Report ISW2- Modulo ML for Software Engineering

Valentina Falaschi-0295947

Sommario

Deliverable 1

Introduzione

L'obiettivo di questa deliverable è quello di misurare la stabilità di un attributo di un progetto al fine di poter avere maggiori informazioni sul software. In generale l'analisi del software è importante poiché permette di raggiungere un maggiore livello di conoscenza di quest'ultimo e di conseguenza di avere un miglioramento della qualità, ottimizzare il processo di produzione e predire futuri possibili difetti.
Nel caso in analisi si misura l'attributo relativo al numero di fixed ticktes, per il progetto open source di Apache: Falcon. I dati risultati dall'analisi vengono graficati tramite un process control chart realizzato con l'ausilio di Excel.

Progettazione

Il progetto si articola nei seguenti punti:

- Recupero dei tickets con status *resolved* o *closed* e resolution *fixed*, attraverso l'uso di una API REST, fornita dalla piattaforma Jira;
- Uso di comandi git per trovare la data relativa all'ultimo commit di ciascun ticket;
- Creare una lista contenente tutte le date riscontrate al punto precedente ed associare a ciascuna data un contatore, relativo al numero di volte che viene trovata una specifica data;
- Recupero delle date restanti, relative ai commit del progetto, che non sono state riscontrate nei punti precedenti, per non alterare i dati statistici;
- Costruzione del file csv.

Risultati

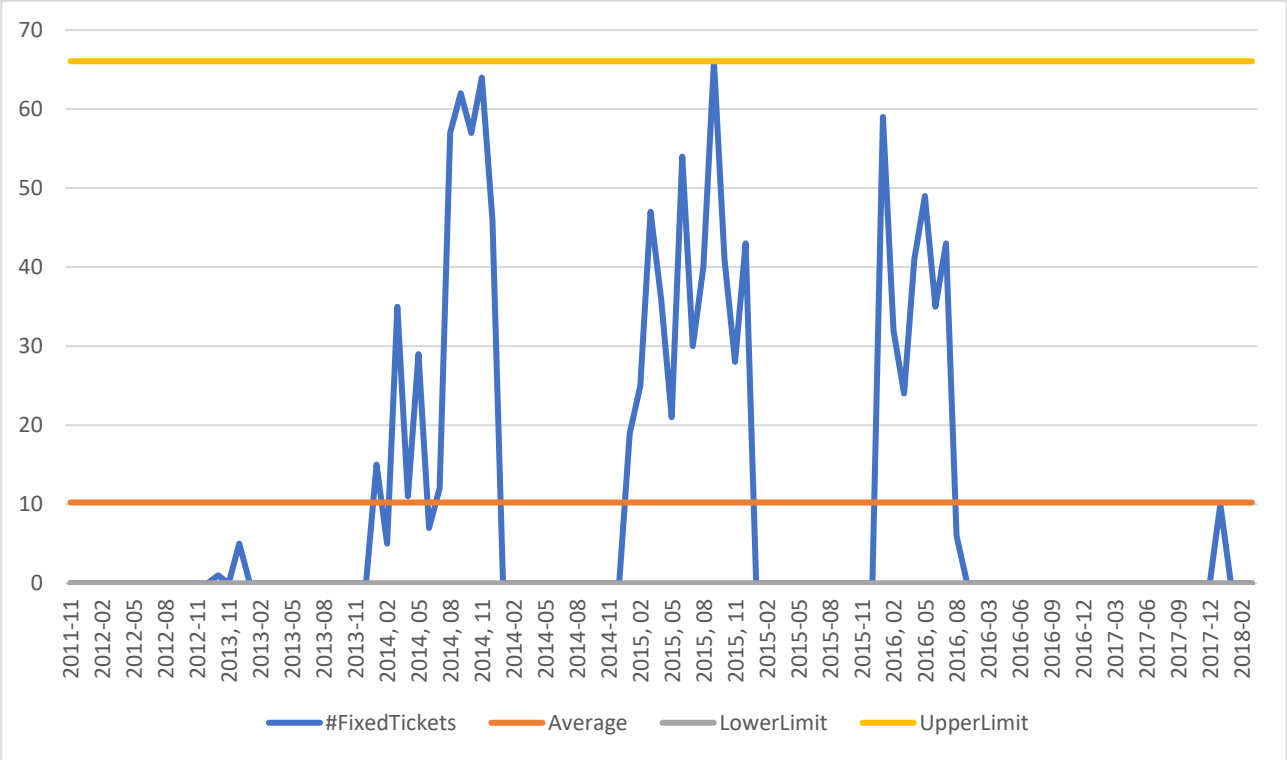
I risultati riportati nel csv sono poi stati graficati ed analizzati. In particolare si è calcolata la media dei ticket chiusi al mese, pari a *10,22*, e la deviazione standard. Tali dati sono stati utilizzati per definire un *Lower limit* e un *Upper limit*, di seguito riportati:

$$Lower\ Limit = max(Mean - 3*StdDev),0) = 0$$

$$Upper\ Limit = Mean + 3 * StdDev = 66,05$$

Di seguito viene riportato il process control chart realizzato con Excel. Come si può notare risulta esserci stabilità. Il numero di ticket al mese infatti non oltrepassa mai il limite superiore. Si ha solamente un punto in cui viene raggiunto l'upper limit. Tale anomalia può

essere utilizzata per individuare problematiche nel processo software e migliorarlo, ad esempio dedicando maggior effort alla fase di testing.



Link

- https://sonarcloud.io/dashboard?id=valevalefra_Deliverable1
- https://github.com/valevalefra/Deliverable1

Deliverable 2

Introduzione

L'obiettivo di questa deliverable è quello di fornire uno strumento per individuare le classi contenenti bug all'interno di un progetto, al fine di poter ottimizzare l'attività di testing. Infatti tramite analisi statistiche dei dati si costruisce una base di conoscenza in grado di predire il comportamento futuro di una classe con un certo grado di accuratezza.
A tale scopo si utilizzano modelli di Machine Learning.
Il Machine Learning è lo studio di algoritmi che hanno come scopo quello di migliorare le performance, attraverso una conoscenza pregressa dei dati in analisi.
Tra i diversi tipi di modelli di previsione, ci concentriamo sui classificatori, cioè modelli che mirano a stimare una variabile categorica, nel caso in esame l'esistenza o meno di un difetto in un modulo software.
Quello che si è cercato di fare per i progetti open source Bookkeeper e Avro è stato calcolare alcune metriche, successivamente descritte, e definire la buggyness per ciascuna classe. I dati raccolti in questa prima analisi sono poi stati utilizzati dai modelli di Machine Learning per definire la tendenza di una classe ad essere buggata in una versione futura.

Progettazione

Il progetto è stato articolato in due fasi:

- Prima fase, si è costruito una dataset contente informazioni relative a determinate metriche e alla buggyness di ciascun file Java contenuto nei progetti in esame.
- Seconda fase, si sono applicati modelli di Machine Learning al dataset ottenuto precedentemente, al fine poi di valutare il classificatore e le tecniche di ML migliori.

1. Costruzione del dataset

In questa prima fase si costruisce il dataset contenente i dati che verranno poi utilizzati dai modelli di Machine Learning. Il file di output conterrà per ogni file del progetto: versione, nome della classe, computazione di diverse metriche, difettosità della classe.
Le metriche selezionate sono:

- Loc_added: numero di linee di codice aggiunte;
- MAX_LOC_added: massimo numero di linee di codice aggiunte, relative ad una *revision* appartenente ad una release;
- AVG_LOC_added: numero medio di linee di codice aggiunte;
- Churn: differenza tra le righe di codice aggiunte e quelle eliminate;
- MAX_Churn: massimo numero di linee di codice aggiunte meno quelle eliminate relative ad una *revision* appartenente ad una release;
- AVG_Churn: numero medio della differenza del numero di righe di codice aggiunte e quelle eliminate;
- NR: numero di *revision* di una *release*;

- CghSetSize: numero di classi committate insieme;
- MAX_CghSet: massimo valore raggiunto per la metrica CghSetSize in relazione ad una certa *revision* di una versione;
- AVG_CghSet: media di CghSetSize sul numero di release.

L'analisi riguarda solo file con codice Java, si è infatti ritenuto opportuno scartare i file di altri linguaggi di programmazione. Inoltre sono state prese in considerazioni solo le release relative alla prima metà di vita del progetto. Questo perché l'esistenza di un difetto non può essere conosciuta prima che il difetto stesso sia stato risolto, quindi le release più giovani potrebbero contenere bug ancora aperti e di conseguenza non individuabili. La dinamica appena descritta prende il nome di snoring, e consiste appunto nella presenza di bug "dormienti" che creano *noise* nel dataset. L'analisi di release meno recenti permette quindi di poter stabilire con una maggiore attendibilità se una classe sia difettosa o meno, così facendo infatti la probabilità di avere classi "dormienti" scende al di sotto del 10%.

La costruzione del dataset si è articolata nei seguenti passi:

1. Recupero dati riguardanti le release di un determinato progetto attraverso l'uso delle API-REST di JIRA;
2. Individuazione release relativa alla prima metà del progetto;
3. Ricerca dei ticket di tipo bug aventi stato *closed* o *resolved*;
4. Per ciascun ticket individuato si calcola:
    - Injection version (IV) versione in cui è stato introdotto il bug;
    - Opening version (OV) versione in cui viene creato il ticket relativo al bug;
    - Fixed Version (FV) versione del codice in cui il bug è stato risolto.

    I valori di OV e di FV, corrispondono rispettivamente all'apertura e alla chiusura del ticket, IV invece è un valore che potrei non conoscere ed in tal caso viene stimato tramite il metodo *proportion* spiegato successivamente.

5. Recupero della lista di classi Java totali, tramite comando *git*;
6. Uso dei dati elencati nel punto 4 per definire la buggynes per ogni versione di ciascuna classe Java. Nello specifico vengono marcate come difettose tutte le versioni delle classi che rientrano nell' intervallo *affected version* (AV): [IV;FV-1].
7. Calcolo delle metriche tramite comandi *git*;
8. Scrittura file csv.

Proportion

Esiste una certa percentuale di casi in cui non si conoscere il valore di IV, per questo si è introdotto un nuovo algoritmo, il quale si basa sulla seguente idea: tutti i difetti hanno un ciclo di vita stabile. Ovvero se l'intervallo tra IV e OV è ampio allora lo sarà anche quello tra OV e FV. Questa intuizione ha portato alla possibilità di individuare una proporzione

costante tra i difetti e a stimare quindi il valore di IV. Quindi, viene calcolato il valore P tramite i ticket che possiedono il valore di IV, usando le seguente formula:

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

Successivamente è possibile calcolare il valore di IV, per i ticket che non lo possiedono, tramite la seguente equazione:

$$IV = FV - (FV - OV) * P$$

Tra i diversi metodi per computare P, si è scelto quello incrementale. Ovvero il valore di P si è calcolato in maniera incrementale come la media tra i difetti risolti nelle versioni precedenti.

2. Applicazione modelli ML

In questa seconda fase vengono usati i dati raccolti precedentemente per valutare quale modello di Machine Learning produce stime più accurate.
Data la dipendenza temporale delle release si è adattata una tecnica di valutazione di dati di tipo *time-sensitive*: *walk-forward*. Questo tipo di evaluation technique divide il dataset in parti, ovvero la più piccola unità che può essere ordinata, nel caso in esame le release.

I classificatori utilizzati sono:

- Random Forest;
- Ibk;
- Naive Bayes.

Le tecniche di balancing utilizzate sono:

- No Sampling;
- Oversampling, si cerca di raggiungere il valore della classe maggioritaria attraverso replicazione dei dati;
- Undersampling, si cerca di raggiungere il valore della classe minoritaria;
- SMOTE, si cerca di raggiungere il valore della classe maggioritaria, creando sinteticamente elementi della classe minoritaria.

Queste tecniche vengono introdotte con lo scopo di bilanciare il numero delle classi difettose e non difettose, così da avere più probabilità di migliorare le performance del classificatore.

Si è inoltre applicata *feature selection,* che consiste nel ridurre il numero di attributi correlati, questo permette sia di ridurre il costo di apprendimento del classificatore, sia di migliorarne l'apprendimento.

Attraverso la combinazione di classificatori, tecniche di *balancing* e *feature selection* si valuta l'accuratezza del classificatore tramite le seguenti metriche:

- Recall, indica quanti positivi il modello è in grado di classificare;
- Precision, indica quante volte il modello ha correttamente classificato un'istanza come positiva;
- Auc, indica quanto il modello è in grado di distinguere le classi;
- Kappa, valore compreso tra -1 e 1, indica quanto il classificatore è migliore rispetto ad un classificatore standard.

Per ciascun progetto si restituisce un file csv contenente le seguenti informazioni:

- Nome del progetto;
- Numero di release considerate nel training;
- Percentuale di dati utilizzati nel training;
- Percentuale di classi difettose nel dataset del training;
- Percentuale di classi difettose nel dataset del testing;
- Classificatore utilizzato;
- Tecnica di Balancing utilizzata;
- Tecnica di feature selection utilizzata;
- Numero di veri positivi (TP);
- Numero di falsi positivi (FP);
- Numero di veri negativi(TN);
- Numero di falsi negativi(FN)
- Precision;
- Recall;
- Roc Area;
- Kappa.

Risultati

Di seguito vengono graficati, medianti box-plot ottenuti da JMP, i valori risultanti dal file csv, per i progetti Bookkeeper [*fig.*1] e Avro [*fig.2*].

Bokkeepper: In questo caso l'uso o meno di feature selection non comporta differenze significative. La metrica Roc sembra presentare per tutti i classificatori, e per tutte le tecniche di balancing, diagrammi piuttosto schiacciati, fattore positivo poiché indica una minore dispersione dei dati, e valori della mediana piuttosto elevati. La metrica Recall, invece, mostra risultati migliori per i classificatori Ibk e Random Forest con tecnica di balancing Under sampling. Infine le metriche Kappa e Precision presentano diagrammi più schiacciati e con un valore della mediana più alto nel caso del classificatore Random Forest e tecnica di balancing SMOTE. Si giunge quindi alla conclusione che la configurazione migliore risulta essere quella con tecnica di balancing SMOTE, e classificatore Random Forest.

Fig1.

Avro: In questo caso i risultati migliori, per tutti i classificatori, sembrano essere raggiunti nel caso in cui non venga applicata la feature selection. In particolare per la metrica ROC i classificatori Ibk e Random Forest, nel caso di SMOTE, No Sampling e Over sampling, come tecniche di balancing, presentano valori elevati della mediana e diagrammi schiacciati, che indicano una minore dispersione dei dati. Inoltre si ha un buon valore della precision, la quale presenta un diagramma simmetrico e un valore piuttosto alto della mediana, con il classificatore Random Forest e tecnica di balancing SMOTE. Concludendo il risultato migliore si ottiene con il classificatore Random Forest, senza feature selection e con tecnica di balancing SMOTE.

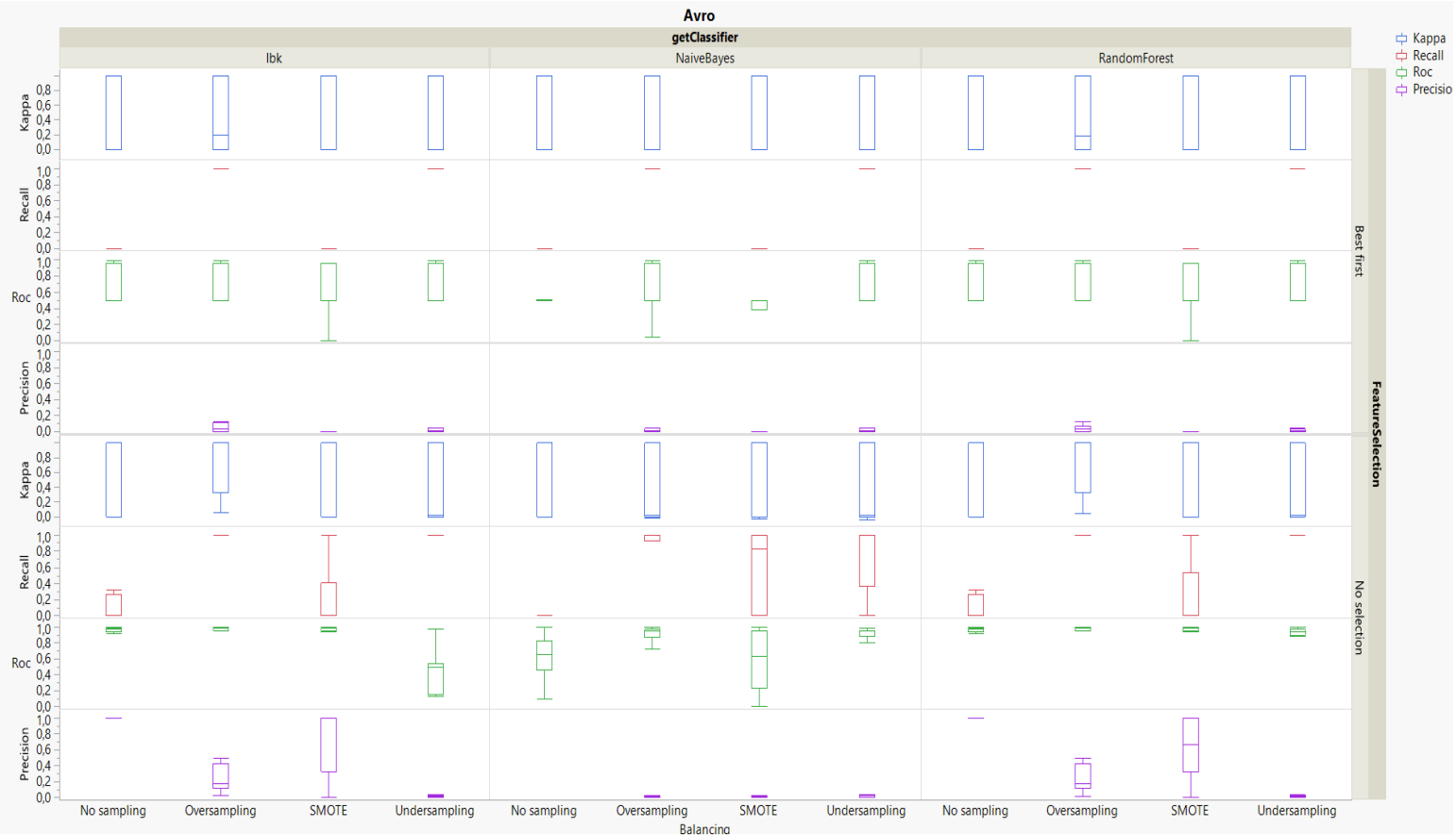


Fig2.

Link

- https://sonarcloud.io/dashboard?id=valevalefra_Deliverable2
- https://github.com/valevalefra/Deliverable2