Machine Learning for Software Engineering

Pier Francesco Contino

Maggio 2020

1 Delivery 1

1.1 Introduzione

Nel software engineering è importante tenere sotto controllo il processo di sviluppo del codice, questo può servire per trovare eventuali problemi e correggerli tempestivamente, o per avere un preventivo delle dimensioni dei risultati che si otterranno.
In questa parte del progetto andremo ad analizzare la stabilità dell'attributo "fixed ticket" (i ticket JIRA completati) nel progetto Apache "C++ Standard Library". Per farlo utilizzeremo un process control chart le cui ascisse sono le mensilità e le cui ordinate sono il numero di ticket completati appunto. Da questo grafico potremo controllare come quest'attributo si comporta rispetto alla media e rispetto ai limiti di controllo ricavati dalla media e dalla deviazione standard dei dati.

In questa delivery (e anche in quella successiva) andremo ad interfacciarci con gli strumenti utilizzati durante lo sviluppo dei relativi progetti e che ormai sono molto affermati nel campo della programmazione e sviluppo di progetti software:

1. *Git* uno strumento per il controllo versione distribuito
2. *JIRA* un sistema di tracking di ticket, le richieste di assistenza per la risoluzione di un problema o la realizzazione una funzionalità.

Come da specifiche la data di chiusura di un ticket viene presa andando a cercare l'ultimo tra i commit di git collegati a quel ticket e prendendo la sua data.

1.2 Progettazione

Per la generazione del dataset è stato scritto un programma in java che:

- Si interfacciasse con le API web di apache per prelevare i ticket di JIRA in formato JSON

- Si interfacciasse con il repository git collocato sul sito github
- Elaborasse i risultati
- Esportasse i risultati nel formato CSV

Per la parte di accesso al repository git è stato scelto di interfacciarsi direttamente all'applicazione git tramite la classe java process builder che consente di lanciare comandi e recuperare i risultati trasmessi sullo standard output e sullo standard error.
Il programma è stato eseguito e testato su una macchina linux (Ubuntu 20.04) nonchè buildato tramite il servizio cloud di continous integration travis.
Il programma si avvale delle librerie:

- commons-csv di apache (org.apache.commons)
- json-java (org.json)

1.3 Risultati

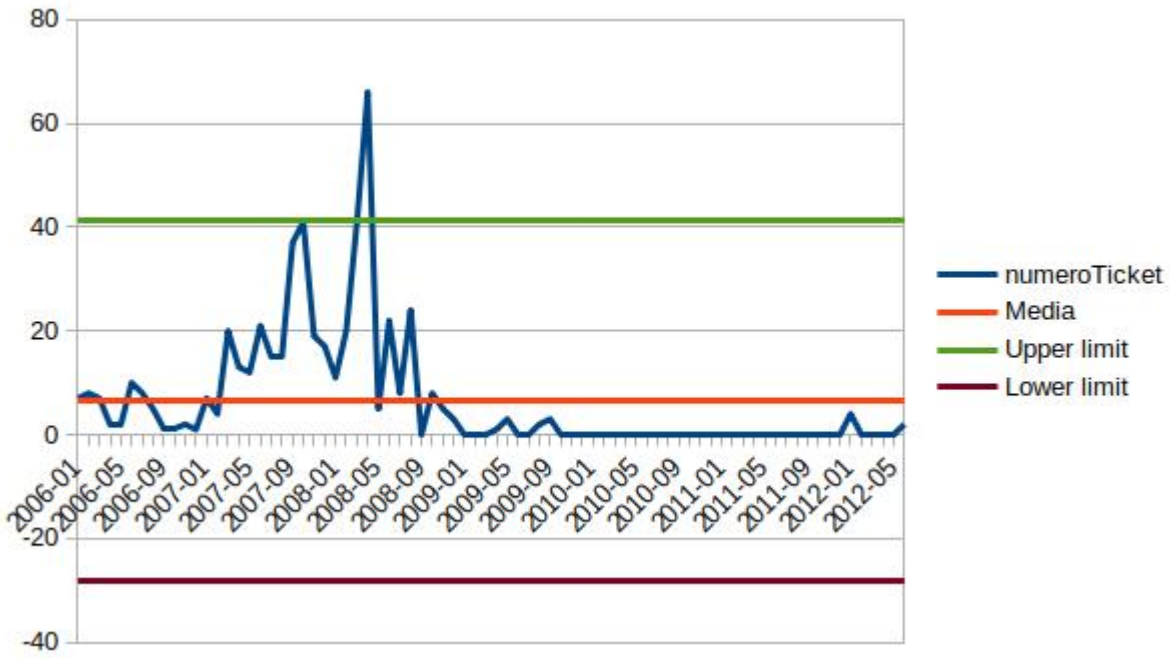


Figura 1: Process Control Chart Ticket Fixed

1.4 Conclusione

Guardando questo grafico si può notare un'alta variabilità nel numero di ticket completati, infatti la deviazione standard di tali valori è **11,587** su una media

di **6,448** pari quindi al 179% della media.
La conseguenza è che il margine inferiore calcolato è negativo e quindi bisogna prendere come margine negativo lo 0 dato che il numero di ticket completati è una valore sempre positivo.
Il numero di ticket sfora l'upper limit **41,212** solo nel mese Aprile del 2008 con un valore di **66 ticket**.
Per studiare questo grafico può essere utile utilizzare delle informazioni sul relativo progetto:

- Inizia nel Settembre del 2005 dopo una donazione del codice da un'azienda che lo rende open source
- Esso diventa nel 15/11/2007 un progetto TLP (Top Level Project) di Apache.[1]
- L'ultima release sviluppata risale al 02/05/2008 ed è la 4.2.1 [1]
- Nel 15/05/2014 il progetto viene concluso e spostato nell'Apache Attic per via della discontinuità nello sviluppo delle release.[2]

Ulteriori notizie sullo status del progetto possono essere reperite dal sito Apache dove sono presenti dei report periodici.[3]
Leggendo il report di Maggio 2008 possiamo vedere che il progetto si stava preparando per implementare il nuovo standard per il C++ che doveva essere ratificato per il 2009; questo è secondo noi il motivo dell'incremento di ticket per l'aprile del 2008.
Leggendo i successivi report si può vedere come il progetto perse di popolarità in seguito allo slittamento del rilascio del nuovo standard nel 2011 (report Agosto 2008).
Gli altri report testimoniano l'abbandono del contributo dell'azienda Rogue Wawe Software, l'azienda che aveva donato il codice della libreria alla fondazione Apache.

2 Delivery 2

Progetti Analizati:

- BOOKKEEPER
- ZOOKEEPER

2.1 Introduzione

In questo progetto andremo ad analizzare come varie tecniche di feature selection e di sampling riescono a migliorare le prestazioni dei classificatori durante l'analisi delle classi di un progetto Apache al fine di trovare le classi affette di bug.
La feature selection è il processo volto a selezionare un sottoinsieme delle features, quelle più rilevanti, che poi in seguito verranno utilizzate nella costruzione del modello. La conseguente semplificazione del modello riduce l'overfitting, cioè l'eccessivo adattamento ai dati analizzati durante il training, una causa del peggioramento delle performance nel testing. Un altro effetto positivo è una riduzione del tempo di training del sistema, aspetto che tuttavia nel nostro caso risulta secondario in quanto è invece la generazione del dataset da analizzare a richiedere la maggior parte del tempo totale dell'analisi.
La tecnica di feature selection utilizzata in questo progetto è la Best First: essa usa un approccio che va a massimizzare la separabilità delle classi, per farlo utilizza un approccio greedy in congiunzione con una tecnica di backtracking che va a ritroso per cercare eventuali combinazioni tralasciate.
Il sampling è una tecnica che consente di andare a bilanciare i dati di training tra le loro classi, sono molto utili quando c'è una differenza accentuata tra la classe di minoranza e la classe di maggioranza (nel nostro progetto queste classi sono i file con o senza un bug). Il sampling ha come effetto principale l'aumento della recall la misura di quanto il modello riesce a richiamare gli elementi richiesti.
Nell'analisi vengono utilizzate 3 principali metologie di sampling:

- Il subsampling dove si vanno a ridurre le istanze della classe di maggioranza preservando tutte le istanze della classe di minoranza.
- L'oversampling dove si vanno ad aumentare le istanze della classe di minoranza preservando tutte le istanze della classe di maggioranza. Nel nostro caso utilizzeremo il resampling dove andiamo a riconsiderare più volte le istanze della classe di minoranza.
- Il Synthethic Minority Oversampling Technique (SMOTE) dove vengono generate delle istanze "sintetiche" della classe di minoranza basandosi sullo spazio delle features delle istanze esistenti. In pratica si scelgono dei valori per le features che vanno a minimizzarne una metrica di distanza con quelli delle caratteristiche già esistenti.

Nell'analisi vengono utilizzati 3 classificatori:

- Random Forest: utilizza una moltitudine di alberi decisionali per poi selezionare come output la loro moda.
- Naives Bayes: un classificatore probabilistico che utilizza il teorema di Bayes assumendo indipendenza tra le features.
- IBk: basato sull'algoritmo k-nearest neighbours che si basa sulle k istanze più vicine nello spazio delle features.

Come tecnica di valutazione è stata utilizzata il walk forward: dove il dataset viene diviso in parti in base alla più piccola unità del campo che le ordina, in seguito queste vengono ordinate cronologicamente per poi in ogni run selezionare una parte per il testing e utilizzare le parti precedenti per il training.

Parte 1	Parte 2	Parte 3	Parte 4
Training	Testing		
Training	Training	Testing	
Training	Training	Training	Testing

Tabella 1: Esempio di walk forward

2.2 Progettazione

Per l'analisi è stato utilizzato un programma java che si interfaccia con JIRA al fine di reperire le versioni e i ticket del progetto, il quale si interfaccia inoltre con git per prelevare i commit da cui estrapolare le metriche di ogni file java (le modalità di interfacciamento sono le medesime descritte nella sezione della delivery 1).
Per quanto riguarda le metriche sono state selezionate le seguenti:
Size, Loc_touched, NFix, NAuth, Loc_added (Total, Max, Avg), Churn (Total, Max, Avg), Age.
Risulta importante notare che molte di queste metriche sono direttamente correlate tra loro e questo può avere un effetto sul modello generato dal classificatore, la tecnica di feature selection opera un selezione delle metriche togliendo molte delle metriche presenti.
Per la stima delle classy buggy sono state usate le Affected Version fornite da JIRA in congiunzione con la tecnica proportion vista durante il corso. Per l'uso della tecnica proportion è necessario ricavare per ogni bug: l'Opening Version cioè la versione in cui è stato aperto il ticket relativo al bug, la Injected Version la release in cui è stato generato il bug e la Fixed Version la versione dove è stato risolto il bug. Nel progetto si è selezionato la release con la data di rilascio immediatamente successiva alla data di apertura del ticket per l'OV , mentre per IV si è seleziona la AV più recente, infine per la FV si è utilizzato il relativo campo dei ticket JIRA, che però contiene release multiple richiedendo pertanto

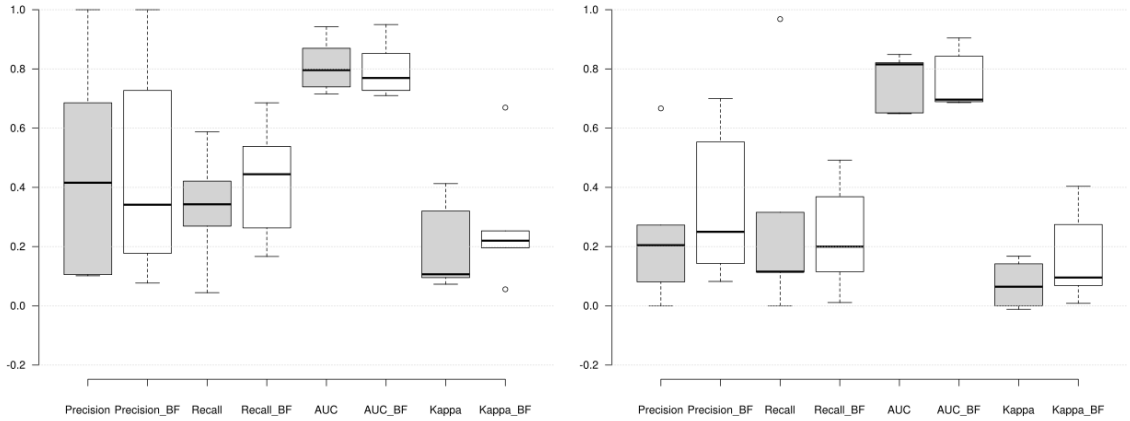
anche qui la selezione della release più recente.
Prima di applicare proportion è stato fatto un controllo di consistenza sui dati presi da JIRA rimuovendo così FV e AV precedenti alla data di creazione del ticket, sono stati inoltre rimossi i ticket privi di commit corrispondenti.
Il metodo proportion è stato usato nella modalità increment, quindi calcolando p come la media dei valori ottenuti dalle versioni precedenti.
Per quanto riguarda la parte di machine learning sono state usate le API di weka sia per la divisione del dataset tra training e testing che per effettuare la valutazione del modello e per applicare le tecniche di features selecting e sampling.
Vengono elencate le opzioni utilizzate per le tecniche di features selecting e sampling poiché anch'esse hanno un impatto sui risultati ottenuti, da notare in particolare le opzioni di BestFirst che permettono una ricerca più approfondita dei parametri ideali rispetto a quelle standard richiedendo solamente un tempo aggiuntivo di ricerca che però è trascurabile rispetto al tempo totale richiesto per eseguire il progetto.

Tabella 2: Opzioni utilizzate

Tecnica	Opzioni
Resample	Distribuzione uniforme e dimensione output pari al doppio della classe di maggioranza
Subsample	Distribuzione massima della classe pari a un rapporto 1:1
Smote	
BestFirst	Ricerca bidirezionale con numero di nodi non migliorativi testati pari al numero degli attributi del dataset

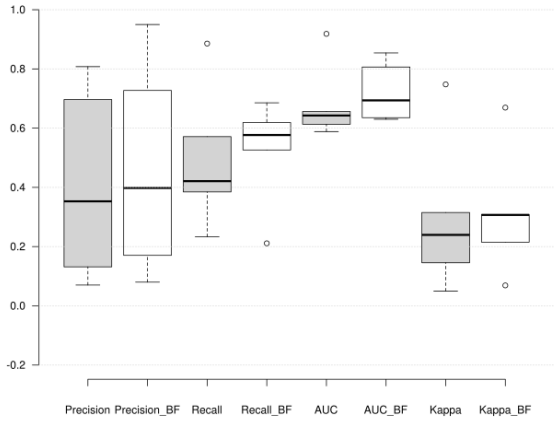
Come da specifiche dalle release dei progetti prelevate è stata rimossa la metà più recente in quanto queste versioni avevano una buona possibilità di celare bug non ancora rilevati andando quindi a falsare il risultato dell'analisi.
Per quanto riguarda i dataset utilizzati nell'analisi essi si compongono di 6 release per *Bookeeper* e 19 release per *Zookeeper*, una differenza sostanziale nella quantità di dati a disposizione del classificatore che costituisce fattore rilevante sui risultati ottenuti.
Un altra considerazione riguardante i dataset è che essi sono molto sbilanciati, infatti la classi di minoranza (le classi buggy) è circa il 12% del totale in entrambi i progetti, in aggiunta questa non è distribuita in maniera uniforme tra le release, difatti nel dataset si possono presentare release con un numero molto esiguo di bug.

2.3 Risultati

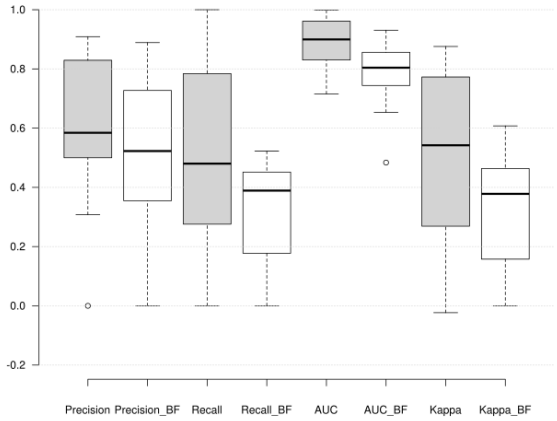


(a) Random Forest Bookkeeper

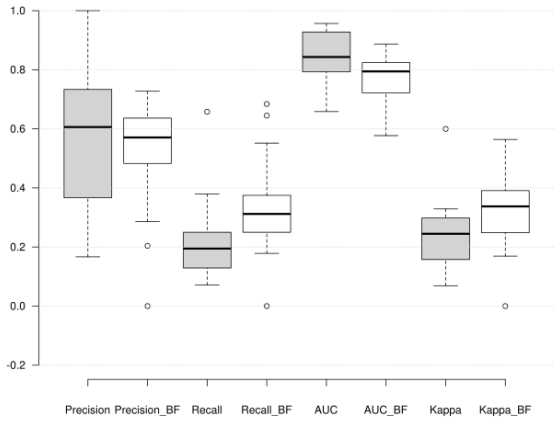
(b) Naive Bayes Bookeeper



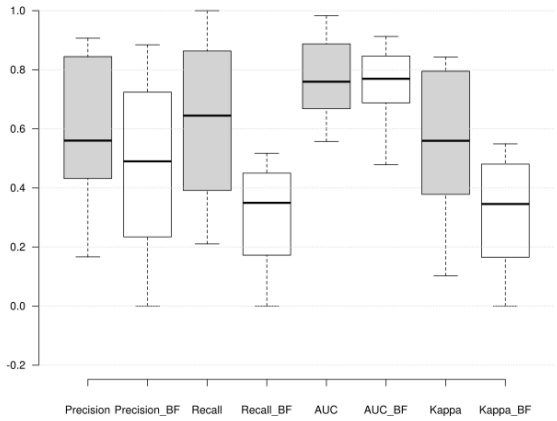
(c) IBK Bookeeper



(d) Random Forest Zookeeper



(e) Naive Bayes Zookeeper



(f) IBK Zookeeper

Figura 2: Confronto effetto feature selection sui vari classificatori

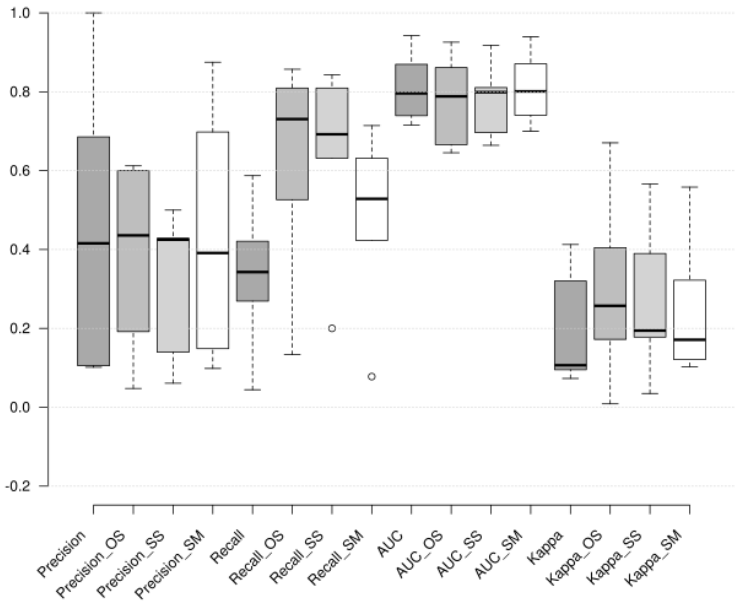


Figura 3: Confronto effetto sampling su Random Forest in Bookkeeper

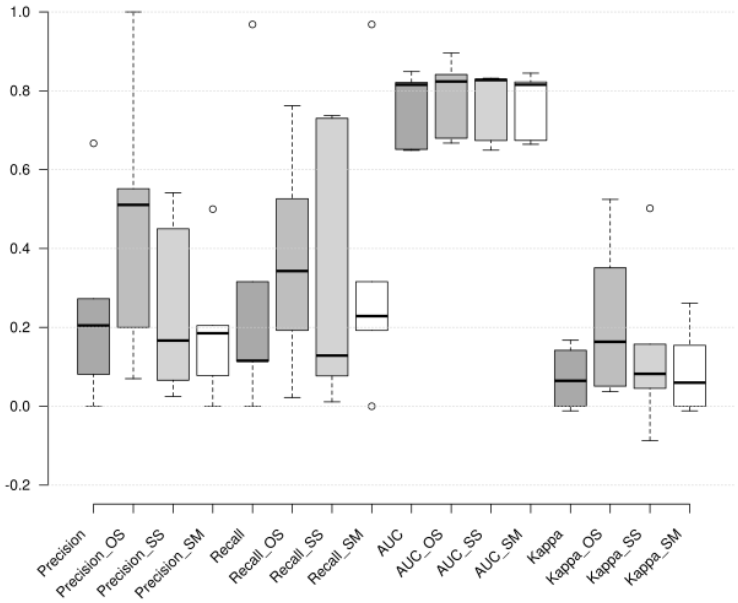


Figura 4: Confronto effetto sampling su Naive Bayes in Bookkeeper

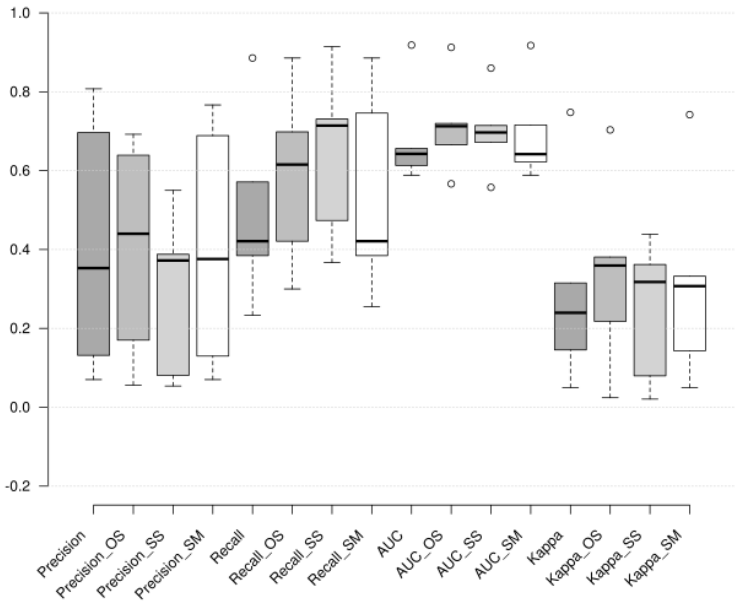


Figura 5: Confronto effetto sampling su IBK in Bookkeeper

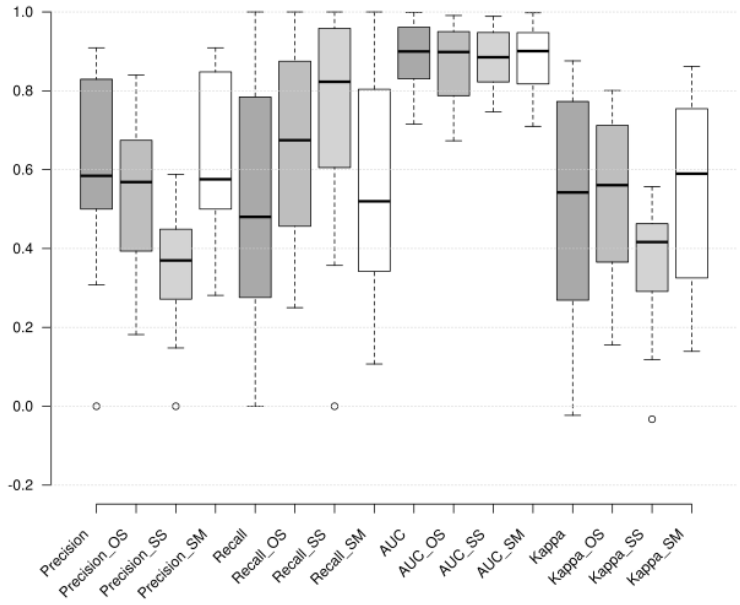


Figura 6: Confronto effetto sampling su Random Forest in Zookeeper

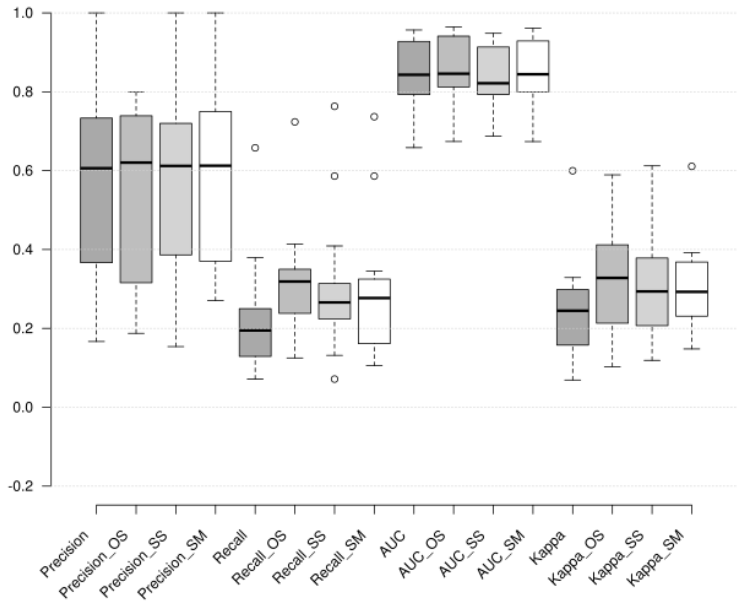


Figura 7: Confronto effetto sampling su Naive Bayes in Zookeeper

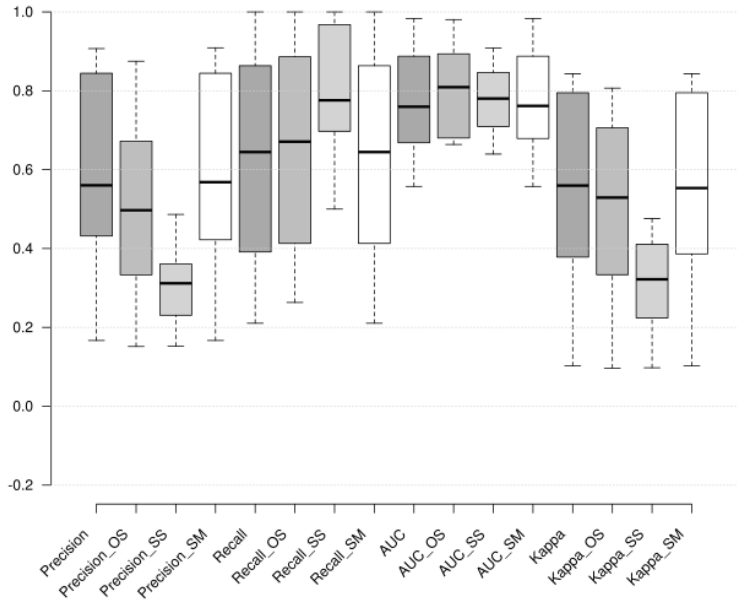


Figura 8: Confronto effetto sampling su IBK in Zookeeper

2.4 Conclusione

Per quanto riguarda gli effetti della features selection nel dataset Zookeeper si evidenzia un peggioramento sia della recall che della precision, con l'eccezione del classificatore Naive Bayes che ha un miglioramento in entrambi i dataset per la recall, mentre nel dataset bookeeper sembrerebbe che la features selection porti dei buoni risultati migliorando la recall.
L'interpretazione dei dati ottenuti è che la features selection è positiva per Naive Bayes in quanto esso assume un'indipendenza tra i dati in input, inoltre sembrerebbe utile anche per semplificare il modello quando si hanno pochi dati a disposizione come nel caso di Bookeeper; bisogna comunque considerare che i risultati dei classificatori su Bookeeper rimangono comunque inferiori rispetto a quelli su Zookeeper.
Il sampling per entrambi i dataset ha come effetto un aumento della recall, in particolare questa metrica è la più alta per il subsampling a scapito però della precision, un buon compromesso tra recall e precision sono l'over sampling e lo SMOTE sampling. Facendo un confronto tra OS e SMOTE balancing l'OS risulta migiore per quanto riguarda il dataset Bookeeper mentre essi si equivalgono su Zookeeper: con il primo che ha maggior recall e il secondo preserva quasi totalmente la precision.
In conclusione data la natura della ricerca, trovare classi affette da bug, si ritiene consigliabile preferire una maggior recall rispetto alla precision e quindi si consiglierebbe l'utilizzo di Oversampling senza features selection.

Riferimenti bibliografici

[1] *Apache C++ Standard Library Releases.* URL: `https://github.com/apache/stdcxx/releases`.

[2] *Apache C++ Standard Library and the Attic.* URL: `http://mail-archives.apache.org/mod_mbox/stdcxx-dev/201307.mbox/%3C0F1B5FEB-9312-44CA-96CF-B586B8CE6BA4@apache.org%3E`.

[3] *Apache C++ Standard Library Status.* URL: `http://stdcxx.apache.org/status/`.