ANALISI DELLE PRESTAZIONI DI CLASSIFICATORI NELLA PREDIZIONE DELLA DIFETTOSITÀ DEL CODICE

Enrico D’Alessandro (0306424)

A.A. 2022/2023

Università degli studi di Roma "Tor Vergata"

AGENDA

- Introduzione
- Progettazione
  - Ottenimento delle versioni dei progetti
  - Ottenimento bug relativi ai progetti
  - Proportion
  - Ottenimento dei commit effettuati sui progetti
  - Costruzione del dataset
  - Metriche
  - Evaluation
- Discussione dei risultati
  - BookKeeper (AUC, Kappa, Precision, Recall)
  - Syncope (AUC, Kappa, Precision, Recall)
- Conclusioni
  - Balancing
  - Classificatori

INTRODUZIONE

Obiettivo: eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di **feature selection** e **balancing** sull'andamento delle prestazioni di tre classificatori nel predire la difettosità di classi Java dei progetti Apache.

- I classificatori sono addestrati su un dataset che rappresenta la difettosità delle classi Java nei due progetti Apache **BookKeeper** e **Syncope**.
- I classificatori considerati sono **RandomForest**, **NaiveBayes** e **IBk**.
- La tecnica di feature selection considerata è **BestFirst**.
- La tecniche di balancing considerate sono **Oversampling**, **Undersampling** e **SMOTE**.

Per la realizzazione dello studio si è fatto uso di:

- **Java**, come linguaggio di programmazione;
- **GitHub API** e **JIRA Rest API** per la costruzione dei dataset per ciascun progetto;
- **WEKA** come toolkit di Machine Learning.

PROGETTAZIONE - OTTENIMENTO DELLE VERSIONI DEI PROGETTI

Per ottenere la lista delle **versioni** rilasciate per ciascun progetto è stata utilizzata la **RestAPI** di **JIRA**.

Dal JSON restituito, per ciascuna versione, sono stati estratti:

- **nome**;
- **data di rilascio**.

Le informazioni così ottenute sono state utilizzate per popolare una lista ordinata in base alla seconda delle due informazioni estratte.

Sono state individuate:

- 14 versioni per **BookKeeper**
- 63 versioni per **Syncope**

PROGETTAZIONE - OTTENIMENTO BUG RELATIVI AI PROGETTI (1)

La lista dei **bug** relativi ai due progetti è stata ottenuta sfruttando l'API JSON di JIRA. In particolare, è stato specificato di voler unicamente gli issue di tipo:

- **Bug**;
- la cui risoluzione sia **fixed**;
- il cui stato sia **resolved** o **closed**.

Per ogni ticket restituito dall'ITS:

- come **FV** è stata considerata l'ultima delle *fixed versions* restituite in ordine temporale;
- come **OV** è stata considerata la prima successiva alla data di *creazione* del ticket;
- come **IV** è stata considerata la prima delle *affected versions* restituite in ordine temporale.

PROGETTAZIONE - OTTENIMENTO BUG RELATIVI AI PROGETTI (2)

Una volta ottenuta la lista è stato necessario sanificarla rimuovendo tutti i bug tali per cui:

- la **FV** non corrisponde ad alcuna versione realmente esistente;
- l'ordinamento **FV** ≥ **OV** non è rispettato.

In questa fase i **bug** per cui l'**IV** è successiva all'**OV** vengono mantenuti non considerando l'*injected version* fornita da JIRA, al fine di impostarla in seguito con il metodo **proportion**.

In definitiva, ai fini dello studio, sono stati presi in considerazione:

- circa il 92.18% dei bug per **BookKeeper**;
- circa l'88.92% dei bug per **Syncope**.

PROGETTAZIONE - PROPORTION

Non per tutti i bug riportati in JIRA è disponibile l'insieme delle *affected versions* e questo è problematico ai fini dello studio, in quanto limita la possibilità di etichettare le classi come **buggy** in determinate versioni.

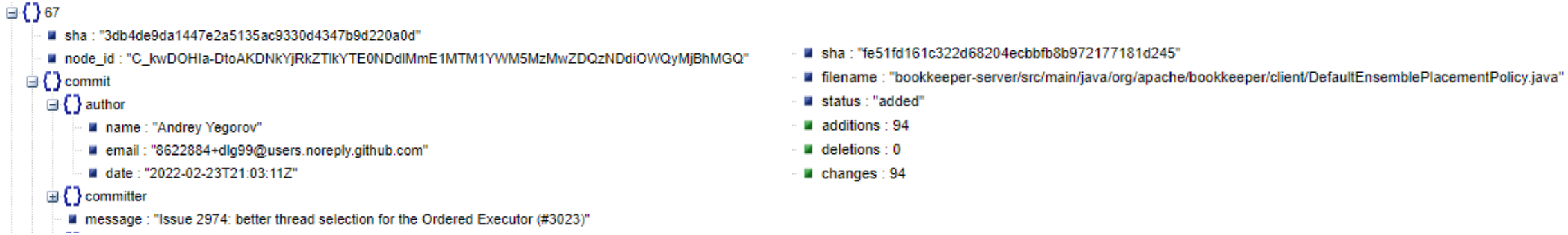
Per ovviare a ciò è stata utilizzata la tecnica **proportion** nella sua variante **incrementale**. In particolare, per ciascun bug presente nella lista ottenuta al passo precedente, in caso di assenza della **IV**:

1. è stato calcolato **P** come media dei rapporti $\frac{\mathbf{FV - IV}}{\mathbf{FV - OV}}$ per tutti i bug *fixed* nelle versioni precedenti;
2. è stata calcolata la **IV** come **IV = FV - (FV - OV) · P**.

PROGETTAZIONE - OTTENIMENTO DEI COMMIT EFFETTUATI SUI PROGETTI

Per l'integrazione con **Git** è stata utilizzata la **GitHub REST API**:

- possibilità di effettuare al più **5000** richieste autenticate all'ora;
- implementato un meccanismo di caching in quanto i commit sono oggetti **immutabili**;
- per ogni commit dal JSON di risposta sono stati estratti lo **SHA**, l'**autore**, la **data**, il **messaggio** ed infine il **diff**:
  - Il **diff** non è altro che un array di oggetti JSON, ciascuno dei quali contenente il **nome** dell'i-esimo file toccato dal commit ed il **numero di righe aggiunte** e/o **rimosse**.



PROGETTAZIONE – COSTRUZIONE DEL DATASET

Jira + git

Prima di procedere all'addestramento dei classificatori è necessario costruire il dataset da utilizzare. L'associazione fra le informazioni raccolte da **Git** e **JIRA** è realizzata come segue:

- per ogni **bug B (key)** e per ogni **commit C**, se il messaggio di **C** contiene la chiave di **B**, i file presenti nel **diff** vengono aggiunti all'insieme di quelli associati a **B**;
- per le **versioni** semplicemente tramite l'ordinamento temporale tra le date.

Sono state individuate:

- 884 su 4290 (21%) istanze buggy per **BookKeeper**;
- 6032 su 121015 (5%) istanze buggy per **Syncope**.

Una volta costruito il dataset, sono stati rimossi i dati relativi all'ultimo 50% delle release in modo da ridurre l'effetto dello **snoring**.

Il dataset è costituito da diverse colonne che specificano una **versione**, un file **java**, 11 **metriche** (correlate alla presenza o meno di bug nel file e nella versione considerati) e un'**etichetta** "Y/N" per specificare se il file è risultato o meno difettoso all'interno della versione.

PROGETTAZIONE - METRICHE (1)

Jira + git

Le principali metriche considerate sono le seguenti:

- **Size:** lo storico della dimensione del file in termine di LOC.
  - Un valore alto di linee di codice potrebbe significare un rischio maggiore di avere bug.
- **NRev:** il numero di revisioni al file nella singola versione.
  - Maggiori sono i commit che hanno interessato quel file, maggiori sono le possibilità di introdurre bug.
- **NAuth:** il numero di autori totali che hanno collaborato al file nella versione considerata.
  - Con un valore elevato di autori aumentano le probabilità di generare difetti nel codice a causa di incomprensioni e delle differenti intenzioni tra i collaboratori.
- **LOC added:** il numero di LOC aggiunte in quella determinata versione.
  - Simile alla Size. Più LOC sono state aggiunte al file e maggiore è la probabilità di avere introdotto un bug.
- **Churn:** è la somma sulle revisioni di LOC (added – deleted) per la singola versione.
  - Un valore elevato rappresenta il LOC effettivo che potrebbe contribuire ad introdurre un bug nella classe.

PROGETTAZIONE - METRICHE (2)

Jira + git

Le principali metriche considerate sono le seguenti:

- **ChgSetSize:** è il numero di file che sono stati modificati da commit insieme al file specificato.
  - Più sono i file modificati insieme in un certo commit e più sarà difficile identificare la natura del bug eventualmente generato.
- **Age:** è l'età della classe calcolata in termini di settimane.
  - Una classe con valore di età maggiore risulta più stabile in termini di difettosità e meno propensa ad avere bug.

Sono state considerate anche le seguenti metriche:

- **MAX LOC added**
- **AVG LOC added**
- **MAX Churn**
- **AVG Churn**

PROGETTAZIONE - EVALUATION

- Al termine della fase di costruzione del dataset, questo viene memorizzato in due formati differenti: **CSV** e **ARFF**. Quest'ultimo viene sfruttato come base di partenza per la valutazione oggetto dello studio tramite il toolkit **WEKA**.

- Per valutare la difettosità dei progetti nel corso delle release, è stata utilizzata la tecnica di validazione **walk-forward**, dal momento che i dati sono strettamente legati ad aspetti temporali.
  - Ad ogni iterazione, il dataset iniziale viene diviso in training set e testing set, considerando una release come testing e tutte le precedenti vengono incluse nel training set.

- Per ogni configurazione possibile (i.e., tecnica di selezione delle feature, tecnica di balancing) sono state eseguite $\mathbf{N - 1}$ run di **walk-forward**, con **N** numero di versioni presenti nel dataset, ed è stato popolato un **CSV** contenente i risultati finali dello studio, sfruttato per produrre i grafici mostrati nelle slide successive.

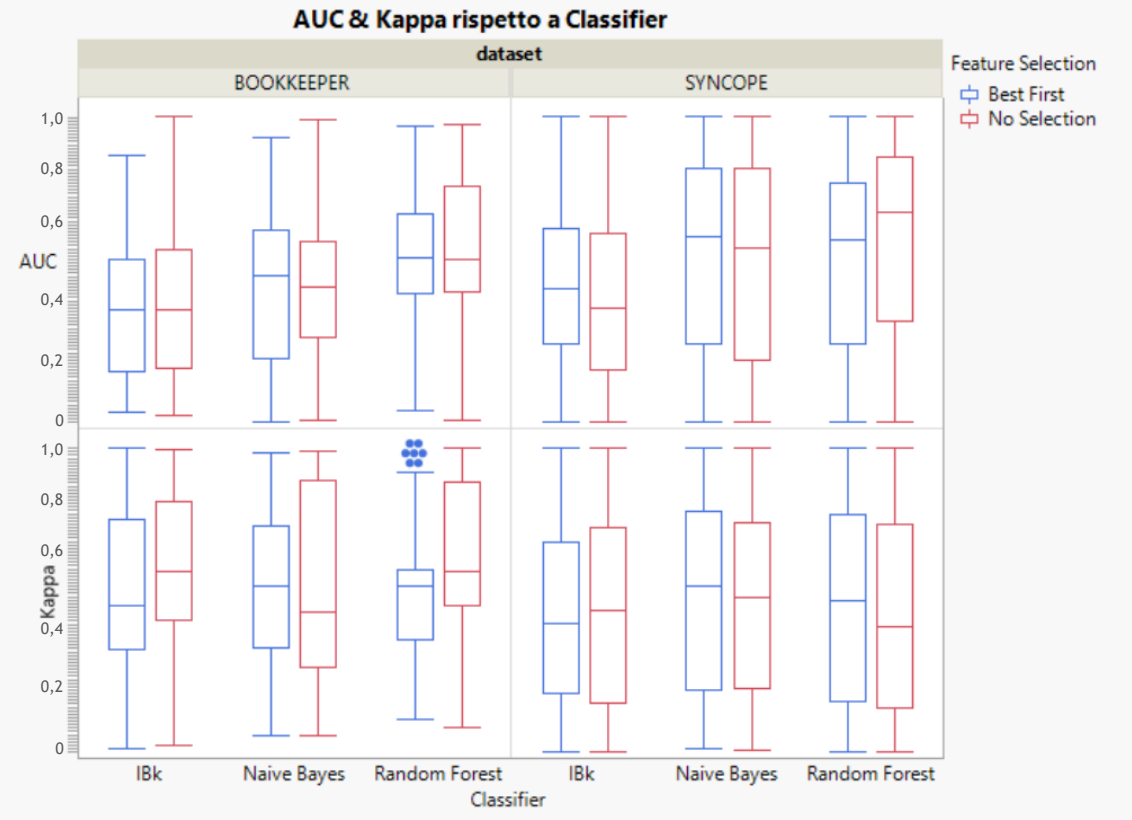
DISCUSSIONE DEI RISULTATI

Per ogni classificatore e tecnica di **feature selection** e **balancing** sono state analizzate:

- **AUC**
- **Kappa**
- **Precision**
- **Recall**

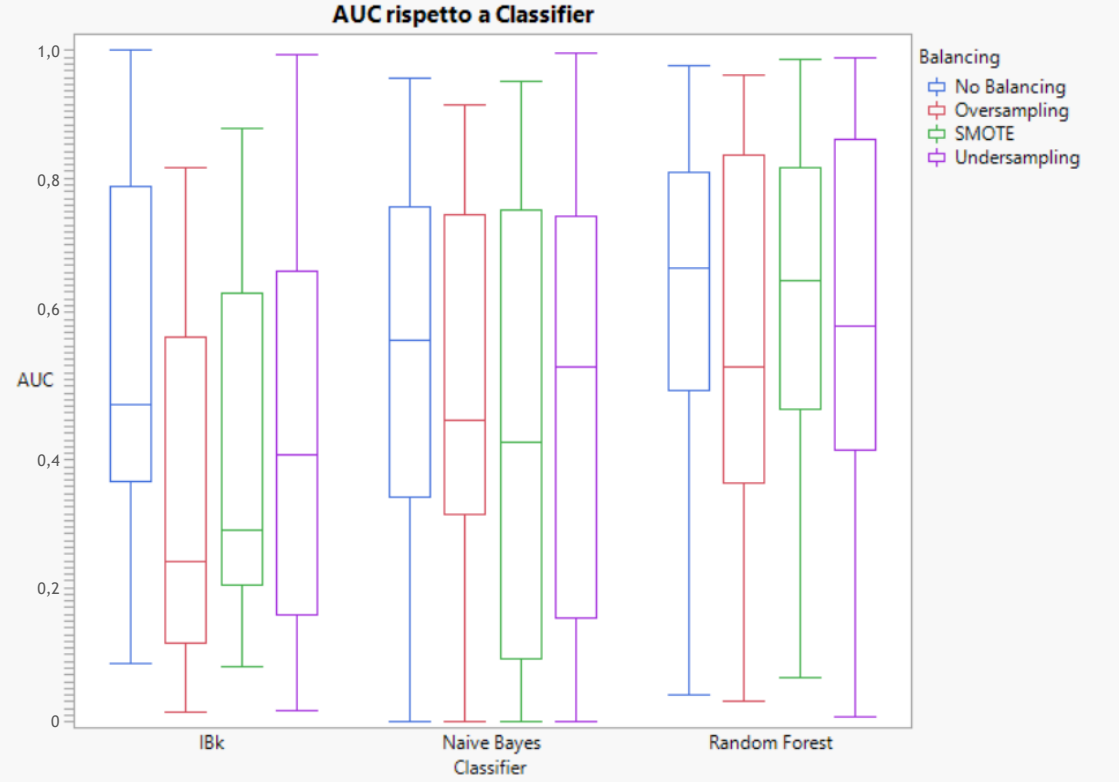
Per confrontare le prestazioni dei classificatori, poiché il tipo di grafico adottato è il **box-plot**, sono state prese in considerazione la mediana, l'ampiezza della distribuzione e la posizione dei quantili principali (i.e., 1Q e 3Q).

DISCUSSIONE DEI RISULTATI - FEATURE SELECTION



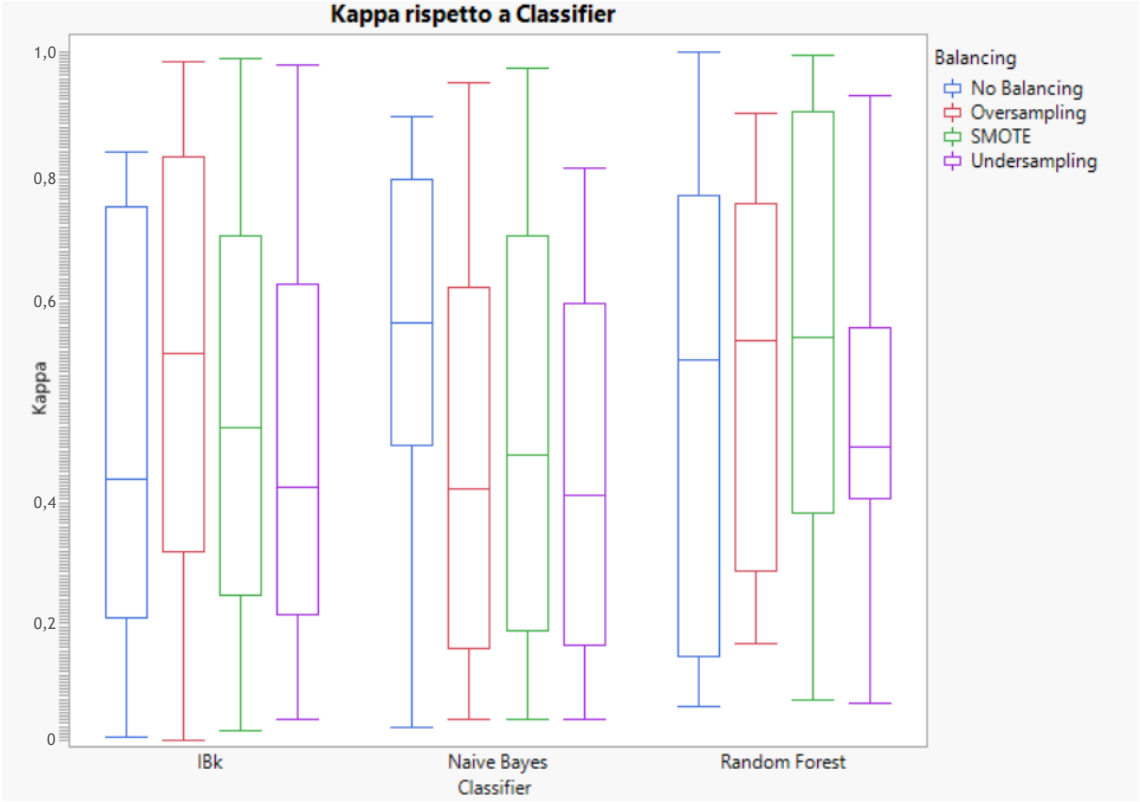
- È interessante notare come, in alcuni casi, applicare la tecnica **feature selection** migliora abbastanza le prestazioni:
  - Applicando **BestFirst** sul dataset di **Syncope** si ottengono miglioramenti per **NaiveBayes** e per **IBk**.
- Il classificatore che migliora sempre applicando **BestFirst** per entrambi i dataset è **NaiveBayes.** Questo potrebbe essere dovuto al fatto che **NaiveBayes** assume che gli attributi del dataset siano il più possibile scorrelati tra loro.

DISCUSSIONE DEI RISULTATI - BOOKKEEPER (AUC)



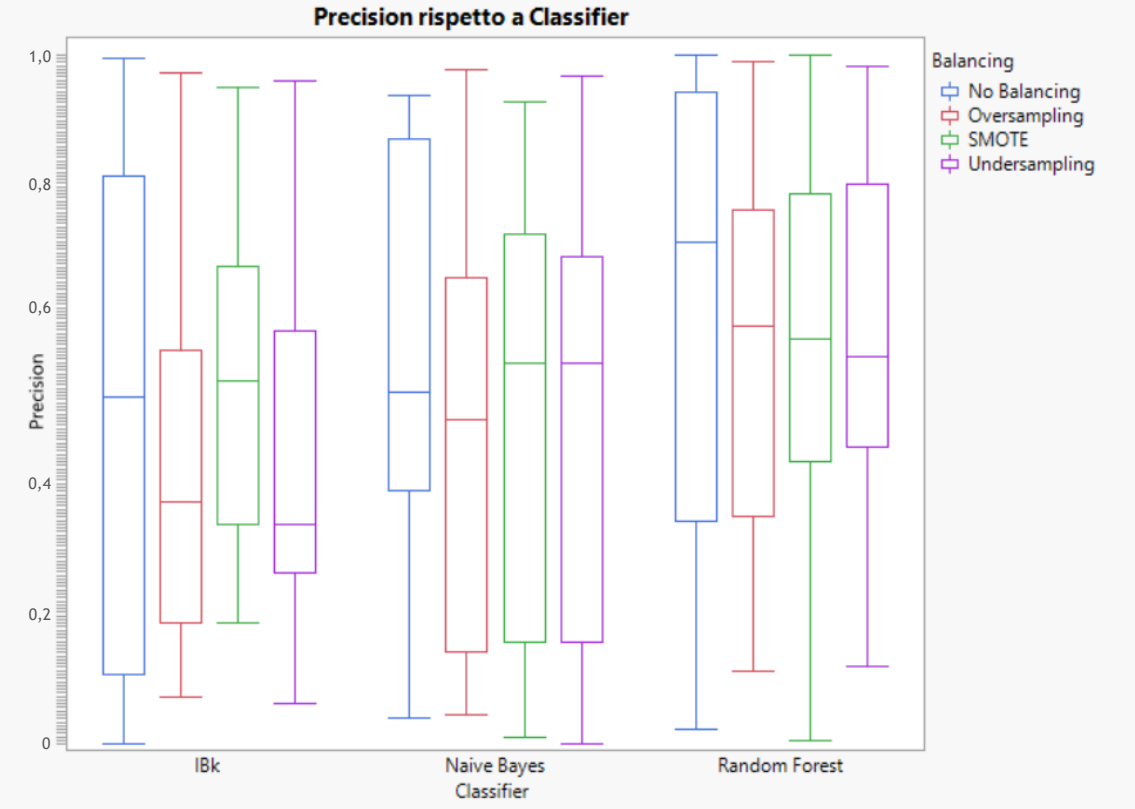
- Si considerano i dati del progetto **BookKeeper**.
- La tecnica **Undersampling** è quella che peggiora di meno le prestazioni dei classificatori **IBk** e **NaiveBayes**.
- La tecnica **Oversampling** peggiora notevolmente le prestazioni di tutti i classificatori rispetto a quando non vengono applicate tecniche di bilanciamento.
- La tecnica **SMOTE** peggiora le prestazioni di tutti i classificatori, maggiormente in **IBk** e **NaiveBayes**.

DISCUSSIONE DEI RISULTATI - BOOKKEEPER (KAPPA)



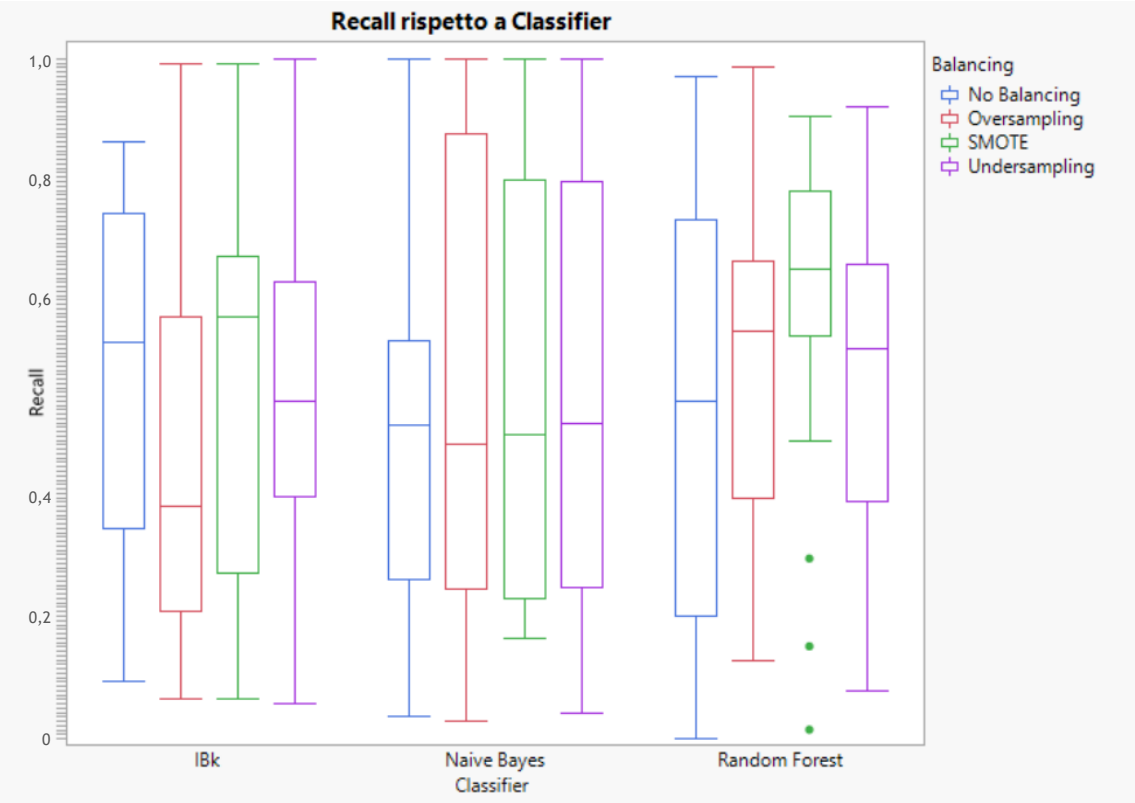
- Si considerano i dati del progetto **BookKeeper**.
- Le prestazioni di **NaiveBayes** peggiorano parecchio rispetto ad un classificatore "*dummy*", applicando qualsiasi tecnica di bilanciamento sul dataset in input.
- Le prestazioni di **RandomForest** migliorano leggermente applicando il bilanciamento con **SMOTE** e peggiorano applicando **Undersampling**.
- Nel caso di **IBk** applicare **Oversampling** migliora parecchio le prestazioni, mentre **Undersampling** sembra influire di meno.

DISCUSSIONE DEI RISULTATI - BOOKKEEPER (PRECISION)



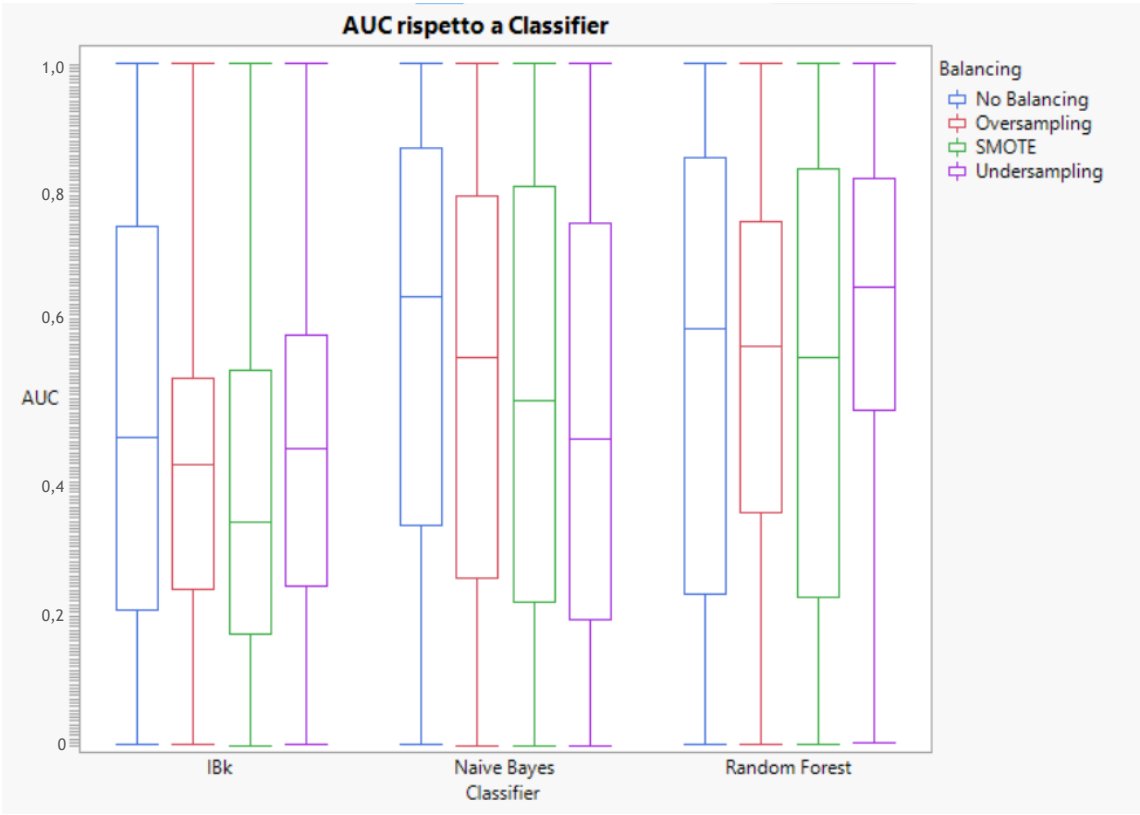
- Si considerano i dati del progetto **BookKeeper**.
- **SMOTE** e **Undersampling** aumentano leggermente la precision nel caso di **NaiveBayes**.
- La precision peggiora notevolmente applicando **Undersampling** ai classificatori **RandomForest** e **IBk**. Il maggior peggioramento si ha nel caso di **IBk**.
- Il bilanciamento ha in generale un impatto minore sulla precision valutata con **NaiveBayes** rispetto agli altri classificatori.
- **SMOTE** aumenta leggermente la precision nel caso di **IBk**.

DISCUSSIONE DEI RISULTATI - BOOKKEEPER (RECALL)



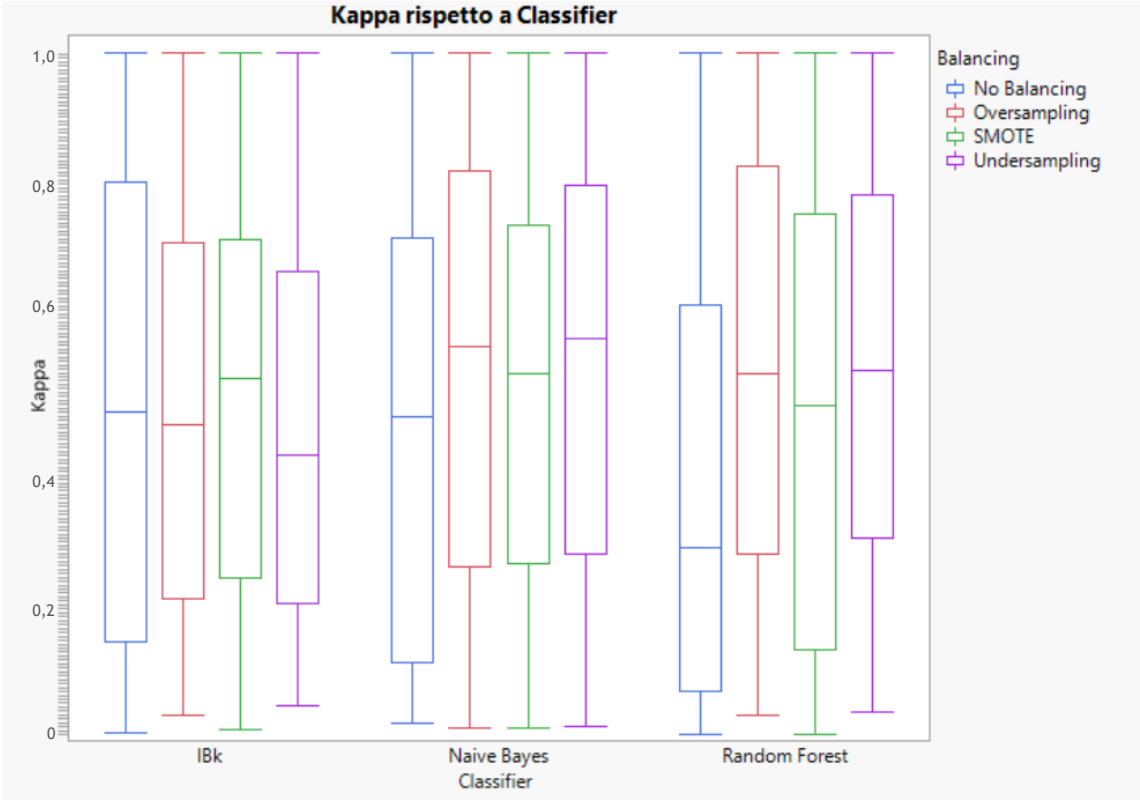
- Si considerano i dati del progetto **BookKeeper**.
- Nel caso di **RandomForest**, tutte e tre le tecniche aumentano il numero di positivi individuati e quindi la recall. Non solo si hanno valori mediani più alti, ma anche la variabilità della distribuzione è minore.
- Nel caso di **IBk** e **RandomForest** si ottengono più positivi individuati applicando **SMOTE**, sebbene la sua influenza sia più evidente per il secondo.
- In **IBk** applicare **Oversampling** o **Undersampling** riduce il numero di positivi individuati.

DISCUSSIONE DEI RISULTATI - SYNCOPE (AUC)



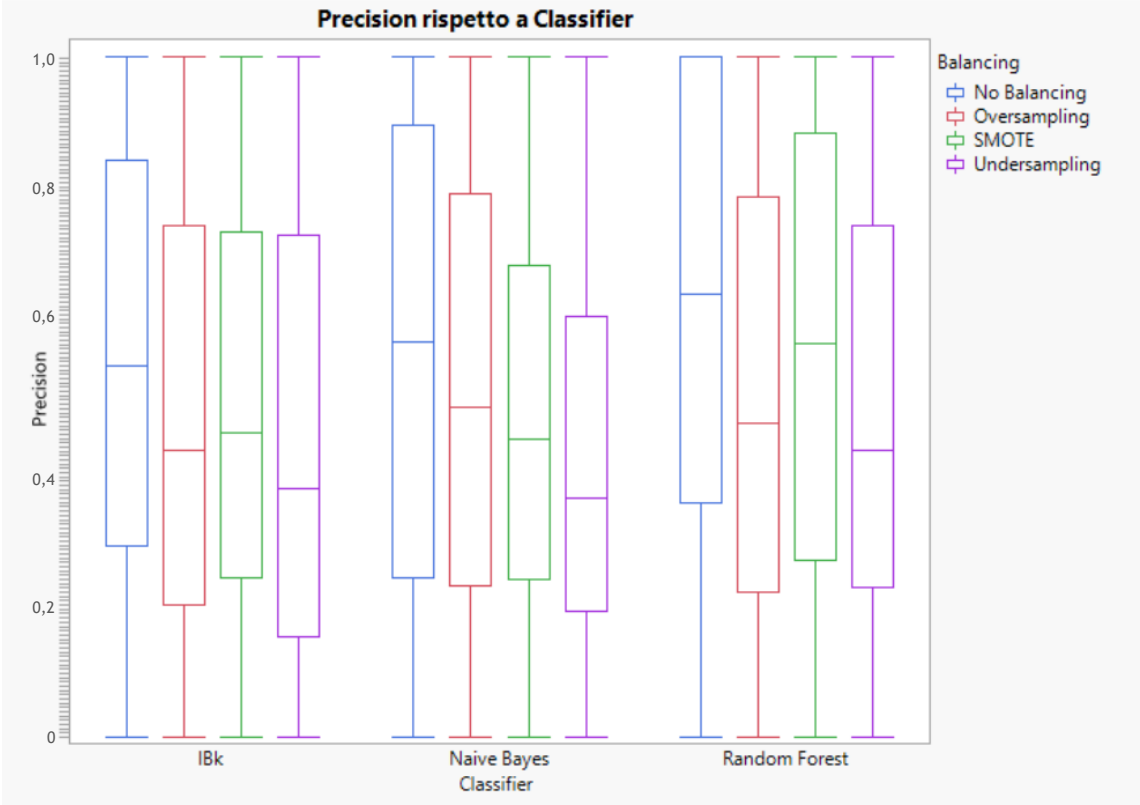
- Si considerano i dati del progetto **Syncope**.
- Sia per **IBk** che per **NaiveBayes** applicare tecniche di bilanciamento peggiora in tutti i casi le prestazioni.
- Per quanto riguarda **RandomForest** si ottiene un leggero peggioramento del valore di AUC con **Oversampling** e **SMOTE**, mentre si ottiene un miglioramento applicando **Undersampling**.
- Il peggioramento maggiore si ottiene con **NaiveBayes** applicando **Undersampling**.

DISCUSSIONE DEI RISULTATI - SYNCOPE (KAPPA)



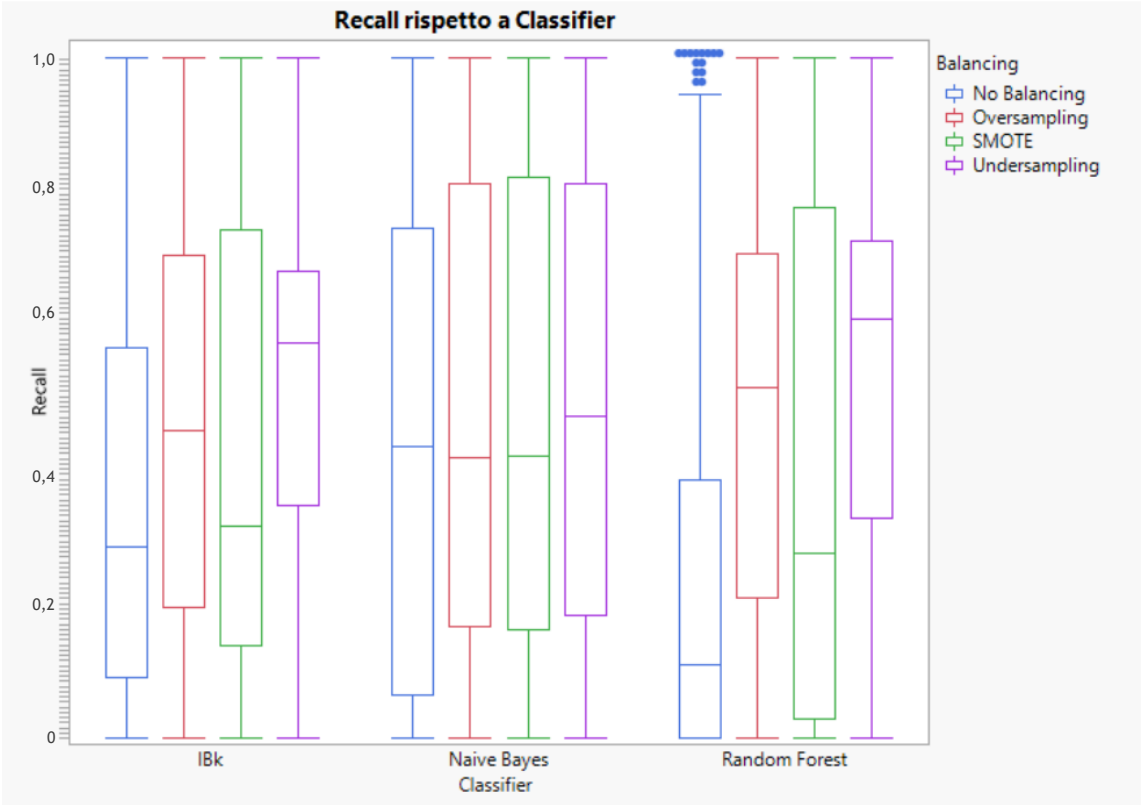
- Si considerano i dati del progetto **Syncope**.
- In questo caso applicando le tre tecniche di bilanciamento del dataset si ottengono aumenti delle prestazioni dei tre classificatori rispetto al classificatore "*dummy*". In maniera inferiore per **IBk**.
- Per **NaiveBayes** e **RandomForest** tutte e tre le tecniche di bilanciamento migliorano notevolmente.
- Per **IBk** l'unico miglioramento si ottiene applicando **SMOTE**.

DISCUSSIONE DEI RISULTATI - SYNCOPE (PRECISION)



- Si considerano i dati del progetto **Syncope**.
- In questo caso il valore di precision diminuisce applicando qualsiasi tecnica di bilanciamento del dataset per tutti i classificatori considerati.
- La tecnica che provoca il peggioramento maggiore del valore della precision è **Undersampling** per tutti e tre i classificatori.
- La tecnica con la precision maggiore è **SMOTE** per **IBk** e **RandomForest**, **Oversampling** per **NaiveBayes**.

DISCUSSIONE DEI RISULTATI - SYNCOPE (RECALL)



- Si considerano i dati del progetto **Syncope**.
- Sia per **RandomForest** che per **IBk** applicare tecniche di bilanciamento aumenta notevolmente il valore di recall.
- Per **NaiveBayes** applicare qualsiasi tecnica di bilanciamento tende a non influire più tanto sul valore di recall.
- Per quanto riguarda la recall, le tecniche di **Undersampling** e **Oversampling** sembrano comportarsi generalmente meglio rispetto a **SMOTE**, per tutti e tre i classificatori.

CONCLUSIONI – BALANCING

- Tra le tecniche di bilanciamento non ce n'è una che è nettamente migliore delle altre, ma dipende intrinsecamente dal dataset di partenza e dal classificatore considerato.

- Su **Syncope** si ottiene un numero maggiore di positivi individuati applicando **Undersampling**, mentre su **BookKeeper** si ottengono più positivi applicando **SMOTE**.

- Confrontando i valori di recall senza applicare le tecniche di bilanciamento è evidente come essa sia maggiore sul dataset di **BookKeeper**, poiché esso è meno sbilanciato rispetto al dataset di **Syncope**:
  - Buggy BookKeeper: 21%
  - Buggy Syncope: 5%

- Per questo motivo il bilanciamento ha un impatto mediamente maggiore su **Syncope** rispetto a **BookKeeper** a prescindere dal classificatore.

CONCLUSIONI - CLASSIFICATORI

- Tra i diversi classificatori non ce n'è uno che ha performance migliori degli altri, ma anche in questo caso i risultati variano in base al dataset iniziale e alle varie tecniche applicate.

- Il classificatore che in generale ha le performance più basse è proprio **NaiveBayes**, lo stesso classificatore su cui hanno meno impatto le tecniche di bilanciamento, nonostante la sostanziale differenza di bilanciamento dei dataset iniziali.

- Performance leggermente migliori si ottengono con **RandomForest**.

- In conclusione, dal momento che è stato usato **walk-forward** come tecnica di validazione, è possibile che ci siano iterazioni particolarmente sfortunate che pesano negativamente sulle prestazioni dei vari classificatori.

GRAZIE PER L'ATTENZIONE!

Link al repository GitHub:

ISW2-Milestone1:
https://github.com/EnricoDAlessandro97UNI/ISW2-Milestone1

ISW2-Milestone2:

https://github.com/EnricoDAlessandro97UNI/ISW2-Milestone2

Link a SonarCloud:

ISW2-Milestone1:
https://sonarcloud.io/project/overview?id=EnricoDAlessandro97UNI_ISW2-Milestone1

ISW2-Milestone2:

https://sonarcloud.io/project/overview?id=EnricoDAlessandro97UNI_ISW2-Milestone2