Ingegneria del Software 2 -2019/20

Relazione Progetto Sezione Falessi

Machine Learning for Software Engineering

Di Simone Mesiano Laureani

0278325

Deliverable 1

Introduzione

La Deliverable 1 consiste nel misurare la stabilità di un attributo di un progetto open-source nelle modalità viste durante il corso. Il progetto analizzato è Apache MAHOUT [1]: un framework distribuito realizzato per la progettazione di applicazioni di Machine Learning da parte di matematici, statistici e data scientists. Lo sviluppo del progetto in esame è stato avviato nel 2008 e ancora oggi riceve piccole modifiche. L'attributo su cui è stato analizzato il progetto è il numero di bugs fixed. Lo scopo di questa analisi è di introdurre un processo di controllo di tipo statistico, sulla base dei concetti espositi del CMMI, per trovare e correggere problemi oltre che per determinarne la stabilità. Inoltre, è possibile utilizzare questi concetti per predire i risultati attesi da un processo in termini di media e deviazione standard. Il process control chart tiene traccia di questi elementi sviluppando sull'asse delle ordinate il numero di bugs fixed e sull'asse delle ascisse viene indicato un arco temporale.

Progettazione

Per lo sviluppo di questa Deliverable si è reso necessario effettuare un *clone* del progetto in esame per poter lanciare in seguito comandi Git-based in grado di soddisfare le specifiche richieste. Durante la progettazione si è dovuto gestire un problema riguardante l'acquisizione dei commit associati a uno specifico ticket Jira basandosi sui commenti associati a questi ultimi. Nello specifico si è scoperto che ricercando su Git tutti i commit, per esempio, associati al ticket "Mahout -18" tramite il comando git *grep,* venivano ottenuti di ritorna anche i ticket per "Mahout -180" "Mahout -181" ecc. Tale problema si è risolto banalmente aggiungendo ":" al termine del ticket ID (quindi per es. "Mahout 18:"). Si è notato però che questo approccio elimina anche dei falsi negativi, cioè esistono dei commit appartenenti a quel specifico ticket ID però senza i due punti finali e che quindi non vengono calcolati nelle statistiche di questo e del successivo Deliverable. Come data di "fixed bug" si è intesa quella relativa all'ultimo commit contenente il ticket ID nel proprio commento.

La Deliverable in questione è stata implementata in due modi diversi: il primo è quello che si sviluppa definendo come arco temporale quello mensile. Ciò indica che per ogni mese dal 2008 fino ad oggi si è raccolto il numero totale di commit avvenuti in quel mese. La raccolta di questi dati è stata effettuata da Jira [2]. Su questi commit si sono applicati dei filtri per ottenere solo quelli relativi a fix di bugs, con tickets associati in stato risolto o chiuso e risoluzione fixed. Il

risultato è mostrato in *Figura 1.* Si noti come la linea riguardante il lower bound non sia presente nel grafico, ciò è dovuto al fatto che sarebbe stata di valore negativo, di conseguenza si è preferito non mostrarla in quanto di scarsa utilità ai fini del controllo statistico.

La seconda implementazione di questo Deliverable è stata presa in considerazione a causa della scarsità del numero di fix bugs ottenuti dalla prima. Di conseguenza si è scelto di evolvere tale rappresentazione in quella mostrata in *Figura 2* che raggruppa i vari commit per anno. In questo modo si può vedere come si riesce ad ottenere una maggior chiarezza sull'andamento del progetto a scapito però di una maggiore granularità temporale (che quindi volge dal mensile all'annuale).

Considerazione dei risultati

I risultati, specialmente quelli dell'implementazione con granularità annuale, evidenziano a grandi linee una tipica curva a campana (nota come campana di Gauss) con una crescita graduale nei primi anni di sviluppo del progetto fino ad arrivare ad un picco nel 2013, che quindi possiamo identificare come l'anno con maggior attività in termini di bugs fixed. Oltre questo picco si può notare un lento decadimento fino ad arrivare a valori nulli riconducibili ai giorni odierni. Questo decadimento può essere spiegato dal fatto di un progressivo abbandono del progetto in esame nell'ottica di dedicare le proprie risorse verso il nuovo framework Apache Spark più performante [3] [4].

Deliverable 2

Introduzione

La Deliverable 2 consiste nell'eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling e di feature selection sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di due grandi applicazioni open-source. Nella scelta dei progetti secondo l'algoritmo proposto dal professore, si è adottato l'ordine alfabetico internazionale da 26 lettere (diverso da quello italiano che non prevede le lettere inglesi). Nello specifico si sono analizzati i progetti Apache Bookkeeper [5] e OpenJPA [6]. Il primo è un servizio di storage a bassa latenza, fault-tolerant e scalabile, ottimizzato per sistemi real-time. Offre meccanismi di replicazione e consistenza rendendosi performante per una grande varietà di use-cases. OpenJPA invece è un progetto java per la persistenza che può essere integrato in qualsiasi container Java EE compliant e molti altri framework come Tomcat e Spring. La Deliverable 2 si propone quindi si identificare, per ogni versione di entrambi i progetti, le classi java definite "buggy" per poterne studiare le caratteristiche sulla base di metriche indicate dal paper fornito durante il corso. **Di queste 16 metriche, si è scelto di implementarne 10** e sono identificabili perchè evidenziate in giallo nella *Figura 3* .

Progettazione

Per l'implementazione di questa Deliverable si è scelto di eseguire una fork su entrambi i progetti in modo da poter lanciare comandi Git su di essi tramite shell locali. Questo metodo è stato

preferito rispetto a quello di utilizzare il protocollo REST che, vista la grande mole di operazioni previste, avrebbe richiesto svariate decine di ore a cause del limite di query per un limite di tempo. Il clone dei progetti viene fatto sulla stessa cartella in cui si sta eseguendo il programma java (ciò è dovuto al fatto che una eventuale variazione della locazione provocherebbe la comparsa di code smells su SonarCloud [7] dovuti ad hard-coded path). Occorre far notare inoltre l'esistenza di molti tickets Jira a cui non corrisponde nessun commit associato. Come data di commit è stata presa in considerazione quella del commiter.

I file presi in considerazione nel dataset sono solo i *.java* e solo quelli presenti all'ultima versione disponibile in data odierna (quindi *i file eliminati, rinominati o spostati nel corso della storia dei progetti non sono riportati nel dataset*).

Riguardo alla metrica Size dei file per release: si sono prese il numero di linee di codice aggiunte dalla prima release fino alla data (inclusa) della versione in considerazione. Occorre far notare come alcune delle metriche siano state intese come "*storiche" (*quindi in continuo aumento nel tempo), mentre altre invece sono state calcolate per release (quindi è possibile ottenere dei risultati minori nel tempo). Tra le metriche considerate storiche si hanno: SIZE, NR, NAuth e le metriche CHURN-based. Invece le rimanenti tecniche, quindi le LOC-based sono state calcolate nell'ambito della singola versione presa in esame, in quanto l'idea di fondo è che modificando molto in una sola release allora è più probabile introdurre bugs. Un altro motivo per il quale si è presa questa decisione è stato quello di evidenziare meglio la contrapposizione con le metriche churn-based che altrimenti sarebbero state molto simili alle LOC-based. Inoltre si segnala che per le metriche AVG_LOC_ADDED e LOC_ADDED si sono prese in considerazione solo le righe aggiunte (cioè non si è tenuto conto dei commit con più righe di codice cancellate che righe inserite per non ottenere valori negativi).

Occorre notare che le metriche, come indicato tra le specifiche della Deliverable, sono state calcolate solo sulla metà delle release meno recenti. Ciò è necessario per evitare che la presenza di bugs dormienti potesse inficiare la bontà di questa analisi. Per quanto concerne la bugginess si è provveduto, al contrario, a calcolarla su tutto l'arco temporale fino ad oggi. Per tale calcolo si è ricorso ai ticket Jira con almeno una affected version indicata e consistente con la fixed version e opening version proposta. Dove ciò non era possibile, si è adottata la tecnica della Proportion vista durante il corso.

Come fixed version si è presa sempre la versione successiva a quella dell'ultimo commit. Si noti che il numero di file ottenuto nel dataset aumenta con l'aumentare della versione a causa della creazione di tali file nella release in esame. In *Figura 4* e in *Figura 5* è possibile vedere i risultati ottenuti al termine della Milestone 2 di questa Deliverable utilizzando la tecnica di validazione "Walk Forward". Per osservare più nel dettaglio il comportamento dei classificatori al variare del numero di Training Release, si sono generati i grafici in *Figura 6* per Bookkeeper e in *Figura 7* per OpenJPA. In essi possiamo riconoscere un generale aumento dell'accuratezza delle metriche all'aumentare delle training release, specialmente per il progetto OpenJPA. Infine, dalla *Figura 8* in poi è possibile vedere i risultati ottenuti per l'ultima Milestone del Deliverable 2.

Considerazione dei risultati

Una volta osservati i risultati finali è possibile rispondere alla domanda su quali tecniche di feature selection o di balancing aumentano l'accuratezza dei classificatori.

Precision:

- Per il classificatore Random Forest con il dataset Bookeeper e anche con il dataset OpenJPA possiamo affermare che la Precision migliore si ottiene senza l'utilizzo di balancing e senza feature selection.
- Per il classificatore NaiveBayes, la Precision migliore si ottiene senza feature selection e con balancing SMOTE per il dataset Bookkeeper, mentre invece per OpenJPA risulta migliore l'applicazione di SMOTE balancing con feature selection.
- Per il classificatore IBK la Precision migliore si ottiene senza balancing con feature selection sia per Bookkeeper che per OpenJPA.

Recall:

- Per il classificatore Random Forest con il dataset Bookeeper e anche con il dataset OpenJPA si può affermare che la Recall migliore si ottiene con undersampling e senza feature selection.
- Per il classificatore NaiveBayes, la Recall migliore si ottiene con feature selection e con SMOTE balancing per il dataset Bookkeeper e per OpenJPA.
- Per il classificatore IBK la Recall migliore si ottiene con undersampling con feature selection per Bookkeeper e senza di essa per OpenJPA.

Roc Area:

- Per il classificatore Random Forest con il dataset Bookeeper e anche con il dataset OpenJPA si può affermare che la ROC area migliore si ottiene senza sampling e senza feature selection.
- Per il classificatore NaiveBayes, la ROC area maggiore si ottiene senza feature selection e con SMOTE balancing per il dataset Bookkeeper, mentre per OpenJPA risulta migliore l'applicazione della feature selection con SMOTE balancing.
- Per il classificatore IBK la ROC area migliore si ottiene con un oversampling senza feature selection per Bookkeeper, mentre per OpenJPA risulta migliore l'applicazione della feature selection però senza sampling.

Kappa:

- Per il classificatore Random Forest con il dataset Bookeeper e anche con il dataset OpenJPA si può affermare che il fattore Kappa migliore si ottiene senza sampling e senza feature selection.
- Per il classificatore NaiveBayes, il fattore Kappa maggiore si ottiene senza feature selection e con SMOTE balancing sia per il dataset Bookkeeper che per OpenJPA.
- Per il classificatore IBK il fattore Kappa migliore si ottiene adottando un oversampling con feature selection per Bookkeeper, mentre per OpenJPA risulta migliore non adottare la feature selection né sampling.

Risultati

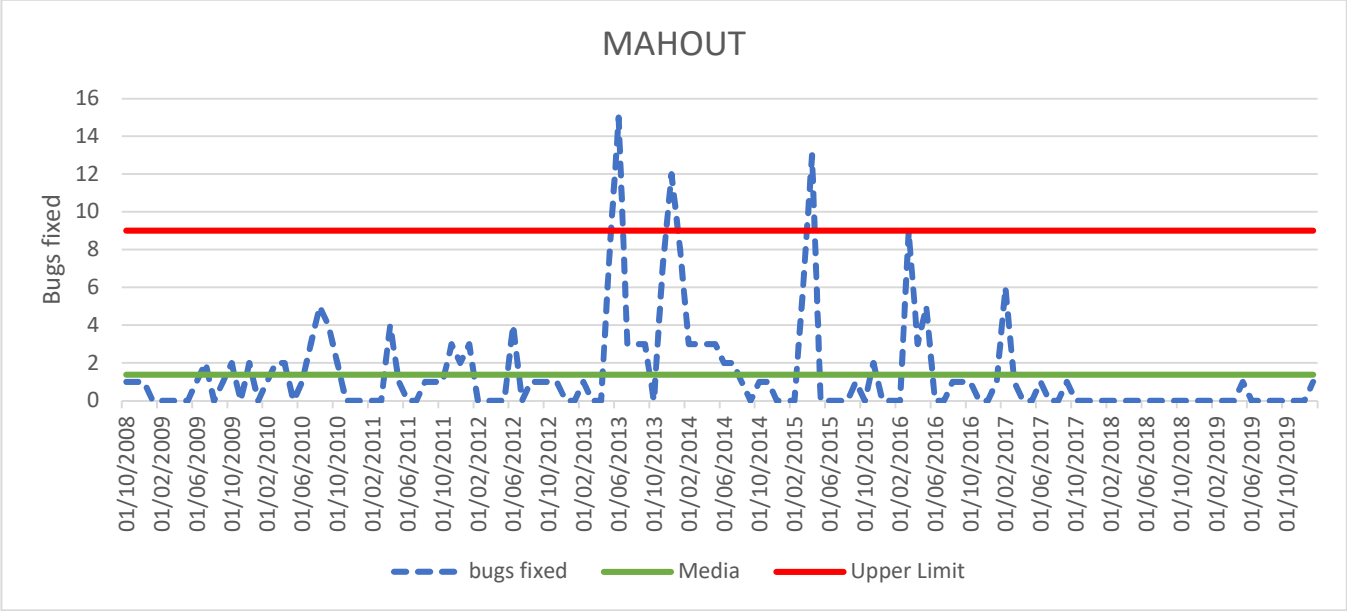

Figure 1 Process Control Chart (months)

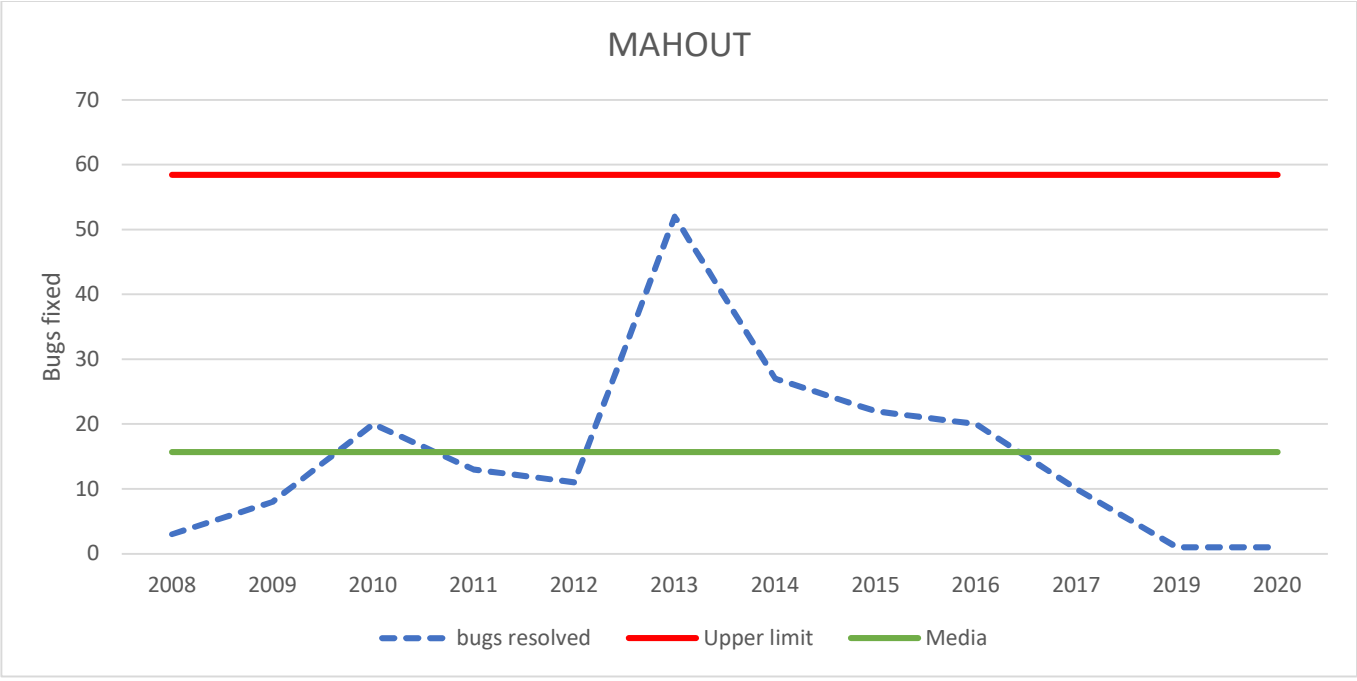

Figure 2 Process Control Chart (years)

METRICS

Metric (File C)	Description
Size	LOC
LOC_touched	Sum over revisions of LOC added+deleted+modified
NR	Number of revisions
NFix	Number of bug fixes
NAuth	Number of Authors
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 revisions of added – deleted LOC in C
MAX_Churn	MAX_CHURN over revisions
AVG_Churn	Average CHURN per revision
ChgSetSize	Number of files committed together with C
MAX_ChgSet	Maximum of ChgSet_Size over revisions
AVG_ChgSet	Average of ChgSet_Size over revisions
Age	Age of C in weeks
WeightedAge	Age weighted by LOC_touched
NSmells	Number of smells

Figure 3 Metriche selezionate

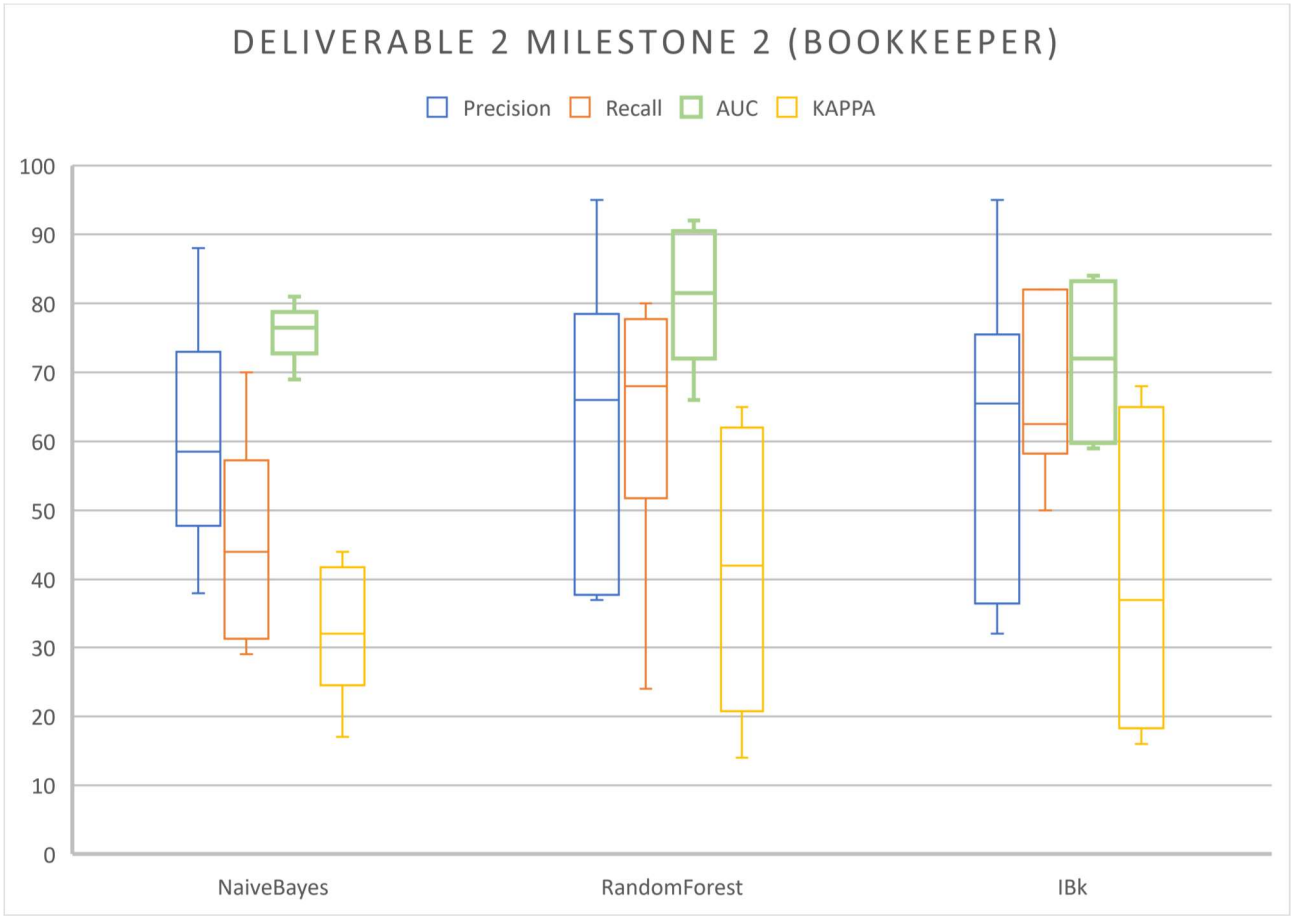

Figure 4

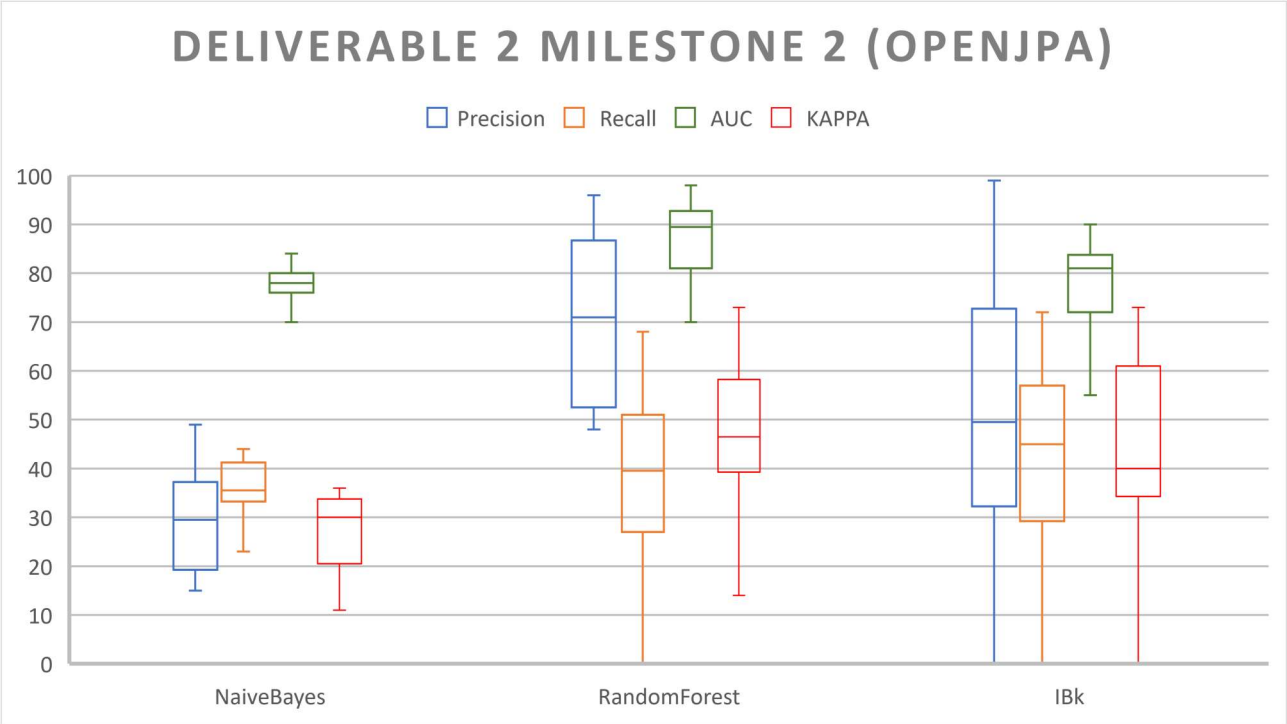

Figure 5

Figure 6 Deliverable 2 Milestone 2

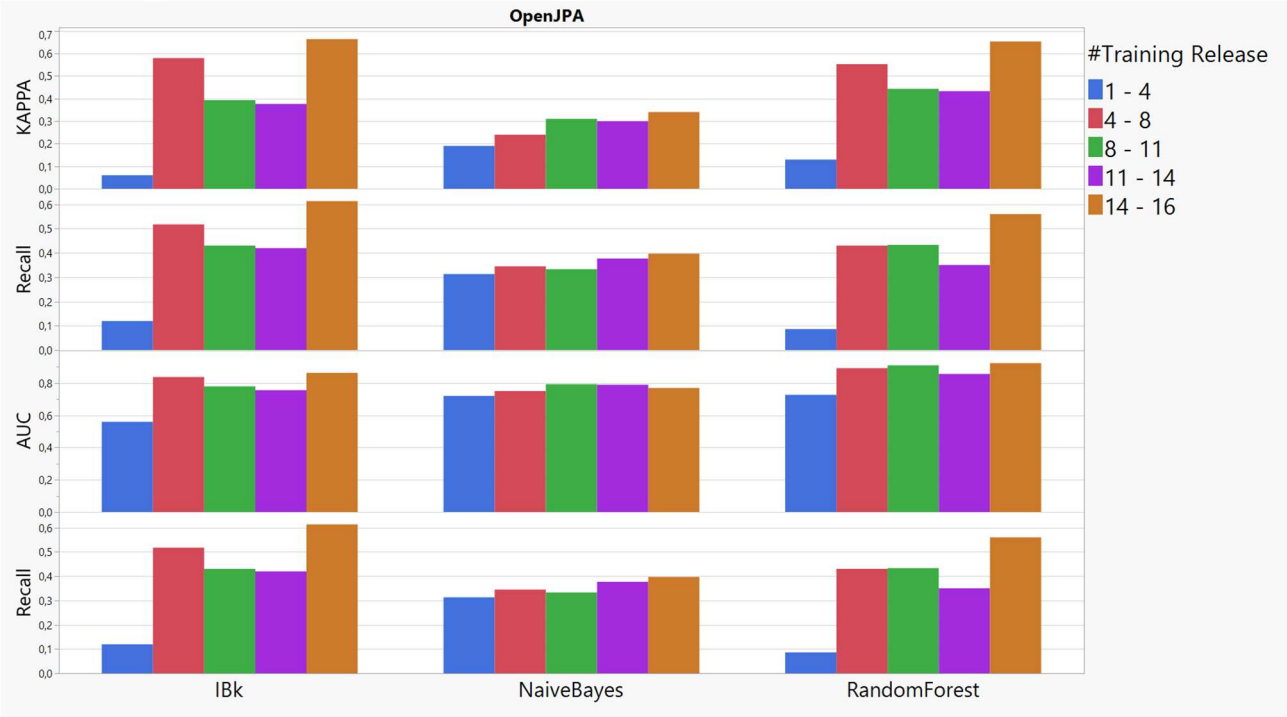

Figure 7 Deliverable 2 Milestone 2

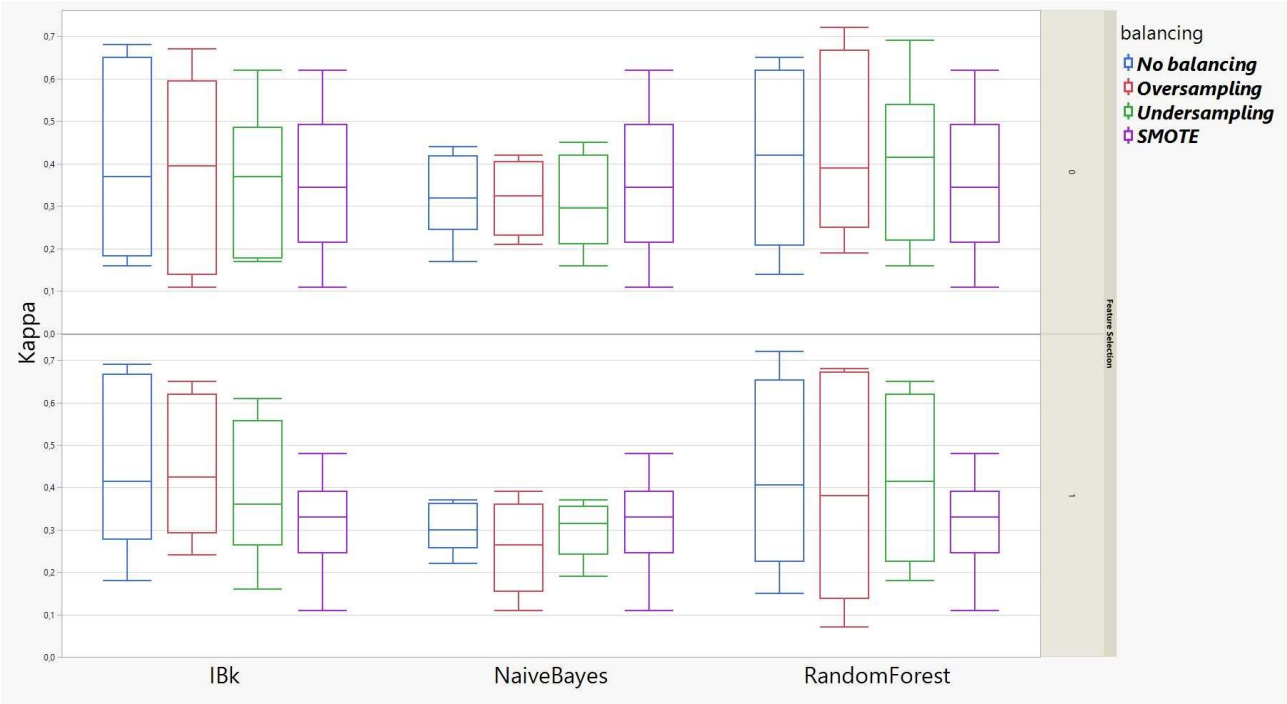

Figure 8 Bookkeeper Kappa

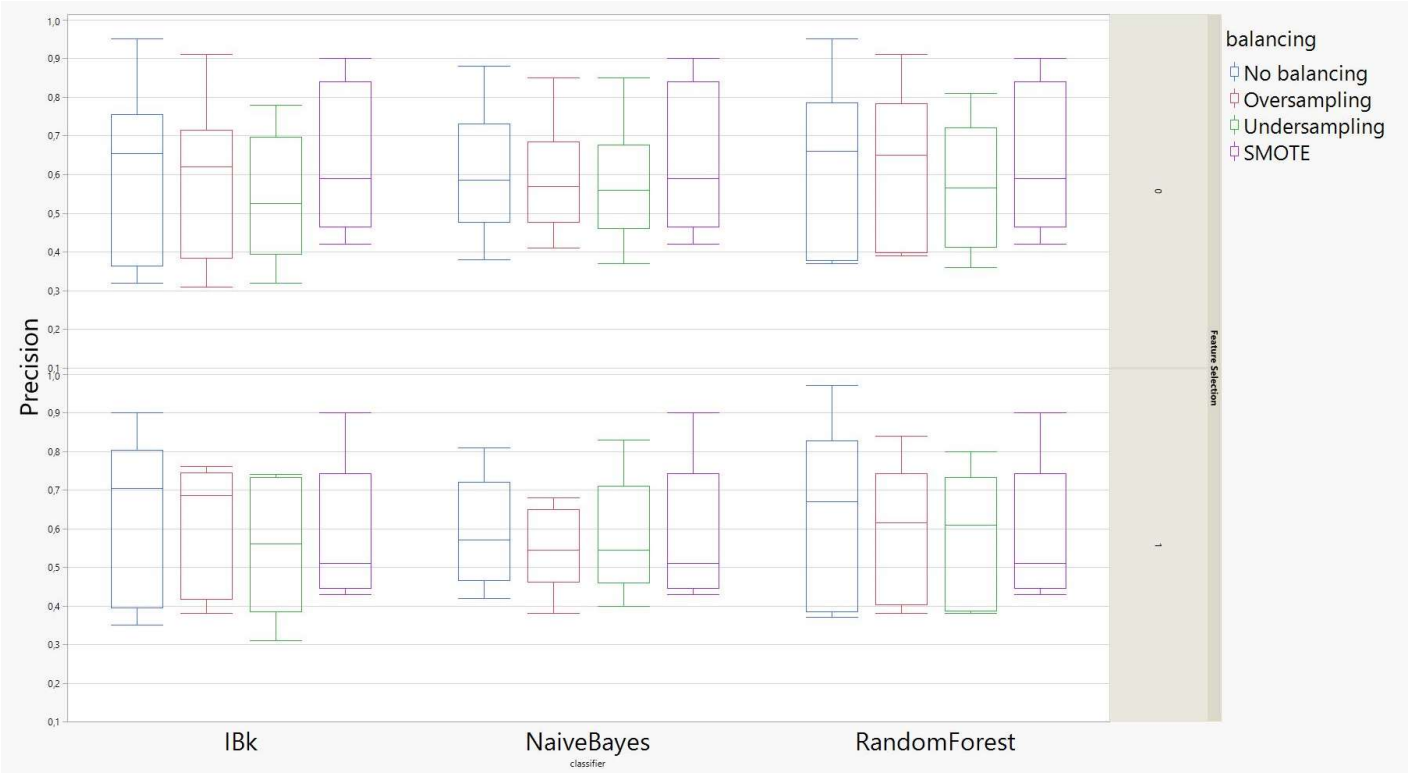

Figure 9 Bookkeeper Precision

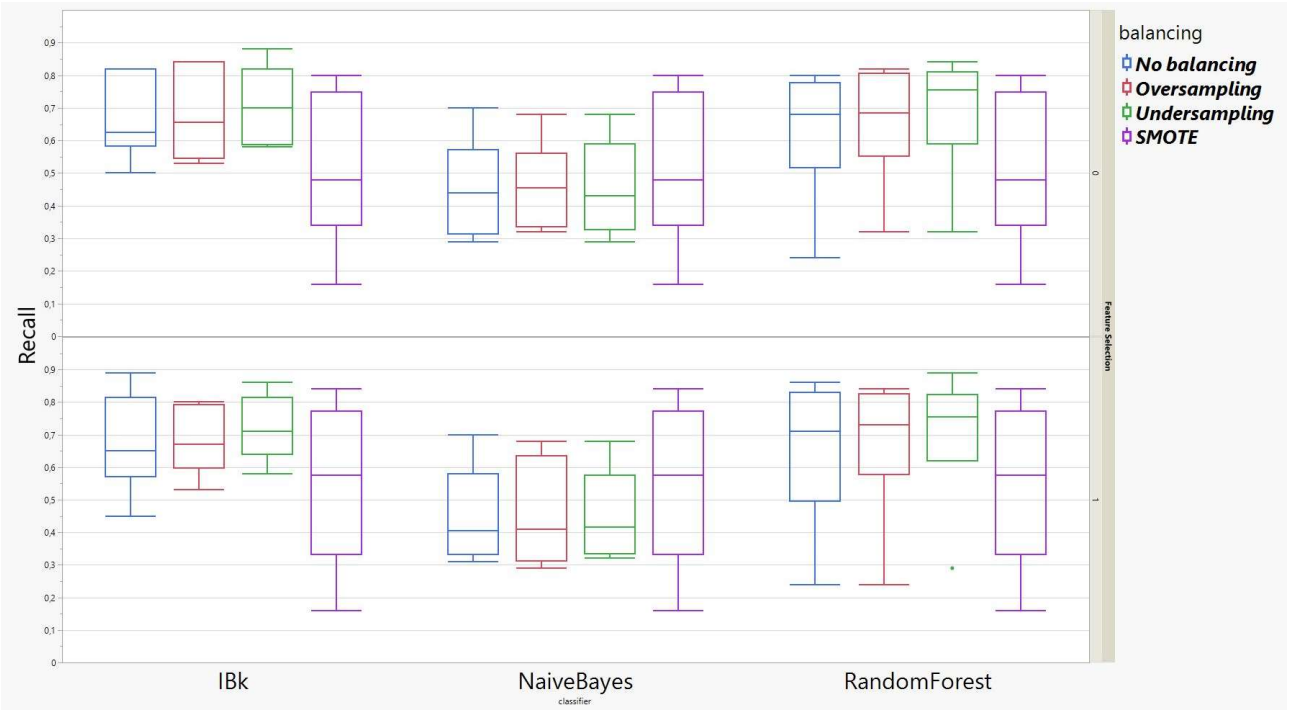

Figure 10 Bookkeeper Recall

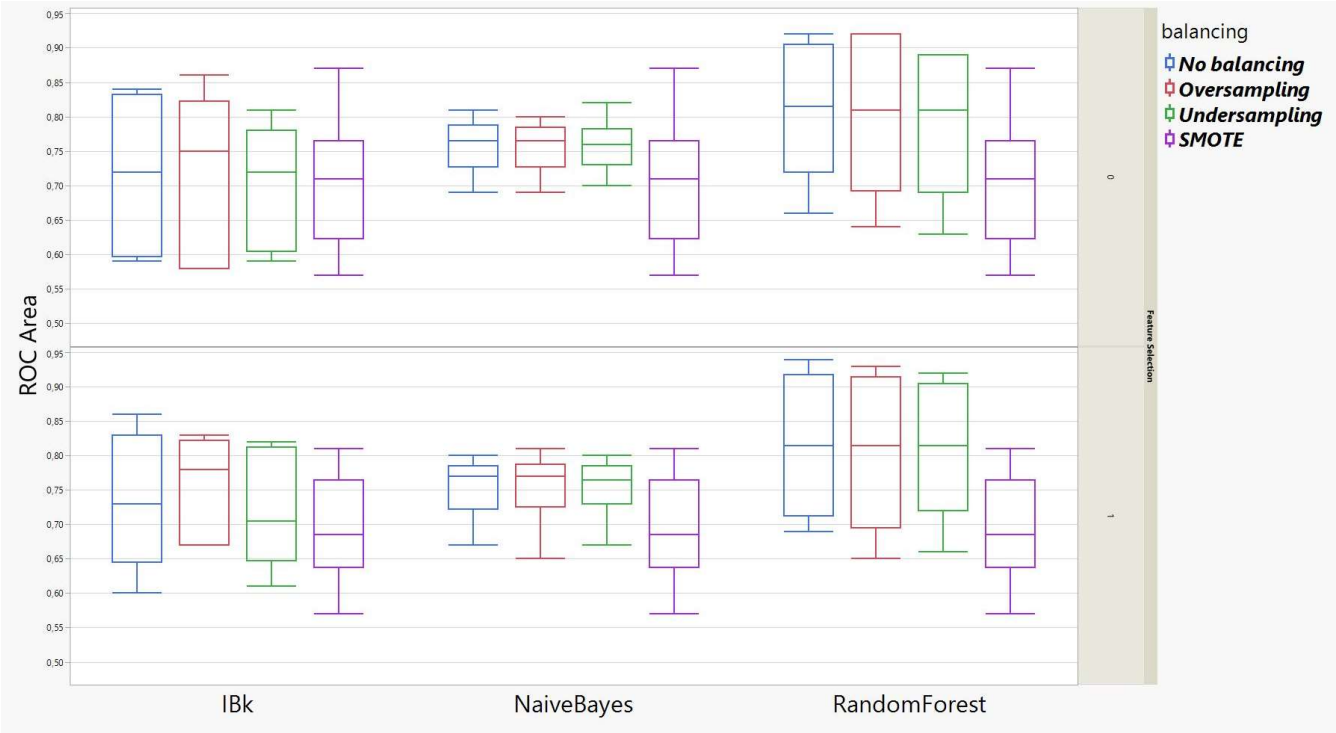

Figure 11 Bookkeeper ROC Area

Figure 12 Bookkeeper all metrics

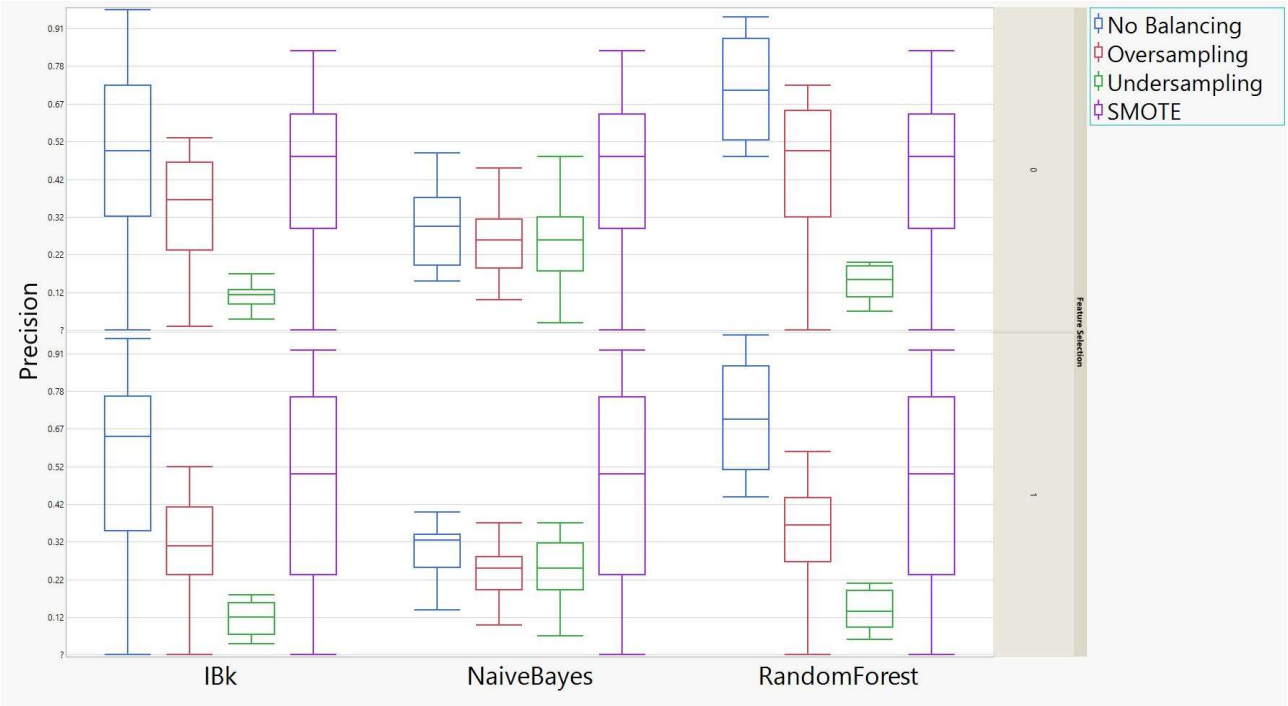

Figure 13 OpenJPA Precision

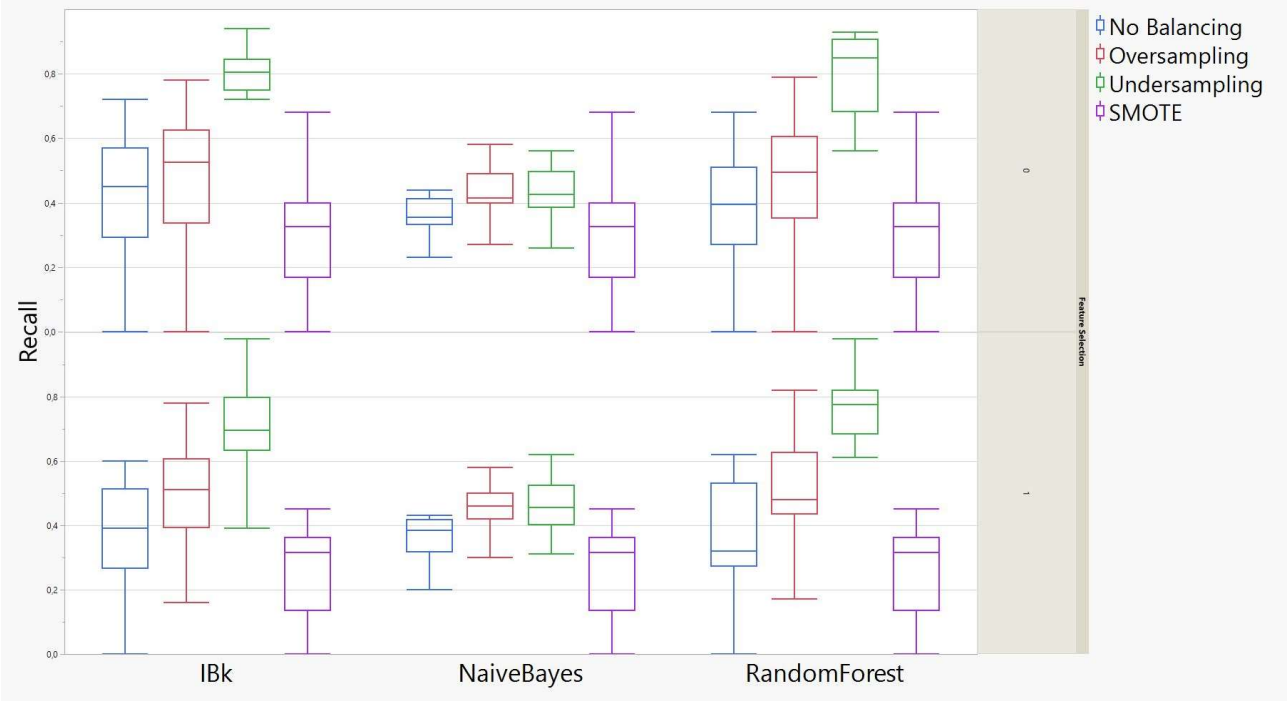

Figure 14 OpenJPA Recall

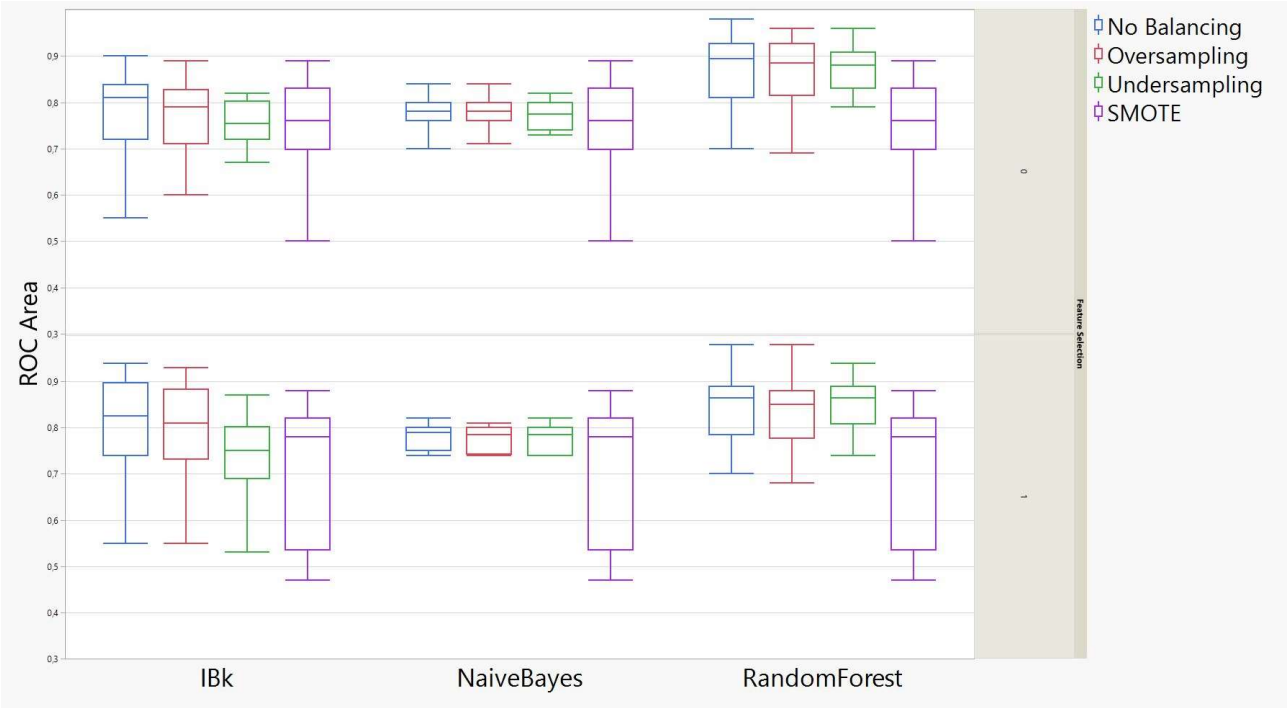

Figure 15 OpenJPA ROC Area

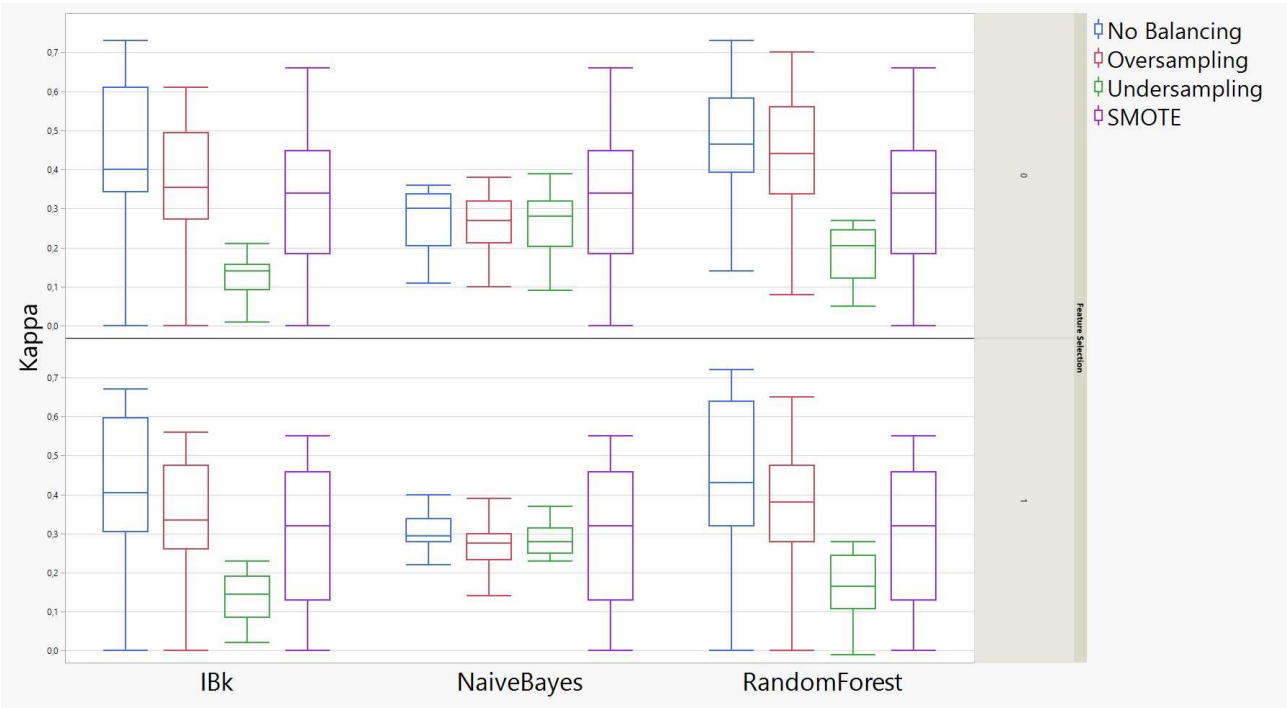

Figure 16 OpenJPA Kappa

References

[1] https://mahout.apache.org

[2] https://issues.apache.org/jira

[3] https://www.quora.com/Is-Apache-Mahout-dying-because-of-MLlib-H2O-and-0xdata

[4] https://stackoverflow.com/questions/23511459/what-is-the-difference-between-apache-mahout-and-apache-sparks-mllib

[5] https://bookkeeper.apache.org/

[6] http://openjpa.apache.org/

[7] https://sonarcloud.io