Deliverable 1

Introduzione

Nello sviluppo software, la raccolta di misure ha un ruolo fondamentale nel miglioramento dei processi di sviluppo. Ne è un esempio il **CMMI**, il modello di riferimento per la quantificazione della capacità e della maturità di una singola azienda nella produzione di software.

Il modello offre un approccio a livelli, con il *quinto* che rappresenta il livello migliore di qualità per l'azienda. Per il raggiungimento del livello, tra le altre cose, è richiesto il controllo statistico del processo che porta ad un miglioramento continuo nel tempo.

Il *process chart* aiuta nella visualizzazione dei dati ed offre un supporto decisionale e di controllo sui processi in atto all'interno di un'azienda. E' utile quando l'obiettivo è quello di controllare il processo per andare a trovare e correggere dei problemi, ma anche nella predizione del range di un outcome da un processo.

Nel corso di questo studio abbiamo misurato la stabilità del numero di ticket risolti per mese relativi al progetto open-source **FALCON** (https://github.com/apache/falcon).

Progettazione

Il workflow di riferimento utilizzato per l'ottenimento delle misure necessarie è il seguente:

1. Memorizzazione dei ticket ottenuti da **Jira**
2. Per ogni commit della repository, controllo se il messaggio del commit contiene l'id di qualche ticket
3. Se il commit è associato ad uno o più ticket validi, incremento del numero di fix rilasciati nel mese

Variabili utilizzate

I ticket presenti sulla piattaforma *Jira* considerati per il singolo progetto sono stati quelli con attributi:

- **Status:** *Closed* o *resolved*
- **Resolution:** *Fixed*

Scelte implementative

E' stata utilizzata la libreria **JGit** per l'interfacciamento con GitHub. La libreria è stata utilizzata per effettuare le operazioni di *clone* della repository e l'iterazione sui singoli *commit*.

Risultati e discussione

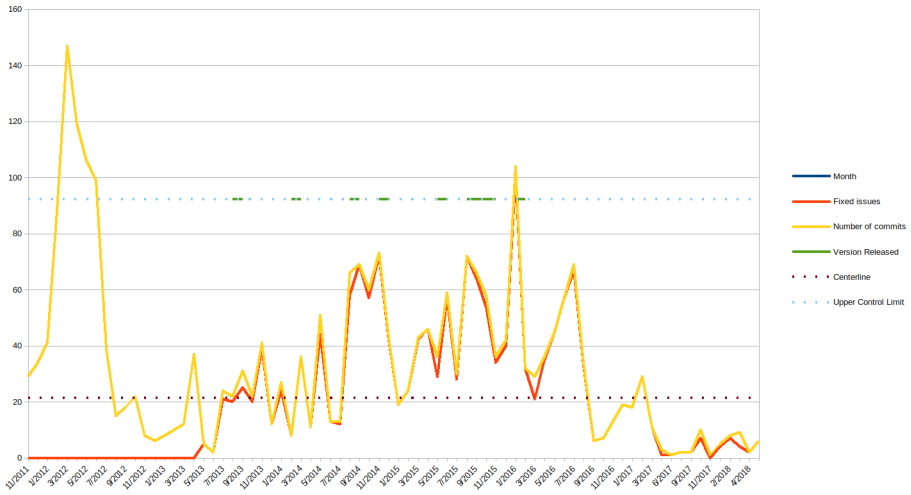

Grafico 1 - Process chart relativa ai ticket risolti - FALCON

Per una migliore analisi e contestualizzazione dei risultati, sono stati considerati il numero di commit e di versioni rilasciate nell'intervallo di tempo considerato. Nel grafico i trattini verdi indicano il rilascio di una nuova versione dell'applicazione software; inoltre, le varie metriche sono relative ad un determinato mese.

Dal grafico è possibile osservare come, ad eccezione di un singolo mese, il processo si è mantenuto stabile nell'intervallo di tempo considerato, non discostandosi troppo dal valore medio. I valori tratteggiati vengono calcolati come:

- **Centerline:** media dei *fixed issues*
- **Upper control limit:** *centerline - 3* (varianza fixed issues)*
- **Lower control limit:** *centerline + 3* (varianza fixed issues)*

Il mese di instabilità del processo, a differenza della tendenza generale, è preceduto dal rilascio di due versioni a soli due mesi di distanza. In oltre, è possibile vedere come successivamente non siano state rilasciate altre versioni al pubblico. Considerando il numero di commit, è possibile vedere come anch'esso è in controtendenza con la media generale, escludendo i primi mesi relativi alle fasi iniziali dello sviluppo del software.

La causa dell'instabilità nel mese può dunque essere dovuta al verificarsi della seguente situazione:

1. Apparizione di bug dormenti a metà dell'intervallo temporale considerato
2. Risoluzione dei bug dormienti e rilascio in tempi brevi di due versioni
3. Testing relativo al rilascio delle due versioni
4. Apparizione di nuovi bug
5. Rilascio dell'ultima versione

Una decisione da considerare nel processo di sviluppo, per prevenire periodi di instabilità del processo, può essere quella di dedicare un maggior effort alla fase di testing.

Deliverable 2

Introduzione

Nello sviluppo del software un aspetto rilevante è relativo alla qualità del software e del relativo testing. Per lo sviluppo di larghe applicazioni possono essere importanti degli strumenti di supporto per l'individuazione delle possibili classi contenenti bug, in modo da andare ad intensificare l'attività di testing.

Tra i vari strumenti a disposizione, l'analisi statistica dei dati offre la possibilità, partendo da un insieme di dati di riferimento, di predirre il comportamento futuro con un determinato grado di accuratezza. Il machine learning aiuta in quest'operazione in quanto offre un supporto software all'utilizzo della statistica per l'analisi dei dati.

Il supporto software è offerto da algoritmi e modelli predittivi che migliorano le proprie performance con l'esperienza accumulata su un determinato insieme di dati. L'apprendimento avviene sulle singole **istanze dei dati**, che rappresenta un esempio individuale del concetto tramite il quale il modello apprende. La singola istanza è composta da **attributi**, ognuno dei quali misura una determinata caratteristica.

Le predizioni riguardando i singoli attributi e possono essere di due tipi:

- **Regressione:** il valore predetto è un numero reale
- **Classificazione:** il valore predetto appartiene ad una categoria

Per le finalità dello studio sono state effettuate predizioni del secondo tipo, in quanto l'obiettivo ultimo è quello di andare a predirre la *buggyness* di una classe Java. In questo caso le possibili categorie dell'attributo sono due, *Yes* o *No*. L'obiettivo è stato quello di misurare l'effetto di tecniche di sampling e di feature selection sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni open-source.

In questo studio empirico sono state analizzate due applicazioni open-source, **Bookkeeper** (https://github.com/apache/bookkeeper) e **AVRO** (https://github.com/apache/avro), concentrando l'attenzione sulla prima metà delle versioni rilasciate pubblicamente.

Progettazione - Creazione del Dataset

Per la creazione del dataset, per segnalare una classe come buggy o meno, sono stati considerati i ticket presenti su Jira relativi ad un singolo progetto. Jira è un software di supporto ai progetti per il monitoraggio di ticket relativi ad issues.

Tra le informazioni che Jira offre sul singolo ticket, l'attenzione è focalizzata sui seguenti attributi:

- **Creation date:** data di creazione del ticket
- **Resolution date:** data di risoluzione del ticket
- **Opening version (OV):** indica la versione rilasciata successivamente all'apertura del ticket
- **Fixed version (FV)** indica la versione rilasciata successivamente alla chiusura del ticket
- **Affected versions (AV)**: indica la lista delle versioni affette dal bug

Le date di creazione e di risoluzione del ticket sono informazioni sempre disponibili, tramite le quali è possibile risalire ad *OV* ed *FV*. Diverso però è il caso dell'*IV* che, in caso di assenza, viene stimato.

Il workflow di riferimento utilizzato per la creazione del dataset è il seguente:

1. Vengono memorizzati i ticket ottenuti da **Jira**
2. Calcolo e/o stimo i valori di *OV*, *IV* ed *AV*
3. Per i ticket validi memorizzo i valori di *IV* ed *FV*
4. Per ogni commit della repository, viene controllato se il messaggio del commit contiene l'ID di qualche ticket
5. Per ogni ticket associato, per i file presenti all'interno del commit, l'attributo *Buggy* del dataset viene impostato a *Yes* per tutte le versioni comprese tra IV ed FV

Cosa si intende per *ticket valido*? In questo caso definiamo un ticket come **valido** se $OV != FV$ ed $IV != OV$, cioè se il singolo ticket possiede una lista di versioni affette diverse da 0.

Per la creazione della lista di *ticket validi*, per ogni ticket vengono calcolati e/o stimati i valori di IV, OV ed FV.

Proportion increment

Si è già parlato di IV e dei casi in cui è necessaria una stima del suo valore. Tra le varie strategie a disposizione, in questo studio empirico è stato considerato l'approccio **proportion**. L'idea alla base è che ci sia una proporzione tra il numero di versioni necessarie per individuare il difetto ed il numero di versioni per il fix.

La proporzione *P* viene calcolata come $P = \frac{FV - IV}{FV - OV}$, che permette di andare a stimare il valore di IV come $IV = FV - (FV - OV) * P$.

Il valore di *P* può essere calcolato in tre modi differenti:

- **Cold start:** mediana tra 75 progetti
- **Increment:** media tra i valori di P dei ticket precedenti
- **Moving Window:** media tra i valori di P degli ultimi 1% di difetti risolti

In questo studio empirico è stato utilizzato il metodo *increment*.

Calcolo fixed version

Per il calcolo del valore di *FV* viene utilizzata la variabile del ticket *resolution date*. Il valore di *FV* viene calcolato come l'indice della versione rilasciata dopo la data di risoluzione del ticket.

Per i ticket con data di risoluzione successiva al rilascio dell'ultima versione potrebbe essere un approccio errato, in quanto non avremmo nessun *FV*. Si tratta di una condizione che non si verifica nella nostro studio empirico, andando a limitare l'analisi solamente alla prima metà delle versioni rilasciate. In questo modo è sempre possibile trovare una versione successiva alla data di ogni singolo ticket considerato.

Calcolo Opening Version

L'idea è di considerare lo stesso approccio appena visto per la *fixed version*, con l'unica variante nell'utilizzare, come riferimento per il confronto, la data di creazione del ticket.

Calcolo Inizialization Version - Affected Version presente

Nel caso in cui l'AV dovesse essere presente tra le informazioni che del singolo ticket, è possibile calcolare in modo preciso il valore di IV come la versione con data di rilascio minore tra le tutte presenti all'interno della lista di *Affected Version*.

Stima Inizialization Version - Affected Version non presente

Nel caso in cui l'AV non dovesse essere presente, viene effettuata la stima del valore di IV. Per la stima, è prima necessario il calcolo del valore di P per i ticket con IV nota. Per ogni ticket

1. Viene calcolato il valore di P come $\frac{FV - IV}{FV - OV}$
2. Il valore di P calcolato viene associato al ticket
3. Successivamente, si procede alla stima del valore di IV, per i ticket sprovvisti di AV. Per il singolo ticket con indice *n*
4. Viene calcolato il valore di P come media tra i ticket con AV presente ed indice compreso tra 0 ed $n-1$
    1. Nel caso in cui non dovessero essere presenti ticket con AV presente nell'intervallo considerato, per la stima di IV considero il valore di OV
5. Con il valore di P trovato, viene effettiato il calcolo del valore di IV stimato. $PredictedIV = FV - (FV - OV) * P$
    1. Nel caso in cui il valore dell'IV stimato dovesse essere minore di 1, risultato nel caso di valori alti di P, IV viene impostato ad 1

Variabili utilizzate

I ticket presenti sulla piattaforma *Jira* considerati per il singolo progetto sono stati quelli con attributi:

- **issueType:** *Bug*
- **Status:** *Closed* o *resolved*
- **Resolution:** *Fixed*

Misure e procedura di misurazione

L'associazione delle misure avviene per la coppia **(Classe Java, Numero versione)**. Dopo aver effettuato il *clone* della repository del progetto da GitHub, viene creata una mappa con chiave la coppia (Classe, #Versione) e come valore l'array di valori relativo alle metriche considerate.

Le singole metriche illustrate di seguito sono relative alla singola coppia di valori.

LOC Added: utilizzando la libreria JGit, per ogni file (*DiffEntry*) all'interno del commit abbiamo a disposizione l'oggetto *Edit*, che contiene le informazioni sulle singole modifiche effettuate al

singolo file. Per il calcolo delle linee di codice aggiunte, vengono considerate le modifiche di tipo *Insert* ed il numero di linee aggiunte.

Max LOC Added: nel momento in cui vado ad aggiornare le metriche relative ad un file, vado ad effettuare il controllo tra le nuove linee di codice aggiunte, e la variabile relativa alla metrica, inizialmente impostata a 0, in modo da aggiornarne il valore, quando necessario.

Average LOC Added: per ogni file viene calcolata come di $\frac{LOC\,Added}{Number\,of\,revisions}$.

LOC Touched: utilizzando la libreria JGit, come visto per le *LOC Added*, sono state considerate le linee di codice modificate dalle modifiche di tipo *insert, delete e replace*.

Number of Revisions: il numero di revisioni viene calcolato come valore incrementale, incrementato ogni qualvolta un file appare in un commit.

Number of Bug Fix: il numero di bug fix viene calcolato come il numero di *ticket Jira* associati al commit.

Change Set Size: il numero dei file contenuti nel commit considerato viene calcolato tramite la size della lista di oggetti DiffEntry presenti nella libreria JGit. Il singolo oggetto indica il singolo file all'interno del commit. Tramite la dimensione della lista è possibile ottenere il numero di file contenuti nel commit.

Max Change Set Size: viene applicata la stessa procedura di misurazione vista per *Max LOC Added*

Average Change Set Size: viene applicata la stessa procedura di misurazione vista per *Average LOC Added*

Buggy: la metrica relativa alla bugginess di una classe viene impostata a *True* in due casi:

1. Il file è presente all'interno di un commit con almeno un ticket Jira associato
2. Il file, in una versione differente, è associato ad un ticket Jira valido e la versione del file rientra nell'intervallo **[IV, FV)**

Scelte implementative

E' stata utilizzata **JGit** come libreria per l'interfacciamento con GitHub. E' stata utilizzata per effettuare le operazioni di *clone* della repository, l'iterazione sui singoli *commit* e l'ottenimento delle modifiche, con relativi file, tra due commit.

Progettazione - Predizione classi buggy

Walk Forward

Data la dipendenza temporale dalle singole versioni del dataset, è stata utilizzata la tecnica **walk forward** per la validazione del dataset.

Utilizzando la tecnica **walk forward** vengono eseguiti $n-1$ run, dove n in questo caso rappresenta il numero di versioni all'interno del dataset considerato. Per ogni singolo run, denominato con i, vengono considerate tutte le versioni comprese tra 1 ed i per il training set, e

la versione $i+1$ per il training set. L'ultima iterazione corrisponde alle versioni da 1 a $n-1$ all'interno del training set, e la versione n come testing set.

Feature Selection

La tecnica **feature selection** permette di ridurre il numero degli attributi non correlati, cercando in questo modo di migliorare l'apprendimento del classificatore rispetto al caso in cui si considera l'intero dataset.

Per la valutazione della correlazione dei singoli attributi è stato utilizzato l'oggetto **CFSSubsetEval** offerto da *Weka*.

- L'idea alla base è quella della creazione di un subset contenente attributi che sono altamente correlati tra di loro.

La ricerca e selezione degli attributi viene effettuata secondo l'approccio **backward**. Inizialmente vengono considerati tutti gli attributi del dataset, procedendo con la rimozione di un solo attributo per volta fin quando non si assiste ad un significativo decremento dell'accuratezza.

Sampling

Dal dataset di partenza è possibile vedere come ci sia uno sbilanciamento tra il numero di istanze delle classi **No buggy** e **Buggy**. Per evitare che i classificatori abbiano un'alta accuratezza sulle istanze del primo tipo, ma bassa sulle secondo, vengono applicate differenti tecniche di **sampling** per andare ad effettuare un bilanciamento del *training set*.

Undersampling: per bilanciare il numero delle classi, vengono estratte dalla classe maggioritaria un numero di istanze pari al numero della classe minoritaria. Per applicare l'*undersampling* abbiamo utilizzato l'oggetto **Resample** offerto da *Weka*, andando a specificare le opzioni *-M 1.0*.

Oversampling: per bilanciare il numero delle classi, vengono ripetute le istanze della classe minoritaria fin quando il loro numero non sarà uguale a quello della classe maggioritaria. Come per l'*undersampling*, abbiamo utilizzato l'offerto **Resample**, andando a specificare le opzioni *-B 1.0 -Z 2*percentuale_classe_maggioritaria*.

SMOTE

La tecnica **SMOTE** è un raffinamento dell'oversampling, in quanto l'estensione della classe minoritaria non avviene con la ripetizione delle istanze, ma bensì con la generazione di queste ultime in modo sintetico.

Per l'ottenimento di questo risultato vengono utilizzati degli algoritmi basati sulle similitudini esistenti tra le istanze della classe minoritaria, per andare a generarne di nuove in numero necessario.

Misure e procedura di misurazione

Per il calcolo delle metriche di ogni singolo run, sono stati utilizzati i metodi dell'oggetto **Evaluation** offerto da *Weka*. Tutte le misure elencate qui di seguito sono ricavate dalla matrice di confusione, contenente il numero di *TP, FP, TN, FN*. Tutte le metriche, con la sola **kappa**

esclusa, sono riferite ad una specifica classe di appartenenza dell'attributo che si vuole andare andare a predirre. In questo caso tutte le misure dell'attributo *Buggy* sono calcolate per la classe **Yes**.

Numero true positive: il numero di istanze positive correttamente predette.

Numero true negative: il numero di istanze negative correttamente predette.

Numero false positive: il numero di istanze predette positive ma che invece erano negative.

Numero false negative: il numero di istanze negative ma che invece erano positive.

Precision: il rapporto $\frac{TP}{TP+FP}$ indica la frequenza con cui il classificatore è stato in grado di indicare in modo corretto un'istanza come positiva.

Recall: il rapporto $\frac{TP}{TP+FN}$ indica la frequenza con cui il classificatore è stato in grado di indicare un'istanza come positiva.

Kappa: valore compreso tra -1 ed 1 che indica quanto il classificatore è migliore di un classificatore standard (*stupido*).

AUC: il rapporto tra $TPR=\frac{TP}{TP+FN}$ e $FPR=\frac{FP}{FP+TN}$, indica la capacità del modello di distinguere tra le classi

Variabili utilizzate

Ai fini dello studio sono stati considerati:

- **Classificatori:** *RandomForest*, *NaiveBayes*, *IBk*
- **Feature selection:** *no selection*, *best first*
- **Balancing:** *no sampling*, *oversampling*, *undersampling*, *SMOTE*

Scelte implementative

Per l'implementazione del modulo relativo alla predizione, è stato utilizzato **Weka**, tramite le API offerte per Java.

Risultati e discussione - Bookkeeper

Figura 2 - Risultati metriche Bookkeeper

Con l'applicazione della tecnica di *feature selection*, tutti i classificatori vedono un decremento ed una maggiore variabilità dei valori di *precision* ed *AUC*. Si ha un incremento del valore della *Recall*, con la variabilità che si riduce sostanzialmente rispetto al caso senza applicazione della tecnica. Risultato "misto" per quanto riguarda il valore di *Kappa*, che aumenta il valore massimo ma aumenta la variabilità e, di conseguenza, segna nuovi valori minimi.

Le varie tecniche di *balancing*, associate a *feature selection*, danno gli stessi risultati. Abbiamo dunque una maggiore variabilità, e valori minimi più bassi, per *precision e AUC*, un aumento della *recall* ed il valore di *kappa* che aumenta valori massimi ma anche la sua variabilità.

Risultati differenti nel caso del *balancing* senza *feature selection*, dove si ha, in generale, un aumento del valore della *recal* e di *kappa*. Per il dataset considerato, il classificatore migliore risulta essere **NaiveBayes** senza *feature selection* e *SMOTE* come tecnica di sampling.

Risultati e discussione - AVRO

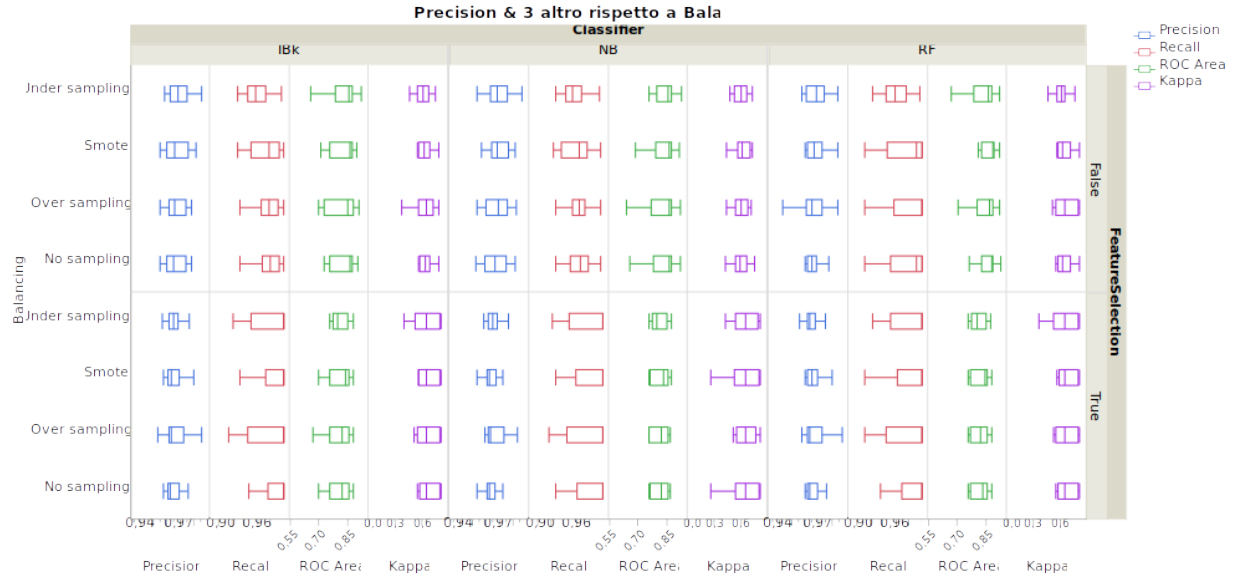

Figura 3 - Risultati metriche AVRO

Dal grafico in *figura 3* è possibile vedere come in questo caso, rispetto a quanto visto per il dataset relativo a *Bookkeeper*, la variabilità dei risultati tra le varie tecniche è maggiore. Le metriche utilizzate sono le medesime, ma la composizione delle singole istanze del dataset è tale da aver generato questi risultati per l'accuratezza della predizione dei vari classificatori.

Con l'applicazione della tecnica di *feature selection*, tutti i classificatori vedono un incremento dell'accuratezza nella classificazione delle classi *buggy*. Rispetto al caso base, abbiamo intervalli minori per *precision*, *AUC* e *recall*, mentre il valore di *kappa* rimane sostanzialmente invariato. NB è il classificatore. Rispetto ai tre, **NaiveBayes** ottiene il miglioramento maggiore, in quanto possiamo vedere come risultino più stabili le distribuzioni di *precision* e di *AUC*, con un valore leggermente maggiore della metrica *Kappa*.

Similitudini tra i risultati che si riscontrano anche nell'applicazione delle teniche di balancing. Dal grafico si evince come *under sampling* vada a migliore *AUC* ma con valori minori di *recall*; questo mentre *over sampling*, al contrario, migliora *recall* ma decrementa *AUC*. La migliore tecnica di sampling si rivela essere, per tutti i classificatori, SMOTE. E' possibile vedere dal grafico come, sopratutto nel caso di **RandomForest**, si ha un notevole incremento dei valori massimi della distribuzione della metrica *kappa*, con una minore variabilità per *precision* ed *AUC*. il classificatore migliore risulta essere **RandomForest** con *feature selection* e *SMOTE* come tecnica di sampling.

Risultati e discussione - Conclusioni

In conclusione, avendo analizzato i due dataset, possiamo dire che le due tecniche applicate hanno permesso un miglioramento dell'accuratezza dei classificatori nel caso di AVRO, ma non per Bookkeeper; la tecnica di *feature selection* ha portato ad una degradazione della precisione dei classificatori. Le tecniche di *sampling* invece, **SMOTE** su tutte, hanno portato ad un aumento della precisione dei classificatori considerati.