Report Ingegneria del Software 2

Marco Marcucci 0286352

Sommario

Derivable 1

Introduzione

Per aumentare la maturità di una organizzazione software, è necessario migliorarne il processo di sviluppo. Il primo passo per raggiungere tale obiettivo è, ovviamente, misurare il processo software in modo da capire gli eventuali problemi da risolvere.

Un processo può essere classificato in base all'esito dell'osservazione che ne deriva in due categorie:

- sotto controllo statistico, quando è influenzato unicamente da fattori casuali;
- fuori controllo statistico, quando l'influenza delle sue variazioni è causata da fattori specifici.

Nello specifico, lo scopo di questa prima derivable è:

- misurare la stabilità del numero di fixed bugs del progetto Mahout, realizzando un process control chart;
- individuare eventuali periodi di instabilità relativamente al numero di fixed bugs e cercare di identificare le possibili cause di tale comportamento ed eventuali soluzioni da adottare in futuro;
- determinare se il processo può essere considerato stabile in relazione all'attributo studiato.

Progettazione

Lo sviluppo di questa derivable si è articolato in diverse fasi:

1. Nella prima fase è stato necessario recuperare tutti i ticket di tipo Bug del progetto Mahout da Jira, con stato 'Closed' e 'Resolved'.
2. Nella seconda fase, invece, sono stati analizzati i dati recuperati nella fase precedente.

Per il recupero dei ticket di tipo Bug, è stata utilizzata una chiamata di tipo REST che permettesse di recuperare tutti i ticket con quelle particolari etichette; in seguito, attraverso delle chiamate di sistema, utilizzando il software di controllo 'Git', è stato possibile recuperare le informazioni necessarie (e.g. data di chiusura del ticket) per analizzare i dati.
A supporto dello sviluppo della prima derivable, sono state implementate classi che permettessero di scrivere codice il più possibile comprensibile e riusabile.

Risultati e discussione

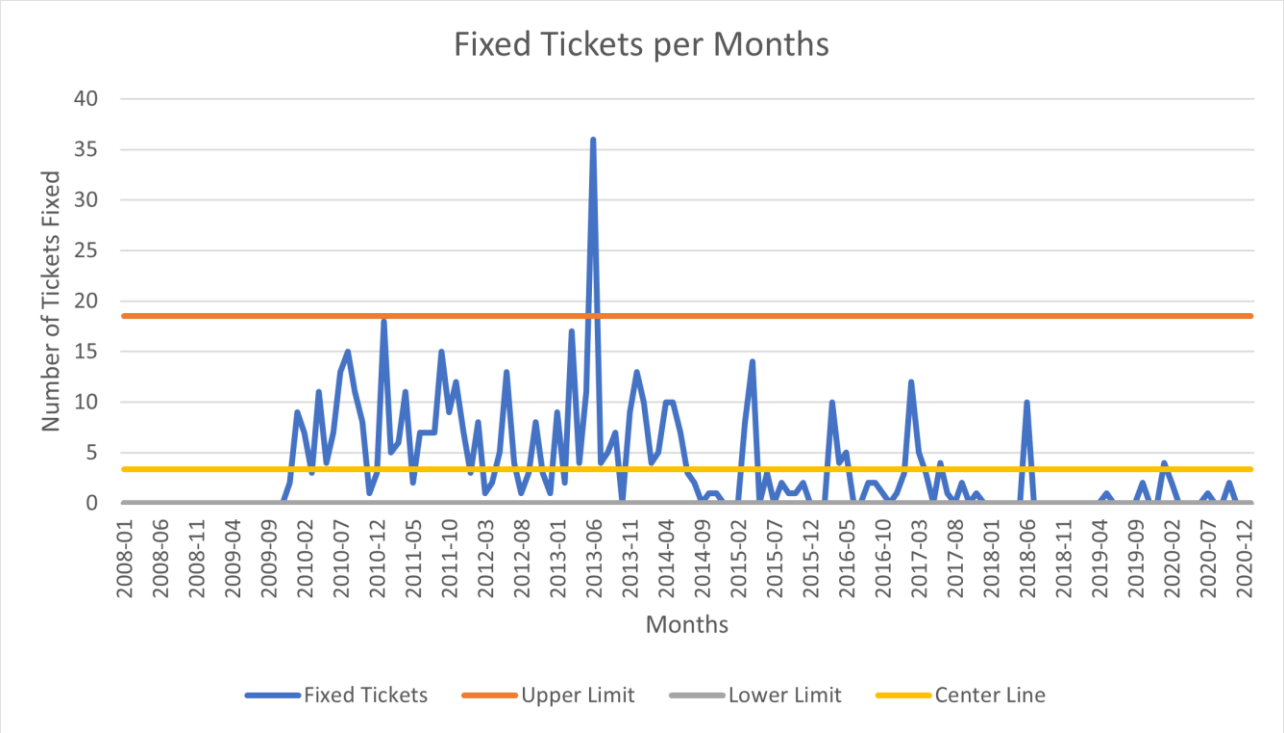

Figura 1 Process control chart del numero di Fixed Tickets nel periodo tra 2008-01 e 2021-01.

È possibile osservare dalla Figura 1, i valori assunti nel tempo del numero di bug fixed nei rispettivi periodi temporali. Immediatamente si nota che nel maggio del 2013 il numero dei bug corretti è molto più alto rispetto agli altri mesi. In particolare, risulta essere anche maggiore dell'Upper Bound.

Il processo risulta essere, quindi, "fuori controllo" in quanto non rispecchia la definizione per cui un processo può essere definito stabile:

$$Bound = avg(values) \pm 3 * DevStand(values)$$

Come si può constatare il valore del Lower Bound calcolato risulta essere negativo. Dal momento che non ha senso parlare di valori negativi per i dati considerati, il Lower Bound è stato fissato a 0.
Nel primo periodo e nell'ultimo, inoltre, si riscontra un sostanziale numero di bug fixed prossimo allo zero Per quanto riguarda i primi mesi considerati, un numero di Fixed Tickets pari a zero potrebbe essere dovuto ad un'iniziale inconsapevolezza dei problemi riguardanti il progetto.
Per quanto riguarda, invece, l'ultimo periodo considerato, un numero così basso di Fixed Tickets potrebbe essere dovuto al fatto che, dopo un numero di anni così elevato, sia stata migliorata la maturità del processo di sviluppo software a tal punto da ridurre notevolmente la presenza di bug.

Derivable 2

Introduzione

Lo scopo della Derivable 2 è analizzare statisticamente dei dati, estratti attraverso metriche ben precise e applicate a repository di codice sorgente di grandi aziende, per cercare correlazione tra le suddette metriche e l'insorgere di bug nel codice. Questi dati, in seguito, sono utilizzati per cercare di predire, attraverso determinate tecniche, l'insorgenza di bug in futuro.
I progetti da valutare, secondo le specifiche, sono BOOKKEEPER ed OPENJPA.

Le tecniche utilizzate nella fase di predizione sono state le seguenti:

- Walk Forward, come tecnica di valutazione;
- Best First e No Selection, come tecnica di features selection;
- Over Sampling, Under Sampling, SMOTE e No Sampling, come tecnica di Balancing;
- Random Forest, Naïve Bayes ed IBK, come classificatori.

L'utilizzo di queste tecniche, come accennato in precedenza, sono state utili per valutare l'accuratezza dei modelli predittivi creati, al variare delle configurazioni.

Progettazione

L'implementazione di questa derivable è suddivisa in due fasi:

- Milestone 1;
- Milestone 2.

L'obiettivo della prima fase è stata quella di gestire nel miglior modo possibile la raccolta dei dati. In questa fase, sono state calcolate sia l'effettiva buggyness delle classi del progetto, sia le metriche necessarie per la seconda Milestone.
L'obiettivo della seconda fase, invece, è stato quello di valutare l'accuratezza dei modelli predittivi, creati combinando le diverse tecniche elencate in precedenza.

Milestone 1

Per raggiungere l'obiettivo è stato necessario creare delle classi apposite che gestissero i vari aspetti richiesti. Di seguito sono descritte brevemente le classi utilizzate:

- *ManageProperties:* Classe che gestisce il file di configurazione;
- *Ticket:* Classe che gestisce tutte le informazioni necessarie per descrivere i Ticket utilizzati;
- *Version:* Classe che gestisce tutte le informazioni necessarie per descrivere le versioni del progetto;
- *Command:* Classe che gestisce tutte le chiamate di sistema utilizzate del software Git;

- *FileProject:* Classe che gestisce tutte le informazioni necessarie per descrivere i Files Java presenti nel progetto considerato;
- *GetReleaseInfo:* Classe che si occupa del recupero di tutte le versioni del progetto considerato;
- *ManageFile:* Classe che si occupa di gestire la creazione e rimozione dei vari files utilizzati (e.g. files .csv);
- *Metrics:* Classe che gestisce tutte le informazioni necessarie per descrivere le metriche calcolate;
- *ManageMetrics:* Classe che si occupa di gestire il calcolo e l'aggiornamento delle metriche descritte in precedenza;
- *Proportion:* Classe che si occupa di gestire il calcolo del Proportion;
- *Main:* Classe contenente il punto di avvio del codice.

Program Flow

L'esecuzione del programma è suddivisa in diversi passi. Alcuni di essi, e le relative scelte effettuate, sono dettagliate più avanti.

All'avvio dell'applicazione, il repository che si intende studiare, viene scaricato attraverso il comando *git clone*, se non presente già nella cartella di riferimento, altrimenti viene aggiornato attraverso il comando *git pull*.
Nel momento in cui si dispone del repository, vengono recuperati da quest'ultimo tutti file *.java* presenti; inoltre, vengono salvate anche tutte le versioni del progetto in un file *.csv*.
A questo punto, si procede al recupero di tutti i ticket di tipo Bug, che sono stati chiusi o risolti, presenti in JIRA, avendo cura di ordinarli cronologicamente in base alla data di creazione.
Per ogni Ticket si procede all'aggiornamento del Proportion e successivamente all'assegnazione, come buggy o meno, di tutte classi che il Ticket ha coinvolto.
Infine, si procede al calcolo delle metriche, scelte precedentemente, per ogni file *.java*.

Retrieve Files

Come accennato in precedenza, per le operazioni sul repository si è utilizzato il software di controllo Git, attraverso l'uso di system calls. Queste ultime, però, risultano essere poco ottimizzate dal punto di vista prestazionale sull'ambiente in cui è stato sviluppato il progetto. Per questo motivo, si è scelto di costruire un file per ogni progetto assegnato, chiamato *Test*NameProject**, che permettesse di eliminare questi tempi di attesa durante la fase di sviluppo.
Alla prima esecuzione del programma, infatti, viene eseguita una funzione che ha il compito di recuperare tutti i file dal repository, attraverso le system call, e inserirli nel file sopra citato. Per ogni nuova esecuzione del programma, i file verranno recuperati dal file appena descritto, salvo diversamente specificato.

Buggyness e Proportion

Ricordiamo, innanzitutto, che i Tickets presenti su Jira dovrebbero avere i seguenti attributi:

- **Opening version:** Versione in cui viene aperto il Ticket relativo al difetto;
- **Fixed version:** Versione in cui il Ticket viene chiuso;
- **Injected version:** Versione in cui viene inserito il difetto;
- **Affected versions:** Rappresenta l'insieme delle versioni contenenti il difetto, ovvero [IV, FV).

Per verificare che un determinata classe *.java* del progetto considerato, sia buggy o meno e in quale versione, è necessario avere l'informazione sull'Affected Versions del Ticket o dei Tickets che coinvolgono quella classe.
Purtroppo, su JIRA alcune di queste informazioni non sono sempre presenti e devono essere recuperate in modo differente (e.g. attraverso l'uso del software Git).

Nel caso dell'Affected Versions, è possibile recuperare tale informazione, nel caso mancasse, tramite la stima ottenuta con il metodo Proportion basata sugli attributi descritti precedentemente. In particolare, il Proportion può essere ottenuto nel seguente modo:

$$P = \frac{FV - IV}{FV - OV}$$

Come descritto in precedenza, può accadere che l'attributo IV non sia presente su JIRA. Pertanto, attraverso la precedente equazione, è possibile stimare IV:

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

Ovviamente il Proportion deve essere aggiornato ogni qual volta si incontri un Ticket in cui l'attributo IV sia presente. In particolare, il Proportion è stato calcolato in modo incrementale. Ogni fixed ticket che presenta la IV, fornisce un contributo, pari agli altri ticket, per il calcolo del Proportion.

Si è deciso di utilizzare questo tipo di calcolo perché risulta essere il giusto compromesso tra tutti quelli discussi a lezione. Infatti, il metodo Cold Start prevede il calcolo del Proportion di 75 progetti ulteriori, mentre il metodo Moving Window, prevede di effettuare lo stesso calcolo di Increment, ma sull'ultimo 1% dei Tickets. Per i progetti presi in considerazione, però, il numero di Tickets non era abbastanza elevato per utilizzare quest'ultimo metodo.
Il metodo Increment, comunque, risulta essere una tecnica che permette di approssimare adeguatamente il Proportion.

Per il calcolo della buggynes sono stati scartati i tickets che avevano le seguenti caratteristiche:

- Fixed Version precedente alla Injected Version;
- Fixed Version coincidente con Injected Version.

In entrambi i casi, infatti, le classi *.java* che venivano coinvolte da questi tickets, non potevano essere considerate buggy.

Per il calcolo del proportion sono stati scartati i tickets che avevano le seguenti caratteristiche:

- Fixed Version precedente alla Injected Version;
- Fixed Version coincidente con Injected Version;
- Fixed Version precedente alla Opening Version;
- Fixed Version coincidente con Opening Version;
- Opening Version precedente alla Injected version.

In tutti questi casi, infatti, non è possibile effettuare un calcolo corretto del Proportion.

Calcolo delle metriche

Le metriche scelte sono le seguenti:

- *LOC_added:* sum over revisions of LOC added;
- *MAX_LOC_added:* Maximum over revisions of LOC added;
- *AVG_LOC_added:* Average LOC added per revision;
- *Churn:* Sum over revision of added – deleted LOC in Class;
- *MAX_Churn:* MAX_Churn over revisions;
- *AVG_Churn:* Average Churn per revision;
- *ChgSetSize:* Number of files commited together with Class;

- *MAX_ChgSetSize:* Maximum of ChgeSetSize over revisions;
- *AVG_ChgSetSize:* Average of ChgSetSize over revisions.

Anche in questo caso, attraverso l'uso del software Git, è stato possibile reperire le informazioni necessarie per il calcolo delle metriche, all'interno del repository precedentemente scaricato.

Milestone 2

Le classi implementate per lo sviluppo della Milestone 2 sono le seguenti:

- *DataSetInstance:* Classe che gestisce tutte le informazioni necessarie per descrivere un'istanza del dataset (e.g. Classifier utilizzato, Feature Selection utilizzata, TP, TN, etc.);
- *EnumContainer:* Classe che contiene tutte le tecniche da utilizzare nella fase di addestramento;
- *ManageDataSet:* Classe che si occupa di gestire la creazione dei vari files utilizzati (e.g. files *.csv* e *.arff*);
- *Main:* Classe contenente il punto di avvio del codice.

Ottenute le metriche di riferimento nella Milestone 1, è possibile creare i training set ed i testing set, estraendo i dati dai file csv, in modo da realizzare validi sets secondo la tecnica di valutazione Walk Forward.
Con questa tecnica, il dataset viene diviso in parti in base alle release, ordinate cronologicamente, del progetto. Ad ogni iterazione, tutti i dati presenti nelle release precedenti alla parte di predizione sono usati come training set, mentre i dati della release che si vuole testare sono usati come testing set.
Ad ogni iterazione, vengono usate tutte le possibili combinazioni delle tecniche di *Sampling, Balancing e Classifier.*
Per lo sviluppo della Milestone 2 è stata utilizzata la libreria *weka.jar.*

BookKeeper

Analizziamo, ora, i risultati ottenuti dai Classificatori con tutte le tecniche descritte in precedenza, per il progetto BookKeeper.

Kappa

Come sappiamo, il valore della metrica Kappa conta il numero di volte in cui il classificatore utilizzato, è più accurato di un classificatore "dummy". Essa può variare da -1 a 1. Se Kappa assume il valore -1 allora possiamo dire che il classificatore è meno accurato di quello "dummy"; se Kappa assume il valore 0 possiamo dire che il classificatore utilizzato è tanto accurato quanto quello "dummy"; se, invece, Kappa assume il valore 1, il classificatore utilizzato è più accurato di quello "dummy".
Come si può osservare dalla Figura 2, i risultati migliori si ottengono con le seguenti configurazioni:

- {*IBK*, *BEST_FIRST*, *NO_SAMPLING*};
- {*IBK*, *BEST_FIRST*, *SMOTE*};
- {*IBK*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*IBK*, *NO_FEATURES_SELECTION*, *SMOTE*};
- {*RANDOM_FOREST*, *BEST_FIRST*, *SMOTE*};
- {*RANDOM_FOREST*, *BEST_FIRST*, *NO_SAMPLING*}.

Queste sei configurazioni hanno dei box plot molto simili tra loro; infatti, tutti hanno un valore massimo discretamente elevato che varia tra 0.55 e 0.625 circa, un valore minimo non basso come le altre configurazioni, una mediana che raggiunge il massimo al valore 0.4 ed inoltre mantengono una buona variabilità.
Analizzando più specificatamente le configurazioni elencate, si può notare che, per la metrica Kappa, il classificatore *IBK* ottiene buoni risultati con entrambe le tecniche di Features Selection e con le tecniche di Sampling *NO_SAMPLING* e *SMOTE*. Mentre il classificatore *RANDOM_FOREST*, ottiene risultati migliori solo se si applica Features Selection e si utilizzano le stesse tecniche di Sampling di *IBK*.

Inoltre, tutti i risultati ottenuti utilizzando il classificatore *NAIVE_BAYES*, sono molto più bassi rispetto agli altri.

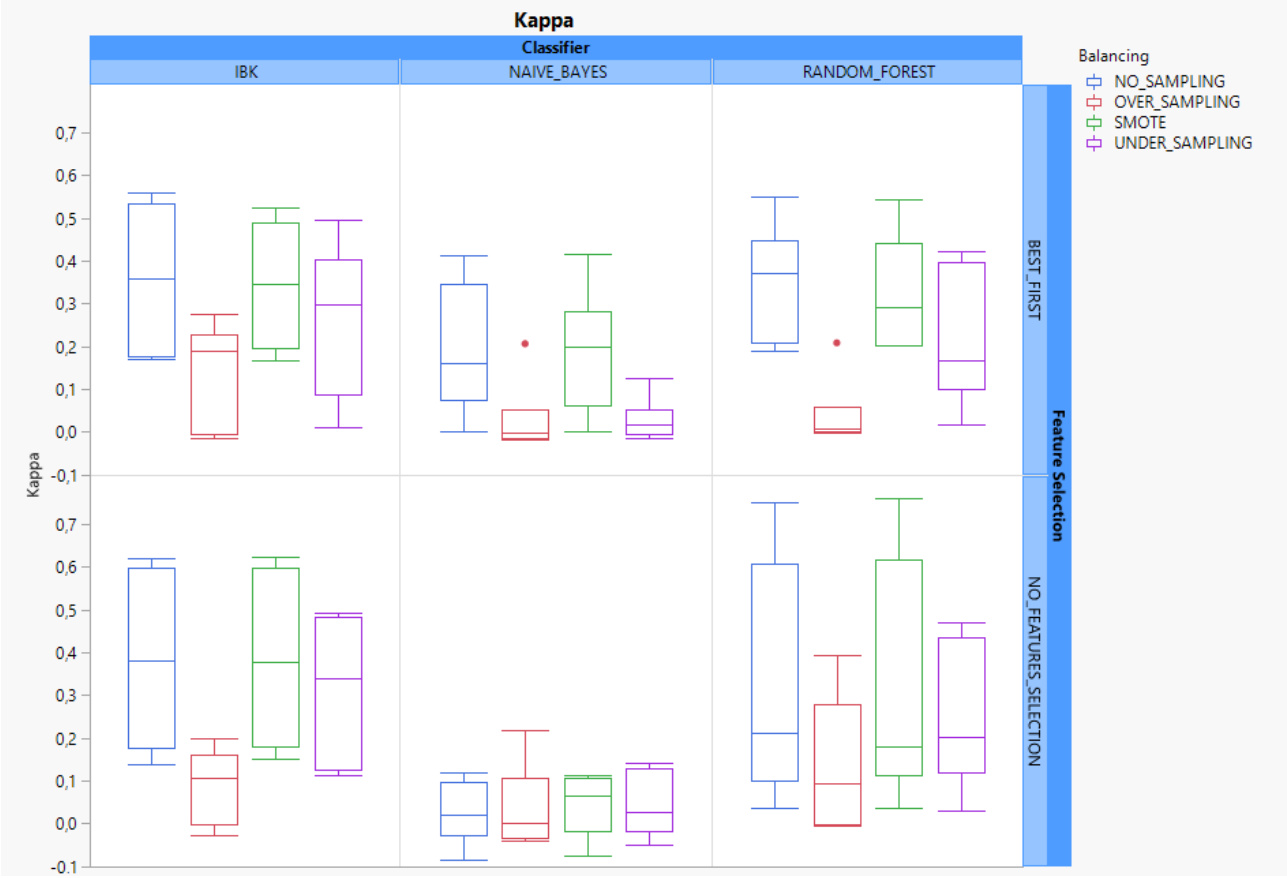

Figura 2 Box Plots Kappa BookKeeper

Precision

La Precision rappresenta, in percentuale, quante volte il classificatore utilizzato ha correttamente classificato un'istanza come positiva. Essa viene calcolata nel seguente modo: $TP/(TP + FP)$, dove:

- TP indica il numero di Veri Positivi (True Positive), ovvero, nell'ambito specifico del report, il numero di File che la valutazione ha indicato come Buggy e che effettivamente lo sono.
- FP indica il numero di Falsi Positivi (False Positive), ovvero, nell'ambito specifico del report, il numero di File che la valutazione ha indicato come Buggy ma che effettivamente non lo sono.

Il valore massimo che la Precision può assumere è 1, mentre il minimo è 0.

Come si evince dalla Figura 3, i valori assunti dalla Precision sono molto alti per quasi tutte le configurazioni. In particolare, si può osservare che con il classificatore *RANDOM_FOREST* senza utilizzare tecniche di Feature Selection, fornisce risultati ottimi per tutte le tecniche di Sampling, tranne che per la tecnica *OVER_SAMPLING*.
Ciò nonostante, il miglior risultato si ottiene con classificatore *IBK*, con la tecnica di Sampling *UNDER_SAMPLING* e senza alcuna tecnica di Feature Selection; infatti, come si può osservare, il valore minimo che assume è circa 0.9, escludendo l'outliers, il valore massimo è pari a 1 e la mediana assume un valore di circa 0.95.
Analizzando più in generale il grafico, per quanto riguarda il Sampling, si evince che la tecnica *OVER_SAMPLING*, con qualsiasi classificatore e Features Selection, fornisce generalmente risultati più bassi;

Considerando, invece, la Feature Selection, i risultati sono molto simili anche se si nota un leggero miglioramento nel momento che non viene applicata la Feature Selection.
Il Classificatore migliore rimane *IBK*.

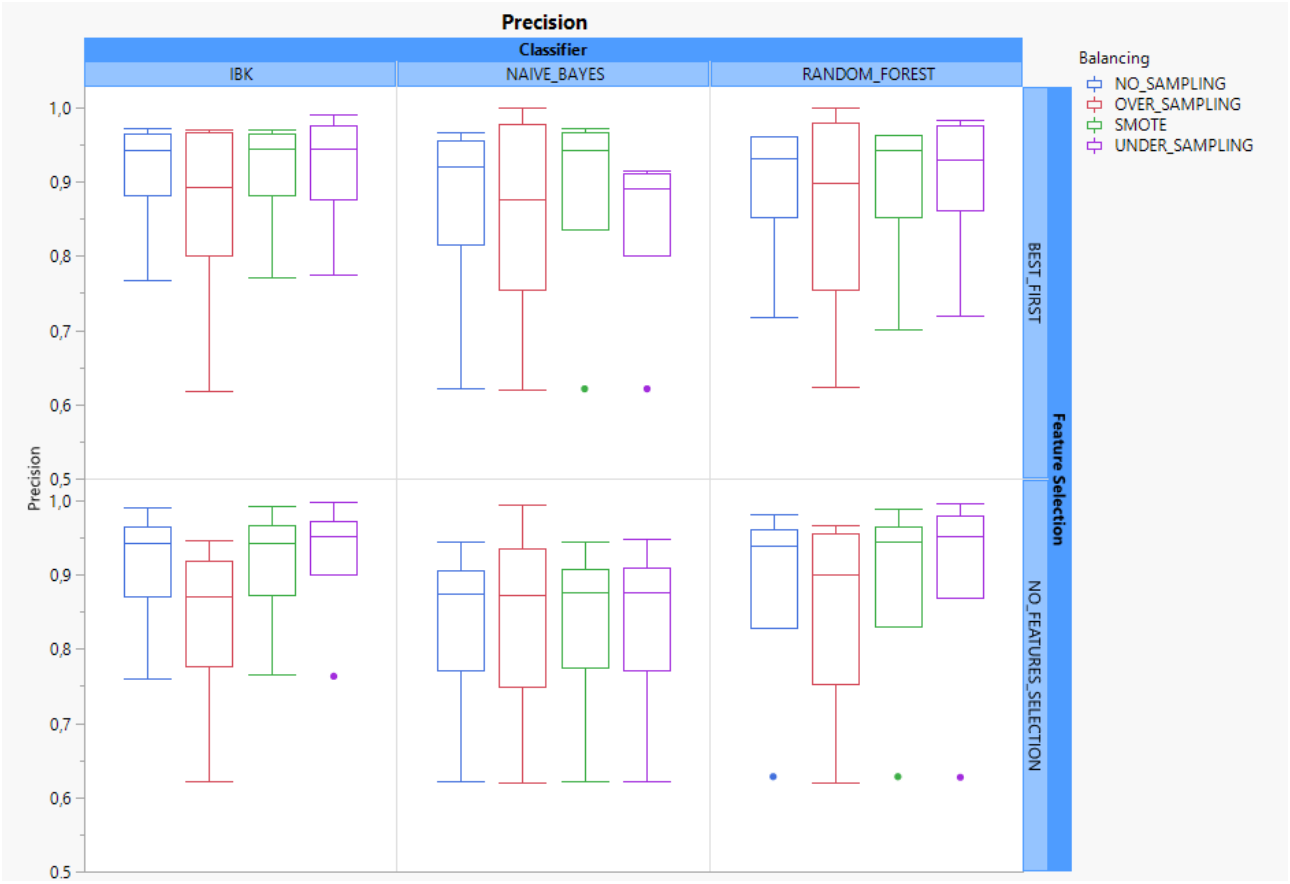

Figura 3 Box Plots Precision BookKeeper.

Recall

La Recall rappresenta, in percentuale, il numero di positivi che il classificatore è stato in grado di classificare. Essa viene calcolata nel seguente modo: $TP\ /\ (TP\ +\ FN)$, dove:

- TP indica il numero di Veri Positivi (True Positive), ovvero, nell'ambito specifico del report, il numero di File che la valutazione ha indicato come Buggy e che effettivamente lo sono.
- FN indica il numero di Falsi Negativi (False Negative), ovvero, nell'ambito specifico del report, il numero di File che la valutazione ha indicato come Non Buggy ma che effettivamente lo sono.

Il valore della Recall può variare tra 0 e 1.

Come si evince dalla Figura 4, la tecnica di Samplig peggiore rimane l'*OVER_SAMPLING*, tranne che nel caso di *RANDOM_FOREST* con *NO_FEATURES_SELECTION*.
In questo caso nessuna delle due tecniche di Features Selection risulta ottenere risultati migliori in generale.
Per quanto riguarda i Classificatori, *NAIVE_BAYES* senza utilizzare nessuna tecnica di Features Selection, dimostra di ottenere i migliori risultati:

- {*NAIVE_BAYES*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*NAIVE_BAYES*, *NO_FEATURES_SELECTION*, *SMOTE*};
- {*NAIVE_BAYES*, *NO_FEATURES_SELECTION*, *UNDER_SAMPLING*}.

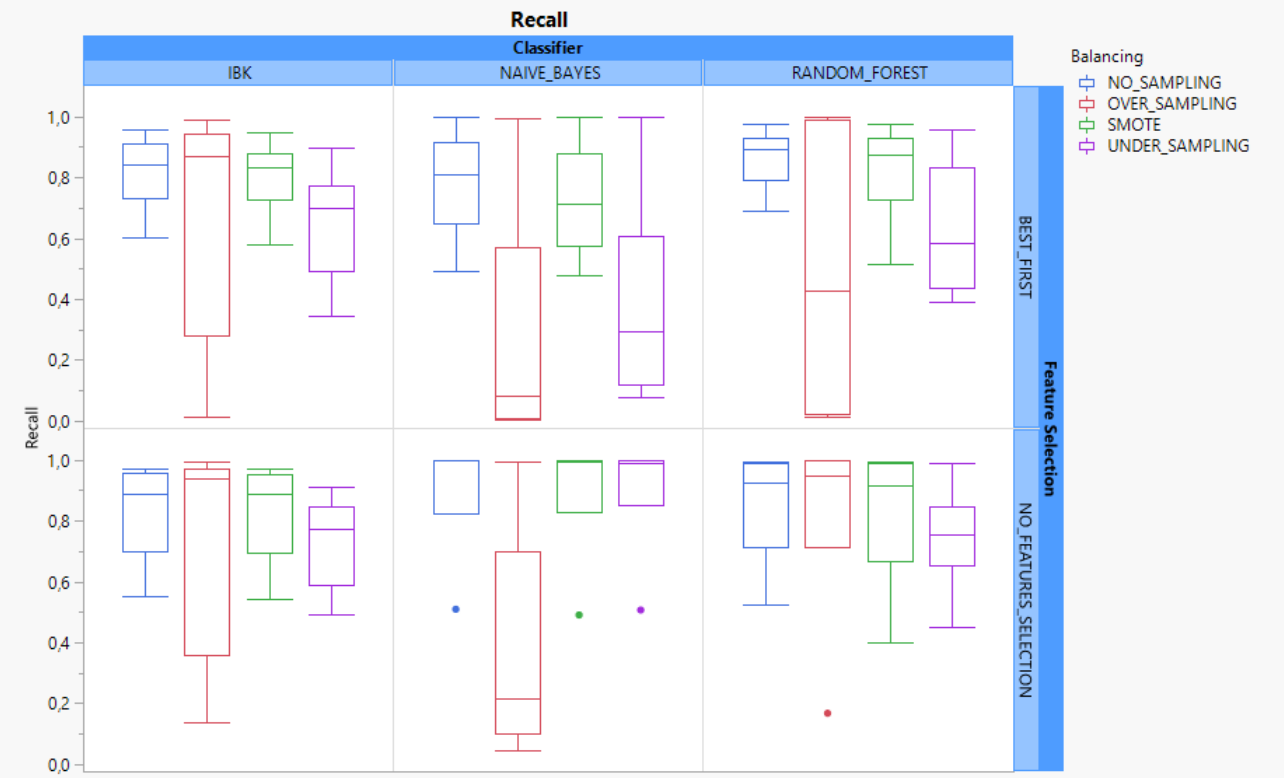

Figura 4 Box Plots Recall BookKeeper.

ROC Area

In questo caso i risultati migliori sono stati ottenuti con le seguenti configurazioni:

- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *SMOTE*};
- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *UNDER_SAMPLING*}.

Per quanto riguarda i classificatori, *NAIVE_BAYES* ha avuto ottimi risultati sia con Features Selection che senza, come si può osservare dalla Figura 5.
Le tecniche di Sampling si dimostrano equivalenti tra loro, tranne che per la tecnica *OVER_SAMPLING*, che ottiene in generale risultati peggiori.

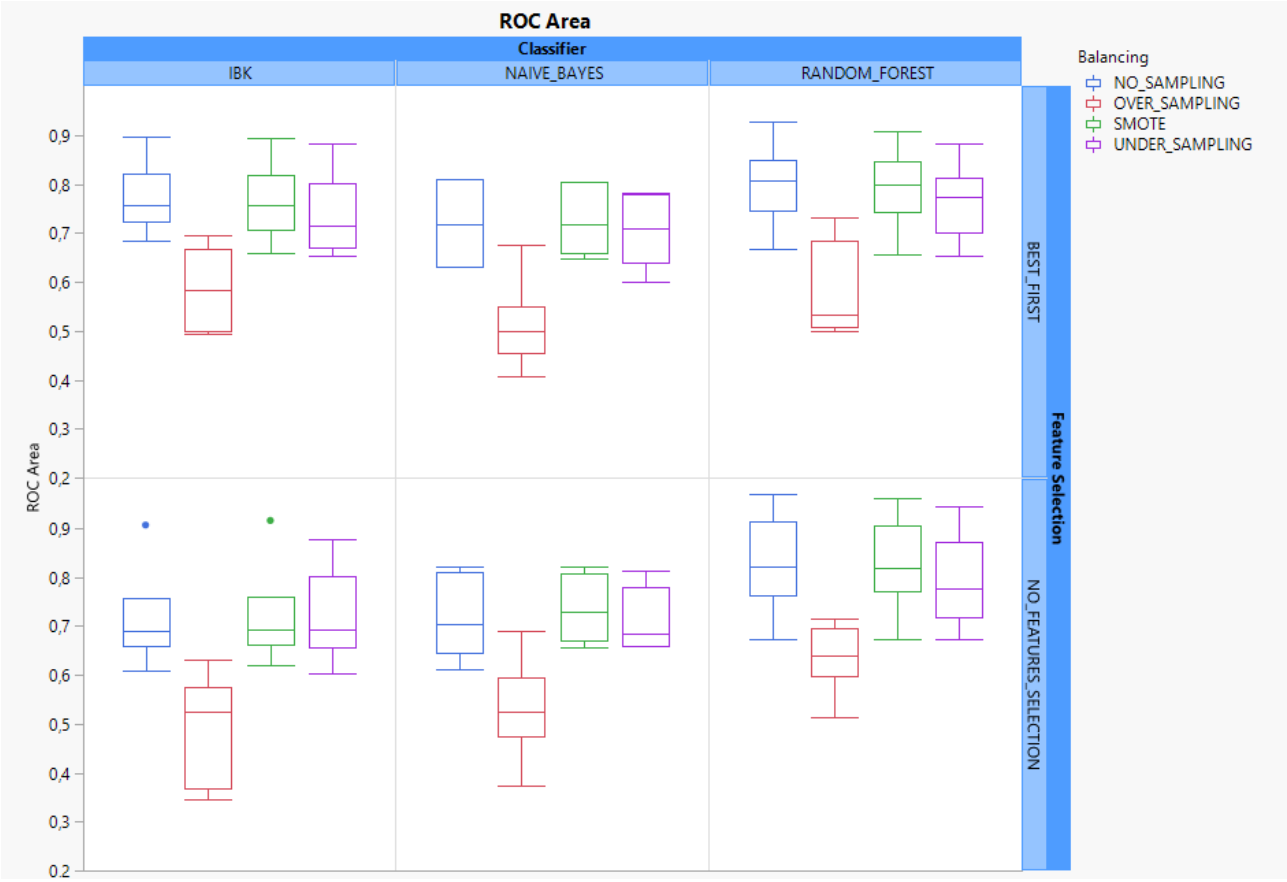

Figura 5 Box Plots AUC BookKeeper.

OpenJPA

Analizziamo, ora, i risultati ottenuti dai Classificatori con tutte le tecniche descritte in precedenza, per il progetto OpenJPA.

Kappa

Come si evince dalla Figura 6, i valori ottenuti dalla metrica Kappa sono migliori, in generale, di quelli ottenuti con il progetto BookKeeper. Anche in questo caso il classificatore *NAIVE_BAYES* oltre ad ottenere risultati non buoni, presenta numerosi outliers, soprattutto con i Sampling *OVER_SAMPLING* e *UNDER_SAMPLING*.

Le configurazioni con una migliore variabilità, ovvero quelle che minimizzano l'ampiezza dell'interquantile, mantenendo comunque un valore della mediana buono sono:

- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *SMOTE*}.

Mentre, le configurazioni che hanno un valore della mediana e del terzo quantile più alto, a discapito di un intervallo di interquantile più elevato sono:

- {*IBK*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*IBK*, *NO_FEATURES_SELECTION*, *SMOTE_SAMPLING*}.

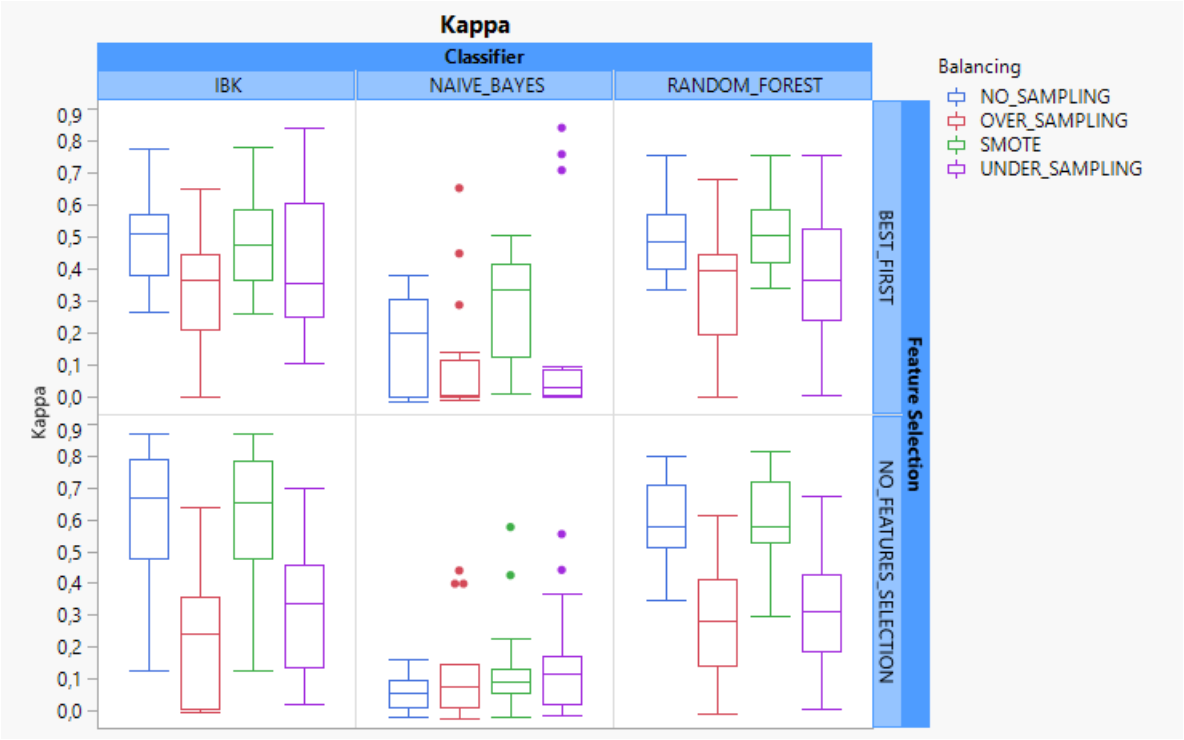

Figura 6 Box Plots Kappa OpenJPA.

Precision

In Figura 7 sono presenti i risultati delle configurazioni utilizzate, per quanto riguarda la metrica Precision. In questo caso, le configurazioni che ottengono i risultati migliori, considerando sempre la mediana, il terzo quantile e l'intervallo di interquantile sono:

- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *SMOTE*}.

Per quanto riguarda le tecniche di Sampling, quelle che ottengono i risultati peggiori in generale sono *OVER_SAMPLING* e *UNDER_SAMPLING*. Infatti, queste tecniche se utilizzate insieme al classificatore *NAIVE_BAYES* e la tecnica di Features Selection, mostrano risultati molto bassi.

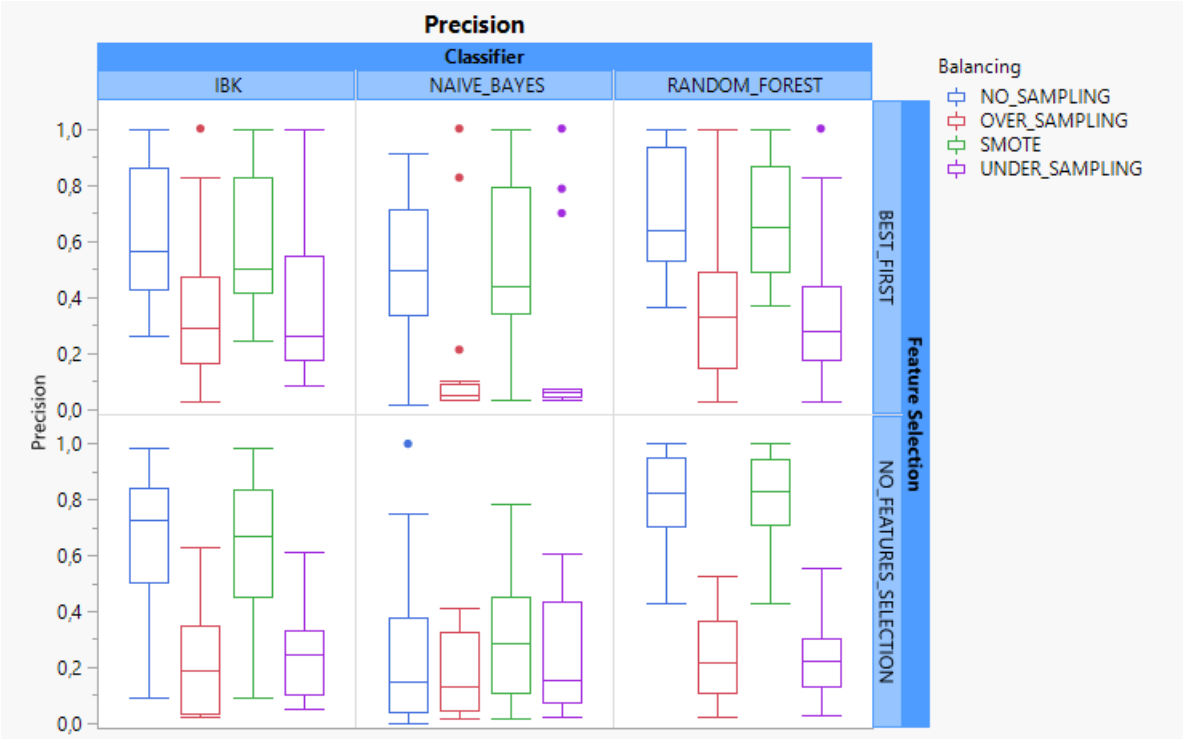

Figura 7 Box Plots Precision OpenJPA.

Recall

In Figura 8 sono presenti i risultati delle configurazioni utilizzate, per quanto riguarda la metrica Recall. In questo caso, la tecnica di Sampling *UNDER_SAMPLING* ha ottenuto tra i risultati migliori per ogni classificatore e tecnica di Features Selection. In particolare, la configurazione che ha ottenuto i risultati migliori, considerando sempre la mediana, il terzo quantile e l'intervallo di interquantile è:

- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *UNDER_SAMPLING*};

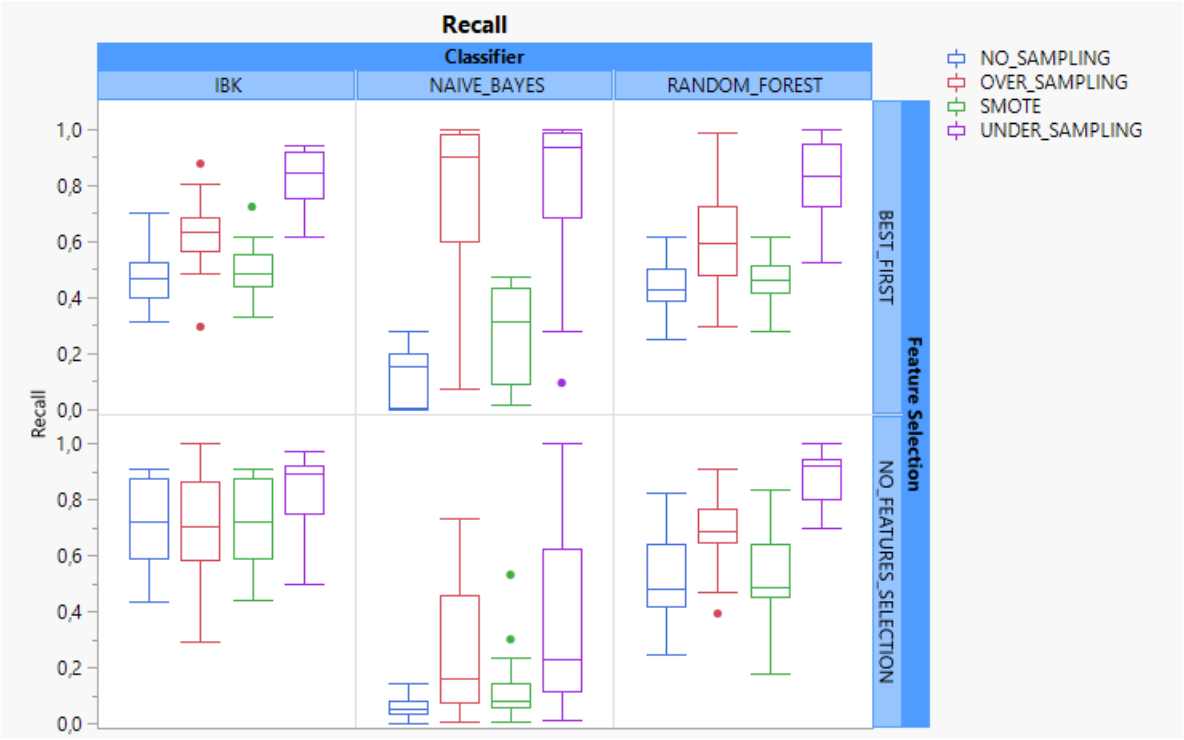

Figura 8 Box Plots Recall OpenJPA.

ROC Area

In Figura 9. sono presenti i risultati delle configurazioni utilizzate, per quanto riguarda la metrica ROC Area. Anche in questo caso, le configurazioni che ottengono i risultati migliori, considerando sempre la mediana, il terzo quantile e l'intervallo di interquantile sono:

- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *NO_SAMPLING*};
- {*RANDOM_FOREST*, *NO_FEATURES_SELECTION*, *SMOTE*}.

In generale, comunque, i risultati ottenuti dalle configurazioni sono molto alti e simili. L'eccezione sono le configurazioni che adottano la tecnica di Sampling *OVER_SAMPLING.*

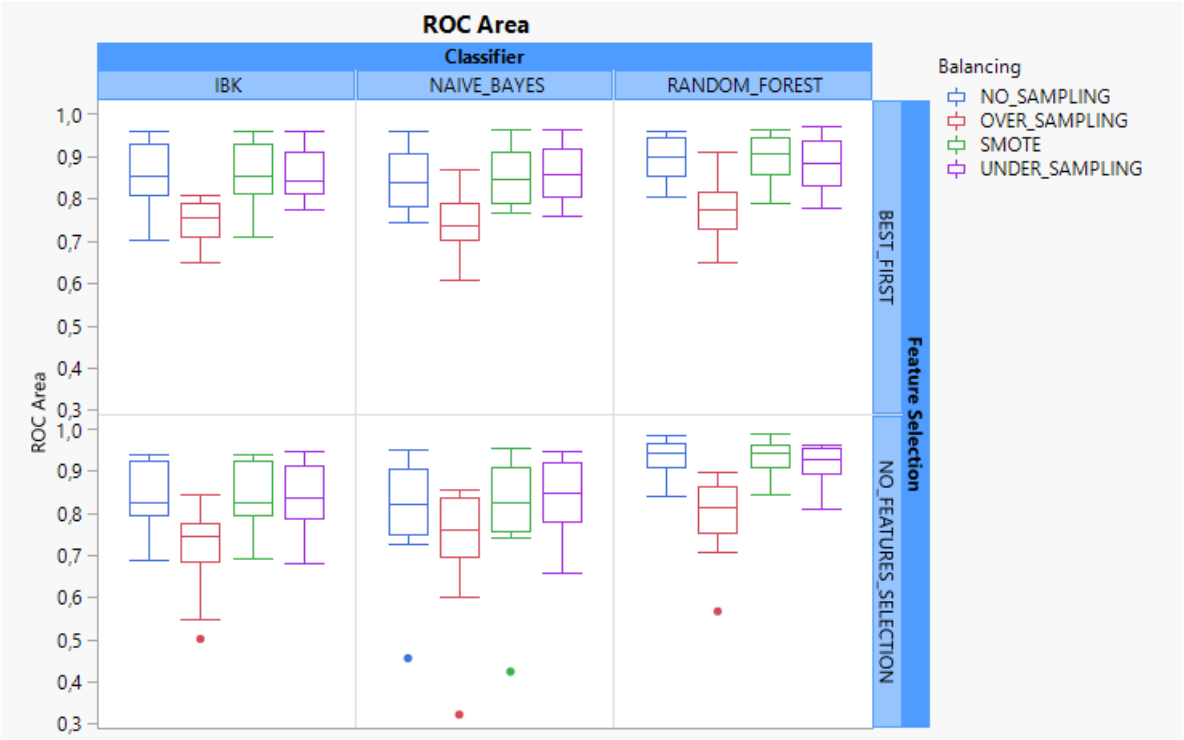

Figura 9 Box Plots AUC OpenJPA.

Conclusioni

In Tabella 1 e in Tabella 2 si possono osservare le configurazioni che hanno ottenuto i migliori risultati, per ogni metrica. Inoltre, sono evidenziate le migliori tecniche di Features Selection e Sampling sia per metrica, sia per classificatore. Queste ultime sono state selezionate, semplicemente, in base alla al numero di volte che tali tecniche sono state considerate le migliori.

Si può osservare come alcune tecniche risultano fornire risultati migliori, in base al classificatore o alla metrica utilizzata, per entrambi i progetti considerati. Per esempio, nel caso del classificatore *RANDOM_FOREST*, sia in BookKeeper che in OpenJPA, le tecniche migliori sono *NO_FEATURES & NO_SAMPLING*.

Invece, per la metrica Precision, in entrambi i progetti si riscontra che non utilizzare una tecnica di Feature che fornisce risultati migliori.

Tabella 1 Riepilogo BookKepeer

	IBK	*NAIVE_BAYES*	*RANDOM_FOREST*	*Tecniche migliori per metrica*
Kappa	NO_FEATURES & SMOTE	BEST_FIRST & NO_SAMPLING	BEST_FIRST & NO_SAMPLING	**BEST_FIRST & NO_SAMPLING**
Precision	NO_FEATURES & UNDER_SAMPLING	BEST_FIRST & SMOTE	NO_FEATURES & UNDER_SAMPLING	**NO_FEATURES & UNDER_SAMPLING**
Recall	NO_FEATURES & NO_SAMPLING	NO_FEATURES & UNDER_SAMPLING	NO_FEATURES & OVER_SAMPLING	**NO_FEATURES**
ROC Area	BEST_FIRST & NO_SAMPLING	NO_FEATURES & SMOTE	NO_FEATURES & NO_SAMPLING	**NO_FEATURES & NO_SAMPLING**

Tecniche migliori per Classificatore	**NO_FEATURES & NO_SAMPLING**	**Nessuna tecnica in particolare**	**NO_FEATURES & NO_SAMPLING**	

Tabella 2 Riepilogo OpenJPA

	IBK	*NAIVE_BAYES*	*RANDOM_FOREST*	*Tecniche migliori per metrica*
Kappa	NO_FEATURES & SMOTE	BEST_FIRST & SMOTE	NO_FEATURES & NO_SAMPLING	**NO_FEATURES & SMOTE**
Precision	NO_FEATURES & NO_SAMPLING	BEST_FIRST & NO_SAMPLING	NO_FEATURES & SMOTE	**NO_FEATURES & NO_SAMPLING**
Recall	BEST_FIRST & UNDER_SAMPLING	BEST_FIRST & UNDER_SAMPLING	NO_FEATURES & UNDER_SAMPLING	**BEST_FIRST & UNDER_SAMPLING**
ROC Area	BEST_FIRST & UNDER_SAMPLING	BEST_FIRST & SMOTE	NO_FEATURES & NO_SAMPLING	**BEST_FIRST**
Tecniche migliori per Classificatore	**UNDER_SAMPLING**	**BEST_FIRST & SMOTE**	**NO_FEATURES & NO_SAMPLING**	

Ambiente di sviluppo, how to e link

Entrambe le Derivable sono state sviluppate su Sistema Operativo Windows a 64 bit.
Per eseguire l'applicazione Milestone 1, è necessario lanciare il main della classe *milestone_one.Main.java*.
Per eseguire l'applicazione Milestone 2, è necessario lanciare il main della classe *milestone_two.Main.java*.
È necessario eseguire le applicazioni in questo ordine.
Nel file di configurazione config.properties, è possibile selezionare il progetto a cui far riferimento, modificando il valore della proprietà PROJECT. Inoltre, sempre in tale file è possibile configurare la variabile *updateFiles*. Se settata a 0, il programma utilizza i file del progetto presenti nel file di appoggio; se settata ad 1, il programma recupera i file del progetto direttamente dal repository.

Github:

- Link Derivable 1: https://github.com/marcomarcucci30/Derivable1;
- Link Derivable 2: https://github.com/marcomarcucci30/Derivable2.

Sonar Cloud:

- Link Derivable 1: https://sonarcloud.io/dashboard?id=marcomarcucci30_Derivable1;
- Link Derivable 2: https://sonarcloud.io/dashboard?id=marcomarcucci30_Derivable2.