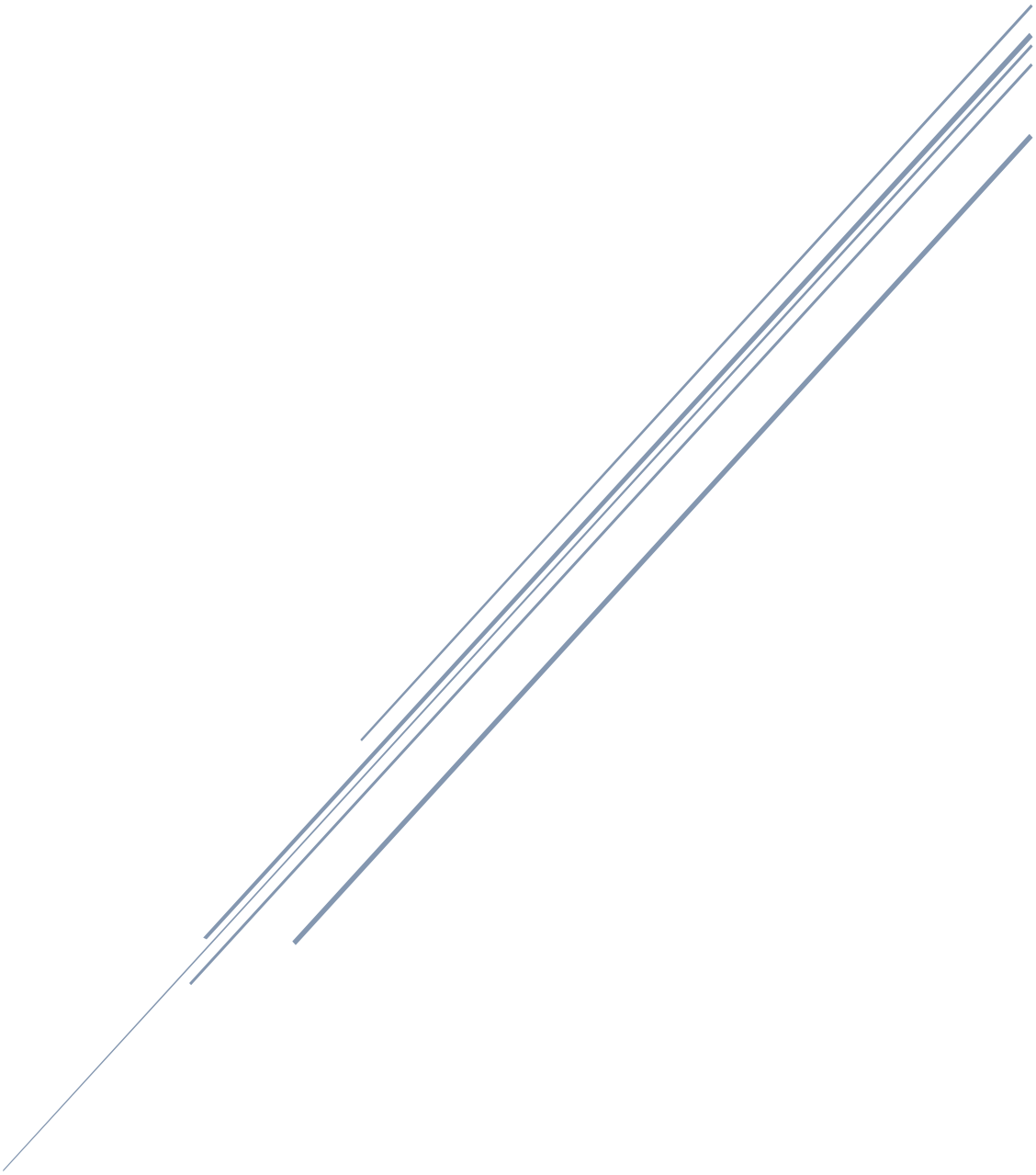
RELAZIONE PROGETTO ISW2

Process Control Chart & Analisi classificatori



Università di Roma TorVergata
Malavasi Alessio mat. 0287437

CONFIGURAZIONE

Entrambe i deliverable del progetto sono realizzati per poter analizzare in realtà qualsiasi progetto Apache e non solamente quelli analizzati nel seguito. Per fare ciò è necessario seguire dei semplici passi di configurazione del programma che riportiamo qui:

- *Nome progetto:* in tutte le classi Java presenti nel progetto è presente la variabile statica ***"PROJ"*** di tipo *String* tra le dichiarazioni delle variabili della classe (Fig.1). Questa stringa deve essere inizializzata con il nome del progetto che si vuole analizzare (es. bookkeeper,tajo,mahout,…).
- *File oauth.txt:* per effettuare le query a Github con un limite di richiese per ora più elevato è necessario creare dal proprio account un codice di autenticazione. Questo codice deve essere poi inserito all'intero di un file sul proprio computer (non nella cartella del progetto altrimenti Github non la accetta!) denominato *"oauth.txt"*. Una volta creato bisogna prendere il **path assoluto** di questo file ed inserirlo all'interno della variabile dichiarata all'inizio di ogni classe chiamata ***"OAUTH"*** (Fig.1).
- *Username Github:* per effettuare le query autenticate a Github c'è bisogno anche di inserire il proprio username Github. Questo dovrà essere inserito nell'apposita variabile chiamata ***"USERNAME"*** presente nella dichiarazione delle variabili del file java (Fig.1).

```
1. ...
2. public class TicketData {
3.
4.   private static final String PROJ = "mahout";
5.   private static final String OAUTH =
   "C:\\Users\\malav\\Desktop\\isw2\\oauth.txt";
6.   private static final String USERNAME = "malavasiale";
7.   private static final String DATA_COMMIT = "dataCommit.csv";
8.   private static final String FINAL_DATA = "finalData.csv";
9.
10.
11.  private static String readAll(Reader rd) throws IOException {
12.  ...
```

Fig.1 Variabili da modificare per configurazione

DELIVERABLE ONE

1. Introduzione

In questa sezione viene analizzata la stabilità di un attributo del progetto "***Apache Mahout***", in questo caso lo sviluppo di New Features. Per stabilità di un attributo si intende la misurazione del numero di features che sono state rilasciate in ogni mese nel periodo di attività del progetto e la verifica del livello di produttività, sopra o sotto la media, grazie alla realizzazione di un grafico chiamato **Process Control Chart** che verrà mostrato nei capitoli seguenti.

I dati necessari per effettuare queste analisi sono stati attinti da due fonti:

- La repository in Jira contenente i ticket del progetto Apache Mahout (https://issues.apache.org/jira/projects/MAHOUT/issues)
- La repository del progetto Apache Mahout su Github (https://github.com/apache/mahout)

Il download e l'elaborazione dei dati è avvenuto tramite codice Java con l'utilizzo dell'IDE Eclipse, mentre la realizzazione dei grafici tramite il software JMP.

2. Ottenere i dati

I dati necessari per la realizzazione del grafico sono stati recuperati in entrambi i casi tramite REST API messe a disposizione dalle due piattaforme sopra indicate. Tramite la formulazione di una query specifica, vengono restituiti uno o più file in formato JSON, successivamente tramite java analizzate ed estratte le sole informazioni necessarie.

Per prima cosa da Jira sono stati estratti tutti i ticket relativi alle New Feature con stato *"Resolution = fixed"* relative al progetto Mahout tramite la query in Fig.2

```
1.  https://issues.apache.org/jira/rest/api/2/search?jql=project=%22MAHOUT%22AND%22issueType%22=%22New%20Feature%22AND%22resolution%22=%22fixed%22&fields=key
```

Fig.2 Esempio query Jira per NewFeature con Resolution = fixed

L'output ottenuto da questa query per ogni ticket sarà di questo tipo:

```
1.  {"expand":"operations,versionedRepresentations,editmeta,changelog,renderedFields","id":"13009079","self":"https://issues.apache.org/jira/rest/api/2/issue/13009079","key":"MAHOUT-1883"}
```

Fig.3 Esempio di un JSONObject risultante dalla query in Fig.1

Una volta ottenuto il file JSON, per ogni elemento si seleziona il campo *"key"* ottenendo così una lista di tutti i Ticket ID di interesse per l'analisi in corso.

Si passa ora al download di tutti i Git Commits del progetto Mahout da Github tramite la query in Fig.4

```
1. https://api.github.com/repos/apache/mahout/commits
```

Fig.4 Esempio query a Github per tutti i commit relativi al progetto Mahout

Un esempio di JSONObject ottenuto dalla query è il seguente:


```
1. {
2.     "sha": "754068bc3ac62bcf4b9b656cd4a6f413b29b18df",
3.     "node_id":
   "MDY6Q29tbWl0MjAwODk4NTk6NzU0MDY4YmMzYWM2MmJjZjRiOWI2NTZjZDRhNmY0MTNiMjliM
   ThkZg==",
4.     "commit": {
5.       "author": {
6.         "name": "Andrew Palumbo",
7.         "email": "apalumbo@apache.org",
8.         "date": "2020-02-21T08:05:08Z"
9.       },
10.        "committer": {
11.          "name": "GitHub",
12.          "email": "noreply@github.com",
13.          "date": "2020-02-21T08:05:08Z"
14.        },
15.        "message": "[MAHOUT-2088] Update Apache parent pom to latest
   version (23) from 18 (#392)\n\n* [MAHOUT-2088] Update Apache parent pom to
   latest version (23) from 18\r\n\r\n* [MAHOUT-2088]update Apache Parent pom
   to latest version (23),prepare for RC5: reset to 14.1-SNAPSHOT\r\n\r\n*
   [MAHOUT-2088] update Apache Parent pom to latest version (23), tweaks",
16.        "tree": {
17.  …
18.  …
```


Fig.5 Esempio di output della query in Fig.4

Nell' output in Fig.5 sono di interesse solo le due parti evidenziate in rosso, ovvero:

- *Date:* che da l'informazione sulla data di esecuzione del commit
- *Message:* si può notare come all'inizio del messaggio di Git Commit sia presente proprio il TicketID del relativo ticket risolto su Jira. Questo ci permette di filtrare tra tutti i commit scaricati soltanto quelli relativi ai dati di interesse.

Tutti questi dati vengono poi salvati su un file CSV *("dataCommit.csv")* in modo tale che non dovranno essere scaricati ad ogni esecuzione dell'applicazione.

Un problema infatti riscontrato nella realizzazione è il ***Rate Limit per hour*** che Github impone sulle query. Nell'implementazione si sarebbe potuta fare una query più specifica per recuperare i Git Commits, facendo restituire a Github solo quelli di interesse. Si è scelto invece di non intraprendere questa soluzione perché le query più specifiche (*Search query)* hanno un ***Rate Limite per minute*** pari 20 al minuto, a differenza delle query semplici che sono limitate a 5000 l'ora. Si è ritenuto più efficiente scaricare anche commit superflui molto più velocemente e filtrarli successivamente in locale.

3. Elaborazione dati

Una volta completato l'ottenimento dei dati, occorre elaborarli per ottenere le informazioni di interesse. In questo caso ciò che è stato fatto può essere riassunto in questi passi:

- Ricerca nel file *"dataCommit.csv"* nella sezione *Messaggio* la corrispondenza con i TicketID restituiti da Jira. Come possiamo vedere in Fig.5 infatti ogni messaggio dovrebbe iniziare proprio con l'ID del ticket a cui si riferisce (ad esempio "*[MAHOUT-1804]"* oppure *"MAHOUT-1804:"*)
- Una volta selezionati i commit corrispondenti alle New Features, si prende di ciascuno la data in cui è stato effettuato (in formato year-month) e si va a contare all'interno del periodo di attività il numero di commit suddivisi per mese.
- Calcolo la media su tutti i valori ottenuti: $\frac{1}{n}\sum_1^n x_i = 0.798$
- Utilizzando media e deviazione standard (std) calcolo l'***Upper Limit*** ovvero un valore che mi serve a identificare nel grafico periodi particolarmente produttivi:

  $$mean + 3 * std(x) = 4.733$$

  Calcolo anche il ***Lower Limit*** utile invece per identificare i periodi di scarsa produttività. In questo caso però risulta essere negativo e quindi non verrà riportato nel grafico in quanto mesi in cui ci sono zero New Features verranno considerati tali:

  $$mean - 3 * std(x) = -3.137$$
- Infine, tutti i dati vengono salvati su un file di output chiamato *"finalData.csv"* dal quale ricaviamo il grafico riportato in Fig.6

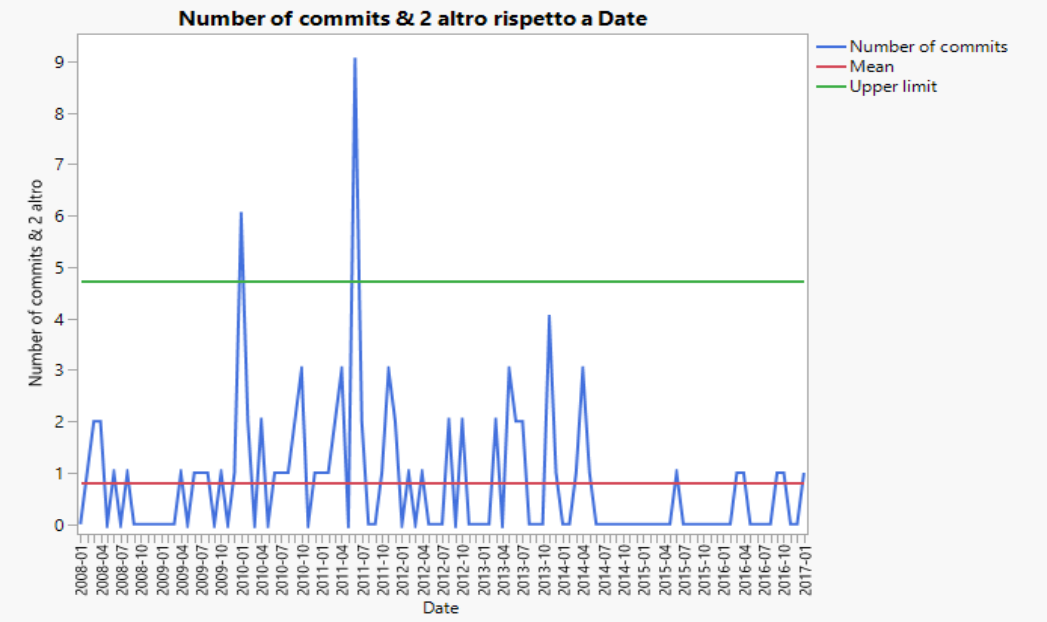


Fig.6 Process Control Chart del progetto Mahout per le NewFeatures

4. Analisi risultati

Il grafico in Fig.6 è realizzato mettendo sull'asse delle ascisse il campo *Date* per indicare il mese preso in considerazione, mentre sull'asse delle ordinate il campo *Number of commits* per indicare il numero di commit realizzati in uno specifico mese.

La linea orizzontale rossa rappresenta la *Media* sul periodo di attività mentre quella verde rappresenta l'*Upper Limit* descritto nel paragrafo precedente. Il *Lower Limit* non è presente in quanto negativo e quindi non rappresentativo di per periodi di poca attività.

Descriviamo in seguito le osservazioni che possiamo trarre dai dati:

- *Periodo di attività*: è un periodo molto lungo e dura circa 9 anni (da gennaio 2008 a gennaio 2017)
- *Maggiore attività*: il periodo che rileva un maggior numero di commit è più o meno nel periodo intermedio di attività, ovvero da gennaio 2010 ad aprile 2014
- *Minore attività:* il periodo che rileva un minore numero di commit si trova verso la fine del periodo di attività, da maggio 2014 con diversi mesi a quota zero.
- *Picchi sopra upper limit:* ci sono solamente due mesi in cui i commit hanno superato il numero di upper limit (pari a circa 5 commit) e sono **gennaio 2010** (6 commit) e **maggio 2011** (9 commit). Possiamo notare che sono periodi molto intermedi all'interno del periodo di maggiore attività.

5. Conclusioni e soluzioni

Dalle osservazioni riguardanti il periodo di attività possiamo trarre come conclusione che il progetto era attivo ed anche molto seguito ed aggiornato negli anni che vanno dal 2010 al 2014, dopodiché il supporto nello sviluppo di New Features è andato ad arrestarsi. Proprio gli ultimi anni, che anche se di poca attività sono stati presi in considerazione, hanno portato ad un abbassamento della media che altrimenti sarebbe stata decisamente più elevata. Probabilmente il progetto è stato concluso o abbandonato, o più semplicemente è terminato il rilascio di nuove features.

Un'ultima osservazione riguarda anche il numero di ticket risolti in Jira che non hanno un loro corrispettivo commit su Git. Questo è dovuto a un'assenza di standardizzazione che si può notare nella scrittura dei TicketID nei messaggi di commit di Git, che porta così ad una perdita dei dati di circa il 21% rispetto ai dati provenienti da Jira come mostrato in Fig.7. Contando anche questi commit mancanti verrebbe quindi una media pari a $\boldsymbol{mean_{corrected} = 0.965}$ che comunque non è molto più alta della precedente in quanto già molto bassa.

```
1. giu 02, 2020 5:34:23 PM deliverableone.TicketData main
2. INFO: Starting processing data
3. giu 02, 2020 5:34:26 PM deliverableone.TicketData rawDataParser
4. INFO: Data not found: 21.37404580152672 %
```

Fig.7 Percentuale ticket Jira non trovati in Git

Una soluzione per una stima migliore sarebbe creare uno standard per la scrittura dei messaggi di commit su Git così da risolvere le differenze almeno all'interno dello stesso progetto o ancor meglio tra tutti i progetti.

6. Link SonarCloud

È riportato qui il link di SonarCloud della prima deliverable con zero Code Smells:

https://sonarcloud.io/dashboard?id=malavasiale_iswproject

DELIVERABLE TWO

1. Introduzione

Lo scopo di questa seconda parte è quello di fare una stima dei migliori modelli predittivi per poter predire se una classe di un progetto (in questo caso ***"Apache Bookkeeper"*** e ***"Apache Tajo"***) conterrà dei bug o meno e quindi dare all'attributo "*buggy*" della classe un valore tra {yes,no}.Si tratta quindi di una classificazione binaria ed in generale questa viene effettuata estraendo informazioni relative a delle metriche delle classi durante le varie versioni del progetto e dividendo poi i dati in due parti. Il ***training set***, che serve ad addestrare il modello, ed il ***testing set***, che il modello cercherà di classificare con i dati in suo possesso per poi verificare la correttezza del suo testing tramite varie metriche (precision,recall,auc,kappa,…). Per generare i set viene usata la tecnica di validazione ***Walk forward*** in quanto i dati dovrebbero essere dipendenti dal tempo, ma viene come controprova eseguita anche la ***K-fold cross validation***.

Vengono fatte inoltre delle prove anche modificando il training set con tecniche di ***feature selection*** (in questo caso *Filter*) e di ***sampling*** (*undersampling*, *oversampling* e *SMOTE*).

Tutte queste tecniche sono effettuate utilizzando tre classificatori diversi

- *Random Forest*
- *Naive Bayes*
- *IBk*

Tutto questo lavoro è reso possibile grazie all'utilizzo di **WEKA**, software open source di machine learing, sottoforma di Java API. Le metriche, analizzate nelle sezioni successive per verificare quale tecnica o quale classificatore si comporta meglio, sono le seguenti:

- *Precision:* quanto è stato accurato il modello a scegliere positivi senza prendere falsi positivi **($Precision = \frac{TP}{TP+FP}$).**
- *Recall:* quanto è stato accurato il modello nel trovare i veri positivi presenti **($Recall = \frac{TP}{TP+FN}$).**
- *Kappa:* di quante volte il modello è stato più accurato rispetto a uno dummy.
- *AUC:* probabilità che un'istanza casualmente scelta positiva sia classificata al di sopra di una casualmente scelta negativa.

I dati necessari sono invece ricavati dalle rispettive repository dei due progetti su i seguenti siti:

- Le repository in Jira contenente i ticket dei progetti (https://issues.apache.org/jira/projects/BOOKKEEPER/issues) (https://issues.apache.org/jira/projects/TAJO/issues)

- La repository dei progetti su Github (https://github.com/apache/bookkeeper) (https://github.com/apache/tajo)

2. Ottenere i dati

I dati per la realizzazione dei grafici sono ottenuti tramite le REST API di Jira e di Github restituiti in formato JSON tramite la costruzione di specifiche query. Per prima cosa in questo caso otteniamo alcune informazioni da Jira tramite la query in Fig.8

```
1. String url = "https://issues.apache.org/jira/rest/api/2/project/" +
   projName;
```

Fig.8 Query a Jira per ottenere le release del progetto

Tramite questa query (eseguita ed elaborata nella classe *"GetReleaseInfo.java"*) salviamo sul file chiamato *"NOME_PROGETTOVersionInfo.csv"* le informazioni relative a tutte le release del progetto assegnando ad ognuna di queste un ID in base ad un ordine cronologico crescente. Questo ci aiuterà in seguito nel capire le classi affette da bug all'interno di ciascuna release.

Dopo questo passaggio, sempre da Jira, andiamo a prendere i ticket di tipo bug con stato *"Resolution = Fixed"* tramite la query in Fig.9

```
1. String url =
   "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
2.                                    + PROJ_NAME +
   "%22AND%22issueType%22=%22Bug%22AND%22resolution%22=%22fixed%22&fields=key
   ,fixVersions,versions,created";
```

Fig.9 Query a Jira per la richiesta di dati per i ticket di tipo bug

Dal file JSON che viene restituito estraiamo i seguenti dati:

- *TicketID:* l'id del ticket di tipo Bug presenti in Jira che sono stati risolti.
- *Created:* data di creazione del ticket su Jira, necessario per capire quale è l'Observed Version del bug, ovvero la release in cui è stato scoperto.
- *Affected Versions (AV):* le versioni in cui il bug è stato introdotto per la prima volta, può anche non essere presente in quanto è lo sviluppatore che deve inserirla manualmente e possono essere anche più di una.
- *Fixed Versions (FV):* versione in cui il bug è stato risolto. Anche questa inserita manualmente dal programmatore e dovrebbe essere sempre presente (a volte purtroppo non lo è).

Dopodichè dalle AV prendiamo in considerazione (se presente) solamente quella più vecchia in quanto questa è l'***Injected Version (IV)***, ovvero la prima release nella quale il bug è stato introdotto.

Tramite un'altra query (la stessa in Fig.4), questa volta usando le REST API di Github, ricaviamo tutti i commit con TicketID nel campo del messaggio corrispondenti a quelli scaricati da Jira. Da

questi ricaviamo il codice identificativo chiamato *"sha"* che ci servirà per ricavare informazioni più approfondite di ciascun commit usando la query in Fig.10

```
1. String url = "https://api.github.com/repos/apache/"+PROJ+"/commits/" +
   sha;
```

Fig.10 Query per ricevere informazioni più specifiche su ogni commit

In Fig.11 sono evidenziati i campi di nostro interesse per ogni commit:

```
1. {
2.   "sha": "a9e604525f4a784133d23666e94d16f101749197",
3.   "node_id":
   "MDY6Q29tbWl0MTU3NTk1NjphOWU2MDQ1MjVmNGE3ODQxMzNkMjM2NjZlOTRkMTZmMTAxNzQ5M
   Tk3",
4.   "commit": {
5. ...
6. ...
7. "files": [
8.     {
9.       "sha": "546474577451cfc718b980f04bff54948d814fac",
10.        "filename": "bookkeeper-
   server/src/main/java/org/apache/bookkeeper/bookie/BookieStateManager.java"
   ,
11.        "status": "modified",
12.        "additions": 1,
13.        "deletions": 1,
14.        "changes": 2,
15.        "blob_url":
   "https://github.com/apache/bookkeeper/blob/a9e604525f4a784133d23666e94d16f
   101749197/bookkeeper-
   server/src/main/java/org/apache/bookkeeper/bookie/BookieStateManager.java"
   ,
16.        "raw_url":
   "https://github.com/apache/bookkeeper/raw/a9e604525f4a784133d23666e94d16f1
   01749197/bookkeeper-
   server/src/main/java/org/apache/bookkeeper/bookie/BookieStateManager.java"
   ,
17.        "contents_url":
   "https://api.github.com/repos/apache/bookkeeper/contents/bookkeeper-
   server/src/main/java/org/apache/bookkeeper/bookie/BookieStateManager.java?
   ref=a9e604525f4a784133d23666e94d16f101749197",
18.        "patch": "@@ -229,7 +229,7 @@ public Void call() throws
   IOException {\n
   shutdownHandler.shutdown(ExitCode.ZK_REG_FAIL);\n                    }\n
   }\n-                return (Void) null;\n+                return null;\n
   }\n         });\n     }"
19.     },
20.     {...
```

Fig.11 Parte del file JSON restituito dalla query

- *Files -> filename:* questo array contiene tutti i files che sono stati modificati relativi a un singolo commit in git. Prendiamo in considerazione esclusivamente quelli con estensione *.java*.
- *Additions:* numero di linee di codice aggiunte in un determinato file nel commit considerato.

- *Deletions:* numero di linee di codice eliminate in un determinato file nel commit considerato.
- *Number of files:* numero di file contenuti nell'array Files.

3. Elaborazione dati

Una volta ottenuti i dati precedenti, possiamo passare all'elaborazione per ottenere le metriche di nostro interesse. Per prima cosa si eliminano i commit relativi all'ultima metà delle versioni del progetto. Questo perché in media i bug rimangono dormienti per in media l metà delle release, quindi questa mancanza falserebbe le statistiche finali.

Per prima cosa ora andiamo ad analizzare quali file del progetto contengono un bug per ogni release. Per fare ciò sfruttiamo le AV e FV nel caso queste siano presenti per determinare l'attributo *Buggy* di una classe. In caso però le AV non siano presenti, il nostro scopo è stimarla per poter comunque assegnare l'attributo di nostro interesse. Per stimare l'AV utilizziamo una tecnica chiamata ***Incremental Proportion*** che si basa su una certa proporzionalità tra AV-OV e OV-FV:

i. Man mano che scorriamo i ticket che hanno una AV, andiamo ad utilizzare la seguente formula per calcolare il parametro $\boldsymbol{P = \frac{FV-IV}{FV-OV}}$
ii. Eseguiamo una media su P per ogni classe su cui viene calcolato
iii. Una volta che siamo arrivati ad una classe che non ha una IV, possiamo predirla utilizzando la formula $\boldsymbol{IV_{predicted} = FV - (FV - OV) * P}$

In seguito con le altre informazioni che abbiamo possiamo andare a calcolare le diverse metriche di cui abbiamo bisogno. In questo caso le metriche considerate sono dieci:

- *LOC Touched:* somma del numero di linee di codice aggiunte o cancellate in un file durante un'intera release.
- *LOC Added:* somma del numero di linee di codice aggiunte in un file durante un intera release.
- *MAXLOC Added:* numero massimo di linee di codice aggiunte ad un file durante una release.
- *AVGLOC Added:* numero medio di linee di codice aggiunte ad un file durante una release.
- *Churn:* somma del numero di linee di codice aggiunte meno quelle cancellate in un file durante una release.
- *MAX churn:* numero massimo delle linee di codice aggiunte meno quelle cancellate in un file durante una release.
- *AVG churn:* media delle linee di codice aggiunte meno quelle cancellate in un file durante una release.
- *Change Setsize:* somma del numero di file in commit insieme a un determinato file in una release.
- *MAXChange Setsize:* massimo numero di file in commit insieme a un determinato file in una release.

- *AVGChange_Setsize:* media del numero di file in commit insieme a un determinato file in una release.

1	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
2	bookkeepe	48	40	40	40	32	32	32	3	3	3	no
3	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
4	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
5	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
6	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
7	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
1	bookkeepe	1	1	1	1	1	1	1	12	12	12	no
2	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
3	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
4	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
5	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
6	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
7	bookkeepe	6	4	4	4	2	2	2	45	45	45	no
1	bookkeepe	1	1	1	1	1	1	1	12	12	12	no
2	bookkeepe	3	2	2	2	1	1	1	8	8	8	no
3	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
4	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
5	bookkeepe	0	0	0	0	0	0	0	0	0	0	no
6	bookkeepe	0	0	0	0	0	0	0	0	0	0	yes
7	bookkeepe	10	6	5	3	2	2	1	47	45	23	no

Fig.12 Esempio di output finale

4. Weka Testing

Una volta ottenuto il file con tutte le metriche il nostro obiettivo è quello di addestrare dei modelli che ci possano classificare delle classi di testing come contenenti dei bug o meno, ed analizzare le metriche di accuratezza risultanti. Innanzitutto, dividiamo l'insieme dei nostri dati in *Training Set* e *Testing set*.

La tecnica utilizzata per effettuare questa divisione è quella del ***Walk forward***: si mette, iniziando dalla prima una versione nel testing set e tutte le precedenti nel training set, e si ripete l'operazione per ogni versione. Questa tecnica viene utilizzata perché tiene conto del fatto che le varie versioni possano avere metriche in qualche modo dipendenti dal tempo, in modo da non mettere mai in training delle versioni successive a quella di testing.

Un'altra tecnica utilizzata in questa fare è quella del ***Feature Selection:*** nelle metriche elaborate nel dataset, non è detto che tutte siano rilevanti nella classificazione di una classe. Feature selection si occupa infatti di prendere dal dataset solamente gli attributi che ritiene utili e di fare training esclusivamente su quelli.

Ultima tecnica utilizzata per migliorare la classificazione è il ***Sampling:***

- *Under sampling:* consiste nell'eliminare alcune istanze della classe maggiormente presente nel training set per renderle uguali in numero all'altra.
- *Over Sampling*: consiste nel duplicare alcune istanze della classe presente in numero minore nel training set in modo da eguagliarla in numero all'altra.

- *SMOTE*: consiste nel creare, tramite un algoritmo, delle istanze fittizie della classe meno presente nel training set così da eguagliarla in numero all'altra. E' essenzialmente una tecnica di over sampling più raffinata.

A questo punto vengono mostrati tramite i grafici i risultati ottenuti dall'incrocio nell'utilizzo delle varie tecniche.

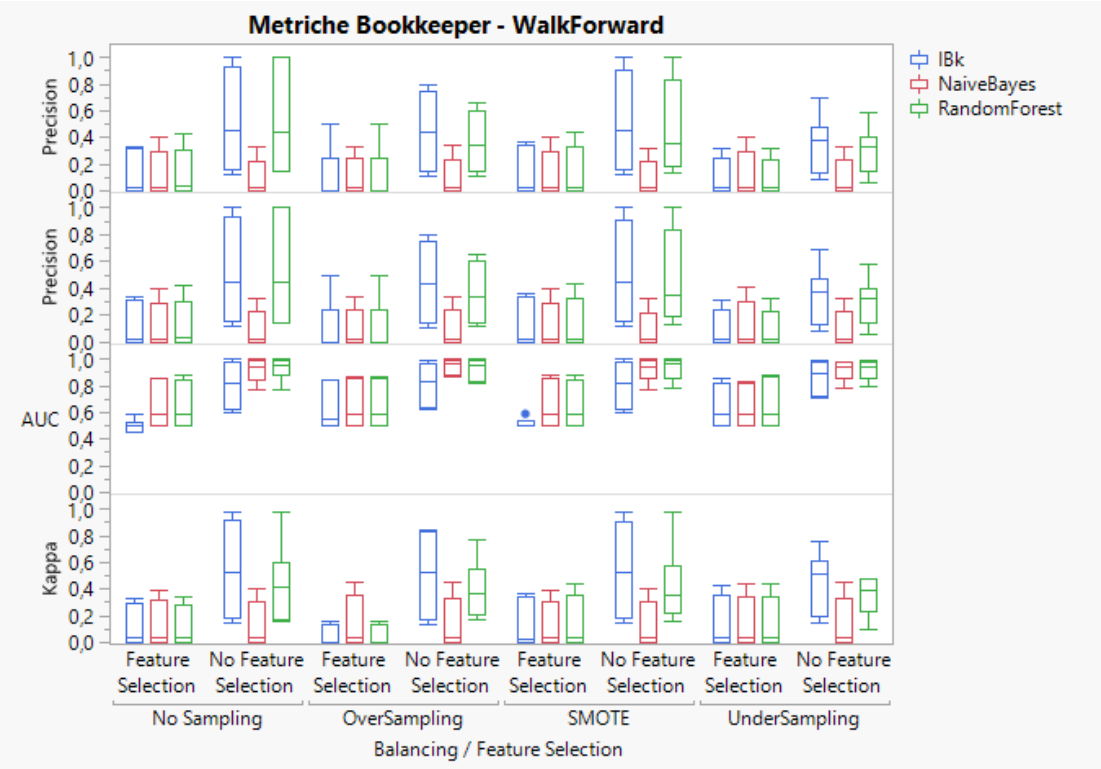


Fig.13 Metriche riguardanti il progetto bookkeeper

Fig.14 Metriche riguardanti il progetto tajo

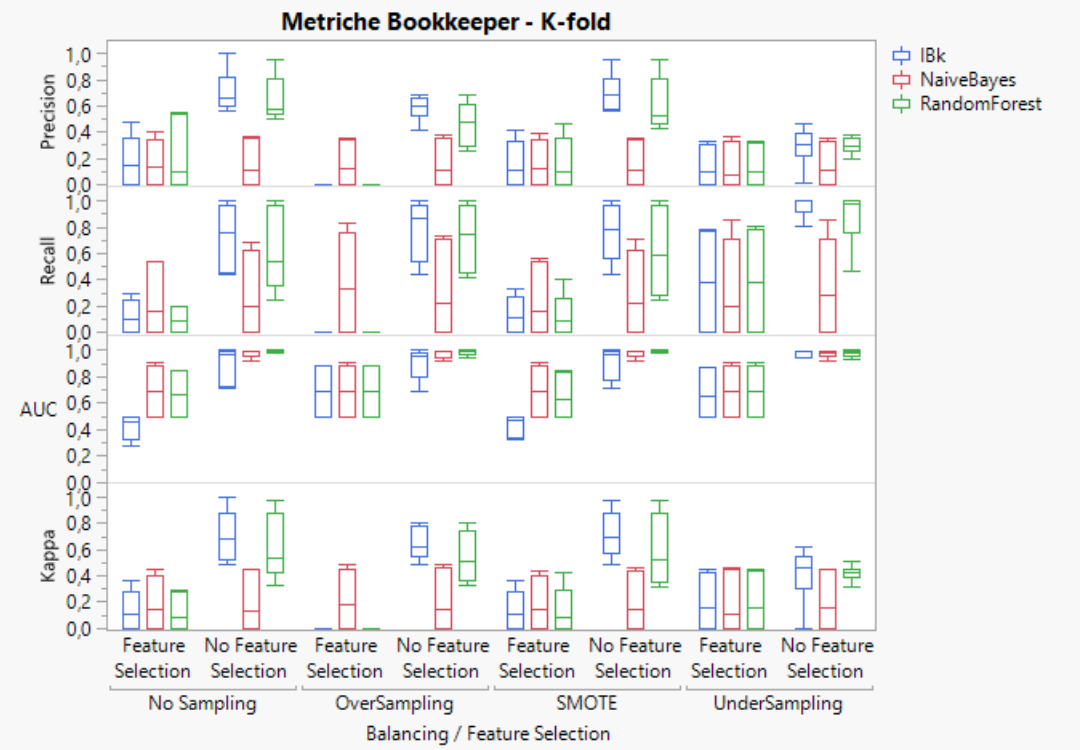


Fig.15 Metriche bookkeeper con K-fold

Fig.16 Metriche Tajo con K-fold

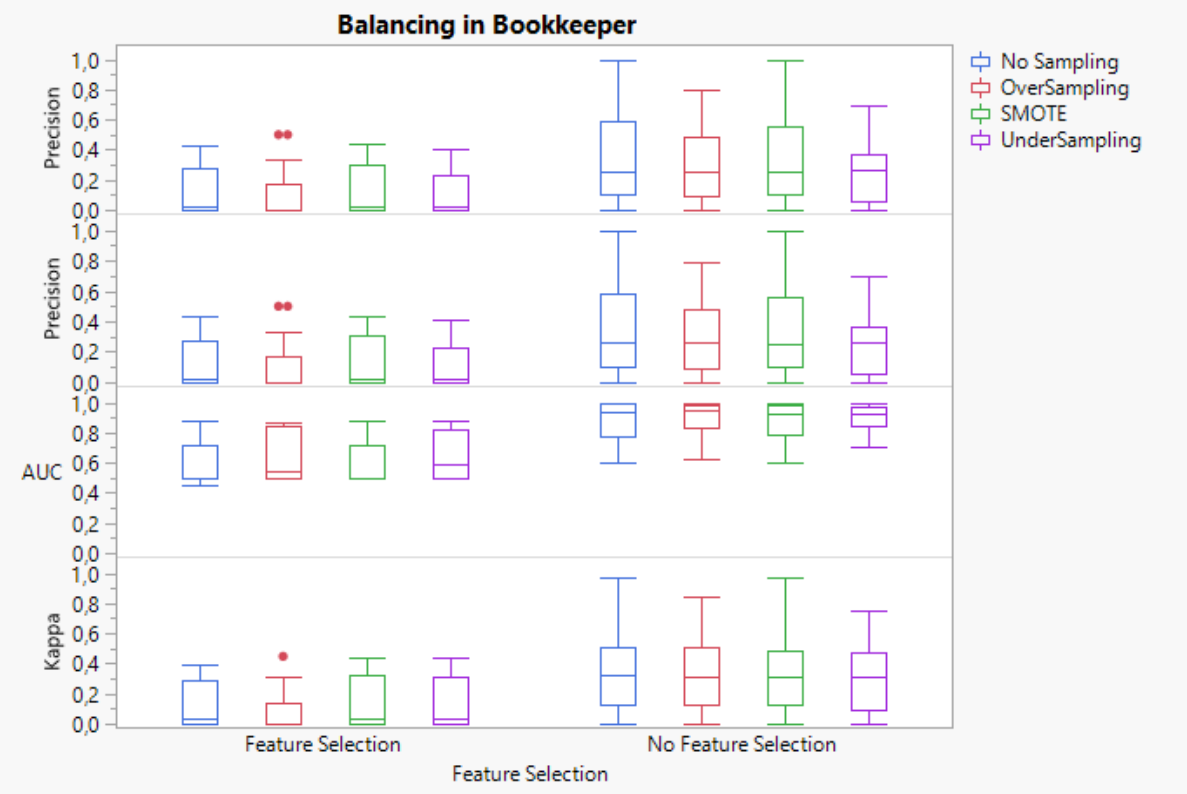


Fig.17 Metriche Bookkeeper miglior sampling

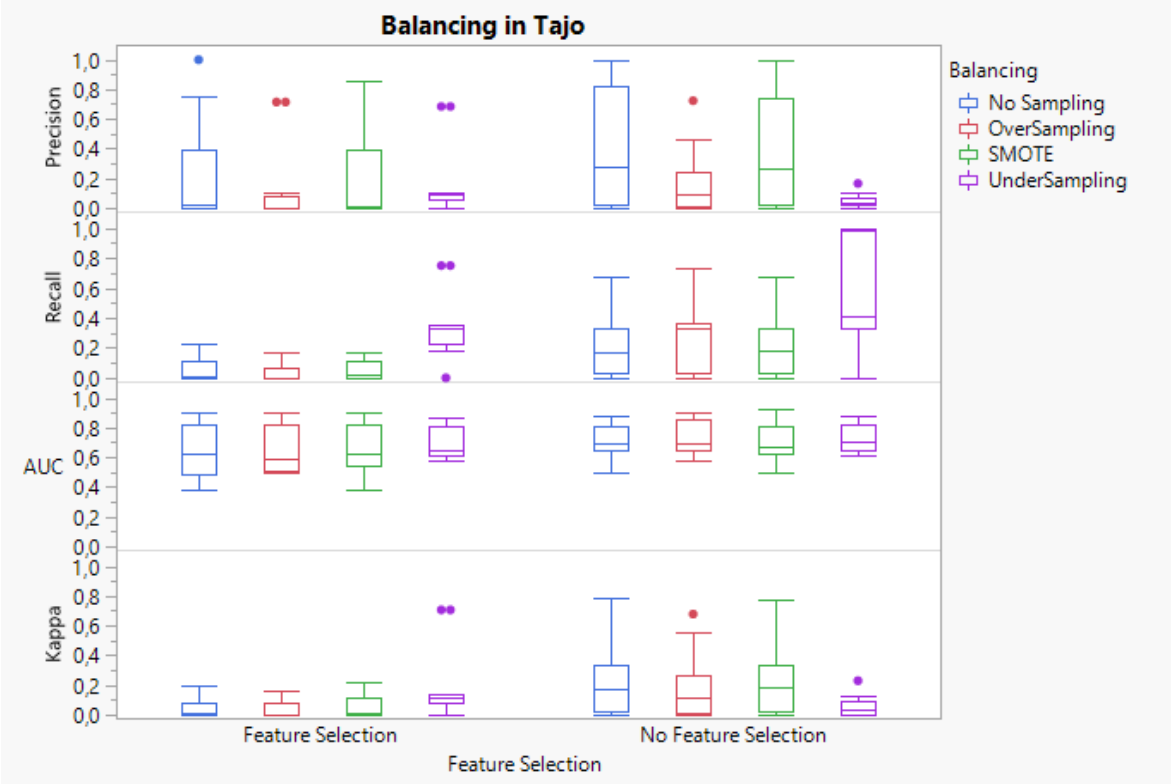


Fig.18 Metriche Tajo miglior sampling

5. Analisi e conclusioni

I grafici in Fig.17 e Fig.18 mostrano le metriche risultanti dall'analisi dei progetti Tajo e Bookkeeper.

Le 4 metriche (precision, recall, AUC e kappa) in generale più alte sono e migliore è il modello utilizzato. Questi due grafici ci evidenziano, in linea generale e senza tener conto del numero di release e del classificatore scelto, quali solo le tecniche di sampling più efficienti nei due progetti:

- *Miglior sampling in Bookkeeper:* in questo caso si può notare come le tecniche adottate sicuramente danno risultati migliori senza l'utilizzo di feature selection. In questo ambito le tecniche di sampling non danno risultati medi eccessivamente diversi, ma comunque le migliori sono **No Sampling** e **SMOTE** perché danno i valori massimi più elevati.
- *Miglior sampling in Tajo:* a prima vista anche in questo progetto è visibile la migliore efficienza delle tecniche di sampling quando non si effettua feature selection. In questo progetto però i valori medi delle varie metriche sono più distanti e si può vedere in maniera più netta la migliore efficienza di **No Sampling** e **SMOTE**. Unica eccezione è il valore della recall, che stranamente risulta in media più alto con le altre due tecniche di sampling.

Possiamo quindi notare come i risultati sulle migliori tecniche di sampling siano concordi tra i due progetti.

Per il progetto ***Bookkeeper*** (Fig.13) possiamo trarre ulteriori risultati:

- *No feature selection:* queste tecniche portano in qualsiasi caso le metriche ad abbassarsi. Il motivo probabilmente risiede nella scelta delle metriche calcolate per il dataset. Essendo tutte utili al fine del modello, quando le si prova a ridurre non si causa altro che una perdita di informazione.
- *Sampling peggiore:* la tecnica di sampling che porta risultati peggiori è quella di Under Sampling. Diminuendo di molto la grandezza del training set ci si poteva aspettare un peggioramento dei risultati.
- *Classificatore migliore:* il migliore classificatore risulta essere ***IBk*** con utilizzo di Oversampling oppure SMOTE.

Anche per il progetto ***Tajo*** (Fig.14) possiamo trarre ulteriori risultati:

- *No feature selection:* anche in questo caso la feature selection porta prestazioni peggiori per lo stesso motivo descritto precedentemente.
- *Sampling peggiore:* anche in questo caso la tecnica di Under Sampling porta a un peggioramento dei risultati.
- *Classificatore migliore:* il miglior classificatore sembra essere ***Random Forest***, anche se ***IBk*** fornisce comunque risultati simili.

Un'altra analisi da effettuare può essere quella di andare a visualizzare l'andamento delle metriche rispetto al numero di release inserite nel training.

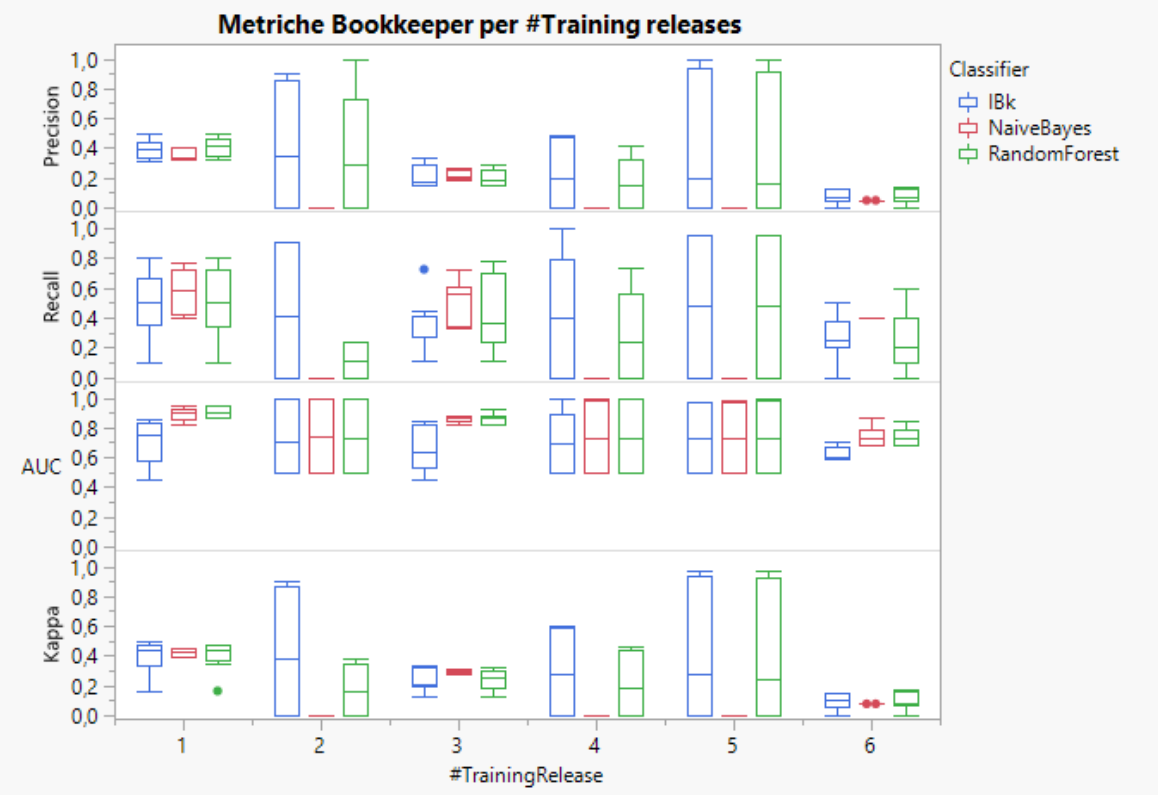


Fig.19 Metriche Bookkeeper rispetto al numero di training release

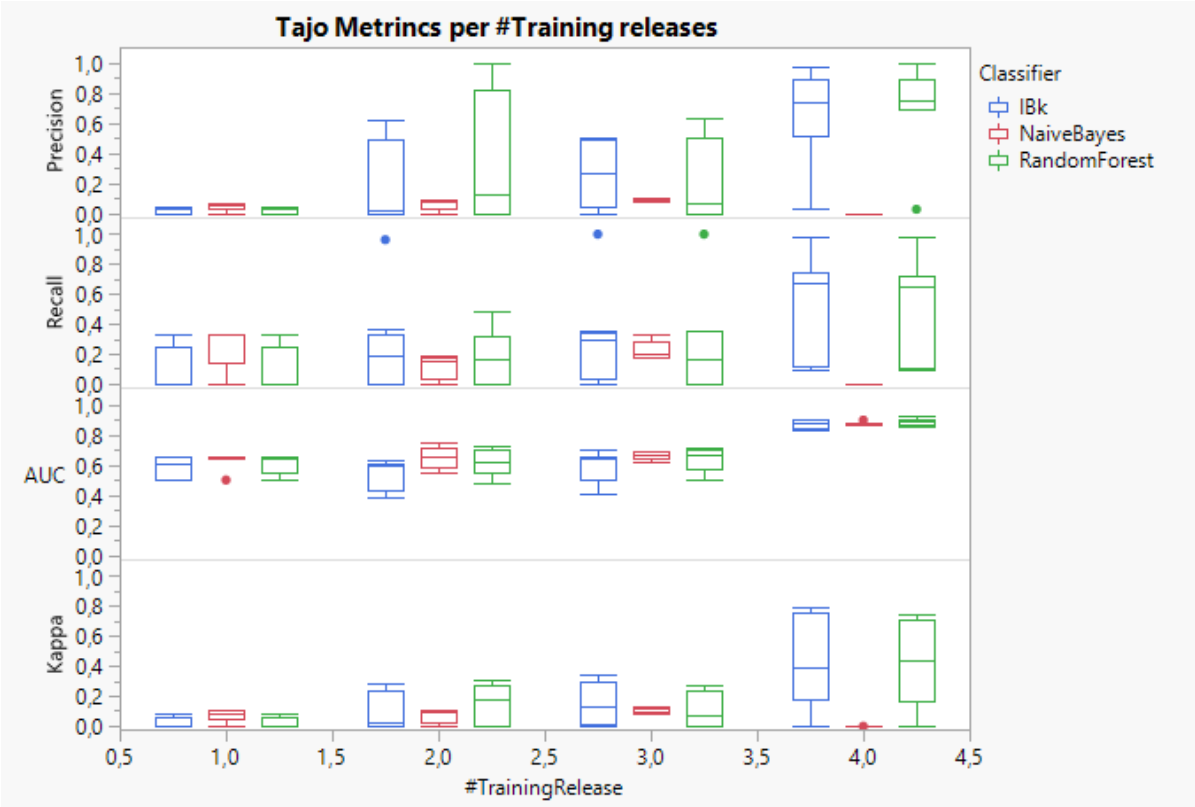


Fig.20 Metriche Tajo rispetto al numero di training release

Quello che ci aspettiamo è che le metriche aumentino all'aumentare del numero di release presenti nel training set. Questo è ciò che accade in Tajo (Fig.20), ma non in Bookkeeper (Fig.19)

dove le metriche vanno a calare soprattutto nella seconda metà. Un possibile problema potrebbe essere la scarsa presenza di classi positive, che portano al modello una carenza di informazioni anche utilizzando tecniche di Oversampling a causa della duplicazione delle informazioni. Andiamo allora a vedere la percentuale di classi positive nel training e testing set all'aumentare del numero di release (Fig.21).

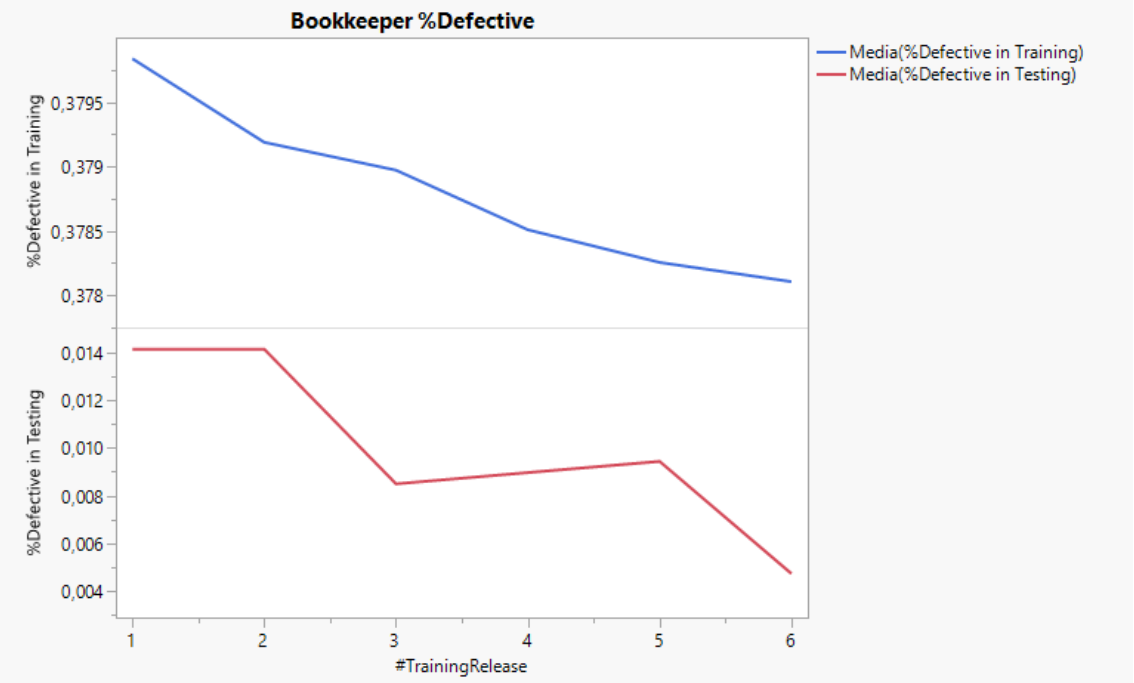


Fig.21 Classi defective in training e testing set

Si può notare come nel training set le classi defective diminuiscono leggermente, mentre nel testing set passano da 1.4% alle 0.4%, che essendo già la percentuale di partenza bassa, rende molto difficile l'individuazione delle classi. Questa diminuzione può essere dovuta a diversi fattori, uno di questi potrebbe essere la breve durata di una release e quindi un minor numero di difetti introdotti. Come possiamo notare infatti in Fig.22 dalla release 3 alla 6 si sono succedute in maniera molto rapida. Una soluzione al problema potrebbe quindi essere determinare un limite minimo di tempo nel quale rilasciare nuove release successive.

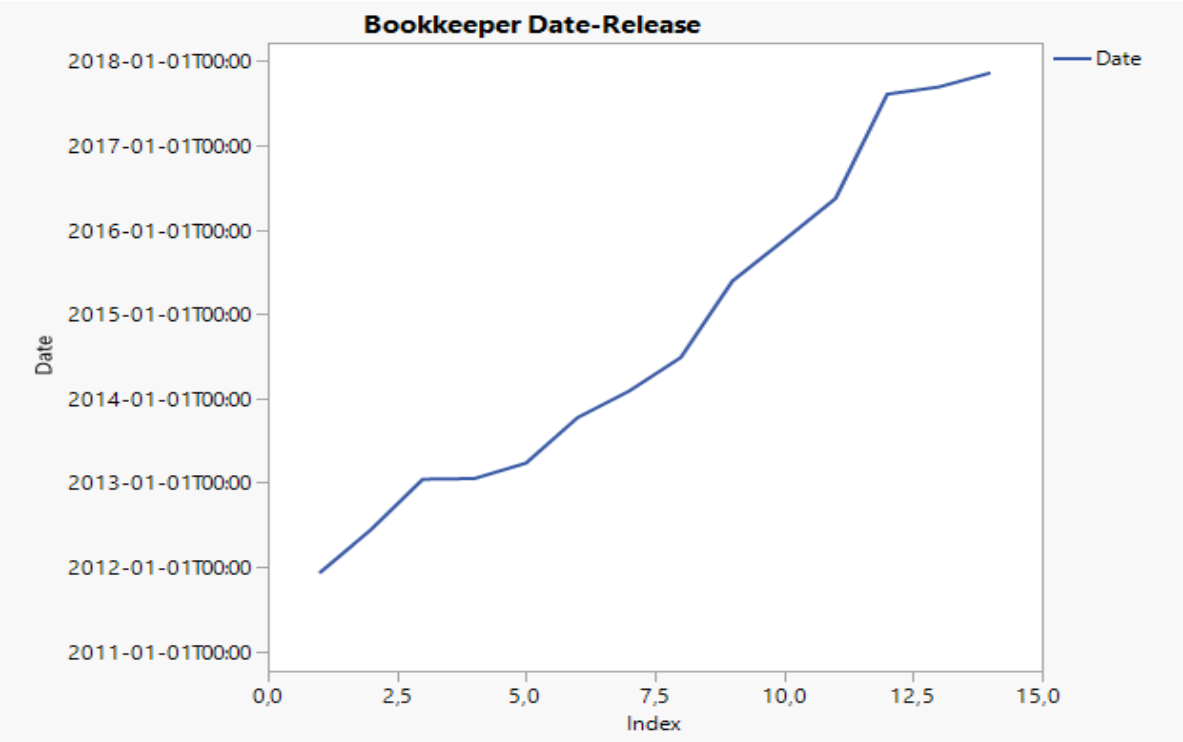


Fig.22 Bookkeeper date release

Un'ultima analisi effettuata riguarda un'altra tecnica di validazione, ovvero ***K-fold validation.*** Tramite questa strategia si riesce ad avere un testing set formato da una singola release, ed un training set formato invece da tutte le altre (quindi in generale più ampio rispetto al Walk forward). In questo caso però non viene tenuto conto dell'eventuale dipendenza dal tempo delle varie metriche. Come possiamo notare dai grafici in Fig.15 e Fig.16 le metriche (soprattutto AUC e Kappa) valutate con K-fold non concordano con quelle di Walk forward. In questo caso particolare quindi K-fold offre risultati migliori, infatti nel dataset originario sono state scelte metriche non dipendenti dallo scorrere del tempo come Age o Size.

6. Link SonarCloud

È riportato qui il link di SonarCloud della prima deliverable con zero Code Smells:

https://sonarcloud.io/dashboard?id=malavasiale_deliverabletwom1