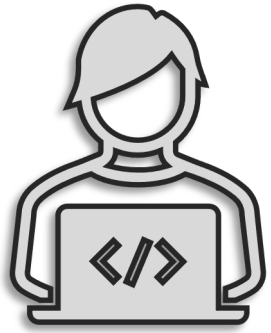


Machine Learning for Software Engineering

Emanuele Valzano – Matricola 0339334

A.A. 2022/2023

AGENDA

Introduzione

Obiettivo e Contesto

Metodologia

Risultati e Conclusioni

Links a GitHub & SonarCloud

INTRODUZIONE

Nell'ambito dell'Ingegneria del Software è fondamentale condurre un'attività di testing mirata, in modo da far emergere eventuali bug del software.

- Garantire l'affidabilità e la qualità del software mediante un valido processo di sviluppo.
  - Meccanismi di V&V e QA

- Tenere sempre in considerazione le risorse ed il budget a disposizione.

PROBLEMA: Come identificare, tramite delle predizioni, le classi che conterrano errori ?

OBIETTIVO

L'obiettivo è di duplice natura:

- **Il più generale è rendere il processo di testing mirato, in modo da predire efficacemente le classi che attualmente possono contenere dei difetti.**
  - Sfruttare gli strumenti del Machine Learning

- **Identificare il classificatore che effettua le predizioni migliori e con quali tecniche di utilizzo.**
  - Applicare diverse tecniche sul dataset fornito per l'addestramento dei classificatori.
  - Analizzare l'andamento delle prestazioni di tali classificatori, al variare delle suddette tecniche.

CONTESTO

Presi in esame due progetti di Apache Foundation:

- ✓ Apache BookKeeper
- ✓ Apache Syncope

Grazie alla presenza di progetti di questa portata, di carattere open-source, è possibile condurre delle sperimentazioni più avanzate sull'Ingegneria del Software, utili ad identificare attività di testing efficaci.

Sui progetti in esame è importante avere un quadro completo delle classi caratterizzate da bug andati in produzione in passato, e in quali release del progetto.

METODOLOGIA

1. **Identificazione coppie (classe, release) buggy**
   - Raccolta di metriche che ragionevolmente incidono sulla bugginess.
   - Costruzione del Dataset di Training e Testing, per l'addestramento e valutazione dei classificatori.

2. **Utilizzo di modelli di Machine Learning**
   - Valutazione di tre classificatori
     - ✓ Random Forest
     - ✓ Naïve Bayes
     - ✓ IBK
   - Tre tecniche combinate
     - ✓ Feature Selection
     - ✓ Balancing
     - ✓ Cost Sensitive

METODOLOGIA (strumenti di misurazione)

- Version Control System per risalire ai commit dei progetti esaminati, in modo da poter calcolare le metriche.

- Issue Tracking System per la raccolta dei ticket e delle versioni
- Opening Version e Fixed Version disponibili

PROBLEMA: Come gestire i Ticket di Jira che non contengono le Affected Versions ?

METODOLOGIA (proportion)

IDEA: Ciascun bug è caratterizzato da un ciclo di vita

- Per conoscere le **AV,** oltre che all'attimo di risoluzione del bug, è necessario risalire alla prima release dopo l'injection del difetto (**IV**).

  - **Proportion:**
    - Si risale all'Injected Version attraverso la tecnica di **Proportion Incremental** sulla media del Proportion dei ticket a disposizione fino a quel momento.

    - **Cold Start:**
      - Se i ticket disponibili non sono sufficienti, si calcola la mediana dei valori medi del Proportion ottenuti dai ticket significativi di altri progetti Apache, che contengono informazioni sulle AV, per stimare il risultato.

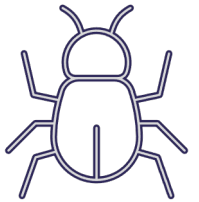
$$p = \frac{FV - IV}{FV - OV}, \qquad se\ FV = OV \Rightarrow p = \frac{FV - IV}{1}$$

METODOLOGIA (metriche)

Nome	Descrizione
Size*	Numero di linee di codice
Nfix*	Numero di revisioni che risolvono bug issues
Nauth*	Numero di autori
Average LOC Added*	Numero medio di LOC aggiunte per revisione
Churn*	Somma delle LOC aggiunte e rimosse sulle revisioni
Average Churn*	Media dei churn sulle revisioni
Age	Età della release
Average Fix Churn*	Media della somma delle LOC aggiunte e rimosse sulle revisioni che risolvono bug issues
Fan Out*	Numero di dipendenze esterne della classe e invocazioni a metodi esterni
NR*	Numero di revisioni

* = Intra-release e non cumulative

METODOLOGIA (bugginess)



- Ciascuna coppia **(classe, release)** è associata alle suddette **metriche**, che verranno sfruttate dai classificatori per effettuare il training dei dati ed essere in grado di fare delle predizioni.

- Ogni istanza del dataset sarà relativa ad una coppia (classe, release) e sarà composta dalle metriche sopraindicate e dal **valore da predire** (is_buggy):
  - **is_buggy** è un valore booleano che indica se la coppia è buggy o meno.

- **Come etichettare la bugginess ?**

  - Vengono presi tutti I ticket di tipo ***BUG*** con risoluzione ***FIXED*** e con stato ***CLOSED*** o ***RESOLVED*** da Jira.
  - Per tutte le **AV**, ogni classe modificata da un commit linkato ad uno dei ticket estratti viene etichettata come buggy.

METODOLOGIA (snoring & datasets)

- Per mitigare il fenomeno dello **snoring** – che consiste nell'avere coppie (classe-release) etichettate come non buggy, ma che in realtà hanno dei bug dormienti – è opportuno scartare dal dataset il 50% delle release più recenti.

- I classificatori necessitano di **Training Datasets** per essere addestrati, cosicchè possano effettuare delle predizioni.
  - **Affetti da Snoring**, per essere il più fedeli possibili all'utilizzo reale del classificatore.
  - Non possono sfruttare i dati del futuro.

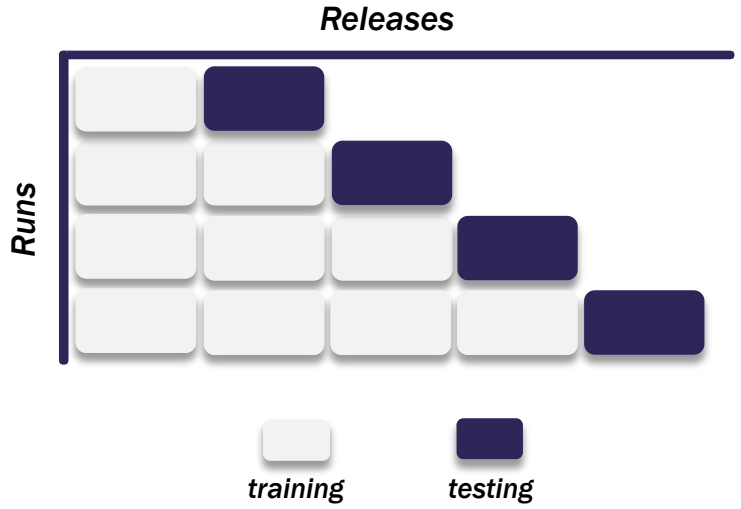
- Per effettuare una valutazione sulle predizioni dei classificatori è essenziale disporre di **Testing Datasets**.
  - **Esenti da Snoring**.
  - Sfrutta tutti i dati a disposizione.

METODOLOGIA (validazione)

- Si ricorda che l'obiettivo è stabilire quale classificatore abbia le prestazioni migliori, e con quali tecniche di utilizzo.

- A questo scopo, si effettua una valutazione dei classificatori con l'ausilio di tecniche di validazione.

    - Tecnica utilizzata: **Walk Forward**
        - Di tipo **Time-Series**, tiene quindi conto dell'ordine temporale dei dati.
        - Processo iterativo.



- **Training set**: viene costruito con le prime *k* releases* e il labeling avviene esclusivamente con le informazioni che si hanno fino a quel momento.

- **Testing set**: costruito sulla release k*+1–esima, ma eseguendo il labeling in funzione di tutte le informazioni accessibili.

* = il numero della *run* di Walk Forward considerata

METODOLOGIA (metriche di interesse e tecniche di utilizzo)

Classificatori	Tecniche di Balancing	Tecniche di Selection	Sensitive Learning
Random Forest	No Balancing	No Feature Selection	No Sensitive Learning
Naive Bayes	Undersampling	Best First Bi-directional	Sensitive Learning*
IBK	Oversampling		
	Synthetic Minority Oversampling Technique (SMOTE)		

- I classificatori sono stati "costruiti" combinando le tecniche.
- Per l'analisi delle metriche ottenute dalla valutazione di Weka, è stato implementato uno script in **Python** con l'ausilio delle librerie **pyplot** per i grafici e **pandas** per l'analisi e manipolazione dei dati.
- Sono stati prodotti due tipologie di grafici:
  - ✓ *Box plots*
  - ✓ *Linear plots,* che mostrano l'andamento dei classificatori all'aumentare delle releases.

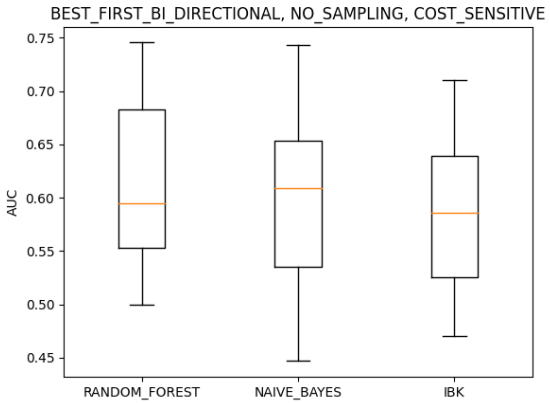
matplotlib

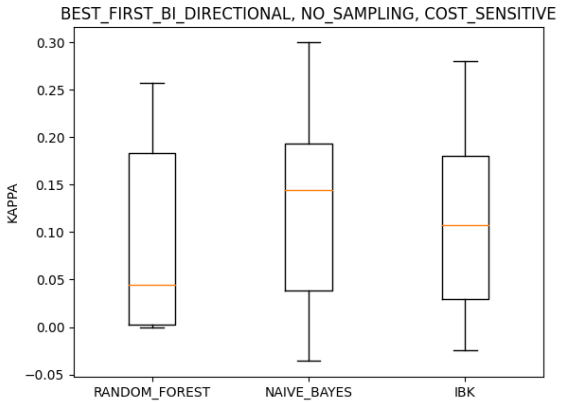
pandas

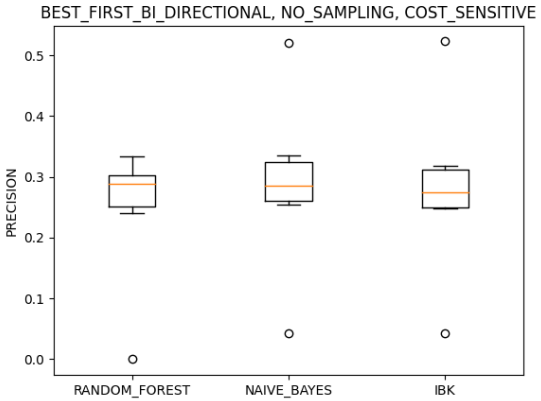
N.B. I linear plots non sono stati inclusi nelle slides, pertanto possono essere visionati su GitHub. Il link è riportato alla fine della presentazione.

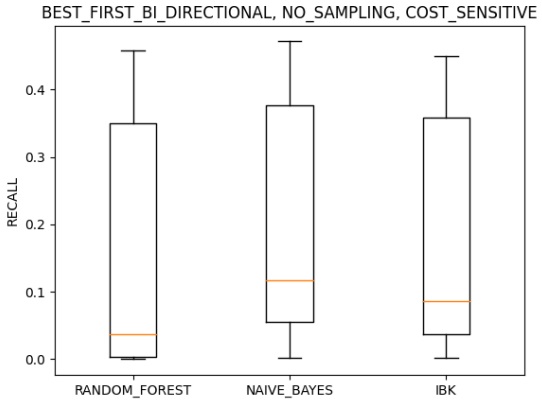
* = Utilizzato solo senza tecniche di Balancing

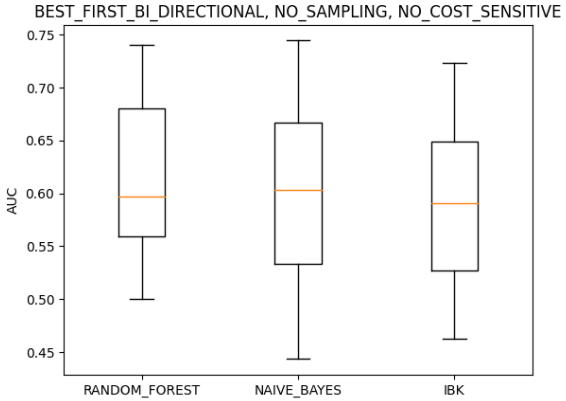
RISULTATI (BookKeeper – box plots [1])

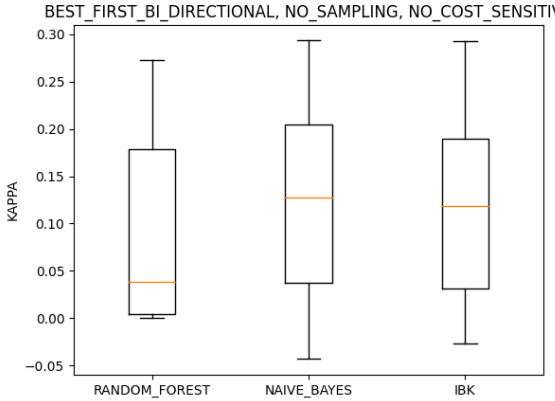


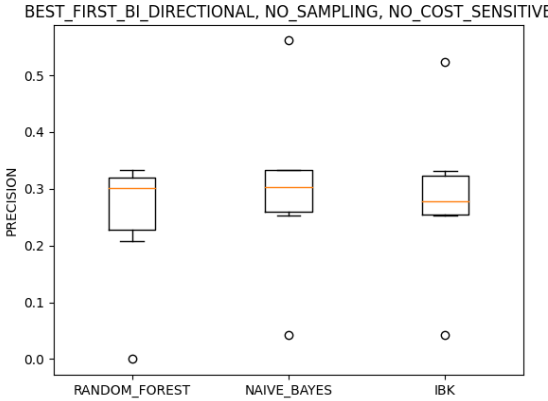


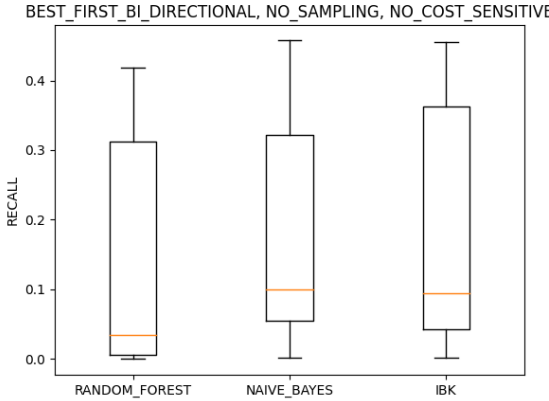




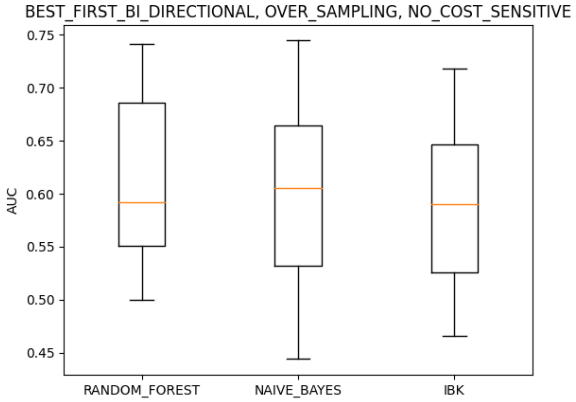


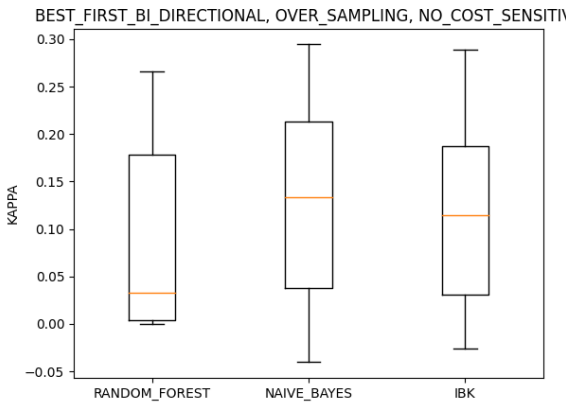


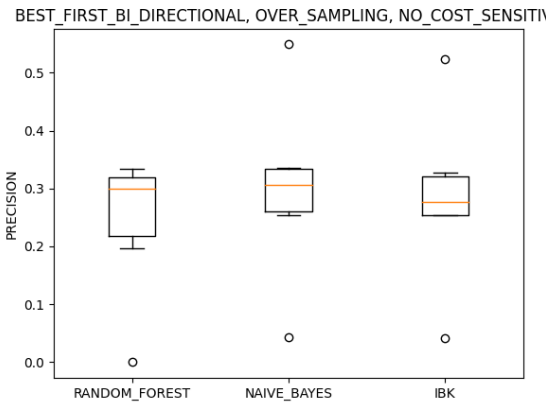


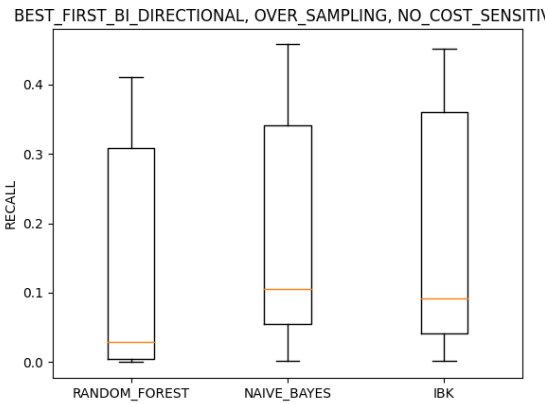


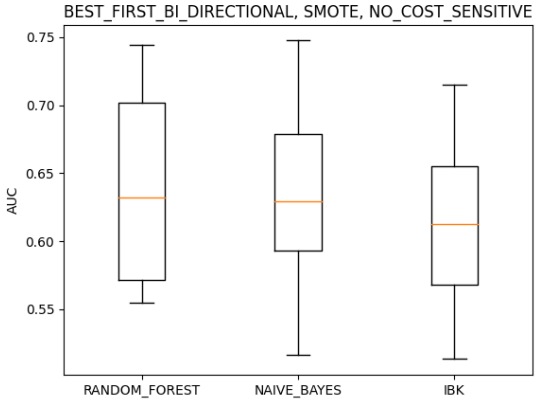
RISULTATI (BookKeeper – box plots [2])

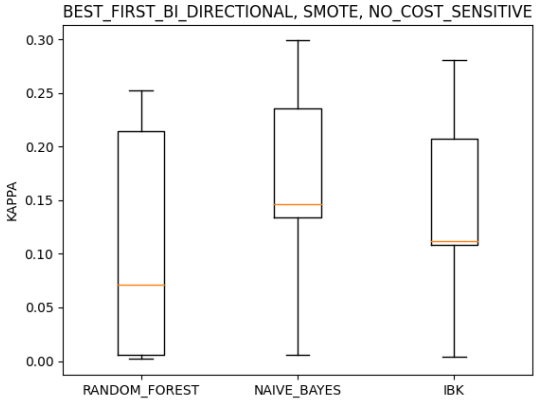


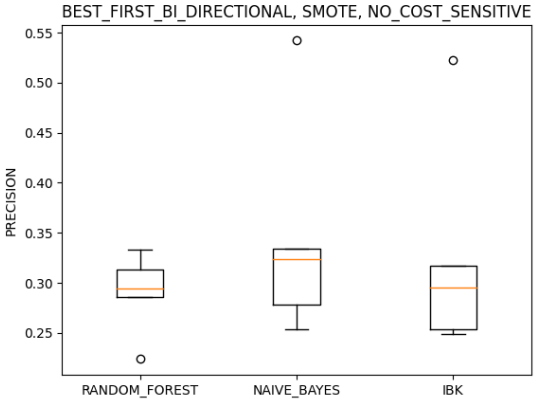


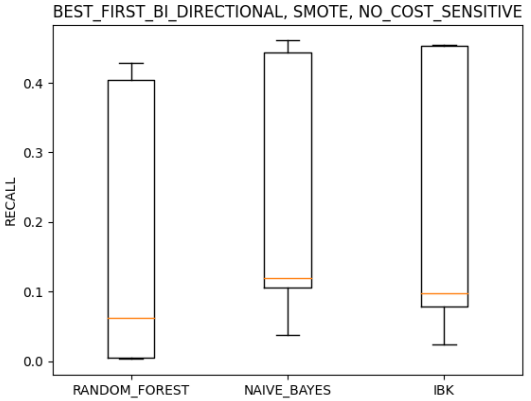




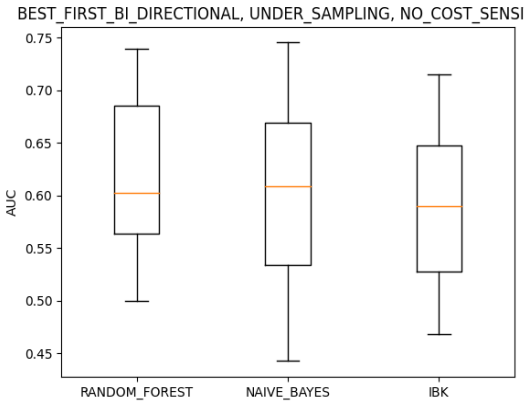


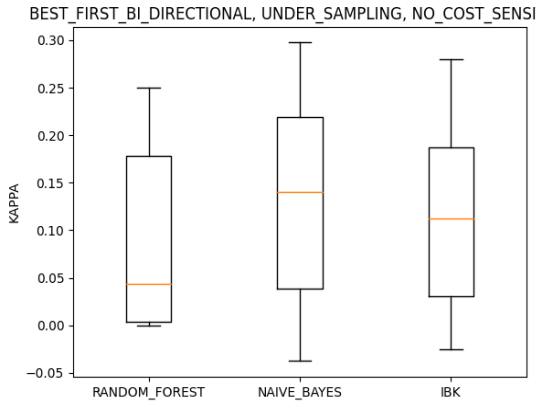


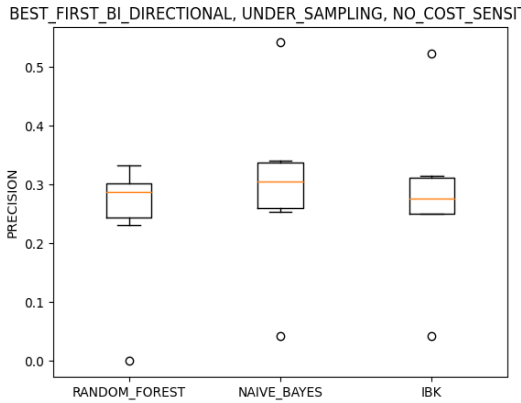


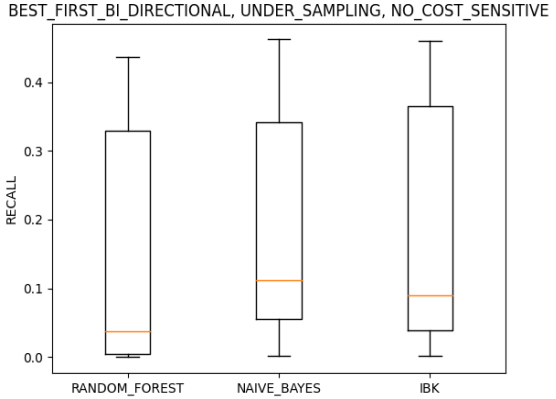


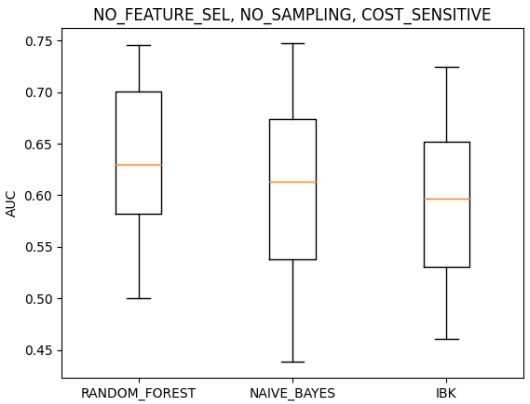
RISULTATI (BookKeeper – box plots [3])

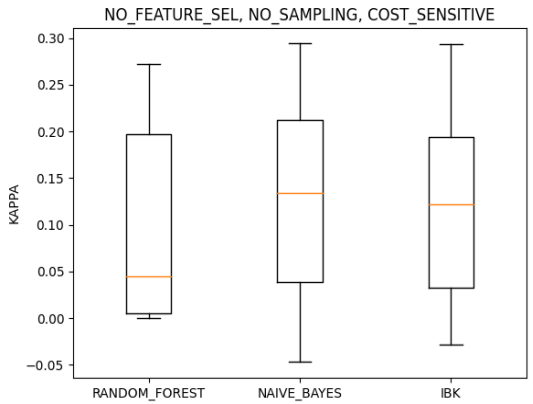


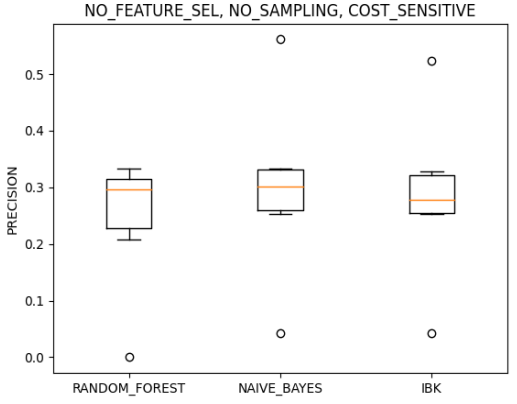


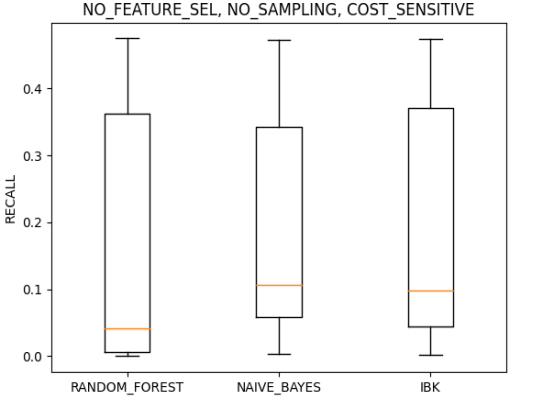




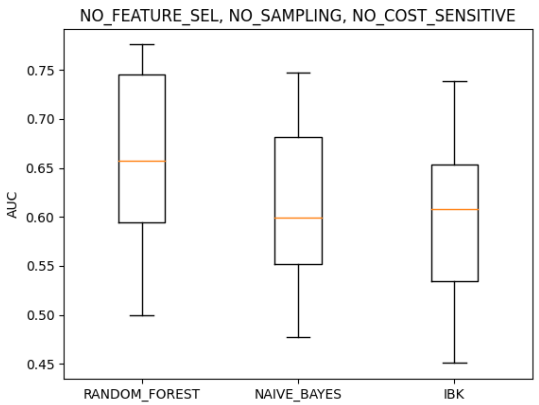


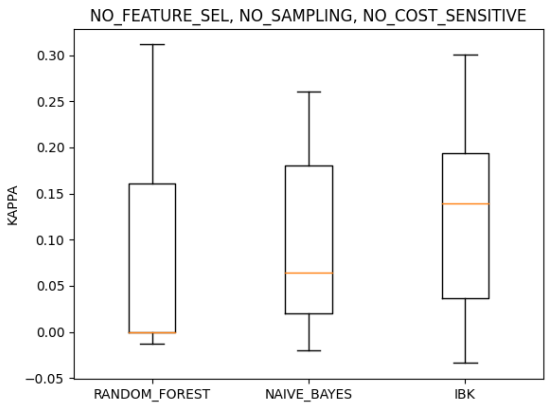


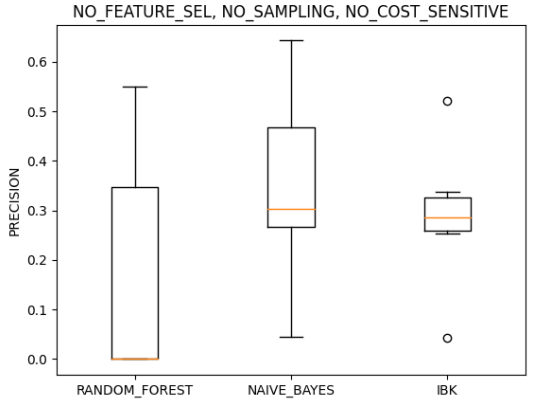


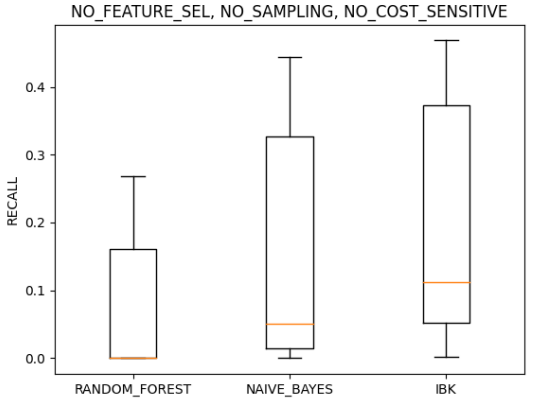


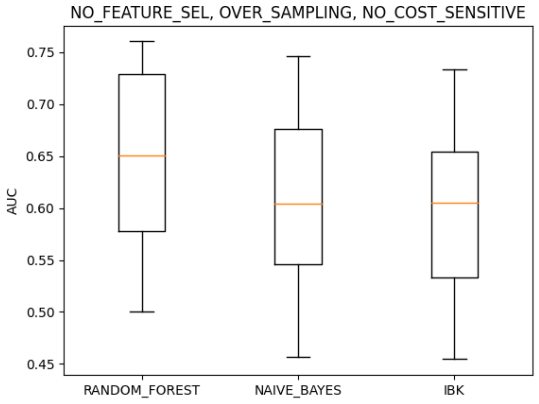
RISULTATI (BookKeeper – box plots [4])

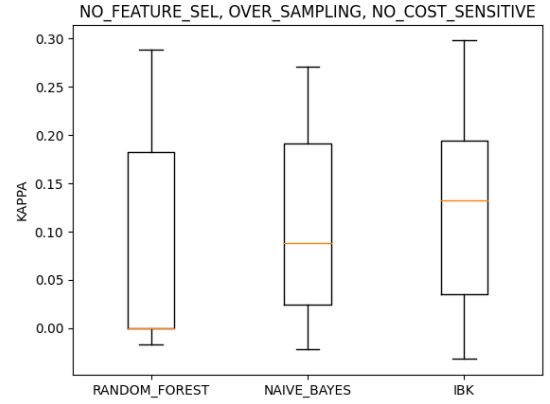


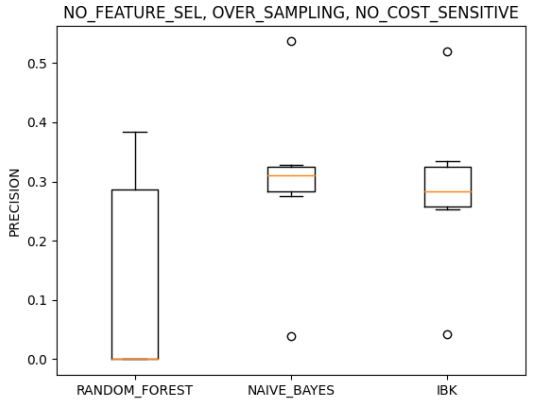


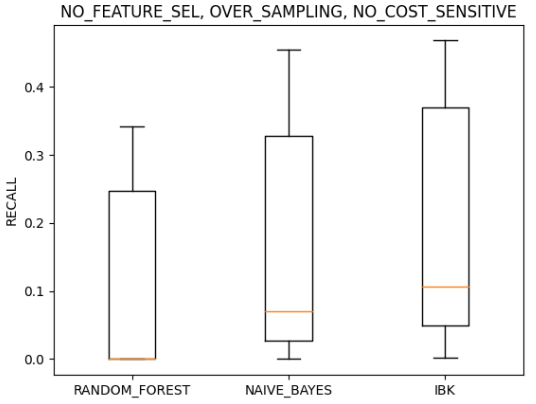




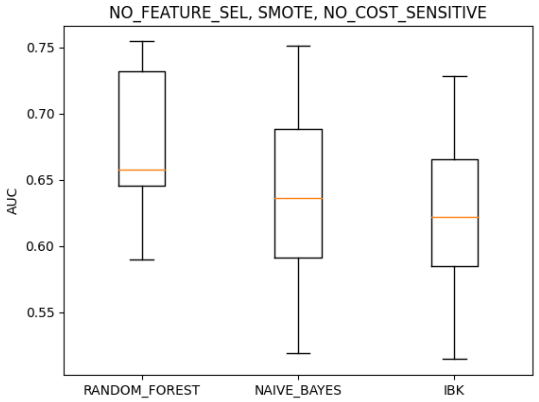


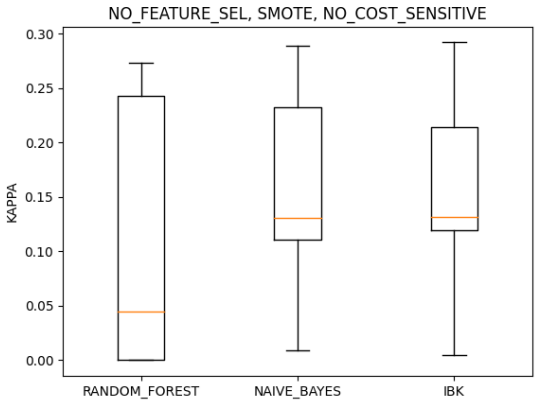


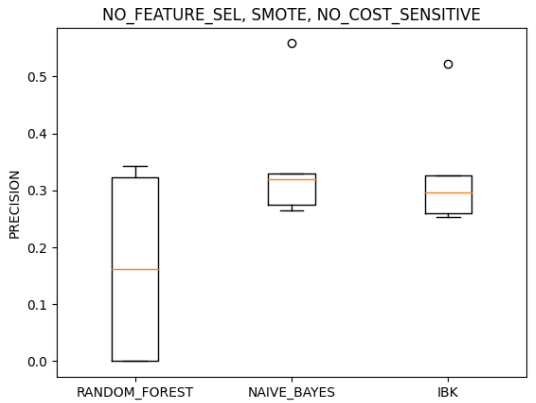


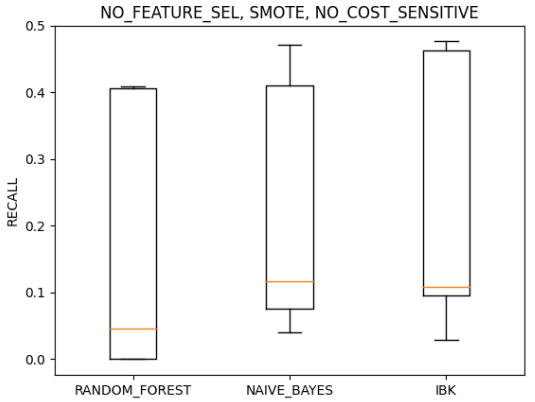


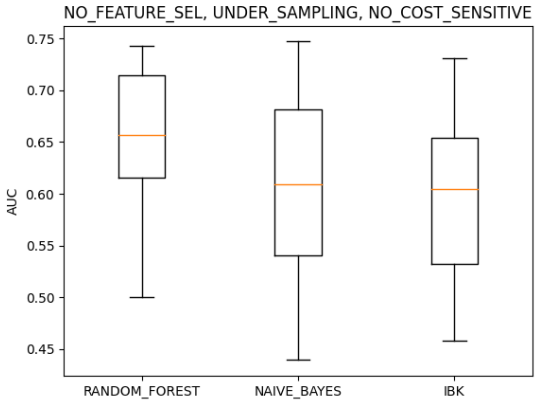
RISULTATI (BookKeeper – box plots [5])

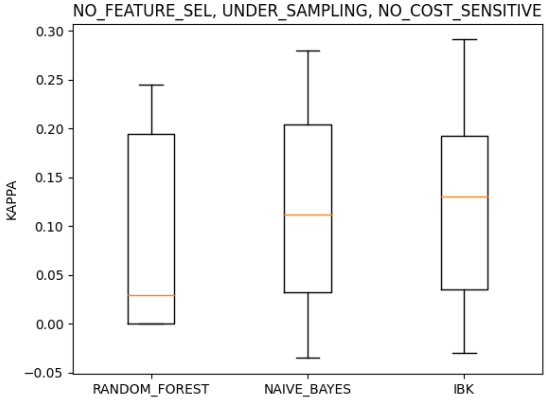


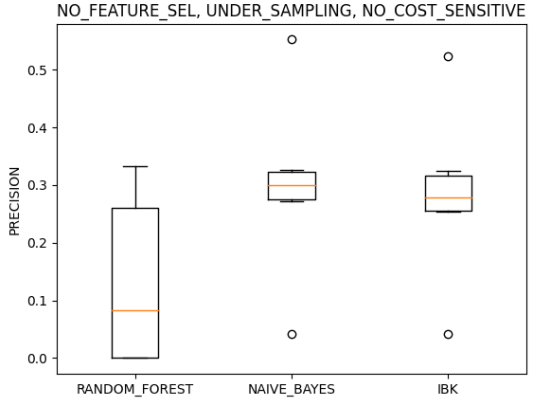


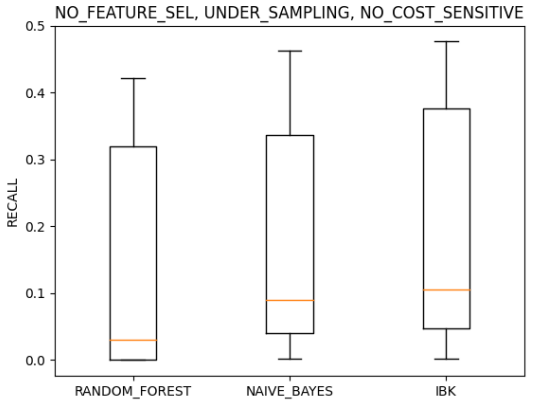










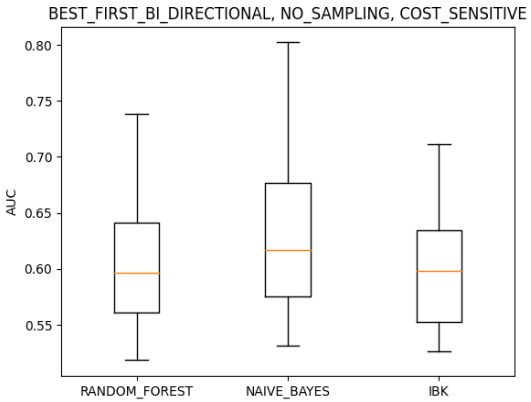


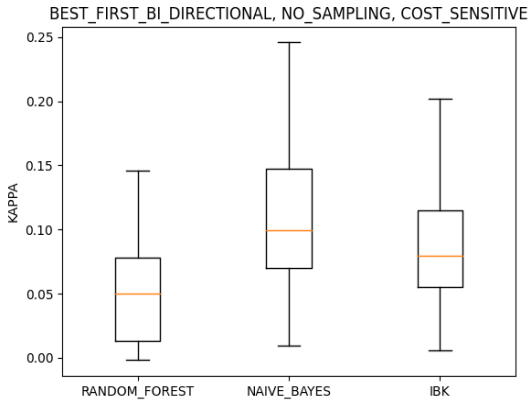
RISULTATI (considerazioni BookKeeper)

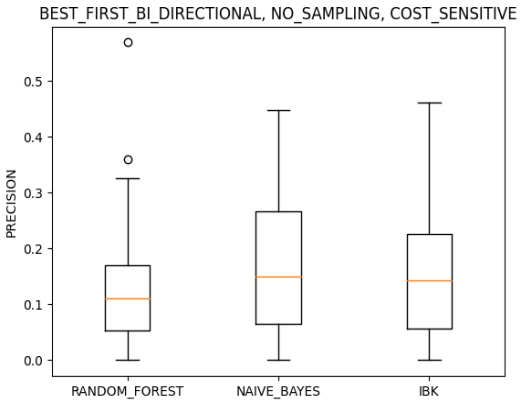
- Tra i classificatori, non ce n'è uno che spicca nettamente rispetto agli altri in termini di performance. In media però, **Naïve Bayes** mette in evidenza delle performance di pochissimo più elevate.

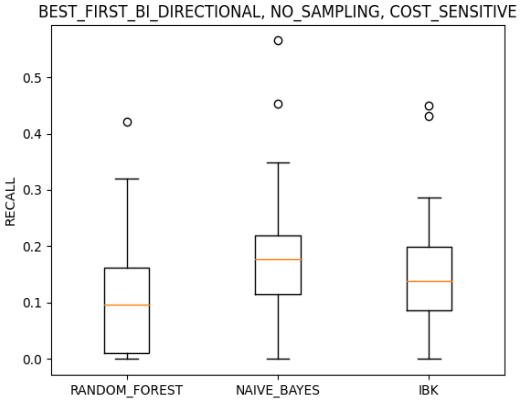
- Non si identifica una tecnica di bilanciamento particolarmente migliore delle altre, ma dipende intrinsecamente dal dataset di partenza e dal classificatore considerato. Tuttavia, **SMOTE** presenta delle leggere migliorie, in media, su tutti i classificatori.

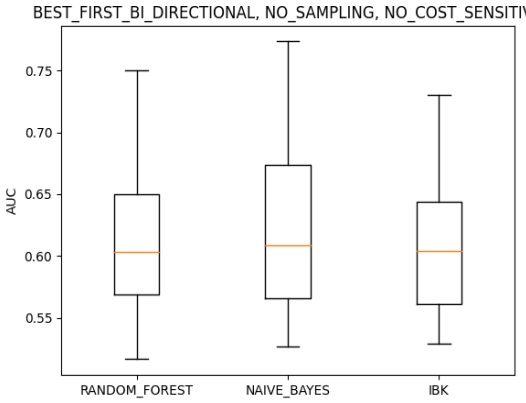
RISULTATI (Syncope – box plots [1])

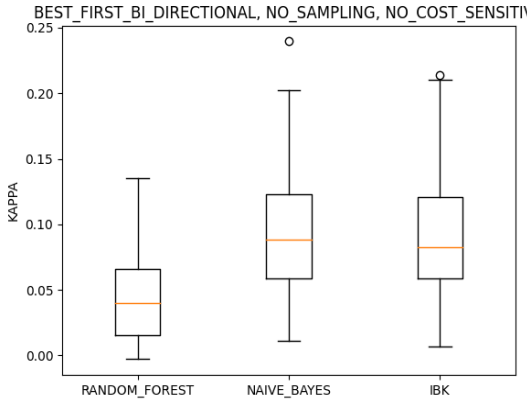


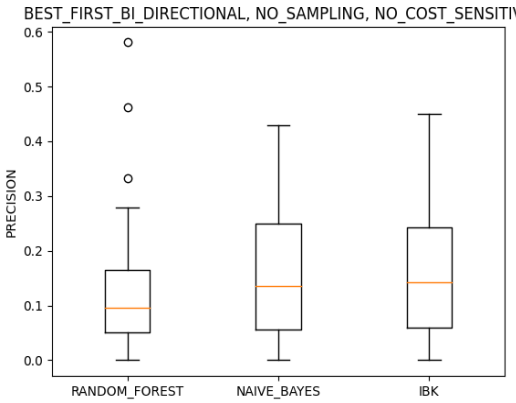


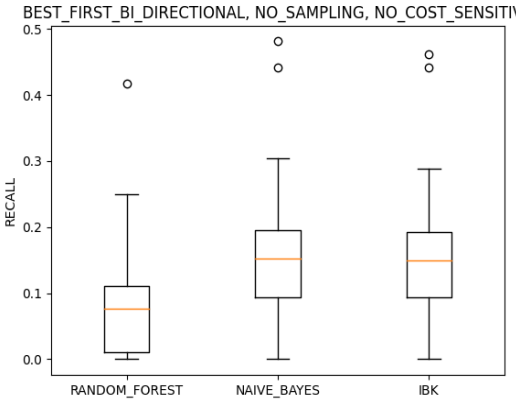




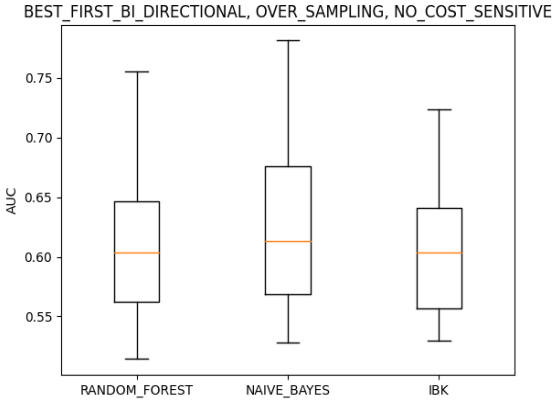


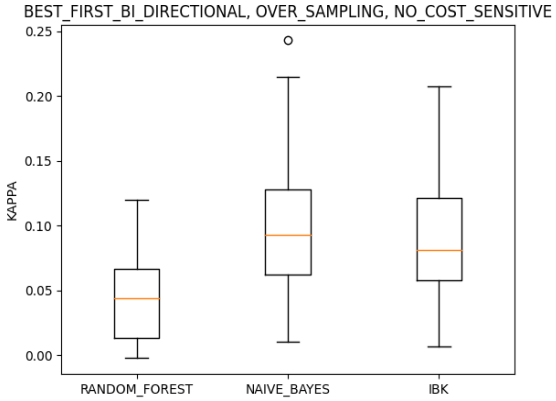


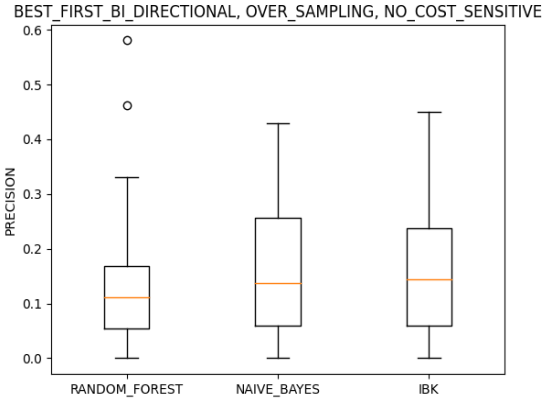


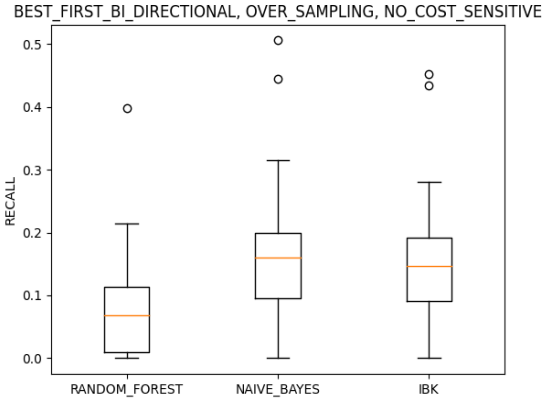


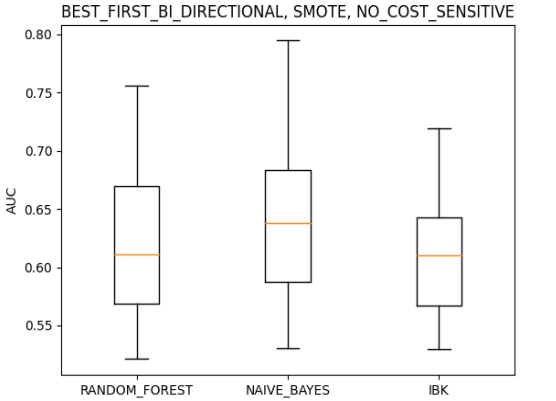
RISULTATI (Syncope – box plots [2])

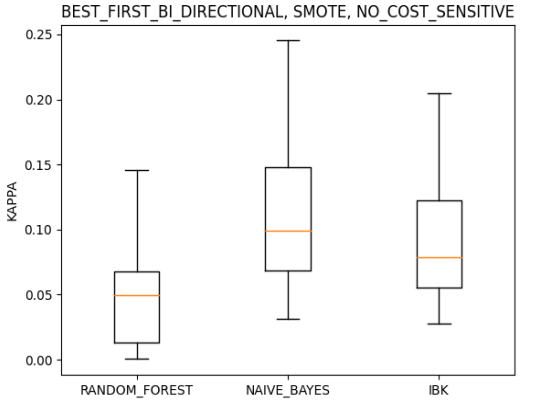


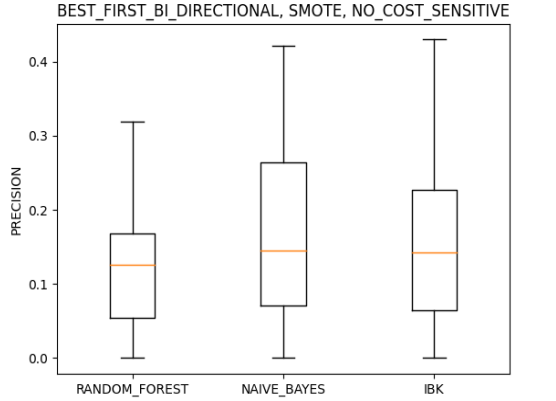


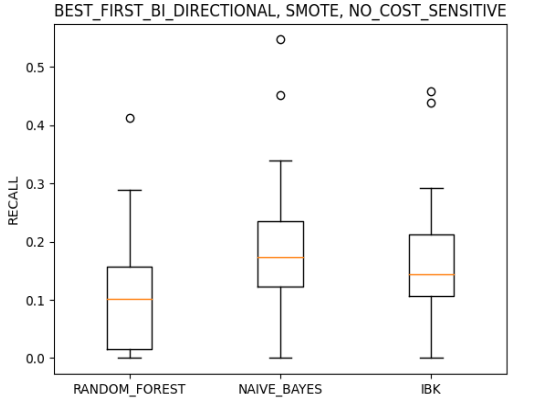




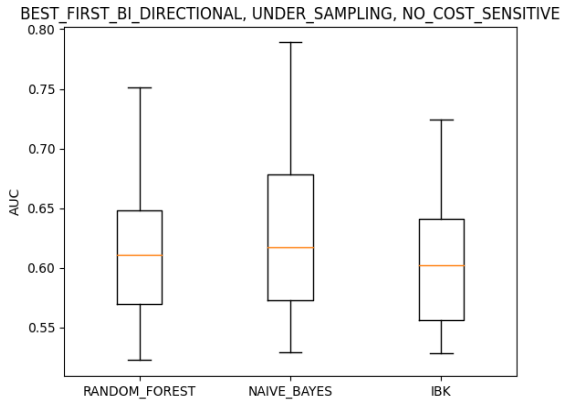


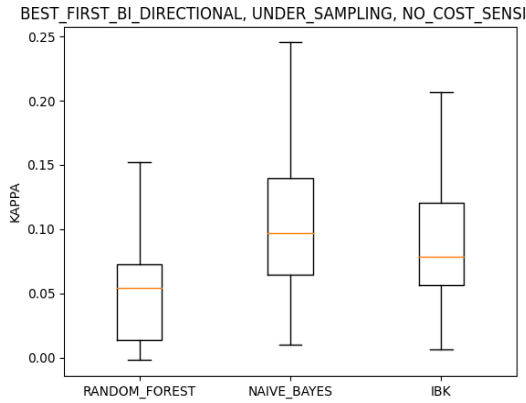


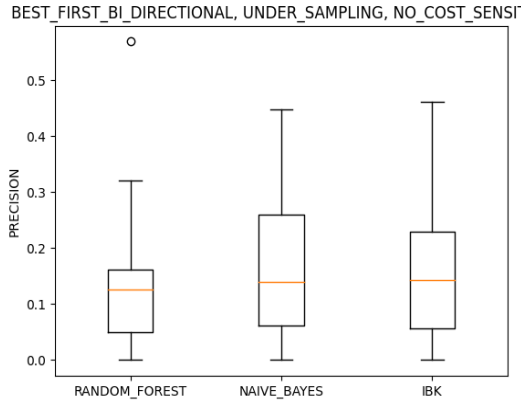


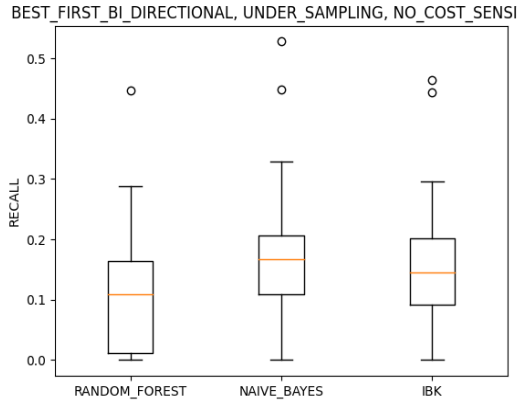


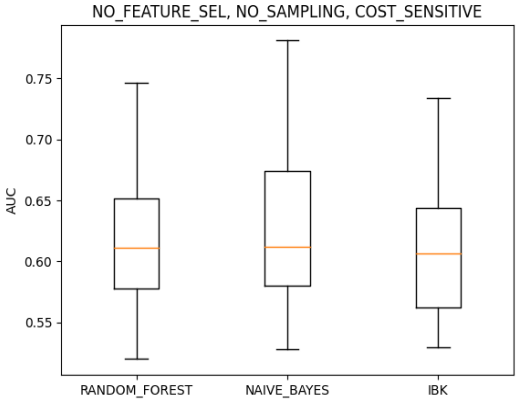
RISULTATI (Syncope – box plots [3])

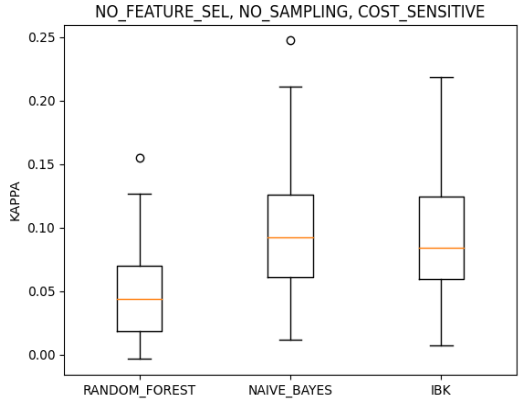


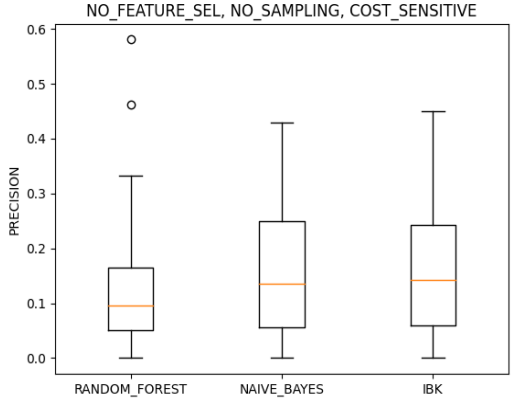


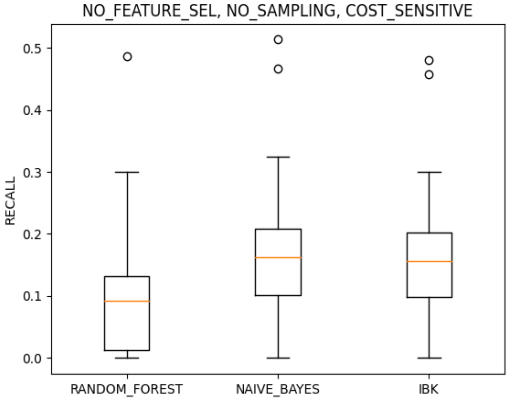




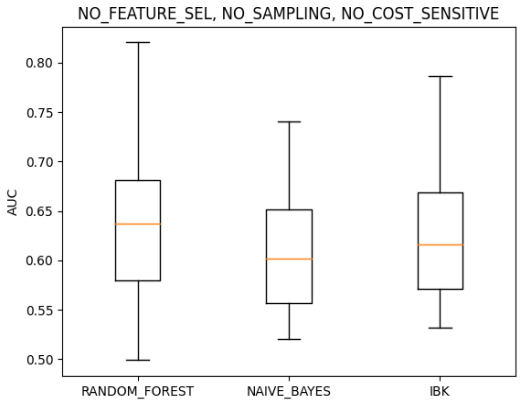


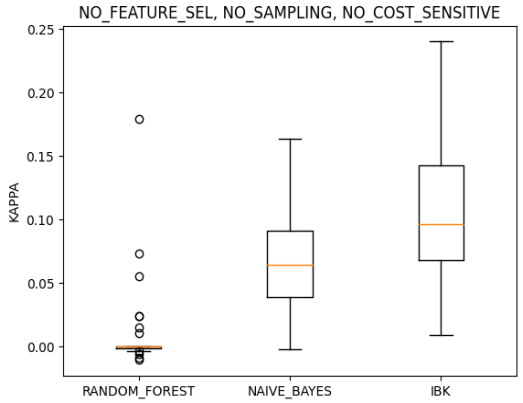


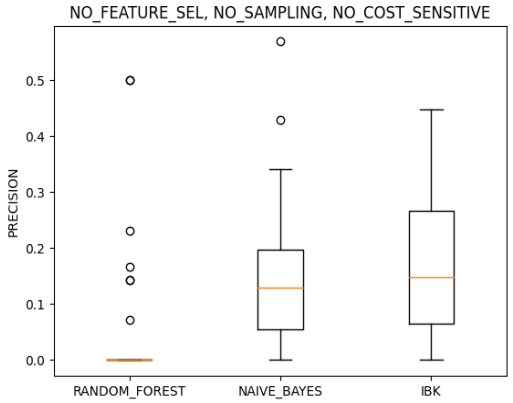


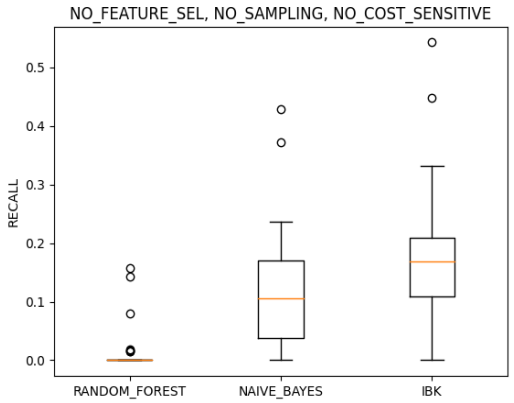


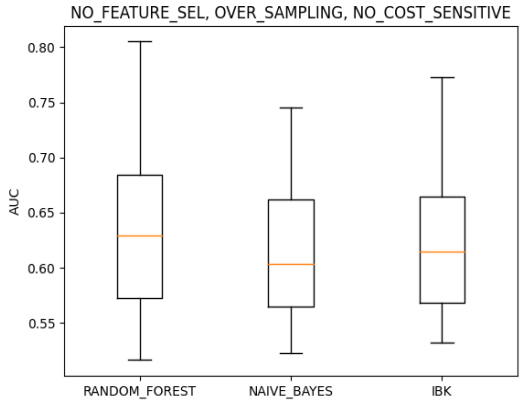
RISULTATI (Syncope – box plots [4])

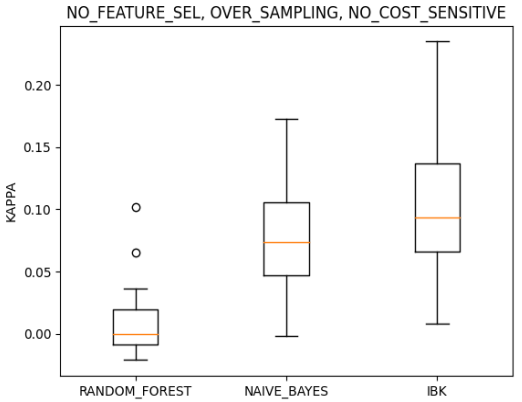


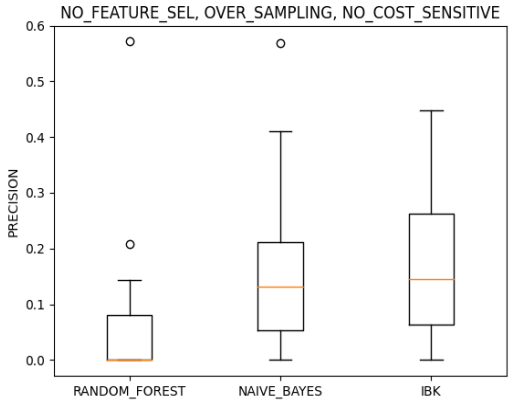


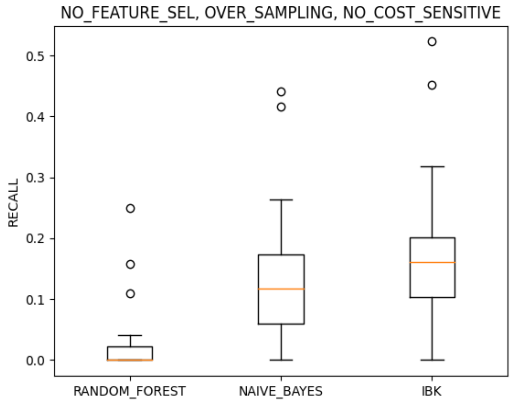




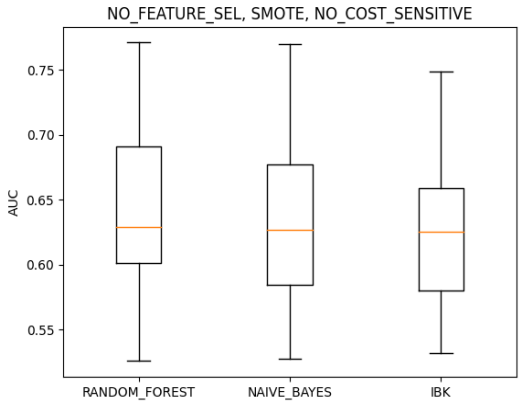


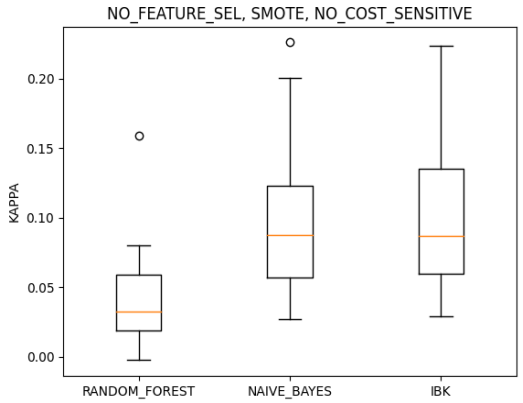


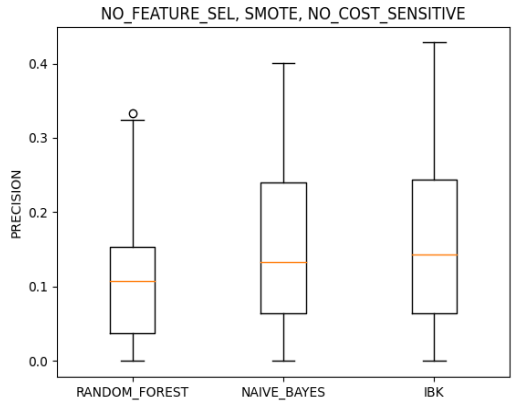


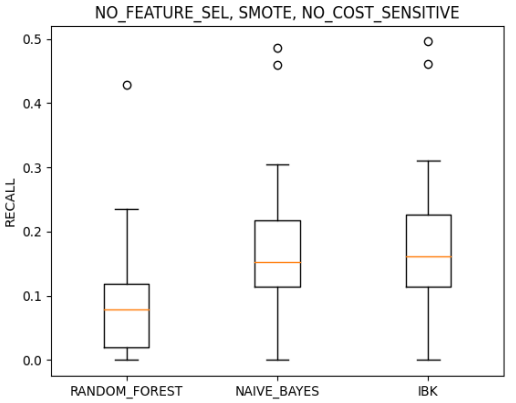


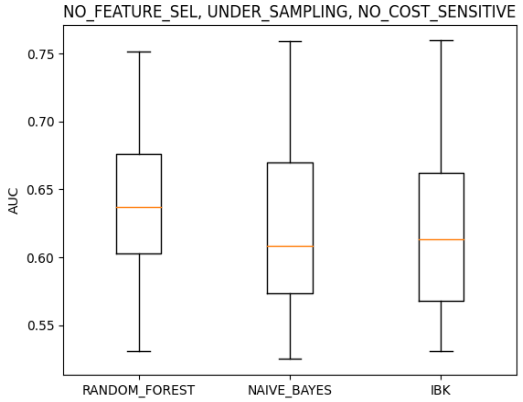
RISULTATI (Syncope – box plots [5])

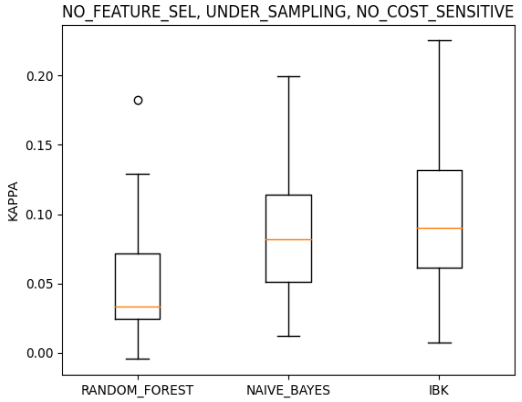


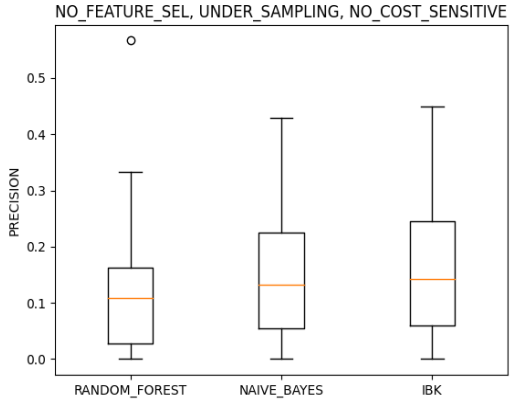


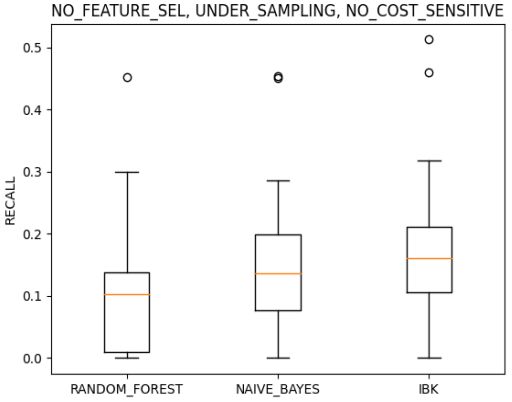












RISULTATI (considerazioni Syncope)

- Come per BokKeeper, non emerge un classificatore nettamente migliore rispetto agli altri in termini di performance. **Naive Bayes** però, anche nel caso di Syncope, evidenzia delle performance, in media, leggermente migliori. Ma comunque migliorie quasi trascurabili, soprattutto rispetto a IBK. Inoltre, il classificatore Random Forest risulta quello che si comporta peggio, sulle metriche Kappa, Precision e Recall.

- La presenza delle tecniche di balancing non manifesta comportamenti nettamente differenti in termini di performance. È interessante però notare come l'utilizzo di Over Sampling su **Random Forest**, rispetto ad un classificatore privo di tecniche di utilizzo, non abbia portato alcun tipo di beneficio, mentre **SMOTE** si.
    - Le istanze della classe minoritaria copiate mediante over sampling non erano particolarmente significative, mentre quelle sintetizzate con SMOTE hanno riscosso un successo sul training del classificatore.

CONCLUSIONI

- Dei **classificatori** esaminati non ce n'è uno particolarmente migliore degli altri a livello di performance. Pertanto, i risultati dipendono dal dataset preliminare e dalle tecniche applicate.
  - Il classificatore indicato in un contesto applicativo reale, dipende fortemente dalle esigenze dell'applicazione (e.g. in alcune circostanze potrebbe essere più importante la precision piuttosto che la recall).
  - Tenere sempre a mente i trade-offs.
- **Random Forest** però, presenta delle **performance peggiori** rispetto agli altri.
- Tra le varie **tecniche di bilanciamento** non se ne identifica una che si discosta nettamente dalle altre. Anche qui, i risultati variano in funzione del classificatore considerato e del dataset di partenza.
  - Una tendenza interessante è stato il miglioramento delle performance, in media, dovuto alla presenza della tecnica **SMOTE**. Questo indica che le istanze della classe minoritaria erano particolarmente significative se combinate con delle variazioni "sintentiche" introdotte.

CONCLUSIONI (take home message)

GitHub

Code:

https://github.com/0xMenna01/isw2-project

Evaluation plots:

https://github.com/0xMenna01/Evaluation-plots-ISW2

sonarcloud

https://sonarcloud.io/project/overview?id=0xMenna01_isw2-project