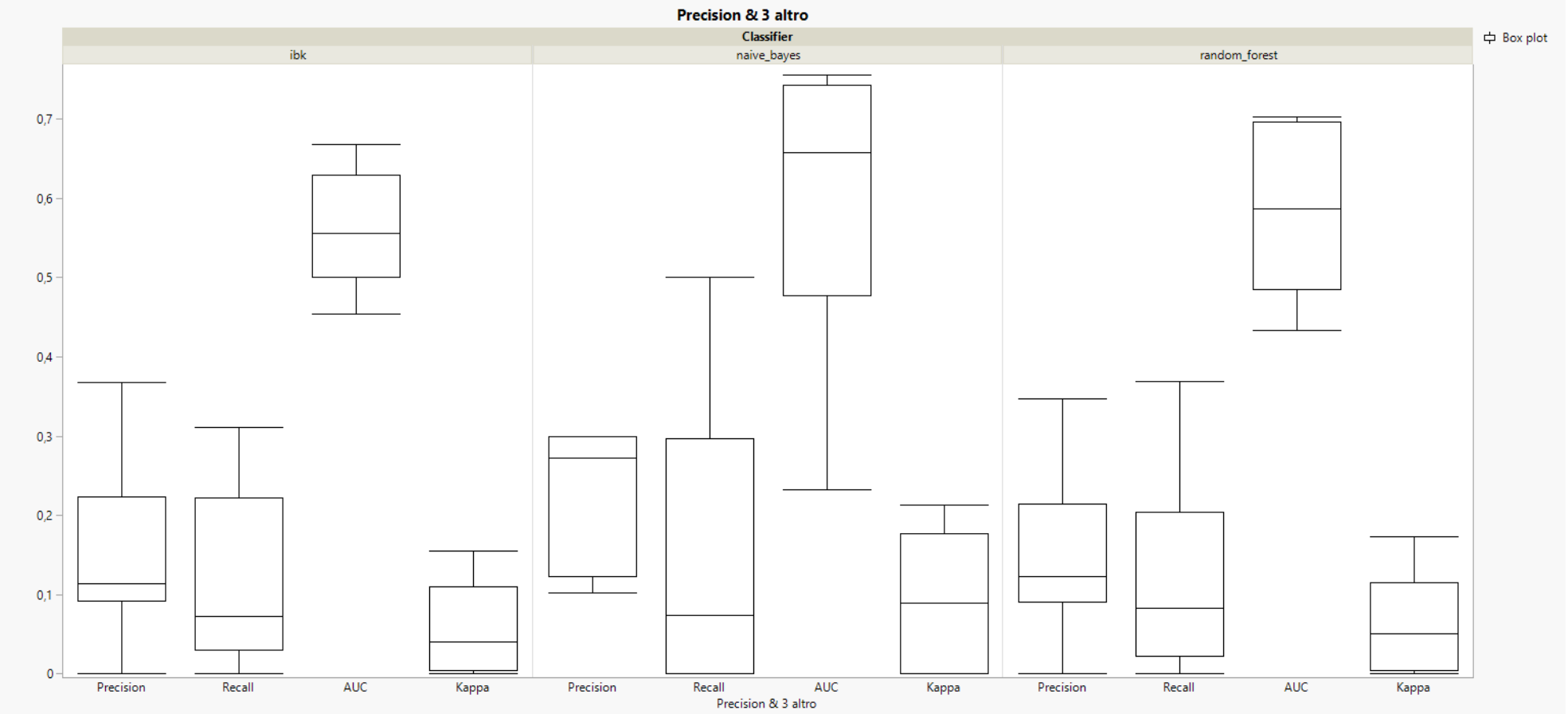
I classificatori sono circa equivalenti per Precision. Naive Bayes è dominante per recall.

Risultati: Storm, con FS (best first, forward search) + oversampling



Naive Bayes è dominante per precision. Hanno un comportamento simile per quanto riguarda recall, con IBk che raggiunge, però, un valore mediano più alto.

Risultati: Storm, con FS (best first, forward search) + CSC (CFN = 10*CFP)



Naive Bayes è dominante per precision. Hanno un comportamento simile (nelle mediane) per quanto riguarda recall, con Naive Bayes che raggiunge un valore massimo più alto.

Analisi dei risultati: Storm

- Consideriamo il valore mediano massimo di ognuna delle 4 metriche di valutazione:
  - Precision: Naive Bayes con FS (forward search) + CSC (il valore mediano è più alto, anche se decrescono i valori massimi)
  - Recall: IBk con FS (forward search) + oversampling
  - AUC: Naive Bayes, senza filtri
  - Kappa: i classificatori nelle varie configurazioni hanno comportamenti equivalenti, il valore mediano massimo si ottiene con Naive Bayes, senza applicare filtri.
- In questo caso, la scelta migliore sembra essere Naive Bayes con FS + oversampling che, pur non raggiungendo il valore mediano massimo di recall, raggiunge un valore simile, mantenendo però piuttosto alte le altre metriche, rispetto agli altri classificatori.

Link utili

- Github repository: https://github.com/simoneb00/ISW2
- SonarCloud: https://sonarcloud.io/summary/overall?id=simoneb00_ISW2