Machine Learning for Software Engineering

FEDERICA VILLANI - 0309716

Agenda

- Introduzione ed obiettivi
- Progettazione:
  - Individuazione classi buggy
  - Raccolta dati e costruzione dataset
  - Valutazione dei classificatori
  - Tecniche di utilizzo per i classificatori
- Risultati ottenuti
- Considerazioni

Introduzione e obiettivi

- Una delle attività più importanti nel processo di sviluppo di un software è il testing, che ha come obiettivo quello di evidenziare comportamenti inaspettati del sistema dovuti alla presenza di bug.
- L'attività di testing, se applicata a progetti software di grandi dimensioni, risulta molto onerosa sia in termini di tempo sia in termini economici.
- E', quindi, impossibile sottoporre al testing l'intero progetto ed è necessario attuare una scelta su quali classi testare per prima.

Obiettivo 1: Definire delle tecniche che permettano di stabilire quali classi hanno più probabilità di avere dei bug, in modo da concentrare il test su di esse.

Introduzione e obiettivi

- Per raggiungere l'obiettivo 1, è utile applicare il **Machine Learning**, per poter ottenere delle predizioni sulla bugginess delle classi.
- Il requisito principale per poter utilizzare un modello predittivo è la raccolta dei **dati passati**, che permettono di effettuare l'addestramento del modello e di valutarne le prestazioni.
- Le misurazioni effettuate sul codice e sulla sua evoluzione, quindi, vengono date in pasto ad un **classificatore**, che, tramite anche il possibile utilizzo delle tecniche di **feature selection**, **sampling** e **cost sensitivity**, etichetta le classi come buggy o meno.

Obiettivo 2: individuare quale classificatore effettua le predizioni migliori per il dominio considerato, analizzando diverse combinazioni delle tecniche nominate

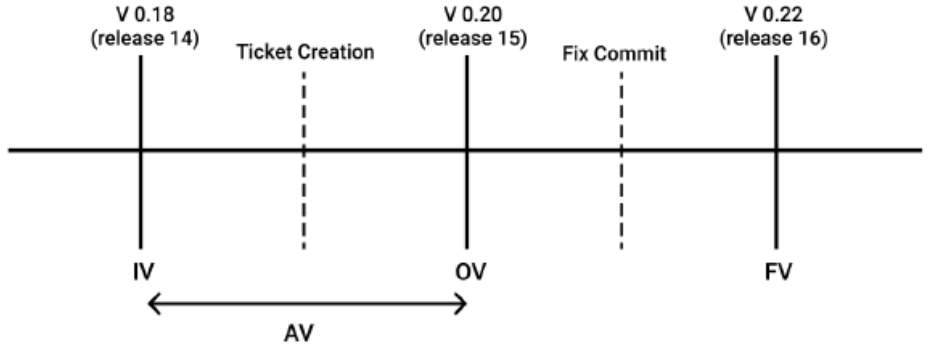
Introduzione e obiettivi

- Per questo caso di studio, vengono presi in considerazione due progetti open-source di Apache:
  - Apache BookKeeper
  - Apache Syncope

- Come classificatori, sono stati presi in considerazione **Random Forest**, **IBK** e **Naive Bayes**.
- Sono state considerate tutte le combinazioni delle tecniche di feature selection, sampling (oversampling e undersampling) e cost sensitivity.

Progettazione: individuazione classi buggy

- Ad ogni bug, possiamo associare un determinato ciclo di vita:
  - il bug viene introdotto nell'**Injected Version**
  - il bug viene rilevato nell'**Opening Version**
  - il bug viene eliminato nella **Fixed Version**



- Tutte le release che vanno dall'injected version inclusa alla fixed version esclusa sono da considerarsi affette dal bug (**Affected versions**)
- Possiamo considerare una classe come buggy se appartiene a una delle affected versions ed è stata modificata dal commit di fix del bug.

Progettazione: raccolta dati e costruzione dataset

- Per la raccolta dei dati del passato relativi alle releases del progetto e alle issues, è stato utilizzato l'issue tracking system **Jira.**
- Le issues prese in considerazione sono solo quelle di tipo «bug», con risoluzione «fixed» e con stato «closed» o «resolved».
- All'interno di Jira, sono sempre indicate la opening version e la fixed version di un'issue, mentre, in molti casi, non vengono indicate le affected version.
- Per determinare l'injected version quando non indicata, quindi, è stata utilizzata la tecnica di **Proportion Cold Start**, la quale calcola il parametro p in modo che

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

Dove p corrisponde alla media del valore $\frac{FV - IV}{FV - OV}$ calcolato per tutte le issues, di cui è presente l'injected version, di tutti i progetti Apache

Progettazione: raccolta dati e costruzione dataset

- Per la costruzione del **dataset**, sono stati recuperati tutti i commit del progetto tramite il version control system **Git**.
- Mettendo insieme le informazioni recuperate da Jira con i commit, è stato possibile determinare il commit di fix per ogni issue ed assegnare, quindi, la bugginess alle classi.
- Ogni istanza del dataset, quindi, è costituita da:
  - Release
  - Classe software, misurata al termine della specifica release
  - Metriche relative alla specifica classe
  - Etichetta che indica se la specifica classe, nella specifica release, era buggy
- A causa del problema dello **snoring**, per il quale possiamo avere dei bug «dormienti», ovvero che non sono stati ancora rilevati, diamo in input al classificatore solo la prima metà delle releases. Le releases più recenti, infatti, sono maggiormente sottoposte a questo problema e non determinano, quindi, dei dati accurati.

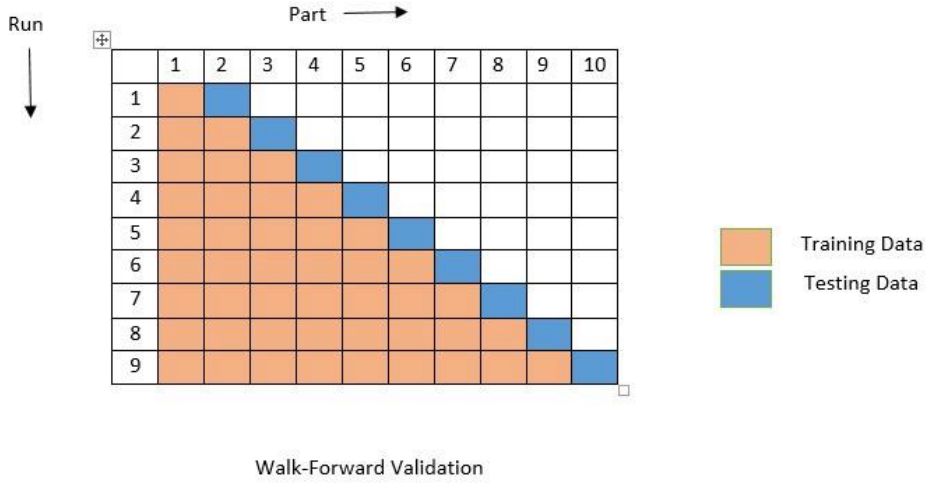
Progettazione: raccolta dati e costruzione dataset

Le metriche calcolate per ogni istanza del dataset sono le seguenti:

METRICA	DESCRIZIONE
Size	Dimensione in LOC della classe (cumulativa tra le releases)
NR	Numero di revisioni
Age	Età della classe in settimane (cumulativa tra le releases)
NAuth	Numero di autori diversi che hanno modificato la classe (cumulativa tra le releases)
ChgSetSize	Numero di classi di cui è stato fatto il commit insieme alla classe in esame
MAX_ChgSet	Massimo ChgSetSize tra le revisioni della classe nella release
AVG_ChgSet	ChgSetSize medio tra le revisioni della classe nella release
Churn	LOC aggiunte – LOC rimosse nella release
MAX_Churn	Massimo churn della classe nella release
AVG_Churn	Churn medio della classe nella release

Progettazione: valutazione dei classificatori

- Per effettuare la valutazione dei classificatori, utilizziamo la tecnica del **Walk Forward**, che è di tipo **time-series**, ovvero tiene conto dell'ordine temporale dei dati.
- Ad ogni run, quindi, le releases di training non devono essere influenzate dal «futuro». Quindi, supponendo di considerare l'iterazione con le prime k releases, il labeling delle classi deve essere effettuato tenendo in considerazione solo i bug con fix version <= k.
- Il testing set, al contrario, deve essere il più accurato possibile, quindi il labeling può essere effettuato tenendo in considerazione tutti i dati a disposizione.



Walk-Forward Validation

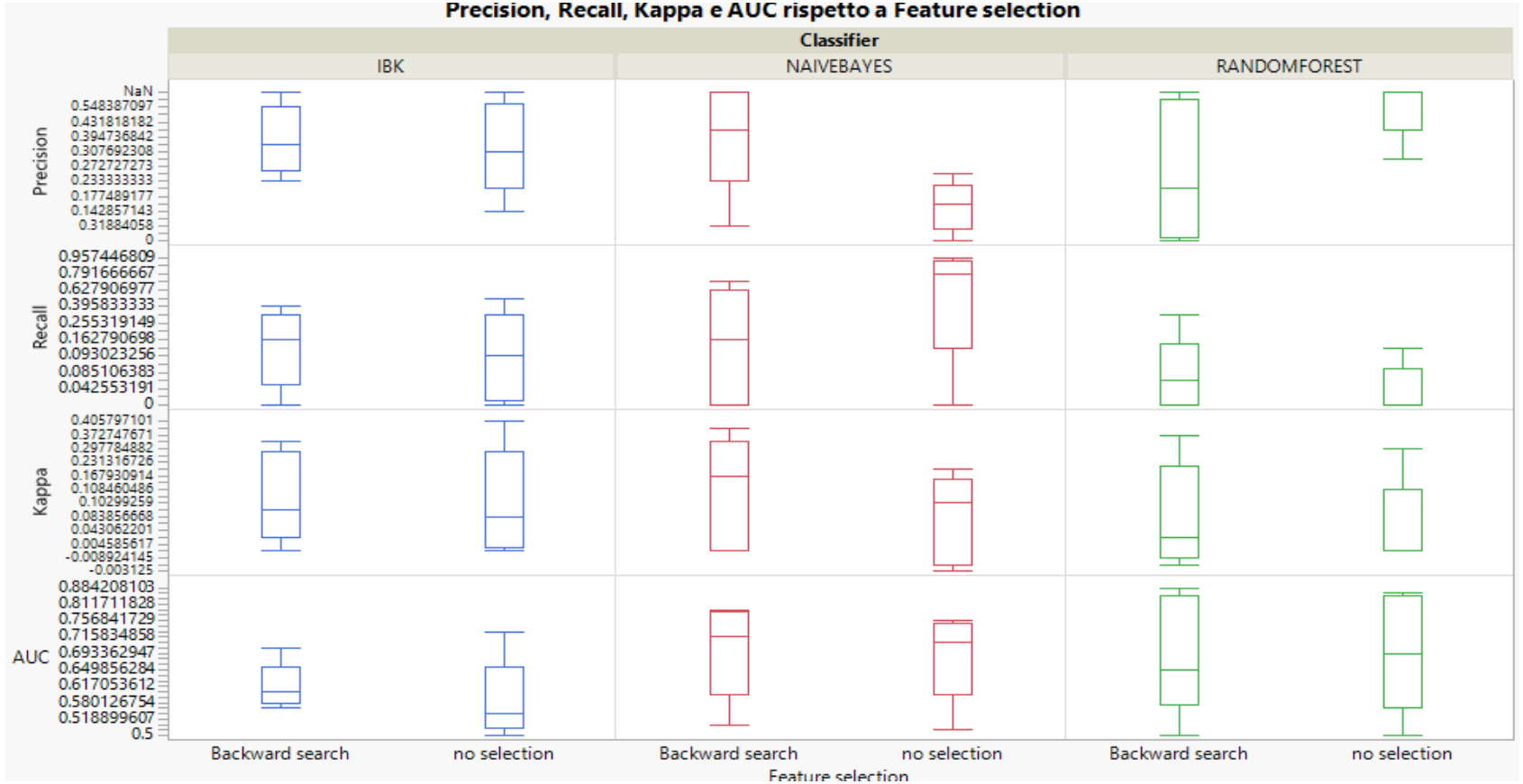
Progettazione: tecniche di utilizzo per i classificatori

- Definiti i classificatori da utilizzare e la tecnica di validazione, andiamo ora a considerare le tecniche di utilizzo:
  - **Feature Selection:** è stato applicato l'algoritmo greedy **backward search**, poiché il numero delle metriche considerato non è molto elevato e poiché si ha come obiettivo quello di eliminare le metriche superflue.
  - **Sampling:** Sono state considerate sia la tecnica di **undersampling** che quella di **oversampling**. Per quanto riguarda BookKeeper la percentuale di positivi è del 13,6%, mentre per Syncope è del 6,73%. Il dataset di Syncope, quindi, è fortemente sbilanciato e presenta un totale di istanze molto elevato, motivo per cui in questo caso non è stato applicato l'oversampling.
  - **Cost sensitivity:** Sono state considerate la tecnica del **Sensitivity Threshold** e la tecnica del **Sensitivity Learning** (CFN = 10* CFP)
- Sono state quindi considerate tutte le possibili combinazioni tra queste tecniche, per valutarne la migliore.
- Per la valutazione dei classificatori con queste tecniche, è stato utilizzato il tool **Weka**

Risultati ottenuti

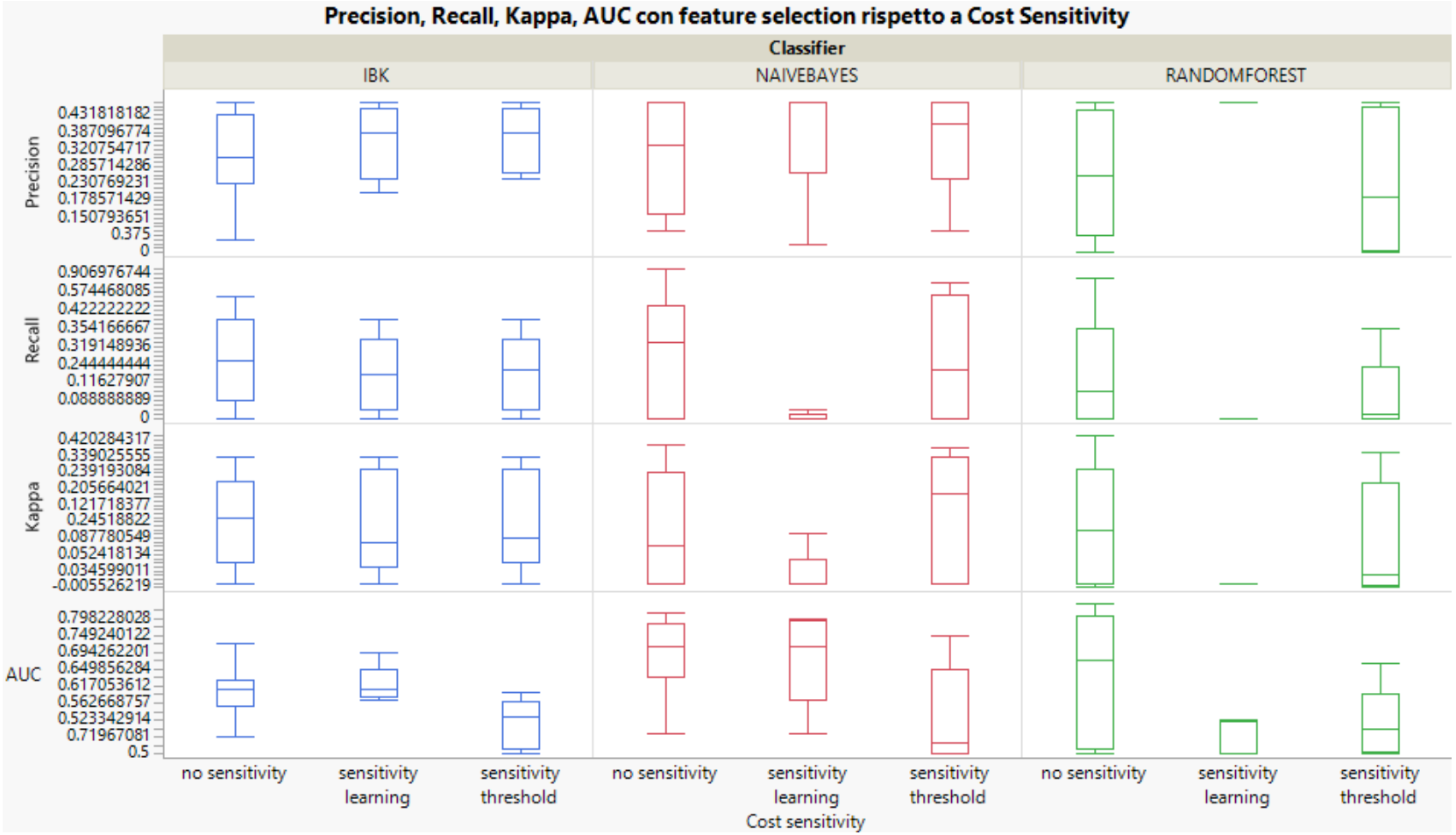
- Nelle slides seguenti sono riportati dei grafici di tipo box-plot, raffiguranti i risultati ottenuti dalla valutazione dei tre classificatori sui progetti BookKeeper e Syncope, al variare delle tecniche utilizzate.
- Le metriche prese in considerazione per la valutazione delle performance dei classificatori sono:
  - **Precision**: indica la percentuale di volte che una istanza, classificata positiva, è effettivamente positiva
$$\frac{TP}{TP+FP}$$
  - **Recall**: indica la percentuale di veri positivi che si è stato in grado di classificare
$$\frac{TP}{TP+FN}$$
  - **Kappa**: percentuale di volte in cui la classificazione è stata più accurata di un classificatore dummy
  - **AUC**: misura la capacità di un classificatore di distinguere tra i positivi e i negativi.

Risultati ottenuti: BookKeeper



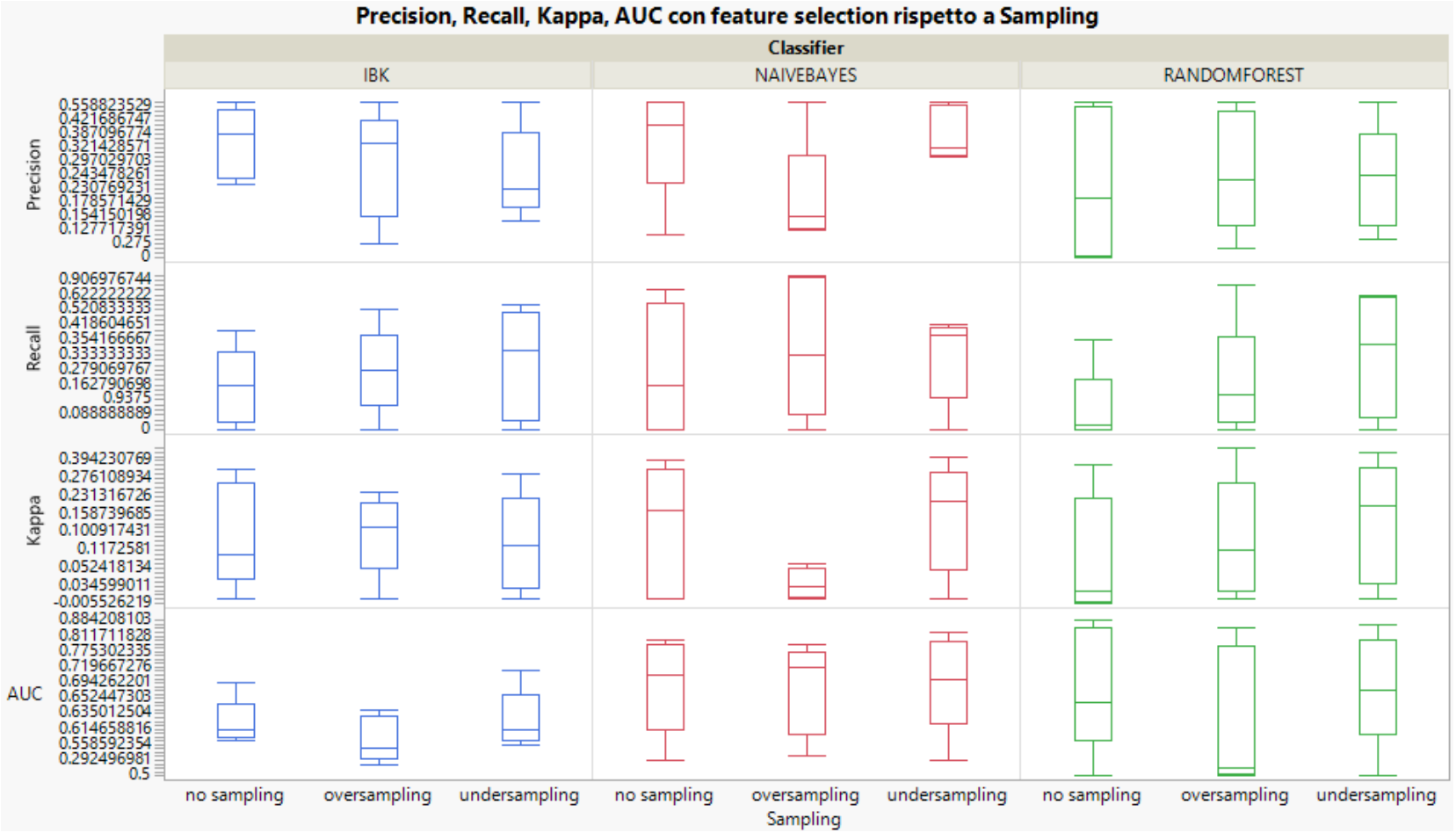
- Analizzando le metriche con e senza **feature selection**, vediamo che per IBk e Naive Bayes otteniamo un miglioramento delle metriche se applichiamo la selezione degli attributi, mentre in Random Forest questo non accade.
- Il classificatore che in questo caso sembra avere le prestazioni migliori è **Naive Bayes** con applicata la **feature selection**.

Risultati ottenuti: BookKeeper



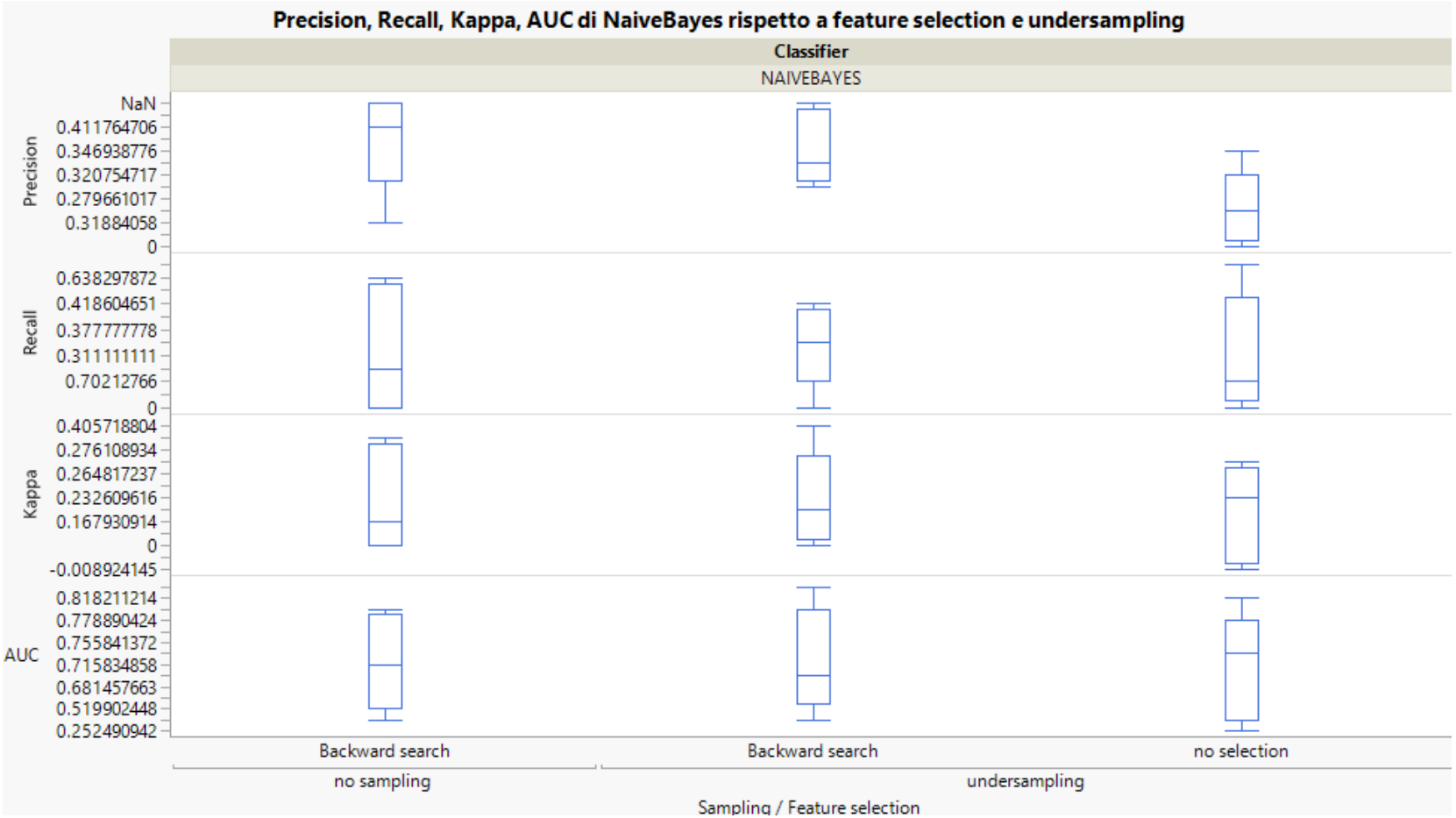
- Analizzando ora le metriche al variare della **cost sensitivity** e fissando la feature selection, vediamo che il **sensitivity learning** peggiora tutte le metriche per tutti i classificatori, escludendo la precision.
- Con il sensitivity threshold, osserviamo un peggioramento per tutte le metriche per Random Forest, e un lieve miglioramento della precision per gli altri classificatori. Non sembra, quindi, molto conveniente il suo utilizzo.

Risultati ottenuti: BookKeeper



- Applicando il sampling, oltre alla feature selection, possiamo vedere che con **l'undersampling** otteniamo un miglioramento di tutte le metriche, tranne la precision, come ci si aspetta. Inoltre, l'undersampling sembra in generale funzionare meglio rispetto all'**oversampling**.
- Anche in questo caso, il classificatore che ha delle migliori prestazioni è **Naive Bayes** con undersampling e feature selection.

Risultati ottenuti: BookKeeper

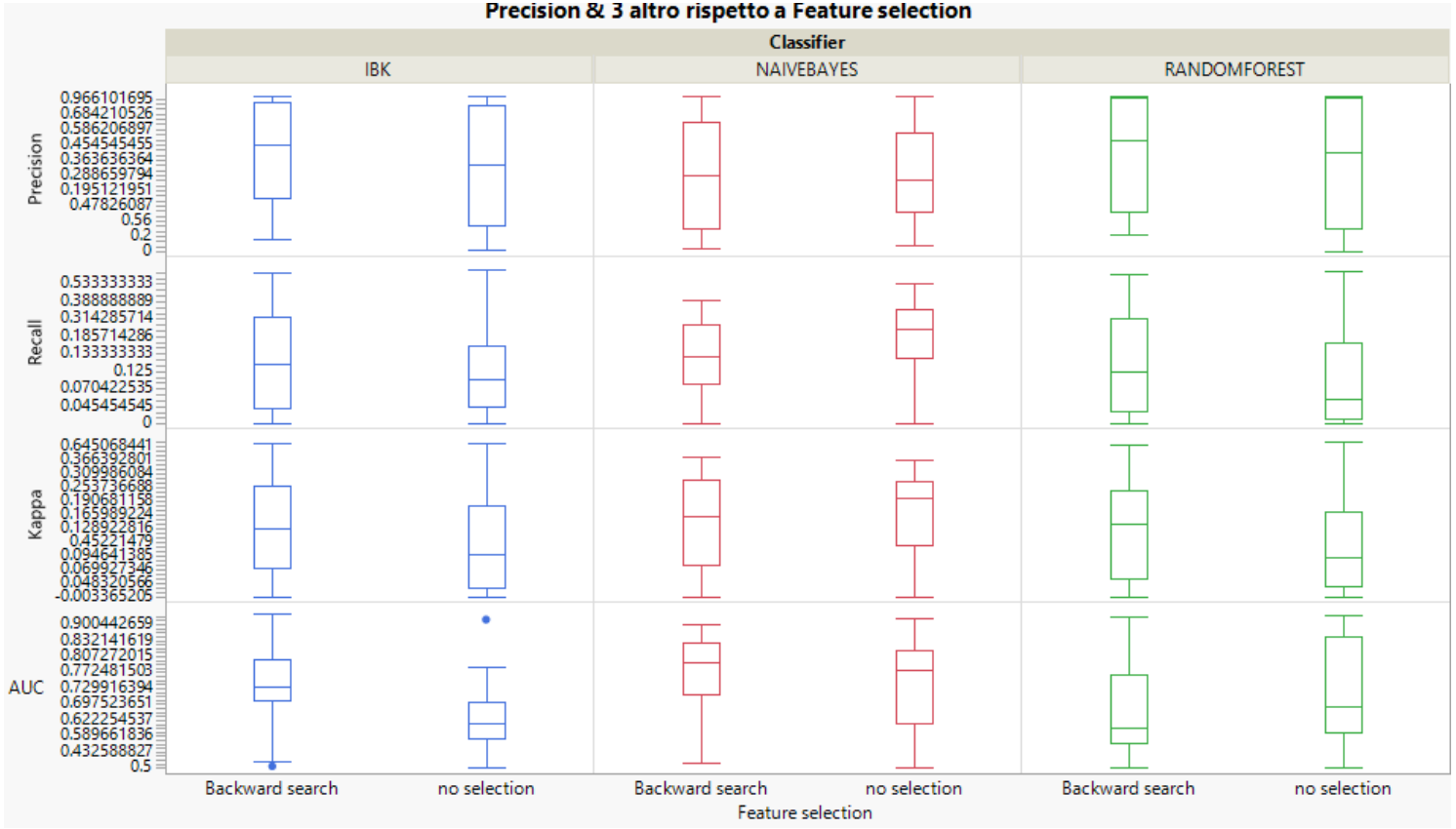


- Mettiamo ora a confronto la backward search senza sampling, con undersampling e l'undersampling senza feature selection, e valutiamo le prestazioni del classificatore Naive Bayes, che dai grafici precedenti sembra essere il più promettente.

Considerazioni: BookKeeper

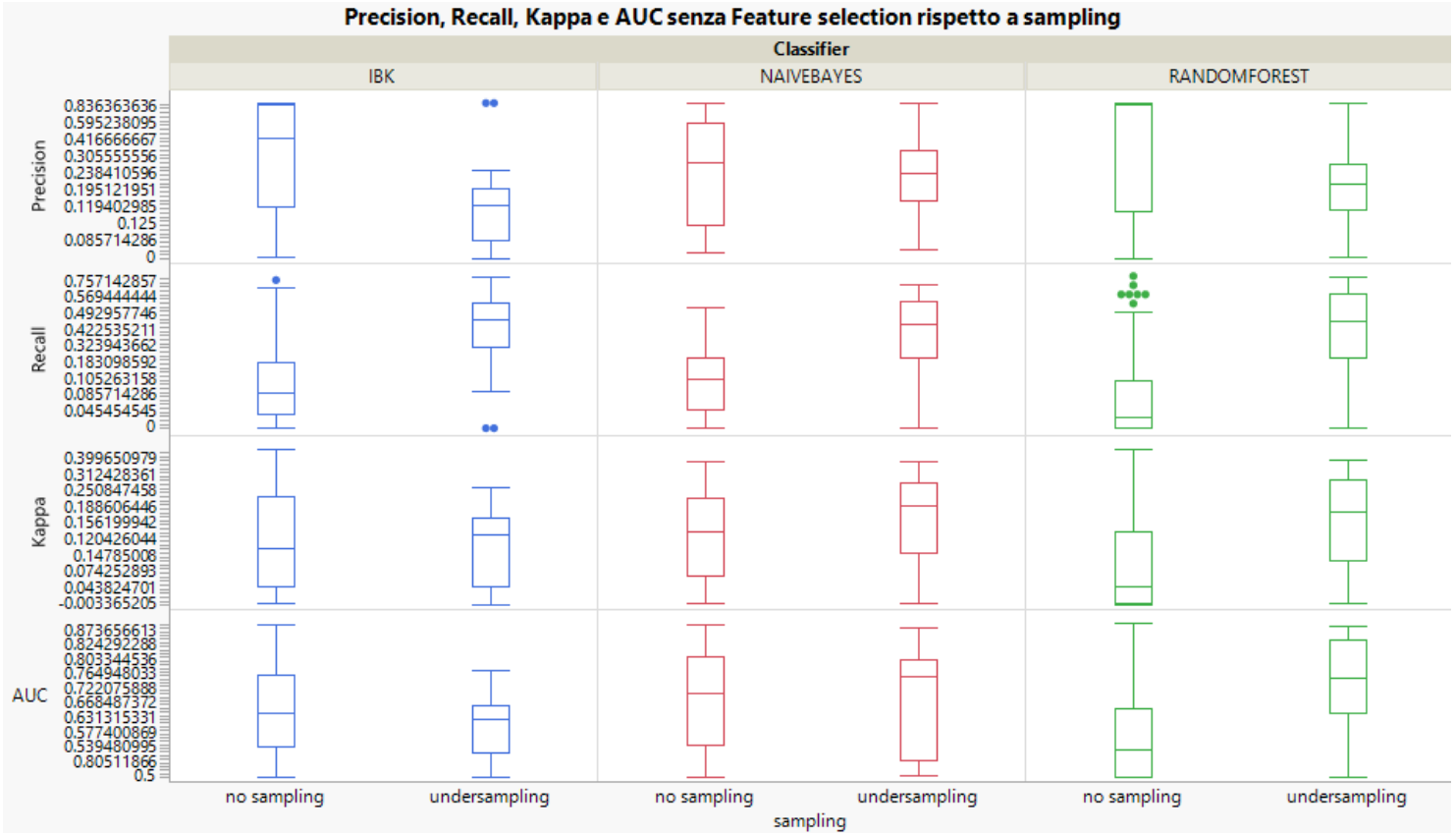
- Dal grafico precedente possiamo notare che la tecnica dell'undersampling utilizzata da sola permette di avere, rispetto alle altre due combinazioni, un aumento delle metriche AUC e Kappa, ma una diminuzione significativa per Precision e Recall.
- La combinazione di feature selection e undersampling, invece, rispetto alla sola feature selection, presenta una Precision leggermente minore del caso con sola feature selection e una Recall maggiore, come atteso. Le altre due metriche, invece, rimangono pressocché le stesse.
- Considerando, quindi, la combinazione **feature selection – undersampling** con **Naive Bayes**, riusciamo ad ottenere delle buone prestazioni. Inoltre abbiamo il vantaggio di considerare un set di dati di dimensione minore, quindi più gestibile e che permette di addestrare il modello in tempo minore.

Risultati ottenuti: Syncope



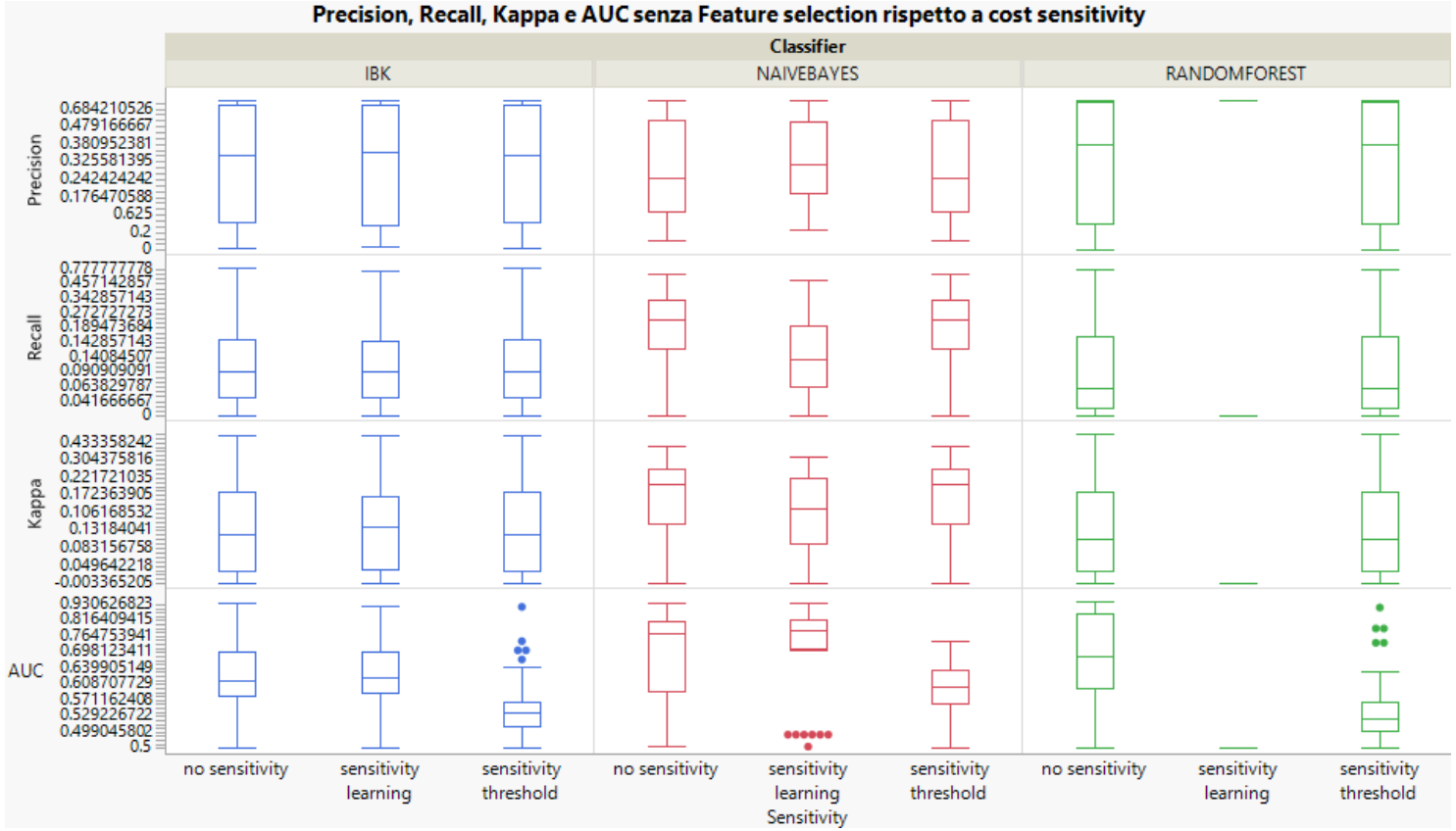
- Analizzando le metriche con e senza **feature selection**, vediamo che per IBk e Random Forest otteniamo complessivamente un miglioramento delle metriche se applichiamo la selezione degli attributi, mentre in Naive Bayes il miglioramento non avviene.
- Il classificatore che in questo caso sembra avere le prestazioni migliori è **Naive Bayes** senza feature selection, anche se Ibk con feature selection presenta comunque delle buone prestazioni.

Risultati ottenuti: Syncope



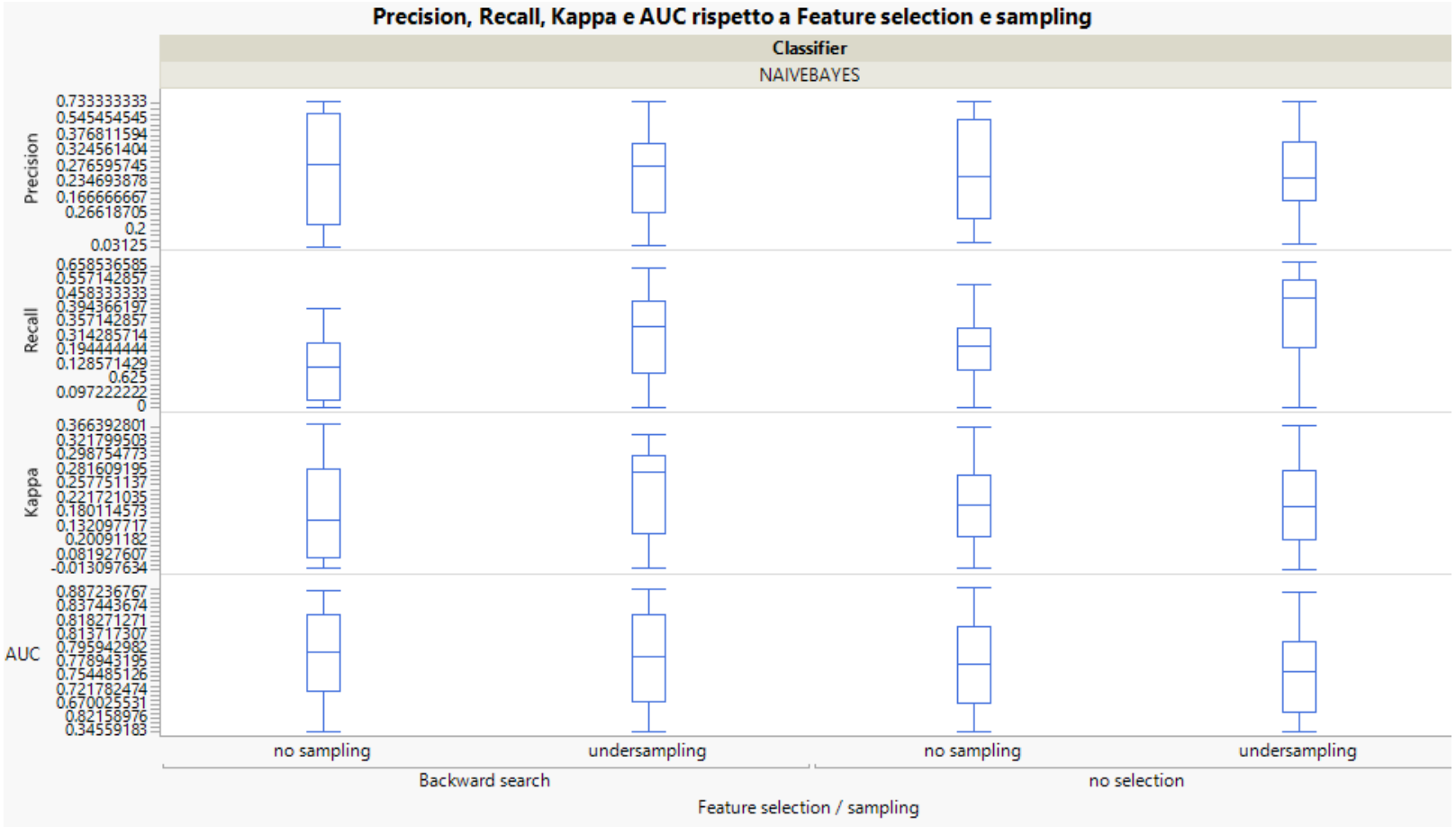
- Consideriamo ora, quindi, la tecnica del **sampling** senza la feature selection. Possiamo vedere che, escludendo la Precision che peggiora per tutti i classificatori (come atteso), l'undersampling, in generale, porta un miglioramento delle prestazioni, soprattutto per Naive Bayes e Random Forest.
- Anche con **l'undersampling**, il classificatore che sembra andare meglio è **Naive Bayes**, superando anche le prestazioni del caso senza sampling.

Risultati ottenuti: Syncope



- Prendiamo ora in analisi la **cost sensitivity**. Per Ibk non vediamo grossi cambiamenti né con il sensitive learning né con il sensitive threshold. Anche per Random Forest l'utilizzo della cost sensitivity non porta miglioramenti anzi, applicando il sensitive learning, le prestazioni subiscono un crollo. Infine, con Naive Bayes, si osservano dei peggioramenti con il sensitive learning e una situazione molto simile al caso senza sensitivity per il sensitive threshold.

Risultati ottenuti: Syncope



- Consideriamo, ora, solo il classificatore Naive Bayes, che sembra comportarsi meglio degli altri, e analizziamo le metriche confrontando le combinazioni delle tecniche di oversampling e feature selection.

Considerazioni: Syncope

- Dal grafico precedente, possiamo osservare che, in generale, **l'undersampling** migliora le metriche sia in combinazione con la feature selection che da solo.
- Nella combinazione feature **selection – undersampling**, la Precision, la Kappa e AUC risultano essere migliori rispetto alla combinazione con solo undersampling. La Recall, invece, risulta più alta per quest'ultima, anche se non di molto.
- Anche se sarebbe auspicabile una Recall più alta, tenendo conto che i falsi negativi pesano di più dei falsi positivi, la presenza delle altre tre metriche con valori più alti ci fa protendere verso la configurazione feature selection – undersampling. Inoltre, applicando la feature selection, otteniamo anche, come vantaggio, un tempo necessario all'addestramento del classificatore minore.
- Quindi, per Syncope, **Naive Bayes con feature selection e undersampling** sembra essere la scelta giusta.

Link utili

- Repository GitHub: https://github.com/federicavil/csvLab
- SonarCloud: https://sonarcloud.io/project/overview?id=federicavil_csvLab