MACHINE LEARNING FOR SOFTWARE ANALYTICS
INGEGNERIA DEL SOFTWARE II

Petrolo Melissa

0286160

Università degli Studi di Roma ‘Tor Vergata’

INDICE

INTRODUZIONE

Obiettivo: raccogliere metriche per ***stimare*** la ***difettosità*** di una ***classe*** tramite l'utilizzo di tre classificatori al variare di tecniche di balancing, tecniche sensibili al costo e feature selection.

In questo studio sono stati analizzati due progetti opensource: *Apache Bookkeeper* e *Apache Syncope*.

Nel progetto verranno identificate le classi più predisposte ad essere difettose e successivamente, verranno utilizzate dei modelli di machine learning per predire quali file abbiano maggiore predisposizione nell'insorgere di bug, in base a delle metriche calcolate.

Si cercherà, di identificare quali tecniche di balacing, feature selection e sensitive permettono di avere un'accuratezza e i risultati migliori.

INTRODUZIONE

Verranno considerati:

- ❑ *RandomForest, NaiveBayes e IBK* come classificatori;
- ❑ *No Sampling, Oversampling, Undersampling e Smote* come tecniche di balancing;
- ❑ *No Selection, Best First* come tecniche di feature selection;
- ❑ *No Cost Sensitive, Sensitive Threshold, Sensitive Learning* come tecniche di sensitive cost.

Per la creazione del dataset, vengono presi in considerazione i ticket presenti su Jira di tipo bug per ogni progetto. Jira è un Issue Tracking System, ossia un software che permette di monitorare ticket relativi a bug issue.

PROGETTAZIONE

Prima di procedere all'addestramento dei classificatori, è necessario costruire il dataset.

Il dataset comprende:

- Una versione;
- Un file java;
- 13 diverse metriche;
- Etichetta yes/no che determina se il file è risultato difettoso nella versione corrente.

Per recuperare le informazioni sul progetto sono stati utilizzati:

- Jira: per ottenere informazioni relativi ai ticket e le versioni, sfruttando la sua API;
- Git: per ottenere informazioni relativi ai commit del repository, tramite API Git.

PROGETTAZIONE-JIRA

- Per ottenere la lista dei ticket abbiamo specificato, attraverso l'API JSON di Jira, di voler ottenere gli issue di tipo:
  - Bug;
  - Fixed;
  - Resolved o closed;
- Le informazioni prese da Jira, riguardanti i ticket, sono:
  - ID Ticket: id che identifica il ticket;
  - Creation Date: data di creazione del ticket;
  - Resolution Date: data di risoluzione del ticket;
  - Injected Version (IV): indica la versione in cui è stato introdotto il bug;
  - Opening version (OV): indica la versione rilasciata a seguito della creazione del ticket,
  - Fixed Version (FV): indica la versione rilasciata a seguito della chiusura del ticket, ovvero alla risoluzione del bug;
  - Affected Version (AV): indica l'insieme delle versione affette dal bug.

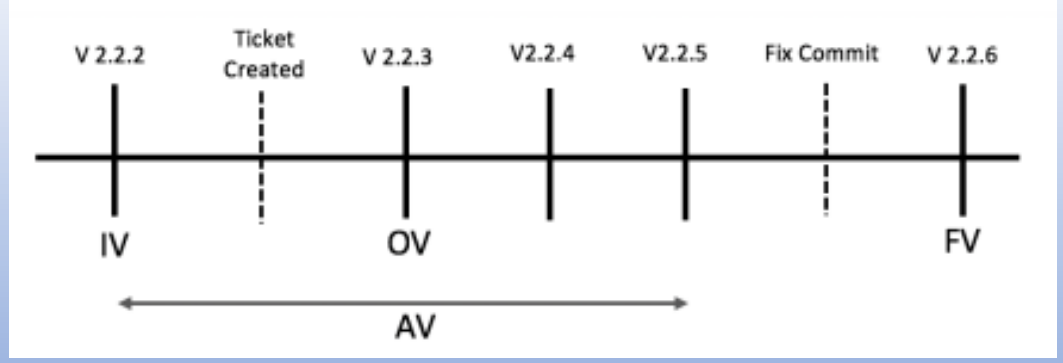
PROGETTAZIONE-VERSIONI

La data di creazione e risoluzione del ticket risultano essere sempre presenti su Jira e risultano affidabili; al contrario dell'AV che non sono sempre disponibili e affidabili.

- OV: viene generata al momento della creazione del ticket, quindi è la data di creazione presente su Jira;
- FV: versione rilasciata a seguito dell'ultimo commit che risolve il bug contenente l'ID, ed è la data di risoluzione del ticket presente su Jira;

Per i ticket con AV, comprese tra IV e FV esclusa, la stima del suo valore si traduce nella stima dell'IV.

- Per stimare IV quando non è presente, è stato utilizzato il metodo **Proportion**.
- la proporzione P viene calcolata come: P= (FV –IV)/(FV−OV)
- IV predetto viene calcolata come: IV = FV – (FV – OV) * P
- Tra le varie tecniche per il calcolo di P, in questo caso si è utilizzato **Increment ,** che consiste nel calcolo della media tra difetti risolti nelle versioni precedenti e nel caso in cui non fossero presenti valori prima di quella release si considera il metodo ***ColdStar***t.



PROGETTAZIONE-BUGGYNESS

La buggyness è una metrica che classifica una classe come difettosa o meno in una determinata release.

Per determinare la difettosità di una classe:

- ❑ Sono stati raccolti tutti i ticketi da Jira contenenti tutte le informazioni necessarie;
- ❑ Sono stati identificati i commit che fanno riferimento ad almeno uno di questi ticket;
- ❑ Se per il ticket non esiste una AV, è stato applicato il metodo Proportion per il calcolo della IV;
- ❑ Per ogni ticket, si è verificata la consistenza delle AV;
- ❑ Stimata la difettosità di una classe modificata dal commit che fa riferimento ad almeno un ticket con IV pari al valore della release corrispondente.

PROGETTAZIONE-METRICHE

Le principali metriche prese in considerazione sono:

- Size: dimensione del file in termini di LOC;
- LOC_touched: la somma delle LOC aggiunte e rimosse nella versione corrente;
- NR: numero di revisioni del file nella singola versione;
- Nfix: numero di bug risolti nella versione corrente;
- Nauth: numero di autori che hanno contribuito alla revisione del file;
- LOC_added: numero di LOC aggiunte;

PROGETTAZIONE-METRICHE

- MAX_LOC_added: numero massimo di LOC aggiunti per revisione;
- AVG_LOC_added: numero medio di LOC aggiunti per revisione;
- Churn: somma sulle revisione di LOC aggiunti – eliminati per singola versione;
- MAX_churn: numero massimo churn per revisione;
- AVG:churn: numero medio churn per revisione;
- Age: età della classe espresso in settimane;
- WeightedAge: età pesata della classe sui LOC_touched.

PROGETTAZIONE - MACHINE LEARNING

In questa fase, si prende in input il dataset precedente per valutare la difettosità del progetto, attraverso i tre classificatori: *Random Forest, IBK, Naive Bayes*.

La tecnica utilizzata è il **Walk-forward** poiché i dati sono legati ad aspetti temporali. Il dataset viene diviso in due parti, dove per ogni esecuzione, tutti i dati disponibili prima della parte da predire è usata come training set e la parta da predire è usata come testing set.

Il modello di accuratezza è calcolato come media tra le diverse esecuzioni.

Per la valutazione del progetto è stata utilizzata l'API fornito da Weka.

RISULTATI

Per valutare le migliori feature del dataset è stata utilizzata la GUI di Weka.

Per analizzare le metriche per realizzare grafici che è stato utilizzato il programma Jmp.

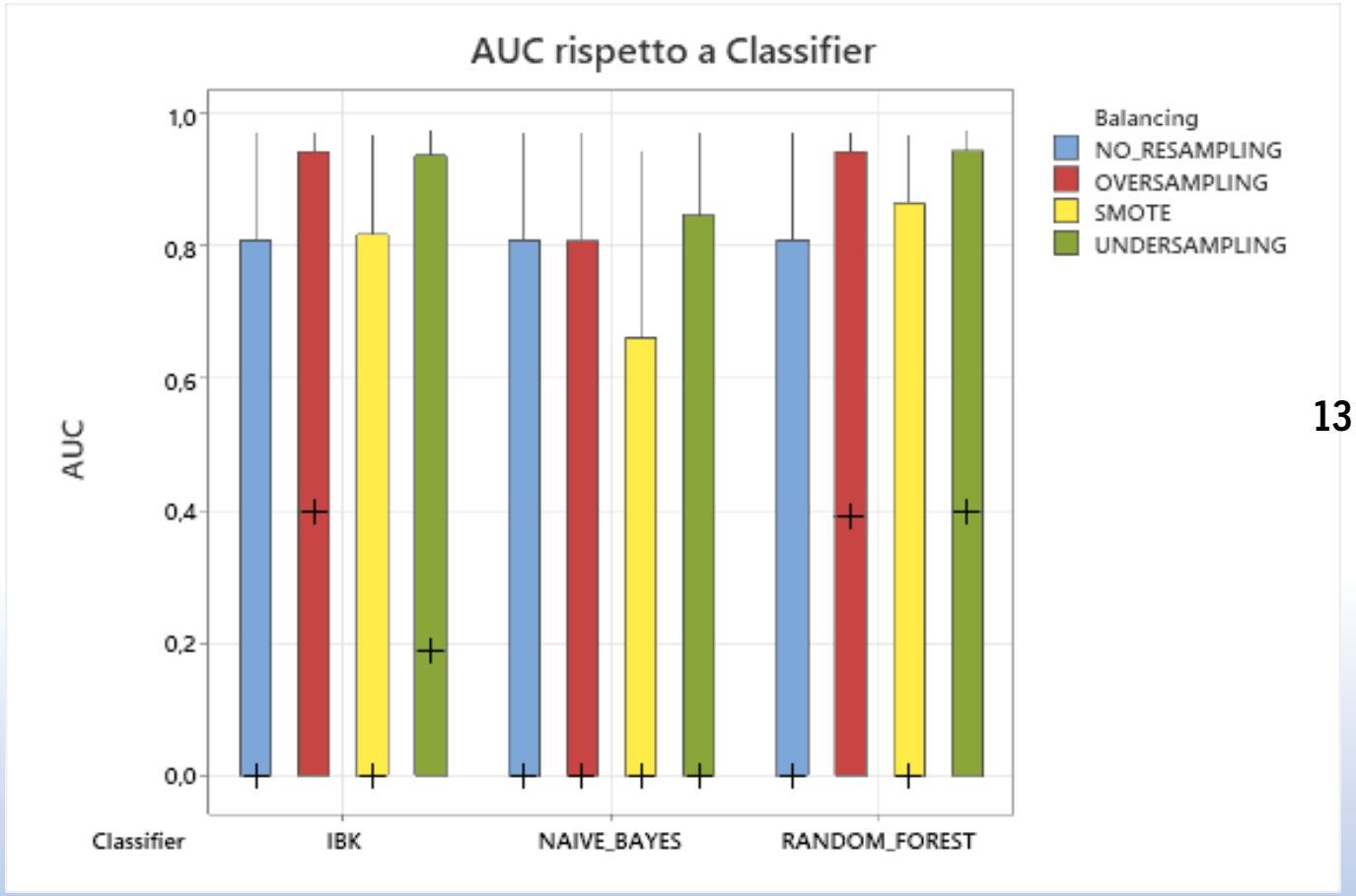
Inoltre, per evitare l'effetto di ***snoring*** è stato scartato il 50% delle versioni più recenti del dataset, prima di essere valutato dal classificatore.

Per ogni classificatore, tecniche di feature selection, di cost sensitive e balancing sono state analizzate:

- Precision
- Recall
- AUC
- Kappa

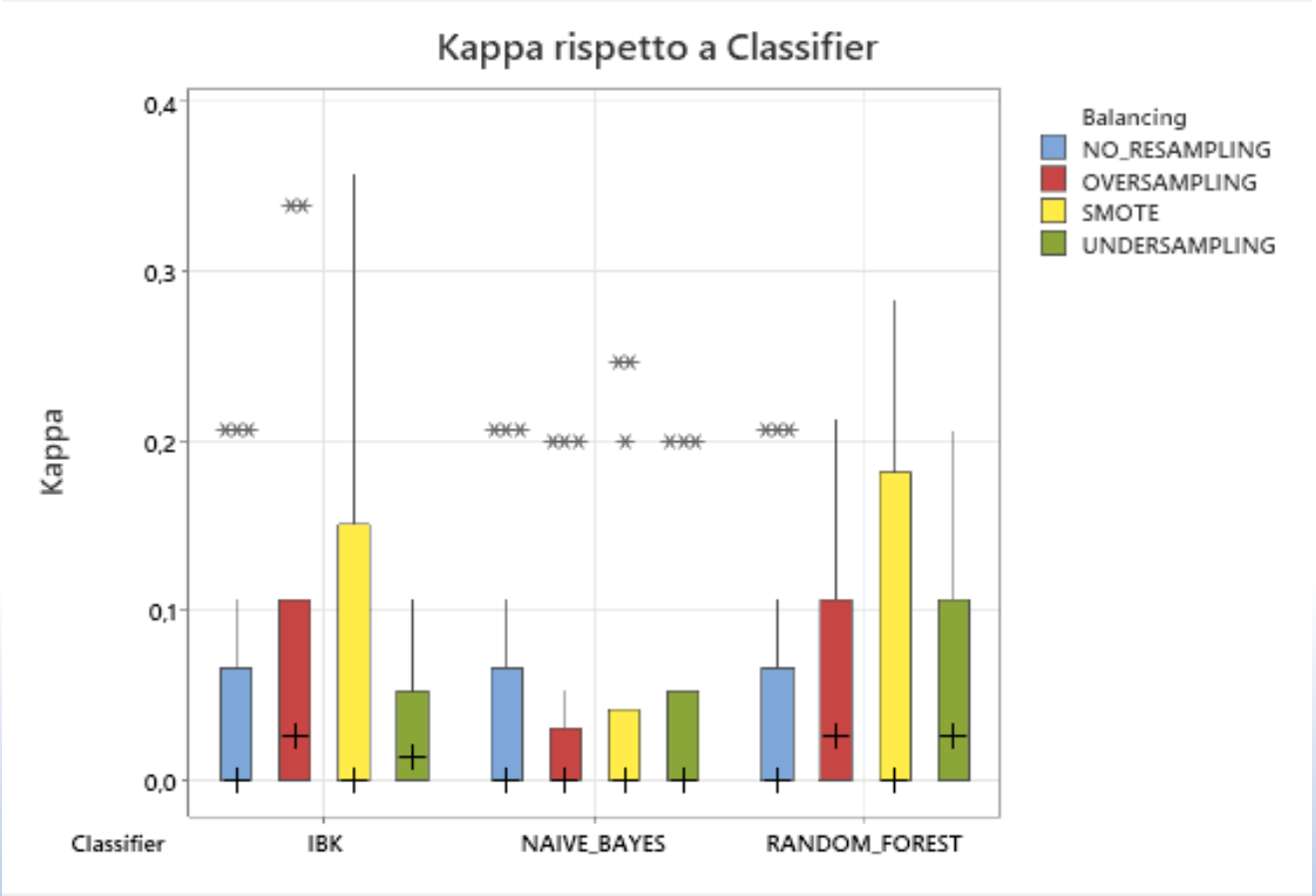
DISCUSSIONE BOOKKEEPER-AUC

- Per IBK, la tecnica di bilanciamento migliore risulta essere Oversampling;
- Per RandomForest, le tecniche di bilanciamento migliori risultano essere sia Oversampling che Undersampling;
- Mentre, per Naive Bayes le diverse tecniche si comportano allo stesso modo.



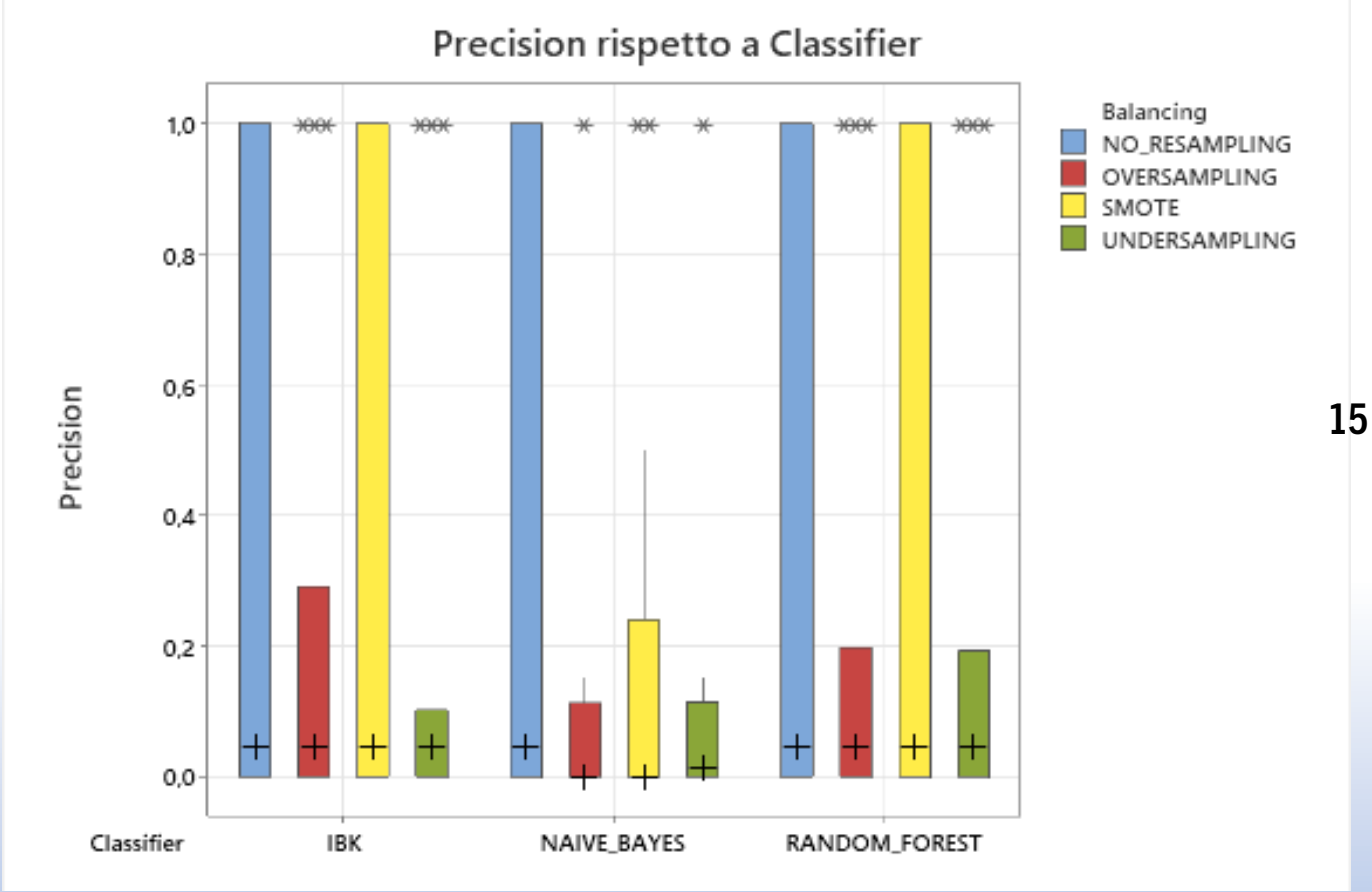
DISCUSSIONE BOOKKEEPER-KAPPA

- Per IBK, la tecnica di bilanciamento migliore risulta essere Oversampling;
- Per RandomForest, le tecniche di bilanciamento migliori risultano essere sia Oversampling che Undersampling;
- Mentre, per Naive Bayes le diverse tecniche si comportano allo stesso modo.



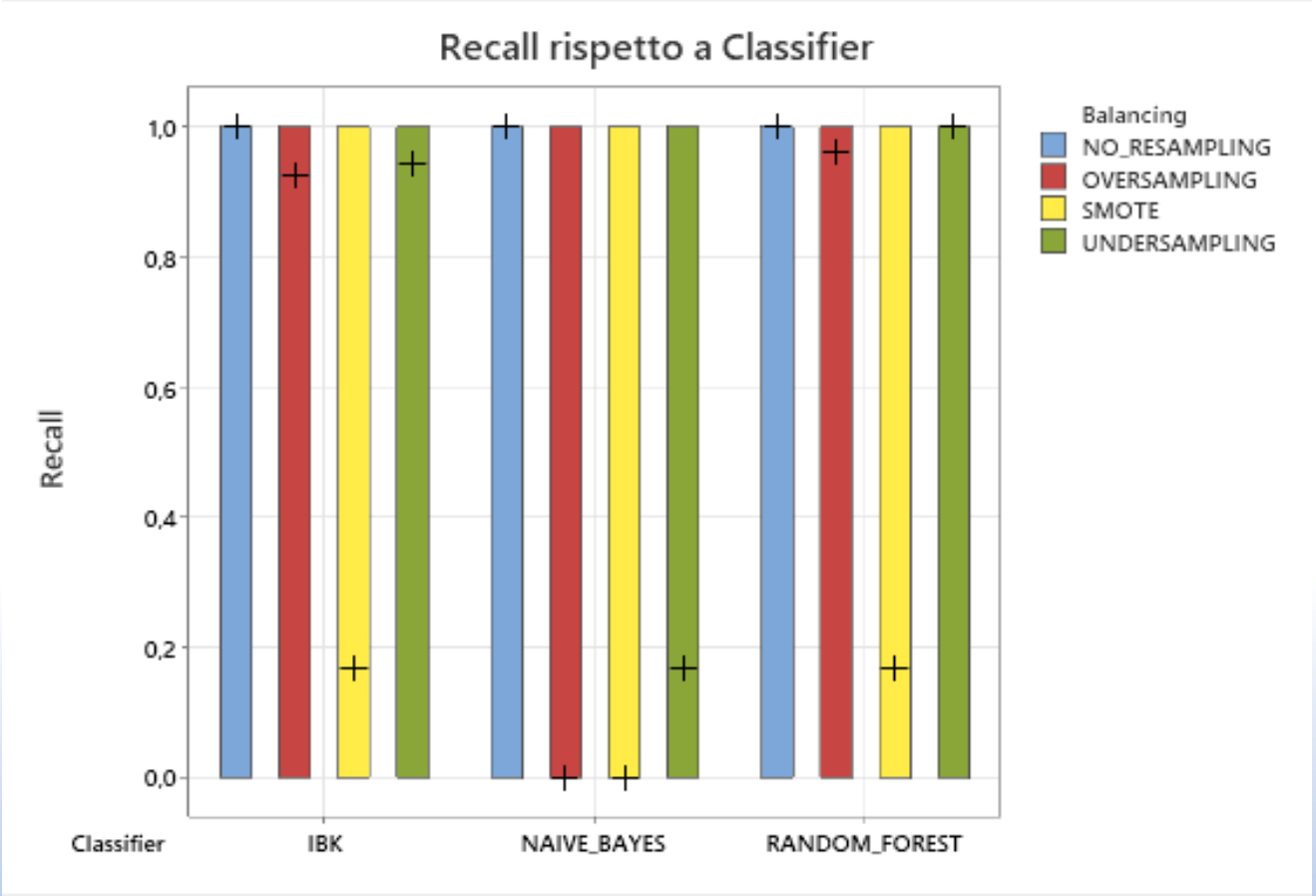
DISCUSSIONE BOOKKEEPER-PRECISION

- Per IBK e Random Forest, la precision di comporta allo stesso modo per le diverse tecniche di bilanciamento.
- Per Naive Bayesla precision migliora rispetto a quando non si utilizza nessuna tecnica di bilanciamento.



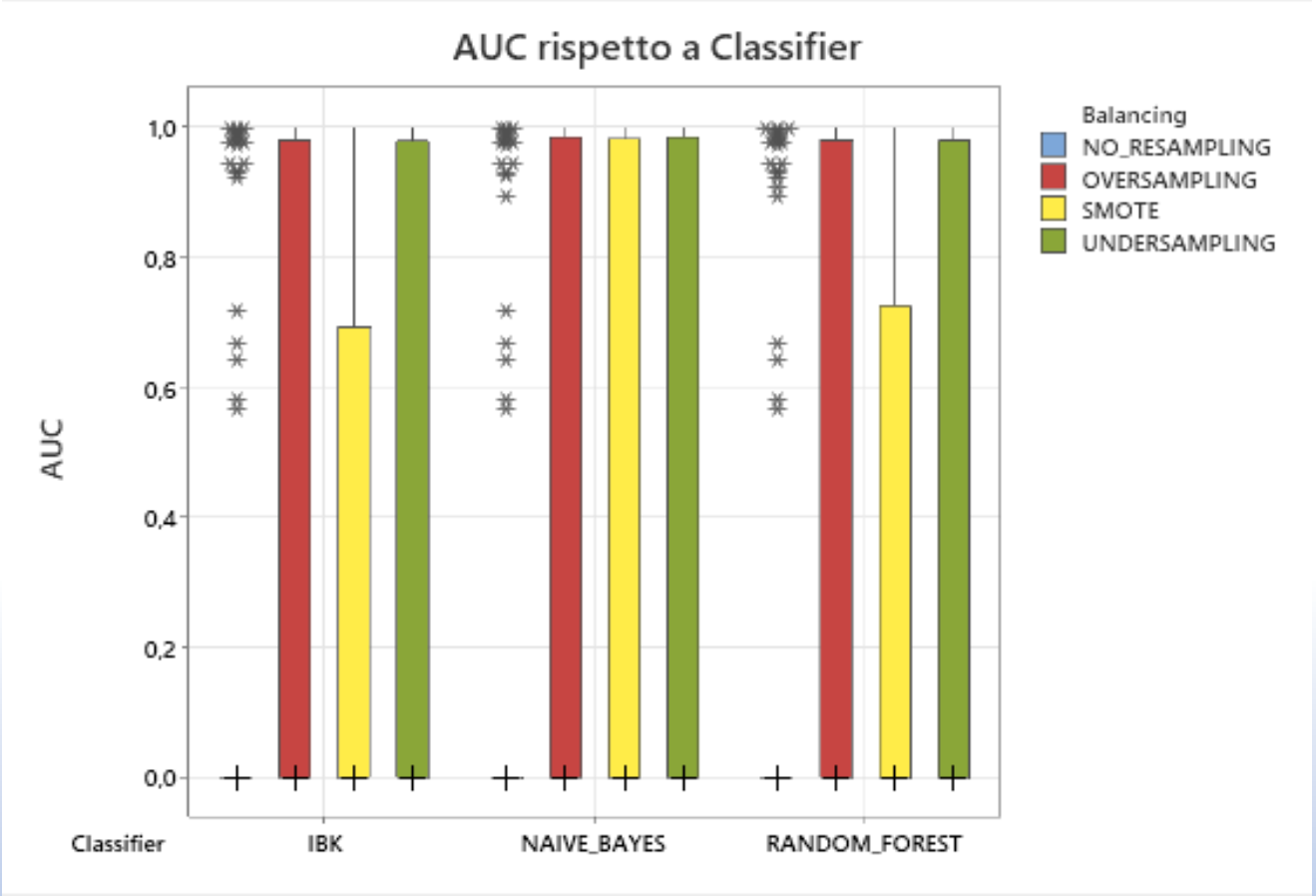
DISCUSSIONE BOOKKEEPER-RECALL

- In Ibk e Naive Bayes, tutte e tre le tecniche riducono il numero di positivi individuati.
- Nel caso di RandomForest, Undersampling aumenta il numero di positivi individuati rispetto alle altre due tecniche di bilanciamento;
- Per tutti e tre i classificatori si ottengono più positivi applicando nessuna tecnica di bilanciamento;
- Smote risulta essere il peggiore.



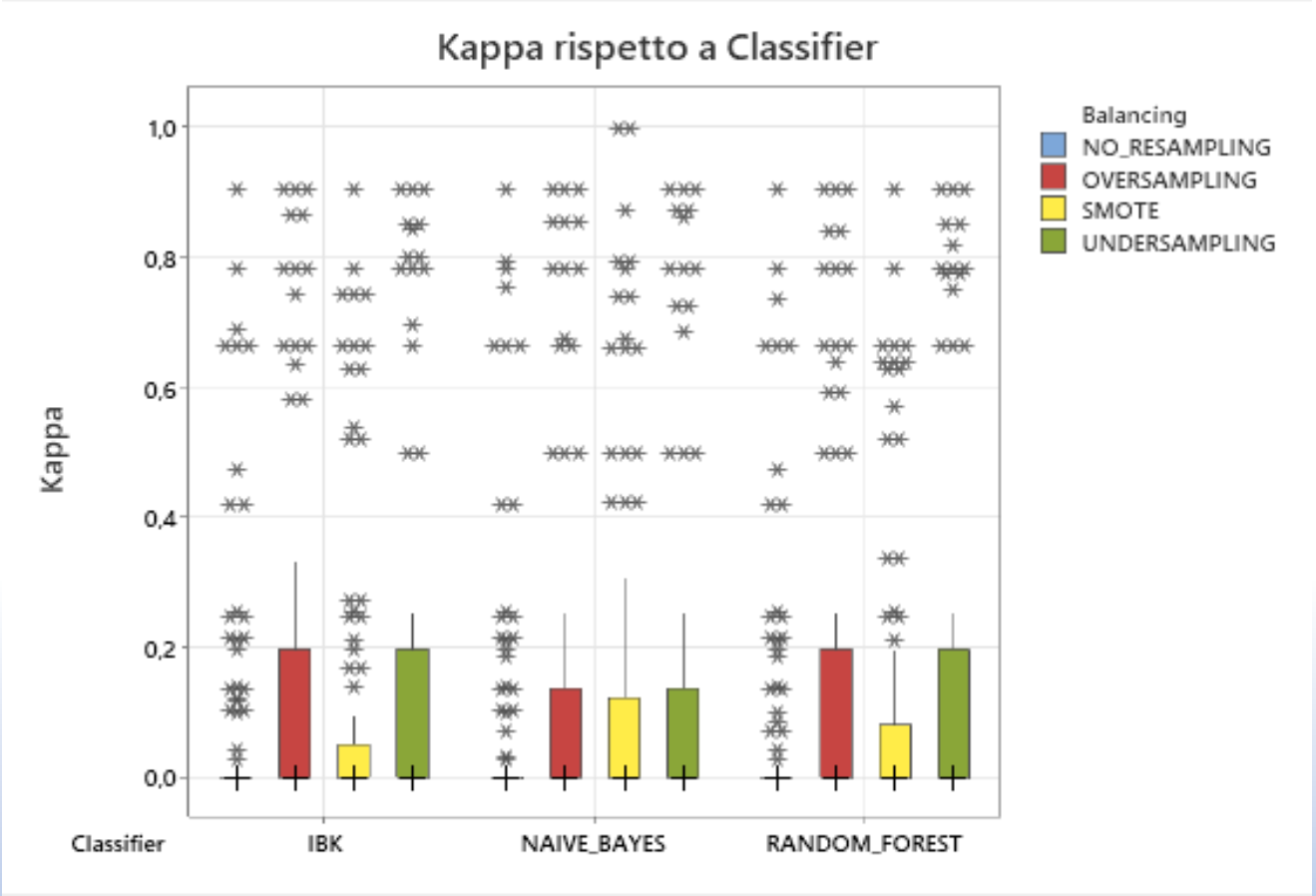
DISCUSSIONE SYNCOPE-AUC

- Analizzando questo grafico possiamo dire che le diverse tecniche di balancing si comportano allo stesso modo rispetto a quando non viene applicato nessuna tecnica di bilanciamento.
- La mediana dei tre classificatori per ogni tecnica di bilanciamento risulta essere intorno allo zero.



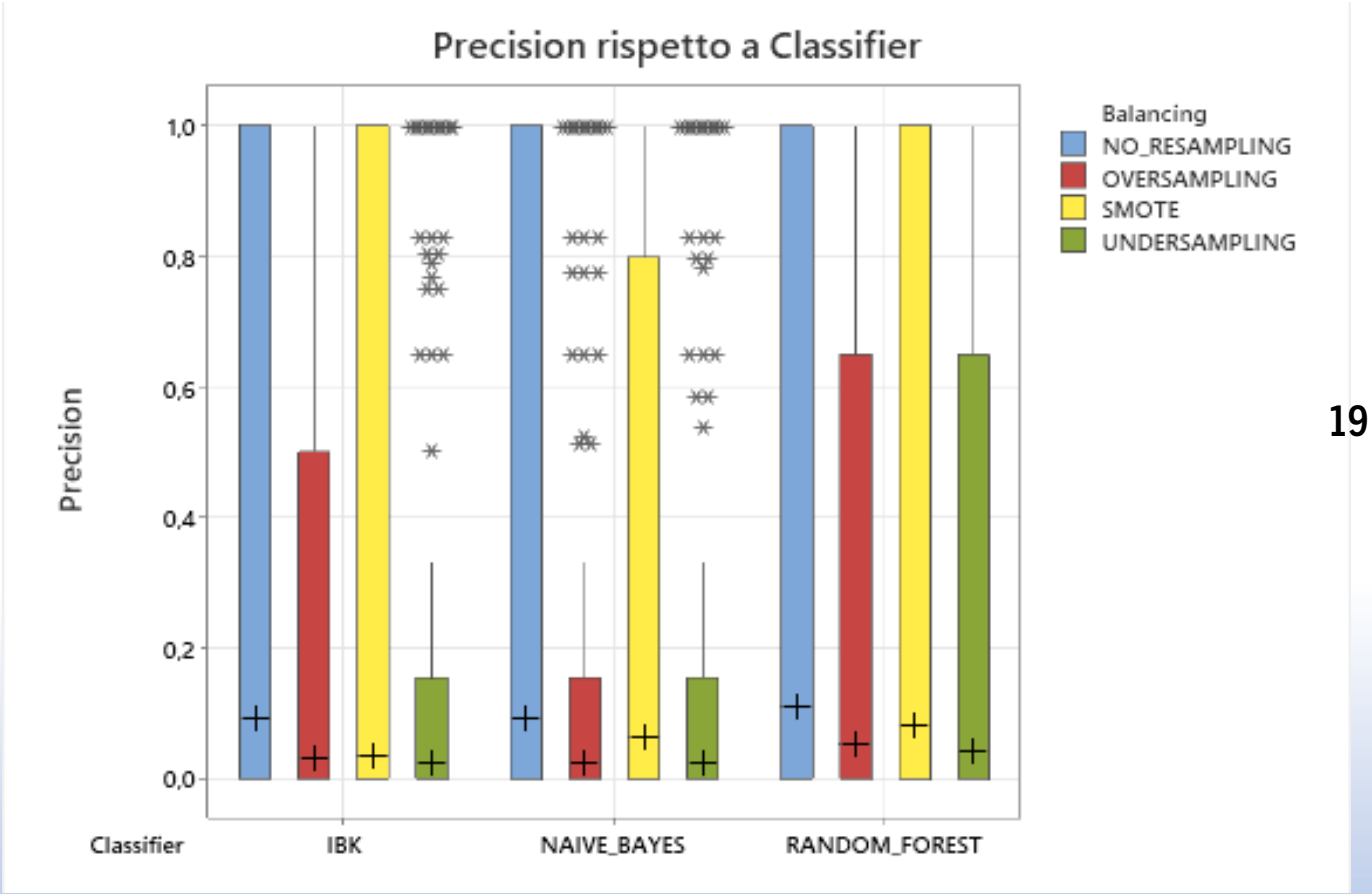
DISCUSSIONE SYNCOPE-KAPPA

- Analizzando questo grafico possiamo dire che le diverse tecniche di balancing si comportano allo stesso modo rispetto a quando non viene applicato nessuna tecnica di bilanciamento.
- Possiamo notare che ci sono numero di outilier e che anche in questo caso la mediana è intorno a zero.



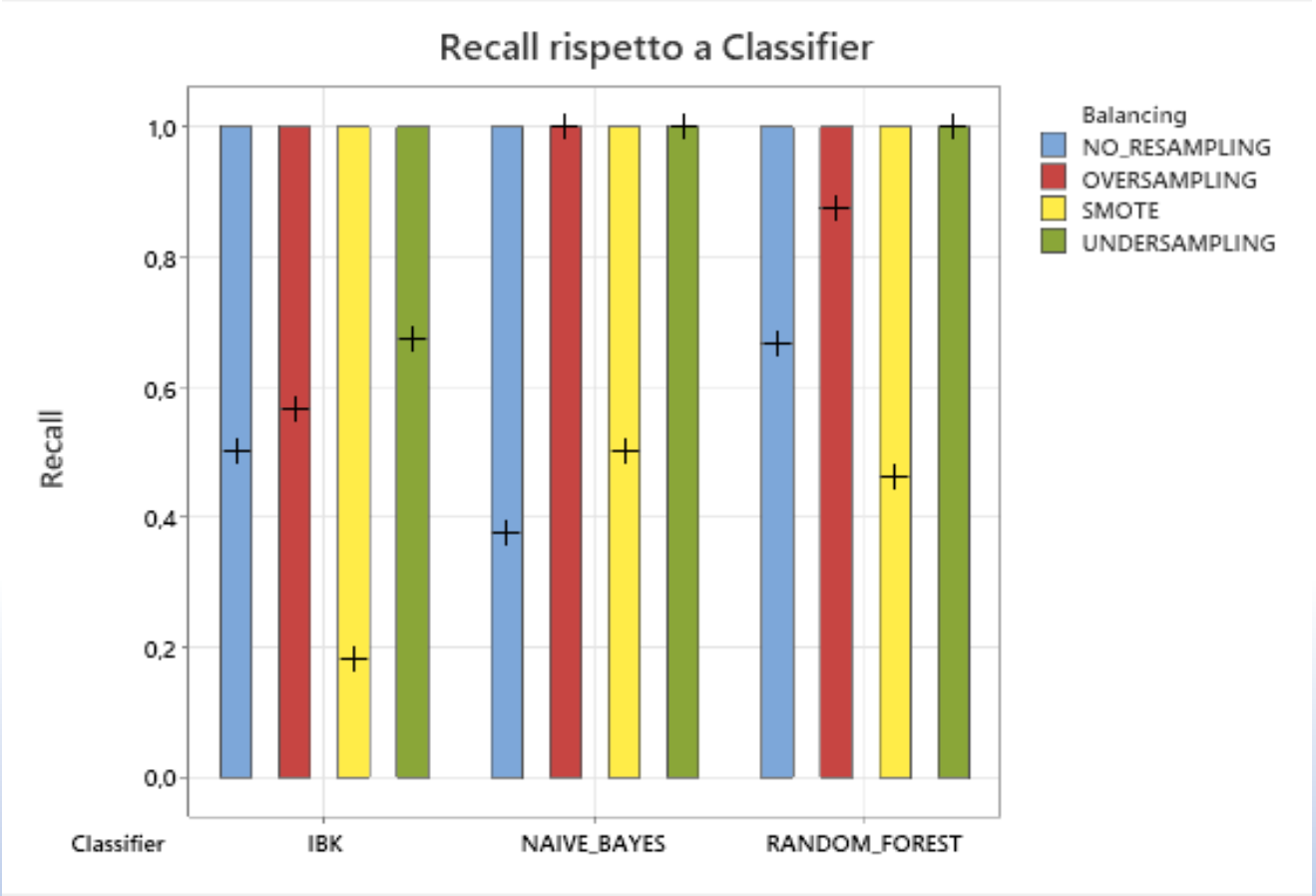
DISCUSSIONE SYNCOPE-PRECISION

- La precision è migliore senza nesuna tecnica di bilanciamento per tutti e tre i classificatori.
- Smote è quello che ha una precision migliore rispetto ad Oversampling e Undersampling, soprattutto per i classificatori Naive Bayes e Random Forest.



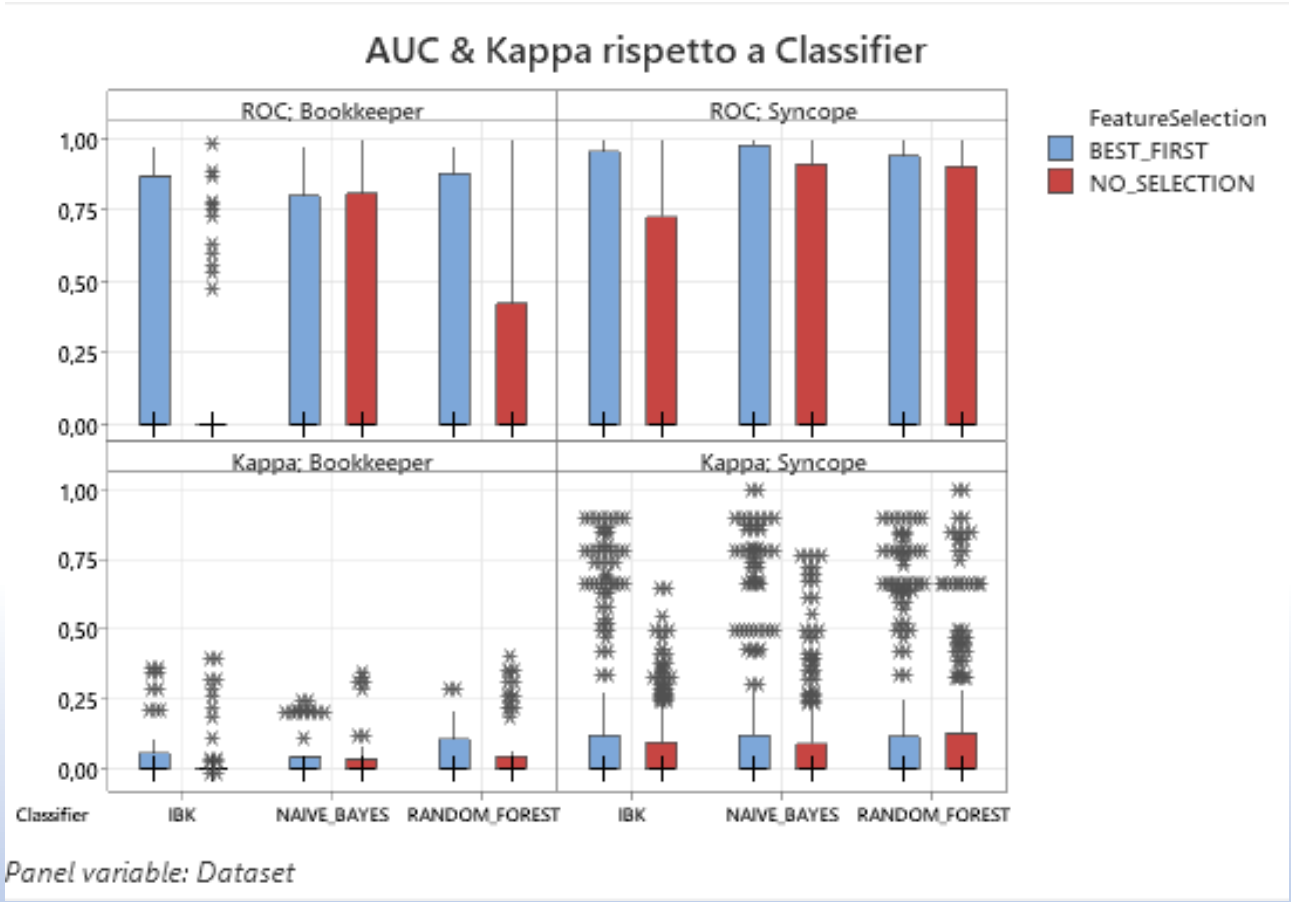
DISCUSSIONE SYNCOPE-RECALL

- In Ibk, Undersampling aumenta il numero di positivi individuati, quindi aumenta la Recall.
- In RandomForest sia Oversampling che Undersampling aumentano il numero di positivi.
- In NaiveBayes, tutte e tre le tecniche di bilanciamento aumentano notevolmente il numero di positivi trovati.
- Smote risulta essere il peggiore sia per IBK che RandomForest.



DISCUSSIONE- FEATURE SELECTION

- Tra le 14 feature considerate solo 1 è risultata rilevante al fine di predire la difettosità delle classi.
- Questo spiega il perché applicando feature selection non migliora, poiché non ha abbastanza informazioni che porterebbe a migliorare il modello.
- Possiamo notare comunque, che utilizzando BestFirst si ha una variabilità dei dati maggiori rispetto a non utilizzarla.



CONCLUSIONE- BILANCIAMENTO

Tra le diverse tecniche di bilanciamento notiamo che non c'è un migliore delle altre, infatti a volte si comportano bene mentre altre volte no e questo potrebbe essere influenzato dal dataset considerato e dai classificatori.

Possiamo notare che sia su BookKeeper e sia su Syncope le tecniche di Undersampling e Oversampling aumentano il numero di positivi individuati.

Smote è la tecnica di bilanciamento che risulta essere il peggiore.

Inoltre, possiamo notare attraverso il grafico della Recall che il bilanciamento ha un impatto maggiore su BookKeeper che su Syncope , infatti il primo riesce ad individuare più positivi.

CONCLUSIONE- CLASSIFICATORE

Tra i diversi classificatori non c'è un migliore delle altre, anche in questo caso potrebbe essere influenzato dal dataset considerato e dai classificatori.

In particolare, nella maggior parte dei casi i classificatori si comportano male e non danno buoni risultati.

Ma se vogliamo considerare il classificatore che risulta migliore rispetto agli altri è il RandomForest, che a differenza degli altri classificatore rientra nella categoria dei classificatori Bagging..

Mentre il classificatore che si comporta peggio è NaiveBayes, infatti possiamo notare che proprio in questo classificatore si ha un impatto minore sulle tecniche di bilanciamento rispetto ad altri.

GRAZIE PER L'ATTENZIONE

software-analytics:
https://github.com/emelis-ptr/software-analytics

software-analytics:
https://sonarcloud.io/project/overview?id=emelis-ptr_software-analytics