Deliverable 2

MARCO CALAVARO

MATRICOLA: 0295233

RoadMap

- Introduzione p.3
- Progettazione p.5
  - Struttura generale
  - Costruzione del Dataset
  - Analisi tramite Weka
- Variabili p.9
  - Del Dataset
  - Di weka
- Risultati p.15
  - Introduzione
  - BookKeeper
  - ZooKeeper
- Discussione dei risultati e conclusioni p.21
- Link p.26

Introduzione

L'obiettivo di questo Deliverable è stato quello di effettuare uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling, classificazioni sensibili al costo, e feature selection, sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni open-source.

I progetti Apache analizzati sono: BookKeeper, ZooKeeper

Per la produzione dei risultati è stato sviluppato un solo codice descritto nel capitolo di Progettazione.

Ambiente di sviluppo: sistema operativo Windows 10 e Eclipse come IDE.

Deploy: il progetto è stato caricato su GitHub, con collegati sia TravisCI che SonarCloud.

- I link sono disponibili alla pagina finale.

Introduzione

Per i progetti analizzati sono state utilizzate le seguenti versioni:

- BookKeeper versione 4.5.2: è una versione piuttosto vecchia di tale progetto, tale scelta è dovuta al ticket BOOKKEEPER-1107 nel viene interrotto l'utilizzo di Jira per il mantenimento di ticket e poiché l'analisi parte proprio dai risultati che tale sito offre non era necessario usufruire di versioni superiori.
- ZooKeeper versione 3.8.0: ultima versione disponibile per tale progetto.

Librerie utilizzate:

- Per il recupero dei ticket sono state usate le API di Jira in combinazione a Json per l'analisi delle query a tali servizi
- Per ottenere informazioni dal version control system è stato utilizzata la libreria Jgit
- Per l'analisi del dataset è stata utilizzata la libreria di Weka

Progettazione – Struttura generale

Il codice per poter ottenere i dati di analisi è stato realizzato suddividendo il suo funzionamento in due parti che seguono la seguente logica:

- Prima parte: realizzazione del dataset attraverso l'analisi del repository del progetto e degli issue di tipo buggy recuperati da Jira.
  - Il programma è stato eseguito con un clone locale del repository effettuato prima del lancio effettivo, ma è stata sviluppata anche la parte che lo esegue in automatico. (In questo caso viene recuperata l'ultima versione disponibile)
- Fase intermedia: conversione del dataset da file csv ad arff e rimozione degli attributi superflui nell'analisi di Weka . (Nello specifico viene rimosso l'attributo "File name")
- Seconda parte: Analisi del dataset attraverso la libreria Weka e salvataggio di essi nel file (…_result.csv)

Progettazione - Costruzione del Dataset

In questa fase del progetto viene creato il file projectName_dataset.csv che conterrà le misurazioni effettuate sulle varie release del progetto.

Sono state misurate 12 metriche differenti descritte nel capitolo Variabili.

Tramite Jira vengono recuperati i ticket dei bug con resolution fixed, stato chiuso e risolto. Tale lista viene filtrata dai ticket che non hanno un relativo commit (non è stato trovato un commit che ha nel commento l'id del ticket). Se invece il commit esiste viene aggiornata la data di risoluzione del ticket impostandola a quella dell'ultimo commit trovato.

In seguito se il ticket non presenta valori nel campo affected versions vengono calcolate le possibili versioni infette tramite proportion. In particolare si è usato un approccio **coldStart**.

Si passa poi all'analisi release per release.

Progettazione - Costruzione del Dataset

Una volta ottenuti i ticket e settati i vari branch per le release, si passa all'effettiva realizzazione del dataset.

Per motivi di analisi poiché il 50% finale di release vengono scartati, si è scelto per rendere più veloce il codice di eseguire la costruzione del dataset solo sulla prima metà.

L'analisi ciclica delle release avviene seguendo tali passi:

- Estrazione dei ticket relativi alla release
- Settaggio della branch a quella indicata dalla release
- Ottenimento delle classi java (non in src/test) della branch
- Settaggio della bugginess basata sui ticket
- Computazione delle metriche
- Salvataggio delle misurazioni per la release sul file di output

Progettazione - Costruzione del Dataset

Alcune note sul settaggio della bugginess.

Poiché Jira viene gestito “manualmente” dagli sviluppatori di un progetto non è garantito che tutte le informazioni in un ticket siano corrette. La data di risoluzione è stata modificata prendendo in considerazione la data del relativo commit. Mentre per le versioni affette nel caso fossero presenti nativamente in Jira non è stata compiuta nessuna modifica, il proportion viene eseguito solo nel caso in cui non è stato specificato tale campo.

Anche per quanto riguarda i commit vi è una forte componente umana che può far variare il numero di effettive classi buggy. Trovare e analizzare tali casi è un lavoro molto dispendioso, che avrebbe complicato ulteriormente il progetto. È stato quindi scelto di non coprire tutti i casi, ciò ha come impattato un rilevamento inferiore di classy buggy.

Si fanno comunque presente alcuni problemi noti:

- Cambio del nome o del path di una classe: poiché le misurazioni vengono eseguite relese per release non sono state valutate quelle classi che cambiano tali valori all’interno della stessa release. (Vengono considerate solo le classi che corrispondono al path finale nella release)
- Alcuni commit possono comparire in una pull request che risulta non essere merged con il repository
- L’affected version può essere confusa con fix version in Jira

Variabili – Del Dataset

Variabili base:

- Version: release index della classe analizzata (indice incrementale intero parte da 1)
- File name: path della classe analizzata
- Buggy: indica se la classe analizzata è risultata in un ticket oppure no (booleano di valore Yes No)

Metriche di complessità del codice (12):

- Loc: linee di codice
- Loc_touched: somma sulle revisioni delle linee di codice (aggiunte+eliminate+modificate)
- NR: numero di revisioni
- Nfix: numero di bug risolti
- Loc_added: somma sulle revisioni delle linee aggiunte
- Max_Loc_added: massimo delle linee aggiunte
- Avg_Loc_added: media delle linee aggiunte
- Churn: somma sulle revisione di aggiunte – eliminate (LOC)
- Max_churn: massimo churn nelle revisioni
- Avg_churn: media del churn nelle revisioni
- ChgSetSize: numero di file committati insieme al file considerato
- Max_chgset: chgset massimo
- Avg_chgset: chgset medio

Progettazione – Analisi tramite Weka

Una volta completato il dataset è stato possibile effettuare un'analisi predittiva sulla bugginess di una classe, per far ciò è stato utilizzato il tool Weka.

Per poter lavorare su tale dataset il file csv è stato convertito in arff.

- Importante far notare che, nonostante nel codice vi sia la possibilità di selezionare l'indice del valore che deve essere considerato da Weka come positivo, ho preferito modificare l'ordine dei valori dell'attributo Buggy, impostando Yes come primo valore.(Il convertitore imposta il valore più frequente come primo ed è il primo ad essere considerato positivo ai fini dell'analisi)

Una volta ottenuto questo nuovo file è stata eliminata la colonna superflua per l'analisi contenente i nomi delle classi.

Lo scopo del progetto era quello di analizzare e confrontare diverse tecniche, per coprire tutti i vari casi il codice impiega molto tempo, circa un ora di esecuzione.

- Vengono lanciate 72 analisi differenti per release

Progettazione – Analisi tramite Weka

La tecnica di validazione usata è **walk forward**, è stato quindi previsto che prima di iniziare l'analisi della release vengano preparati il giusto training set e il giusto testing set.

- Il test set si sposta ciclicamente da release a release, mentre il training si espande aggiungendo il precedente test set.

Per ogni walk vengono poi eseguiti i seguenti passi:

- Viene scelto un classificatore tra RandomForest, NaiveBayes, Ibk
- Per ogni classificatore si entra in una seconda fase in cui vengono calcolate tutte le possibili combinazioni tra le varie tecniche:
  - Feature Selection: No selection , best first
  - Balancing: No sampling, oversampling, undersampling, SMOTE
  - Sensitive Learning: No cost sensitive, sent+sitive threshold, sensitive learnig con matrice

Completata l'analisi vengono salvati i risultati in un file csv

Variabili – Di Weka

L'output file generato dal codice contiene numerose misurazioni(17 colonne). Di seguito sarà descritto il significato di ognuna di esse.

- Dataset: Nome del dataset (progetto Apache) analizzato.
- #Training release: corrisponde all'indice del walk forward che indica l'attuale release in analisi
- %Training: rappresenta la percentuale di dati nel training set (data on training / total data)

N.B. l'esecuzione del codice è stata effettuata calcolando erroneamente data on training / total data e per motivi di tempo non è stato possible rieffettuare due run complete per i due progetti. Per correggere tale errore è stata aggiunta una colonna in excel che calcola tale valore (100-% testing). Mi sono accorto dell'errore poiché non potevano essere corretti i valori tra undersampling e oversmpling. Tale errore non impatta sul resto dei risultati.

- %Defective in training: percentuali di difetti nel training set
- %Defective in testing: percentuali di difetti nel testing set

Variabili – Di Weka

- Classificatore
- Balancing
- Feature selection
- Sensitivity

Attuale tecnica analizzata, tali variabili possono assumere valori nell'insieme descritto in precedenza

- TP: Numero delle istanze stimate buggy in modo corretto
- FP: Numero delle istanze stimate buggy in modo scorretto
- TN: Numero delle istanze stimate non buggy in modo corretto
- FN: Numero delle istanze stimate non buggy in modo scorretto

Variabili – Di Weka

Performance metrics:

- Precision: Indica il numero di positivi correttamente stimati
  - TP/(TP+FN)
- Recall: Indica quante volte è stata correttamente classificata un istanza come positiva
  - TP/(TP+FP)
- Auc (Area under the ROC): Rappresenta la probabilità che il classificatore classifichi un'istanza positiva scelta a caso al di sopra di una scelta a caso tra le negative.
- Kappa: Indica quante volte si è stati più accurati rispetto a un classificatore dummy

Risultati - Introduzione

I risultati ottenuti sono stati analizzati tramite il software JMP.

Sono stati costruiti quindi numerosi box plot e tabelle per studiare il comportamento dei 3 classificatori, per comprendere quali tecniche di feature selection, balancing o sensitivity riescono a migliorare l'accuratezza del classificatore.

Nello specifico i grafici riporteranno:

- Sull'asse y le metriche di precision, recall, AUC, Kappa
- Tramite sovrapposizione vengono inseriti i classificatori
- L'asse x e i raggruppamenti sono invece usati per ottenere i risultati sulle tecniche

Per brevità non saranno riportati interamente tutti i grafici intermedi utilizzati per effettuare l'analisi finale, ma semplicemente il primo grafico prodotto che confronta i classificatori senza raggruppamenti, e quello che invece produce tutti i box plot applicando le varie tecniche

Risultati - BookKeeper

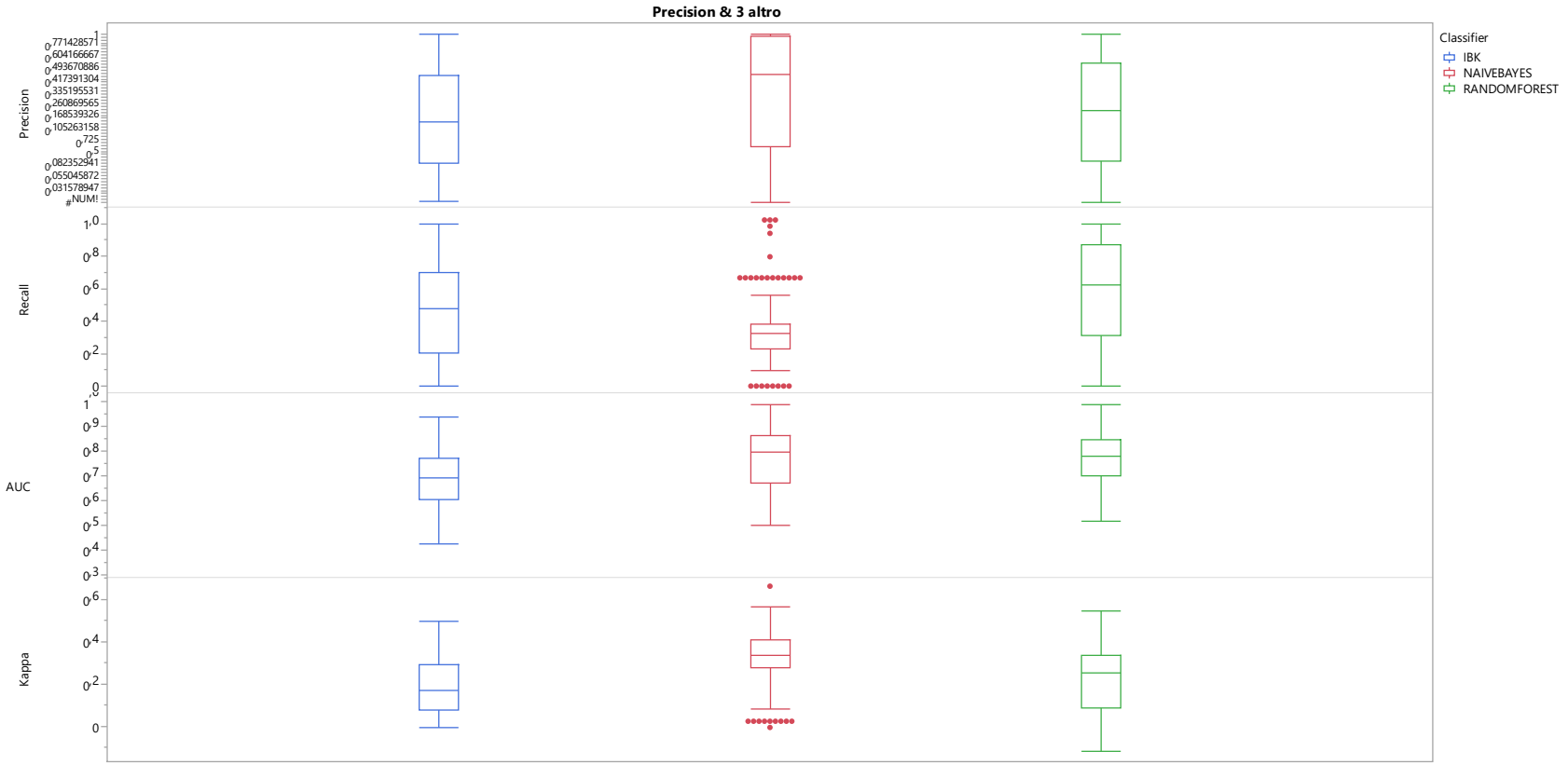
<u>Primo grafico di confronto tra i tre classificatori</u>

Tale grafico riporta sull'asse y le 4 metriche di accuratezza considerate

In blu vi sono i risultati per Ibk

In rosso vi sono i risultati per NaiveBayes

In verde vi sono i risultati per RandomForest



Risultati - BookKeeper

Grafico che raggruppa i vari risultati in base alla tecnica

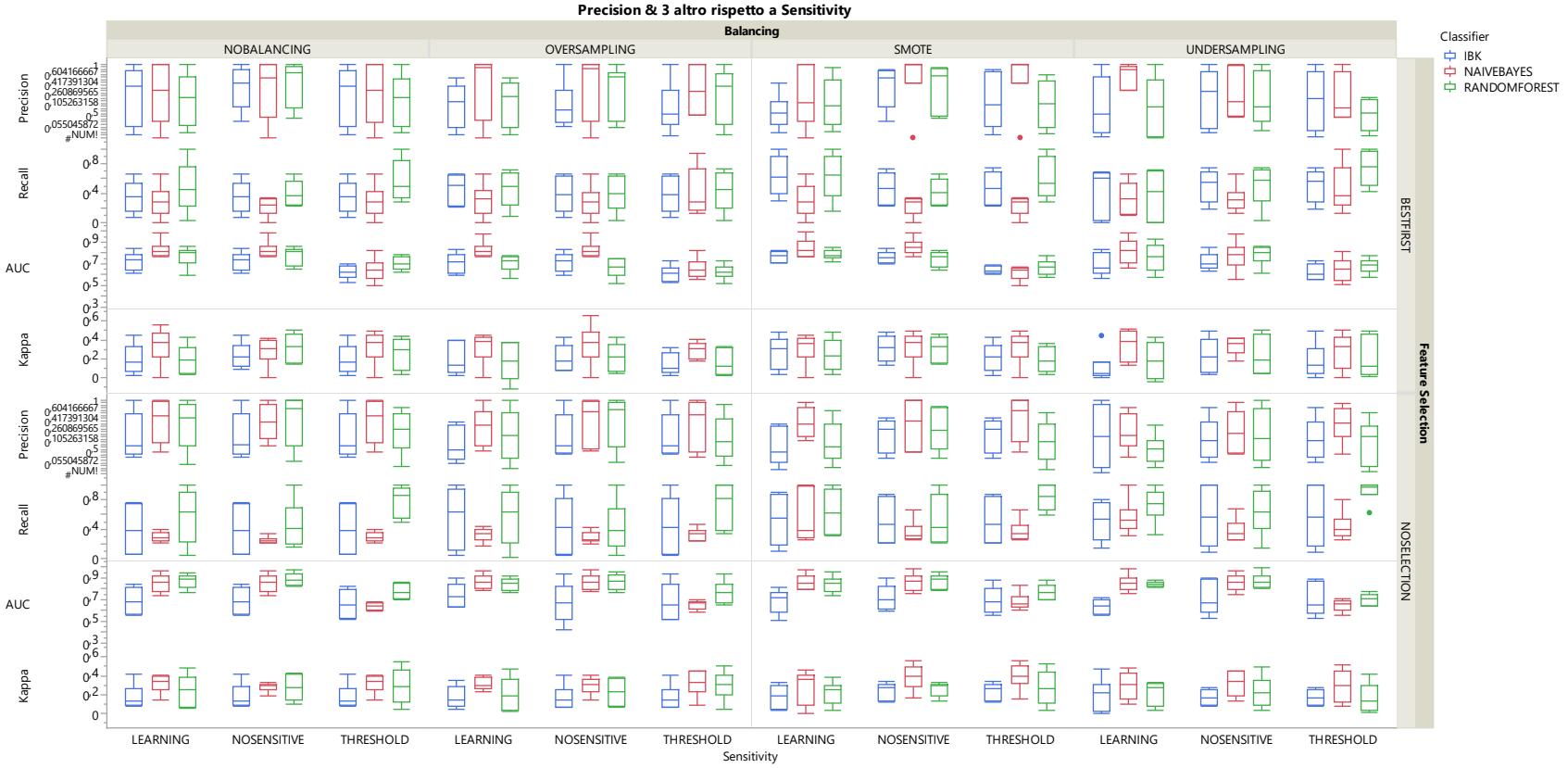
A fine presentazione è presente un grafico con un'altra vista degli stessi risultati (Final 2)

Tale grafico riporta sull'asse y le 4 metriche di accuratezza considerate

In blu vi sono i risultati per Ibk

In rosso vi sono i risultati per NaiveBayes

In verde vi sono i risultati per RandomForest



Risultati - ZooKeeper

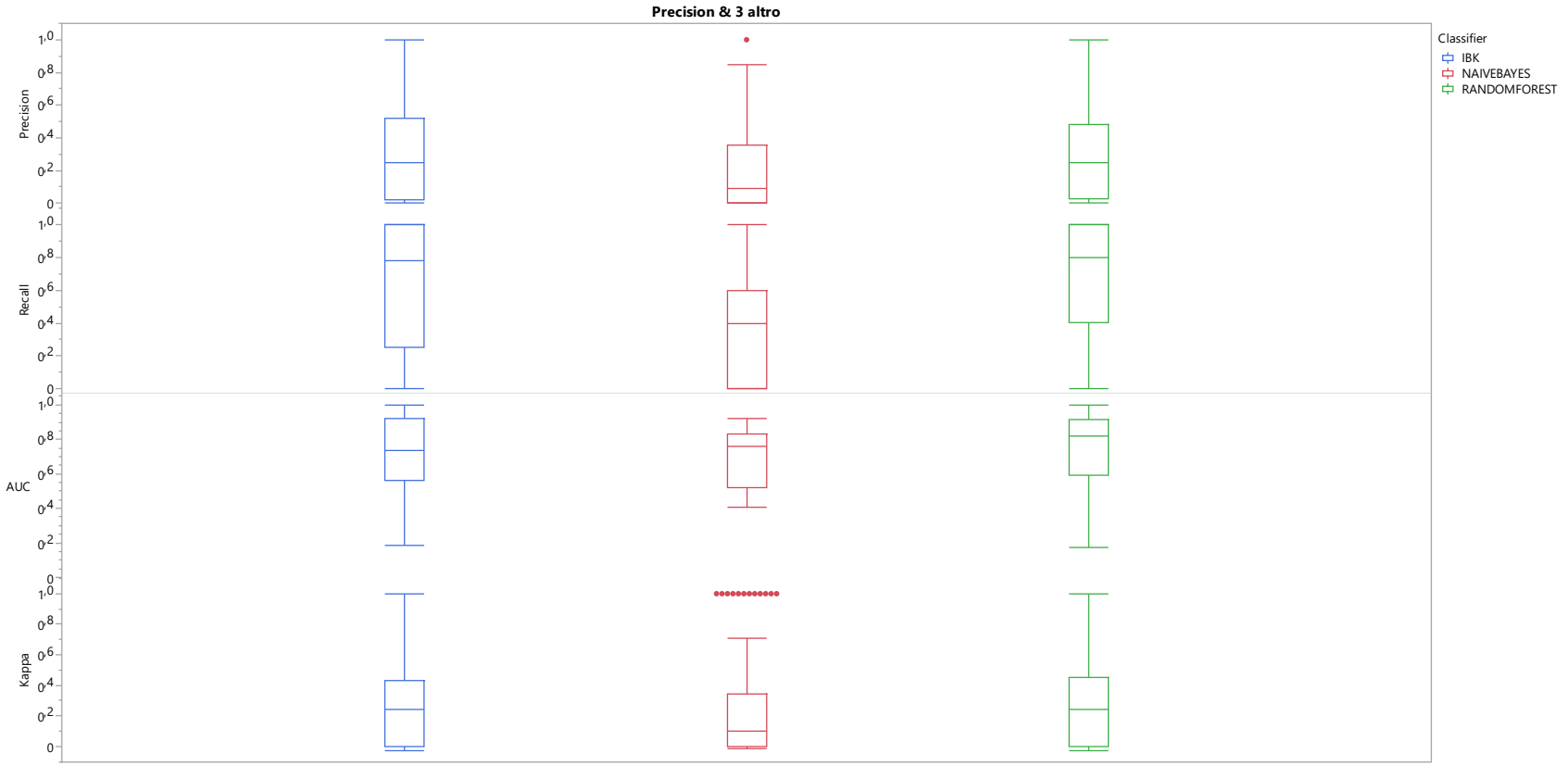
<u>Primo grafico di confronto tra i tre classificatori</u>

Tale grafico riporta sull'asse y le 4 metriche di accuratezza considerate

In blu vi sono i risultati per Ibk

In rosso vi sono i risultati per NaiveBayes

In verde vi sono i risultati per RandomForest



Risultati - ZooKeeper

Grafico che raggruppa i vari risultati in base alla tecnica

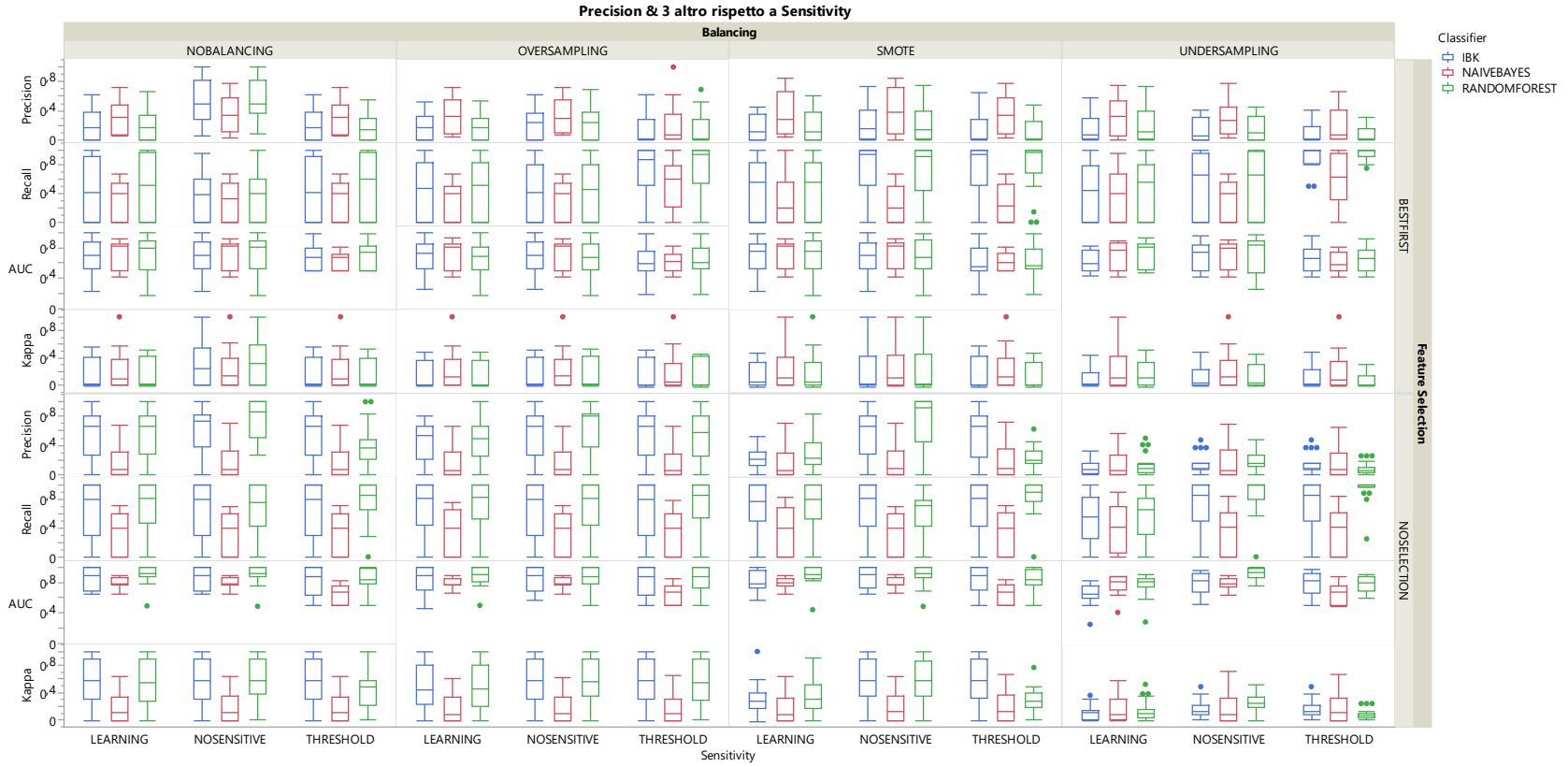
A fine presentazione è presente un grafico con un'altra vista degli stessi risultati (Final 2)

Tale grafico riporta sull'asse y le 4 metriche di accuratezza considerate

In blu vi sono i risultati per Ibk

In rosso vi sono i risultati per NaiveBayes

In verde vi sono i risultati per RandomForest



Risultati

Sono qui riportate le tabelle riguardante il conteggio di TP, FP, TN, FN, per i due progetti

Si mette in evidenza come l'utilizzo di un cost sentive classifier porta ad una variazione del valori misurati, in particolare avendo dato un maggior peso al CFN il valore di FN si abbassa tranne nel caso indicato dalla freccia

Disponi in tabella

		Dataset								
		BOOKKEEPER								
		Classifier								
		IBK			NAIVEBAYES			RANDOMFOREST		
		Sensitivity			Sensitivity			Sensitivity		
		LEARNING	NOSENSITIVE	THRESHOLD	LEARNING	NOSENSITIVE	THRESHOLD	LEARNING	NOSENSITIVE	THRESHOLD
TP	Somma	690	637	640	596	500	645	779	664	1077
FP	Somma	2156	1263	1803	583	183	683	2228	1203	3465
TN	Somma	10788	11681	11141	12361	12761	12261	10716	11741	9479
FN	Somma	990	1043	1040	1084	1180	1035	901	1016	603

Disponi in tabella

		Dataset								
		ZOOKEEPER								
		Classifier								
		IBK			NAIVEBAYES			RANDOMFOREST		
		Sensitivity			Sensitivity			Sensitivity		
		LEARNING	NOSENSITIVE	THRESHOLD	LEARNING	NOSENSITIVE	THRESHOLD	LEARNING	NOSENSITIVE	THRESHOLD
TP	Somma	919	930	1004	772	697	785	993	974	1210
FP	Somma	5046	5425	11278	2257	1702	5252	4410	5005	14109
TN	Somma	44194	43815	37962	46983	47538	43988	44830	44235	35131
FN	Somma	617	606	532	764	839	751	543	562	326

Discussione dei risultati

In generale l'analisi non è risultata un compito facile, vi sono molti fattori che influenzano i risultati ottenuti e molti punti di vista da poter discutere. Ricercando regole generali di ottimalità, soprattutto nel confronto tra i due progetti, si è constatato che il comportamento dei classificatori è altamente influenzato dalla natura del dataset di partenza.

Il dataset di BookKeeper presenta un numero inferiore di dati (solo 7 release) e un numero superiore di classi buggy ciò porta ad avere, a prescindere dal classificatore, dei valori più alti nelle metriche di accuratezza.

Mentre il dataset di ZooKeeper rispecchia la natura stabile del progetto, quindi un elevato numero di release (22 in analisi) e conseguente maggior quantitativo di dati, inoltre essendo stabile il numero di classi buggy è abbastanza basso in confronto a BookKeeper che è invece un progetto abbastanza più recente e meno stabile.

Discussione dei risultati

Per entrambi i progetti l'analisi della prima release (release 1 training set e release 2 testing set) ha prodotto dei valori abbastanza particolari. Per ZooKeeper sono stati misurati valori molto alti di recall (spesso pari a uno) e bassi di precision, situazione contraria per BookKeeper.
Proseguendo con il walk forward i valori si sono stabilizzati.

Si evidenzia che la release 9 di ZooKeeper presenta una percentuale di defective nel traning set pari a zero e i calcoli su di essa sono stati poco rilevanti.

Tramite JMP sono stati evidenziati altri valori che erano outlier delle varie distribuzioni evidenziate nei grafici tramite dei punti isolati.

Per BookKeeper confrontando i 3 classificatori si nota che NaiveBayes presenta i valori più alti di precision, AUC e kappa mentre più basso per la recall. Partendo da questa osservazione mi sono chiesto da cosa dipendesse questo risultato e analizzando i valori di TP,FP,TN,FN ho scoperto che tale classificatore, su entrambi i progetti, è stato quello con il maggior numero di FN e minor numero di FP. Mi sarei quindi aspettato un comportamento simile in ZooKeeper che invece si comporta quasi in modo opposto presentando i valori più bassi tra i 3 classificatori. Questo valore dimostra che la natura del dataset ha molto peso sul comportamento del classificatore, tale dataset ha infatti un discostamento maggiore tra classe minoritaria e maggioritaria rispetto a BookKeeper.

Si deduce quindi che tra i tre classificatori studiati NaiveBayes è quello che tende a classificare maggiormente le istanze come non buggy (negative)

Discussione dei risultati

Per poter rispondere alla domanda quali tecniche di feature selection, balancing o sensitivity aumentano l'accuratezza dei classificatori, ho preferito compiere un'analisi separata per i due progetti.

BookKeeper:

- Per tutti e tre i classificatori viene evidenziato un comportamento abbastanza simile se è stata effettuata o meno feature selection. In questi casi poiché feature selection riduce i costi computazionali, risulta essere conveniente introdurla.
- Per quanto riguarda il balancing invece i risultati sono abbastanza differenti e dipendono dal classificatore:
  - Ibk in generale aver introdotto del balancing migliora le metriche. SMOTE è la tecnica che riesce ad ottenere risultati migliori
  - NaiveBayes solo nel caso dello SMOTE riesce ad ottenere miglioramenti
  - RandomForest invece non presenta miglioramenti con l'introduzione del balancing
- In generale usando un cost sensitive classifier non si è registato un netto miglioramento, poiché in base alla tecnica utilizzata si riescono ad alzare dei valori ma si abbassano degli altri per i 3 classificatori:
  - Learnig migliora la recall ma abbassa la precision, lasciando invariate le altre metriche
  - Threshold migliora leggermente la recall ma abbassa drasticamente l'AUC

Discussione dei risultati

ZooKeeper:

- Per quanto riguarda la feature selection, con questo dataset, si evidenzia un netto peggioramento delle metriche di accuratezza nel momento in cui è stato utilizzato best first, a prescindere dal classificatore. Conviene quindi non applicare tale tecnica.
- Nel balancing invece si nota che utilizzare l'undersampling peggiora di molto le metriche su tutti i classificatori. Negli altri casi introdurre il balancing non ha comunque portato a un netto vantaggio:
  - In generale la precision si abbassa nei 3 classificatori rispetto a no balancing.
  - Per Ibk e RandomForest introdurre oversampling o SMOTE permette di ridurre la dispersione per quanto riguarda recall e kappa, ma ne fa comunque abbassare leggermente il valore medio
  - NaiveBase parte da volori bassi con no balancing e introducendo tale tecnica rimane pressoché stabile
- Anche per la sensitive si può escludere una delle tecniche nello specifico utilizzare il learning comporta o un abbassamento o una stabilità delle metriche considerate. Mentre il threshold non riesce a ottenere un completo miglioramento su tutte le metriche.
  - Similmente al balancing anche per la sensitive i valori di NaiveBayes non vengono migliorati.
  - In Ibk con sensitive threshold si riesce a migliorare la recall di molto ma gli altri valori si abbassano
  - Con RandomForest invece si migliora molto la recall ma la precision si abbassa drasticamente, anche Auc e kappa peggiorano

Conclusioni

Dalle analisi effettuate si è constatato che non vi è una scelta univoca su quale classificatore usare e quali tecniche applicare per migliorare le metriche di adeguatezza.

Per poter compiere tale scelta bisogna contestualizzare lo studio nell'ambito di progetto e compiere le opportune scelte sui possibili compromessi da dover prendere. Bisogna quindi trovare il giusto trade off tra l'accuratezza cercata, quali sono quindi i valori "accettabili" per una metrica e quali si discostano troppo, inoltre c'è anche da valutare il fattore tempo che può essere influenzato da una scelta piuttosto che un'altra.

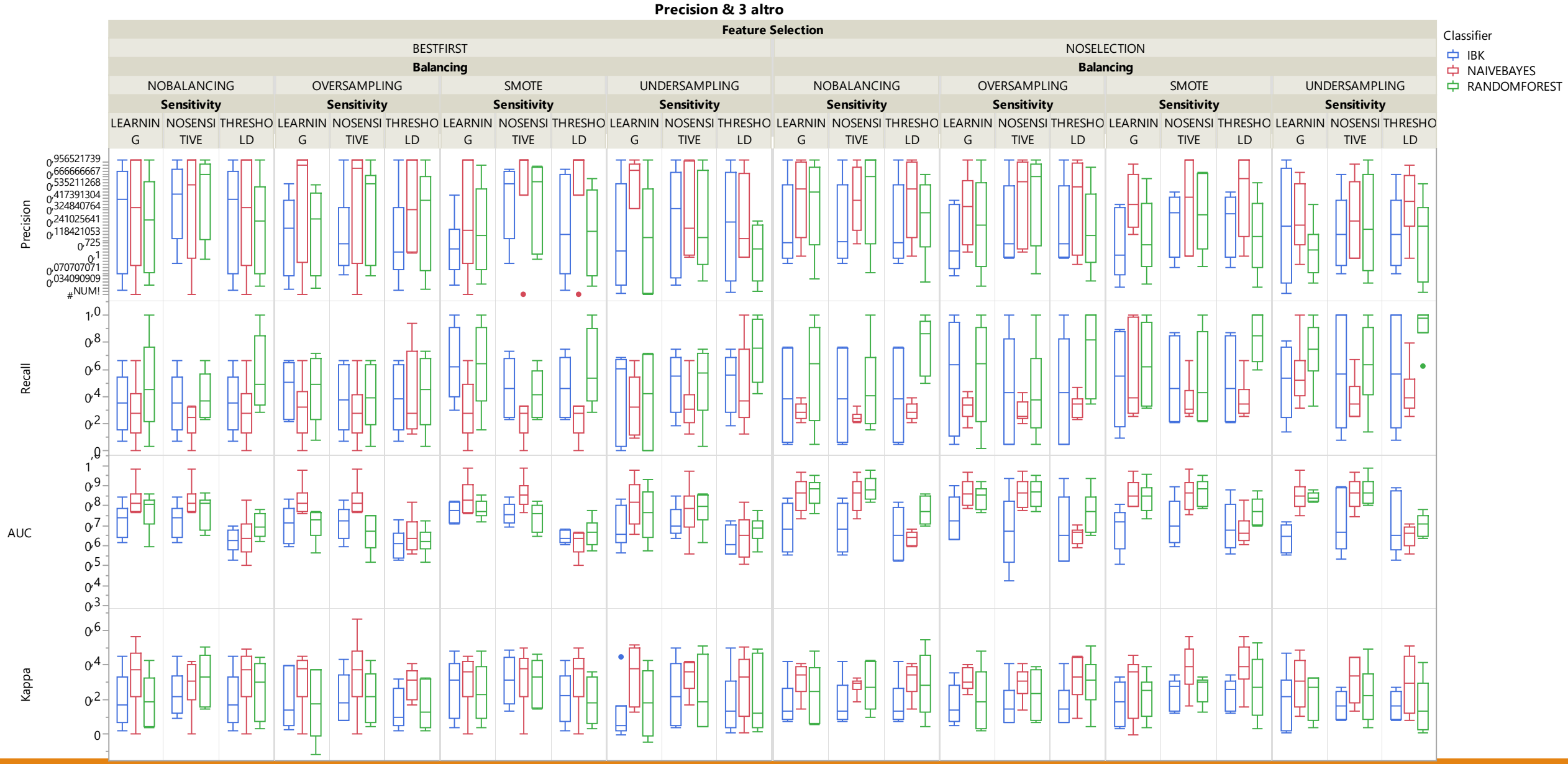
Volendo comunque scegliere un caso migliore per i singoli progetti

- BookKeeper mostra i risultati migliori utilizzando Ibk con best first, Smote, no cost sensitive.
- ZooKeeper mostra i risultati migliori utilizzando RandomForest con no selection, no balancing, no cost sensitive.

Link

- GitHub: https://github.com/IlConteCvma/Deliverable2
- Travis : https://travis-ci.com/github/IlConteCvma/Deliverable2
- SonarCloud: https://sonarcloud.io/dashboard?id=IlConteCvma_Deliverable2

Grafici con viste differenti BookKeeper



Grafici con viste differenti ZooKeeper