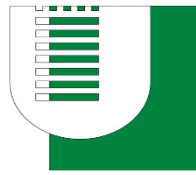



Magistrale in Ingegneria Informatica



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA



STABILITÀ DEL PROGETTO E PREDIZIONE DI BUG DI CLASSI JAVA IN PROGETTI JIRA

ISW2 2019-2020

Valentino Perrone 0264636

INDICE

Capitolo1 Introduzione	2
1.1 Tecnologie usate	2
1.2 Organizzazione codice ad alto livello	2
Capitolo2 Delivery1	3
2.1 Descrizione del problema	3
2.2 Analisi della stabilità	3
2.3 Ulteriori analisi.....	6
2.3.1 Parquet-mr default branch.....	7
2.3.2 Parquet-mr tutti i branch	8
2.3.3 Tutti repo default branch.....	10
2.3.4 <i>Tutti repo tutti i branch</i>	12
2.3.4 Considerazioni	13
2.4 Dettagli implementativi	13
2.5 Conclusioni	14
Capitolo3 Delivery2.....	15
3.1 Descrizione del problema	15
3.2 Costruzione del dataset	15
3.2.1 Dettagli sul calcolo delle metriche	16
3.2.2 Calcolo del LifeCycle	16
3.2.3 Bug da escludere	17
3.2.4 Tecniche per calcolare P.....	17
3.2.5 Classi buggy	17
3.3 Dettagli implementativi.....	18
3.3.1 Scelta del branch	18
3.4 weka machine learning	19
3.4.1 Risultati: Bookkeeper – moving window ml.....	20
3.4.2 Risultati: Bookkeeper – moving window ml – modello vs feature selection.....	20
3.4.3 Risultati: Bookkeeper – moving window ml – modello vs balancing	22
3.4.5 Risultati: Bookkeeper – moving window ml – risultato del tuning IBK	24
3.4.6 Risultati: Bookkeeper – moving window ml – percentuale bug per release.....	24
3.4.7 Risultati: Syncope – moving window ml – modello vs feature selection.....	25
3.4.8 Risultati: Syncope – moving window ml – modello vs balancing.....	27
3.4.9 Risultati: Syncope – moving window ml – risultato del tuning IBK	29
3.4.10 Risultati: Syncope – moving window ml – percentuale bug per release.....	30
3.4.11 Risultati: Confronto dataset Syncope & Bookkeeper.....	30
3.4.12 Risultati: Bookkeeper Brevi confronti tra risultati ottenuti con increment e moving window	31
3.4.13 Risultati: Syncope Brevi confronti tra risultati ottenuti con increment e moving window	32
Riferimenti sul Web.....	33

CAPITOLO1 INTRODUZIONE

Gli obiettivi dello studio descritto in questo documento rappresentano la stabilità del progetto considerato attraverso la tecnica dello statistical control process e predizione di bug in classi java in progetti gestiti dal sistema di ticketing Jira e di versioning Git.

I progetti esaminati sono i seguenti:

1. Apache Bookkeeper
2. Apache Parquet
3. Apache Syncope

Tale studio si compone di due parti:

- Delivery1: riguarda la stabilità del progetto preso in considerazione
- Delivery2: si focalizza sul confronto di tecniche di machine learning applicate alla predizione dei bug di classi Java

1.1 TECNOLOGIE USATE

Il software utilizzato per lo studio è stato scritto in linguaggio Java8 su IDE Eclipse. E' stato utilizzato SonarCloud come piattaforma per il rilevamento dei difetti relativi alla qualità del software, Travis CI come builder di progetto ed infine SVN connesso alla piattaforma GITHUB come sistema di versioning. Per quanto riguarda la visualizzazione dei risultati si è scelto di utilizzare Excel.

1.2 ORGANIZZAZIONE CODICE AD ALTO LIVELLO

Il codice scritto in java è stato organizzato in 3 package principali:

1. common: codice comune ad entrambe le delivery
 - a. entity
 - b. gestione directory per scaricare il repository
 - c. parser git e jira
 - d. parser output da rest api
 - e. classi utils
 - f. stringhe
2. firstdelivery: codice relativo alla prima consegna
 - a. main
 - b. controller
 - c. classe output
 - d. gestore scritture su file
 - e. classi utils
3. secondelivery: codice relativo alla seconda consegna
 - a. main
 - b. controller
 - c. calcolatore life cycle
 - d. calcolatore proportion method
 - e. calcolatore metriche
 - f. gestore scritture su file
 - g. classi machine learning
 - h. stringhe

CAPITOLO2 DELIVERY1

2.1 DESCRIZIONE DEL PROBLEMA

La prima consegna richiesta, come detto precedentemente riguarda la stabilità del progetto. Un processo è statisticamente stabile se il numero di bug è sempre limitato superiormente ed inferiormente in ogni periodo considerato. La stabilità fa riferimento al quinto livello del Capability Maturity Model Integration (**CMMI**).

CMMI è un approccio focalizzato sul miglioramento dei processi il cui obiettivo è di aiutare un'organizzazione a migliorare le sue prestazioni. Il CMMI può essere usato per guidare il miglioramento dei processi all'interno di un progetto, una divisione o un'intera organizzazione, per questo studio viene presa in considerazione il miglioramento del progetto. Il quinto livello di CMMI prevede un continuo miglioramento e quindi viene fatta un'analisi sui bug per controllare la loro stabilità così da poter prevenire eventuali problemi.

2.2 ANALISI DELLA STABILITÀ

L'analisi effettuata per lo studio della stabilità prevede una rappresentazione grafica temporale del numero di Bug Fixed di Jira in ogni mese da quando è stato iniziato il progetto. Come detto precedentemente il progetto del caso di studio è **Apache Parquet**.

I Bug Fixed sono stati estratti facendo una query verso Jira e prendendo tutti i bug con status closed o resolved e come campo resolution il valore fixed.

Per calcolare il numero di Bug Fixed si procede prendendo per ogni bug fixed la commit più recente temporalmente che contiene l'id del bug nel commento, si procede sommando tutte le commit dello stesso mese. Da notare che se eventualmente una commit dovesse contenere nel suo commento o titolo più di un bug allora ovviamente verranno presi tutti i bug trovati.

Per quanto concerne il progetto considerato, la finestra temporale va da Giugno 2014 ad Aprile 2020.

Le commit sono state raccolte sul branch di default della repository "parquet-mr".

Il grafico della stabilità contiene espresso come serie temporale del numero di bug fixed per mese contiene 3 rette orizzontali che riguardano :

- Media aritmetica della serie
 - o 3.27
- Upper bound della serie temporale:
 - o $average + 3 * devStd = 14.13$
- Lower bound della serie temporale:
 - o $max(0, average - 3 * devStd) = 0$
 - o $average - 3 * devStd = -6.46$, non ha senso avere una misura negativa se parliamo di quantità ≥ 0 quindi si è scelto di calcolare il max con 0.

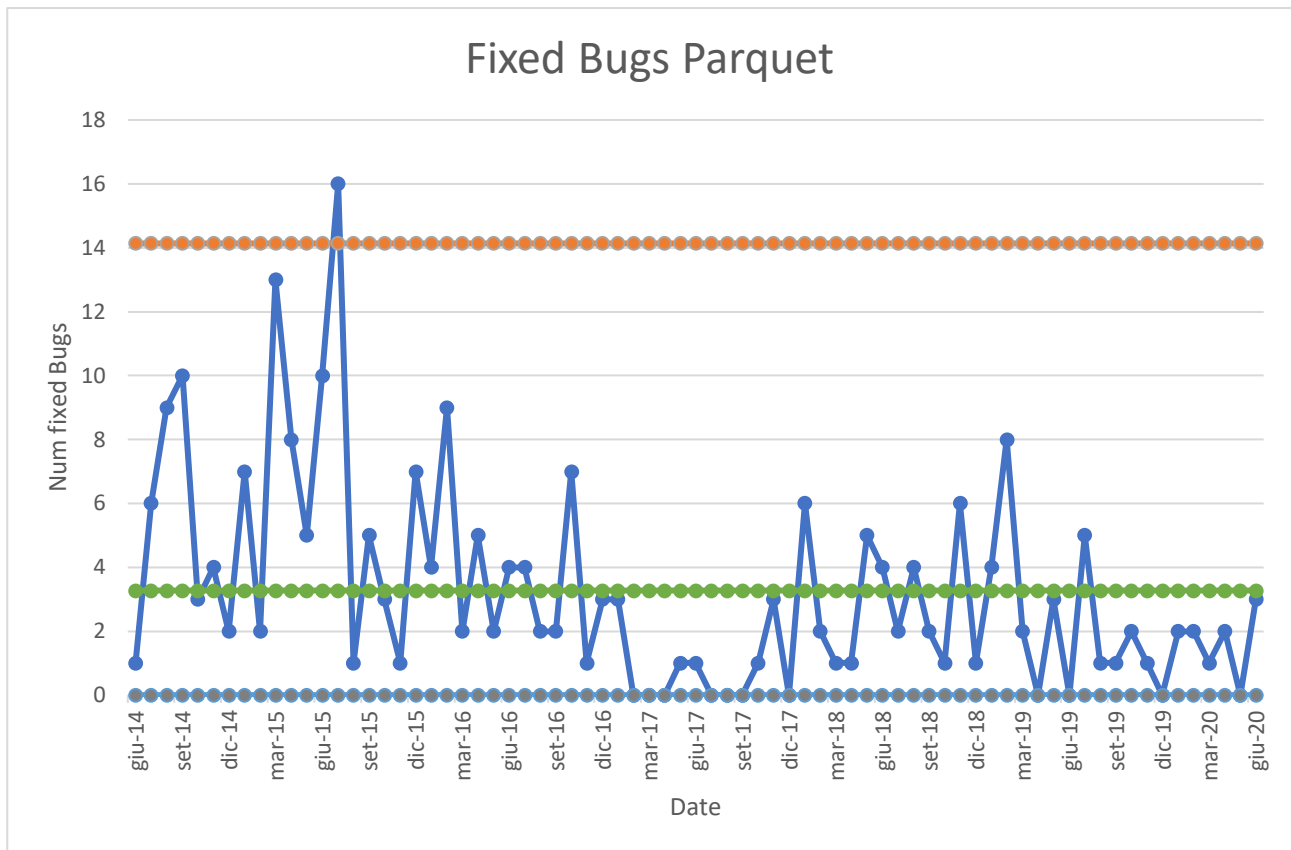


FIGURA 1 PROCESS CONTROL CHART

Come si evince dal grafico (Figura 1) l'unico mese in cui il numero di bug è superiore all'upper bound riguarda luglio 2015 con 16 bug. Il motivo di tale picco riguarda un incremento sostanziale della produttività in quel periodo, infatti si riscontra che è stata rilasciata una nuova versione proprio il 17 luglio 2015 (apache-parquet-1.8.1) e solo quel giorno sono state effettuate 7 commit.



FIGURA 2 PRODUTTIVITÀ REPOSITORY PARQUET-MR

La Figura 2 rappresenta un grafico della frequenza delle varie modifiche di add e delete per settimana. E' semplice notare che verso il mese di Luglio 2015 esiste un picco di frequenza.

E' stata inoltre studiata la distribuzione dei dati (Figura 3).

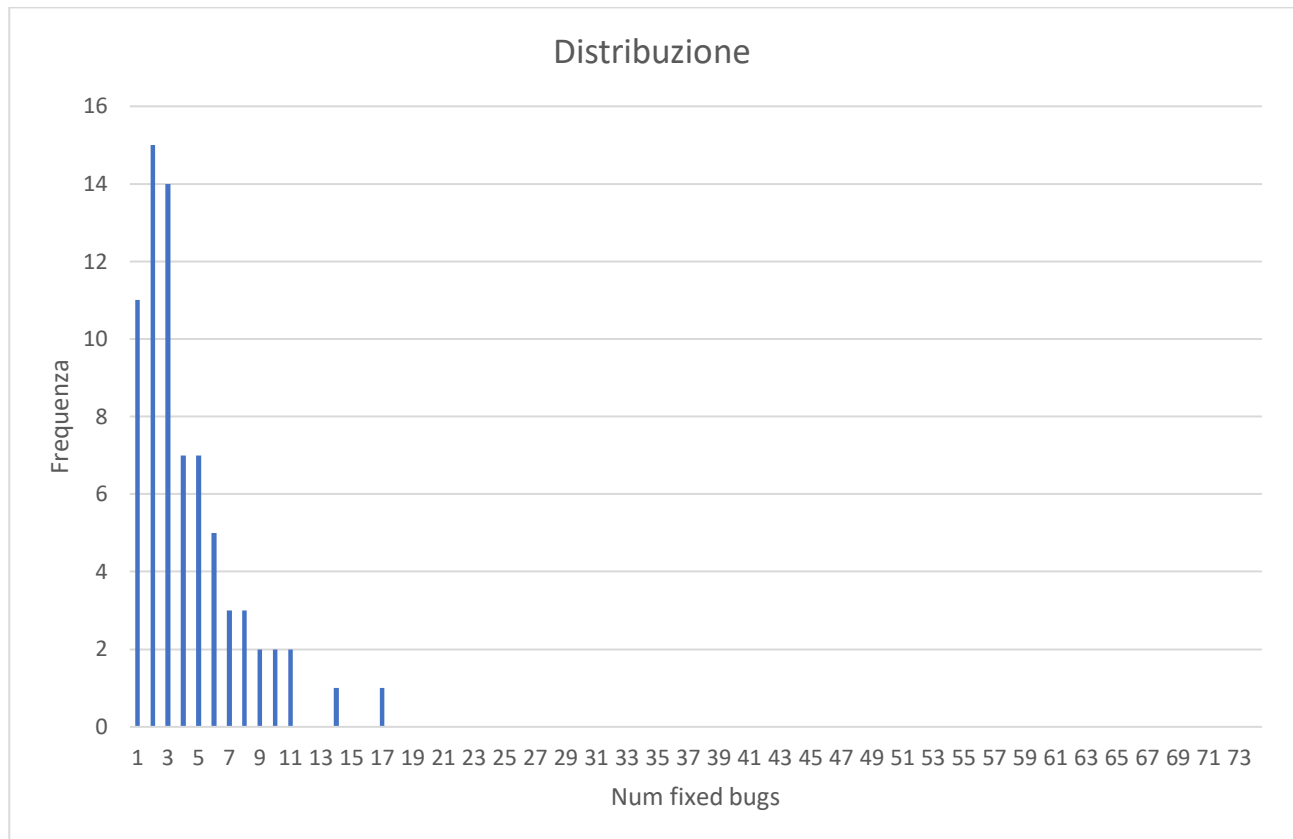


FIGURA 3 DISTRIBUZIONE DEL NUMERO DI FIX

Si è potuto notare che la distribuzione si avvicina ad una Chi quadro con pochi gradi di libertà. E' ragionevole aspettarsi una distribuzione del genere siccome il numero di fixed bug è strettamente positivo. Dallo stesso grafico si può inoltre vedere che esistono molti mesi che hanno pochi fixed bug, questo significa che il progetto si sta evolvendo bene e mantiene una certa stabilità intorno ai valori bassi.

2.3 ULTERIORI ANALISI

Informazioni raccolte:

1. Altri repository git trovati

Dallo studio dei bug del progetto PARQUET di Jira, è stato possibile notare che alcuni bug sono associati ad altri repository git oltre a quello ufficiale "parquet-mr". Sono stati identificati 2 ulteriori repository :

- arrow
- arrow-testing

2. Numero commit e bug

Sono state poi ricavate le seguenti informazioni :

- il numero di commit che hanno associato un id di un bug fixed "PARQUET-####"
- il numero di commit senza bug fixed
- il numero di commit che presentano id di altri progetti

- numero bug con commit associata
 - numero bug senza commit associata
3. Branch

Infine sono stati confrontati i progetti prendendo solo il branch di default e i progetti con tutti branch.

Per studiare queste informazioni è stata studiata la stabilità in 4 scenari :

1. Repository parquet-mr, branch di default
2. Repository parquet-mr, 35 branch coinvolti
3. Bug dei 3 repository con branch di default
 - parquet-mr
 - arrow, 5 branch
 - arrow-testing, 2 branch
4. Bug dei 3 repository con tutti i branch

2.3.1 PARQUET-MR DEFAULT BRANCH

Il default branch del repository principale è stato studiato nel capitolo precedente. In questo capitolo vengono aggiunte le informazioni riguardanti il numero di commit e di bug:

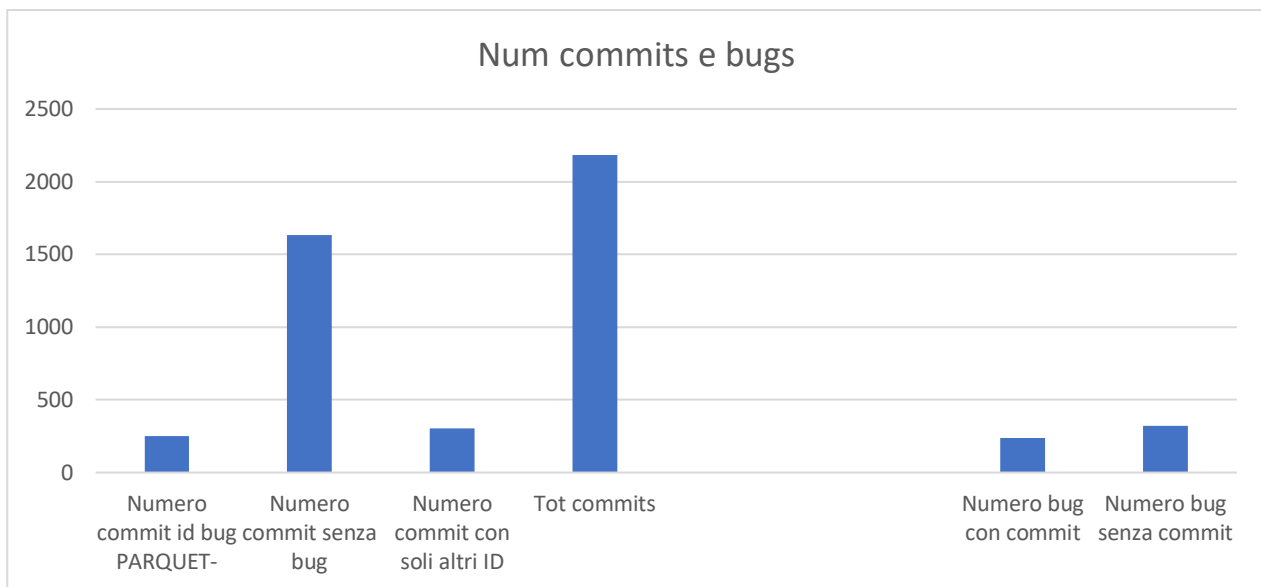


FIGURA 4 CONTEGGIO COMMITS E BUGS PER JIRA PARQUET DA REPOSITORY PARQUET-MR GITHUB

Commits

Numero commit id bug PARQUET-	251	11.49%
Numero commit senza bug	1632	74.69%
Numero commit con soli altri ID	302	13.82%
Tot commit	2185	100%

Bugs

Numero bug con commit	239	42.75%
Numero bug senza commit	320	57.25%

2.3.2 PARQUET-MR TUTTI I BRANCH

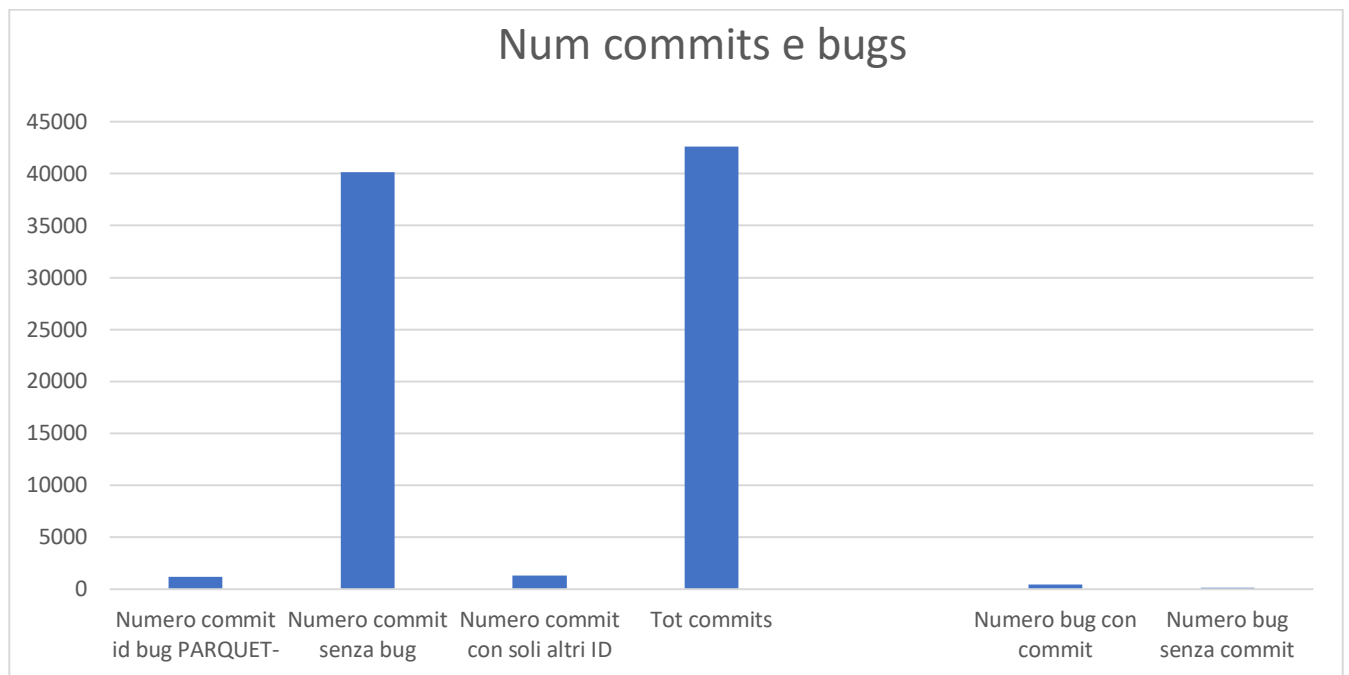


FIGURA 5 CONTEGGIO COMMITS E BUGS PER JIRA PARQUET DA REPOSITORY PARQUET-MR GITHUB CONSIDERANDO TUTTI I BRANCH

Commits

Numero commit id bug PARQUET-	1160	2.73%
Numero commit senza bug	40115	94.18%
Numero commit con soli altri ID	1317	3.09%
Tot commit	42592	100%

Bugs

Numero bug con commit	422	75.49 %
Numero bug senza commit	137	24.51%

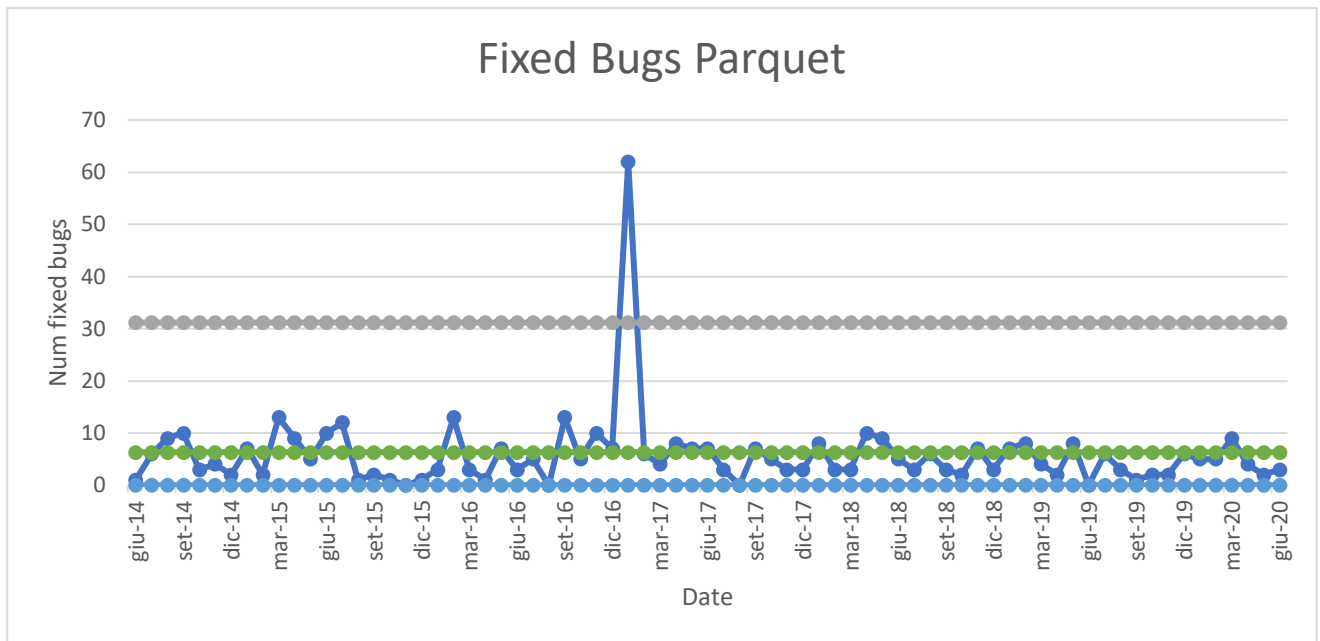


FIGURA 6 PROCESS CONTROL CHART PER JIRA PARQUET DA REPOSITORY PARQUET-MR CONSIDERANDO TUTTI I BRANCH

Per quanto concerne l'inclusione delle commit per tutti e 36 i branch di parquet salta subito all'occhio dalla Figura 6 che a Gennaio 2017 c'è un picco di 62 bugs. Anche in questo caso è presente un aumento della produttività probabilmente dovuto al rilascio della versione di parquet 1.8.2 il 19 gennaio 2017.

2.3.3 TUTTI REPO DEFAULT BRANCH

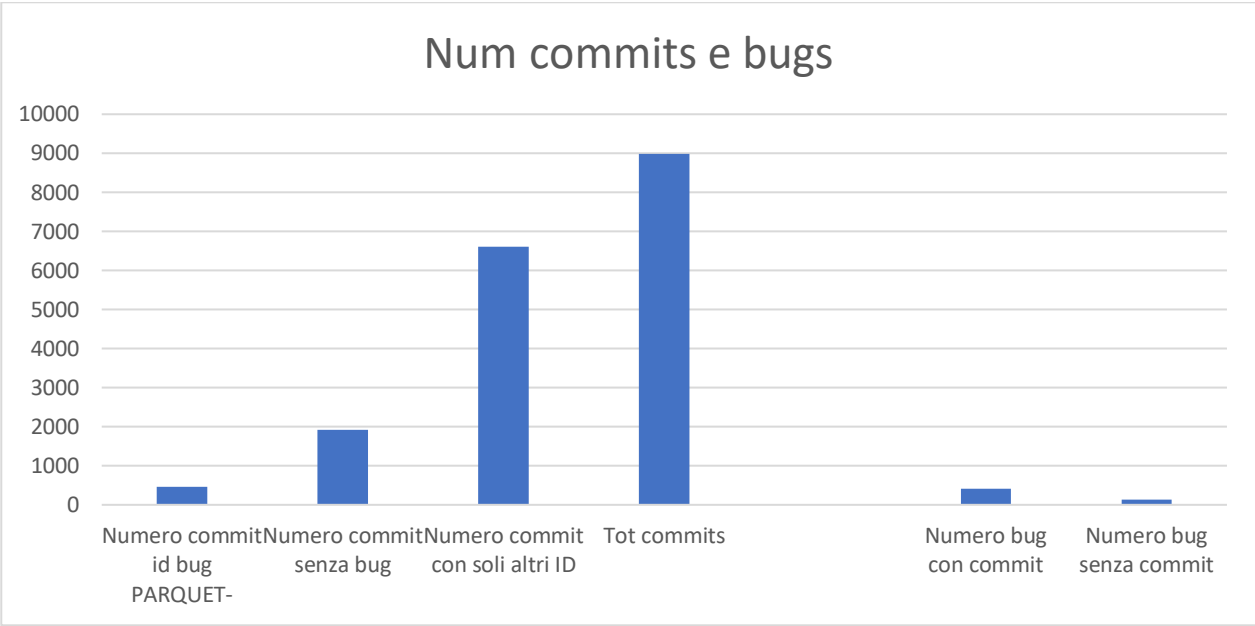


FIGURA 7 CONTEGGIO COMMITS E BUGS PER JIRA PARQUET DA REPOSITORY PARQUET-MR, ARROW, ARROW-TESTING GITHUB CONSIDERANDO IL DEFAULT BRANCH

Commits

Numero commit id bug PARQUET-	465	5.18%
Numero commit senza bug	1923	21.39%
Numero commit con soli altri ID	6601	73.43%
Tot commit	8989	100%

Bugs

Numero bug con commit	419	74.96%
Numero bug senza commit	140	25.04%

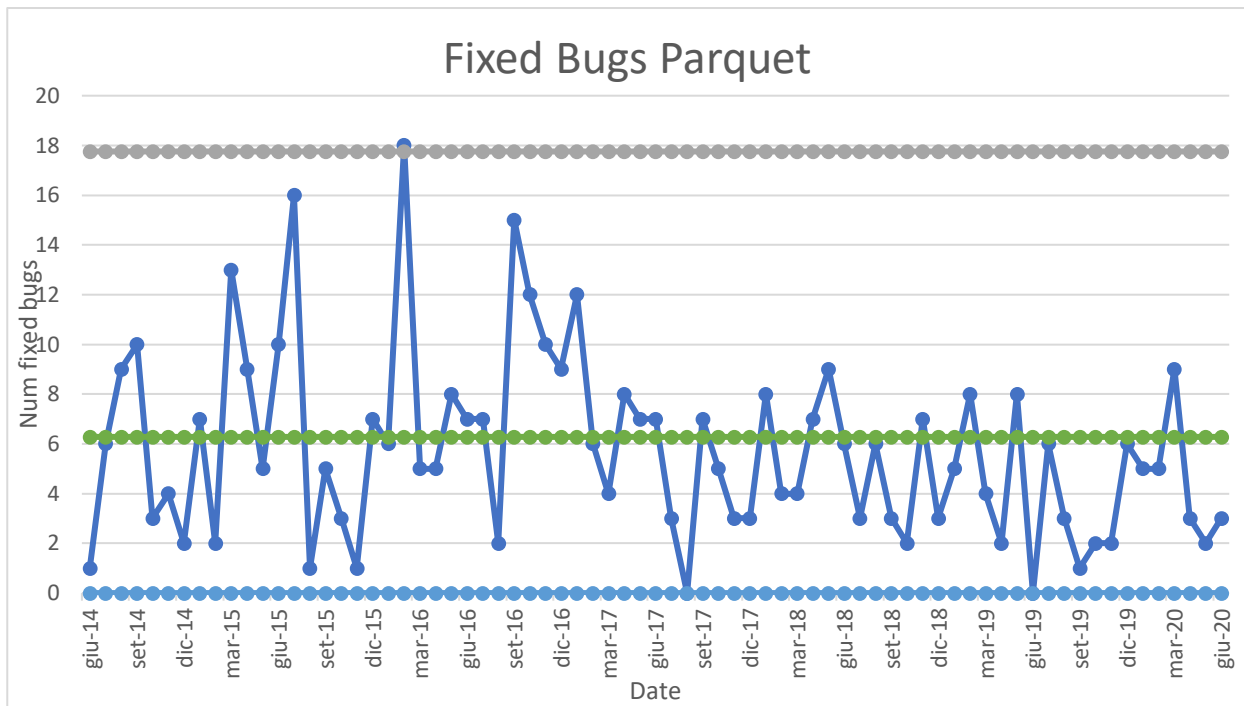


FIGURA 8 PROCESS CONTROL CHART PER JIRA PARQUET DA REPOSITORY PARQUET-MR, ARROW, ARROW-TESTING GITHUB CONSIDERANDO IL DEFAULT BRANCH

Dalla Figura 8 è possibile notare che aggiungendo gli altri repository (arrow e arrow-testing) è possibile raggiungere una stabilità maggiore nel caso delle sole commit del branch di default.

2.3.4 TUTTI REPO TUTTI I BRANCH

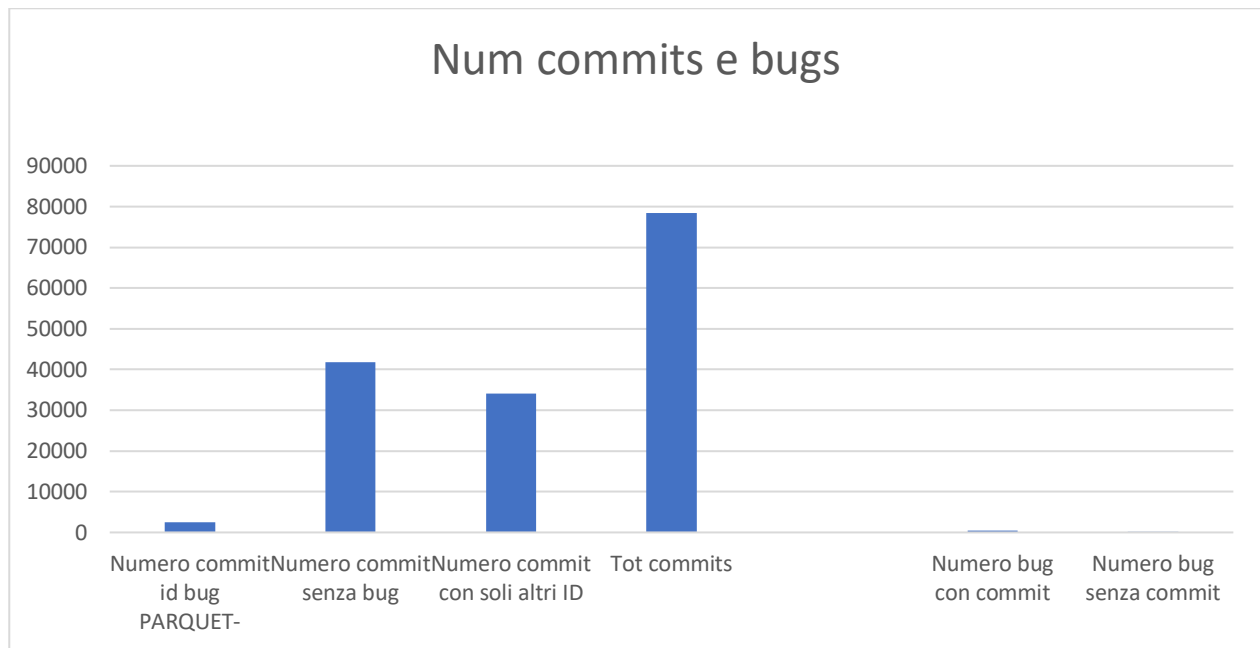


FIGURA 9 CONTEGGIO COMMITS E BUGS PER JIRA PARQUET DA REPOSITORY PARQUET-MR, ARROW, ARROW-TESTING CONSIDERANDO TUTTI I BRANCH

Commits

Numero commit id bug PARQUET-	2443	3.12%
Numero commit senza bug	41822	53.35%
Numero commit con soli altri ID	34126	43.53%
Tot commit	78391	100%

Bugs

Numero bug con commit	422	75.49%
Numero bug senza commit	137	24.51%

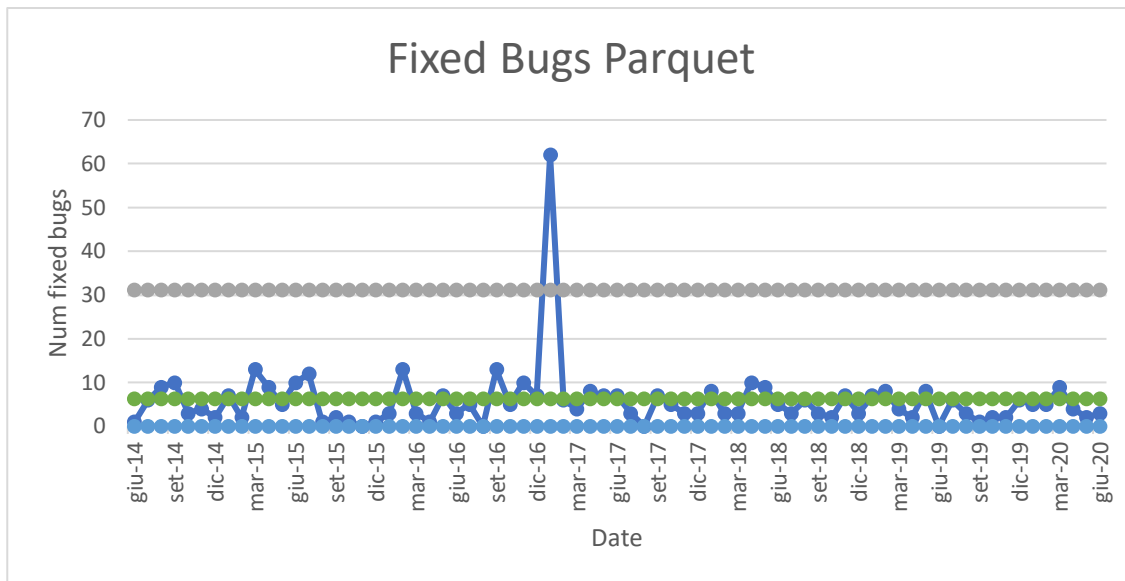


FIGURA 20 PROCESS CONTROL CHART PER JIRA PARQUET DA REPOSITORY PARQUET-MR, ARROW, ARROW-TESTING CONSIDERANDO TUTTI I BRANCH

2.3.4 CONSIDERAZIONI

Sapendo che il branch di default contiene tutte le commit dei branch che sono stati unificati possiamo concludere che le uniche commit aggiunte sono quelle dei branch non ancora unificati al branch di default.

Altre considerazioni sono quelle relative alle commit che hanno un id di ticket di altri progetti Jira, ad esempio alcune commit hanno i seguenti id : HIVE-###, AVRO-###, tali commit sono state escluse nella costruzione del process control chart. Il motivo dell'esistenza di commit legate ad altri bug jira può essere legato al fatto che parquet è un tipo di formato particolare orientato alle colonne che serve per ottimizzare le operazioni di lettura e scrittura effettuate dai framework di big data e quindi deve interfacciarsi con progetti come Hive (fornisce la possibilità di effettuare query sql su dati non strutturati) e Avro (altro formato di dati binario basato su json), tali progetti fanno parte dell'ecosistema hadoop ossia un insieme di progetti che risolvono problemi di big data e si interfacciano tra loro.

2.4 DETTAGLI IMPLEMENTATIVI

Il programma parte dalla classe Main e continua eseguendo il Controller che richiede i dati attraverso chiamate RESTfull http a Jira per i bug e a Github per le commit. Le classi parserGithub e parserJira consentono di leggere il Json ricevuto dalle chiamate RESTfull. Per quanto concerne Jira la query utilizzata consente di raccogliere tutti i ticket chiusi o risolti filtrati per attributo "fixed bug". Mentre da github vengono raccolte tutte le commit con relativa data e testo.

Per conoscere quali commit sono associate ad un determinato id Parquet- è stata utilizzata un'espressione regolare che controlla il matching esatto di ogni id di Jira nel titolo e commento di ogni commit.

L'espressione regolare utilizzata è la seguente:

- `"IDBug" + \d| + "IDBug" + \b`

Quindi applicando tale espressione al corpo della commit e sostituendo a IDBug ogni bug di Jira è possibile verificare il matching esatto.

In particolare:

- \d significa che non devono esserci caratteri altri caratteri numerici dopo l'id del bug nel commento della commit,

- | simbolo di or per catturare il caso in cui il carattere successivo è il terminatore di stringhe
- \b permette appunto di controllare se il ticket è boundary, cioè se non sono presenti caratteri dopo

La scelta di utilizzare le commit di github associate ai bug è stata forzata dal fatto che Jira non ha una data di risoluzione dei bug, in particolare il campo resolutionDate di Jira è un campo che viene riempito dagli sviluppatori durante l'apertura del bug ed è soltanto una previsione del tempo che serve per risolvere il bug, per tanto non è affidabile. L'unico svantaggio riscontrato incrociando bug e commit è che non tutte le commit sono state riferite al bug di interesse, quindi è possibile trovare dei bug scoperti senza commit e che quindi non rientrano nel conteggio finale. Fortunatamente soltanto circa il 30% di bug come mostrato precedentemente non ha associato nessuna commit.

Conclusa la fase di raccoglimento dei dati, la classe ControllerFirstDelivery si occupa di contare il numero di commit/bug in un mese per poi stampare la lista ottenuta su un file.

I grafici con i relativi calcoli inerenti all'upper bound, lower bound e media sono stati effettuati su Microsoft Excell.

2.5 CONCLUSIONI

Possiamo concludere che la stabilità dei progetti è verificata. Possiamo evincere dai grafici che durante la prima parte del progetto esso è un po' meno stabile, mentre avvicinandoci al 2020 i valori sono più vicini alla media. Questo è un chiaro sinonimo di maturità del progetto.

CAPITOLO3 DELIVERY2

3.1 DESCRIZIONE DEL PROBLEMA

Il problema da risolvere è quello di confrontare tecniche e modelli di machine learning al fine di studiare quale combinazione riesce ad effettuare una previsione migliore su quali sono le classi Java definite Buggy.

Tale studio è stato applicato a due progetti in maniera distinta:

- Apache Bookkeeper
- Apache Syncope

Per poter addestrare un modello di machine learning e ricavarne una previsione è necessario costruire un dataset.

Un dataset è una raccolta di dati rappresentata sotto forma di tabella. Ogni colonna rappresenta una particolare feature cioè una caratteristica il cui valore condiziona il risultato che vogliamo predire. Ogni feature ha un dominio di possibili valori.

Nello studio effettuato la previsione viene effettuata su una variabile di tipo binario, ossia una variabile che può assumere solamente due valori. Tale variabile binaria è la bugginess, ovvero la difettosità di una classe Java appartenente al progetto.

I modelli considerati sono di tipo Supervised ciò significa che per funzionare devono avere in input un set di dati di addestramento (o training set) già classificati, cioè che hanno associate delle etichette (o label) con il nome della classe di appartenenza. In questo modo il modello potrà estrarre dal training set una funzione complicata a piacere. Questa funziona servirà al modello per valutare la classe di appartenenza dei nuovi dati mai visti non ancora etichettati. L'insieme dei dati non ancora etichettati prende il nome di Test Set.

3.2 COSTRUZIONE DEL DATASET

Il tema della predizione dei difetti (defect prediction) è di interesse per molte aziende poiché è possibile prevedere quanto effort impiegare su un progetto in base al numero di bug presenti. Nella letteratura sono stati proposti molti approcci che effettuano previsioni basandosi su differenti metriche. Le metriche prese in considerazione nello studio attuale suppongono che i file difettosi sono quelli che sono stati modificati più frequentemente.

TABELLA 1 METRICHE PER FILE

Metric (File C)	Description
Size	LOC
<u>LOC touched</u>	Sum over revisions of LOC <u>added+deleted+modified</u>
NR	Number of revisions
<u>NFix</u>	Number of bug fixes
<u>NAuth</u>	Number of Authors
<u>LOC added</u>	Sum over revisions of LOC added
<u>MAX LOC added</u>	Maximum over revisions of LOC added
<u>AVG LOC added</u>	Average LOC added per revision
Churn	Sum over revisions of added – deleted LOC in C
<u>MAX Churn</u>	MAX_CHURN over revisions
<u>AVG Churn</u>	Average CHURN per revision
<u>ChgSetSize</u>	Number of files committed together with C
<u>MAX ChgSet</u>	Maximum of <u>ChgSet Size</u> over revisions
<u>AVG ChgSet</u>	Average of <u>ChgSet Size</u> over revisions
Age	Age of C in weeks
<u>WeightedAge</u>	Age weighted by <u>LOC touched</u>

La Tabella 1 mostra l'elenco delle metriche che sono state utilizzate come input al modello di previsione.

3.2.1 DETTAGLI SUL CALCOLO DELLE METRICHE

Le metriche precedentemente elencate sono calcolate leggendo le informazioni delle commit tra una release e la successiva per ogni classe java. Questo significa che per ogni release del progetto si ha un output di tutti i file presenti nella release con le relative metriche calcolate specificatamente in quella release.

Si è scelto di calcolare le metriche release per release azzerandole di volta in volta e non in maniera incrementale per l'intero progetto perché si è constatato attraverso altri studi pregressi di ricerca che il modello avente in input un dataset del genere si comporta meglio e restituisce risultati di accuratezza migliori. Un altro motivo di tale scelta è che l'informazione temporale è già inclusa nell'age della classe e quindi sarebbe ridondante calcolare le metriche in maniera incrementale.

Una metrica in particolare che merita di essere commentata è la WeightedAge, si è scelto di calcolarla secondo l'approccio delle metriche MOSER:

$$\frac{\sum_{i=1}^N Age(i) * addedLoc(i)}{\sum_{i=1}^N addedLoc(i)} \quad (1)$$

Al termine di ogni release le metriche vengono inizializzate a 0 tranne le due metriche di Age.

Le metriche vengono calcolate per ogni tipo di commit:

- Rename: viene mantenuto il vecchio file con le relative metriche ma ne viene cambiato il filename
- Copy: gestito come una delete e poi una create
- Modify: aggiorno tutte le metriche
- Create: inizializzo il file con le metriche
- Delete: elimino il file e le metriche annesse nella release attuale

3.2.2 CALCOLO DEL LIFE CYCLE

Una volta concluso il calcolo delle metriche informative per la difettosità della classe si passa al calcolo della buginess per ogni classe considerata.

Tale stima è stata effettuata attraverso il metodo Proportion. Tale metodo ha come obiettivo quello di impostare una release al valore Affected o non, ossia dire se una certa release è affetta da almeno un certo difetto. Proportion utilizza i seguenti stati del ticket:

- **Injected Version:** release affetta più vecchia. Se da Jira sono presenti le Affected Version allora l'injected è la prima versione di queste altrimenti viene calcolata tornando indietro con Proportion
- **Opening Version:** release dopo l'apertura del ticket
- **Fixed Version:** release in cui è stato risolto o chiuso il bug, viene calcolata prendendo la release dopo l'ultima commit del bug
- **Affected Version:** versioni affette dall'Injected alla Fixed esclusa. Se presenti vengono prese quelle nel campo di Jira altrimenti viene calcolata prendendo le versioni da Injected a Fixed.

Il metodo Proportion suppone che per ogni bug il rapporto tra FV-IV e FV-OV rimanga più o meno costante.

$$P = \frac{FV - IV}{FV - OV} \quad (2)$$

Dalla formula (2) si ricava che:

$$IV = FV - (FV - OV) * P \quad (3)$$

Con la formula (3) è possibile calcolare quale sia la release Injected e da qui impostare ad affected tutte le versioni fino alla Fixed version esclusa.

Bisogna far notare che per evitare problemi di snoring, cioè evitare che esistano dei bug dormienti che non vengono rilevati dal metodo nelle versioni più recenti, si è scelto di utilizzare solamente il 50% delle release più vecchie.

3.2.3 BUG DA ESCLUDERE

Durante lo studio delle release affette è possibile incontrare alcuni casi particolare che sono stati esclusi dall'analisi:

- Bug senza fixed version
- Bug con open versione maggiore o uguale di fixed version
- Bug senza open version

Nel caso in cui Injected version stesse dopo opening version viene applicato Simple method e si porta Injected sull'opening version.

3.2.4 TECNICHE PER CALCOLARE P

Tra i metodi studiati per calcolare P si è scelto di applicarne 2:

- moving window : si stima P facendo una media degli ultimi P suill' 1% dei bug precedenti con AV di Jira
- increment: si stima P facendo una media tra tutti i P dei bug precedenti con AV di Jira

Un problema riscontrato in questa fase è quello di calcolare P all'inizio quando non si hanno bug oppure non ho ancora accumulato 1%, in questo caso specifico si è scelto di utilizzare Simple Method e quindi impostare Injected Version all'Opening Version.

Vale la pena evidenziare che l'1% dei bug nel caso del progetto di Bookkeeper è inferiore ad 1. Si è deciso quindi di settare l'1% ad 1 in casi come questo.

3.2.5 CLASSI BUGGY

Una volta calcolate le metriche e trovate le release affette per ogni bug si conclude la costruzione del dataset settando la categoria della classe a buggy o non buggy.

Per settare la classe come buggy bisogna trovarla in una release affetta di un particolare bug e poi trovare almeno un'operazione di modify su di essa. Si setta quindi, la classe come buggy dall'injected version alla fixed version esclusa.

Una problematica riscontrata in questa fase riguarda i file che hanno cambiato nome durante le commit di rename. Quando una classe viene settata come buggy bisogna settare la classe buggy in tutte le release affette dall'injected alla fixed e quindi bisogna settare buggy anche le la classe con il vecchio nome. Per risolvere tale problematica è stato salvato il vecchio nome in un campo dell'entità File nel progetto Java e si vanno a cercare ricorsivamente tutti i vecchi riferimenti.

#	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	File Name	LOC	LOC_touched	NR	NFix	NAuth	LOC_added	MAX_LOC	AVG_LOC	Churn	MAX_Churn	AVG_Churn	ChgSetSize	MAX_ChgSets	AVG_ChgSet	Age	WeightedAge	Buggy
2	bookkeeper-benchmark/src/main/java/org/apache/bookkeeper/benchmark/MySQLClient.java	137	333	4	0	2	235	235	185.5	137	232	136.5	563	282	140.75	33	17.516172	NO
3	bookkeeper-benchmark/src/main/java/org/apache/bookkeeper/benchmark/TestClient.java	244	436	5	0	2	340	340	304.8	244	334	247.4	565	282	113	33	18.452099	NO
4	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/Bookie.java	545	545	1	0	1	545	545	545	545	545	0	282	282	282	0	0	NO
5	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/BookieException.java	81	81	1	0	1	81	81	81	81	81	0	282	282	282	0	0	NO
6	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/BufferedChannel.java	168	168	1	0	1	168	168	168	168	168	0	282	282	282	0	0	NO
7	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/EntryLogger.java	487	487	1	0	1	487	487	487	487	487	0	282	282	282	0	0	NO
8	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/FileInfo.java	124	124	1	0	1	124	124	124	124	124	0	282	282	282	0	0	NO
9	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/LedgerCache.java	536	536	1	0	1	536	536	536	536	536	0	282	282	282	0	0	NO
10	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/LedgerDescriptor.java	133	133	1	0	1	133	133	133	133	133	0	282	282	282	0	0	NO
11	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/LedgerEntryPage.java	151	151	1	0	1	151	151	151	151	151	0	282	282	282	0	0	NO
12	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/MarkerFileChannel.java	147	147	1	0	1	147	147	147	147	147	0	282	282	282	0	0	NO
13	bookkeeper/src/main/java/org/apache/bookkeeper/client/AsyncCallback.java	126	126	1	0	1	126	126	126	126	126	0	282	282	282	0	0	NO
14	bookkeeper/src/main/java/org/apache/bookkeeper/client/BKException.java	249	249	1	0	1	249	249	249	249	249	0	282	282	282	0	0	NO
15	bookkeeper/src/main/java/org/apache/bookkeeper/client/BookKeeper.java	410	410	1	0	1	410	410	410	410	410	0	282	282	282	0	0	NO
16	bookkeeper/src/main/java/org/apache/bookkeeper/client/BookieWatcher.java	204	204	1	0	1	204	204	204	204	204	0	282	282	282	0	0	NO
17	bookkeeper/src/main/java/org/apache/bookkeeper/client/CKS2DigestManager.java	50	50	1	0	1	50	50	50	50	50	0	282	282	282	0	0	NO
18	bookkeeper/src/main/java/org/apache/bookkeeper/client/DigestManager.java	184	184	1	0	1	184	184	184	184	184	0	282	282	282	0	0	NO
19	bookkeeper/src/main/java/org/apache/bookkeeper/client/DistributionSchedule.java	61	61	1	0	1	61	61	61	61	61	0	282	282	282	0	0	NO
20	bookkeeper/src/main/java/org/apache/bookkeeper/client/LedgerCreateOp.java	167	167	1	0	1	167	167	167	167	167	0	282	282	282	0	0	NO
21	bookkeeper/src/main/java/org/apache/bookkeeper/client/LedgerDeleteOp.java	80	80	1	0	1	80	80	80	80	80	0	282	282	282	0	0	NO

FIGURA 21 ESTRATTO DEL DATASET PRODOTTO DEL PROGETTO APACHE BOOKKEEPER

3.3 DETTAGLI IMPLEMENTATIVI

Per la seconda delivery si è scelto di ricavare le commit attraverso la libreria Jgit poiché le chiamate RESTfull http sono di numero limitato per fascia oraria. Con tale libreria viene effettuato il download del progetto ed è possibile scorrere tutte le commit esistenti.

3.3.1 SCELTA DEL BRANCH

Un altro aspetto che vale la pena menzionare riguarda i branch del progetto Git.

La scelta fatta è quella di considerare solamente il ramo principale del branch Master e quindi escludere una buona parte dei commit che risiedono nei vari rami mergiati con il branch Master.

Tale scelta è motivo della problematica che fa riferimento alla differenza cronologica commit per commit partendo dal nodo di merge tra più branch.

Quindi se si fossero analizzate anche le commit di percorsi dovuti al merge con altri branch si sarebbe dovuto partire dal nodo di merge e tornare indietro analizzando la differenza tra le commit in ordine cronologico fino ad arrivare al nodo genitore. Il problema nasce quando alla fine della release esistono branch che non sono ancora stati mergiati con il master e quindi l'analisi impatterebbe su più release. Impattando su più release diventa di difficile gestione l'analisi delle metriche relative ad una sola release.

Un'altra scelta possibile sarebbe stata quella di escludere solamente i branch che hanno durata maggiore ad una release e mantenere gli altri, ma tale implementazione avrebbe richiesto più tempo del necessario.

```
Format the indent to 4 spaces (to align with bookkeeper coding style)

* commit 5d749d7b8cb152e2b004e50c2a8e8c4c3e09903f
Author: Sijie Guo <sijie@apache.org>
Date: Mon Jan 29 07:42:37 2018 -0800

    repack under `org.apache.bookkeeper`

* commit 80530b3c786225f1567bba1beaf65e350845b582
Author: Sijie Guo <guosijie@gmail.com>
Date: Thu Jan 25 16:20:53 2018 -0800

    Use github repo as a temp location for deploying artifacts (#20)

* commit cab8fd365b53fa5a7ae8a35b6c98abdf4e52f371
Author: Sijie Guo <guosijie@gmail.com>
Date: Wed Jan 24 22:47:13 2018 -0800

    Provide a standalone cluster and a storage cli to interact with it (#19)

* commit eba314291a21a05a862ccb4060da79b66cebc3b6
Author: Sijie Guo <sijie@apache.org>
Date: Wed Jan 24 15:29:49 2018 -0800

    Support Increment (part 3): fix integration test

* commit 9935bb0a5624b4d820601c9826263d80e9500051
Author: Sijie Guo <sijie@apache.org>
Date: Wed Jan 24 15:13:48 2018 -0800

    Support Increment (part 2): provide increment op support in storage container, server and client

* commit 1e1aa52419558cc6584c82e2ccf06b0828304d1a
Author: Sijie Guo <sijie@apache.org>
Date: Wed Jan 24 14:08:30 2018 -0800

    Support Increment (part 1): provide increment op support in local state store.
```

FIGURA 22 GIT LOG COMMIT

Il ramo di commit considerato nel progetto è quello di colore verde (Figura 21). Come spiegato precedentemente i rami di colore blu e viola potrebbero durare diverse feature e quindi sarebbe complicato attendere il nodo di merge e poi analizzare a ritroso tutte le commit tra più release.

Tuttavia la percentuale di commit escluse risulta comunque bassa.

Un'altra peculiarità riguarda la presenza di branch orfani presenti ad esempio in bookkeeper.

I branch orfani sono dei branch particolari che non risultano essere figli di nessuna commit esistente, essi iniziano da una differente radice di commit. I branch orfani sono privi di storia pregressa e possono essere creati da Git con il flag “—orphan”. Solitamente vengono creati per scopi di testing oppure per sviluppare una nuova feature del tutto scorrelata dal resto dei branch.

```
Make all interfaces
* commit 3171228d0b8bfebcec81faac584854f93ae888f9
  Author: Jia Zhai <zhaijia@apache.org>
  Date: Thu Oct 26 14:43:22 2017 -0700

  Async and Sync MVCC Store

* commit 0cd6116717b725dc5528a32ae78b5cc6a9074685
  Author: Jia Zhai <zhaijia@apache.org>
  Date: Thu Oct 26 03:08:45 2017 -0700

  State Store: Add MVCC Store Interface

* commit 8315883abd0f1d01cd7e0b0bbe5d947ca2bf9c11
  Author: Jia Zhai <zhaijia@apache.org>
  Date: Fri Oct 27 01:50:31 2017 -0700

  Fix checkstyle and findbugs errors

* commit 5555e0bb3e82a1aed90a1ab174e6a17499ac6c01
  Author: Jia Zhai <zhaijia@apache.org>
  Date: Wed Oct 25 17:23:13 2017 -0700

  State Store: Define the k/v interface
* commit d237b537dc62b53d7ffc51860b98e5bd0a2ca9b8
  Author: Sijie Guo <sijie@apache.org>
  Date: Tue Feb 27 13:20:19 2018 -0800

  ISSUE #1147: Move bookkeeper-server/{bin,conf} to root module

  Descriptions of the changes in this PR:

  *Motivation*
```

FIGURA 23 ESTRATTO DAL GIT LOG DEL PROGETTO DI BOOKKEEPER, COMMIT 5555E0BB3E.. È L’INIZIO DI UN BRANCH ORFANO

3.4 WEKA MACHINE LEARNING

La parte di progetto inerente alla predizione è stata sviluppata in Java utilizzando la libreria di machine learning Weka.

Il modello prende in input il dataset creato e lo utilizza per la fase di addestramento dei parametri per poi riutilizzarli in fase di testing.

Sono state quindi confrontate le seguenti tecniche e i seguenti modelli:

Tecnica di valutazione	Walk Forward
Modelli	<ul style="list-style-type: none">- Naive Bayes- Random Forest- IBK con tuning su K (1,3,9,19)- <u>Logistic regression</u>
Feature selection	<ul style="list-style-type: none">- No feature selection- Best First- <u>Wrapper con Naive Bayes 5 fold</u>- <u>Principal component analysis(PCA)</u>
Balancing	<ul style="list-style-type: none">- No sampling- Undersampling- Oversampling- SMOTE

Tali tecniche e modelli sono state applicate ai seguenti dataset:

- Dataset Apache Bookkeeper con metodo moving window
- Dataset Apache Bookkeeper con metodo increment
- Dataset Apache Syncope con metodo moving window
- Dataset Apache Syncope con metodo increment

Al fine di effettuare un confronto tra i modelli sono state scelte le seguenti metriche:

- AUC
- KAPPA
- Precision
- Recall
- FMeasure

Da notare che nel tuning ad ogni run del walk forward di IBK, sono stati scelti dei parametri per K tra quelli menzionati sopra e viene scelto il migliore sulla base della metrica ROC valutandolo le features in 5 fold.

Prima di iniziare l'addestramento dei modelli è stata eseguita una fase di standardizzazione dei dati per evitare di avere una certa feature più grande in termini di scala di un'altra feature. Questa differenza di scala può provocare danni al modello in quanto la feature più grande potrebbe avere un peso maggiore. La standardizzazione permette di pesare allo stesso modo tutte le features.

3.4.1 RISULTATI: BOOKKEEPER – MOVING WINDOW ML

Per rispondere alle domande richieste, cioè per quali tecniche di feature selection o balancing si ottiene un aumento delle prestazioni dei classificatori, sono stati prodotti i seguenti grafici che aiutano a rispondere a tali domande. Da notare che è stata utilizzata la stessa scala di valori anche tra metriche diverse per avere un'immagine comparabile delle grandezze.

3.4.2 RISULTATI: BOOKKEEPER – MOVING WINDOW ML – MODELLO VS FEATURE SELECTION



FIGURA 24 BOX PLOT MODEL-FEATURESELECTION-NOSAMPLING

Come si nota dalla Figura 24 i classificatori si comportano abbastanza bene pur non avendo incluso nessuna tecnica di balancing. In particolare osservando la metrica AUC (misura di quanto il classificatore è lontano da quello randomico) si evince che le combinazioni migliori sono quelle Classificatore-bestFirst e Classificatore-NofeatureSelection. Non vale lo stesso per la tecnica PCA aggiunta, tale tecnica rimane mediamente più bassa rispetto alle altre e per alcune metriche ha anche una variabilità maggiore.

Facendo un confronto dei modelli per quanto riguarda BestFirst e NoFeatureSelection possiamo notare che:

- **IBK** si comporta molto bene siccome ha un AUC minimo di 0.8 per **bestFirst** e mediana 1.0 mentre per noFs valore minimo sempre 0.8 e mediana 0.9.
- La **Logistic Regression** mostra dei risultati ancora migliori, non in termini di AUC siccome è più o meno uguale a IBK ma in termini di variabilità per Recall ed Fmeasure. Viene preferito **non avere tecniche di feature selection** in quanto si ottiene un AUC meno variabile del caso bestFirst
- **RandomForest** risulta essere il classificatore con meno variabilità per tutte le metriche, addirittura la precision per quanto riguarda **noFs** ha un massimo di 1 e un minimo di 0.9
- **NaiveBayes** non ottiene risultati particolarmente brillanti. Si sceglie **bestFirst** siccome ottiene mediane più alte in confronto a noFs (no feature selection) a scapito di una variabilità maggiore

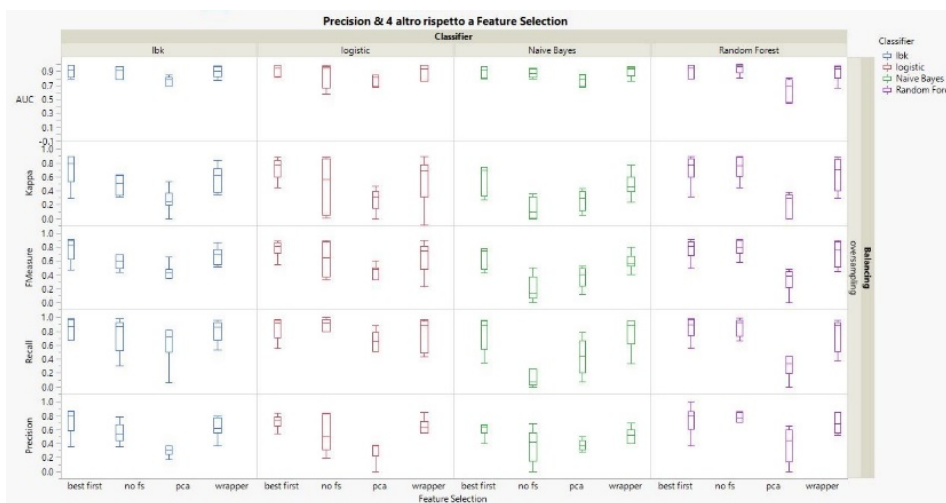


FIGURA 25 BOX PLOT MODEL-FEATURESELECTION-OVERSAMPLING



FIGURA 26 BOX PLOT MODEL-FEATURESELECTION-SMOTE



FIGURA 27 BOX PLOT MODEL-FEATURESELECTION-NOSAMPLING-UNDERSAMPLING

Dalle figure 25 26 27 è possibile vedere che variando la tecnica di balancing il risultato rimane mediamente lo stesso. Aumenta solamente la varianza in alcuni casi specifici.

Nel caso di **Logistic** con tecnica di **undersampling** è preferibile usare **bestFirst** invece di Nofs.

3.4.3 RISULTATI: BOOKKEEPER – MOVING WINDOW ML – MODELLO VS BALANCING

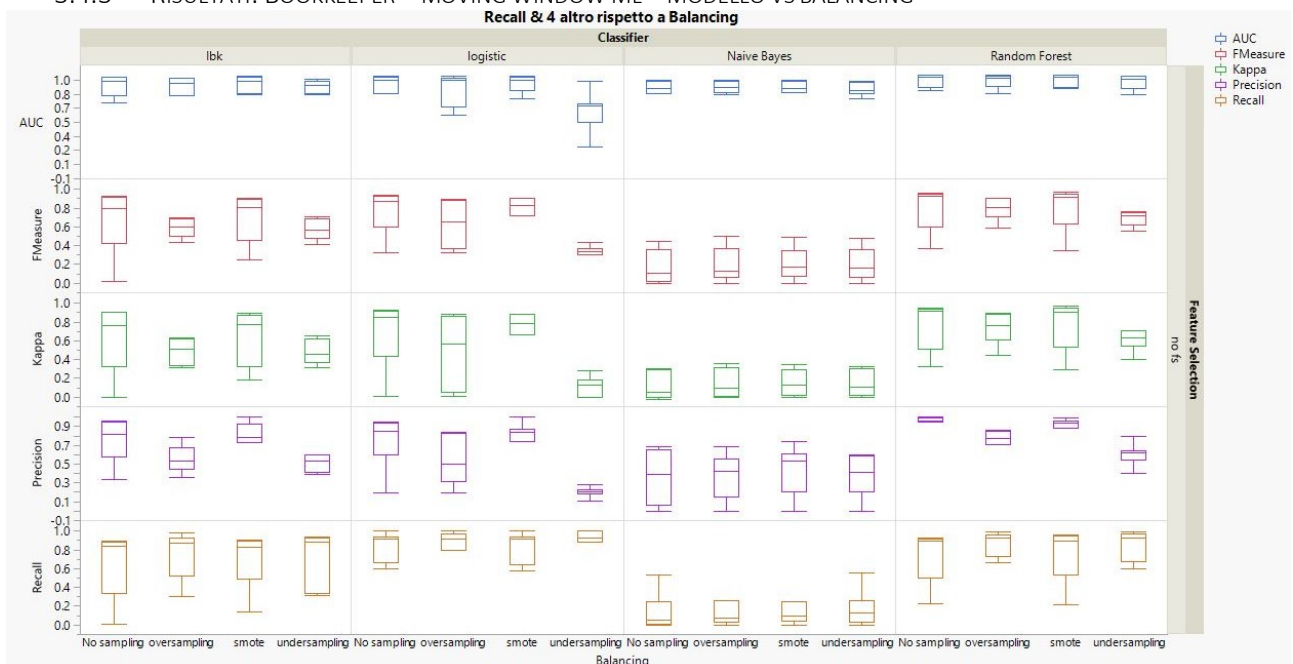


FIGURA 28 BOX PLOT MODEL-BALANCING-NO FEATURE SELECTION

Come mostrato nella figura 28 i classificatori IBK, Naive Bayes e Random Forest offrono prestazioni simili al variare delle tecniche di balancing.

- **IBK**: la tecnica **undersampling** offre prestazioni migliori, vince il confronto con oversampling poiché ha una variabilità più bassa in termini di precision e AUC. Una variabilità più bassa consente di avere una maggiore confidenza dei risultati in un intervallo più stretto.
- **Logistic**: **smote** offre valori alti con bassa variabilità

- **Naive Bayes:** il confronto tra le varie tecniche sembra essere di difficile valutazione poiché i risultati ottenuti sono molto simili. Tuttavia si è preferito scegliere come tecnica vincente **smote**, poiché offre livelli più alti di AUC.
- **Random Forest:** Il giusto compromesso tra bassa variabilità ed un'alta mediana risulta essere **oversampling**

I modelli che si comportano meglio in termini di stabilità rispetto a tutte le tecniche sono Naive Bayes e Random Forest.

Naive Bayes ottiene buoni risultati su tutte le tecniche, i valori di AUC risultano essere alti e stabili mentre invece i valori di FMeasure risultano essere più bassi con variabilità più alta e sono conformi a precision e recall essendo la media armonica. Anche Kappa non si discosta molto dai valori di Fmeasure.

Random Forest può essere considerato il modello che si comporta meglio con le tecniche di sampling mantenendo sempre valori alti e bassa variabilità.

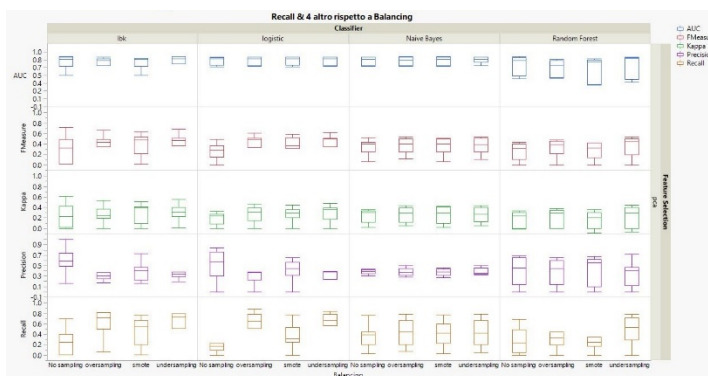


FIGURA 29 BOX PLOT MODEL-BALANCING-PCA

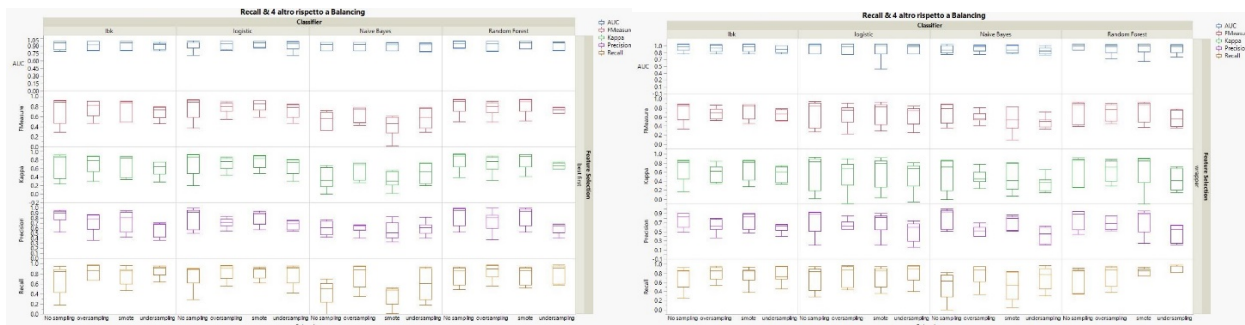


FIGURA 30-31 BOX PLOT MODEL-BALANCING-BEST FIRST BOX PLOT MODEL-BALANCING-WRAPPER NAÏVE BAYES

Le tecniche di feature selection introdotte nei grafici 29-30-31 confermano che le tecniche di balancing che migliorano le prestazioni sono quelle menzionate precedentemente. Non rimangono invariate le prestazioni di random forest che sembra peggiorare di molto nel caso di PCA.

3.4.5 RISULTATI: BOOKKEEPER – MOVING WINDOW ML – RISULTATO DEL TUNING IBK

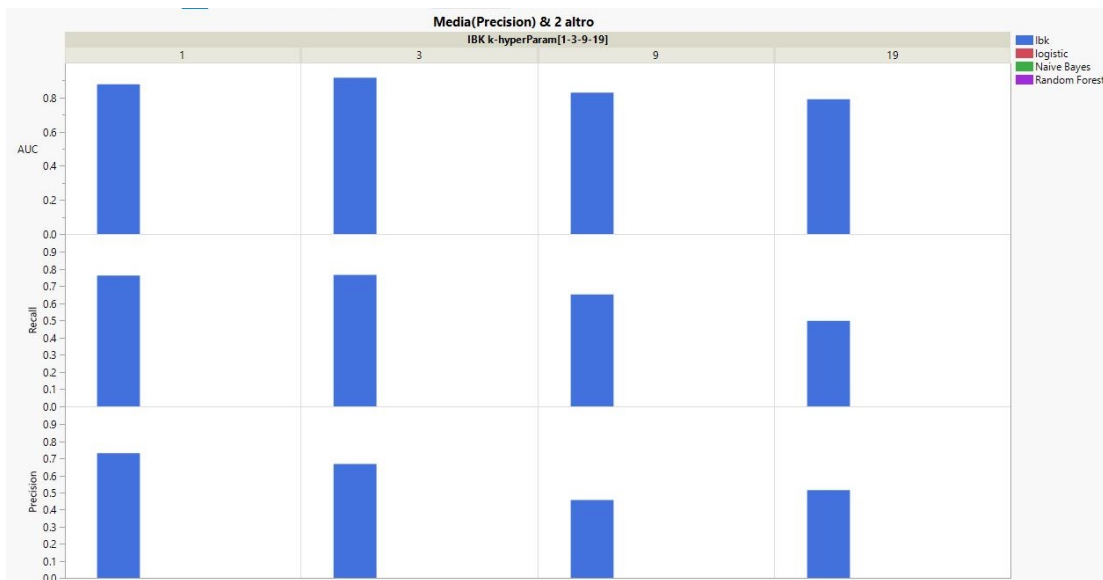


FIGURA 32 METRICHE AL VARIARE DI K

La figura precedente mostra il valore della misura in corrispondenza dei vari valori di K scelti in fase di tuning per il modello IBK (Figura 32). I valori bassi con $k=1$ e $k=3$ offrono performance migliori. Il parametro K viene utilizzato in fase di addestramento e rappresenta il numero di punti più vicini al punto i -esimo da classificare che servono appunto a decidere la classe di appartenenza. Questa figura spiega che il scegliendo K si hanno dei cambiamenti rilevanti alle prestazioni.

Sicuramente per un problema di classificazione semplice avere un K basso può bastare al fine di avere buone previsioni

3.4.6 RISULTATI: BOOKKEEPER – MOVING WINDOW ML – PERCENTUALE BUG PER RELEASE

Viene mostrato nel grafico seguente (Figura 33) il numero di difetti presenti nel training set e testing set ad ogni Run che risulta essere molto stabile.

Dalla Figura 33 si nota che i difetti occupano in media il 20% del dataset, questo non permette di avere un dataset bilanciato e quindi sono state provate le tecniche di balancing.

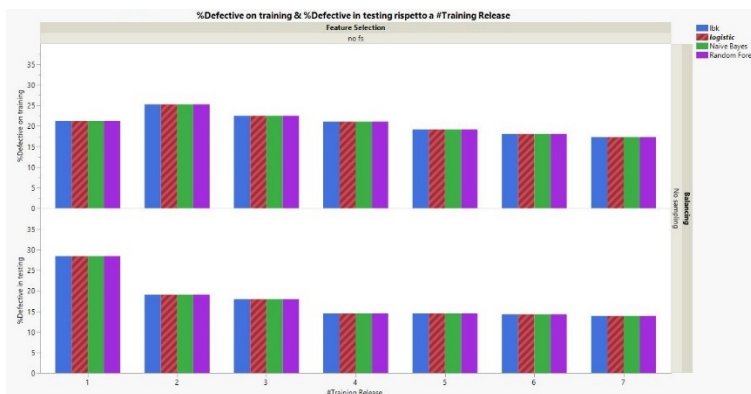
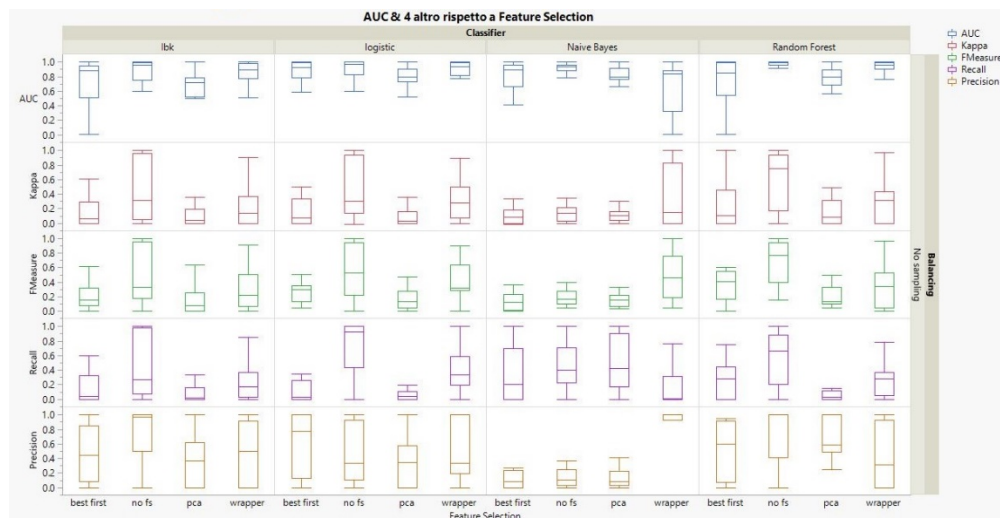


FIGURA 33 PERCENTUALE DEI DIFETTI AD OGNI RUN NEL TRAIN E TEST SET

FIGURA 34 BOX PLOT MODEL-FEATURESELECTION-NOSAMPLING



Si analizzano ora le metriche ottenute in figura 34 senza considerare tecniche di balancing.

Dal punti di vista di AUC, i modelli funzionano bene utilizzando tutte le feature, in particolare IBK, Naive Bayes e Random Forest mantengono un valore di AUC con variabilità molto bassa appunto senza applicare tecniche di feature selection.

Volendo approfondire la lettura dei risultati si ottiene la seguente tecnica migliore modello per modello:

- **IBK** migliora le prestazioni utilizzando la tecnica **wrapper** introdotta. Mantiene poco variabile AUC mantenendo pochi valori sotto il terzo quartile. In generale il modello risulta poco brillante in termini di Kappa e Fmeasure.
- Il modello **Logistic** ottiene buone performance con tutte le tecniche per la metrica AUC. Guardando però anche Fmeasure e Kappa si può dire che le performance aumentano particolarmente con la tecnica **Wrapper** che mantiene una variabilità più bassa ed una recall ed una precision simile. Le altre tecniche ottengono una recall alta e una bassa precision, sintomo di un modello che risponde troppo spesso in maniera negativa.
- **Naive Bayes** migliora le prestazioni con la tecnica di "**No feature selection**" ottenendo un alto AUC e minore variabilità in generale tra le metriche. Si nota però, che la tecnica wrapping ottiene un alto valore di precision e comunque un'altissima variabilità per le altre metriche.
- **Random Forest** rimane dell'idea che la tecnica migliore sia ancora una volta "**No feature selection**" mantenendo un'alta mediana, minore variabilità e valori massimi e minimi più alti per tutte le metriche.

Si conclude che in media l'assenza di tecniche di feature selection aumenta le prestazioni del classificatore.

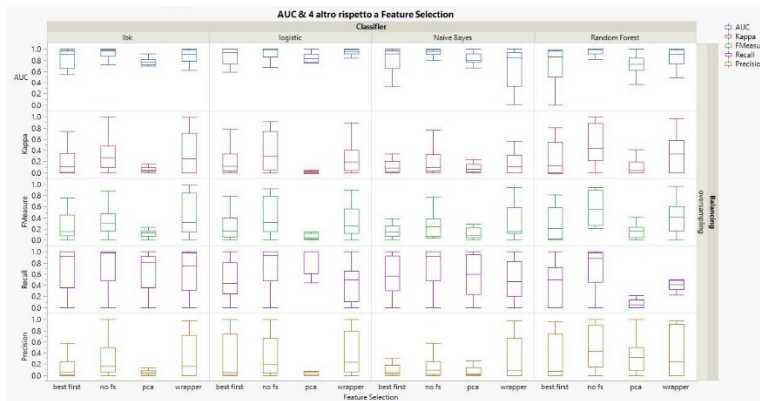


FIGURA 35 BOX PLOT MODEL-FEATURESELECTION-OVERSAMPLING

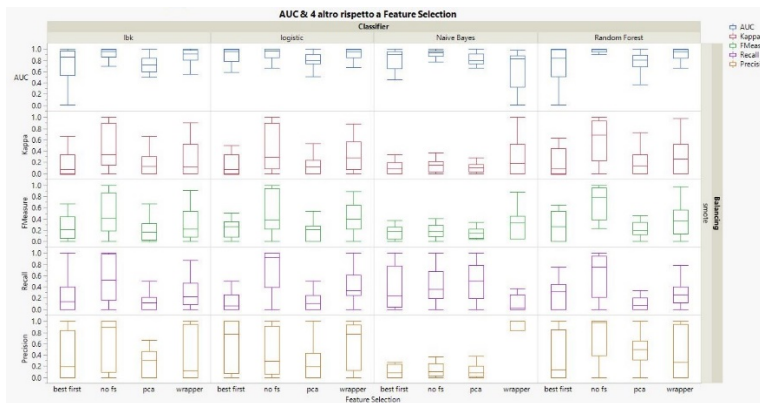


FIGURA 36 BOX PLOT MODEL-FEATURESELECTION-SMOTE

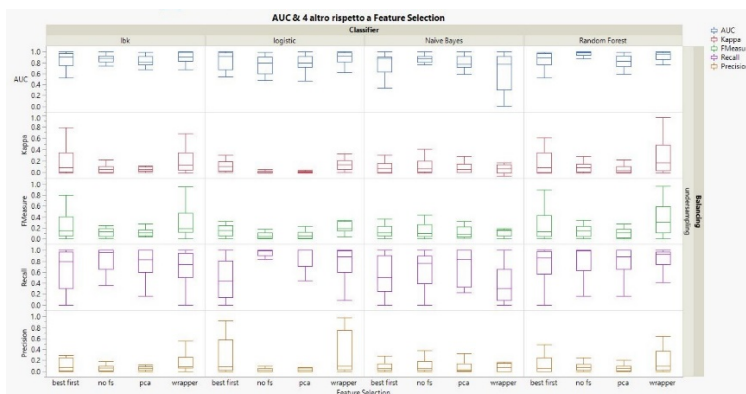


FIGURA 37 BOX PLOT MODEL-FEATURESELECTION-UNDERSAMPLING

Dalle figure 35-36-37 analizzando le diverse tecniche di feature selection fissando una alla volta le tecniche di balancing si deduce ancora una volta che l'assenza di tecniche di feature selection e la wrapper sono le migliori in termini di prestazioni.

3.4.8 RISULTATI: SYSCOPE – MOVING WINDOW ML – MODELLO VS BALANCING

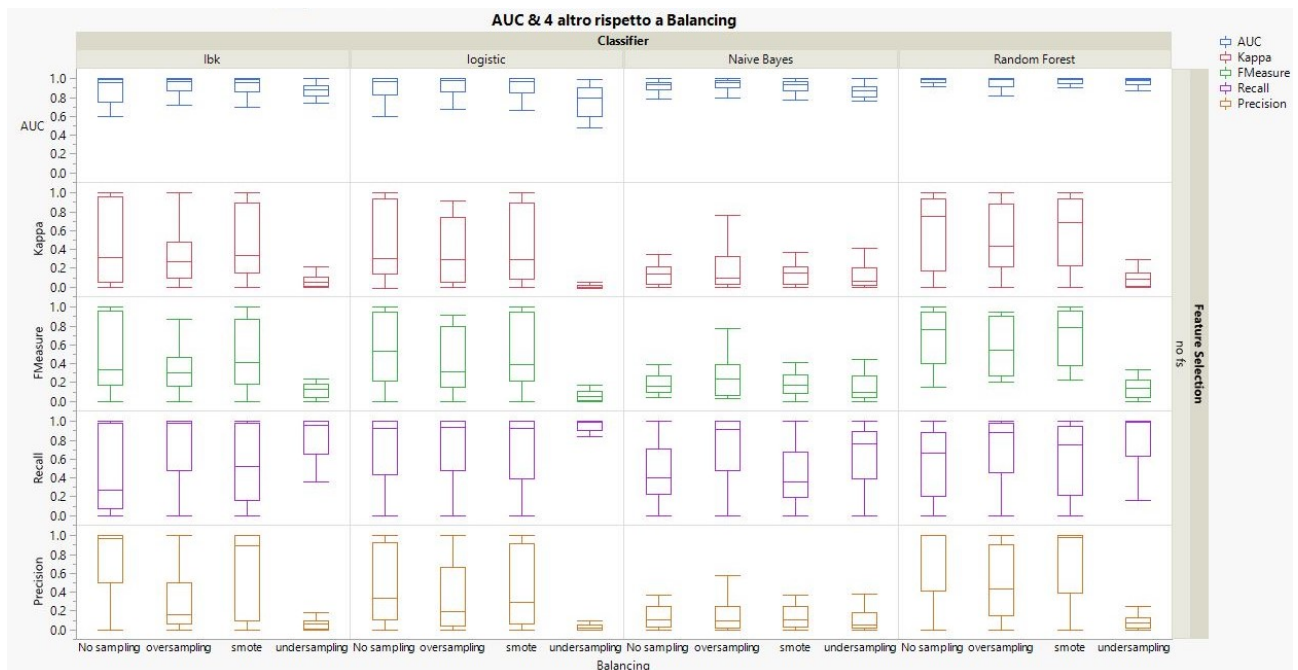


FIGURA 38 BOX PLOT MODEL-BALANCING-NO FEATURE SELECTION

Per quanto concerne le tecniche di balancing vengono ora analizzate le prestazioni senza tecniche di feature selection.

- **IBK:** **smote** ottiene performance migliori su tutte le metriche con lo svantaggio di avere un'alta variabilità su tutte le metriche tranne AUC. Oversampling mantiene una variabilità più bassa in termini di Fmeasure e Kappa con lo svantaggio di avere una bassissima precision e quindi sbaglia molte volte la classificazione dei positivi.
- **Logistic:** oversampling e smote si comportano in maniera molto simile in metrica AUC, ma l'assenza di tecniche di balancing ottiene una mediana di Fmeasure poco più alta. La tecnica migliore rimane **Smote** in questo caso.
- **Naive Bayes:** AUC risulta essere più alta con la tecnica di **oversampling** e confermano anche Fmeasure e KAPPA
- **Random Forest:** **smote** si riconferma essere la tecnica migliore anche se non migliora di moltissimo rispetto all'assenza di sampling.

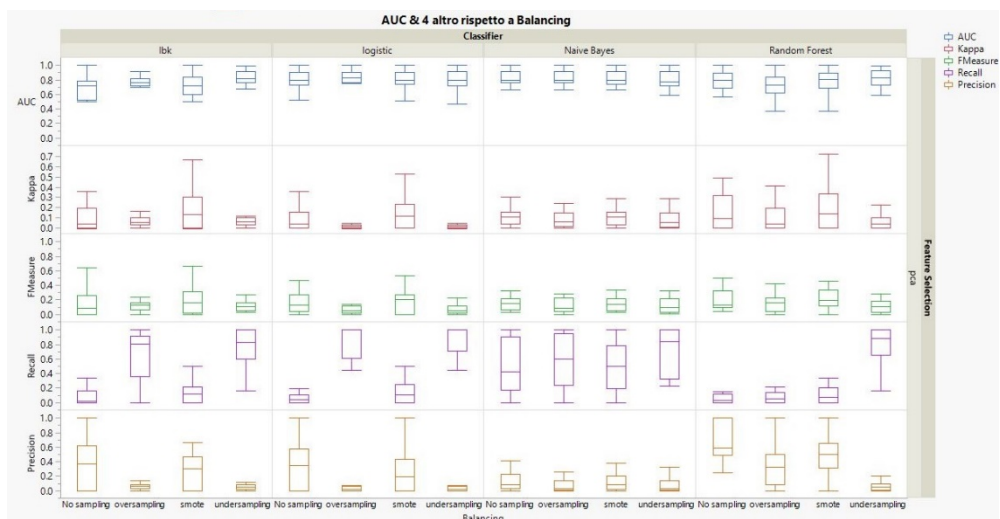


FIGURA 39 BOX PLOT MODEL-BALANCING-PCA

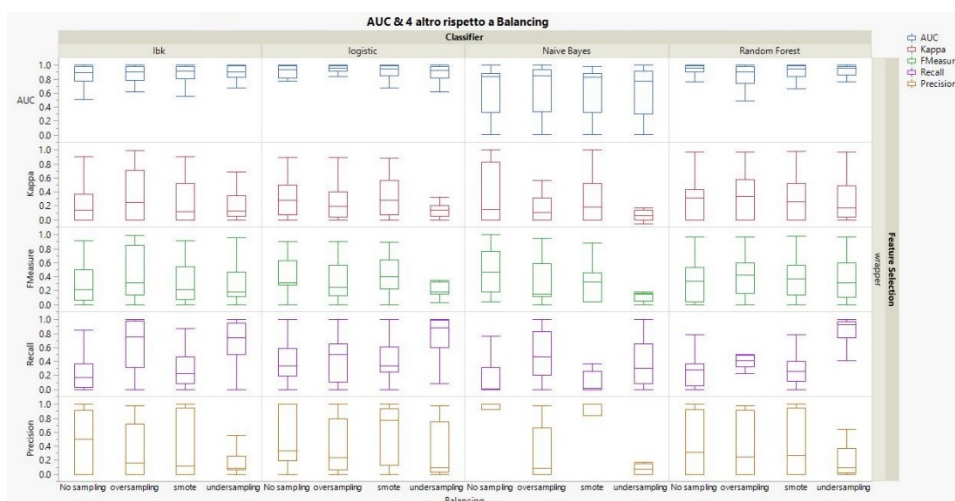


FIGURA 40 BOX PLOT MODEL-BALANCING-WRAPPER

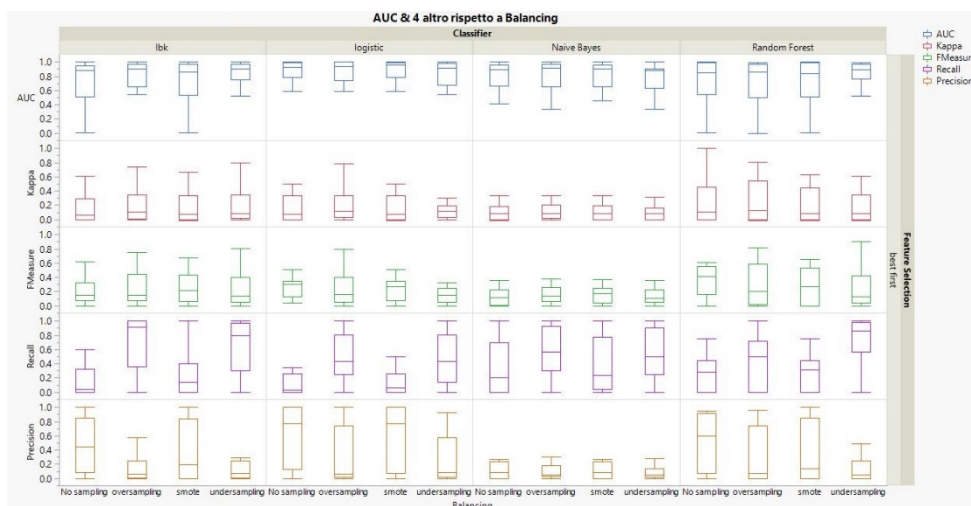


FIGURA 41 BOX PLOT MODEL-BALANCING-BEST FIRST

Anche in questo caso l'impatto delle tecniche di feature selection non provoca sostanziali cambiamenti nei risultati ottenuti per le tecniche di balancing.

3.4.9 RISULTATI: SYSCOPE – MOVING WINDOW ML – RISULTATO DEL TUNING IBK

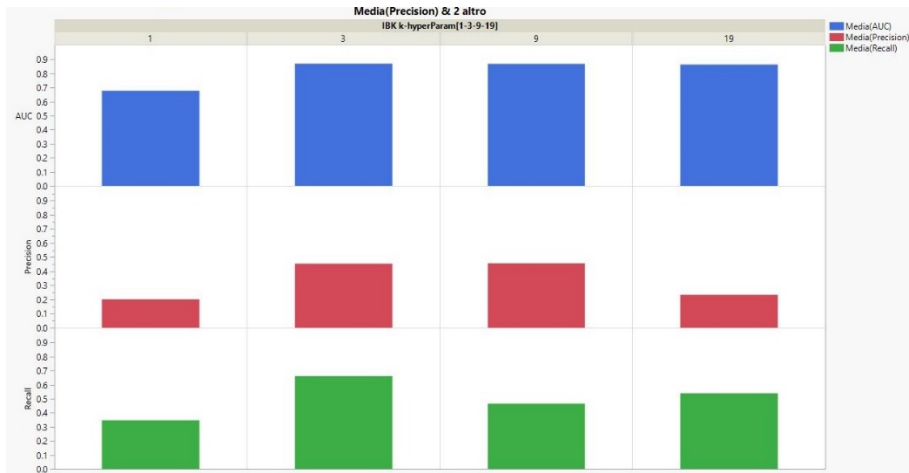


FIGURA 42 METRICHE AL VARIARE DI K

Ancora una volta viene mostrato il valore della misura al variare delle 3 metriche in corrispondenza dei vari valori di K scelti in fase di tuning per il modello IBK (Figura 42). K 3 e 9 ottiene performance migliori rispetto a 1 e 19.

3.4.10 RISULTATI: SYSCOPE – MOVING WINDOW ML – PERCENTUALE BUG PER RELEASE

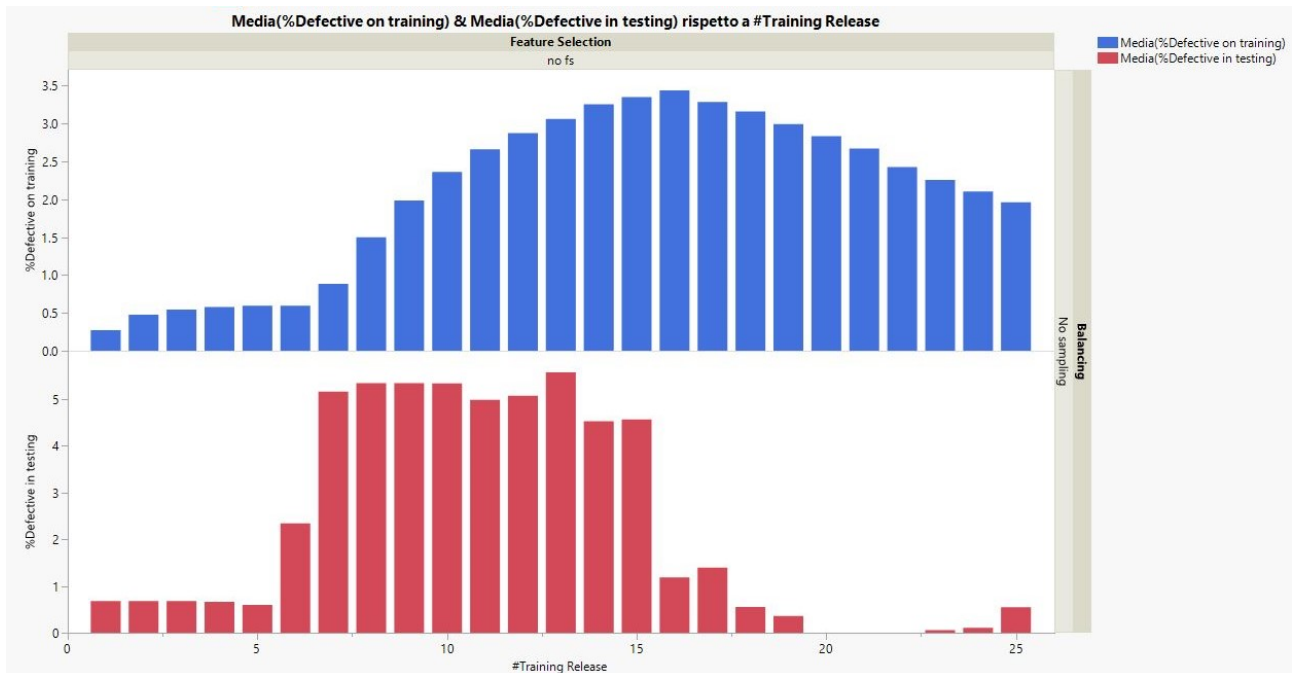


FIGURA 43 PERCENTUALE DEI DIFETTI AD OGNI RUN NEL TRAIN E TEST SET

Come visto precedentemente undersampling ottiene risultati peggiori rispetto a tutte le altre tecniche di sampling. Un motivo è che il numero di difetti in molte release è molto basso (inferiore all'1%) come si nota dalla Figura 43. Nel testing set i valori più alti di difettosità raggiungono il 5/6%. Anche il training è affetto dallo stesso problema siccome i valori più alti raggiungono il 3.5%. Alcune release come la 21,22,23 non hanno nessun'istanza difettosa nel testing.

3.4.11 RISULTATI: CONFRONTO DATASET SYSCOPE & BOOKKEEPER

Risulta importante confrontare quale tecnica risulta migliore in generale per il problema di defect prediction. Il machine learning si presenta come un lavoro d'arte e quindi è importante capire cosa funziona meglio per la tipologia di problema.

Per quanto concerne le tecniche balancing, dallo studio effettuato sui due progetti si evince che le tecniche di **Smote e Oversampling** risultano le migliori per la classe di problema considerato. Il risultato viene dettato dal numero di bug che un progetto può avere, un progetto abbastanza buono ha un numero di bug bassi e quindi non risulta conveniente utilizzare una tecnica di Oversampling che ridurrebbe di molto le istanze considerate.

Parlando invece di quale tecnica di feature selection è risultata migliore per entrambi i dataset possiamo concludere che BestFirst e Wrapper con NaiveBayes ottengono buoni risultati. Tuttavia anche l'assenza di tecniche di feature selection ha ottenuto buone prestazioni.

3.4.12 RISULTATI: BOOKKEEPER BREVI CONFRONTI TRA RISULTATI OTTENUTI CON INCREMENT E MOVING WINDOW

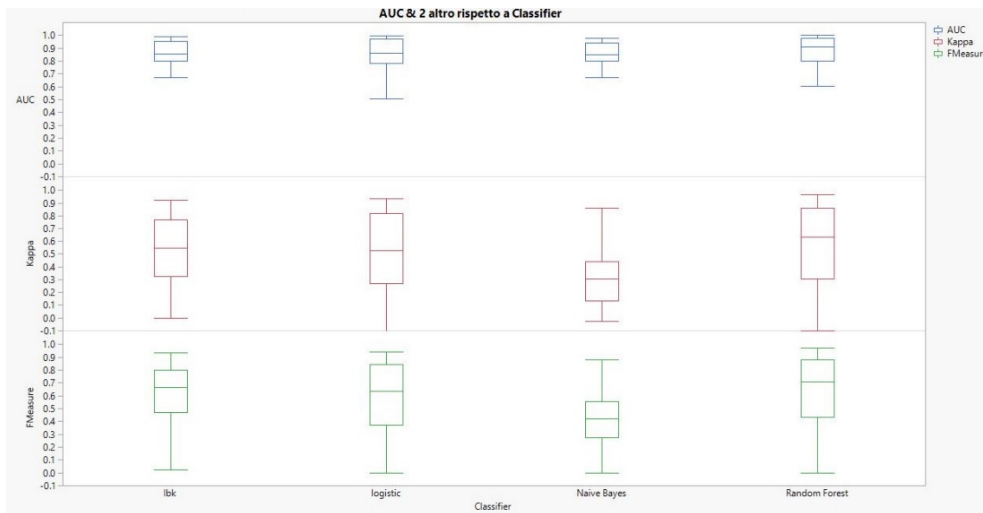


FIGURA 44 CLASSIFICATORE METODO MOVING WINDOW BOOKKEEPER



FIGURA 45 CLASSIFICATORE METODO INCREMENT BOOKKEEPER

I risultati ottenute dalle metriche per i classificatori riguardanti i due metodi increment e moving window risultano essere esattamente uguali. Questo perché moving window ottiene una media molto simile all'increment che utilizza tutti i dati precedenti.

3.4.13 RISULTATI: SYNCOPE BREVI CONFRONTI TRA RISULTATI OTTENUTI CON INCREMENT E MOVING WINDOW

Vengono fatti infine dei brevi confronti tra increment e moving window confrontando semplicemente i classificatori.

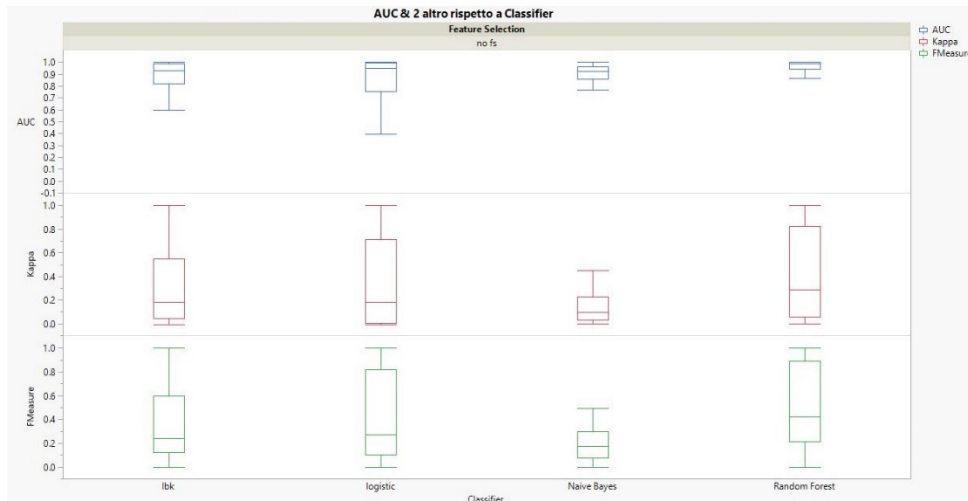


FIGURA 46 CLASSIFICATORE METODO MOVING WINDOW SYNCOPE

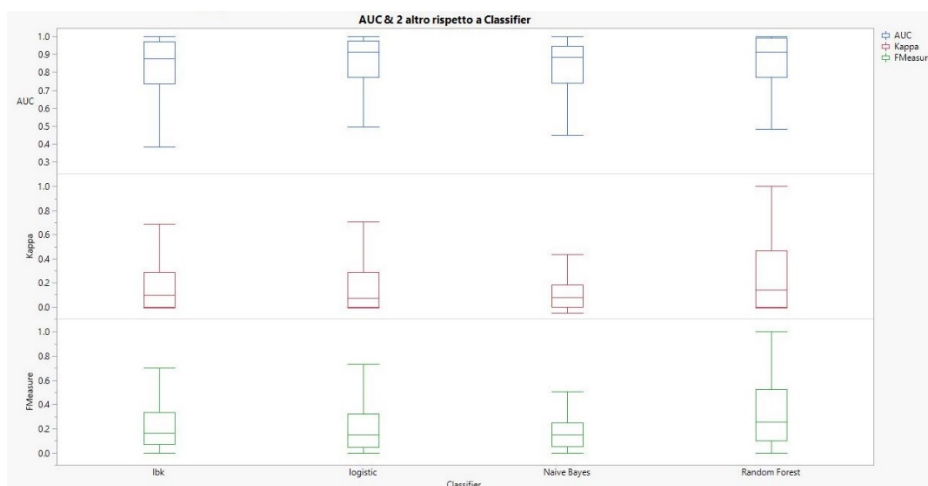


Figura 47 Classificatore metodo increment Syncope

Syncope Moving Window risulta avere un picco massimo e una mediana superiore dello 0.1 rispetto Syncope Increment.

RIFERIMENTI SUL WEB

1. <https://docs.atlassian.com/fisheye-crucible/latest/wadl/crucible.html#d2e221>
2. <https://docs.atlassian.com/software/jira/docs/api/REST/7.6.1/#api/2/project-getAllProjects>
3. <https://developer.atlassian.com/cloud/jira/platform/rest/v3/#version>
4. <https://confluence.atlassian.com/display/JIRASOFTWARESERVER083/Advanced+searching>
5. <https://asankhaya.github.io/pdf/automated-identification-of-security-issues-from-commit-messages-and-bug-reports.pdf>
6. <https://github.com/apache/parquet-mr/tags>