01/07/2020

Quality and Software Engineering:

Uno studio sulla localizzazione e sulla predizione di bugs in progetti open-source Apache

Domenico Verde | 0289890

Il presente documento descrive il lavoro fatto per il completamento delle Milestone assegnate dal prof. Falessi D. durante il corso di Ingegneria del Software 2 (9 CFU) presso l'Università degli Studi di Roma Tor Vergata, anno accademico 2019 – 2020.

Il progetto del corso si compone di due distinte deliverables, che indicheremo con deliverable 1 e 2.

La Deliverable 1 è composta da un'unica milestone; mentre la deliverable 2, a sua volta, si compone di tre milestone.

Deliverable 1 – Software Analytics by Jira & Git

Introduzione

Per software analytics, s'intende, quell'insieme di tecniche, metodologie e risorse utilizzati per ottenere informazioni su un sistema software target, al fine di migliorarne la qualità, ottimizzare il processo produttivo oppure per trarre conclusioni su eventuali decisioni future da prendere. La deliverable 1 consiste, sostanzialmente, nel fare software analytics su un progetto open-source Apache, utilizzando i tools Jira e Git. Il progetto assegnatomi è risultato "VCL": l'obbiettivo è quindi quello di realizzare un process control chart per il progetto in questione, avente un asse delle ascisse (asse x) rappresentato mediante delle date in mesi ed un asse delle ordinate (asse y) che indica quanti bug sono stati fixati in un determinato mese. L'intervallo temporale scelto varia dalla data della prima commit alla data dell'ultima commit (incluse) effettuate nella repository Github del progetto. In addizione al lavoro da svolgere, si è preferito utilizzare i tool Git e Jira anche da interfaccia grafica, al fine di verificare i risultati forniti dalle API.

Design

Phase 1 (Querying the Jira DB)

In questa prima fase di progettazione, dopo aver riflettuto sul lavoro da farsi, si è deciso di analizzare la classe Java "RetrieveTicketsID.java" fornita dal professore. Tale classe utilizza le JIRA Rest API per prima ricavare e poi stampare a schermo tutti i Bug ID dei fixed bugs del progetto Apache QPID, bypassando di fatto il limite dei mille risultati ottenibili in uno stesso momento gratuitamente. Tale classe è stata successivamente modificata e poi adatta al progetto in esame per recuperare dal database di Jira, tutti i fixed bugs relativi ad esso. In particolare, sempre tramite le JIRA Rest API è stato possibile recuperare tutti i 434 fixed bugs, rappresentati mediante JSON Object e JSON Array. Dopodichè è stata effettuata una piccola verifica utilizzando il browser, al fine di constatare la veridicità dei dati ottenuti mediante le API, come è possibile notare nella figura 1.

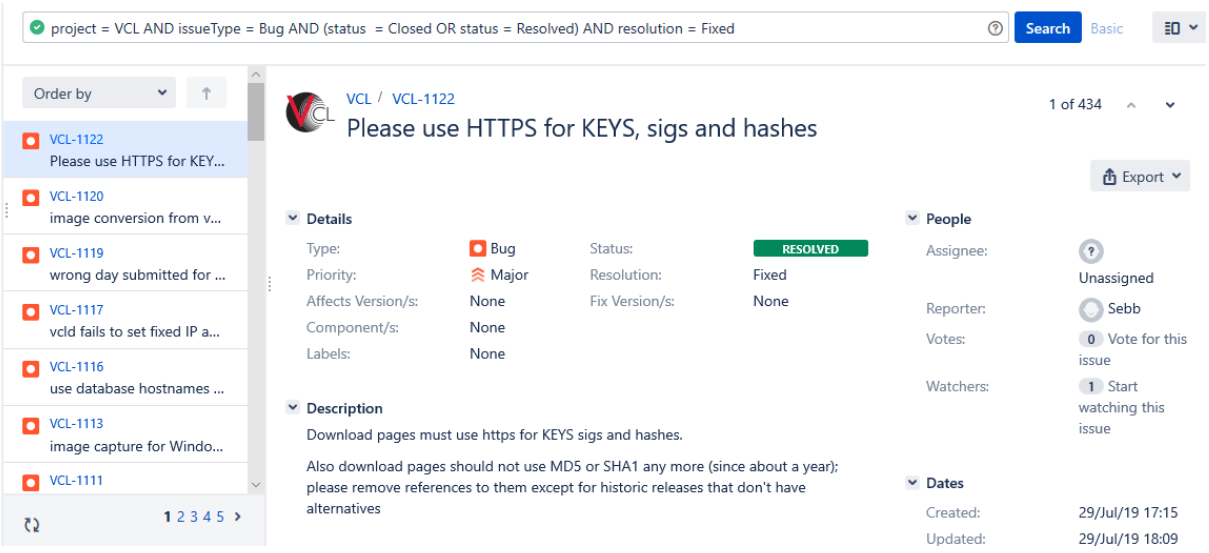


Figura 1. Screenshot della query in linguaggio JQL (Jira Query Language) eseguita. In alto a destra vengono indicati i 434 risultati.

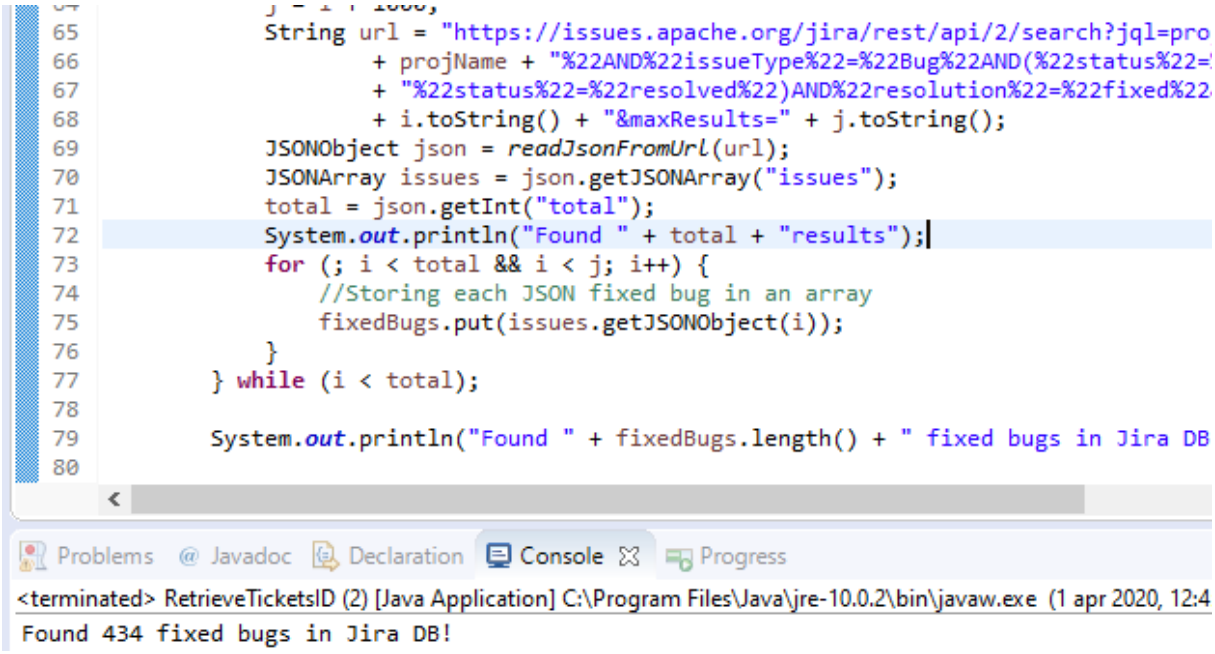


Figura 2. Screenshot di una porzione del codice Java utilizzato. Notare che la query effettuata permette di ricavare un JSON Object ed un JSON Array che consentono di ottenere ulteriori informazioni sui risultati. Il programma stampa i risultati ottenuti: 434.

Phase 2 (Commit Github)

Successivamente, tramite l'ausilio delle API JGIT, è stato possibile clonare la repository Github del progetto VCL (effettuando, di fatto, l'equivalente di un comando git clone da terminale). Dopodichè un log(), ha permesso di ricavare tutte le commit del progetto dato; iterando tra le commit, sono state ricavate le date in cui sono state effettuate la prima e l'ultima commit ed il numero di commit totali del progetto. Le date, essendo utili per determinare l'intervallo temporale nel grafico, vengono poi scritte in un file chiamato VCL.csv; tale file, di fondamentale importanza per generare lo chart finale, verrà esportato al termine di questa fase e conterrà due colonne: nella prima verrà indicato l'id del bug e nella seconda la relativa fixed date. Per fixed date di un bug s'intende la data dell'ultima commit (la più recente in termini di tempo) avente nel messaggio l'id del bug in questione. Quindi, per ogni bug presente in Jira, si è cercato il proprio ID tra tutte le commit, si è selezionata l'ultima e la si è scritta nel file CSV. Infine, sono stati eseguiti dei controlli a campione utillizzando il tool Git da terminale, per verificare quanto fatto tramite le API. Un dettaglio di notevole importanza rilevato in questa fase è il fatto che esistono alcuni ticket id (in Jira) che non possiedono nessuna commit associata (in Github) e ciò può essere dovuto a varie cause, ad esempio errori dei programmatori. Tale fenomeno è noto come Linkage e, per questo motivo, in genere la percentuale di linkage di ogni progetto è minore del 100%.

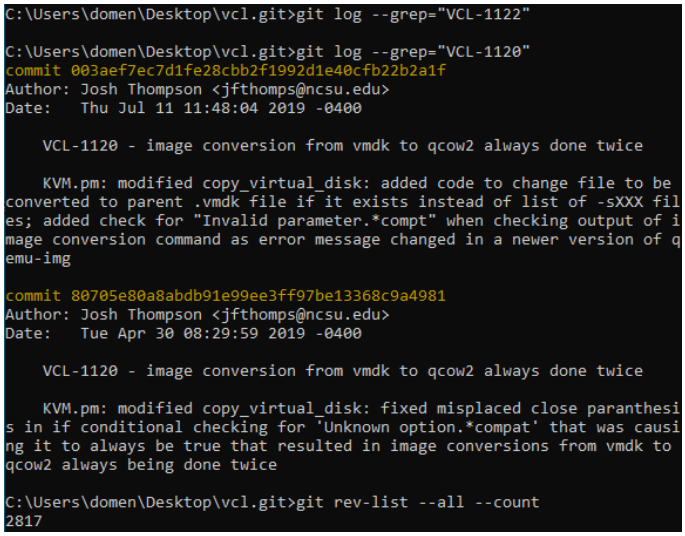


Figura 3. Esempio di utilizzo del tool Git da terminale. L'opzione –grep viene utilizzata per ricercare il bug id all'interno dei messaggi. Notare che "Vcl-1122" non presenta nessuna relativa commit anche se presente come fixed bug in Jira (pertanto esso verrà scartato, ovvero non sarà presente nel file .csv) e che sono presenti 2817 commit totali per il progetto in questione.

```
113             System.out.println("Found " + results + " commits in Git!");
114             csvWriter.append("VCL-first;" + (firstDate.getMonth() + 1) +
115                     "/" + (firstDate.getYear() + 1900) + "\n");
116             csvWriter.append("VCL-last;" + (lastDate.getMonth() + 1) +
117                     "/" + (lastDate.getYear() + 1900) + "\n");
118
119             //for each fixed bug, take the ticket id
120             for (int z=0; z < fixedBugs.length(); z++) {
121                 String key = fixedBugs.getJSONObject(z).get("key").toString();
122                 //do git log --grep=ticketID -> search for all commits containing that ticket id
123                 commits = git.log().all().call();
```

Problems @ Javadoc Declaration Console Progress

```
RetrieveTicketsID (2) [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (2 apr 2020, 11:47:43)
Found 2817 commits in Git!
```

Figura 4. Uno screenshot del codice Java utilizzato. Notare che è stato trovato il medesimo numero di commit del metodo precedente.

	A	B
1	Ticket ID	Fix Date
2	VCL-first	nov-08
3	VCL-last	dic-19
4	VCL-1120	lug-19
5	VCL-1119	apr-19

Figura 5. Screenshot che mostra un pezzo del file .csv esportato. Notare che i primi due risultati, indicati con "VCL-first" e "VCL-last" rappresentano la data (il mese) della prima e dell'ultima commit. Notare, inoltre, che il bug vcl-1120 ha come fixed date quella attesa, come è possibile verificare nella figura 3. Il file CSV contiene 403 righe.

Phase 3 (Manipulating data in Excel)

In questa terza ed ultima fase è stato importato il file CSV in Excel e, tramite l'ausilio di alcune funzioni come "CONTA.SE()" è risultato possibile contare, per ogni mese compreso tra quello in cui è stata effettuata la prima commit e quello in cui è stata effettuata l'ultima, quanti bugs sono stati fixati in tale mese. Inoltre, tramite l'ausilio di ulteriori funzioni come "MEDIA()" e "DEV.STANDARD()" si è potuto calcolare facilmente l'Upper ed il Lower Control Limit tramite le formule: mean +/- 3*dev.st. Infine, si è preferito calcolare la percentuale di linkage effettuando il rapporto tra numero di ticket trovati in Jira e numero di bug con fixed date.

Asse X	nov-08	dic-08	gen-09	feb-09	mar-09	apr-09	mag-09	giu-09
Asse Y	0	0	0	0	0	3	2	8

f_x =CONTA.SE(C4:C403;C405)

Figura 6. La tabella dati utilizzata per creare il grafico in Excel. Notare che l'asse x parte proprio dalla data di "VCL-first".

Fase 4 (Github, TravisCI and SonarCloud)

Il codice è stato caricato, inoltre, all'interno di una repository Github, integrato con TravisCI ed analizzato mediante SonarCloud, i cui links al progetto sono riportati nella sezione Links. Per poter usare le API JGit, molto potenti e ben documentate, si è dovuto necessariamente utilizzare Maven per gestire le dipendenze: ciò potrebbe sembrare, a prima vista, un conflitto con ANT; tuttavia, a partire dalle ultime versioni ANT supporta pienamente questa modalità di progettazione, in effetti TravisCI riesce ad analizzare il codice senza problemi (configurato tramite il file .travis.yml) stesso discorso per Sonar Cloud (configurato tramite il file sonar-project.properties). Infine, analizzando i risultati mostrati da Sonar, è stato azzerato il debito tecnico, utilizzando un logger al posto della funzione

println(), eliminando tutti i metodi deprecati e adottando misure per migliorare la leggibilità e la manutenibilità del codice.

```
<path id="Milestone1.classpath">
    <pathelement location="target/classes"/>
    <pathelement location="others/java-json.jar"/>
    <path refid="Maven Dependencies.libraryclasspath"/>
</path>
```

Figura 7. Una porzione di codice presente nel file ant-build.xml: in questo modo esso supporta la gestione delle dipendenze di Maven.

Risultati: Analisi e Considerazioni

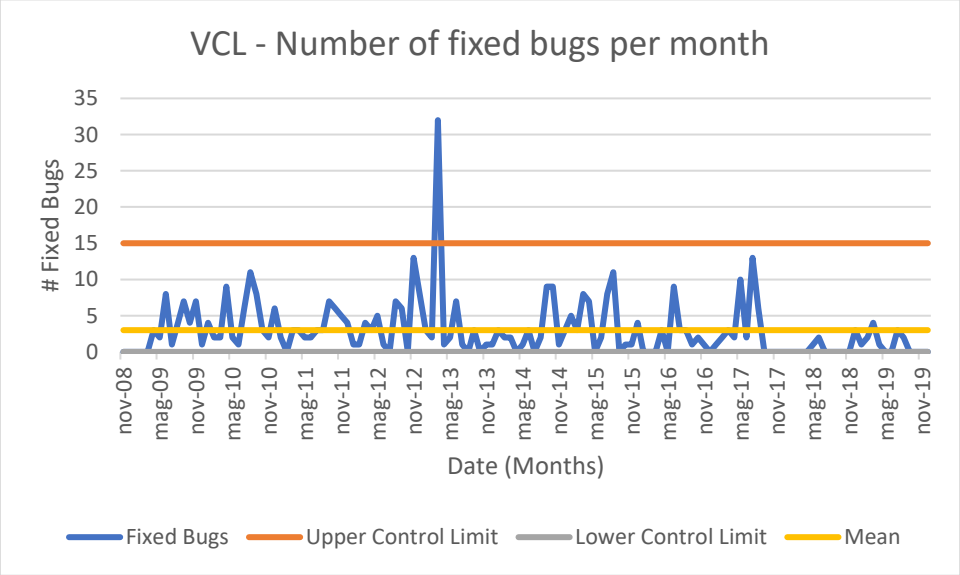


Figura 6. Il process Control Chart del progetto Apache VCL.

La figura 6 mostra il Process Control Chart ottenuto: si può notare facilmente che possiede un punto di instabilità (che è anche un punto di massimo globale) raggiunto nel mese di marzo 2013, in cui sono stati fixati 32 bugs. Tale valore, essendo ampiamente al di fuori dei bounds calcolati, mostra un chiaro errore di process management. Una possibile soluzione a questo problema potrebbe essere quella di intensificare l'attività di testing, al fine di scovare e quindi risolvere più bugs possibili, in modo da alzare sia la media che la deviazione standard per poter far rientrare quel punto all'interno dei bounds di controllo. Per completezza, si riporta la media, che è attualmente di 3 bugs fixati al mese e la percentuale di linkage del progetto, che è del 92, 59%. Una piccola nota finale: i valori di Upper e Lower Control Limit calcolati in precedenza sono rispettivamente 14,877 e -8,907. Essendo l'entità misurabile intera e sicuramente non negativa si è preferito utilizzare i valori 0 e 15. Inoltre, è da evidenziare il fatto che sono presenti degli zeri all'inizio e alla fine del grafico, che indicano che nei mesi iniziali e finali non sono stati fixati bugs e ciò può essere dovuto a diversi motivi: ad esempio, a riguardo degli zeri iniziali, è ragionevole supporre che al rilascio del progetto, esso non avesse/avesse pochi collaboratori e/o che essi non abbiano effettuato un'attività di testing adeguata. In ogni caso, comunque, tali zeri non influenzano per nulla le statistiche ed in effetti, eliminando gli zeri iniziali e finali né la media né la deviazione standard vengono alterate e, di conseguenza, nemmeno i bounds.

Deliverable 2 – Datasets

Introduzione

La Deliverable 2, invece, consiste nel realizzare dei datasets (intesi come file .csv) che potranno poi essere utilizzati da algoritmi di machine learning per effettuare stime. I dataset da realizzare sono 2, relativi ad altrettanti progetti Apache: nel caso in questione saranno "Bookkeeper" e "Syncope". È utile specificare che tutte le informazioni sui progetti sono state ottenute da Jira e dai repositories Github di Apache Gitbox. In particolare tali dataset, scelte delle metriche di riferimento opportune, conterranno una misura delle caratteristiche delle classi presenti nel codice sorgente dei progetti citati in precedenza ed infatti una buona parte del lavoro svolto è consistito nel capire come e nel misurare tali caratteristiche. In addizione al lavoro da svolgere, si è preferito fornire anche un'implementazione dell'algoritmo simple per la stima della buggyness delle classi ed includere una sezione nel report in cui vengono effettuate statistiche (sulla consistenza dei bugs, affidabilità dei dati di Jira) su entrambi i progetti in esame.

Designing The Datasets

Columns 1,2: Version (or Release) and Filenames

Tramite il codice fornito dal docente, sono state selezionate tutte le release presenti in Jira ed aventi una release date, avendone cura di memorizzare il nome e la data di rilascio, ed ordinandole in base a quest'ultima (ordine crescente). Successivamente, leggendo la documentazione di Git, si è notata l'esistenza delle tags, una funzionalità di Git che permette di marchiare un punto temporale specifico in una repository; spesso esse vengono utilizzate, appunto, per taggare le release: si è ricercato quindi in Github, per ogni release presente in Jira, se esistevano delle tags associate al nome di tali release e così si è potuto verificare quindi che l'intuizione avuta era corretta, per questo motivo si è ritenuto necessario creare una nuova classe "Release" per memorizzare le informazioni ricavate dai due tool utilizzati. Quindi, per ogni release avente una release date in Jira ed avente una relativa tag in Git è stato creato un nuovo oggetto Release ed è stato aggiunto ad un array che conterrà tutte le release da esaminare. Ovviamente, tutte le release non aventi una data in Jira o una tag in Git sono state scartate e non appariranno nel dataset; inoltre, si è preferito non includere anche le eventuali release candidates (rilasciate solo internamente ed utili solamente per scopi di test – non sono presenti in tutti i progetti, in particolare non sono presenti né in BOOKKEEPER né in SYNCOPE, ma in ZOOKEEPER, ad esempio, sì). A questo punto, tramite alcuni comandi come peel(), log() messi a disposizione dalle API JGIT sono state ricavate tutte le commit associate ad ogni release. Ogni oggetto Commit (più precisamente, RevCommit) in JGIT punta ad un tree contenente tutti i file vissuti in quella release: da essi sono stati selezionati solamente tutti i file .java. Iterando sui file è possibile computare tutte le metriche di cui si ha bisogno. Infine, è necessario precisare che non sono stati considerati tutti i file presenti nei vari branch e che perciò non fanno parte della release finale (il programma, al termine, stampa in particolare quanti).

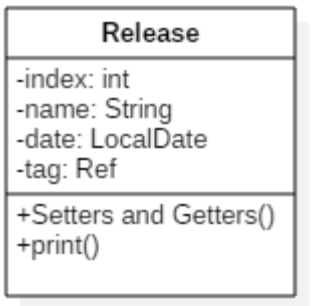


Figura 1. La classe Release creata ad-hoc: essa contiene un index che identifica univocamente la release, il nome della release, la data di rilascio e la relativa tag in Git. In particolare, notare che la tag è rappresentata da un oggetto Ref in JGit; inoltre, la classe è dotata di un metodo print() utile nel debugging. Infine, una piccola precisazione sull'attributo index: esso è un identificatore temporalmente della release nella storia del progetto: se la release è stata la prima ad essere pubblicata, allora il suo index sarà 1, se è stata la seconda allora 2, ecc.

Column 3: Lines of Code (LOC)

Nella parte del corso svolta dal prof. Cantone, si è analizzato oltre alle numerose metriche utilizzate per misurare il software, anche le diverse tipologie di SLOC (Source Lines Of Code) e la loro importanza: esempi calzanti sono le LLOC (Logical Lines of Code), ELOC (Executable Lines of Code) ed infine le più classiche LOC (Physical Lines of Code). Nel caso dei progetti in questione, si è preferito (per semplicità) includere nel conteggio tutte le linee di codice presenti in un file, comprendendo quindi anche commenti (CL – Commented Lines) e linee vuote (BL – Blank Lines).

Columns 4 - 11: Other Metrics

Inoltre, al fine di effettuare una stima sufficientemente accurata, è necessario includere nel dataset anche ulteriori metriche, in modo tale che esso abbia almeno 12 colonne in tutto; nella tabella seguente possiamo notare evidenziate le metriche scelte.

Metric	Description
Size	Lines of code(LOC)
LOC Touched	Sum over revisions of LOC added + deleted
NR	Number of revisions
Nfix	Number of bug fixes
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
Max Churn	Maximum churn over revisions
Average Churn	Average churn over revisions
Change Set Size	Number of files committed together
Max Change Set	Maximum change set size over revisions
Average Change Set	Average change set size over revisions
Age	Age of Release
Weighted Age	Age of Release weighted by LOC touched

Figura 2. La Tabella che mostra le possibili metriche utilizzabili nel dataset: sono evidenziate in giallo quelle utilizzate. È bene precisare che tutte queste metriche si riferiscono ad un dato file.

In particolare, per quanto riguarda il metodo di conteggio delle LOC, il quale è stato descritto teoricamente nel paragrafo precedente, consiste in pratica nel leggere il file per righe, ovvero nel contare il numero di carattere "\n" presenti nel file target; ora, invece, si descriveranno le procedure di conteggio delle restanti metriche, seguendo l'ordine con cui sono rappresentate nella tabella: la prima metrica da approfondire risulta, quindi, Number of Revisions: per poter calcolare il numero di revisioni di un dato file, si è dapprima effettuato un log().addPath(filename), ottenendo così tutte le commit associate ad un determinato file. Contare il numero di queste commit equivale a contare il numero di revisioni a cui è stato sottoposto il file. Per ottimizzare il codice e quindi velocizzare l'esecuzione del programma, allo stesso tempo (cioè iterando sulle stesse commit) si è potuto

computare anche alcune delle restanti metriche. In effetti, per contare Number of Authors si sono dapprima ricavati gli autori delle precedenti commit e poi messo tali autori in una struttura dati di tipo set, per eliminare i duplicati, ottenendo di fatto il numero di autori distinti tramite il metodo set.size(). Sempre allo stesso tempo, inoltre, è stato computato il valore delle metriche LOC Added e Average LOC Added. Per fortuna le JGIT Api mettono a disposizione classi di notevole interesse quali DiffFormatter, Edit, DiffEntry. In particolare, tramite il metodo scan(commit1, commit2) di DiffFormatter, date due commit è possibile ricavare una lista di oggetti di tipo DiffEntry e Edit che rappresentano, in pratica, tutte le differenze tra queste due commit: le DiffEntry si riferiscono ai file modificati, creati o cancellati, mentre le Edit si riferiscono proprio alle linee di codice che sono state aggiunte, modificate o eliminate; creando perciò un oggetto DiffFormatter, impostando come filtro il file che vogliamo misurare con il metodo setPathFilter(), effettuando uno scan() tra due commit ed iterando sulle Edit è possibile osservare quante e quali linee di codice sono state aggiunte, modificate, e/o eliminate. Considerando solo le linee aggiunte, è stato possibile computare il valore della metrica LOC Added. È evidente ora che modificando leggermente questa procedura è possibile computare il valore di altre metriche quali LOC Touched e Churn. Per quanto riguarda la metrica AVG LOC added, invece, è bastato dividere il valore di LOC Added per il numero totale di revisioni del file (NR). Per quanto riguarda il valore del Changeset, invece, la procedura di conteggio utilizzata è molto simile a quella illustrata poco fa: viene creato un oggetto DiffFormatter, non viene questa volta impostato nessun filtro e viene invocato il metodo scan() ottenendo una lista di oggetti DiffEntry, che ricordiamo rappresentano tutti i file differenti in queste due commit; tramite il metodo size(), è possibile computare il changeset, ovvero il numero di file committati nella seconda commit. Ovviamente, per calcolare il ChangeSet totale, ovvero la metrica a cui ci si sta riferendo, occorre ripetere questa procedura per tutte le commit riferite ad un dato file. Per calcolare AVG Changet è sufficiente dividere tale valore diviso il numero di revisioni, mentre per calcolare MAX ChangeSet è stata inizializzata una variabile di tipo int nel codice ed, ogni qualvolta veniva eseguita questa procedura, veniva confrontato il changeset attuale con quello memorizzato in tale variabile: essa veniva aggiornata solo nel caso in cui il changeset diventava più grande di quello memorizzato. Infine, l'ultima metrica da analizzare risulta Age: è semplicemente la differenza tra la data della prima commit associata al file e la data della commit associata al rilascio della release; il valore di tale metrica è espresso in settimane.

Column 12: Buggyness

Per quanto riguarda quest'ultima colonna del dataset, infine, si ricorda che l'informazione riguardo la buggyness di una classe può essere:

- Ricavata da Jira e Github;
- Stimata con un algoritmo (es. Blame, Simple, Proportion), se non può essere ricavata.

In base a quanto detto a lezione e riportato nelle slides, si utilizzerà ove possibile le informazioni presenti in Jira e Github, diversamente si utilizzerà l'algoritmo Proportion. Si è proceduto eseguendo una query tramite il metodo getJiraBugs(); tale metodo oltre a ricavare informazioni su tutti i fixed presenti in Jira, li ordina secondo la data di scoperta e consente di bypassare il solito limite di mille risultatati ottenibili in maniera gratuita tramite le Jira Rest Api.

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
        + projName + "%22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR"
        + "%22status%22=%22resolved%22)AND%22resolution%22=%22fixed"
        + "%22&fields=key,versions,fixVersions,created&startAt="
        + i.toString() + "&maxResults=" + j.toString();
```

Figura 1. Screenshot della query in linguaggio JQL (Jira Query Language) eseguita. Da notare che sono stati ottenuti i campi key (contenente l'ID del bug), versions (contenente le AVs), fixVersions (contenente, appunto, la FV) e created (che consente di ricavare la data in cui il bug è stato scoperto, ovvero la data di apertura del ticket, e quindi permette di calcolare la OV).

I bugs ricavati, rappresentati in formato JSON, sono stati dapprima convertiti in oggetti Bugs (istanze di una classe creata da me ad-hoc per ridurre la complessità del progetto) e poi filtrati: da essi sono

stati scartati: tutti i bugs not-post-release (ovvero tutti quelli scoperti e fixati in una stessa release), poiché considerando tali casi non è possibile determinare se una classe in una release è buggy oppure no; tutti quelli non aventi una correlata fix commit in Github, dato che in questi casi non è possibile ricavare i file affetti dal bug in questione. Inoltre, per mantenere una certa consistenza dei dati, ho preferito escludere anche tutti i bug non aventi una FV in Jira e quindi non ricavarla in Git osservando la data della fix commit.

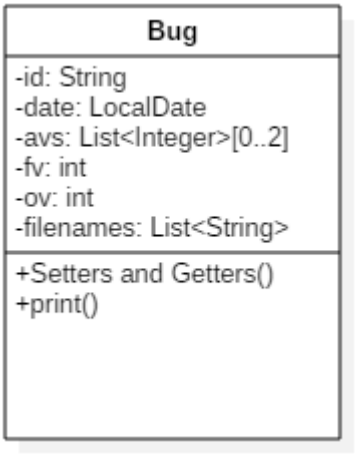


Figura 3. La classe Bug: essa contiene in ordine l'identificativo del bug, le affected versions, l'opening version, la fixed version ed infine i files affetti da quel bug. I primi quattro attributi vengono ricavati direttamente da Jira mediante la query vista in precedenza, mentre l'ultimo viene ricavato dalla fix commit di Github. Da notare che le AVs sono rappresentate da una lista di interi di al massimo due elementi: se tale lista non possiede alcun elemento, allora le AVs andranno stimate; se tale lista possiede un solo elemento, allora esso rappresenta la IV – Initial Version; se essa possiede due elementi allora le AVs saranno tutte le versioni con indice compreso tra questi due valori.

12.1: A simple Approach

Una piccola nota, nel codice è presente anche un'implementazione dell'algoritmo "Simple" che è servita in fase di progettazione e che può essere utilizzata, appunto, per stimare la buggyness o per visualizzare la differenza tra le stime restituite tra i due algoritmi. Nel codice, è possibile quindi cambiare algoritmo di stima, modificando il contenuto della variabile-stringa algorithm.

12.2: A Proprortion approach

È noto che da numerosi studi è emerso che Proportion risulta in media molto più preciso di Blame o Simple. Esso tuttavia presuppone che i bugs considerati abbiano uno stable-lyfecicle: in pratica, spesso non è così e lo è stato anche nei progetti considerati. Inoltre, per poter applicare Proportion, si suppone che le informazioni prese da Jira siano affidabili e consistenti, e ciò nel prossimo paragrafo verrà discusso ed analizzato quantitativamente per i progetti considerati. Un altro problema pratico manifestatosi è il fatto che alcuni bugs presentano una OV coincidente con la FV. Nella stima, si è preferito non considerarli, poiché in tali casi il valore di P divergerebbe, come possiamo notare dalla formula di calcolo:

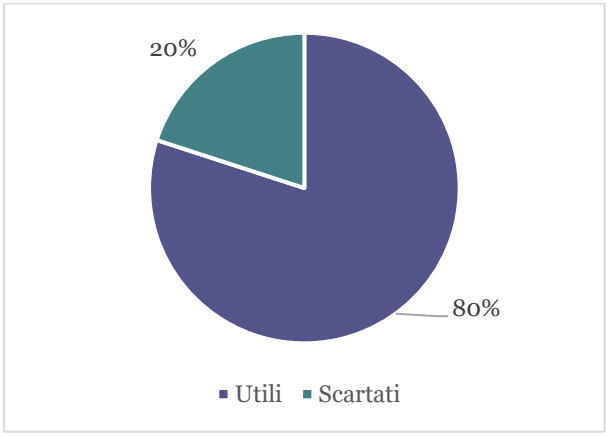
$$P = \frac{(FV - IV)}{(FV - OV)}$$

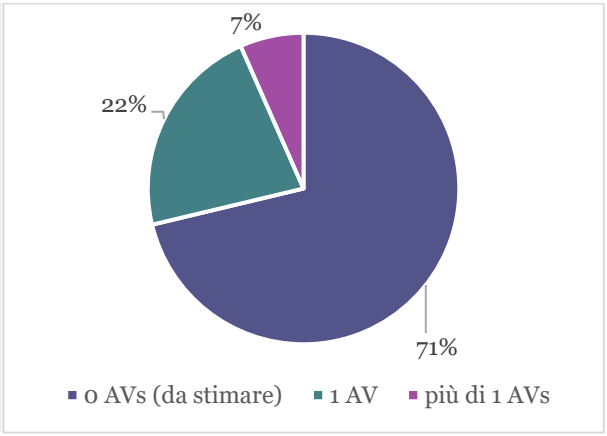
Infine, è utile specificare che la versione di Proportion implementata è "Proportion_Increment" che, a mio parere, risulta un ottimo compromesso tra la qualità della stima dell'informazione (dato che tiene conto dell'informazione passata di ogni progetto) e la semplicità con cui si effettua tale stima.

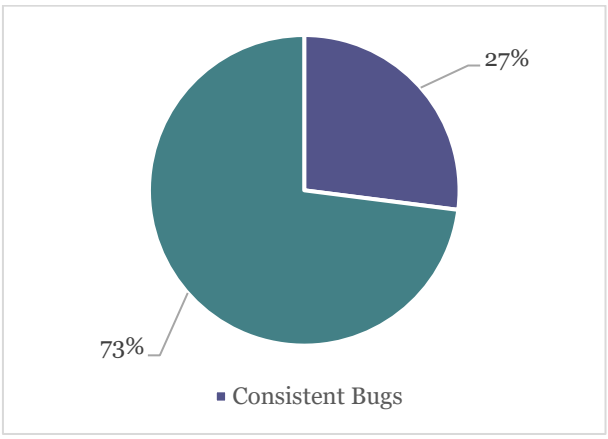
12.1 - Some Statistics about Projects

Il progetto BOOKKEEPER, comune a tutti gli studenti del corso, presenta alcune peculiarità:

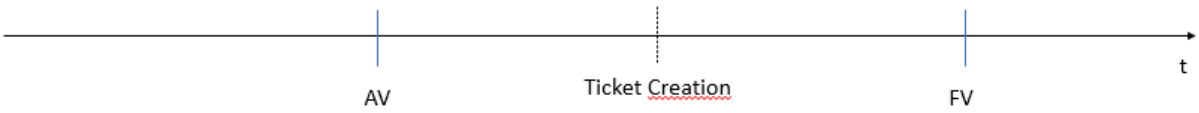
- Dei 435 bugs trovati in Jira, 12 non sono presenti in Github, 20 non possiedono alcuna FV, e 55 sono not-post-release: in totale ne risultano utili 348;
- Dei bugs utili, 248 non presentano AVs, 77 ne presentano solo una e 23 ne possiedono più di una;
- Solo 94 bugs sono consistenti, ovvero tutte le AVs sono precedenti alla OV;







Innanzitutto, è bene notare che è possibile trarre informazioni e quindi effettuare statistiche su un buon numero di risultati (più del 80 %); sorprendente è la percentuale di linkage del progetto che è del 97,25 %. Poi, si osserva che un numero molto alto di bug non presenta affected versions (circa il 71%). Inoltre, circa il 4% dei bug non possiede una Fixed Version in Jira: ecco la ragione per cui (essendo tale numero irrisorio) si è preferito scartare tali bugs senza ricavarla da Github. Infine, è bene concludere osservando che, essendo la maggior parte dei bugs nella situazione seguente:

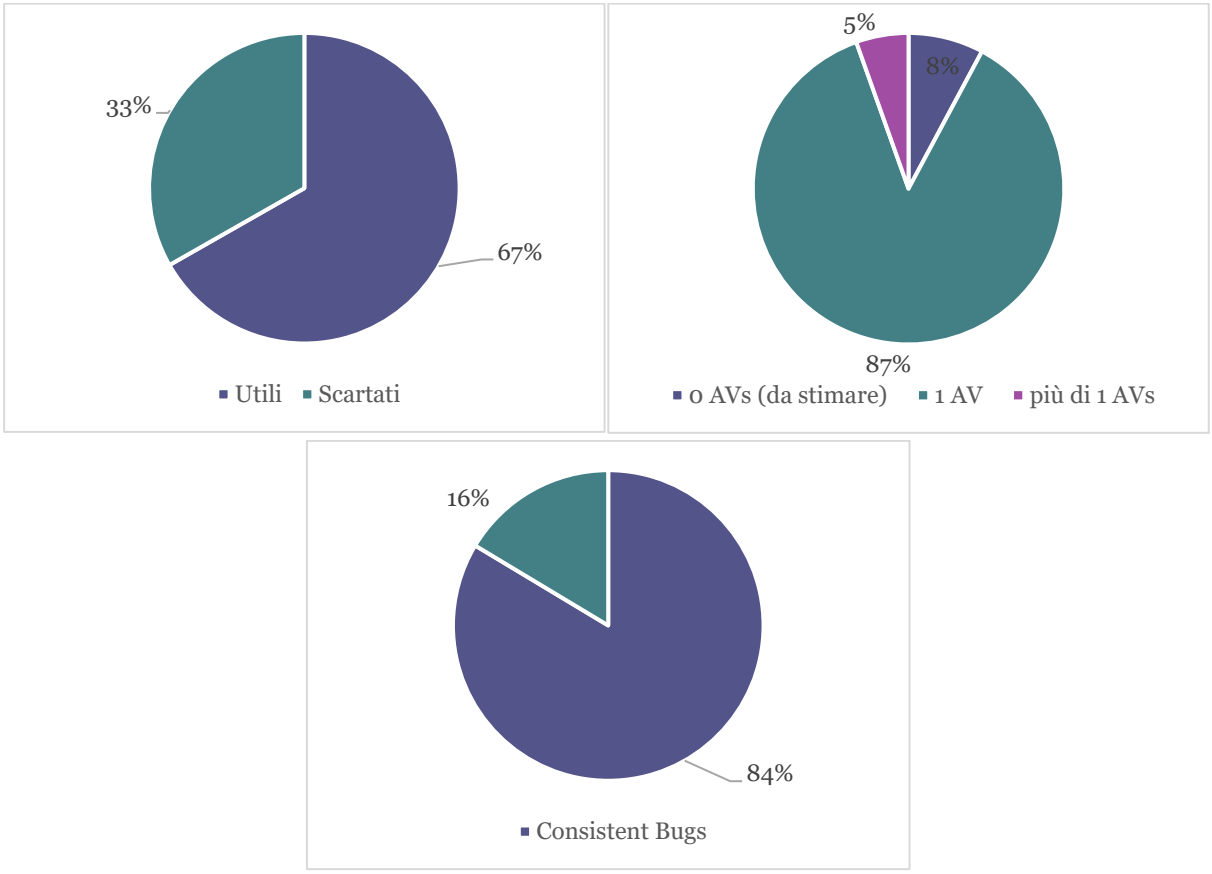


Anche l'algoritmo Simple potrebbe dare buoni risultati, ragione per cui si è preferito includerlo nel codice. Tramite Simple si è osservato, infatti, che la distanza media tra FV e AV è di circa 1,69 release.

Analizzando invece l'altro progetto assegnatomi, SYNCOPE, le considerazioni fatte sono tutt'altre:

- Dei 713 fixed bugs presenti in Jira, 689 sono stati matchati in Github e 213 sono stati scartati per gli stessi motivi elencati nel paragrafo precedente: 476 sono quindi risultati utili;

- Di questi ultimi, 33 bugs non presentano AV, 77 ne presentano più di una e 366 ne presentano una sola;
- 398 bugs sono consistenti;



Qui, è facile notare che il numero di fixed bugs a nostra disposizione è inferiore in percentuale rispetto al progetto precedente; sostanzialmente identica è la percentuale di linkage del progetto (circa il 96,6%). Anche in questo caso, ma in modo molto più evidente i bugs sono nella situazione riportata in precedenza, ovvero quasi la totalità dei bugs utili presenta una sola AV (più del 87%). Inoltre, tutti i bugs considerati presentano una FV in Jira. Infine, si osserva che la qualità dell'informazione sui bugs ottenibile da questo progetto è migliore, nel senso che in questo progetto è presente un numero molto elevato di bugs consistenti.

Deliverable 2 – Machine Learning

Introduzione

In quest'ultima parte del report verranno analizzati i dati riportati nei datasets per dapprima creare, poi validare ed infine valutare la precisione dei modelli di machine learning, utilizzando le Weka API. In particolare, verranno utilizzate tutte le metriche del dataset per stimare la difettività di una data classe, ovvero stimare se essa contiene oppure no almeno un bug, avendo cura di comparare diversi classificatori e diverse tecniche di balancing e di feature selection. Oltre a determinare se tali tecniche di sampling e/o feature selection aumentano l'accuratezza dei classificatori, si è preferito valutare il lavoro svolto anche a grana molto più fine, determinando quale tra i classificatori considerati risulta il migliore in media, quale risulta il migliore per ogni progetto e quale risulta il migliore per ciascuna metrica scelta.

Snoring and Evaluation Technique

Numerosi studi scientifici evidenziano il fatto che, soprattutto nelle ultime versioni di un progetto software, sono presenti degli sleeping bugs, ovvero bugs i quali verranno scoperti ed eventualmente corretti solo in release successive a quella considerata; per questo motivo, al tempo presente, si potrebbe erroneamente classificare come non buggy una classe che, in futuro, risulterebbe buggy, ottenendo di fatto un falso negativo (FN). Ciò in pratica viene modellato come un'introduzione di rumore nel dataset, rumore che impatta negativamente sull'accuratezza del modello di machine learning considerato. Al fine di ridurre tale rumore, chiamato snoring noise, si è preferito eliminare il 50% delle ultime release dal dataset: così facendo, il missing rate secondo diversi studi empirici dovrebbe ridursi a circa il 10%. Esso è definito come il rapporto tra FN e (TP+FN).

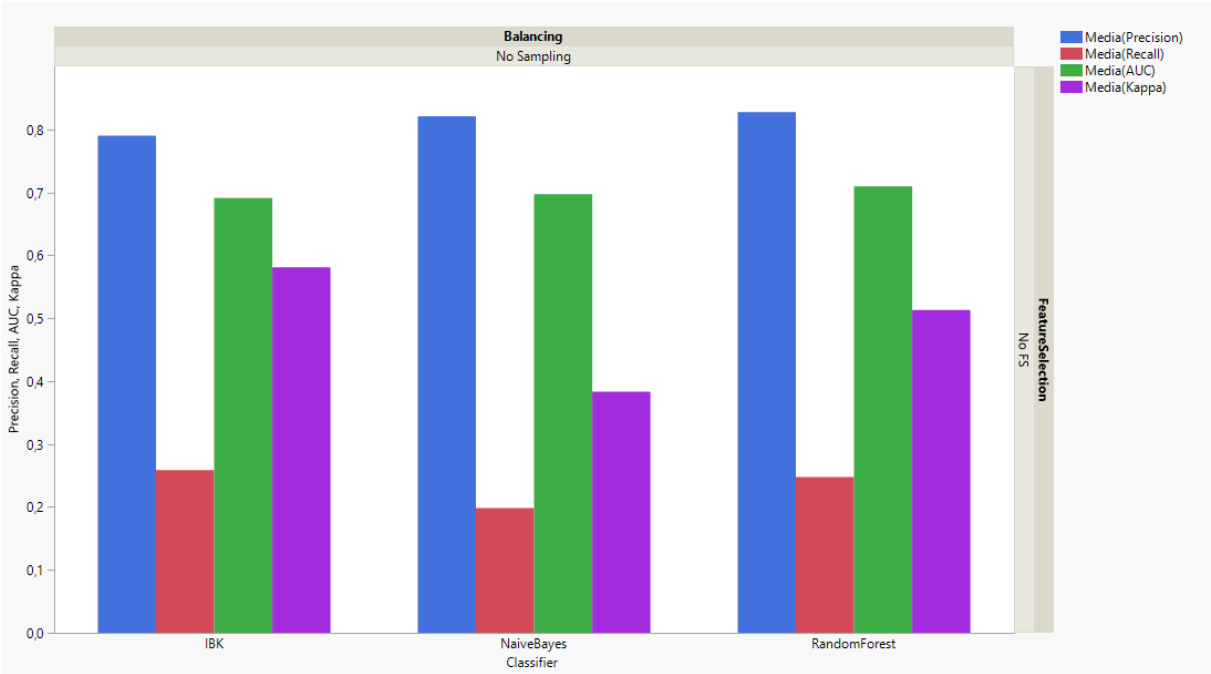
Per quanto riguarda la validazione dei modelli creati, si è scelto di utilizzare come evaluation technique una tecnica di tipo walk-forward: tale tecnica consiste nel suddividere i dataset creati in precedenza in diverse unità elementari che possono essere ordinate cronologicamente: nel lavoro in questione si assumono come tali unità le release di ogni progetto. Ad ogni iterazione, vengono utilizzate tutte le release precedenti a quella attuale come training set e quella immediatamente successiva come testing set. Per poter realizzare ciò, sono state manualmente selezionate le release dal dataset ed utilizzate per creare dei file .arff, al fine di consentire al tool Weka di poter analizzare tali dati.

Classifiers, Sampling and Feature Selection

I modelli di machine learning creati fanno uso di 3 classificatori: IBK, RandomForest e NaiveBayes. Tali classificatori verranno combinati, inoltre, con diverse tecniche di Sampling e di Feature Selection al fine di aumentare la loro accuratezza. Le metriche di riferimento per valutare tale accuratezza saranno Precision, AUC, Kappa e Recall. Al termine dell'analisi del dataset si determinerà quale risulta il miglior classificatore e per quale progetto, e tale classificatore potrà anche essere utilizzato al fine di supportare il processo di decision-making aziendale. Tra le tecniche di sampling utilizzate vi è l'Undersampling che consente di ridurre il numero di istanze della classe maggioritaria presente nel dataset in modo da avere lo stesso numero di istanze della classe minoritaria. Per poter implementare tale tecnica, si è configurato un oggetto SpreadSubsample tramite le opzioni "-M 1.0". Inoltre, vi è l'Oversampling, che permette di aumentare il numero di istanze della classe minoritaria nel dataset fino ad ottenere lo stesso numero di istanze della classe maggioritaria: per poter implementare la tecnica, in questo caso, è stato configurato un oggetto Resample tramite le opzione "-B 1.0 -Z 2*Y", dove Y rappresenta la percentuale di istanze che appartengono alla classe maggioritaria, calcolata volta per volta. Infine, vi è la tecnica SMOTE. Anche tale tecnica consente di maggiorare il numero di istanze della classe minoritaria, scegliendo però randomicamente un'istanza di essa, selezionando le k istanze più vicine (k-neighbours) e generando ulteriori istanze sintetiche aventi metriche molto simili a quelle selezionate. I grafici riportati di seguito sono stati realizzati mediante il software JMP pro. In allegato

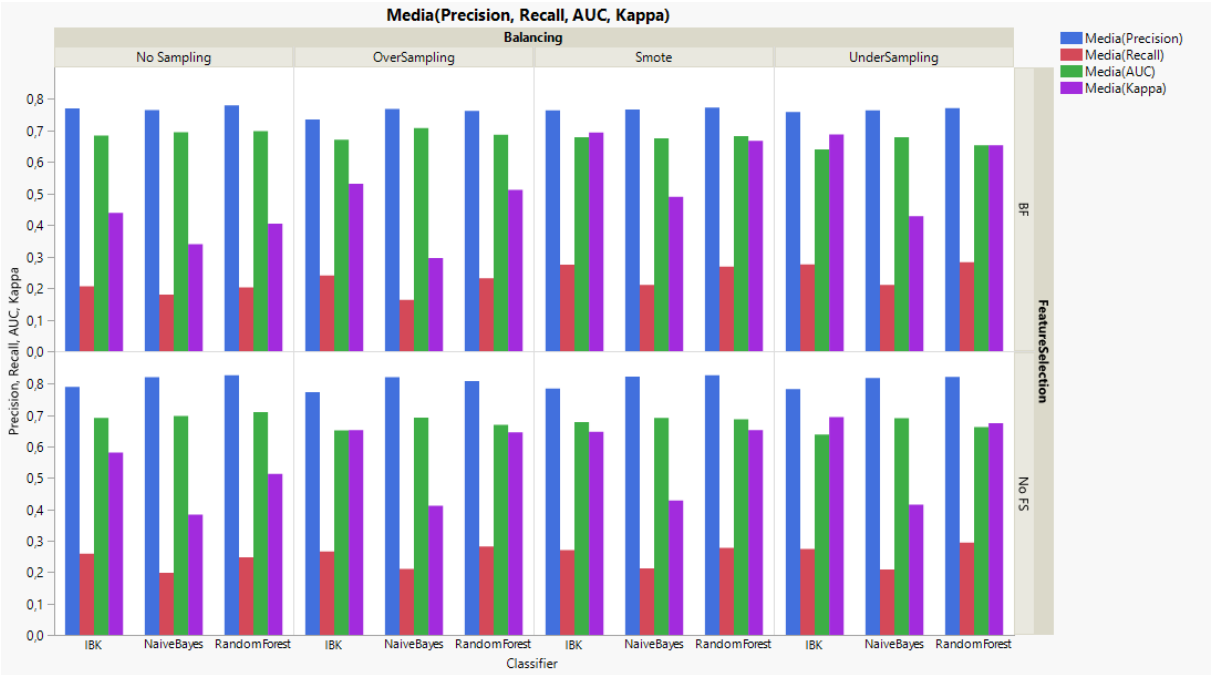
al presente report, vi è il file csv utilizzato per analizzare quantitativamente l'accuratezza dei modelli di machine learning creati.

R1 – What is the best Classifier?



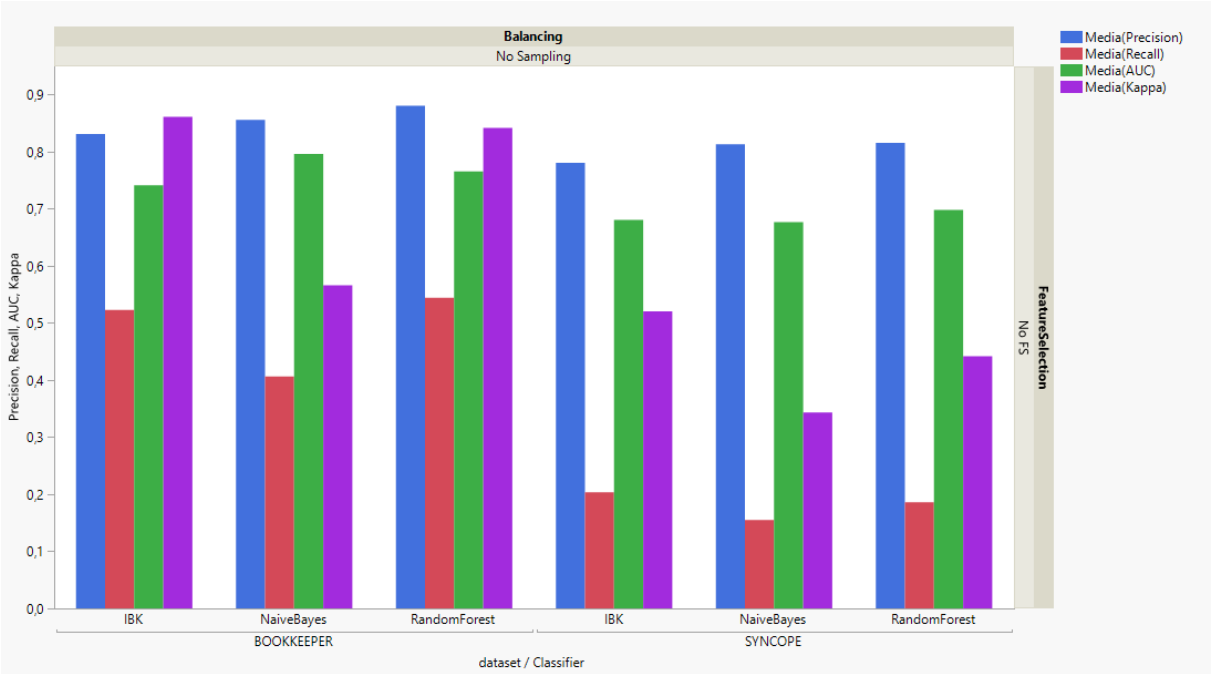
Il grafico sovrastante mostra i valori medi delle metriche considerando unicamente entrambi i progetti. In particolare, da tale grafico si può notare chiaramente come il classificatore IBK risulti il più accurato per la metrica Kappa. Inoltre, da esso emerge chiaramente che tutti i classificatori considerati risultano molto accurati per le metriche Precision e AUC, ma non risultano sufficientemente accurati per la metrica Recall, che si attesta attorno ai valori 0,2-0,3. Ciò è dovuto a causa di un numero considerevole di falsi negativi (FN) catalogati. Analizzando in maniera più approfondita tale grafico, si è notato che *RandomForest risulta il classificatore più accurato secondo le metriche AUC* (con un valore di quest'ultima pari a 0,71) *e Precision* (0,828), mentre *IBK per le metriche Recall e Kappa* (valori 0,259 e 0,581). Per poter decretare il miglior classificatore, si calcolerà la media tra i valori delle quattro metriche considerate per tutti e tre i classificatori e si definirà come migliore quello che consente di ottenere la media più elevata. In base a tale calcolo, si è osservato che *il miglior classificatore risulta essere IBK*, con una media complessiva pari a 0,58025. Di notevole interesse anche le performance del classificatore RandomForest che consente di ottenere una media di 0,57475.

R2 – Do Balancing and/or Feature Selection increase accuracy of Classifiers?



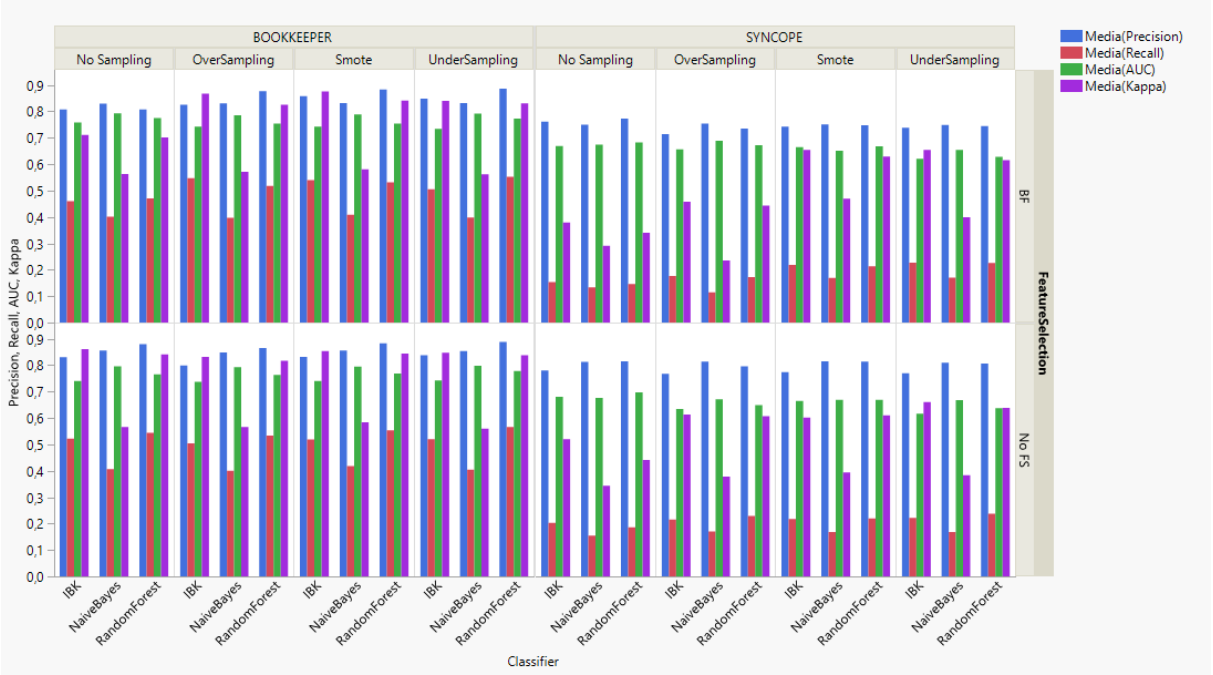
Il grafico riportato di sopra consente di visualizzare i valori delle metriche e dei relativi classificatori considerando tutte le tecniche di balancing e tutte le tecniche di feature selection utilizzate. Da una prima analisi di esso, è possibile notare che in generale le tecniche di sampling migliorano, per ogni classificatore, di poco l'accuratezza della metrica Recall e notevolmente il valore della metrica Kappa; mentre la tecnica di feature selection Best-First (BF), invece, sembri peggiorare l'accuratezza delle metriche nella maggior parte dei casi. In effetti, analizzando quantitativamente i dati ottenuti dal grafico, si è osservato che quest'ultima tecnica peggiora notevolmente i valori di Kappa: ad esempio, nel caso in cui essa venga applicata sul classificatore IBK, il valore di tale metrica passa da 0,581 a 0,413 (con una perdita quindi del 41% circa). Nell'unico caso in cui BF viene combinato con la tecnica di balancing SMOTE, utilizzando lo stesso classificatore IBK, esso consente di migliorare il valore di Kappa, seppur di poco (circa 1,30%). In tutti gli altri casi, invece, *l'uso della tecnica di feature selection Best-First, anche se combinata alle tecniche di balancing, peggiora l'accuratezza media dei classificatori dall' 1% al 10% circa.* Al contrario, *le tecniche di sampling considerate consentono di migliorare dall' 1% al 7% l'accuratezza media di tutti i classificatori;* infatti, l'utilizzo della tecnica di undersampling permette al classificatore RandomForest di risultare il più accurato di tutti, con una media dei valori delle metriche pari a 0,61325 e con un aumento percentuale del 6,7% rispetto al caso in cui non viene applicato alcun tipo di sampling. Tale tecnica consente al classificatore, inoltre, di ottenere il massimo valore di Recall presente, pari a 0,294 (con un aumento del 18,5%). I massimi valori delle metriche Precision ed AUC, invece, vengono raggiunti dallo stesso classificatore, ma senza l'ausilio dell'undersampling, che ha quindi peggiorato l'accuratezza di queste due metriche (di poco meno dell'1% nel primo caso, del 16,5% nel secondo caso). Per quanto riguarda l'ultima metrica rimasta, Kappa, il massimo valore presente è stato raggiunto (0,694) in due diversi casi: utilizzando il classificatore IBK in combinazione con SMOTE e BF, ed utilizzando il classificatore IBK con undersampling e senza feature selection.

R3 – For each project considered, what is the best classifier?



Il grafico in questione mostra i valori delle metriche ed i rispettivi classificatori suddivisi per progetto. Da esso è possibile notare che il modello di machine learning adottato per il progetto Bookkeeper risulta molto più accurato di quello adottato per Syncope, essendo quasi tutti i valor medi delle le metriche considerate maggiori nel primo progetto. Ciò potrebbe essere dovuto, in parte, anche al differente numero di release considerate per ogni progetto (6 per Bookkeeper, 24 per Syncope); in effetti, un più alto numero di release nel dataset in genere corrisponde ad un più alto numero di istanze nel dataset stesso e quindi anche ad un maggior rumore introdotto in esso. È possibile notare che le differenze più evidenti tra i due progetti riguardano le metriche Kappa e Recall. *Per quanto riguarda il progetto Bookkeeper*, analizzando quantitativamente i dati ottenuti *si è osservato che il classificatore più accurato in media risulta RandomForest*, con una accuratezza media pari a 0,7575; notevoli anche i risultati ottenuti dal classificatore IBK con un valore medio di 0,739. RandomForest, inoltre, consente di raggiungere i massimi valori presente per le metriche Precision, Recall ed AUC; mentre, IBK, il massimo valore presenti per la metrica Kappa. *Per quanto riguarda il progetto Syncope*, invece, *si è osservato che il classificatore più accurato risulta IBK*, con un valore di 0,5465, seguito da RandomForest con un valore medio pari a 0,5355. Il primo tra questi consente di ottenere i massimi valori presenti per le metriche Recall e Kappa, mentre il secondo consente di ottenere i massimi valori possibili per la metriche Precision ed AUC. Da notare, infine, che il modello creato utilizzando il classificatore migliore in Bookkeper risulta circa il 39% più accurato del modello creato utilizzando il miglior classificatore di Syncope.

R4 – For each project considered, sampling and/or feature selection increase classifiers accuracy?



Infine, è bene concludere analizzando i due progetti separatamente ed osservando il comportamento dei modelli creati su entrambi i progetti. Tramite il grafico sopra riportato, è possibile vedere che, anche in questo caso, i valori delle metriche risultano molto più alti per il progetto Bookkeeper. *Per quanto riguarda proprio Bookkeeper, la tecnica di feature selection best-first nella maggior parte dei casi peggiora l'accuratezza delle metriche dall'1% al 2%*. In alcuni casi, ad esempio se combinata alle tecniche di Oversampling e di Smote, essa consente addirittura di aumentare l'accuratezza media del classificatore IBK, del 2% nel primo caso e del 4% nel secondo caso. In altri casi, però, ad esempio se non combinata a nessuna tecnica di balancing, essa peggiora notevolmente l'accuratezza media dei classificatori IBK e RandomForest di circa il 7-9%. In tali casi, le metriche più penalizzate dalla feature selection risultano Kappa e Recall, che subiscono rispettivamente una riduzione del 20% e del 13,5% circa. *Le tecniche di sampling applicate a questo progetto permettono, invece, di variare l'accuratezza media delle metriche in positivo ed in negativo, in un range compreso tra -3% e +1,5%*, anche se in numerosi casi esse non consentono un miglioramento significativo. Il classificatore più accurato risulta RandomForest che, combinato alla tecnica dell' Undersampling, consente di raggiungere un accuratezza media di 0,768. In tal caso, esso consente di raggiungere i massimi valori per le metriche Precision e Recall, rispettivamente di 0,888 e 0,567. I massimi valori delle metriche AUC e Kappa, invece, sono stati raggiunti rispettivamente dai classificatori NaiveBayes con Undersampling (valore: 0,798) e da IBK con Smote (0,876). *Per quanto riguarda il progetto Syncope*, infine, *si è osservato che anche in questo caso la tecnica di feature selection BF peggiora notevolmente i valori delle metriche, con picchi anche del 10% - 11%* nel caso essa venga usata in combinazione con l' Oversampling. Se combinata con Smote, tuttavia, le perdite in percentuale su tali valori sono più moderati, attestandosi attorno allo 0,5%-2%. Per quanto riguarda *le tecniche di sampling, invece, esse consentono in tutti i casi un incremento dell'accuratezza dei classificatori*; notevole il caso di classificatore RandomForest con Undersampling, in cui è stato ottenuto un incremento del 13%. Tale incremento consente a RandomForest di risultare il miglior classificatore, con una accuratezza media di 0,6055. Tale combinazione consente, inoltre, di ottenere anche il massimo valore della metrica Recall, pari a 0,338. L' Undersampling, tuttavia, ha peggiorato i valori delle metriche Precision ed AUC che raggiungono il loro massimo sempre utilizzando il classificatore

RandomForest ma senza utilizzare a supporto alcuna tecnica di Balancing e di FS. Tale tecnica, tuttavia, ha consentito di raggiungere il massimo valore di Kappa per il classificatore IBK, pari a 0,661.

Conclusions and Future Works

Dallo studio fatto, è emerso che il modello più accurato per i progetti in questione, considerati sia separatamente sia nel loro insieme, risulta quello composto dal classificatore RandomForest combinato alla tecnica dell' undersampling. È emerso, inoltre, che la tecnica di Feature Selection BF peggiora, per ogni progetto, l'accuratezza media delle metriche. Tra i possibili lavori futuri, quindi, vi è sicuramente la possibilità di utilizzare più metriche di quante utilizzate sinora e verificare sia se l'accuratezza del modello aumenti, sia se in tal caso BF permette un netto miglioramento dei risultati. Inoltre, soprattutto per quanto riguarda il progetto Syncope, potrebbe essere molto interessante eliminare ancora più release al fine di ridurre ulteriormente lo snoring noise presente nel dataset ed osservare se, anche in tal caso, l'accuratezza del modello aumenta. Infine, potrebbe essere interessante implementare ulteriori algoritmi per la stima della buggyness delle classi e confrontare i risultati ottenuti da ognuno; successivamente, inoltre, si potrebbero creare modelli utilizzando questi diversi algoritmi e valutare quali di questi risulta più preciso.

Links

- Link Deliverable 1 Github:
  https://github.com/DomenicoVerde/Deliverable1
- Link Deliverable 1 TravisCI:
  https://travis-ci.com/github/DomenicoVerde/Deliverable1
- Link Deliverable 1 Sonarcloud:
  https://sonarcloud.io/dashboard?id=DomenicoVerde_Deliverable1
- Link Deliverable 2 Github:
  https://github.com/DomenicoVerde/Deliverable2
- Link Deliverable 2 TravisCI:
  https://travis-ci.com/github/DomenicoVerde/Deliverable2
- Link Deliverable 2 SonarCloud:
  https://sonarcloud.io/dashboard?id=DomenicoVerde_Deliverable2