Deliverable del Corso - Prof. Falessi

Leonardo Petraglia

January 26, 2021

Summary

1 Introduzione

Lo scopo del progetto è stato quello di effettuare analisi su due progetti Apache, BOOKKEEPER e ZOOKEEPER. Il progetto è stato cosi suddiviso:

- Deliverable 1, con lo scopo di misurare la stabilità di uno specifico attributo, nel particolare il numero di Fixed Tickets;
- Deliverable 2 Milestone 1, con lo scopo di creare un dataset contenente metriche calcolate per ogni classe del progetto volte a misurarne la difettosità;

- Deliverable 2 Milestone 2, con lo scopo di eseguire uno studio empirico volto a misurare l'effetto di tecniche di sampling e feature selection sull'accuratezza di modelli predittivi di localizzazione di bug nel codice.

I dati necessari sono stati raccolti tramite l'utilizzo combinato di due software: JIRA è stato utilizzato per recuperare i ticket dei progetti associati ai bug e alle nuove features, Git invece per recuperare tutti i commit relativi ai progetti. L'uso di questi software ha portato alla creazione di file csv la cui intersezione ha fornito la creazione del dataset desiderato.

2 Deliverable 1

2.1 Scopo

La prima parte del progetto è volta a misurare la stabilità di un attributo specifico del progetto e precendetemente selezionato tra: Fixed Bugs, ovvero il numero di bug corretti; Fixed New Feature, ovvero il numero di nuove funzionalità corrette e Fixed Tickets, ovvero l'insieme dei due precedenti attributi. Nel progetto in questione è stato analizzato il progetto PARQUET e in particolare il numero di Fixed Tickets. Come suddetto è stato necessario utilizzare JIRA, un software proprietario che consente il bug tracking, per recuperare tutti i ticket relativi al progetto tramite una opportuna query, e Git per recuperare tutti i commit relativi al progetto. Successivamente le informazioni precedentemente trovate sono state intersecate per avere un'associazione tra i ticket e i commit. Nel particolare è stato necessario filtrare i commit di Git sia per quanto riguarda uno standard predefinito, nel nostro caso [PROJECTNAME - TICKETID], sia per quanto riguarda il numero di commit relativi allo stesso ticket, infatti la soluzione migliore di avere un'associazione uno ad uno tra i due, ovvero che ad un determinato ticket corrisponda un solo commit, non è sempre rispettata e si è quindi scelto di prendere in considerazione solamente il commit più recente.

2.2 Implementazione

Questa porzione del progetto è organizzata in un unico package chiamato *logic* contenente due classi:

RetrieveJiraTicket.class

Questa classe effettua una query opportunamente creata su JIRA che restituisce tutti i Fixed Tickets del progetto PARQUET e li scrive all'interno di un file csv denominato *ticket.csv*

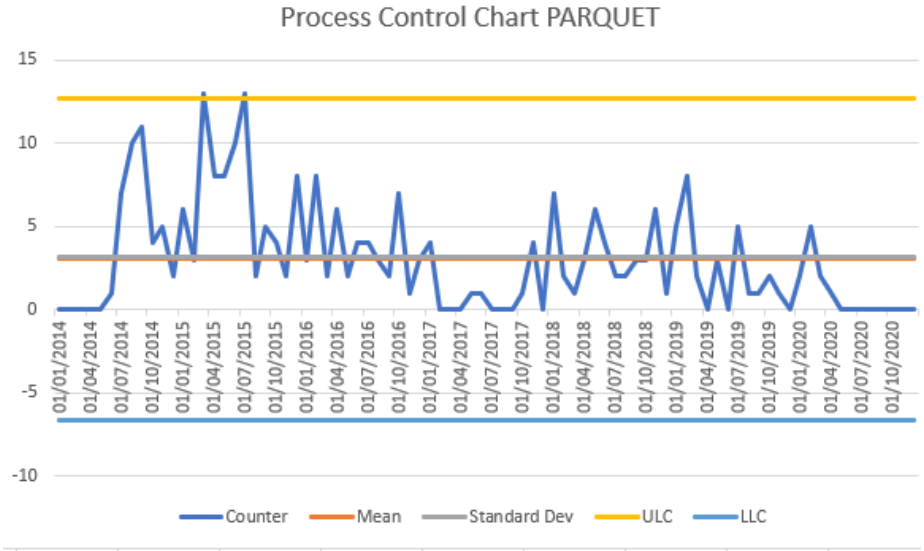
RetrieveCommit.class

Dopo aver clonato la repository in locale tramite Git, i commit del progetto vengono filtrati in base allo standard precedentemente definito e vengono quindi

scritti in un file csv denominato *commit.csv* avente come colonne la data e l'ID del ticket. Sucessivamente viene effettuata l'intersezione tra i file suddetti in base ai ticket, che è l'unico elemento in comune tra i due che infine viene scritta in un file csv denominato *final.csv*

2.3 Grafici

Prendendo in considerazione la figura notiamo come il periodo che va da Aprile 2014 a Luglio 2015 è quello di maggiore attività con notevoli picchi di commit, in particolare notiamo due picchi che superano il valore del limite superiore di controllo UCL, questo è probabilmente dovuto ad un flusso maggiore di commit relativi a nuove funzioni introdotte nel periodo iniziale di sviluppo piuttosto che di commit relativi a bug sistemati. Considerando il periodo di tempo successivo, il numero di commit è per la maggior parte delle volte superiore a zero con molti picchi che portano il numero al di sopra della media. Più si va avanti con il tempo più tali commit vanno a diminuire, a differenza del periodo iniziale suddetto è ipotizzabile che questi siano relativi a bug sistemati piuttosto che a funzionalità introdotte. Considerazione importante è che tale grafico potrebbe non includere tutti i ticket del progetto preso in considerazione, questo perchè è stata effettuata un'operazione di filtraggio di questi in base ad uno standard prescelto, in particolare "PROJECTNAME-IDTICKET", pertanto i commit che presentano errori ortografici nel nome del progetto o nell'identificativo del ticket non sono stati inclusi.



3 Deliverable 2 Milestone 1

3.1 Scopo

Lo scopo di questa porzione è la creazione di un dataset contenente delle specifiche metriche calcolate per ogni classe e ogni release dei progetti presi in esame. Nello specifico sono state considerate le seguenti metriche:

- Size, il numero di linee di codice
- Age, l'età della classe espressa in settimane
- NAuth, il numero di autori che hanno contributo alla scrittura del codice
- LOC touched, la somma delle linee di codice aggiunte, eliminate e modificate
- ChgSetSize, il numero di file commitati insieme alla classe
- MAX ChgSet, il numero massimo di file committati insieme alla classe per una determinata release
- AVG ChgSet, la media dei file committati insieme alla classe per una determinata release
- MAX LOC added, numero massimo di linee di codice della classe per una determinata release
- AVG LOC added, media delle linee di codice della classe per una determinata release

In aggiunta a ciò, opportunamente inserito in un file csv, ad ogni classe è associata la sua difettosità indicata con YES in caso affermativo, NO altrimenti. Tale difettosità viene determinata in base ai dati recuperati da JIRA, infatti le classi appartenenti ai ticket che hanno Affected Version vengono dichiarate difettose, mentre per quelle classi che appartengono a ticket senza Affected Version, è stato predetta la Injected Version tramite il metodo *Proportion*. Per una maggior chiarezza riportiamo di seguito le definizioni di alcuni termini:

- **Affected Version(AV):** è la versione del software che è affetta dallo specifico bug identificato dal ticket;
- **Injected Version (IV):** è la più vecchia Affected Version, ovvero la versione del progetto in cui tale bug/feature è stato introdotto;
- **Opening Version (OV):** è la versione del software in cui ci si è accorti del bug e si è aperto il ticket ad esso riferito;
- **Fixed Version (FV):** è la versione del software in cui il bug è stato sistemato.

È importante notare come la IV, OV ed FV sono uniche per ogni ticket, l'AV può non esserlo, è infatti ragionevole pensare che ci siano svariate versioni del software che presentino lo stesso bug e che esso sia stato scoperto e di conseguenza risolto soltanto diverse release successive.

3.2 Implementazione

Questa porzione è organizzata in tre package: uno denominato *milestones.uno* adibito alle operazioni effettuate su JIRA e Git e al calcolo delle metriche associate alle classi; uno denominato *entity*, contenente classi di supporto; uno denominato *utility* adibito alla creazione di directory, import di stringhe statiche relative ai percorsi dei file csv e ad operazioni riguardanti le date. Riportiamo a seguire alcune delle classi più importanti contenute nel package *milestones.uno*

JiraVersion.class

La classe consta di due metodi che effettuano una jQuery per recuperare da JIRA i ticket dei bug risolti che non hanno AffectedVersion e i ticket dei bug risolti che presentano sia Fixed Version sia Affected Version. In particolare in quest'ultimo caso, essendo le Affected Version molteplici, viene presa in considerazione soltanto la prima dal punto di vista temporale.

JiraBlameCommitIntersection.class

Troviamo qui tre metodi per effettuare un'associazione tra i dati trovati tramite JIRA e quelli di Git. Viene dapprima effettuata un'intersezione tra il file csv contenente i commit e quello contenente i blame dei file *.java* attraverso l'unico elemento in comune che troviamo tra i due, ovvero il git tree, un oggetto di Git che crea la gerarchia tra i file all'interno di una repository Git. In seguito il risultato di tale intersezione viene nuovamente intersecato separatamente prima con i ticket con Affected Version, dopo con i ticket con solo Fixed Version, il risultato di entrambe queste operazioni viene salvato infine in due file csv separati, rispettivamente *blameJiraIntersection.csv* e *blameJiraIntersectionFVOnly.csv*.

DefectiveClasses.java

Lo scopo di questa classe è quella di determinare quali classi sono difettose per ogni release del software. Come prima cosa vengono scritte in un file csv tutte le classi contenute nel file *blameJiraIntersection.csv* appartenenti alla prima metà delle release e viene indicata la loro difettosità con YES in quanto in quella release presentavano un bug. Successivamente per le classi contenute nei file di intersezione precedenti determiniamo la Opening Version, essa viene ricavata dalla data di creazione del ticket e "approssimata" alla data della release del software ad essa più prossima e infine trasformata nel suo corrispettivo indice utile successivamente per il calcolo tramite il metodo *Proportion*. Per le classi che non presentano una Affected Version non è naturalmente possibile determinare una Injected Version tuttavia non utilizzare questi dati potrebbe influire

negativamente sulla bontà del dataset, per questo motivo si è cercato di avere una previsione più accurata possibile della IV tramite il metodo proportion. Nello specifico si è scelto di usare la variante *Increment* che prevede il calcolo di un valore P come media tra i Fixed Defects nelle versioni precedenti del software. La formula utilizzata per il calcolo di P è la seguente:

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

Mentre per predirre la IV si usa:

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

A questo punto le classi con PredictedIV sono state aggiunte a quelle con AffectedVersion, sono state indicate come difettose e salvate in un file denominato *preFinalCsv.csv* insieme alle classi per ogni release assenti tra tutte quelle suddette e quindi indicate come non difettose.

CalculateMetrics

La classe serve a calcolare le metriche specificate nella sezione 3.1 oltre a contenere il main per l'esecuzione di tutti i metodi del progetto. Per evitare di avere valori duplicati all'interno del dataset finale è stato necessario l'utilizzo di ripetuto di un'HashMap⟨Keys, Values⟩ utilizzando come chiavi la classe e la sua versione del software di appartenenza e come valore tutte le metriche prima descritte e impostate a zero di default, nel momento in cui queste metriche possono essere calcolate per una determinata classe e release i valori di default vengono sostituiti. L'output finale consiste in un file csv denominato *finalCSV.csv* e che sarà il file di input per la milestone successiva.

4 Deliverable 2 Milestone 2

4.1 Scopo

La parte finale del progetto si pone come obiettivo quello di analizzare un dataset eseguendo uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling e feature selection sull'accuratezza di modelli predittivi di localizzazione di bug nel codice. Nello specifico andremo a comparare l'accuratezza di tre classificatori sui progetti Apache precedentemente selezionati tramite Weka, un software adibito all'analisi dei dati e all'apprendimento automatico.

4.2 Tecnica di Validazione

In accordo con le specifiche la tecnica di validazione è la Walk-Forward, una tecnica di tipo time-series che preserva quindi l'ordine cronologico dei dati, in cui il dataset è diviso in parti, nel nostro caso le release del software, e una volta ordinate temporalmente ad ogni esecuzione tutti i dati disponibili prima della

sezione da predirre sono usati come training set e la parte da predirre come test-set. Per fare ciò il dataset contenuto nel file *finalCSV.csv* è stato diviso come segue: sono stati creati tanti file di test quante le release del progetto, ovvero uno per ogni esecuzione, ognuno di essi contiene esclusivamente le classi con le rispettive metriche che appartengono ad una sola release e uno stesso numero di file di train contenenti tutte le classi che appartengono a tutte le release precedenti. Successivamente tutti questi file creati sono stati convertiti in file *.arff*.

4.3 Metriche

Tramite Weka sono state calcolate le seguenti metriche:

- True Positive: la percentuale di elementi predetti positivi e realmente positivi
- False Positive: la percentuale di elementi predetti positivi ma in realtà negativi
- True Negative: la percentuale di elementi predetti negativi e realmente negativi
- False Negative: la percentuale di elementi predetti negativi ma in realtà positivi
- Precision: il numero di volte in cui si è classificata una istanza come positiva
- Recall: il numero di positivi che sono stati classificati
- ROC Area: la probabilità che un'istanza positiva scelta casualmente è classificato al di sopra di un'istanza negativa scelta casualmente
- Kappa: quante volte si è stati più accurati rispetto ad un classificatore fittizio, tale valore è compreso tra -1 e 1, più esso si avvicina al valore ottimo k = 1 più l'accuratezza è buona

4.4 Feature Selection

Al fine di ridurre i costi di apprendimento e fornire delle performance migliori di apprendimento è importante selezionare soltanto un sottoinsieme degli attributi totali piuttosto che tutti quanti. Nel caso in esame è stata utilizzato l'approccio wrapper e in particolare la tecnica Backward Search che prevede di considerare tutti i possibili sottoinsiemi dell'insieme formato dal Feature Set meno un attribbuto e di scartare la feature che causa la minore perdità di accuratezza. Tale approccio risulta essere molto dispendioso dal punto di vista computazionale ma è adatto alle situazioni in cui il numero di attributi è contenuto, nonostante ciò porta dei vantaggi rispetto al considerare tutti gli attributi o rispetto ad altre tecniche come Best SubSet Selection.

4.5 Sampling

In prima istanza è stata effettuata l'analisi senza alcun tipo di sampling, lasciando il dataset cosi come è, sono state poi utilizzate tre tecniche diverse: **l'Over-Sampling**, **l'Under-Sampling** e **Smote**, un metodo di implementazione dell'Oversampling. L'Oversampling prevede di andare ad aumentare le classi minoritarie mentre l'UnderSampling va a diminuire il numero di esempi delle classi maggioritarie, entrambe hanno il fine di compensare uno squilibrio che è gia presente nei dati o che potrebbe svilupparsi se fosse prelevato un campionamento casuale, andando a creare pertanto un dataset bilanciato. In particolare per l'UnderSampling è stato configurato l'oggetto Resample con i parametri: -M 1.0, per l'OverSampling invece è stato configurato con -B 1.0 -Z percentage.

4.6 Implementazione

Oltre al package *utility* già descritto nella sezione 3.2 quest'ultima parte è organizzata in un unico package denominato *milestones.due* contenente tre classi:

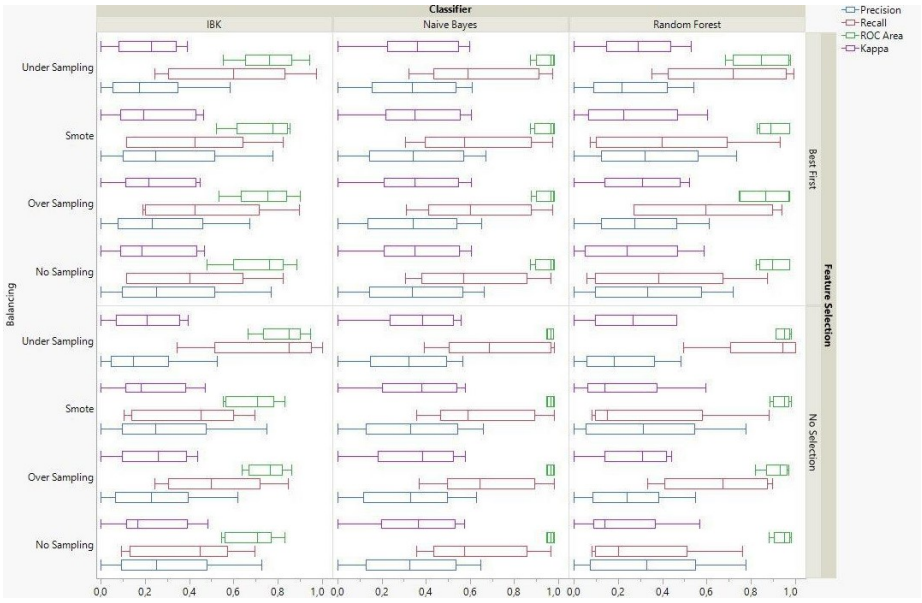
- **OrderCSV.class**, si occupa di creare i file csv per il training e per il testing, tramite i metodi *divideTraining* e *divideTesting*, con le modalità descritte nella sezione 4.2 oltre ad ordinare temporalmente i dati in essi contenuti;
- **ConvertToArff.class**, si occupa di convertire i file csv creati precedentemente in file *.arff* per avere una maggiore compatibilità con Weka;
- **Weka.class**, si occupa di eseguire la tecnica di validazione Walk Forward, tramite il metodo *executeWalkForward* che prende come parametro di input il numero di iterazioni che devono essere eseguite, e di calcolare le metriche suddette. I risultati vengono salvati all'interno di un file csv denominato *m2final.csv*.

4.7 Grafici

Di seguito vengono riportati alcuni commenti basati sui grafici ricavati dall'analisi dei due progetti, cercando di evidenziare quali tecniche di Feature Selection e Balancing aumentano l'accuratezza dei vari classificatori utilizzati.

4.7.1 Bookkeeper

Consideriamo dapprima la Feature Selection. Con la tecnica Best First utilizzata sembra esserci un aumento della variabilità riguardo il valore di Recall per tutti e 3 i classificatori utilizzati, con un cambiamento meno evidente per quanto riguardo Naive Bayes. Un discorso del tutto analogo può essere fatto per il valore della ROC Area che in questo caso però subisce un cambiamento meno marcato nel classificatore IBK. Per quanto riguarda il valore di Kappa esso rimane per lo più lo stesso con o senza Feature Selection e per tutti i classificatori. Consideriamo ora il valore della Precision, con i classificatori IBK e Naive Bayes diminuisce leggermente la variabilità se non usiamo la Feature Selection, con il classificatore Random Forest invece la variabilità della Precision aumenta se non utilizziamo la tecnica Best First.
Riassumendo notiamo che l'utilizzo di Best First comporta un aumento della variabilità dei valori di ROC Area e Recall, dove però nel classificatore IBK i cambiamenti sono meno visibili. Vediamo infine come le tecniche di sampling SMOTE e Over Sampling sembrano essere quelle che ottengono valori di Recall e Precision più alti, nel particolare se usati con il classificatore Naive Bayes e Feature Selection costituiscono la migliore configurazione.



4.7.2 Zookeeper

Come nel caso precedente andiamo a considerare come prima cosa l'impatto che ha avuto la Feature Selection sui nostri classificatori. Per quanto riguarda il classificatore IBK il valore di Kappa aumenta leggermente senza Feature selection; il valore della ROC Area diminuisce di variabilità ad eccezione del caso di OverSampling dove c'è un notevole aumento nonostante il valore della mediana rimanga uguale; la variabilità della Recall aumenta con l'eliminazione della feature selection ad eccezione dell'OverSampling in cui c'è una sostanziale diminuzione; infine il valore della variabilità della Precision diminuisce se non viene applicata la feature selection. Prendendo in considerazione il Classificatore Naive Bayes il valore di Kappa rimane pressochè invariato con o senza la feature selection, un discorso simile può essere fatto per il valore della ROC Area ad eccezione della tecnica di UnderSampling dove c'è una piccola diminuzione della variabilità; per quanto riguarda invece la Recall e la precision la variabilità rimane alta con qualsiasi tecnica di Sampling e feature selection, l'unico cambiamento riscontrabile è lo spostamento della mediana. Considerando infine Random Forest, la variabilità di Kappa aumenta senza feature selection, la variabilità della ROC Area diminuisce senza feature selection; la variabilità della Recall rimane invariata per le varie tecniche di sampling e feature selection ad eccezione dell'UnderSampling dove sembra aumentare; per quanto riguarda infine la Precision, nell'UnderSampling e in Smote la variabilità rimane invariata, nell'OverSampling diminuisce senza Feature Selection e senza sampling aumenta.