REPORT ISW2

MACHINE LEARNING APPLICATO ALL'INGEGNERIA DEL SOFTWARE

Pontis Alessandro

IN SINTESI

- *Introduzione*
- *Progettazione*
- *Proportion*
- *Calcolo fixed, affected, opening version*
- *Metriche*
- *Walk forward, Feature selection, Sampling*
- *Risultati e discussioni finali*

INTRODUZIONE

- Nello sviluppo software, le misurazioni di informazioni relative a ticket e task è molto importante, ne è un esempio il CMMI.

- Per lo sviluppo di grandi applicazioni è importante porre un focus sugli strumenti di supporto per l'individuazione delle possibili classi contenenti bug.

- Il focus principale dello studio è quello di, dato un sistema di dati di riferimento, di **predire** il comportamento futuro di questo sistema con un certo grado di accuratezza attraverso l'utilizzo del **machine learning**

INTRODUZIONE

- Per i nostri scopi è stata applicata la **classificazione binaria,** in quanto il nostro scopo è stato studiare la buggyness di classi Java.

- Sono state applicate inoltre delle tecniche di **sampling** e **feature selection**.

- Si vuole quindi studiare e misurare l'effetto sull'accuratezza del modello predittivo che abbiamo sviluppato attraverso le tecniche suddette.

- Lo studio empirico in questione è stato concretizzato analizzando le applicazioni open-source **Bookkeeper** e **OpenJPA**.

PROGETTAZIONE

- La prima fase dello sviluppo consisteva in:

    1. Raccogliere i ticket di tipo BUG nello stato: CLOSED o RESOLVED e resolution FIXED.
    2. Analizzando tutti i commit relativi, facendo riferimento all'ID di Jira.
    3. Se il commit è associato ad uno o più ticket validi allora vado ad alimentare una struttura contenente tutte le Issue.

Jira Software

PROGETTAZIONE

- Le informazioni che offre Jira e che sono stati utilizzate sono le seguenti:
  - **Creation date**: data di creazione del ticket.
  - **Resolution date**: data di risoluzione del ticket.
  - **Opening version (OV)**: indica la versione rilasciata successivamente all'apertura del ticket.
  - **Fixed version (FV)**: indica la versione rilasciata successivamente alla chiusura del ticket.
  - **Affected version (AV)**: indica la lista delle versioni del progetto affette dallo specifico bug.
  - **Injected version (IV)**: indica la versione in cui viene inserito il bug

Jira Software

PROGETTAZIONE

Il flusso di riferimento utilizzato per la creazione del dataset è il seguente:

1. Viene fatta una query sui task di Jira in modo da costruire una mappa delle versioni dalle release date
2. Vengono memorizzati i ticket (relative alle Issue) ottenuti dalla query verso Jira
3. Vengono compilate le OV e le FV
4. Vengono compilate le AV e le IV di cui abbiamo informazioni
5. Vengono stimate le restanti AV e IV dei task
6. Faccio il retrieve dei commit collegati alle issue di cui prima (attraverso il commit message)
7. Vado ad effettuare il calcolo delle metriche
8. Genero un file CSV suddividendo il CSV in base alla versione e al filename indicando metriche e buggyness
9. Applico tecniche di classificazione con tre diversi classificatori a partire dal CSV generato al punto 8 applicando inoltre feature selection e sampling.

PROPORTION

Ci sono dei task per cui non è possibile individuare una IV, di conseguenza è necessaria una stima e la strategia adottata è stata la **proportion**.

Questa si basa sull'idea che ci sia una proporzione tra il numero di versioni necessarie per individuare il difetto e il numero di fix di questo. Va calcolato il valore P come segue e ricavandosi il valore di IV

$$P = \frac{FV - IV}{FV - OV} \quad \rightarrow \quad IV = FV - (FV - OV)P$$

Con P che può essere stimato in vari modi, in questo caso abbiamo utilizzato la modalità **increment** che effettua la media tra i valori dei P ticket precedenti.

In casi rari in cui non è possibile ottenere le IV, FV e OV dai ticket precedenti, quello che possiamo fare è calcolare P andando a considerare un altro progetto.

CALCOLO FV E OV

Per il calcolo delle **fixed version** viene utilizzata la **resolution date** ricavata dai ticket, andandola a calcolare come un numero intero relativo all'indice dato dalla versione rilasciata durante la resolution date stessa. Questo è stato possibile grazie alla mappa delle versioni che avevamo costruito nel punto 1 della fase di progettazione.

In modo simile a quanto visto sopra andiamo a ricavarci la **opening version** ma andando a considerare la **creation date** data dal ticket.

CALCOLO AFFECTED VERSION

Abbiamo due possibilità per il calcolo dell'affected version

- AV è presente tra le informazioni ricavabili dal ticket, in questo modo sarà possibile risalire al valore di IV come la versione tale per cui la data di rilascio è inferiore a tutte le presenti nella lista degli AV

- AV non è presente e di conseguenza andiamo a stimare IV come segue:
  1. Viene calcolato il valore di P
  2. Associo P al ticket
  3. Stimo il valore di IV per i ticket che non hanno AV. Considero il ticket con indice i
  4. Viene calcolato P come media tra i ticket con AV ed indice compreso tra 0 e i-1
  5. Con il valore di P trovato, effettuo il calcolo di IV stimato

$$IV = FV - (FV - OV)P$$

  6. A questo punto vado a considerare le AV come tutte le versioni che sono comprese tra IV e FV

METRICHE

L’associazione tra le varie misurazioni effettuate avviene attraverso la coppia [Classe, Versione]. Le metriche sono le seguenti:

- LOC ADDED
- MAX LOC ADDED
- AVG LOC ADDED (media fatta sulle revisioni)
- LOC TOUCHED
- CHURN
- AGE
- AVG CHANGESET
- MAX CHANGESET
- BUGGYNESS
- NUMERO ATTRIBUTI PRIVATI
- NUMERO ATTRIBUTI PUBBLICI

WALK FORWARD

Data la dipendenza temporale dalle singole versioni del dataset è stata utilizzata la tecnica **walk forward** per la validazione del dataset.

La tecnica consiste nell'eseguire n-1 run dove ad ogni run h che il numero di release considerate come **training set** aumenta di un'unità.

In questo modo l'ultima iterazione corrisponderà alle versioni che vanno da 1 a n-1 per il **training set** e la versione n come **testing set**

FEATURE SELECTION

La tecnica di **feature selection** ci permette di ridurre il numero di attributi non correlati, cercando di migliorare la fase di training

L'idea di base è quella della creazione di un subset contenente gli attributi che sono più scorrelati tra loro ottenendo quindi un numero ridotto di colonne. Inoltre quello che abbiamo è che non ci aspettiamo un aumento della precisione ma di rimanere stabili ottenendo comunque un vantaggio a livello **computazionale**.

La selezione degli attributi avviene secondo la **backward search**, ovvero partendo da un set completo andando poi a rimuovere una alla volta le feature, andando a rimuoverle basandoci sul fatto che andiamo a migliorare l'accuracy.

SAMPLING

/l **sampling** è una tecnica che viene utilizzata nell'ambito in cui abbiamo class imbalance ovvero quando si ha una popolazione di training sbilanciata. Quello che vogliamo fare quindi è pareggiare la classe minoritaria con quella maggioritaria.

Le tecniche che possono essere applicabili sulle istanze di classe **buggy** e **non buggy** sono le seguenti:

- **Undersampling**: vado a rimuovere le istanze della classe maggioritaria in modo da bilanciare con quella minoritaria
- **Oversampling**: vado ad aggiungere un numero di istanze (ripetendo quelle di classe minoritaria) per andare a pareggiare il conto
- **SMOTE**: è un raffinamento dell'oversampling, in quanto le classi minoritarie che vengono aggiunte non sono copiate ma vengono generate in modo sintetico andandoci a base sulle istanze della classe stessa.

CLASSIFICATORI E INDICATORI

Ai fini dello studio, sono state considerate le seguenti scelte implementative:

- **Classificatori**: RandomForest, NaiveBayes, IBK
- **Feature selection:** attiva, non attiva
- **Balancing**: non attiva, oversampling, undersampling, SMOTE

Per quanto riguarda le metriche che andremo ad analizzare riguardanti i classificatori abbiamo:

- **Precision**: Percentuale di classificazioni positive corrette rispetto a tutti i positivi (anche quello classificati erroneamente tali)

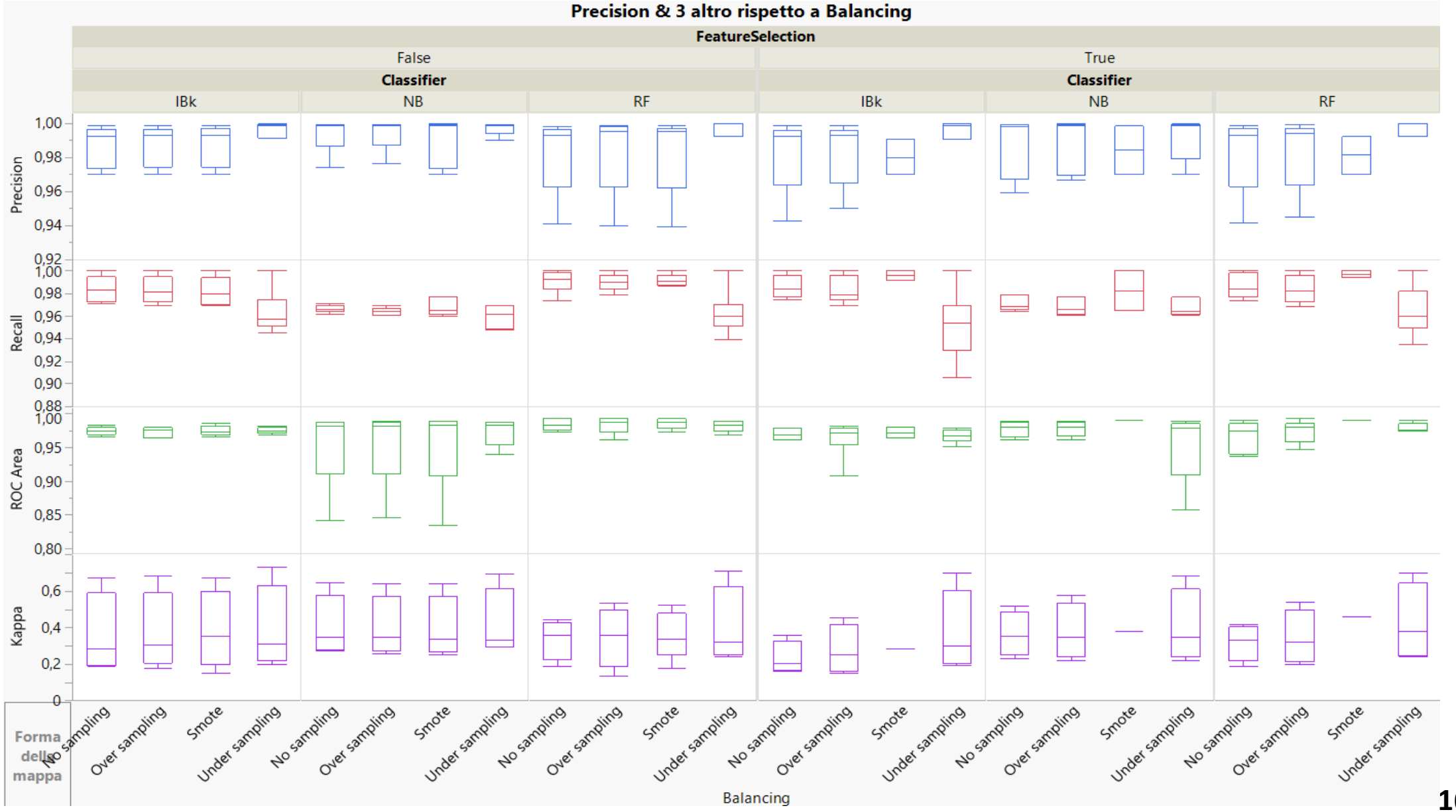
$$\mathrm{P} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$$

- **Recall:** Percentuale di elementi positivi del testing set che sono stati classificati positivi in modo corretto, infatti è fatta rispetto ai risultati che effettivamente erano positivi.

$$\mathrm{R} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$$

- **ROC area (AUC)**: Area sotto la curva ROC che mette in relazione la variazione di recall e precision al variare di threshold su una stima.
- **Kappa**: Quanto si è stati più accurati rispetto ad un classificatore dummy.

RISULTATI BOOKKEEPER



CONSIDERAZIONI

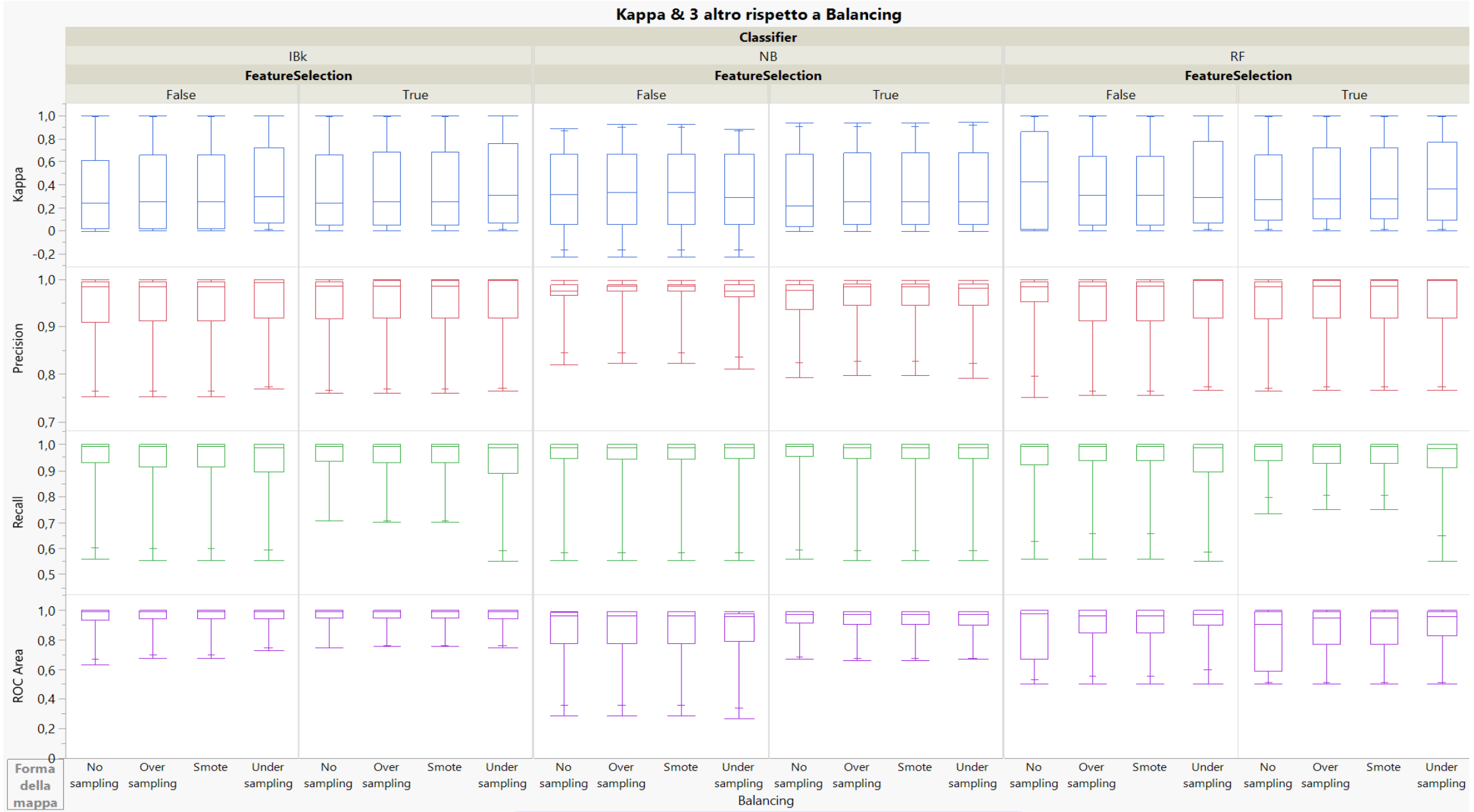
Possiamo notare come l'impatto maggiore che ha l'utilizzo della tecnica di **feature selection** si ha per il classificatore Naive Bayes il cui utilizzo aumenta la varianza di metriche quali Kappa, Recall e Precision area in modo considerevole, incrementando però la ROC area in alcune combinazioni.

Per quanto riguarda invece la tecnica di **sampling** possiamo notare come l'undersampling vada a generare un livello di varianza più elevato in quasi tutte le grandezze, anche se maggiormente sulla recall, mentre oversampling e no sampling ottengo dei risultati comparabili, andando a migliorare (seppur di poco) il valore mediano.

Da notare inoltre come la tecnica di undersampling non abbia buoni risultati, questo è probabilmente causato dalla scarsità di dati relativi alla classe minoritaria. Per lo stesso motivo possiamo notare come si abbia una costanza nel valore di ROC e Kappa relativi alle implementazioni con feature selection e SMOTE, questo dovuto all'impossibilità del tool di generare classi per numero troppo basso nella classe minoritaria

È possibile notare come in base ai risultati il miglior classificatore risulti essere, facendo delle considerazioni riguardanti anche la variabilità dei risultati, NB senza feature selection e con tecnica si sampling oversampling.

RISULTATI OPENJPA



CONSIDERAZIONI

Facendo un rapido confronto con il grafico di Bookkeeper salta all'occhio un aumento di varianza rispetto ai risultati di accuracy che abbiamo con OpenJPA, essendo le metriche utilizzate le stesse, questo risultato è dovuto alla composizione delle singole istanze del dataset.

Possiamo notare come l'impatto maggiore che ha l'utilizzo della tecnica di **feature selection** si ha ancora una volta per il classificatore Naive Bayes il cui utilizzo riduce la varianza di metriche quali Kappa, Precision e ROC area in modo considerevole, lasciando invariata la Recall.

Per quanto riguarda le tecniche di **sampling** abbiamo che per tutti i classificatori fatta eccezione di Naive Bayes, possiamo notare come l'undersampling porti ad un miglioramento della precision ed in generale anche del valore Kappa. A differenza di Bookkeeper in questo caso l'undersampling sembra funzionare meglio probabilmente a causa del fatto che essendo OpenJPA sia un progetto più ampio e creato in precedenza.

Il classificare migliore risulta essere **IBK** con feature selection e undersampling.

CONCLUSIONI

In conclusione, dopo aver analizzato i due dataset e dopo aver applicato le varie tecniche descritte sopra possiamo dire che l'utilizzo di queste in alcune configurazioni abbia portato a dei miglioramenti, soprattutto per quanto riguarda il valore di **precision**, soprattutto per quanto riguarda Bookkeeper.

Per quanto riguarda OpenJPA abbiamo avuto si dei miglioramenti in certe configurazioni ma non di particolare rilevanza numerica.

Ulteriore menzione per la tecnica di sampling che meglio si è comportata su Bookkeeper è stata SMOTE in quanto probabilmente avendo un numero ridotto di entry nella classe minoritaria ha beneficiato della generazione artificiale di queste permettendo un addestramento migliore.

Grazie per l’attenzione