On Statistical Process Control and class defect prediction: a better approach

Matteo Esposito[1]

[1] Studente Laurea Magistrale in Ingegneria Informatica,
Università degli Studi di Roma "Tor Vergata", Roma, Italia

E-mail: matteo.esposito.nettuno@alumni.uniroma2.eu

Abstract

Statistical Process Controll let developer and organizations know how the sofwtare development cycle is effective or not surely this can't be the only metrics involved in controlling a development process. A much more proactive approach is the defect prediction but in open source projects where, given projects restarts, merges and so on, it is not always possible obtains all the metrics needed for building a good model, the approach presented in this paper aims to develop a more accurate and less prone to spoil model.

Keywords: statistical process control, machine learning, weka, F1-metrics, kappa, filters, apache, open source

1. Deliverable 1

1.1 Specifiche e breve introduzione

Lo scopo della prima deliverable, così come espresso dalle specifiche nel materiale del corso, è incentrata nella raccolta incrociata e plot di dati presenti sulle piattaforme GitHub e Jira su di una specifica metrica al fine di formulare osservazioni sullo statistical process control. Lo Statistical process control in lingua inglese o controllo statistico di processo è l'applicazione di un metodo matematico o statistico che consente di contenere l'esito di un processo all'interno di specifici limiti, determinati attraverso lo studio della variazione naturale dei limiti del processo.

Uno dei principali metodi statistici applicabili è la rilevazione dello scarto quadratico medio, detto anche deviazione standard. Attraverso la correlazione tra questo valore e la distribuzione normale dei dati rilevati si ottengono i limiti naturali della variazione di un processo.

Un processo viene poi classificato in base all'esito dell'osservazione che ne deriva in due categorie:

- sotto controllo statistico, quando è influenzato unicamente da fattori casuali, cioè da "madre natura"
- fuori controllo statistico, quando l'influenza delle sue variazioni è causata da fattori specifici.

In particolare, se un processo è sotto controllo statistico, l'osservazione del suo andamento nel tempo, effettuando un campionamento di dati, seguirà una distribuzione di frequenza che sarà molto simile a una distribuzione normale. Questo implica che l'analisi del processo dal punto di vista della media e della sua variabilità sia fondamentale per determinare i "limiti naturali" del processo, entro i quali, se non accadono cause specifiche, il processo manterrà il proprio andamento. Viceversa, una volta studiato un processo che si mantiene entro certi limiti con una distribuzione di probabilità ben definita, quando questa distribuzione cambia, oppure cambiano i limiti, si può pensare ragionevolmente dal punto di vista statistico che stia agendo una "possibile causa di fuori controllo". Nello specifico, le specifiche prevedono di:

- misurare la stabilità del numero di fixed bugs del progetto Eagle, realizzando un process control chart;
- individuare eventuali periodi di instabilità relativamente al numero di fixed bugs e cercare di identificare le possibili cause di tale comportamento ed eventuali soluzioni da adottare in futuro;
- determinare se il processo può essere considerato stabile (stable) in relazione all'attributo studiato.

1.2 Descrizione della Soluzione

Raccolta dei dati

Al fine di effettuare la raccolta dei dati è stato necessario ottenere attraverso le API RESTfull delle rispettive piattaforme, le seguenti informazioni:

- JIRA: ticket chiusi / risolti del progetto in esame filtrati sull'attributo "fixed bug"
- GitHub: data del commit che ha chiuso / risolto il ticket catturato prima da JIRA

L'utilizzo delle due piattaforme si è reso necessario in quanto Jira, non essendo completamente automatizzato e spesso scarsamente moderato, non contiene la data corretta della risoluzione del ticket, inoltre è privo di qualsiasi riferimento, nella maggior parte dei casi, di riferimenti diretti ai commit su GitHub: la ricerca incrociata infatti è stata effettuata attraverso espressioni regolari, semplificate dal tipo "String" di Java, per tanto io dati non sono corretti al 100%, ciò è inoltre dovuto ad un'assenza di standardizzazione realmente applicata nelle repository sullo stile dei commit, infatti, sono diversi i commit che tecnicamente risolvono un ticket ma non riportano la canonica stringa "[projectName – JiraTicketID]" inficiando ulteriormente sui risultati finali che non possono rispecchiare la veridicità del processo, differentemente da quanto accadrebbe in un'azienda con un buon livello di maturità e che quindi si possono solo avvicinare per approssimazione grossolana ma comunque utile al nostro attuale scopo.

Per maggiore chiarezza si riporta tra le reference l'URL [2] usata come chiamata alle API di Jira ed il cui risultato, come da codice, è stato quindi salvato in una **HashMap**<*String,Date*> ove i rispettivi campi sono stati valorizzati con:

- **String**: JiraTicketID
  il campo Date viene semplicemente inizializzato
- **Date**: new Date(0)
  verrà popolato nei metodi successivi

Per perseguire i prefissati scopi ora si rende necessario valorizzare correttamente il campo data attraverso un controllo incrociato con la piattaforma di GitHub dove risiede il codice sorgente del progetto in esame. Vengono quindi salvati in memoria l'interezza della history dei *commit* della *repository* contenenti tutti i rispettivi attributi aggiuntivi messi a disposizione da GitHub.

Preparazione all'elaborazione dei dati raccolti

Al termine del precedente step sono stati raccolti in via preliminare i seguenti dati:

- Lista di tutti i ticket di Jira conformi alla query
- Lista di tutti i commit sulla repository di GitHub

È possibile quindi incrociare i dati delle due piattaforme, attraverso manipolazioni di stringhe come da codice, ed ottenere la struttura dati finale contenente tra i suoi attributi l'attributo "*aggregated*" consistente in una semplice **HashMap**<*String,Integer*> contenete questa volta:

- **String**: rappresentazione testuale di un mese compreso tra il primo e l'ultimo commit della repository nell'intervallo temporale preso in considerazione
- **Integer**: numero dei bugfixed nel mese in questione

Da questa struttura dati è stato quindi possibile creare il file *csv* finale contenente le informazioni sull'attributo scelto per lo *statistical process control*.

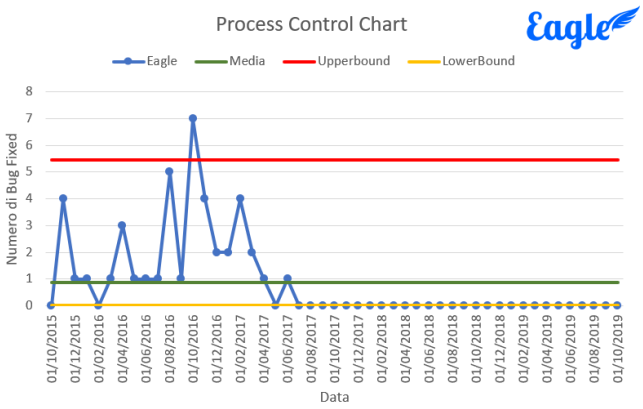

Grafico 1 Process Control Chart - Eagle

Elaborazione dei dati raccolti ed osservazioni

E possibile osservare dal Grafico 1 Process Control Chart - Eagle i valori assunti nel tempo del numero di bug fixed nei rispettivi archi temporali e inoltre osservare fin da subito l'anomalia presentatesi nel'ottobre 2016 ove l'alto numero di bug correti, ovvero ben 7, ha decretato inesorabilmente il giudizio di processo "fuori controllo" in quanto violava il prinicpio per cui un processo può essrre definito stabile. Il principio in questione sanciva che il numero di bug fixed dovesse trovarsi nell'intervallo sifatto:

$$Bound = avg(values) \pm DevStand(values)$$

I valori assunti dai parametri nella formula sono mostarti in Tabella 1. Si noti che il lowerbound risutlat essere negativo,

data la natura dei dati è privo di senso porre un lwoerbound negativo o pararle di "numero di bug fix negativi" in quanto ci si aspetta che sia un valore strettamente maggiore di zero al più ad esso uguale, per questo motivo si può constatare che nel Grafico 1 il lowerbound ha un valore pari a 0.

Degno di nota anche se risultato noto e logico il numero frequente pari a 0 di bug fixed nella finestra temproale più recente. In effetti i processi di sviluppo softwarte subiscono una maturazione con il passare degli anni e l'instaurazione di *policies*più stringenti e adeguate, non è quindi un caso osservare tale andamento se pur forse eccessivamente marcato in questa data istanza. In ultima analisi è posisbile osservare che nella finestra di correzione di bug considerata si evidenzia un sostanziale andamento nel rispetto del valor medio con alcune fluttuazioni dovute, probabilmente ad un acerbità del processo stesso.

Valori Chiave del Grafico 1	
Valor Medio	0,857142857
Deviazione Standard	1,538618516
Boundaries	± 5,472998406

Tabella 1

Dettagli implementativi

Al fine di avere accesso ai risultati delle API e presentandosi quest'ultime in formato JSON è stato possibile generare automaticamente tutte le entità coinvolte (generando inoltre un considerevole numero di code smells) attraverso il servizio online JsonSchema2Pojo[3] il quale consente, a partire da una stringa JSON (e non solo) di ricavare un *Plain Old Java Object*[6] serializzabile e con le annotazioni per due librerie di serializzazione di oggetti java Jackson[4] e Gson[5].

Per esperienza pregressa del corso di Mobile Programming è stata scelta la libreria Gson la quale è in grado di parsificare automaticamente una stringa JSON e restituire il corrispettivo in oggetti instanziati sfruttandone il costruttore pubblico più lungo.

2. Deliverable 2 – Defect Prediction

Diversi sono gli studi svolti nell'ambito dell'Ingegneria del Software volti ad analizzare statisticamente dati estratti da metriche ben precise applicate a repository di codice sorgente di grandi aziende per cercare correlazione tra le suddette metriche e l'insorgere inevitabile di bug nel codice.

Con il passare del tempo e con l'avvento di calcolatori personali sempre più performanti e potenti si è aperta la possibilità di applciare le conoscenze statistiche sul pregresso in modo proattivo e quindi di cercare di predire con una data precisione e date metriche la possibilità dell'insorgenza di bug nel codice sorgente. Come da specifiche progettuali sono state sfruttate le seguenti tecniche:

Evaluation Technique	Feature Selection	Balancing	Classificatori
Walk Forward	*No Selection* *Best First*	*No Sampling* *Over Sampling* *Under Sampling* *SMOTE*	*Random Forest* *Naive Bayes* *IBK*

Tabella 2 strumenti usati nella fase di predizione

Al termine della fase di raccolta di dati essi sono stati elaborati al fine di valutare quali eventuali tecniche di feature selection e balancing considerate aumentano l'accuratezza dei modelli predittivi, in base:

- al classificatore utilizzato;
- alla metrica di accuratezza considerata.

Secondo le specifiche, infine, questo studio viene svolto su due diversi progetti di proprietà dell'Apache Software Foundation, il sistema di log distribuito BookKeeper ed il sistema di warehouse per database relazionali od orientati ai BigData Tajo (pronuncia spagnola, "tacho").

1.3 Descrizione della Soluzione

Iniziallizazione e preparazione dell'ambiente

Per affrontare uno studio così complesso e dettagliato è necessario preparare l'ambiente di esecuzione al fine di:

- produrre più dati intermedi possibili lasciando comunque il resto della struttura del file system ordinata e pulita
- tenere sotto controllo e "a portata di reference" diverse strutture dati nel corso dell'esecuzione dell'applicativo
- tenere traccia dell'esecuzione per un debug più approfondito

L'applicativo di base accetta come parametri obbligatori il nome del *maintainer*/*proprietario* della repository ed il nome del progetto stesso, in assenza del terzo parametro opzionale ovvero il *path* locale ad un clone della *repo*, la routine si occupa, su *thread* separato, del clone e del setup su cartella temporanea recante prefisso specifico *"me_java_isw2_[random]"* al fine di essere automaticamente eliminata dal gestore delle cartelle temporanee del Sistema Operativo. Terminato il download della repository in locale la utility scritta è pronta a collezionare tutte le informazioni necessarie allo studio con machine learning.

Raccolta dati preliminare

Nella fase di raccolta dati preliminare l'applicativo, per scelta progettuale e per motivazioni di inconsistenza, richiede alla piattaforma di GitHub di ottenere tutti i Tag (= release) del progetto attualmente sotto esame e, come da specifiche, si concentra esclusivamente sulla prima metà delle release disponibili scartando la metà più recente.

In questa fase, attraverso un *thread* instanziato con le lambda di Java si occupa di ottenere ulteriori informazioni sui tag ovvero l'intervallo di tempo reale che intercorre tra una release e la successiva, dato utilissime per approssimare eventuali Opening Version assenti dai ticket di Jira.

Ricerca della buggyness

Attraverso i dati precedentemente raccolti e l'utilizzo delle API di JIRA è possibile ottenere tutti i ticket risolti / chiusi nella time frame attualmente considerata al fine di ottenere informazioni sulla buggyness già riscontrata nelle release in esame. La stima della buggyness delle classi java può essere effettuata in due modi, con il verboso utilizzo di zzs e blame molto prono a falsi positivi o attraverso la stima ottenuta con il metodo della proportion basata su attributi che, almeno in teoria, i ticket di Jira possiedono:

- **Opening Version**: Versione in cui è stato aperto il ticket
- **Fixed Version**: Versione in cui è stato risolto / chiuso il ticket
- **Affected Versions**: Lista delle versioni affette dal bug contenuto nel ticket aperto nella Opening Version e risolto / chiuso nella Fixed Version
- **Injected Version**: Affected Version più vecchia

Secondo il metodo della proportion i sopra citati attributi sono legati da una relazione matematica:

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

Spesso nei ticket di Jira non è possibile ottenere l'*Injected Version* poiché non è valorizzato il campo delle *Affected Version* ciò nonostante è possibile invertire la precedente equazione ed ottenere un'espressione per *l'Injected Version*: $IV = FV - (FV - OV) \cdot P$ tuttavia il valore di P non può essere statico e soprattutto non sarebbe utilizzabile se, nell'interazione dei presenti calcoli, il primo ticket fosse sprovvisto dell'*affected version*. Il metodo della *proportion*, nella realizzazione da me scelta delle *moving windows* consente di ottenere una stima del valore di P pari all'1% degli ultimi valori di P assegnati e / o calcolati.

Scelta progettuale: è giusto sottolineare in questa sede le scelte fatte di esclusione di Ticket che cadessero in una delle seguenti categorie:

- Malformati:
  - Injected Version successive ad Opening Version
  - Fixed version precedent ad Opening Version
  - Mancanti di Fixed Version
- Estranei al periodo preso in considerazione

Al termine dell'analisi di tutti i ticket e dei relativi commit associati (ottenuti in modo similare alla D1) è possibile ottenere quindi una **HashMap**<*String*,***List***<*String*>>:

- **String**: nome della classe affetta da Bug
- **List**<*String*>: nome delle release in cui la classe era buggy

Con i presenti dati è quindi ora possibile proseguire con la raccolta di tutte le altre metriche scelte per la realizzazione dello studio.

Raccolta delle metriche

Scelta progettuale importante: Nel corso dello sviluppo del presente progetto è stata presa una scelta peculiare nella modalità di recupero delle metriche al fine di limitare le metriche dal viziare eccessivamente il modello. I commit e quindi le metriche sono difatti calcolate esclusivamente su tutte le classi e commit presenti tra una release e la sua successiva nella lista di release in considerazione. Questo diverso punto di vista sposta l'analisi dalla predizione della buggyness generale alla predizione della prima volta che una classe sviluppi un difetto, il riuso della precedente iterazione del modello, nella fase di calcolo con Weka, consentirà di rispondere, secondo l'evidenza empirica, ad entrambi i quesiti con una precisione più elevata rispetto alla controparte.

Terminato il thread della Buggyness viene effettuato lo spawn di n thread dove n = numero di release prese in considerazione. I thread di calcolo delle metriche agiscono per "riempimento lineare" calcolano cioè in maniera parallela di tutte le metriche per ogni singolo file in ogni singolo commit presente tra una release e la sua successiva.

Metric (File C)	Description
Size	LOC
LOC_touched	Sum over revisions of LOC added+deleted+modified
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 in C
MAX_Churn	MAX_CHURN over revisions
AVG_Churn	Average CHURN per revision
ChgSetSize	Number of files committed together with C
MAX_ChgSet	Maximum of ChgSet_Size over revisions
AVG_ChgSet	Average of ChgSet_Size over revisions
Age	Age of C in weeks
WeightedAge	Age weighted by LOC_touched
NSmells	Number of smells

Tabella 3 - Metriche disponibili e relativa descrizione

Le Metriche escluse dalla Tabella 3 per motivi di tempo e di ottimizzazione sono state le metriche di NFix e NSmells, mentre tutte le altre metriche sono state prese in considerazione e sono state calcolate così come dscritto dal codice allegato.

WEKA – Preprocessamento ed elaborazione dati

Ottenute le metriche di riferimento diveine possibile creare i training set ed i testing set estraendo dai csv tutti i dati atti a creare validi sets secondo la tecnica di valutazione *walk forward*. Dall'analisi dei risulati del precedente stage nasce però una riflessione sulla natura di alcui valori. Esistono progetti come i due in questione che provengono da cicli di sviluppo software sensibilmente diversi tra loro, su repository ormai scomparse e di cui non si ha più traccia. Tali progetti si prestano in effetti ad un'analisi meno accurata di altri di cui si ha la storia completa e in effetti è il caso in esame: l'analisi ad occhio nudo dei dati rivela infatti una percentuale alta di classi sprovviste di metriche poichè non modificate nel corso di una release specifica e tale eprcewntuale oscilla sensibilmente tra il 15% ed il 25% del totale dei dati.

Basti pensare che la prima release di BookKeeper sull'attuale repository è la versione 4.0.0 difettando di men 3 precedenti Major Realases. Da qui un'intuizione: effettuare i calcoli solo con batch di dati esenti da questa macnanza di metriche per evitare di viziare il mdoello stesso. Ovviare gettare dati non è una soluzione fattibile poiché rappresenterebbe una perdita e si ridurrebbero i test allorchè si è pensato invece di lanciare un algoritmo in grado di inferire le metriche mancanti dall'analisi delle metriche di tale classi in release precdenti e/o successive arrivadno a coprire l'intero dataset, questa procedura è stat seguita per il modello di training, mentre per il modello di testing si è scelto in effetti di scartare i dati nulli poiché la *concern* era sul modello che si sarebbe creato attraverso il set di training.

Un'altra nozione degna di nota è la problematicità dei valori nulli nei report finali Weka ovvero l'annosa questione della divisione per zero per il calcolo dell'AUC, Recal e Precision. A tale riguardo si rimanda alla reference 8 ove è presente un link ad un post sulla piattaforma di GitHub dove vengono discussi proprio questi temi e da cui è stata presa la decisione progettuale su come valorizzare i campi NaN.

Analisi dei dati

La Research Question al centro del presente report è definita come la ricerca della tecnica di ***feature selection*** e-o di ***balancing*** che siano in grado di migliorare l'accureatteza di un classificatore tra quelli precedentemente considerati attraverso l'analisi delle metriche di accurattezza considerate in Weka (**AUC, Kappa, Precision, Recall**) .

L'analisi verrà svolta seguendo uno schema ben preciso per cercare di arrivare ad una soluzione il più precisa possibile, si comincerà da un'analisi delle singole metriche per poi aggregare i dati dei classificatori ed allargare il fuoco nel contesto più ampio possibile. I dati usati per elaborare i grafici da ora mostrati sono tratti direttamente dal nuovo filtro mentre nell'ultima sezione verranno discussi e confronati rispetto all'approccio classico.

BookKeeper

Iniziamo l'analisi del progetto BookKeeper con il commento sull'AUC (Area Under Curve) premettendo che l'analisi dei boxplot sarà basata su considerazioni mediane e di distribuzione statistica della "popolazione". Dall'analisi del Grafico 2 è possibile evincere che per i tre classificatori i valori milgiori in termini medianici e distributivi si hanno rispettivamente per le coppie:

- IBk: UnderSampling & BestFirst
- NaiveBayes: NoBalancig & BestFirst
- RandomForest: UndeSampling & No Selecetion

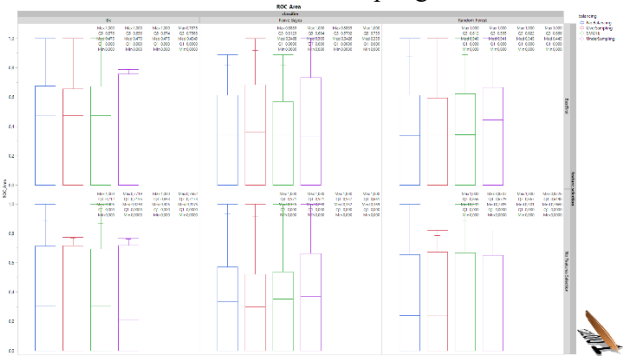

Grafico 3 BK-ROC Area

Le scelte sono state dettate dalla configurazione stessa del grafico, infatti dal grafico si evince che nel caso di IBk, mentre negli altri casi si avevano valori anche pari ad uno la variabilità era eccessiva infatti si aveva un 50 % dei valori al di sotto dello 0,5 mentre poi si aveva un 25 % concentrato tra 0,5 e 0,7 ed un altro 25% tra 0,7 ed 1, discorso similare per tutte le altre features di balacnig e per gli altri classificatori. È ncessario infatti prediligere una minore varabilità, indice di consistenza dei risultati piùttosto che un'alta varaibilità ed un range compleot includendo il massimo. Inoltre nell'instanza corrente un valore mediano alto ed un primo quantile

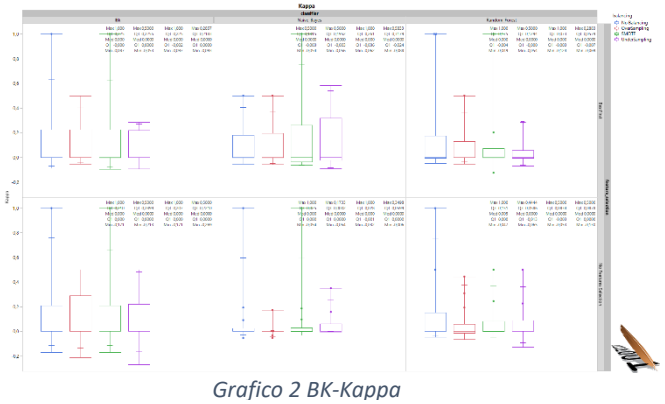

Grafico 2 BK-Kappa

coincidnete con il minimo assicura una buona consistenza di risultati positivi, non ottimi ma pur sempre positivi

Dal Grafico 3 BK-Kappa è possibile invece osservare nei rispettivi casi una varianza davvero notevole e la comparsa di diversi *outliers.* Nel classificatore NaiveBayes, infatti, la sua iterazione priva di Feature Selection risente sensibilmente degli outliers portando il grafico ad evidenziare un'altissima variabilità e quindi una forte inconsistenza. Rispettivamente seguendo il criterio precedentemente descritto si può affermare:

- IBk: UnderSampling & BestFirst
- NaiveBayes: UnderSampling & BestFirst
- RandomForest: UndeSampling & BestFirst

Grafico 4 Precision-BK

Il caso della precision si presenta leggermente differente e più complesso da analizzare richiedendo un'osservazione più sottile. Non ci sono dubbi che il classificatore IBk abbia l anmigliore accuratezza in termini di valori assunti e di variabilita con l'OverSampling e Nessuna Features Selection, il caso invece di Naive Bayes porta un'interessante riflessione di sommaria equivalenza dei balancer nelle rispettive features selection contando i parametri però fin qui attuati è logico pensare che la migliore coppia sia rappresentata da OverSampling e BestFirst mentre è lapalissiana la scelta di BestFirst e Oversmapling per Random Forest.

Grafico 5 Recall-BK

La scelta per la metrica di Recall merita una brevissima trattazione nel caso di IBk è anche in questo caso lampante la scelta di OverSampling e BestFirst mentre è molto interessante osservare i valori relativi Naive Bayes ove una sostanziale equivalenza domina i dati senza Feature Selection e dove la SMOTE ottiene il primato per valore del quantile 2 ovvero la mediana idem dicasi per Random Forest.

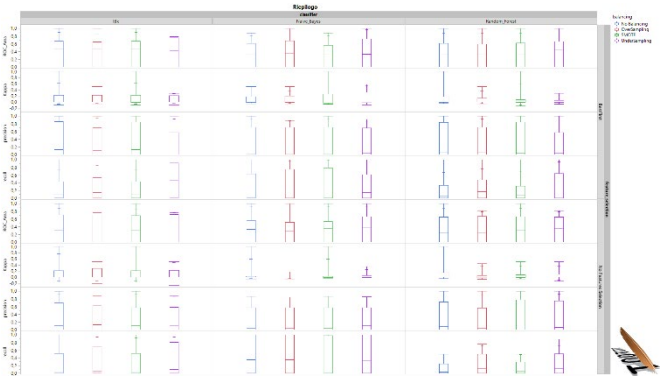

Grafico 7 Riepilogo delle Metriche

Il confronto tra Tajo e BookKeeper verrà svolto al termine della discussione delle singole metriche del proggetto Tajo.

Tajo

Similmente a quanto svolto per BookKeeper si analizzano ora le principali metriche di adeguatezza per poi estrarre al termine del confronto successivo con BookKeeper le rispettive combinazioni di FeatureSelection Balancing e Classificatori.

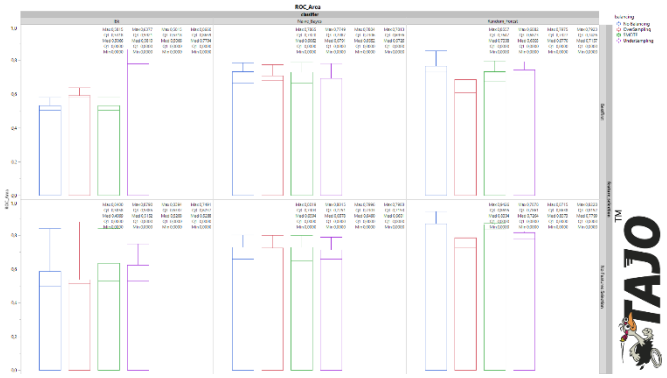

Grafico 6 AUC-TJ

Le riflessioni sono analoghe a quanto espresso in precedenza per le medesime metriche nel progetto di BookKeeper. Dal Grafico 5 AUC-TJ si evvince che per il classificatore IBk, dando precedenza ad un armonizzazione tra valore del secondo quantile e variabilità del campione, il migliore risultato è conseguito dalla copia Smote & BestFirst, nel caso di NaiveBayes invece la coppia UnderSampling & BestFirst ottiene il risultato più omogeno un improvement infine è dato dal classficatore RandomForest con la combinazione OverSampling e BestFirst.

Grafico 8 Kappa-TJ

È possibile notare nel Grafico 8 Kappa-TJ che le coppie che massimizzano la precisione siano rispettivamente:

- IBk: OverSampling & No Feature Selection
- Naïve Bayes: Undersampling & No Feature Selection
- Su Random Forest andrebbe premesso che lo scopo della presente trattazione è di trovare le coppie che minimizzino l'errore quindi facile sarebbe identificatre le coppie BestFit & (Smote || OverSampling) ma all'onor del vero siamo interessatri, tutto sommato, anche ai valori assuntidalle realizzazioni per questo sceglierei come compromesso migliore UnderSampli & BestFirst

Grafico 9 Precision - TJ

Certamente metrica di nostro interesse, la precision si presenta in un'instanza particolarmente semplice da analizzare:

- IBk: UnderSampling & BestFirst
- Naive Bayes: OverSampling & No Feature Selection
- Random Forest:UnderSampling & BestFirst

Grafico 10 Recall-TJ

Infine, concludiamo la trattazione delle singole metriche analizzando la Recall per Tajo. Si noti la grande variabilità della tecnica di balancing UnderSampling su questa instanza di metrica, in effetti questa eccessiva variabilità consente di scegliere in tutti i casi l'OverSampling e rispettivaemnte:

- IBk: OverSampling & Best Fisrt
- Naive Bayes: OverSampling & BestFirst
- RandomForest: OverSampling & BestFirst

In questa particolare istanza in effetti la risposta è stata univoca data la particolare distribuzione del campione e l'alta varianza dell'UnderSampling.

Riepilogo e confronto tra progetti

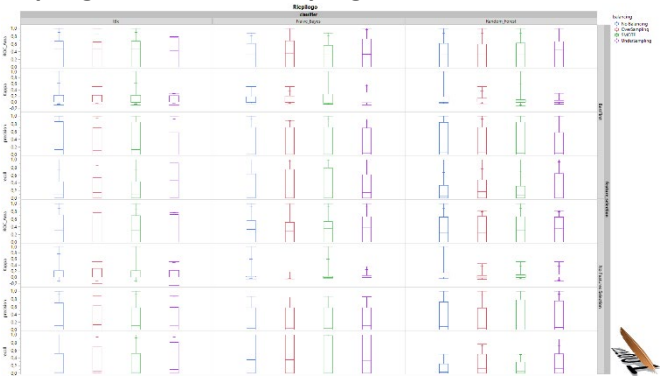

Grafico 11 Riepilogo BK

Il Grafico 11 e la Tabella 4 sintetizzano in un'unica soluzione i valori assunti dalle varie metriche in relazione alla Feature Selection, alla Balancing ed in ultima istanza divise per classificatori. Si noti che, come ci si aspettava, non si è potuta sollevare una coppia di tecniche nettamente superiore alle altre anche se un trend sembra esistere per il dataset corrente. Con riferimento alla Tabella 4 da un puro ragionamento di quorum si possono estrarre in alcuni casi delle vere e proprie coppie che hanno riportato i valori migliori sull'attuale dataset mentre nella maggior parte dei casi questo incremento non è imputabile ad entrambi gli strumenti di balancing o di feature selection pertanto è stata riportata solo una delle due tecniche dominanti.

Metrica	IBk	Naive Bayes	Random Forest	Tecnica Dominante per metrica
AUC	US & BF	NB & BF	US & NFS	Under Sapling
Kappa	US & BF	US & NFS	US & BF	Under Sampling & Best First
Precision	OS & NFS	OS & BF	OS & BF	Over Sampling & Best First
Recall	OS & BF	SM & NFS	SM & NFS	Smote & No Feature Selection
Tecnica Dominante per classificatore	Best First	Nessuna Tecnica	Nessuna Tecnica	

Tabella 4 Riepilogo Tabellare BookKeeper

In effetti è possibile notare una certa tendenza per le prime due metriche all'uso dell'UnderSampling mentre la preicsion e la Recal anche per la loro stess natura si sono identificate in due coppie completamente differenti. Mentre per il classificatore ha ragiunto un quorum la sola tecnica di Feature Selection BestFirst in exequo come balancing Over Sampling ed Under sampling. Nel caso del dataset di BookKeeper è possibile quindi in media affermare che la coppia di Under Sampling e Best First sembrebbero dominare lo scenario qualitativo del grafico e dei dati uniti ai Classificatori quasi in exequo di IBk e Random Forest, portando in realtà ad IBk una stabilità ed un'accuratteza leggermente superiore a Random Forest il quale, similmente a Naive Bayes, ha dimostrato segni di instabilità. Medesima è la trattazione che ci accingiamo ad avviare per il progetto Tajo per poi concludere con un confronto progetto a progetto con il metodo classico e commenteremo quindi in ultima instanza la stabilità dei classificatori in relazione ai nuovi filtri

Grafico 12 Riepilogo Tajo

Dal Grafico 12 e dalla trattazione precedente è possibile inferrire la Tabella 5, dalla quale, a sua volta, è possibile ottenere una visione d'insieme e valutare, qualitativamente la migliore combinazione, in effetti la Best First in questa realizzazione domina sulla No Feature Selecion come altrì continua ad essere la tecnica di bilanciamento più accurata l'Under Sampling. Per concludere la trattazione e rispondere alla Research Question posta in essere manca solo sceglierel classsificatore che, con le scelta combinazione è stato il più stabile. Nel caso del presente dataset è molto curiosa la

Metrica	IBk	Naive Bayes	Random Forest	Tecnica Dominante per metrica
AUC	SM & BF	US &BF	OS & BF	Best First
Kappa	OS & NFS	US&NFS	US & BF	Under Sampling & No Feature Selection
Precision	US &BF	OS& NFS	US &BF	Best First
Recall	SM & BF	US &BF	OS & BF	Best First
Tecnica Dominante per classificatore	Best First	Nessuna Tecnica	Best First	

Tabella 5 Riepilogo Tabellare Tajo

situazione per cui sembrerebbe esserci un exequo tra IBk e Naive Bayes mentre Random Forest ha dato prova di una maggiore instabilità al contrario di quanto successo per BookKeeper. Nel caso di Tajo il in effetti più stabile però risulta essere IBk il quale confermandosi anche per Tajo pone in essere una nuova Research Question affrontabile separatamente: è un caso che IBk sia il classificatore più performante con il nuovo filtro del dataset?

Confronto tra i Filtri

Grafico 12 Confronto tra il nuovo filtro ed il metodo classico BK

Il Grafico 12 riassuntivo (presente anche nella repository per una consultazione in risoluzione maggiore) evidenzia come l'applicazione del nuovo filtro miglior i risultati ottenuti dall'elaborazione con weka dei medesimi dataset con l'unica differenza consistente nel nuovo filtro FE. Un esempio sono la diminuzione degli outliars e la repentina diminuzione della vairabilità del campione, esemplificative sono le istanze della precision, di AUC e della Recal, tutti risultati interessanti che aprono le porte ad una futura investigazione.

3. Acknowledgements

Si ringrazia il prof. Falessi per i consigli e l'incoraggiamento a portare avanti l'ipotesi del nuovo filtro. I loghi dei progetti sono di proprietà esclusiva dei rispettivi detentori dei diritti d'autore, il codice sorgente dei rpoegtti è rilasciato sotto l'Apache Software Foundation License. Il presente Report, i suoi dati ed i codici sorgenti allegati sono rilasciati in accordo alla MIT License.

4. References

[1] Matteo Esposito, Università degli Studi di Roma Tor Vergata

[2] https://issues.apache.org/jira/rest/api/2/search?jql=project=%22**EAGLE**22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR%22status%22=%22resolved%22)AND%22resolution%22=%22fixed%22&fields=key&page=[**pageNumber**]&maxResults=1000

[3] http://www.jsonschema2pojo.org/

[4] https://github.com/FasterXML/jackson

[5] https://github.com/google/gson

[6] https://it.wikipedia.org/wiki/Plain_Old_Java_Object

[7] https://www.spcforexcel.com/knowledge/control-chart-basics/control-chart-rules-interpretation

[8] https://github.com/dice-group/gerbil/wiki/Precision,-Recall-and-F1-measure