Predizione di bug nel codice di grandi progetti software Open-Source: uno studio sui classificatori

Caporaso Pasquale (0286761), Ingegneria del software e progettazione Web 2

1 Deliverable 1

1.1 Introduzione

La misurazione di un processo di sviluppo è il primo passo verso il suo miglioramento. L'obiettivo di questa prima deliverable è, quindi, quello di cercare di misurare la stabilità di un processo di sviluppo, quello del progetto STDCXX, e cercare di trovare in esso dei periodi di tempo anomali per studiarli e, se necessario, trovare misure per evitare che si ripresentino in futuro.

Lo strumento che viene utilizzato per tale scopo è il Process Control Chart. Nelle prossime sezioni verrà descritto il processo che ha portato alla raccolta dei dati necessari per questo grafico e le informazioni che sono state dedotte da essi.

1.2 Progettazione

Lo svolgimento di questa deliverable si è diviso in due grandi fasi:

- Recupero delle chiavi dei ticket del progetto da Jira.
- Analisi dei commit sulla repository.

Il recupero dei ticket da Jira è stato relativamente semplice, ho utilizzato le REST API messe a disposizione da Apache limitando le risposte a singolo campo "Key" per velocizzare il processo. Ho dovuto comunque fare diverse richiste alla piattaforma in quanto il numero massimo di ticket recuperabili con una singola query è 1000.

Dopo aver fatto ciò ho esaminato i commit della repository di STDCXX, cercando quelli con le chiavi recuperate, contandoli in modo da avere alla fine il numero di commit di fix, e quindi di ticket risolti, in ogni mese per il tempo di vita del progetto. Ho preferito, in questa fase, scaricare la repository del progetto sulla mia macchina e chiamare da Java direttamente il comando "git", in questo modo ho evitato le limitazioni delle REST API di Git e, non dovendo utilizzare la rete, ho velocizzato il completamento delle operazioni.

1.3 Risultati

La prima cosa che ho notato durante lo svolgimeto del progetto è che molti commit non avevano al loro interno nessuna chiave e quindi è stato impossibile associarli al relativo ticket.

Il numero di commit che ho dovuto ignorare è superiore alla metà del numero totale di commit come evidenziato in Figure 1

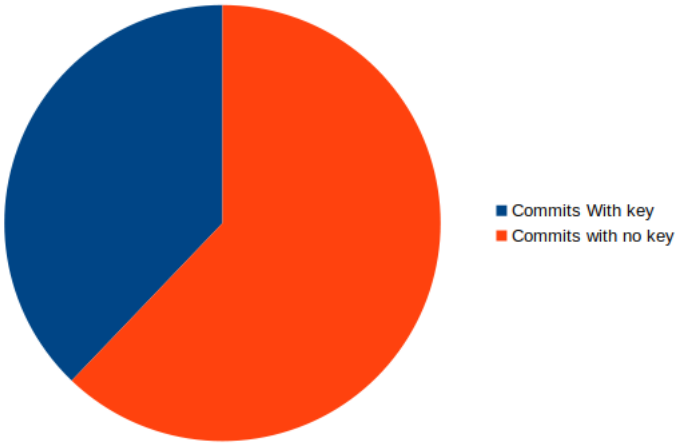

Figure 1: .

Dopo aver eliminato i commit che non posso classificare il process control chart è quello in Figure 2

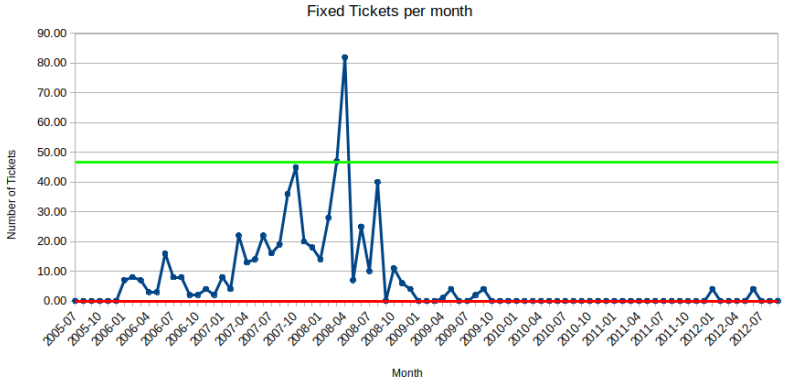

Figure 2: .

Si può notare che la deviazione standard è molto alta e che quindi il limite inferiore è 0, questo può essere dovuto al periodo ”morto” che si nota nella seconda parte del grafico. Tale periodo ha, infatti, pochi commit anche se includo quelli senza chiave, questo è evidenziato in Figure 3.

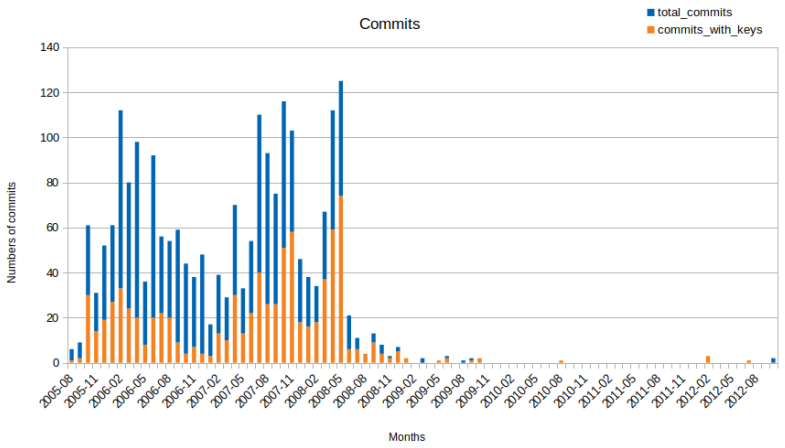

Figure 3: .

Avendo notato questo ho provato ad eliminare i commit relativi a tutti i mesi più recenti con meno di

4 commit, il risultato, evidenziato in Figure 4, rimane però simile al precedente, anche in questo caso la deviazione standard è molto alta e i due limiti sono molto distanti

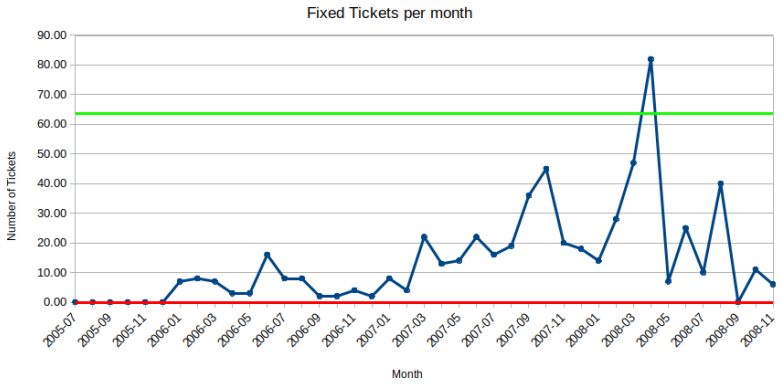

Figure 4: .

1.4 Discussione

Il Process Control Chart mostra che questo processo è rimasto all'interno dei limiti stabiliti per la quasi totalità del periodo esaminato e presenta anche un mese in cui la produzione è superiore alle aspettative.

Tale periodo però è probabilmente dovuto al maggior numero di commit avvenuti in quel particolare mese ed è, quindi, non molto significativo per quanto riguarda la stabilità del processo.

Tuttavia l'alta variabilità dei risultati ci impedisce di analizzare adeguatamente il limite inferiore ed è quindi impossibile determinare se questo progetto ha avuto o meno dei periodi in cui il processo era significativamente rallentato.

Inoltre, la presenza di un alto numero di commit senza chiave potrebbe aver influenzato significativamente i risultati ottenuti. Per questo motivo i dati riportati potrebbero essere inaffidabili.

Quindi, per le motivazioni sopra riportate, risulta impossibile fare considerazioni definitive su questo progetto.

1.5 Links

Github: www.github.com/Capo80/Deliverable_1
SonarCloud: www.sonarcloud.io/dashboard?id=Capo80_Deliverable_1

2 Deliverable 2

2.1 Introduzione

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

-Brian W. Kernighan

La fase di debugging è una delle più difficoltose nel processo di sviluppo, in particolare, la ricerca dei bug all'interno del codice è un lavoro complesso e, in generale, può essere effettuato solo dopo che il bug ha causato danni.

L'obiettivo di questo studio è quello di usare varie tecniche di machine learning per cercare di identificare bug prima che essi si manifestino. Fatto ciò procederò a misurarne la loro efficacia utilizzando delle versioni in cui si conoscono già le classi Buggy.

In particolare analizzerò due progetti open-source, BOOKKEEPER e ZOOKEEPER, cercherò di capire qual è il miglior classificatore da usare e se utilizzare tecniche di Balancing e Feature Selection migliora la loro efficacia.

2.2 Progettazione

Lo svolgimento di questa deliverable si è articolato in diverse fasi:

- Recupero della lista dei file per ogni versione da git
- Identificazione di classi Buggy da git e jira
- Analisi di ogni file per ricavare delle metriche con git
- Utilizzo di Weka java API per applicare le techine di Machine Learning

Per tutte le fasi sopra menzionate, per utilizzare git, ho preferito scaricare la Repository del progetto in locale ed interagire con git chiamando direttamente comandi da Java. Questo mi ha permesso di evitare le limitazioni delle REST API di git e, evitando di usare la rete, ho velocizzato l'esecuzione del programma.

Recupero dei file La prima cosa che ho fatto per questa deliverable è stato cercare un modo per ottenere i file presenti in ogni versione. Per fare ciò ho prima ricavato le date di pubblicazione di ogni versione utilizzando il codice del professore con qualche piccola modifica. Dopo aver fatto ciò ho utilizzato tali date per ottenere, nel periodo di tempo della versione, tutti i file toccati da commit. Questo mi ha permesso di costruire una lista di file divisa per versione.

Ho deciso di mantenere nel dataset anche i file che alla fine della versione risultavano avere 0 righe, questo perchè ritengo che anche i file "di lavoro" usati durante la versione possono avere un influenza sulle classi Buggy

Identificazione di classi Buggy Per identificare le classi buggy ho utilizzato i ticket etichettati come **Bug** prenseti su Jira, la prima cosa che ho fatto è cercare di ottenere per ognuno di essi:

- la chiave, l'id univoco del ticket, che poi ho utilizzato per trovare i commit
- la Fixed Version, la versione in cui il bug è stato risolto
- la Opening Version, la versione in cui il bug è stato trovato (quella corrispondente alla data di creazione del ticket)
- le Affected Version, dove presenti, sono tutte le versioni in cui il bug era presente nel sistema

Dopo aver ricavato tali informazioni ho dovuto scartare un po' di dati in quanto alcuni ticket non avevano fixed version, in questo caso ho dovuto scartare tutto il ticket, e altri avevano delle Affected Version che cadevano dopo la Fixed Version che non è possibile, per questi ho deciso semplicemente di buttare le Affected Version.

Un'altra correzione che ho fatto è stata quella di controllare tutti i commit per ogni ticket e, se trovavo commit in date successive alla Fixed Version di Jira la sostituivo con quella del commit.

Avendo ricavato queste informazioni da Jira ho usato Proportion (con il metodo Increment) per ricavare tutte le Afffected Version mancanti. *In questa fase ho deciso di mettere come valore di P uno se la IV, OV e FV combaciavano e come valore FV-IV se la FV e la OV combaciavano. Per le altre ho usato la formula stardard di proportion*

Infine, adesso che ho tutte le Affected Version per ogni ticket, ho recuperato da git, usando le chiavi, tutti i commit per ogni Defect e ho impostato come Buggy, in ogni AV, tutti i file che tale commit modifica. I file non toccati, di conseguenza, sono quelli Non Buggy.

Analisi delle metriche Per ottenere informazioni riguanti i file ho utilizzato le seguenti metriche:

Metric (File C)	Description
Size	LOC
LOC touched	Sum over revisions of LOC added+deleted+modified
NR	Number of revisions
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

Per quanto riguarda il calcolo delle LOC, ho utilizzato il numero di righe aggiunte, modificate e rimosse per ogni file da ogni commit diviso per revisione, ho ricavato questa informazione da git usando il parametro "–num-stat", con questa modalità è stato semplice anche calcolare tutte le altre metriche collegate alla LOC e quelle delle CHURN, in quanto già esaminavo le righe aggiunte/modificate/eliminate da ogni revision. Per quanto riguarda il numero di autori ho semplicemente tenuto il conto mentre scorrevo i commit, così come per il numero di revisions. Per il ChgSetSize, ho, ancora una volta, analizzato ogni singolo commit diviso per versione, per ognuno di essi ho ricavato la lista di file nel commit e ho aggiornato le misurazioni della metrica per ognuno di essi.

Per il calcolo di queste metriche ho deciso di proseguire in maniera incrementale, ovvero considerando anche i valori delle versioni precedenti, questo perchè credo che sia necessario guardare l'intera storia di un file per ottenere le corrette informazioni

Training e Valutazione del modello In questa fase ho preso il dataset ricavato dalle informazioni raccolte e, tramite l'apporccio Walk Forward, l'ho utilizzato per allenare e valutare i 3 classificatori: IBk, NaiveBayes, RandomForest. Inoltre per ognuno di essi ho sperimentato ogni possibile combinazione delle tecniche di Balancing (UnderSampling, OverSampling, SMOTE) e la tecnica di Feature Selection (Best First).

2.3 Risultati

Iniziamo analizzando i risultati ottenuti misurando il progetto comune, BOOKKEEPER.

Innanzitutto andiamo a considerare le varie tecniche di Balancing e Feature Selection, dal BoxGraph in Figure 5 sembra difficile fare osservazioni definitive per via dell'alta varibilità dei valori ottenuti. Questo è probabilmente dovuto alle piccole dimensioni del progetto. Tuttavia sembra che, almeno per IBk e Random Forest, nessuna delle tecniche porti ad un sostanziale miglioramento delle metriche.

Per quanto riguarda NaiveBayes, sembra esserci un leggero peggioramento della recall e un leggero miglioramento delle altre metriche quando usiamo feature selection.

Ora guardiamo come si sono comportati i singoli classificatori all'aumentare delle versioni senza applicare tecniche di Balancing o Feature Selection.

Notiamo in Figure 6 che i classificatori IBK e Random Forest hanno una perfomance simile in praticamente tutte le metriche, mentre Naive Bayes è significativamente peggiore quando misuriamo la precision e la kappa.

Un'altra osservazione che possiamo fare è che tutti i classificatori hanno avuto un calo di performance nella versione 7. Questo è probabilemente dovuto al fatto che in tale versione c'è un picco di classi Buggy, l'83% in confronto al circa 60% delle versioni precedenti, probabilmente in questa versione il team di sviluppo di BOOKKEPPER ha lavorato a una funzionalità particolarmente complessa che ha portato ad un aumento del numero di Bug, ciò ha causato una alterazione significativa del livello di Buggyness che ha tratto in inganno i nostri classificatori.

Per confermare o smentire questa ipotesi sarebbe però necessaria un'analisi più approfindita del progetto e del processo di sviluppo.

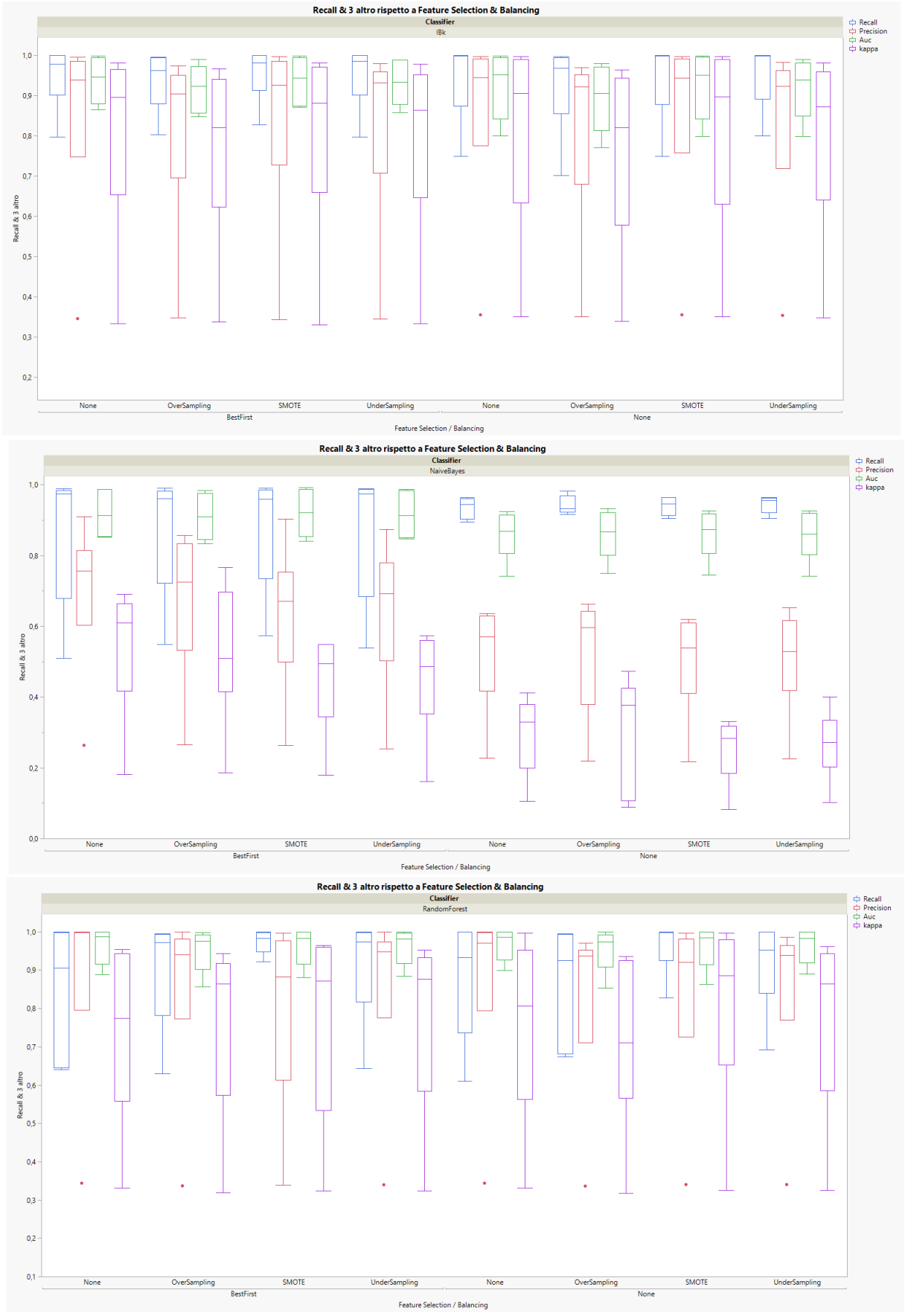
Recall & 3 altro rispetto a Feature Selection & Balancing
Classifier
IBk
Recall
Precision
Auc
kappa
Recall & 3 altro
1,0
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
None
OverSampling
SMOTE
UnderSampling
None
OverSampling
SMOTE
UnderSampling
BestFirst
None
Feature Selection / Balancing
Recall & 3 altro rispetto a Feature Selection & Balancing
Classifier
NaiveBayes
Recall
Precision
Auc
kappa
Recall & 3 altro
1,0
0,8
0,6
0,4
0,2
0,0
None
OverSampling
SMOTE
UnderSampling
None
OverSampling
SMOTE
UnderSampling
BestFirst
None
Feature Selection / Balancing
Recall & 3 altro rispetto a Feature Selection & Balancing
Classifier
RandomForest
Recall
Precision
Auc
kappa
Recall & 3 altro
1,0
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
None
OverSampling
SMOTE
UnderSampling
None
OverSampling
SMOTE
UnderSampling
BestFirst
None
Feature Selection / Balancing

Figure 5: .

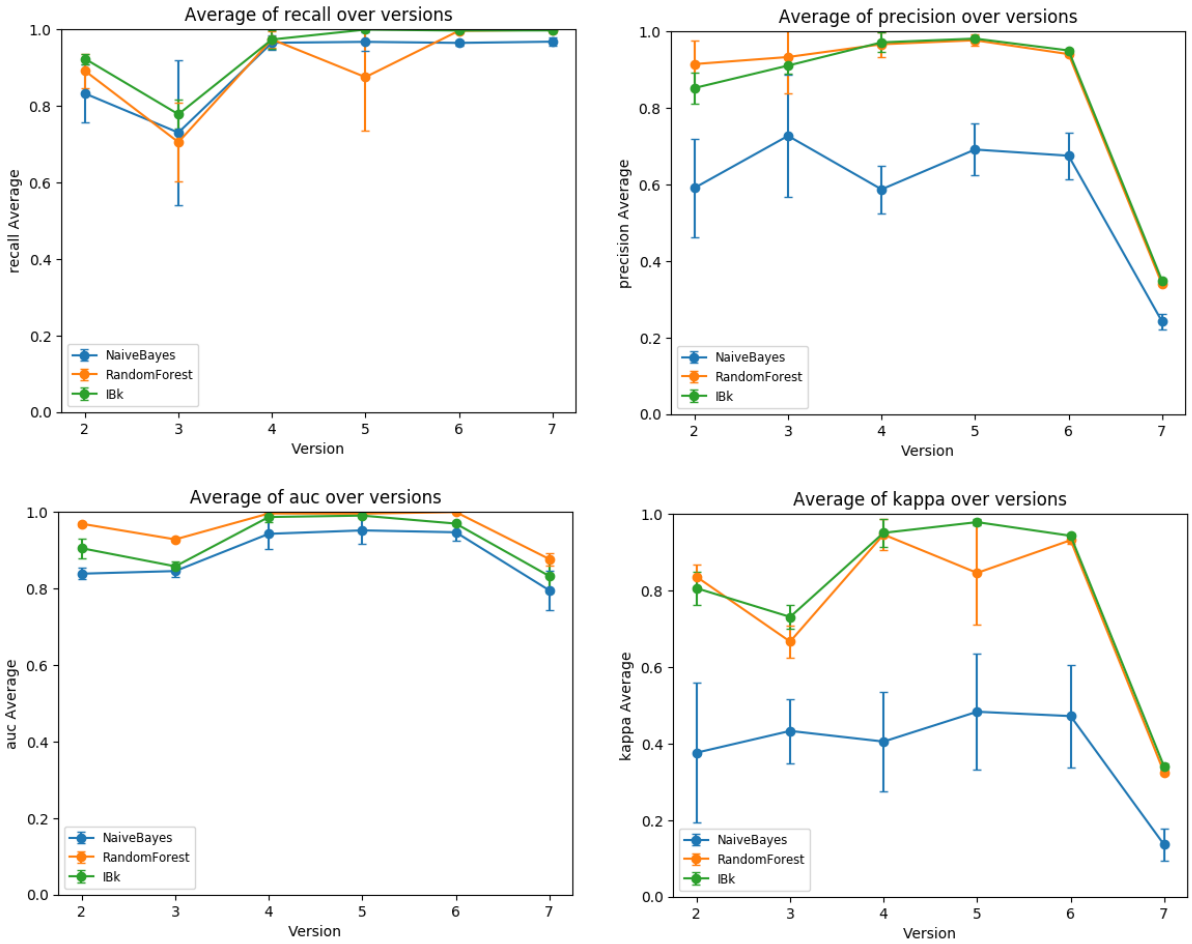

Figure 6: .

Passiamo, adesso, all'analisi di ZOOKEEPER, mi aspetto per questo progetto dei risultati più attendibili, visto il numero maggiore di dati raccolti.

Iniziamo guardando Balancing e Feature Selection, evidenziati in Figure 7. Per ZOOKEEPER, a differenza dell'altro processo, sono molto più evidenti le differenze di performance associate alle diverse tecniche, questo è probabilmente dovuto alla maggiore quantità di dati disponibili.

Quello che notiamo è che l'utilizzo di Feature Selection peggiora significativamente la performance di tutti i nostri classificatori. Questo ci fa pensare che non abbiamo informazioni ridondanti nei nostri dati e che ogni attributo è necessario per la corretta costruzione del nostro modello.

Per quanto riguarda il Balancing la situazione sembra più complessa, per NaiveBayes sembra che le tecniche usate non portino a nessuna differenza significativa, in Random Forest sembra invece che il loro utilizzo abbia portato ad un peggioramento delle metriche, in paritcolare la kappa e la recall. Infine per IBk sembra che UnderSampling e Over Sampling abbiamo una performance peggiore, mentre SMOTE instroduce un leggero miglioramento. Questi dati sembrano suggerire che il nostro modello è abbastanza bilanciato è che non è necessario utilizzare tecniche di balancing.

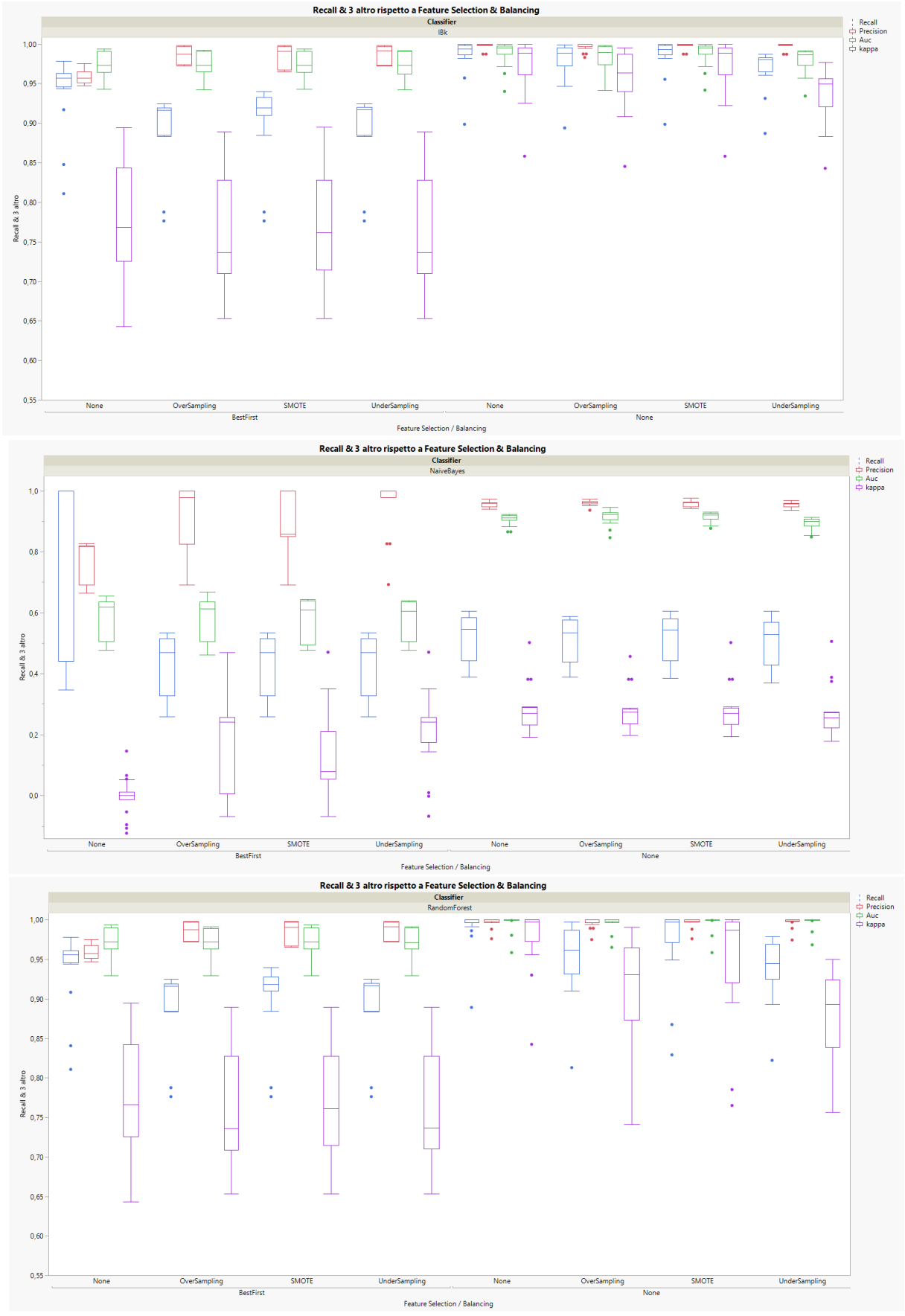

Figure 7: .

Infine vediamo come si comportano i classificatori nelle varie versioni di questo progetto. Quello che

notiamo in Figure 8 è che, anche in questo caso, se non usiamo tecniche di Balancing o Feature Selection, NaiveBayes ha una performance decisamente peggiore degli altri due classificatori, che invece risultano molto simili.

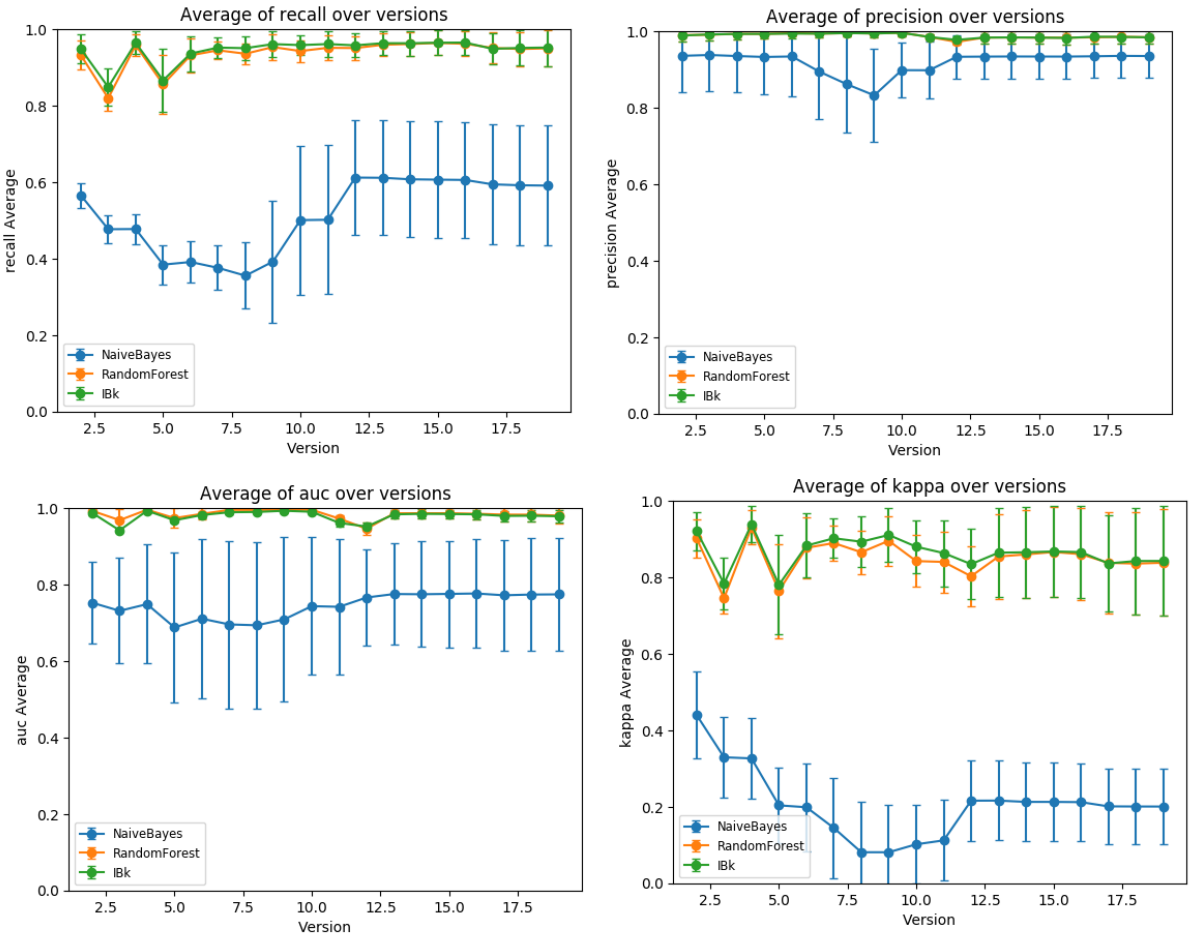

Figure 8: .

2.4 Discussione

I dati ottenuti da questo studio sembrano suggerire che IBk e Random Forest sono classificatori migliori di NaiveBayes sia per ZOOKEEPER, sia per BOOKKEEPER e che la loro accuratezza rimane pressocchè stabile con l'aumentare delle versioni, fatta eccezione dell'ultima versione di BOOKKEEPER.

Per quanto riguarda le tecniche di Feature Selection e Balancing, risulta impossibile per BOOKKEEPER fare considerazioni definitive vista la quantità ridotta di dati. Per ZOOKEEPER invece, esse sembrano avere tutte un comportamento simile tra loro e comunque una performance inferiore all'analisi fatta senza di esse, le ragioni di questo comportamento sono state già discusse nella sezione precendete.

In conclusione ritengo che il miglior classificatore per ZOOKEEPER sia RandomForest senza alcuna tecnica di Feature Selection o Balancing, mentre per BOOKKEEPER sceglierei IBk, sempre evitando le tecniche di FS e Balancing, eliminando, inoltre, la versione 7 dal nostro testing in quanto sembra avere delle caratteristiche diverse dalle altre.

2.5 Links

Github: www.github.com/Capo80/Deliverable_2

SonarCloud: www.sonarcloud.io/dashboard?id=Capo80_Deliverable_2