MACHINE LEARNING FOR SOFTWARE ENGINEERING

Ingegneria del Software 2 – Di Marco Luca 0333083

AGENDA

1. Introduzione
2. Obiettivo
3. Metodologia:
    1. Recupero delle metriche
    2. Costruzione del dataset
    3. Tecniche di utilizzo
4. Risultati:
    1. BookKeeper
    2. Syncope
5. Conclusioni
6. Minacce alla validità
7. Link

INTRODUZIONE

- L'operazione di testing all'interno dei progetti SW è un'operazione **dispendiosa** in termini di tempo e di costo monetario:
  - Svolgere le operazioni di testing in maniera **esaustiva** sull'intero progetto è oneroso e difficilmente applicabile.
- Riuscire a ridurre queste due variabili è cruciale per risparmiare risorse:
  - La **soluzione** può ricadere nella **predizione** delle future classi buggy mediante **informazioni passate**.

OBIETTIVO

- L'obiettivo è quello di rendere più **mirata** l'operazione di testing:
  - Per raggiungere questo obiettivo, nel progetto di studio:
    - Sono stati **applicati** diversi **modelli** di ML per ottenere predizioni sulla bugginess delle classi dei progetti sott'esame;
    - Si sono **analizzati** i risultati di questi modelli al variare di tecniche di addestramento.
- L'obiettivo è quello di trovare la **configurazione ottimale** per i due progetti Apache sott'esame: BookKeeper e Syncope.

METODOLOGIA

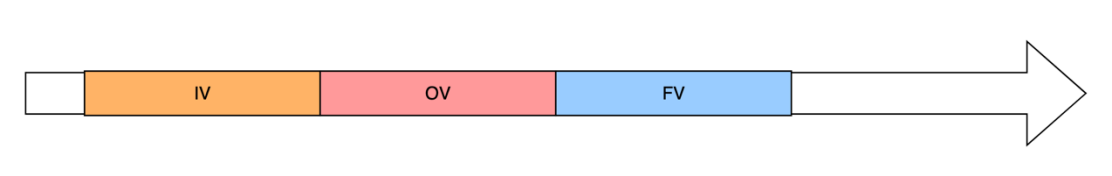
- Vengono raccolte le **metriche** dei seguenti **progetti**:
  - Apache BookKeeper;
  - Apache Syncope.
- Vengono costruiti i **dataset** di training e di testing set;
- Vengono applicati gli **algoritmi** di ML per eseguire le predizioni:
  - Utilizzo di 3 **modelli**: Random Forest, Naive Bayes, IBK.
  - Utilizzo di 3 **tecniche**: Feauture Selection, Sampling (Over e Under) e Cost Sensitivity.

METODOLOGIA – RECUPERO DELLE METRICHE

- Si introduce il concetto di »**ciclo di vita**» del bug: le classe definite affette sono quelle contenute tra IV (injected version) e FV (fixed version).

$$IV \leq \text{BUGGY} < FV$$

- La **timeline** completa dell'arco temporale di vita del bug è la seguente:



- Nello specifico:
  - **IV:** # della release dalla quale il bug viene introdotto;
  - **OV**: # della release dalla quale il bug viene rilevato;
  - **FV**: # della release nella quale il bug viene risolto e chiuso.

METODOLOGIA - RECUPERO DELLE METRICHE

- Queste informazioni di IV, OV ed FV vengono recuperate tramite issue sulla piattaforma **JIRA**:
  - **Problema**: non tutte le versioni su JIRA presentano dati riguardo le IV.
- Ove le IV non siano presenti, si ricorre al calcolo delle stesse mediante tecnica di **proportion**: mediante l'utilizzo del parametro di proporzionalità p, calcolato su ticket aventi informazioni sulla IV, è possibile stimare IV sui ticket che non lo presentano.

$$p = \frac{FV - IV}{FV - OV}, \text{ se FV=OV allora } p = \frac{FV - IV}{1}$$

$$IV = max\{1, FV - (FV - OV) * p\}$$

METODOLOGIA - RECUPERO DELLE METRICHE

- Nel progetto vengono utilizzate **due modalità** di calcolo nella proportion:
  - **Cold start**: modalità utilizzata quando non sono presenti abbastanza dati per calcolare una proportion nel progetto (i ticket presi sono meno della grandezza della finestra). La proportion viene calcolata utilizzando ticket di **progetti** correlati o **simili**.
  - **Moving Window**: modalità utilizzata per calcolare dinamicamente la proportion basandosi solo su una quantità limitata di ticket recenti (grandezza della window).

METODOLOGIA – COSTRUZIONE DEL DATASET

- La metà più recente delle release viene eliminata per ridurre il fenomeno dello **snoring**.
- Per ciascuna coppia release + classe vengono recuperate delle **metriche** (illustrate nella slide successiva) per permettere ai classificatori di effettuare le predizioni.
- Per ciascuna coppia release + classe viene inserita una label (come ultima colonna del dataset) che indica se la stessa era affetta da **bug** o meno, nella release specifica.

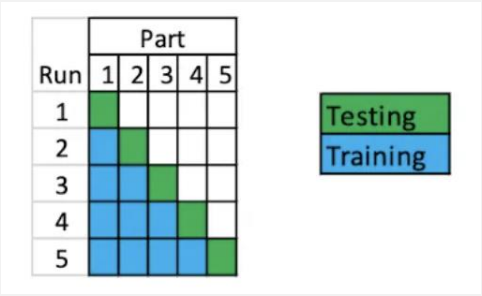
METODOLOGIA – COSTRUZIONE DEL DATASET

- Le **metriche (intra-release e non cumulative)** recuperate per ogni classe sono le seguenti.
- La metrica **BUGGY** viene calcolata nel seguente modo:
  - Vengono recuperati tutti i ticket di tipo «bug» con risoluzione **«fixed»** e con stato **«closed»** o **«resolved»** da JIRA;
  - Le classi **modificate** da un commit linkato ad un ticket estratto precedentemente, vengono **etichettate** come buggy per tutte le AV (quelle comprese tra IV ed FV).

LOC	size (lines of code)
LOC TOUCHED	somma delle LOC aggiunte e cancellate
NUMBER OF REVISIONS	numero delle revisioni sulla classe
LOC ADDED	LOC aggiunte
AVG LOC ADDED	media delle LOC aggiunte per revisione
NUM OF AUTHORS	numero di autori
MAX LOC ADDED	massimo delle LOC aggiunte sulle revisioni
TOTAL LOC REMOVED	LOC totali rimosse sulle revisioni
MAX LOC REMOVED	massimo delle LOC rimosse sulle revisioni
AVG LOC TOUCHED	media delle LOC aggiunte e cancellate sulle revisioni
BUGGY	classe buggy o non buggy

METODOLOGIA – COSTRUZIONE DEL DATASET

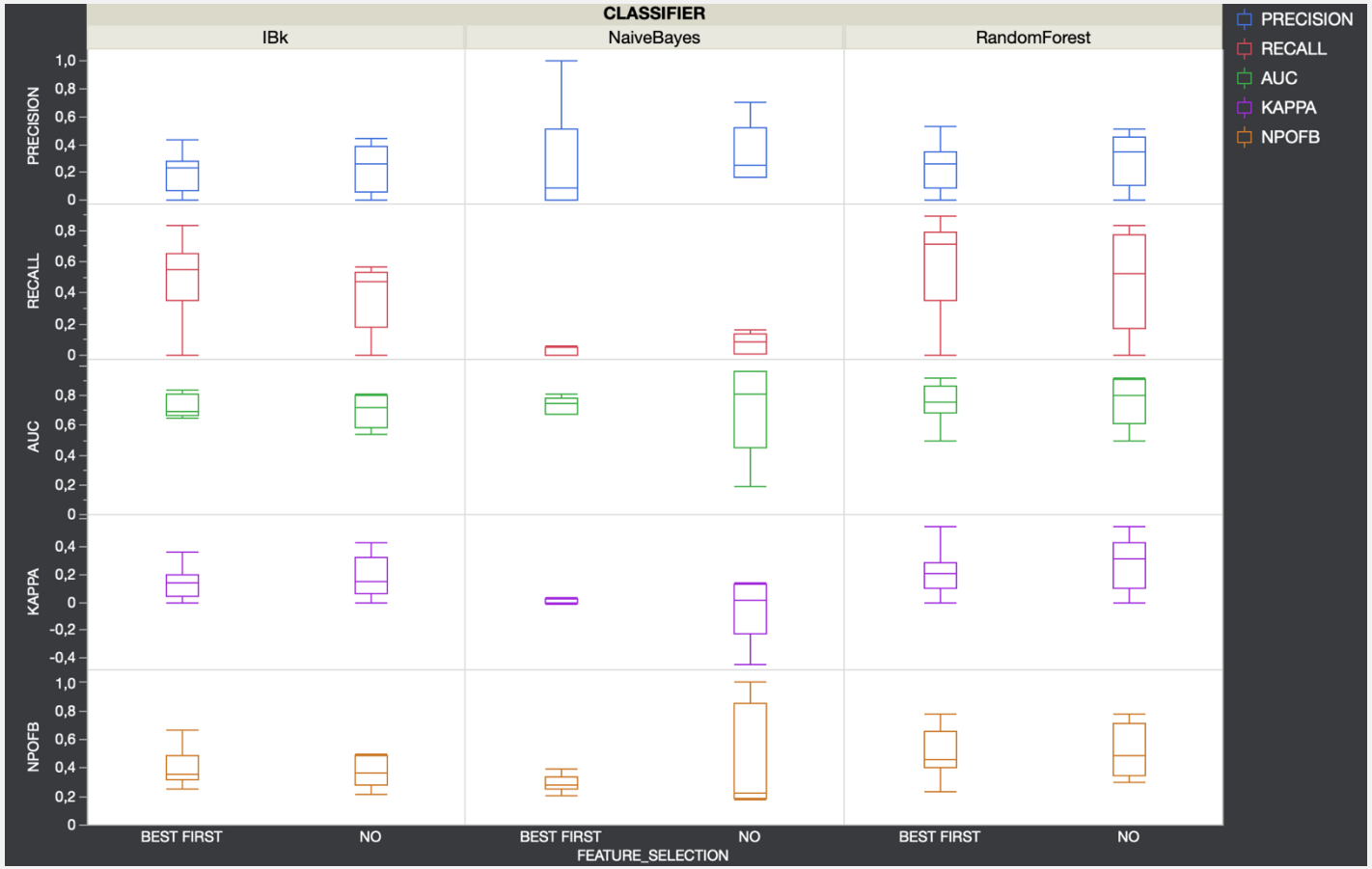
- Al fine di effettuare predizioni, i classificatori devono essere addestrati mediante training set;
- Al fine di raccogliere le statistiche dei classificatori, le predizioni devono essere verificate mediante testing set.
- La costruzione del training e del testing set avviene mediante tecnica di **Walk Forward**, modalità time-series ed iterativa che segue i seguenti step:
  - Il **training set** viene costruito mediante l'utilizzo delle **prime «n» release**, rieffettuando il labeling esclusivamente con le **informazioni disponibili fino a quel momento**;
  - Il **testing set** viene costruito, per ogni iterazione, con le informazioni presenti nella «n+1»-esima release, su cui andranno fatte le predizioni che avranno il labeling in base a **tutte le informazioni disponibili**.



METODOLOGIA – TECNICHE DI UTILIZZO

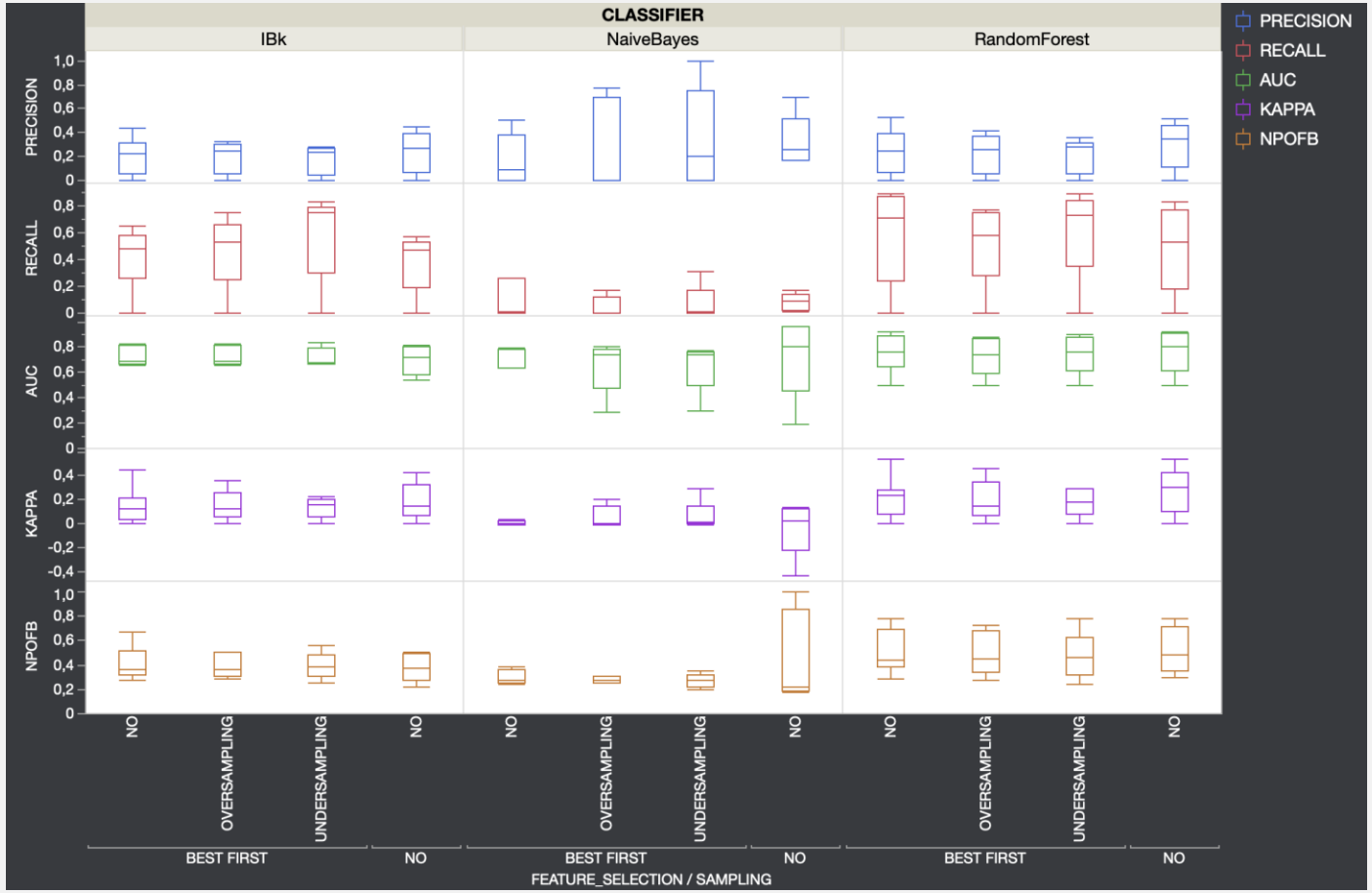
- Vengono utilizzati i seguenti classificatori e le seguenti configurazioni:
  - **Classificatori:**
    - Random Forest;
    - Naive Bayes;
    - IBK.
  - **Configurazioni:**
    - Nessun filtro presente;
    - Feature selection (Best First);
    - Feature selection (Best First) + Balancing (Under Sampling);
    - Feature selection (Best First) + Balancing (Over Sampling);
    - Feature selection (Best First) + Sensitive learning (CFN = 10*CFP).

RISULTATI – BOOKKEEPER
ESECUZIONE CON FEATURE SELECTION



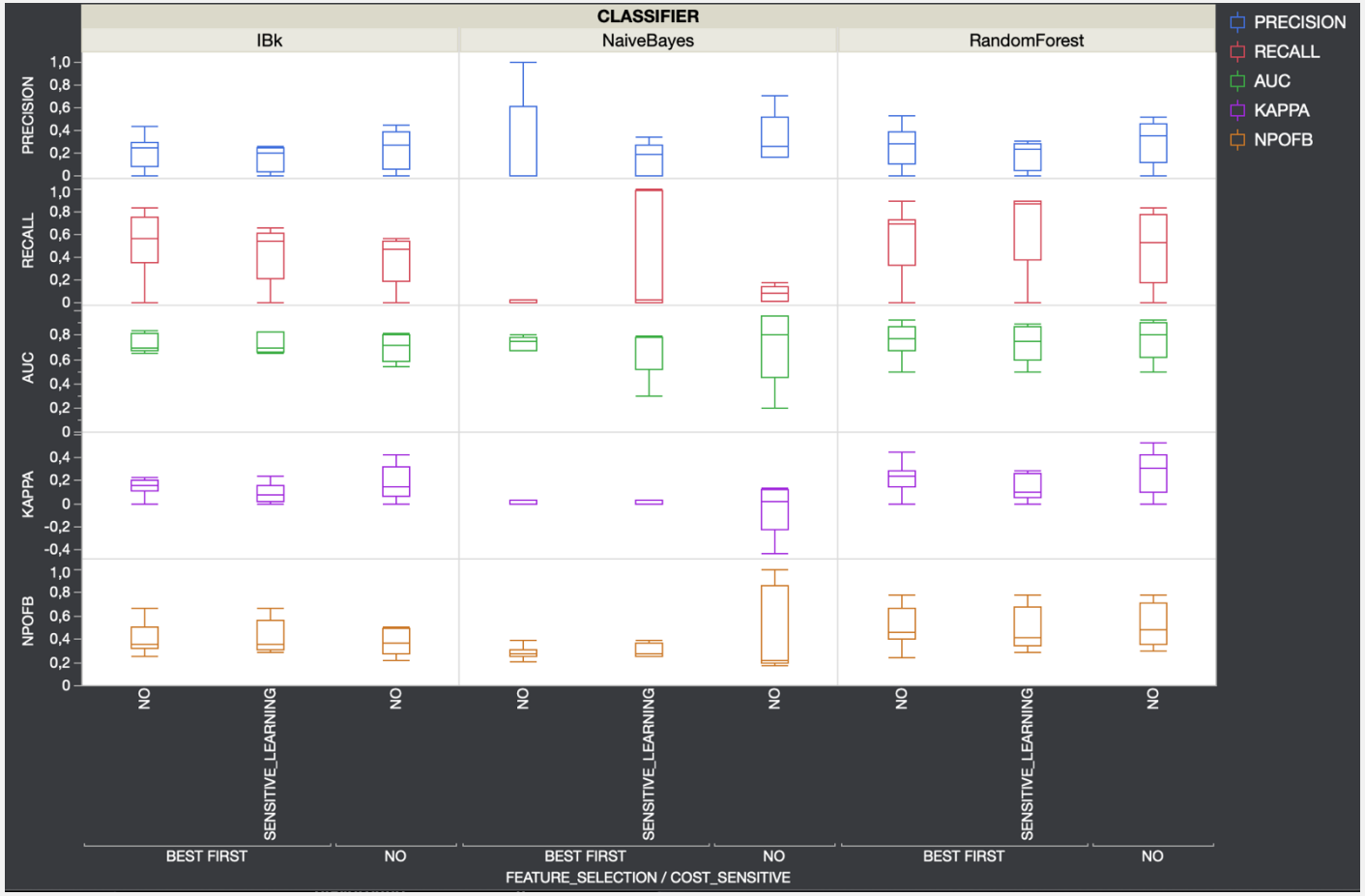
- Nelle esecuzioni con solo **Feature Selection**, rispetto alle esecuzioni senza alcun filtro, troviamo:
  - **IBk**: leggera riduzione della precision e miglioramento accentuato nella recall;
  - **Naive Bayes**: riduzione più accentuata della precision e della AUC. In generale, è il classificatore che si comporta peggio;
  - **Random Forest**: leggera riduzione della precision, ma sostanziale miglioramento nella recall.
- In generale, **Random Forest** è il classificatore **migliore**.

RISULTATI – BOOKKEEPER
ESECUZIONE CON FEATURE SELECTION E SAMPLING



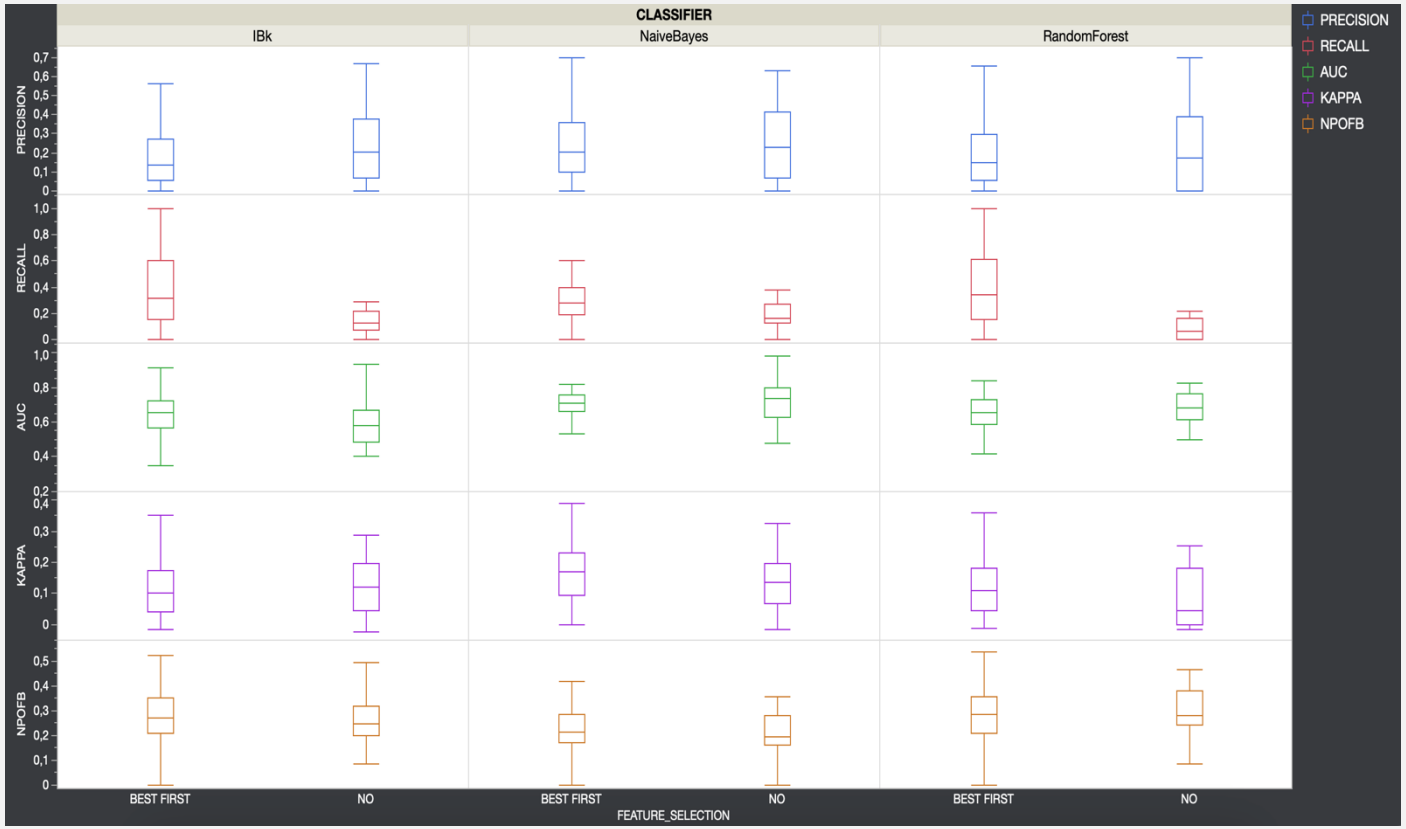
- Nelle esecuzioni con **Feature Selection e Over Sampling** troviamo:
  - **IBk**: miglioramento più accentuato della recall e miglioramento leggero della precision, dovuto all'aumento della percentuale di positivi nel training set;
  - **Naive Bayes**: risulta essere ancora il classificatore peggiore;
  - **Random Forest**: miglioramento della recall rispetto alle esecuzioni senza filtri, come atteso.
- Nelle esecuzioni con **Feature Selection** e **Under Sampling** troviamo:
  - **IBk**: miglioramento sostanziale della recall, dovuto alla riduzione dei negativi all'interno del training set;
  - **Naive Bayes**: migliora leggermente precision e recall ma rimane il classificatore peggiore;
  - **Random Forest**: miglioramento della recall e della precision.
- In generale, sia per Over Sampling che Under Sampling, **Random Forest** è il classificatore **migliore**.

RISULTATI – BOOKKEEPER
ESECUZIONE CON FEATURE SELECTION E SENSITIVE LEARNING



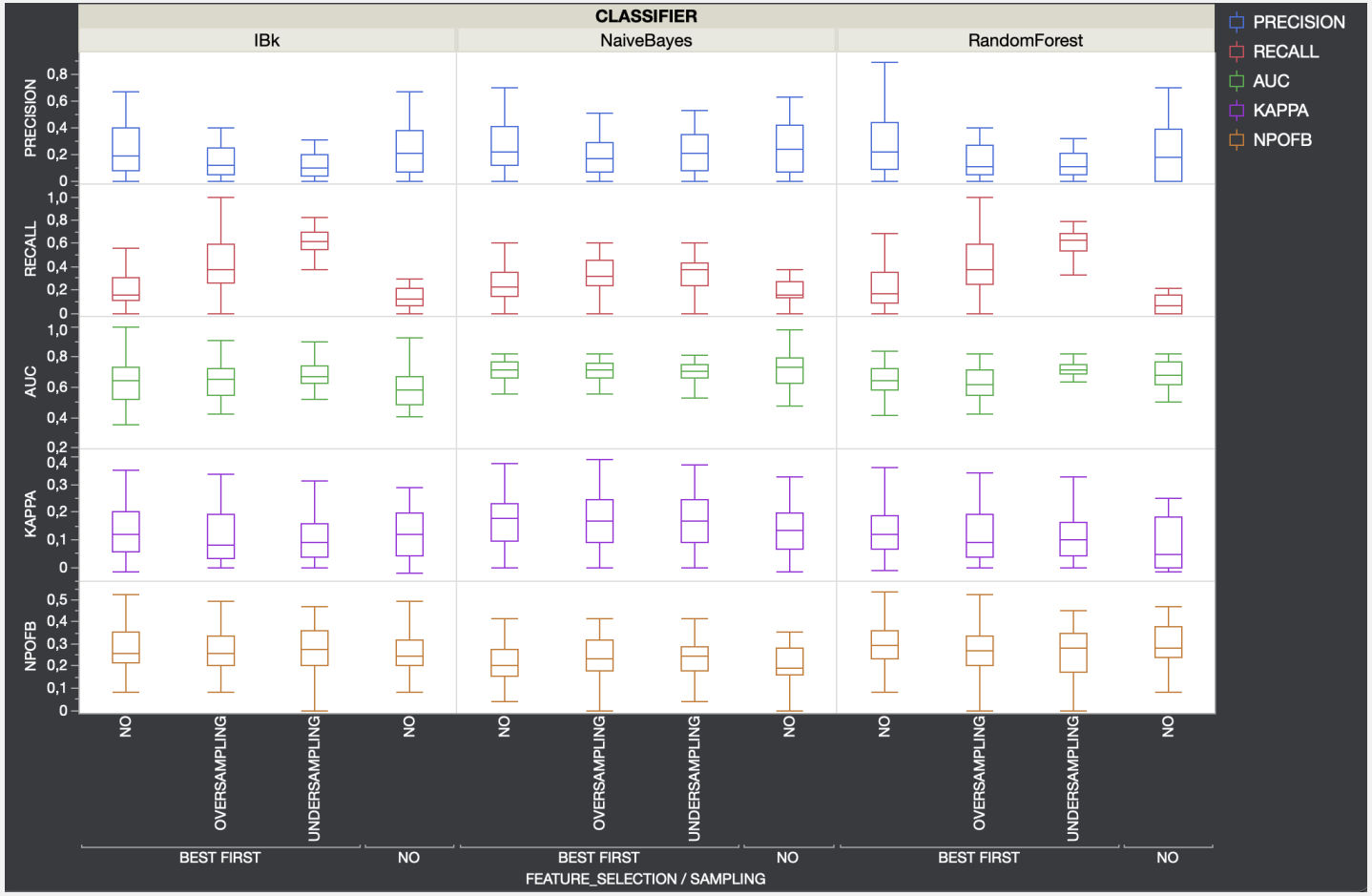
- Nelle esecuzioni con **Feature Selection e Sensitive Learning** troviamo:
  - **IBk**: un peggioramento della precision ed una recall quasi costante, con un leggero miglioramento, come atteso;
  - **Naive Bayes**: un leggero miglioramento della recall, ed un miglioramento della precision, ma risulta ancora essere il classificatore peggiore;
  - **Random Forest**: un leggero peggioramento della precision ed un miglioramento evidente della recall.
- **Random Forest** risulta essere ancora il classificatore **migliore**.

RISULTATI – SYNCOPE
ESECUZIONE CON FEATURE SELECTION



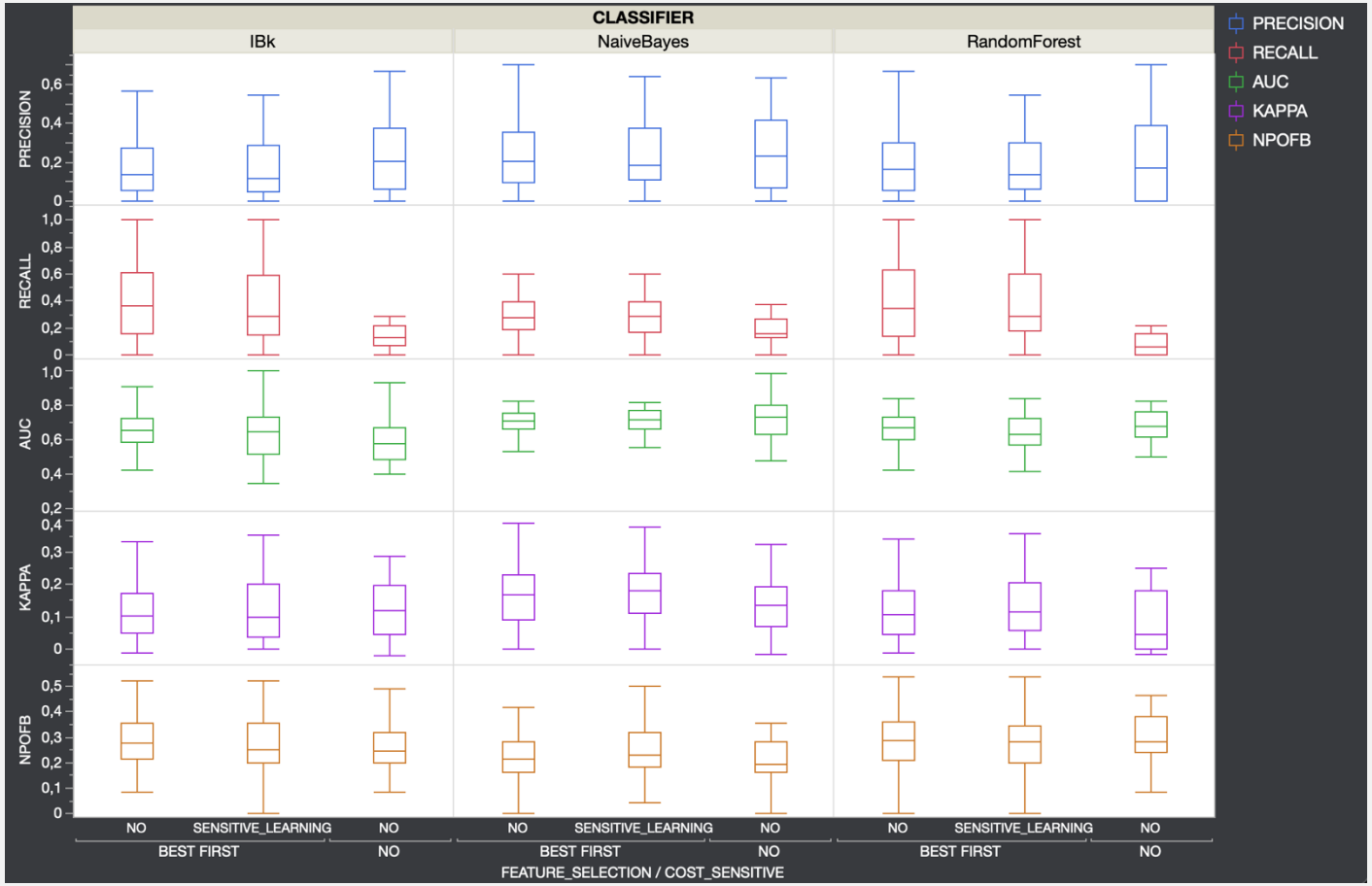
- Nelle esecuzioni con solo **Feature Selection**, rispetto alle esecuzioni senza alcun filtro, troviamo:
  - **IBk**: leggera diminuzione nella precision ma miglioramento nella recall;
  - **Naive Bayes**: leggera diminuzione nella precision ma leggero miglioramento nella recall
  - **Random Forest**: anche qui leggera diminuzione nella precision, ma miglioramento più accentuato nella recall.
- Attualmente, **Random Forest ed IBk** sembrano essere i classificatori **migliori**, con performance quasi simili.

RISULTATI – SYNCOPE
ESECUZIONE CON FEATURE SELECTION E SAMPLING



- Nelle esecuzioni con **Feature Selection e Over Sampling** troviamo:
  - **IBk**: una leggera diminuzione nella precision ma un sostanziale aumento nella recall, come atteso, dovuto all'aumentare della percentuale di positivi;
  - **Naive Bayes**: una leggera diminuzione nella precision, ma anche qui un sostanziale aumento nella recall;
  - **Random Forest**: una leggera diminuzione nella precision ma anche qui un sostanziale aumento nella recall. Le sue performance sono equiparabili a quelle di Ibk.
- Nelle esecuzioni con **Feature Selection** e **Under Sampling** troviamo:
  - **IBk**: un riduzione più accentuata nella precision ed un sostanziale aumento nella recall, come da attese, dovuto alla diminuzione della percentuale dei negativi;
  - **Naive Bayes**: come in IBk, diminuzione leggera nella precision ed un aumento nella recall più basso rispetto oversampling;
  - **Random Forest**: come in IBk, diminuzione più accentuata nella precision ma sostanziale aumento nella recall, come atteso.
- In generale, per Over Sampling **Naive Bayes** risulta essere il classificatore migliore, per Under Sampling **IBk** sembra essere il classificatore **migliore**.

RISULTATI – SYNCOPE
ESECUZIONE CON FEATURE SELECTION E SENSITIVE LEARNING



- Nelle esecuzioni con **Feature Selection e Sensitive Learning** troviamo:
  - **IBk**: precision e recall generalmente stabili rispetto la sola Feature Selection, con una leggera diminuzione della precision, più accentuata rispetto al caso senza filtraggio ed una leggera diminuzione anche nella recall;
  - **Naive Bayes**: diminuzione leggera nella precision ma leggero aumento nella recall, come atteso, soprattutto rispetto il caso senza filtraggio.
  - **Random Forest**: leggera diminuzione della precision e aumento nella recall, rispetto al caso senza filtraggio.
- In questo caso **Naive Bayes** sembra essere il classificatore **migliore**.

CONCLUSIONI

- In generale, non è possibile definire il classificatore migliore in **assoluto**. I dati variano molto dal dataset iniziale e alle varie tecniche applicate:
  - La configurazione migliore per **BookKeeper** risulta essere **Random Forest** nella configurazione **Feature Selection + Sensitive Learning**;
  - La configurazione migliore per **Syncope** risulta essere **Naive Bayes** nella configurazione **Feature Selection + Over Sampling**.
- I risultati ottenuti sono coerenti con la trattazione teorica, portando risultati attesi a seconda della variazione delle tecniche usate.
- Le tecniche di **Under Sampling**, in **dataset ridotti** come nel caso di BookKeeper, potrebbero portare ad uno sbilanciamento ed una **perdita di informazioni eccessiva**. Sono da preferire tecniche alternative, come Over Sampling oppure Sensitive Learning.

MINACCE ALLA VALIDITÀ

- L'operazione di **Cold Start** nel calcolo della Proportion si basa sull'assunzione che alcuni progetti Apache selezionati (ad esempio Avro, ZookKeeper per BookKeeper, Proton e OpenJPA per Syncope) siano simili ai progetti di studio.
- I **ticket** che **non presentano commit** associati vengono esclusi dal calcolo, poiché non forniscono informazioni.
- Le **informazioni** su JIRA, come le date di apertura, risoluzione e le versioni affette o fixed, vengono considerate **vere a priori**.
- Se un ticket presenta una lista di Affected Version (AV), viene considerata come Injected Version (IV) la **prima AV nella lista.**

LINK

- GITHUB:
  - https://github.com/lucamjdimarco/ISW2
- SONARCLOUD:
  - https://sonarcloud.io/summary/new_code?id=lucamjdimarco_ISW2&branch=main