Report Ingegneria del Software e Web 2

Alex Ponzo

June 2020

Indice

Introduzione

1.1 Informazioni preliminari

Questo report ha lo scopo di descrivere due progetti i cui riferimenti sono specificati nella sezione 1.2, assieme ad uno studio dei risultati ottenuti tramite essi. I due progetti in questione sono:

- Deliverable 1: misurazione della stabilità di un attributo di progetto
- Deliverable 2: misurazione dell'effetto di diverse tecniche di sampling e feature selection sull'accuratezza dei modelli predittivi

In questo primo capitolo verranno descritti in breve gli strumenti utilizzati, per poi concentrare l'attenzione sui due deliverable negli altri due capitoli.

1.2 Strumenti utilizzati e riferimenti

1.2.1 SVN

Apache Subversion (SVN) è stato scelto come software di versionamento e controllo di versione, utilizzando **TortoiseSVN** come client grafico; la sua migliore gestione della storia dei commit (in particolare per quanto riguarda operazioni come la rinominazione) lo rende un'ottima alternativa a Git.
I repository dei due progetti utilizzati in questo studio possono essere reperiti su Github:

- Deliverable 1: https://github.com/Lexie93/RetrieveTickets
- Deliverable 2: https://github.com/Lexie93/Deliverable2

1.2.2 Maven

Apache Maven è uno strumento di gestione dei progetti in Java particolarmente efficace, che permette di definire le dipendenze e le librerie necessarie per effettuare la build del progetto utilizzando un costrutto denominato *Project Object Model* (POM). Questo strumento effettua automaticamente il download delle

librerie necessarie indicate nel file POM, in questo modo viene garantito che, qualora avvenga uno spostamento, vengano utilizzate sempre le stesse versioni di queste ultime. Per comodità è stato utilizzato il plugin **M2E** che rappresenta una versione integrata di Maven nell'IDE di Eclipse.

1.2.3 Travis-CI

Travis-CI è un servizio per il continuous integration utile ad automatizzare la build ed il test di progetti contenuti in repository Github; tramite il file di configurazione "*.travis.yml*" permette di specificare il linguaggio e l'ambiente desiderato su cui effettuare build. Una volta collegato alla repository Github, Travis avvierà una build automaticamente ogni volta che un commit viene effettuato su tale repository, indicandone l'esito.

1.2.4 SonarCloud

SonarCloud è un servizio online per il controllo della qualità del codice, risulta particolarmente utile per l'individuazione di bug e smell, offrendo anche una stima del debito tecnico, della duplicazione del codice e della copertura dei test; è possibile collegarlo alla build del progetto Maven realizzata su Travis-CI, in questo modo il commit su Github, oltre a comportare la build su quest'ultimo, porterà anche all'analisi del codice. Per comodità ed al fine di diminuire il numero di commit superflui è stato utilizzato **SonarLint**, un estensione per l'IDE di Eclipse che fornisce in locale un controllo della qualità simile a quello di SonarCloud.
Le analisi del codice di questo servizio sui due progetti utilizzati sono disponibili ai seguenti link:

- Deliverable 1: https://sonarcloud.io/dashboard?id=Lexie93_RetrieveTickets
- Deliverable 2: https://sonarcloud.io/dashboard?id=Lexie93_Deliverable2

1.2.5 Weka

WEKA (*Waikato Environment for Knowledge Analysis*) è un software open source per l'apprendimento automatico; consente di elaborare dei dataset (la parte di training) per cercare di stimare una classificazione di altre istanze istanze (testing). WEKA consente l'uso di diversi classificatori, tecniche di feature selection e di balancing dal dataset, risulta pertanto perfetto per lo studio eseguito nel secondo deliverable, dove costituirà la parte centrale eseguita dopo la costruzione del dataset. Per permettere una migliore integrazione di WEKA nel progetto viene utilizzata una libreria Java integrata tramite Maven (più una seconda libreria che conterrà il componente aggiuntivo per la tecnica di SMOTE balancing); ad ogni modo si è fatto uso anche dell'interfaccia grafica per il controllo dei risultati in molti passi.

Deliverable 1

2.1 Introduzione

In un'organizzazione risulta di fondamentale importanza cercare di migliorare le prestazioni seguendo un approccio atto al miglioramento dei processi di sviluppo; a tale scopo è necessario avere un modello che fornisca gli elementi essenziali per inseguire questo miglioramento, ed uno dei modelli più utilizzati è il **CMMI** (*Capability Maturity Model Integration*).
Tale modello si struttura su 5 diversi livelli (detti livelli di maturità), in cui per il raggiungimento di un determinato livello è necessario aver prima raggiunto quello precedente (non è quindi possibile saltarli) e ottenere determinati ulteriori progressi in campi definiti *Process Area*.
Con questa consapevolezza si vuole effettuare una misurazione della stabilità di un attributo in un progetto open source reperibile su Github [1]; lo studio della stabilità degli attributi è infatti una componente necessaria al raggiungimento dei livelli più alti del modello CMMI. Il progetto scelto per tale misurazione è **Parquet**, e l'attributo misurato è il numero di bug fix del progetto nel tempo; a tale scopo si è reso necessario reperire le informazioni congiuntamente dai riferimenti a Github ed a Jira [2], reperibili direttamente dalla pagina web del progetto [3], nel campo "*Get Involved*".

2.2 Progettazione

2.2.1 Descrizione del progetto

Il primo passo consiste nel reperire le informazioni del progetto su Jira, concentrando l'attenzione esclusivamente sui ticket inerenti bug che sono stati già risolti. A questo punto si rende necessario ricercare per ognuno di tali ticket la data di fix del bug inerente; Jira ha un campo presupposto a tale scopo ma si ritiene certamente più affidabile la storia dei commit del progetto su Github, motivo per cui tale data viene considerata essere quella dell'ultimo commit contenente l'identificativo del ticket Jira.
Va fatto presente comunque che in molti casi non viene rispettata una regola nell'uso di tale identificativo all'interno del progetto (la best practice sarebbe l'uso di parentesi quadre per contenere il ticket, es: "[PARQUET-123]", o co-

munque sarebbe bene attenersi ad una regola costante), ciò rende certamente più arduo il reperimento di tutti i commit cercati (un semplice controllo sul contenimento dell'id potrebbe infatti portare all'assunzione di falsi positivi, come il caso della ricerca del ticket "PARQUET-1" che otterrebbe un match anche su "PARQUET-12" e simili); a tale scopo è stata quindi necessaria l'implementazione di due **regex** che consentono l'ottenimento esclusivamente degli id cercati:

- una che esclude i pattern con un numero subito dopo il match;
- una per comprendere i match a fine riga senza caratteri dopo.

Le date ottenute vengono a questo punto ordinate temporalmente e poi esportate su un file denominato "*dates.csv*", che verrà utilizzato come base per un'analisi che farà leva, tra le altre cose, su concetti statistici.

2.2.2 Organizzazione del codice

Il codice del repository *RetrieveTickets* è organizzato in 2 package:

- *retriever*, contenente le seguenti classi:
  - Main, è la classe principale
  - RetrieveGitLog, incaricata delle interrogazioni al log di Git per recuperare le date di fix di ogni bug
  - RetrieveJiraTickets, incaricata delle interrogazioni a Jira per il reperimento dei ticket
- *utilities*, contenente le seguenti classi:
  - CSVWriter, necessaria all'esportazione dei risultati in formato csv
  - JSONReader, necessaria all'elaborazione delle interrogazioni che restituiscono risultati in formato JSON

Le dipendenze necessarie per la build del progetto, specificate nel file POM, sono 2:

- *common-csv*
- *json*

2.3 Risultati ed analisi

Il file in cui viene effettuata l'analisi dei risultati ottenuti dai passaggi precedenti è nominato "*ProcessChart.xlsx*", qui il file precedente viene importato ed usato come base per la costruzione di una tabella pivot ed un grafico pivot; questi strumenti consentono un'analisi il più possibile dinamica, raggruppando le date per periodi, includendo anche quelli senza alcuna attività eseguita.
A questo punto vengono calcolati degli indici statistici basati sul conteggio delle date analizzate:

- Media calcolata rispetto al periodo di attività del progetto;
- Deviazione standard nello stesso periodo.

I dati calcolati che dovranno essere inseriti nel grafico pivot vengono dunque inseriti all'interno della tabella (il grafico può rappresentare esclusivamente valori presenti sulla propria tabella); un unico limite riscontrato nell'uso della tabella pivot è che i valori dei campi in essa non possono fare riferimento al valore di una singola cella esterna alla tabella stessa, ma dovranno essere specificati staticamente (calcolando prima il valore ed inserendolo poi).

2.3.1 Risultati ottenuti

I valori rappresentati nel grafico pivot costruito a partire dalla tabella appena descritta saranno:

- il conteggio dei bug fix raggruppati mensilmente, con i valori specificati punto per punto;
- una retta parallela all'asse x con ordinata pari alla media calcolata precedentemente;
- una retta parallela all'asse x che rappresenta un valore limite inferiore, pari alla media meno tre volte la deviazione standard (tale valore viene posto in una funzione *Max* assieme allo 0, per assicurarne la non negatività);
- una retta parallela all'asse x che rappresenta un valore limite superiore, pari alla media più tre volte la deviazione standard.

Tali valori calcolati vengono qui di seguito esposti:

Valore	**Descrizione**	**Risultato**
Ticket totali	Raggruppati poi mensilmente	234
Media	Calcolata sul periodo di attività	3.34286
STDV	Calcolata sul periodo di attività	3.2421
Max(0, Media - 3*STDV)	Limite inferiore	0
Media + 3*STDV	Limite superiore	13.0691

Tabella 2.1: Valori calcolati

Il grafico costruito con le precedenti informazioni è il seguente:

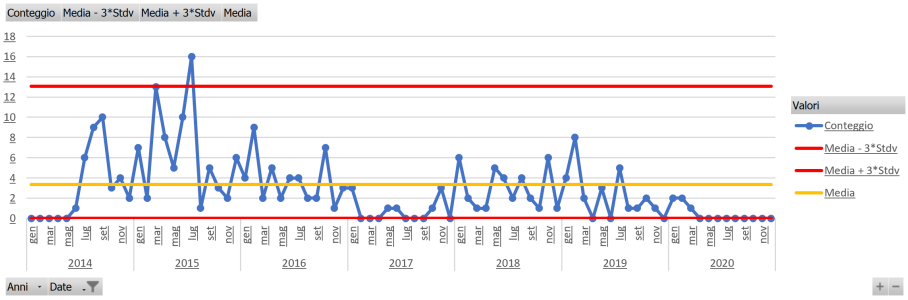

Figura 2.1: Parquet - Software Analytics

2.3.2 Analisi dei risultati

Analisi preliminare

Come è possibile notare il conteggio dei bug fix del progetto rinvenuti a partire dai ticket di Jira e dalla storia dei commit su Github è contenuto per la maggior parte del tempo all'interno dei limiti calcolati (le due rette rosse); l'unico momento in cui risulta essere superiore al limite è a Luglio del 2015 con un valore pari a 16, dopo essersi avvicinato molto alla soglia a Marzo del 2015, per poi stabilizzarsi sempre più con l'avanzare del tempo su valori più bassi, il che porta anche la media ad avere un valore relativamente basso, più vicino alla soglia inferiore che quella superiore (chiaramente il grafico non potrà mai risultare inferiore alla soglia minore poiché questa è posta pari a 0).
I motivi dietro questa differenza tra la fase iniziale e quella avanzata del progetto possono essere molteplici, un progetto ancora in fase embrionale avrà probabilmente più bug da risolvere, alcuni dei quali potrebbero aver bisogno anche di qualche mese prima di essere scoperti (questo fenomeno è denominato **snoring** o **sleeping defect**), ciò spiegherebbe la crescita iniziale.
Un altro fenomeno che può risultare interessante analizzare è la grande differenza che si presenta in alcuni casi tra valori successivi del grafico (ad esempio Luglio - Agosto 2015), essa può essere dovuta alla volontà di ottenere una versione più stabile nel primo mese o ad una scarsa attività (intesa come numero di commit) nel secondo.
Una semplice analisi come quella appena effettuata può comunque suggerire soltanto delle ipotesi dell'accaduto, senza poter mostrare quelle che sono le cause scatenanti; a tal scopo risulta necessario uno studio più approfondito con una raccolta di maggiori informazioni. Nella seguente sezione verranno utilizzati degli strumenti che saranno presentati più avanti (nel corso del capitolo inerente la deliverable 2) per reperire informazioni aggiuntive che siano in grado di chiarire il comportamento anomalo del grafico in determinati punti; l'attenzione sarà pertanto focalizzata, per ora, esclusivamente sull'analisi di tali informazioni.

Analisi approfondita

Grazie al codice relativo alla deliverable 2 è stato possibile reperire le informazioni relative alle varie release del progetto Parquet. Dall'analisi di queste ultime è possibile notare che, come ipotizzato, il picco di bug fix (a Luglio 2015) è dovuto quasi certamente alla necessità di ottenere una versione più stabile al fine del rilascio di una release; in particolare in quello stesso mese sono state rilasciate ben due diverse release (il minor update 1.8.0 e la patch 1.8.1), questo accade soltanto in questo mese, motivo per cui non vi sono altri picchi simili in periodi diversi. Il valore anomalo precedente (a Marzo 2015), che quasi raggiunge la soglia superiore, risulta invece il mese prima del rilascio di un altro minor update (1.6.0), che è il primo rilascio di una release dopo un periodo di quasi un anno, a conferma della volontà degli sviluppatori di ottenere una versione il più stabile possibile poco prima dell'uscita di una release; questo trend è anche utile a chiarire il motivo della differenza così netta del numero di bug fix in mesi consecutivi.
Un ultimo aspetto da tenere in considerazione è la presenza nel grafico di mesi in cui il numero di bug fix è pari a 0 (ad esempio molti mesi del 2017); in questi casi è stata effettuata un'analisi del numero di commit totali effettuati in tali periodi (senza filtri di sorta), ed è possibile riscontrare che gli zeri risultano in concomitanza con la scarsità o la totale assenza di questi ultimi, evidente segno che il basso numero di risoluzioni di bug è dovuto principalmente alla bassa attività in generale sul progetto.
Rappresentando graficamente l'andamento nel tempo del numero di commit mensili si può notare immediatamente la correlazione mostrata:

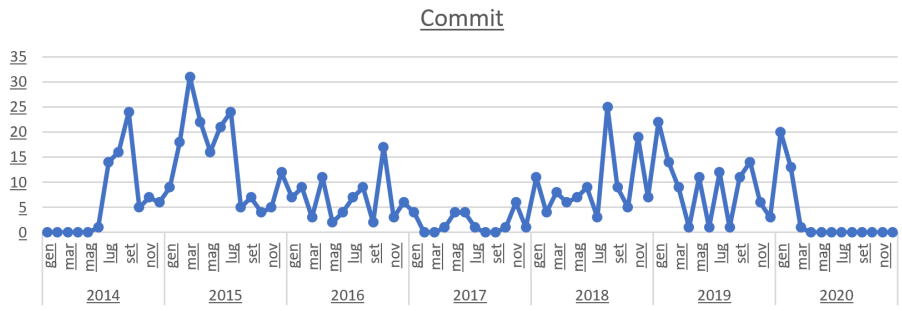

Figura 2.2: Parquet - Commit

Deliverable 2

3.1 Introduzione

Lo scopo di questo studio è quello di analizzare come le prestazioni di determinati tipi di modelli predittivi vengano modificate dell'uso di:

- **Feature Selection**: una tecnica in grado di ridurre sostanzialmente il "costo" dell'apprendimento del modello riducendo il numero di attributi, in molti casi permette anche di migliorare la performance finale;
- **Sampling**: una tecnica che consente di sopperire allo sbilanciamento delle classi di input del modello, in modo da non ignorare quelle meno presenti.

In particolare si vuole misurarne le prestazioni su dei dataset costruiti allo scopo di classificare la presenza di bug nel codice di larghe applicazioni open source. Le applicazioni su cui si concentrerà lo studio sono due:

- Apache Bookkeeper [4]
- Apache Syncope [5]

Lo studio si costituisce di due fasi consecutive (da portare avanti parallelamente in entrambi i progetti presi in esame):

1. Costruzione di un dataset a partire dalle informazioni dei progetti ottenibili da Jira [6][7] e GitHub [8][9]
2. Elaborazione del dataset tramite WEKA, analizzando le varie release dei progetti con la tecnica **walk forward**, e costruzione di file per il confronto degli output con e senza le metodologie di sampling e feature selection prese in esame

I risultati ottenuti saranno poi analizzati con l'ausilio di strumenti quali **JMP** per evidenziarne gli aspetti cruciali.

3.2 Progettazione

3.2.1 Costruzione del dataset

Questo primo passo risulta il più oneroso in termini di progettazione, sarà infatti necessario incrociare i dati forniti dal log di git e da Jira per ottenere le informazioni necessarie ai modelli predittivi nel passo successivo.
Per prima cosa vengono recuperate le release del progetto da Jira assieme alle loro informazioni, come il nome e la data di rilascio; si procede poi all'identificazione dei commit che hanno contribuito alla release (tutti quelli tra la release in esame e quella precedente) e dei file presenti in quest'ultima, reperiti tramite git al momento dell'ultimo commit che gli appartiene. In seguito alla rimozione delle release senza commit associati, verrà poi presa in esame esclusivamente la prima metà delle release ottenute (questo per limitare fortemente il problema dello sleeping defect che finirebbe con il falsare in modo importante il dataset).
Un ulteriore dato fondamentale da recuperare da Jira è l'insieme dei ticket associati ai bug risolti del progetto, assieme alle release affette dal bug stesso (in questo modo sarà possibile identificare la prima di queste come **Injected Version**); le altre release indicate saranno invece scartate, poichè la **Fixed Version** è ritenuta più affidabile se rinvenuta cercando l'identificativo del ticket tra i commit di git ed associando la release a cui appartiene l'ultimo di tali commit (per questa operazione vengono utilizzate le **regex** in maniera similare a quanto visto nel primo deliverable). Infine la **Opening Version** sarà trovata associandogli la prima release successiva alla data di creazione del ticket su Jira.
Passando agli aspetti che riguardano git, questo verrà utilizzato per recuperare dalla storia dei commit le informazioni riguardanti la creazione e la modifica[1] dei file nel corso delle varie release, distinguendo le operazioni relative al fix dei bug dalle altre.
I campi con cui il dataset verrà costruito sono i seguenti:

Indice	**Nome**	**Descrizione**
1	Version	il numero della release in esame
2	File	il nome del file appartenente alla release
3	NR	numero di revisioni al file effettuate dai commit appartenenti alla release
4	LOC	lunghezza del file espressa in linee
5	LOC_touched	somma delle linee del file modificate dai commit della release (uguale alla somma di linee aggiunte e linee rimosse)
6	LOC_added	somma delle linee aggiunte al file dai commit della release

[1] *Nel corso dell'analisi dei commit viene tenuta traccia delle rinominazioni effettuate sui file tramite due strutture dati (speculari tra loro) in modo tale da poter associare ad ogni file le proprie modifiche anche nel caso esse siano state effettuate su un diverso nome*

7	MAX_LOC_added	il massimo di linee che sono state aggiunte al file da un commit nella release in esame
8	AVG_LOC_added	numero medio di linee aggiunte al file dai commit nella release in esame
9	Churn	differenza tra linee aggiunte e linee rimosse al file sommate su tutti i commit della release
10	MAX_Churn	massimo dei churn del file nei commit della release
11	AVG_Churn	media dei churn del file nei commit della release
12	NAuth	numero di autori del file (al momento del rilascio della release)
13	ChgSetSize	numero di file inclusi in un commit della release assieme al file in esame (somma sui commit)
14	MAX_ChgSetSize	massimo numero di file inclusi in un commit della release assieme al file in esame
15	AVG_ChgSetSize	media del numero di file inclusi in un commit della release assieme al file in esame
16	AgeInWeeks	età del file espressa in settimane (tempo tra la creazione ed il rilascio della release)
17	AgeWeighted	età del file pesata sulle LOC_touched
18	NFix	numero di bug fixati nel file dai commit della release
19	Bugged	indica se il file in esame è buggato o meno, sarà il classificatore delle istanze del dataset

Tabella 3.1: Campi del dataset

Scelte progettuali

Nel corso dello sviluppo del progetto è stato necessario fare alcune scelte (come il metodo per stimare le IV, che verrà approfondito nella prossima sezione), ma quella probabilmente più importante consiste nell'utilizzare nel calcolo delle metriche esclusivamente i commit appartenenti al branch principale. Tale scelta dipende dal fatto che altrimenti risulta possibile (ed in Bookkeeper accadrebbe molto spesso) che vengano associati ad una determinata release i commit di un branch secondario il cui merge nel master avviene solo dopo la data di rilascio della release (solo perchè il commit è avvenuto prima di quest'ultima temporalmente); il lato negativo di questa decisione è che una parte dei dati vada persa, ma il lato positivo è che tali dati, se presenti, avrebbero falsato il dataset perché inseriti con valori sbagliati.
Anche nel calcolo di alcune delle metriche vengono prese delle decisioni sul metodo da utilizzare, ad esempio le LOC di un file possono essere contate in vari modi; in questo caso si è deciso di non distinguere le righe tra loro (se vuote, commenti, ecc.), per mantenere una coerenza interna con le metriche che riguardano le modifiche alle righe stesse effettuate dai commit, che non fanno tali distinzioni. Per altre metriche invece, come la AgeWeighted, si è fatto riferimento a Moser *et al*[10].

Individuazione dell'Injected Version

Come già indicato l'IV del bug (la release nel quale il bug ha fatto la sua comparsa) è un'informazione che viene prelevata da Jira, ma non sempre questa viene specificata; in questi casi sorge la necessità di stimarne una con le informazioni in possesso. Un valido metodo per la stima è il **Proportion**, che si serve della conoscenza pregressa degli altri ticket analizzati e della FV ed OV di quello attuale; nello specifico l'ipotesi di base è che esiste una certa proporzione tra il tempo necessario a scoprire un bug ed il tempo necessario a risolverlo. Esprimendo in formule questa affermazione è possibile calcolare:

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

stimando tale propozione come costante e calcolandola per i ticket precedenti, nel momento in cui ci si trova con un ticket privo di IV è dunque possibile ricavarlo tramite:

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

Esistono diversi metodi per ottenere la P del ticket a partire da risultati pregressi, quello utilizzato in questo studio è il **Moving Window** che la calcola come la media dell'ultimo 1% dei bug risolti (ordinati temporalmente per FV); questa scelta è dovuta alla convinzione che tale metodo si adegui meglio all'evoluzione nel tempo del progetto, ignorando i risultati più vecchi.
Prima di poter attuare tale stima comunque è necessario avere a disposizione dei dati espliciti tramite cui ricavare P, per far ciò, dopo aver ordinato temporalmente i bug per FV in modo da rispettarne il life-cycle, nel caso in cui si abbiano dei ticket senza IV specificata prima di aver ricavato una P utilizzabile, si ricorre al metodo **Simple**, ovvero viene posta IV=OV; è importante tenere in considerazione come per il calcolo di P sia necessario ignorare tutti i ticket che presentino FV=OV, visto che ciò renderebbe nullo il denominatore nella formula usata. Infine si fa notare che questa stima potrebbe portare ad una IV≥OV, il che sarebbe assurdo, pertanto nel progetto del deliverable vengono inclusi dei controlli per accertarsi che FV≥OV≥IV.

3.2.2 Elaborazione del dataset

In questo seconda fase viene fatto uso di WEKA per cercare di predire la classificazione dei file delle release del dataset (*testing*) utilizzando ad ogni passo le precedenti release per l'algoritmo di apprendimento (*training*)[2]; pertanto per N release vengono effettuati N-1 passi, in cui al passo i-esimo vengono utilizzate le release fino alla i per il training e la release i+1 per il testing, questo approccio viene definito walk forward ed è particolarmente indicato nei casi come questo

[2]Nella classe Weka del codice è presente una costante nominata "*SPLIT_PERCENTAGE*" attualmente fissata ad 1, tale valore esprime la percentuale del dataset da utilizzare per il metodo walk forward; cambiando tale percentuale la parte del dataset rimanente non verrà usata e potrà essere utilizzata in un secondo momento come testing

in cui si abbiano istanze ordinate temporalmente.
Ad ogni passo del walk forward vengono effettuate stime delle classificazioni dei file della release di test e conseguenti valutazioni dei risultati ottenuti con tutte le possibili combinazioni dei seguenti tre fattori:

- Feature Selection:
  1. nessuna
  2. **Best First**: seleziona gli attributi da utilizzare per il training con metodo greedy, provandoli tutti singolarmente, scegliendo il migliore, provando tutti gli altri in coppia con quello scelto precedentemente ed aggiungendo il migliore; così via finchè non si ottengono più miglioramenti.
- Balancing:
  1. nessuno
  2. **Undersampling**: bilancia il dataset diminuendo il numero di istanze della classe presente in quantità maggiore utilizzate per il training
  3. **Oversampling**: bilancia il dataset selezionando più volte le stesse istanze della classe presente in quantità minore nel training (vengono selezionate con reinserimento)
  4. **SMOTE**: bilancia il dataset generando delle istanze sintetiche della classe di minoranza a partire da quelle presenti
- Classificatori:
  1. **Naive Bayes**
  2. **Random Forest**
  3. **IBk**

Le informazioni riguardanti le valutazioni delle performance dei vari modelli saranno registrate utilizzando i seguenti campi:

Indice	**Nome**	**Descrizione**
1	Dataset	il nome del progetto in esame
2	#TrainingRelease	il numero di release utilizzate per il training (equivalente al passo del walk forward)
3	%Training	la percentuale del dataset totale utilizzata per il training (calcolata sul numero di istanze, non di release, per sopperire al fatto che release diverse hanno spesso un numero di file diverso)
4	%DefectiveInTraining	la percentuale di istanze nel training che presenta bug
5	%DefectiveInTesting	la percentuale di istanze nel testing che presenta bug

6	Classifier	il classificatore utilizzato
7	Sampling	la tecnica di sampling utilizzata
8	FeatureSelection	presenza o meno del filtro Best First
9	TP	il numero di veri positivi
10	FP	il numero di falsi positivi
11	TN	il numero di veri negativi
12	FN	il numero di falsi negativi
13	FPRate	la proporzione di falsi positivi sui negativi totali ($\frac{FP}{FP+TN}$)
14	TNRate	la proporzione di veri negativi sui negativi totali ($\frac{TN}{FP+TN}$)
15	FNRate	la proporzione di falsi negativi sui positivi totali ($\frac{FN}{TP+FN}$)
16	Precision	la proporzione di veri positivi sui positivi stimati ($\frac{TP}{TP+FP}$)
17	Recall	la proporzione di veri positivi sui positivi totali ($\frac{TP}{TP+FN}$), corrisponde a TPRate
18	ROCArea	fornisce un'indicazione di quanto il modello è in grado di distinguere le due classi
19	Kappa	indica quanto il classificatore è migliore rispetto ad uno che classifica in maniera casuale

Tabella 3.2: Campi della valutazione

3.2.3 Organizzazione del codice e dei risultati

Il codice del repository Deliverable2 è organizzato in 3 package:

- datasetbuilder, contenente le seguenti classi:
  - Main, è la classe principale
  - RetrieveGitLog, incaricata delle interrogazioni al log di Git
  - RetrieveJiraReleases, incaricata delle interrogazioni a Jira per il reperimento delle release
  - RetrieveJiraTickets, si occupa delle interrogazioni a Jira per il reperimento dei ticket
  - Weka, si occupa della fase di predizione

- entities, contenente le seguenti classi:
  - ClassifierEvaluation, necessaria a raccogliere le informazioni delle valutazioni delle prestazioni dei modelli predittivi
  - CommitWithChanges, incaricata di mantenere le informazioni sulle modifiche ai file effettuate da un commit
  - JiraTicket, contenente i dati raccolti riguardanti un ticket
  - Release, si occupa della gestione delle informazioni generiche di una release al momento del suo rilascio
  - ReleaseFile, riguarda i file con le proprie informazioni al momento del rilascio della release di appartenenza, contiene i dati finali del dataset di input per il modello predittivo
- utilities, contenente le seguenti classi:
  - CSVWriter, necessaria all'esportazione dei risultati in formato csv
  - JSONReader, incaricata dell'elaborazione delle interrogazioni che forniscono risultati in formato JSON

Le dipendenze necessarie per la build del progetto, specificate nel POM sono 4:

- commons-csv
- json
- weka-stable
- SMOTE

Il risultato dei passaggi precedenti viene esportato suddiviso in diversi file, ciascuno con nome del progetto posto come prefisso a:

- *VersionInfo.csv*: contenente le informazioni riguardanti le release del progetto
- *Tickets.csv*: contenente le informazioni riguardanti i ticket dei bug risolti del progetto (nell'intera durata) ed oltre a specificare OV, FV e IV indica se quest'ultimo è stato ottenuto direttamente o stimato
- *VersionFiles.csv*: il file principale costruito nel primo passo, contenente il dataset utilizzato dai modelli predittivi, con i campi specificati nella sezione precedente
- *Evaluation.csv*: il file definitivo contenente le valutazioni delle stime dei modelli predittivi sul dataset nei vari passi del walk forward

3.3 Analisi dei risultati

L'utilizzazione di due progetti invece di uno solo consente di fare considerazioni più ampie, distinguendo situazioni generatesi per motivi esterni da quelle realmente comportate da elementi considerati nei dataset. A questo scopo in un primo momento si concentrerà l'attenzione esclusivamente su uno dei due progetti, per poi utilizzare il secondo come confronto.

3.3.1 Bookkeeper

Questo progetto presenta solamente 14 release su Jira (una delle quali non contiene commit nel suo periodo di riferimento), pertanto una volta esclusa la seconda metà per arginare il problema dello sleeping defect, quelle restanti verranno analizzate in 6 passi tramite il walk forward.
Delle metriche raccolte per la valutazione delle stime quelle più interessanti sono senz'altro Precision, Recall, Kappa e ROCArea, che verranno analizzate nello specifico a breve; risulta però di fondamentale importanza evidenziare come le valutazioni delle stime siano affette da una enorme varianza lungo i vari passi del walk forward. Al fine di puntare i riflettori su questo aspetto è stato realizzato un grafico cumulativo delle suddette quattro metriche:

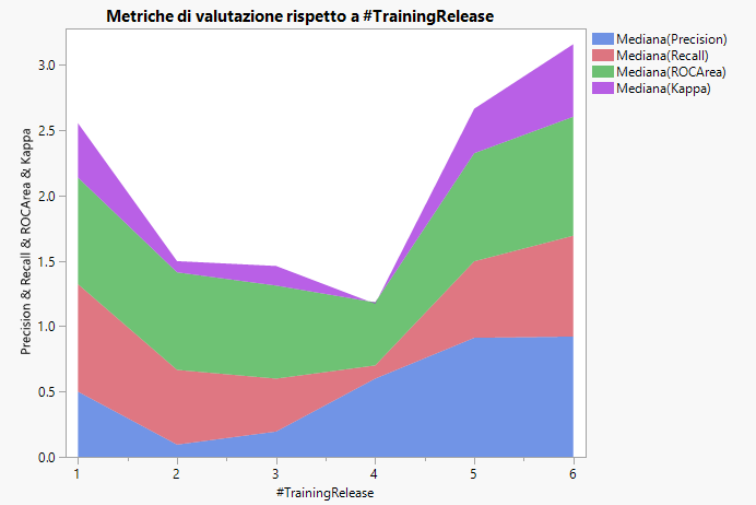

Figura 3.1: Metriche per il Walk Forward di Bookkeeper

Come è possibile notare le prestazioni dei modelli predittivi hanno due punti relativamente bassi in corrispondenza del secondo e quarto passo. Tale comportamento è probabilmente dovuto ad aspetti diversi, nel secondo passo il numero di istanze buggy nel testing set scende molto, passando dal 42% della prima release ed il 23% della seconda al solo 4-5% della terza. Il punto di minimo assoluto nel quarto passo è invece dovuto al comportamento inverso del dataset in

questo punto, ovvero la percentuale di istanze buggy si innalza drasticamente, con un 16% nel training set ed un 67% nel testing set; questo gran numero di istanze buggy continua a rimanere tale anche nei passi successivi, portando così il modello ad assestarsi ed avere prestazioni molto migliori.
Con questa consapevolezza si può ora spostare l'attenzione sulle metriche prese singolarmente:

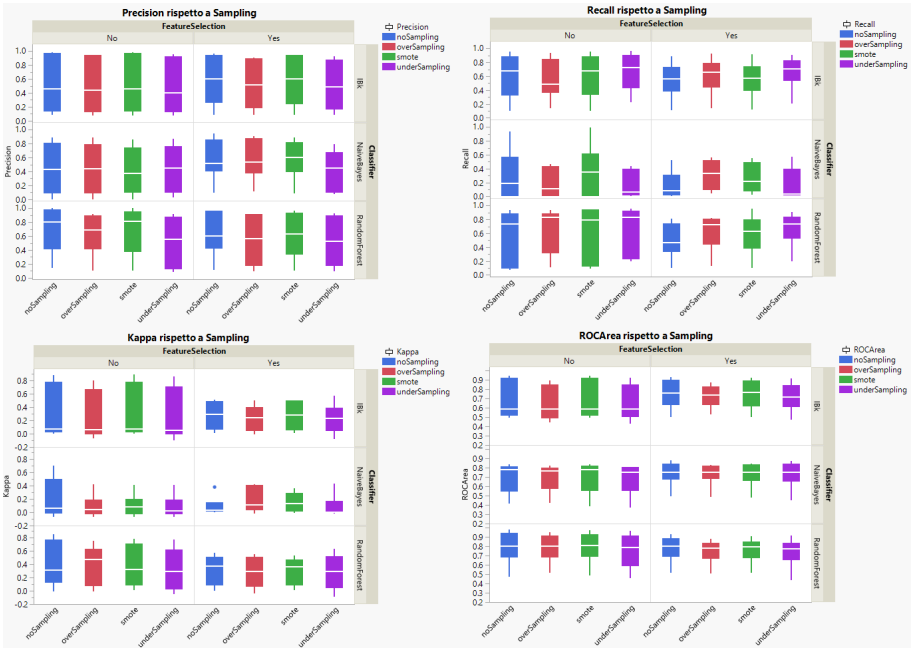

Figura 3.2: Metriche di valutazione in Bookkeeper

L'utilizzo di un box-plot per la rappresentazione permette di notare ancora una volta la caratteristica sopra descritta della grande variabilità delle metriche nei vari passi del walk forward; questo si può vedere dall'altezza dei box (la differenza tra il primo ed il terzo quartile, escludendo quindi gli outlier) e sta ad indicare come in alcuni passi le prestazioni del modello risultino molto migliori che in altri. Avendo già analizzato tale aspetto, ci si può soffermare sulle informazioni principali dello studio, ovvero i valori dei quartili (in particolare il secondo, la mediana) e come questi vengano modificati con l'utilizzo di sampling e feature selection.

Sampling

Dal modo in cui i box-plot sono distribuiti risulta spontaneo confrontare in primo luogo i diversi tipi di sampling rispetto al caso "noSampling", ovvero senza l'uso di tale tecnica, lasciando invariate le altre variabili in gioco (classificatore

e FS). Iniziando l'analisi dalla metrica Precision è evidente come qui l'utilizzo di undersampling sia il meno indicato, poiché peggiora le prestazioni (o le varia di poco rispetto al caso senza sampling) in praticamente tutti i casi, in tutti e tre i classificatori, con o senza filtro; le varianti in cui soffre particolarmente sono Naive Bayes con FS e Random Forest con o senza FS. Le altre tre possibilità di sampling forniscono risultati più simili tra loro, con la tecnica dell'oversampling piuttosto simile al caso noSampling, peggiorandolo in alcuni casi, e la tecnica SMOTE che si rivela la migliore in molte occasioni (assieme al noSampling).
Passando all'analisi della Recall si può notare come anche per questa metrica SMOTE si riveli una delle scelte migliori in praticamente tutti i casi; questa volta però va evidenziato come tutti i tipi di sampling migliorino molto le prestazioni con il classificatore Random Forest, soprattutto con FS. Un altro aspetto interessante è come la tecnica dell'undersampling si comporti in modi profondamente diversi a seconda dei classificatori, passando dall'essere il migliore in IBk e Random Forest al peggiore in Naive Bayes.
Analizzando la metrica Kappa è chiaro che anche qui il noSampling ha buone prestazioni rispetto alle tre tecniche di sampling (tranne in Naive Bayes, in cui comunque tutte le tecniche risultano scadenti), con l'oversampling che però brilla particolarmente nel caso Random Forest senza FS. Infine l'analisi della ROCArea è probabilmente quella meno indicativa sotto l'aspetto del sampling con tutte le tecniche relativamente vicine, e con l'unica costante dell'undersampling leggermente meno performante.
Riassumendo le varie possibilità analizzate è evidente come non sempre l'utilizzo di tecniche di sampling porti a prestazioni migliori dei modelli predittivi, il noSampling è infatti mediamente tra i più performanti; ad ogni modo la tecnica che sembra soffrire meno in tutte le combinazioni è la SMOTE, che in molti casi permette anche di migliorare il dataset pre-sample. Va comunque evidenziato come la scarsità di release non consenta delle valutazioni più approfondite (con soli 6 passi del walk forward i quartili si sbilanciano molto facilmente).

Feature Selection

L'utilizzo di colori diversi per le diverse tecniche di sampling permette più facilmente di confrontare le prestazioni di queste ultime con o senza l'ausilio della tecnica di feature selection Best First; la presenza di un solo tipo di FS consente un raffronto più diretto, rispetto a quello avvenuto per il sampling. Seguendo lo stesso ordine dell'analisi precedente si può notare come per la metrica Precision il FS migliori le performance di due classificatori su tre (IBk e Random Forest) in praticamente tutti i tipi di sampling, con un effetto minore su undersampling.
Nella metrica Recall il comportamento è invece profondamente diverso, ed in questo caso la FS modifica le performance in maniera più dipendente dal sampling; il noSampling e SMOTE risultano costantemente peggiorati, l'oversampling migliorato, mentre l'undersampling vede alzarsi il primo quartile e scendere la mediana (non un trade-off auspicabile).
Nella metrica Kappa, trascurando ancora il classificatore Naive Bayes in cui tutti i risultati risultano pessimi, si nota un netto miglioramento con IBk, ed un

trade-off tra i vari quartili in Random Forest dipendente dal tipo di sampling. Infine la metrica ROCArea è in grado questa volta di fornirci qualche indicazione in più, specialmente con il classificatore IBk in cui il FS Best First sembra compiere un ottimo lavoro; negli altri due classificatori la differenza risulta meno evidente, ma il miglioramento è presente soprattutto sul primo quartile.
Riassumento l'analisi del FS, possiamo notare come anche per questa tecnica sia necessaria un'accurata valutazione dei vari casi prima di procedere ad utilizzarla; a differenza del sampling però è qui possibile fare una considerazione più forte: la tecnica di FS consente di diminuire drasticamente la varianza delle valutazioni, come è facilmente notabile dai box meno alti nella maggior parte delle combinazioni di classificatori e sampling. Tale comportamento è sicuramente da considerarsi un benefit, in quanto l'avvicinarsi del primo e terzo quartile alla mediana è un forte indice di prevedibilità e solidità della valutazione stessa.

3.3.2 Syncope

In questo secondo progetto sono presenti molte più release rispetto a Bookkeeper, ciononostante in molte di queste sono presenti pochissime istanze classificate come bugged; ciò comporta un forte sbilanciamento del dataset che anche le tecniche di sampling hanno difficoltà a colmare in maniera adeguata.
Un aspetto che è importante considerare per un raffronto più diretto con quanto discusso nell'analisi di Bookkeeper è la varianza delle prestazioni nei vari passi dell'approccio walk forward; a tale scopo si fa riferimento al seguente grafico:

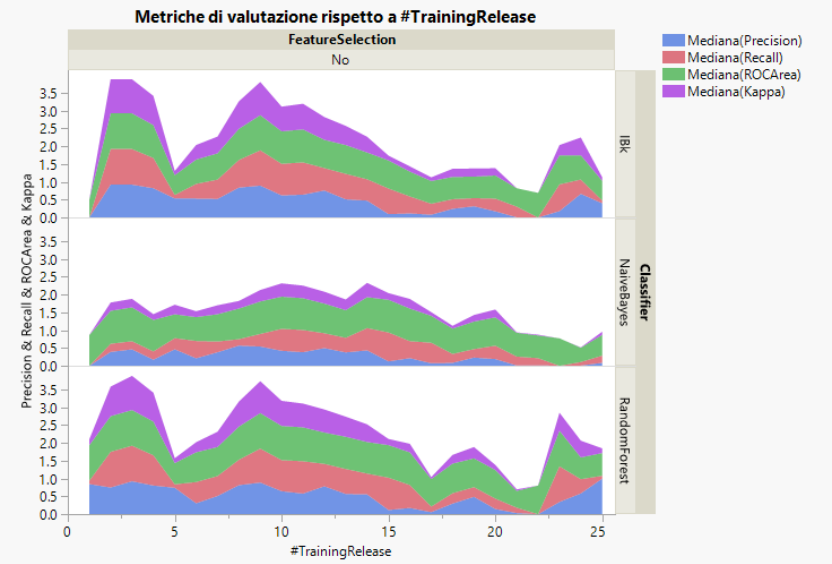

Figura 3.3: Metriche per il Walk Forward di Syncope

Allo scopo di analizzare le differenze sono stati esclusi i valori ottenuti tramite feature selection, che essendo troppo bassi avrebbero appiattito il grafico; la suddivisione in base al classificatore poi conferma come Naive Bayes sia quello con i valori più bassi dei tre. Gli altri due classificatori (IBk e Random Forest) hanno andamenti simili nei vari passi del walk forward, ed essendo quelli più performanti sono senz'altro i due su cui concentrare l'interesse. Diversi sono i passi in cui la performance sembra scendere rapidamente, in particolare al passo 5 e più avanti verso il 21-22; a riprova di quanto visto nel caso di Bookkeeper tali cali di prestazione sono spesso dovuti ad un dislivello marcato tra il numero di classi positive nel training e nel testing. Prendendo ad esempio i due passi appena identificati è infatti possibile notare come nel passo 5 le istanze buggy nel training set siano il 2% mentre nel testing set queste superino il 18%; nel passo 21 il rapporto tra i due valori è addirittura peggiore, con una percentuale di istanze positive nel training set superiore al 7% contro lo 0,6% del testing set. Con un numero così basso di istanze positive inoltre una differenza di poche unità stimate correttamente o meno fa una differenza enorme sulla proporzione finale, pertanto molte differenze risultano amplificate rispetto a quanto dovrebbero.
Vengono ora invece analizzati i dati delle quattro metriche di valutazione precedentemente usate per Bookkeeper:

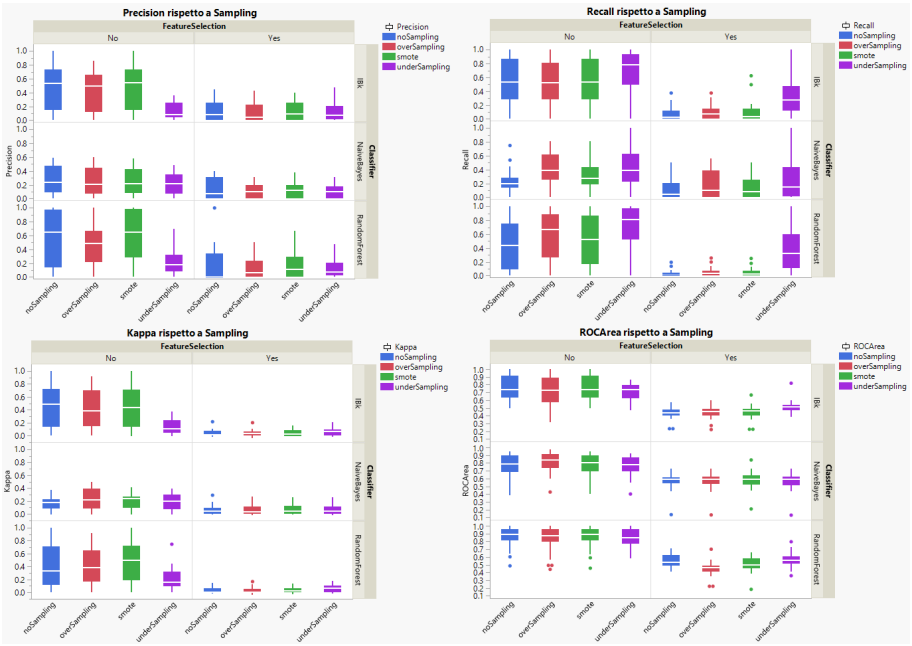

Figura 3.4: Metriche di valutazione in Syncope

Feature Selection

La prima proprietà che viene alla luce dall'analisi delle prestazioni dei modelli sul dataset di questo progetto è il comportamento estremamente negativo della tecnica di FS Best First; in praticamente tutte le condizioni peggiora drasticamente il risultato per tutte le metriche di valutazione. Anche quì si nota come tale tecnica riduca la variabilità della performance di predizione, ma in questo caso questo non può certo compensare un dislivello così netto. Unica nota da considerare è forse che in questo scenario la tecnica dell'undersampling è quella che soffre meno l'utilizzo della FS, avendo un peggioramento inferiore alle altre tecniche di sampling.

Sampling

E' facile notare come anche in questo progetto la tecnica dell'undersampling sia quella che si discosti di più dalle altre, con un comportamento buono per la metrica ROCArea (in cui ad onor del vero tutte le tecniche di sampling hanno valori simili), ed il risultato migliore per quanto riguarda la Recall; tale tecnica tuttavia risulta pessima secondo la metrica della Precision ed anche di Kappa. Una conferma rispetto al progetto precedente è la tecnica SMOTE che anche qui risulta costantemente tra le migliori nelle varie metriche di valutazione, migliorando in alcuni casi il noSampling. Infine la considerazione più importante per quanto riguarda l'oversampling è come tale tecnica fornisca ottime prestazioni sulla metrica Recall pur ottenendo un risultato peggiore nella Precision (rispetto a SMOTE e noSampling).

3.4 Considerazioni conclusive

Grazie all'analisi di due dataset profondamente diversi tra loro è possibile distinguere più chiaramente quelle che sono comportamenti costanti dai casi isolati. In entrambi i progetti il classificatore Naive Bayes risulta il meno performante secondo quasi tutte le metriche, questo comportamento potrebbe essere dovuto all'assunzione di indipendenza degli attributi su cui si basa tale classificatore e che qui è tutt'altro che tale; gli altri due forniscono invece risultati migliori, con Random Forest che sembra il classificatore più adatto. La tecnica di feature selection Best First consente in quasi ogni combinazione di ridurre la varianza tra i passi del walk forward, portando ad una maggiore prevedibilità della valutazione della stima; in alcuni casi comunque, come Syncope tale FS peggiora drasticamente i risultati, risulta pertanto preferibile, prima di fare uso di questa tecnica, utilizzare un set di validazione, in grado di indicare se in tale progetto questa possa essere di aiuto o meno. Le diverse tecniche di sampling portano anch'esse a risultati contrastanti, con dei casi di peggioramento delle prestazioni dei modelli secondo alcune metriche di valutazione; l'undersampling si è dimostrato essere spesso il meno performante, mentre lo SMOTE raramente si è comportato in maniera peggiore del noSampling ed è stato in grado molte volte di portare ad un sostanziale incremento di prestazioni.

Bibliografia

[1] *Github Parquet*. URL: https://github.com/apache/parquet-mr.

[2] *Jira Parquet*. URL: https://issues.apache.org/jira/projects/PARQUET.

[3] *Parquet*. URL: https://parquet.apache.org/.

[4] *Bookkeeper*. URL: https://bookkeeper.apache.org/.

[5] *Syncope*. URL: https://syncope.apache.org/.

[6] *Jira Bookkeeper*. URL: https://issues.apache.org/jira/projects/BOOKKEEPER.

[7] *Jira Syncope*. URL: https://issues.apache.org/jira/projects/SYNCOPE.

[8] *Github Bookkeeper*. URL: https://github.com/apache/bookkeeper.

[9] *Github Syncope*. URL: https://github.com/apache/syncope.

[10] *A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction*. URL: https://ieeexplore.ieee.org/document/4814129.