REPORT ISW2

MISURA DELLA STABILITÀ DI UN ATTRIBUTO DI PROGETTO

&

PREDIZIONE DELLA DIFETTOSITÀ DEI FILES SORGENTI

A cura di

ALESSANDRO AMICI

Matricola : 0280073 Email : a.amici@outlook.it

Nota Introduttiva: tutte le immagini presenti in questo report si possono trovare all'interno della stessa repository di Github nelle cartelle "ImmaginiD1" ed "ImmaginiD2" con migliore risoluzione e maggiore leggibilità.

1 Introduzione

Il seguente report si pone l'obiettivo di descrivere il processo di *analisi* e *studio* effettuato su progetti *open source* di **Apache**, utilizzando strumenti come lo **Statistical Control Process**, **Versioning Control Systems**, **Ticket Tracking** e **Machine Learning** per fornire risultati **quantitativi** e **statisticamente** significativi.
Lo studio è stato diviso in **due deliverables** principali, le quali verranno analizzate in maniera separata nei rispettivi paragrafi.
Tutti i risultati che verranno presentati nel report sono frutto di una *automazione* fornita mediante *codice Java*, pertanto sono facilmente ottenibili per un progetto qualsiasi.

Il codice sorgente delle applicazioni Java è presente nella repository:

https://github.com/CecBazinga?tab=repositories

Entrambe le applicazioni sono quindi presenti sulla piattaforma di **GitHub** e sono connesse a **Travis CI** per il build automatico mediante **Ant** e a **SonarCloud** per l'analisi del codice riguardante eventuali *code smells, vulnerabilità, eccessi di Cyclomatic Complexity*.
Nella repository sono anche presenti i risultati forniti dall'esecuzione dell'applicativo ma non formattiti ne presentati (formato raw) che sono liberamente consultabili nella cartella `csv` di ognuna delle due repositories e dei file di *logging* (`Logger.log`)

2 Tecnologie

L'insieme di tecnologie utilizzate per lo sviluppo ed il mantenimento degli applicativi e l'analisi dei dati risultanti dall'esecuzione di questi ultimi sono:

- **Java JDK 1.8** come enviroment di sviluppo

- **Eclipse IDE** come IDE

- **Github** e **TortoiseSVN** come sistema di **Versioning Control** e **Remote Repository**

- **Github** e **Jira** per il raccoglimento di informazioni

- Le librerie **JGit.jar, json.jar** e **weka.jar** rispettivamente per la connessione con Github, parsing di file JSON e per la valutazione di modelli di **ML**

- **Ant** per la build in locale

- **Travis CI** la build in remoto

- **Sonar Cloud** per l'identificazione di difetti relativi alla qualità software

- **Excel** per manipolazione ed estrapolazione dei dati e realizzazione di alcuni grafici

- **JMP** per al realizzazione dei grafici più complessi

3 Prima Deliverable

3.1 Descrizione

Il primo deliverable ha come scopo quello di **misurare la stabilità** di un **attributo** di **progetto**, in particolare il numero di **bug corretti (fixed) per ogni mese.**
I progetti presi in analisi sono : **Accumulo** , **Bookkeeper** e **OpenJpa.**
Una trattazione riguardante la **stabilità** di un **attributo di progetto** è essenziale nel momento in cui si vuole **monitorare**, **controllare** e **predire** il comportamento dello stesso durante il **processo di sviluppo**. Molteplici ricerche suggeriscono che la **stabilità del progetto** sia una caratteristica fondamentale dello sviluppo di un **prodotto** di **qualità maggiore**. L'attività di misurare permette di **prendere decisioni** basate **su evidenze oggettive** portando ad una maggiore qualità sia del **processo** che del **prodotto** finito.
Lo strumento principale utilizzato per misurare la **stabilità** dell'attributo *bug fix/mese* è stato lo **Statistical Control Process**, il quale mette in relazione il numero di bug fix per ogni mese con lo **scarto quadratico medio** (o *deviazione standard*) al fine di evidenziare quali valori ricadono fuori dai valori limite per la *stabilità* del *processo*.

Di seguito riportato il link a **SonarCloud** per la prima deliverable:

https://sonarcloud.io/dashboard?id=CecBazinga_ISW2_Deliverable1

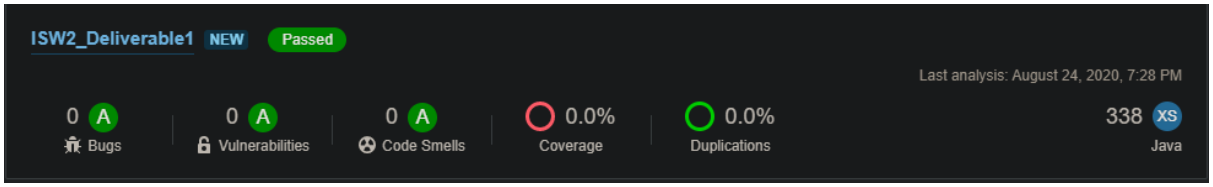

Figura 1 Profilo SonarCloud relativo alla prima deliverable

3.2 Applicativo

Il calcolo della metrica di *bugFix/mese* è stato effettuato usando un applicativo Java (SDK 1.8) che automatizza il processo di *raccoglimento dei dati* dalle varie fonti interessate e del loro processamento:

- Viene utilizzato il sistema di **Ticket Tracking** offerto da **JIRA** per il raccoglimento dei **ticket** relativi a **bug risolti** e le varie **release** con le **rispettive date**. Per fare ciò è stata fatta una interrogazione ai server di JIRA che ritorna un file "`.json`" , il quale , una volta processato , ritorna le informazioni volute.

- Viene invece usato il sistema di **Repository** offerto da **GitHub** per ottenere informazioni sulla data dell'effettivo fix di un determinato bug, cioè *l'ultimo commit* , temporalmente parlando, che *presenta uno specifico ticket*.

Una volta ottenuti i **ticket** e le varie **versioni** da **JIRA**, questi vengono salvati su due *array* distinti. Successivamente si utilizzano le informazioni sui ticket precedentemente ottenute per recuperare la *data dell'ultimo* commit relativo ad un determinato ticket, utilizzando il sistema di **repository** offerto da Github.

La scelta di incrociare le informazioni presenti sulle due piattaforme, anziché usare le informazioni presenti solo su JIRA, proviene dalla consapevolezza che la data del *bug fix* riportata come campo per ogni ticket potrebbe non essere coerente con l'effettiva data in cui il bug in questione è stato risolto, la quale corrisponderà più precisamente alla data in cui è stato effettuato l'ultimo commit relativo al ticket preso in esame.

Una volta raccolti e analizzati, i dati ottenuti vengono *formattati* e *salvati* all'interno di file **`.csv`**, uno relativo ai bug fix per mese e un altro relativo ai commit per mese del progetto preso in analisi.
Si è scelto di realizzare la creazione di questo secondo file csv per comprendere meglio l'andamento della produttività durante il processo di sviluppo del software.

Di seguito vengono riportati gli esempi di come si presentano questi due files:

Month	FixedBugs
Oct-11	17
Nov-11	13
Dec-11	8
Jan-12	27
Feb-12	18
Mar-12	11
Apr-12	20
May-12	17
Jun-12	11
Jul-12	18
Aug-12	2
Sep-12	6
Oct-12	21

Figura 2 Esempio csv dei bug relativo al progetto "Accumulo"

Month	CommitsPerMonth
Oct-11	94
Nov-11	133
Dec-11	80
Jan-12	199
Feb-12	137
Mar-12	142
Apr-12	65
May-12	101
Jun-12	49
Jul-12	63
Aug-12	24
Sep-12	56
Oct-12	110

Figura 3 Esempio csv dei commit relativo al progetto "Accumulo"

Inoltre per tutti i file relativi al numero di bug per mese sono state calcolate le seguenti statistiche:

- **Media** dei bug fix
- **Deviazione standard** usata per il computo del **lower/upper limit**

E' stata posta particolare attenzione sulla **validità** dell'output piuttosto che sull'ottimizzazione del software: questo poiché l'applicativo sviluppato è pensato per essere eseguito *una tantum* e *non iterativamente*, in quanto si trarrebbero pochi benefici da un'eventuale ottimizzazione.

L'interfaccia con **GitHub** è stata creata usando l'**API JGit** insieme ad altre librerie di supporto, tutte **embedded** nel sorgente per garantire massima portabilità ed evitare di dover configurare l'ambiente di esecuzione se l'applicativo venisse eseguito su una macchina diversa da quella di sviluppo.
Il codice è stato creato pensando ad **un'alta modularità** al fine di poter **riutilizzare** le componenti, evitare **design** considerati **problematici** (es. Metodi estremamente grandi, Classi *tuttofare*, ecc.), fornire un buon livello di **leggibilità** del codice (da che ne consegue un mantenimento/aggiornamento del codice più semplice) e un'**organizzazione** basata su **funzionalità/contesto di operazione** delle classi.

Di seguito viene riportata la struttura del progetto:

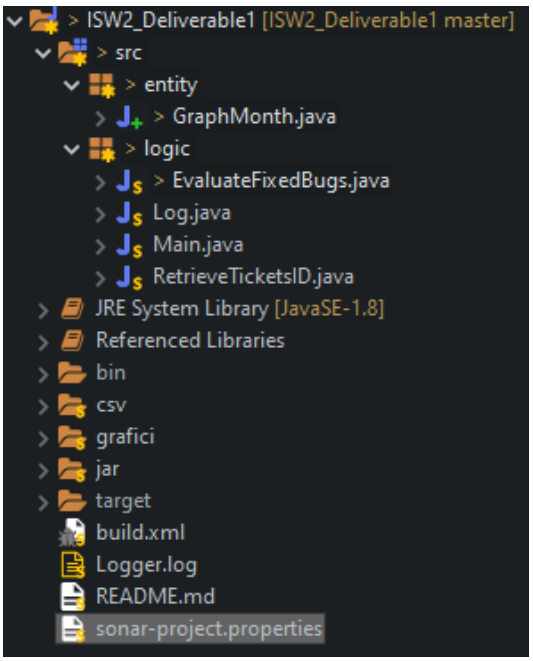

Figura 4 Struttura progettuale della prima deliverable

Per fornire dei dati sull'esecuzione del nostro *parser* è stato implementato un **logger,** il quale fa distinzioni su livello di **gravità** del messaggio (in particolare in informazioni ed errori), che vengono simultaneamente riportati sia sulla console dell'utente che su di un file "`Logger.log`" durante tutto il tempo di esecuzione dell'applicativo.

Le scelte che sono state fatte relativamente al **raccoglimento, interpretazione** ed **esportazione** dei dati sono le seguenti:

- La data del **fix** di ogni **bug ticket** su JIRA è stata assunta *meno affidabile* di quella su **GitHub**, quindi da ignorare.

- Nel caso vi sia presente un ticket su JIRA ma non un commit relativo su GitHub, il ticket viene **scartato** (Ticket without commit).

- Se il **lower limit** è sotto lo **zero**, allora viene automaticamente impostato a 0.

- Se uno dei punti del grafico non è nell'intervallo [lowerLimit, upperLimit] il processo è considerato **instabile**.

3.3 Analisi dati e grafici

Da ogni file csv relativo ai bug fix/mese prodotto con l'applicativo sono stati calcolati i valori di **media** e **lower/upper limit**, questi ultimi secondo la formula:

lower/upper limit = mean -/+ 3 * (standard_deviation)

NB. In caso di negatività del lower limit (priva di senso in questo contesto) questo è stato schiacciato a 0. I lower/upper limit servono per **classificare** la **stabilità** di un **processo di sviluppo**, infatti l'eventualità di valori che ricadano fuori dall'intervallo definito da tali limiti è simbolo di **instabilità** del processo in un intorno temporale del punto.

Per quanto riguarda il progetto **ACCUMULO** abbiamo il seguente andamento:

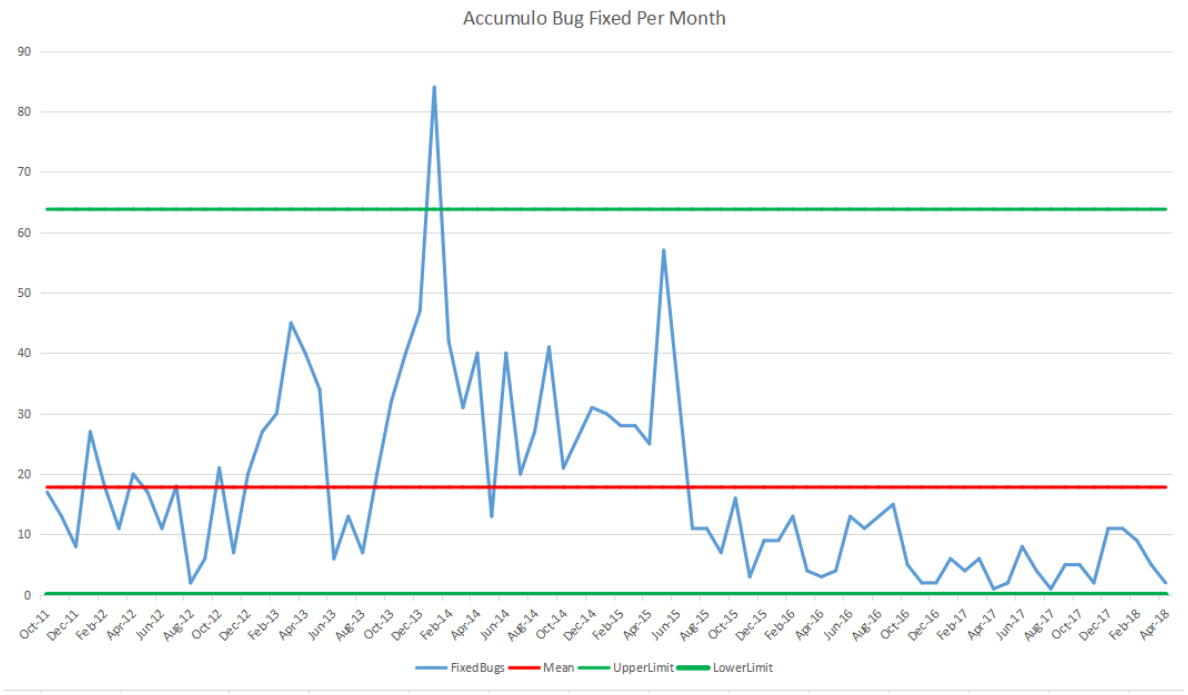

Figura 5 Grafico dei bug fix/mese per il progetto "Accumulo"

Di seguito invece il grafico dei commit per mese dello stesso progetto:

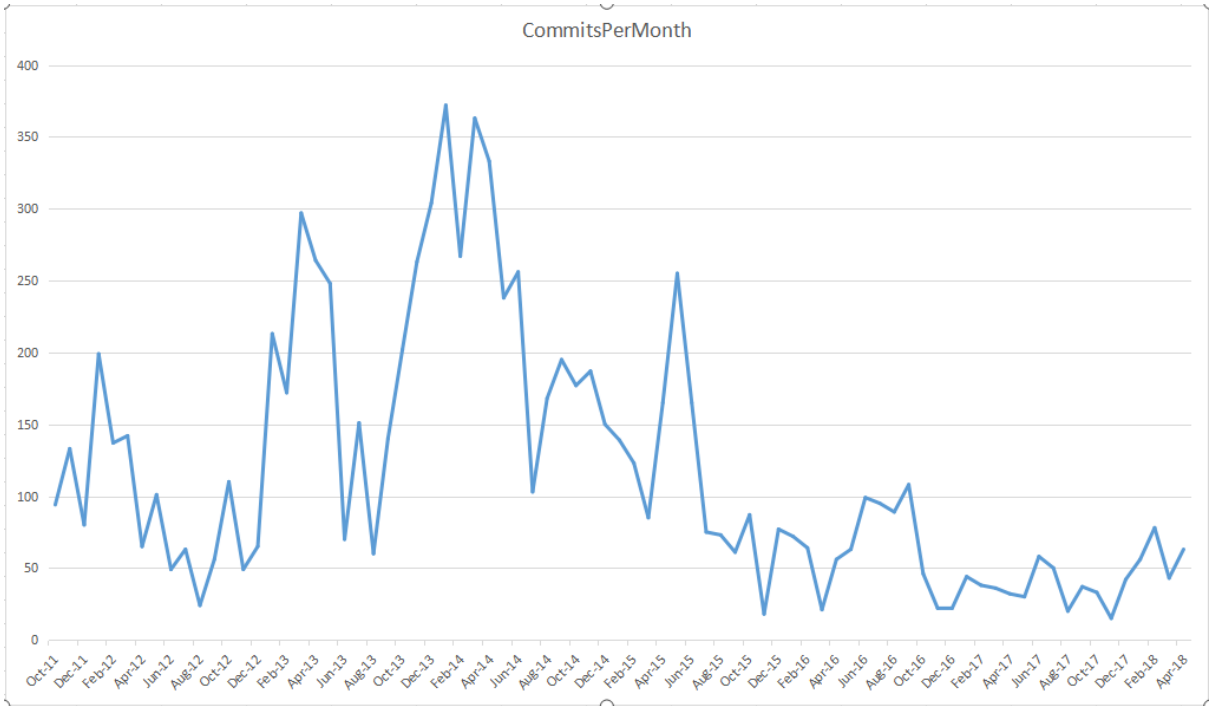

Figura 6 Grafico dei commit/mese per il progetto "Accumulo"

Così come ci si aspettava, si può notare che la quantità di bug fix effettuati è linearmente dipendente con la produttività dell'azienda in quel mese.

E' evidente come con il tempo il processo di sviluppo di **ACCUMULO** si sia stabilizzato per quanto riguarda il numero di bug fix: dall'Agosto del 2015 i valori si attestano tutti vicino alla media, o compresi tra questa ed il lower limit, senza grosse variazioni, dopo aver avuto invece nei primi anni di sviluppo dei picchi molto elevati, anche oltre l'upper limit , indicazione probabilmente dell'instabilità del processo nei suoi primi anni di vita.
Si può notare inoltre che anche i commit si sono **stabilizzati** a partire dallo stesso periodo ed hanno assunto un andamento più **costante** nel tempo, indicazione di una maggiore ***maturità*** del progetto e dell'organizzazione.
Durante le **fasi iniziali** dello sviluppo è facile vedere come il processo fosse più **immaturo** e **instabile**, comportando oscillazioni più marcate. Inoltre, a *Gennaio 2014* viene registrato anche un valore fuori dall'*upper bound*, ulteriore sintomo di *instabilità*, che va però contestualizzato: è evidente che in quel periodo è avvenuto un incremento della produttività significativo (il picco è supportato da un altro picco corrispettivo, questa volta nell grafico dei commit).

Per quanto riguarda il progetto **BOOKKEEPER** abbiamo il seguente andamento:

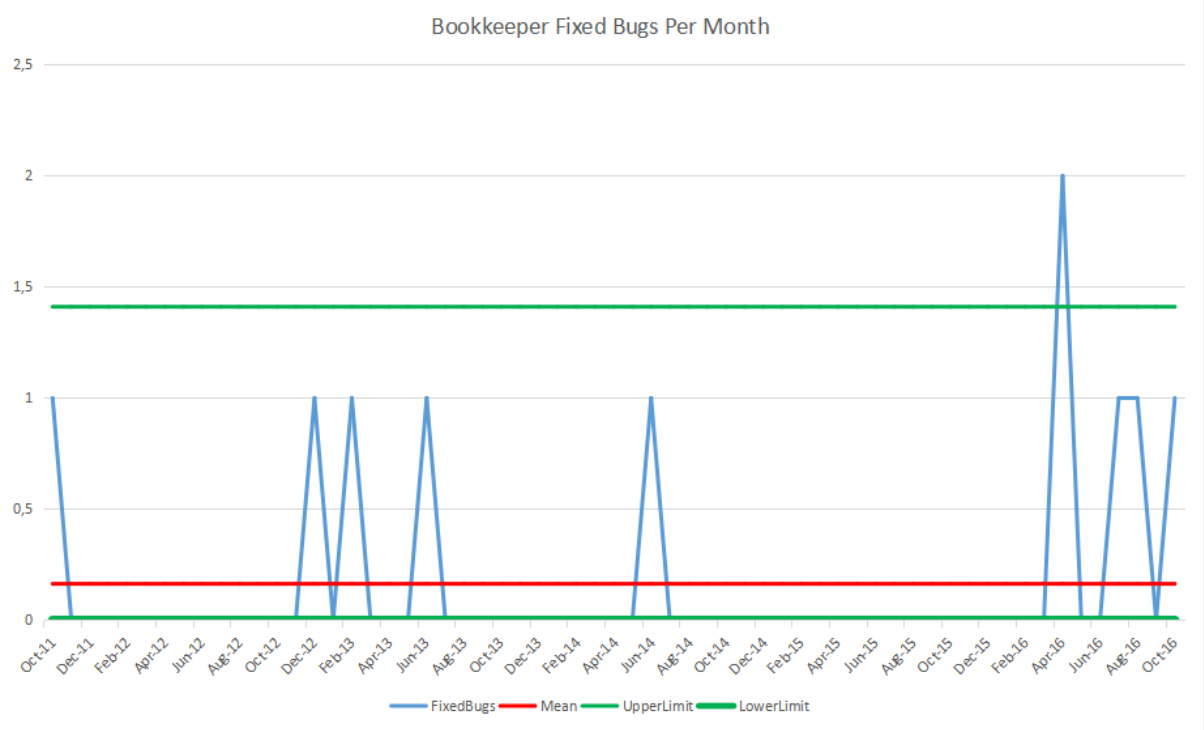

Figura 7 Grafico dei bug fix/mese per il progetto "Bookkeeper"

Di seguito invece il grafico dei commit per mese dello stesso progetto:

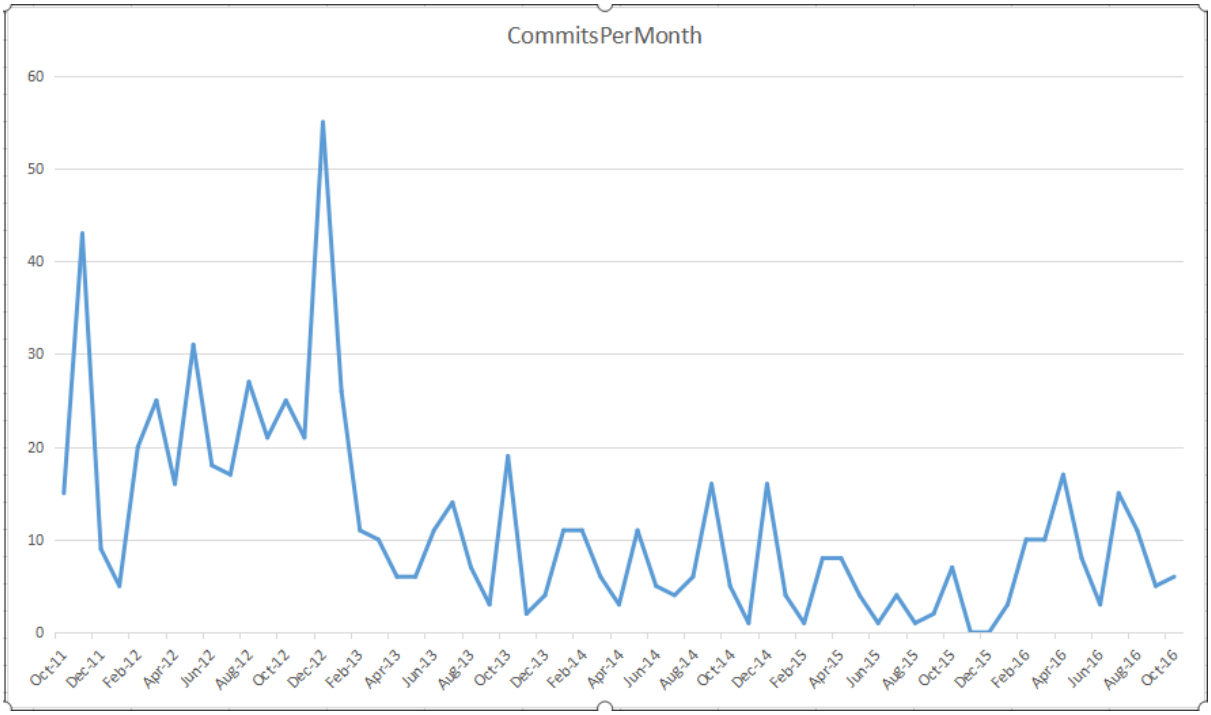

Figura 8 Grafico dei commit/mese per il progetto "Bookkeeper"

Con il progetto Bookkeeper si va a sperimentare uno scenario completamente diverso dal precedente.

Iniziamo con la premessa che i dati sono di un ordine di grandezza inferiore a quelli del progetto "Accumulo" sia per quanto riguarda i bug fix che i commit: la scarsità di dati ha reso difficile un'analisi accurata dell'andamento del processo di sviluppo come è possibile anche vedere dal grafico dei bug fix/mese che presenta una natura discreta.

Ciò già di per sè costituisce un indice dell'**immaturità** ed **instabilità** del processo soprattutto nelle fasi iniziali che vanno dall'Ottobre 2011 all'Aprile 2013. Dopo questa fase i commit si sono **stabilizzati** intorno al valore di 10/mese, tuttavia a differenza dello scenario di Accumulo, qui risulta molto difficile intravedere una proporzionalità tra la produttività ed i bug fix che però diventa un pò più evidente nelle ultime fasi di processo dall'Aprile 2016 in poi, dove a maggiore attività di produzione corrisponde una quantità superiore di bug risolti come conferma anche il "picco" dei bug che si ha proprio nell'Aprile 2016 ed eccede l'upper limit.

Tuttavia si ribadisce ancora una volta che la scarsità dei dati per questo progetto, confermata dal suo andamento discreto, ne rende difficile un'analisi accurata.

Infine andiamo ad analizzare il progetto **OPENJPA**:

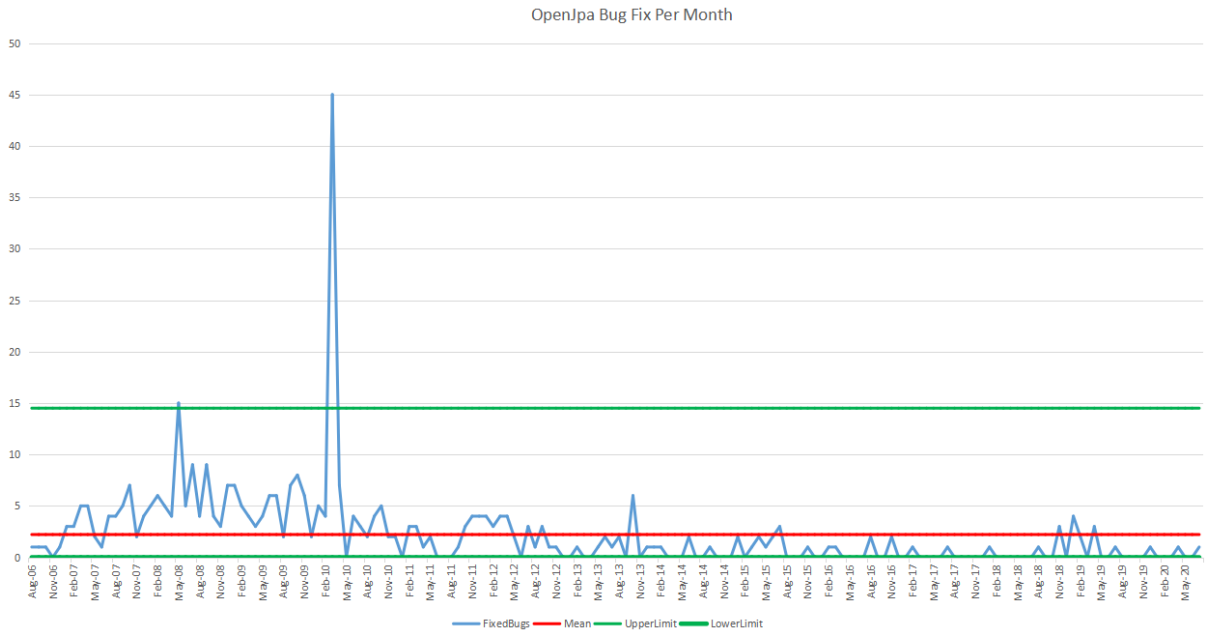

Figura 9 Grafico dei bug fix/mese per il progetto "OpenJpa"

Di seguito invece il grafico dei commit per mese dello stesso progetto:

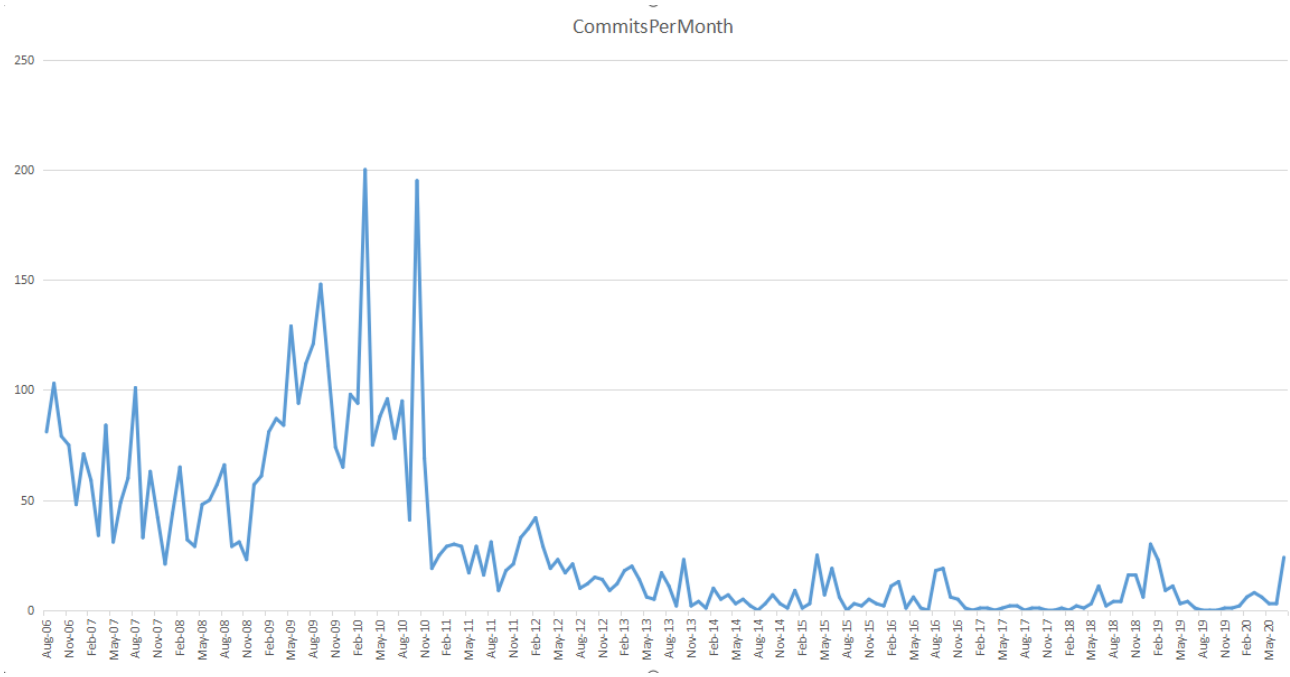

Figura 10 Grafico dei commit/mese per il progetto "OpenJpa"

Tra i 3 progetti è quello in sviluppo da più tempo e su cui si hanno il maggior numero di informazioni.

Si evidenzia molto bene l'andamento del processo di sviluppo, la sua linearità nonchè il costante miglioramento nel tempo: dall'Agosto 2006 al Novembre 2010 il processo è stato caratterizzato da **instabilità,immaturità** ma anche elevata **produttività**: infatti si hanno due picchi nel Maggio 2008 e nel Marzo/Aprile 2010 che lo testimoniano sia relativamente ai bug risolti che ai commit realizzati.

Nel novembre 2010 si ha un elevato picco di produttività che non è corrisposto però da un picco nella risoluzione dei bug: questo potrebbe essere interpretato negativamente all'inizio ma, in seconda battuta, si può notare che da questo istante in poi il processo si stabilizzi sia dal punto di vista della produttività, che si aggira intorno ai 20 commit/mese, sia per la risoluzione dei bug che inizia ad oscillare in modo stabile intorno ad una media di 2 al mese, indice del fatto he il processo ha raggiunto una certa **maturità** e **stabilità** e ciò si può ragionevolmente pensare che abbia comportato una **diminuzione** della presenza stessa di **bug** nel prodotto.

4 Seconda Deliverable

4.1 Descrizione

Il secondo deliverable ha come scopo quello di **eseguire uno studio empirico,** basato su modelli di **Machine Learning,** al fine di identificare le **Classi difettose** di un progetto, con tecniche di **Sampling** e **feature selection** offerte dall'API di **Weka**, basandosi sulle metriche dei file che compongono il progetto stesso, per poi estrapolare quale combinazione di **tecnica di sampling/tecnica di feature selection** produca risultati migliori per ogni **classificatore**.
Questo studio proviene dalla necessità di volere catalogare *i file di progetto* come *potenziali difettosi*, basandosi sulle proprietà del file stesso e alcune altre caratteristiche.
Le motivazioni per cui si ha tale necessità sono molteplici: un esempio è capire quali sono i file che devono essere soggetti a **test più esaustivi** rispetto ad altri, rendendo quindi possibile fare una scelta motivata da dati ed evidenze oggettive e potendo cosi impiegare i budget di progetto in modo più mirato e proficuo.
I progetti presi in analisi sono **BOOKKEEPER** e **OPENJPA**.
Il deliverable si divide in due fasi:

- **Acquisizione** delle metriche e della **difettosità** dei file `.java` del progetto mediante **JIRA** e **GitHub**

- **Analisi** dei dati ottenuti successivamente all'elaborazione del dataset ricavato nella prima fase, con tecniche di **Machine Learning.**

La **prima fase** consiste nell'acquisire la conoscenza se una determinata classe è **difettosa** in una determinata **Release,** insieme all'acquisizione di altre **metriche**, la cui scelta è stata effettuata prendendo in considerazione quali caratteristiche possono essere più significative per la *difettosità* di un file (una metrica come la **size** di un file in questo scenario è più significativa che il **numero di commenti**).
Per l'obiettivo posto in questo studio, ha senso considerare solo ed esclusivamente **files** contenenti **codice java**, scartando i file restanti. Inoltre, vengono **rimosse** il **50% delle release**: questo perchè prima di poter definire un file difettoso è necessario accorgersi della presenza di un bug, ma spesso il tempo che intercorre tra l'introduzione di un bug e la sua rilevazione è non trascurabile (**Class Snoring**), portando così all'incertezza nel definire un file difettoso o meno nelle release più recenti.

L'analisi di release passate non rappresenta la certezza che un file sia defective o meno, piuttosto è meno probabile che il bug non sia stato scoperto: infatti è stato visto che il **Missing rate** diventa del *10%* quando si vanno ad eliminare il *50% delle releases*, portando a risultati decisamente più accurati.

L'output della prima fase contiene, per ogni file di ogni release, l'insieme delle metriche calcolate e la sua eventuale *difettosità*.
Questo output viene poi usato come input della seconda fase.
L'analisi di dati ottenuti mediante modelli **statistici**, o come nel nostro caso mediante modelli di **ML**, permette di effettuare operazioni di **predizione** di dati futuri o non noti.
La **seconda fase** utilizza le informazioni ottenute nella prima per fare valutazioni su quale modello di Machine Learning produca risultati **più accurati** con i **classificatori scelti** nel marcare un file come difettoso o meno.

La **difettosità** è una di quelle caratteristiche che non è sempre possibile definire per un file, specialmente in mancanza di dati essenziali (come per esempio **IV, OV** e **FV**) o nell'eventualità in cui la si vuole definire per una classe in un periodo relativamente "recente".
Esistono vari metodi per lo studio della difettosità, ma non è sempre possibile stabilire con assoluta certezza se si ha a che fare con una classe difettosa o meno ed ecco perché l'impiego di modelli di ML permette di avere una maggiore *confidenza* sull'attribuzione della qualità difettosità.

Di seguito si riporta il link SonarCloud relativo alla repository della deliverable2:

https://sonarcloud.io/dashboard?id=CecBazinga_ISPW2_Deliverable2

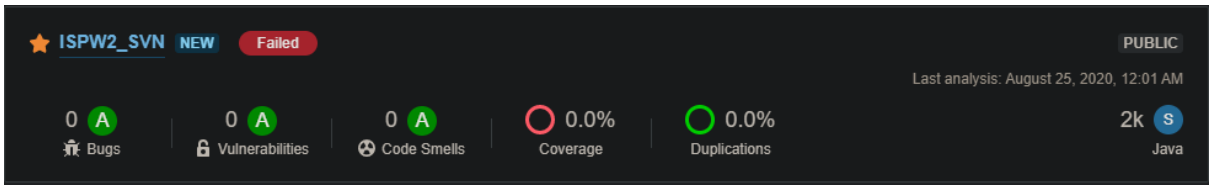

Figura 11 Profilo SonarCloud relativo alla seconda deliverable

4.2 Applicativo

Di seguito viene riportata la struttura del progetto:

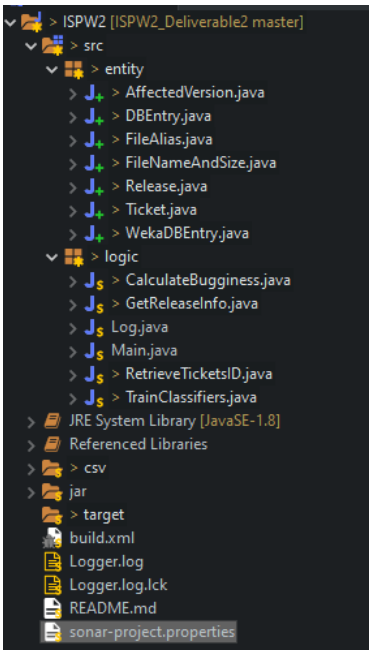

Figura 12 Struttura progettuale della seconda deliverable

4.2.1 Calcolo metriche,bugginess e creazione dataset

La prima parte del Deliverable consiste nell'ottenere le **metriche** e la **bugginess** di una **classe** in una determinata **release**, *iterato per tutti i file presenti per tutte le release*. Le **release** sono state raccolte usando **JIRA**.
L'interrogazione restituisce un file **json** che viene scansionato per ottenere le informazioni relative alle release che vengono poi ordinate cronologicamente per essere usate nella raccolta delle informazioni relative ai **file** e alle **metriche** presenti in ognuna di esse.

Sono state quindi misurate un insieme di **metriche relative ad i ticket** raccolti, così da avere una panoramica descrittiva del progetto a cui ci si stava approcciando: sono state raccolte in un file **csv** presente nella cartella "csv" con la dicitura finale "**TicketMetrics**".

Le **metriche relative ai file java** sono state computate usando i *log* delle commit fornite da **Github** (interfacciata sempre usando la API di **JGit**): anche in questo caso per accedere alle informazioni, è necessario fare prima il *clone* della repository, per poi estrapolare le metriche scorrendo ogni commit.
Le **commit** utilizzate sono solamente quelle del **master branch**, mentre quelle relative a cicli paralleli di sviluppo e successivamente **merged** nel master sono state scartate. Pur venendo scartate, non avviene un'effettiva perdita di informazione: quando i due flussi di lavoro vengono "fusi" nel master branch, vengono prese in considerazione tutte le operazioni fatte nel flusso parallelo, con l'unica differenza che vengono calcolate tutte insieme e non commit per commit.

Prima di procedere con il calcolo delle metriche bisogna avere la **lista dei file** su cui calcolarle. Per fare ciò, si prendono per ogni versione esistente l'*ultima commit pubblicata prima del rilascio della versione,* ottenendo il **file tree** contenente *tutti i file presenti* in quel momento dello sviluppo del progetto in quella release.

Le **14 metriche** considerate sono le seguenti:

- **Size(SLOC):** numero di linee di codice di un file.
- **Number of Fix:** numero di bug fix per il file in quella release.
- **Number of Revision:** numero di commit che hanno interessato il file in questione.
- **Number of Authors:** numero di autori che hanno effettuato operazioni sul file in esame.
- **LOC Touched:** somma di linee di codice *aggiunte, eliminate e modificate.*
- **LOC Added:** numero di linee di codice *aggiunte*.
- **Max LOC Added:** il numero *massimo* di linee di codice *aggiunte* in una revisione.
- **Average LOC Added:** numero *medio* di linee di codice *aggiunte.*
- **Change Set Size:** somma del numero di file *committed* insieme al file in esame.
- **Max Change Set Size:** numero *massimo* di file *committed* insieme al file in esame in una revisione.
- **Average Change Set Size:** numero *medio* di file *committed* insieme al file in esame.
- **Churn:** somma di linee di codice *aggiute* meno il numero linee di codice *eliminate.*
- **Max Churn:** il massimo tra le revisioni di linee di codice *aggiute* meno il numero linee di codice *eliminate.*
- **Average Churn:** la media tra le revisioni di linee di codice *aggiute* meno il numero linee di codice *eliminate.*

Si è scelto di calcolare le metriche in modo incrementale cosi da mantenere *uno storico dell'evoluzione di ogni file*, infatti ad esempio i valori di **number of revision** e **number of authors** vengono incrementati tra una release e un'altra.
Le metriche vengono calcolate unendo le varie operazioni fatte con le **commit** tra una release e un'altra, sfruttando il meccanismo di **git diff** per estrapolare le operazioni fatte in una determinata commit.
Nel caso di operazioni di **RENAME**, il "nuovo file" non viene considerato come nuovo, ma le metriche calcolate fino a quel momento vengono trasferite sotto il nuovo nome del file grazie ad un **lista** di **file alias** che mantiene per ogni file tutto l'insieme di nomi con cui è stato conosciuto e li associa all'ultimo nome con cui è noto quel file .
La **Bugginess** viene calcolata mediante l'impiego di ticket su **JIRA** : Per ogni ticket presente, viene prelevato **l'ID del ticket**, **affected versions** e **data di creazione**.
L'ID del ticket viene usato per trovare le commit che sono state effettuate per fixare il bug citato nel ticket.
Queste commit sono di interesse poiché i file che vengono **modificati** o **eliminati** per fixare un bug sono da considerare **defective**.

L'**affected version** è l'intervallo di versioni **[Injected Version, Fixed Version)** e serve per trovare quali versioni sono affette dal bug in esame, ed è quindi necessario avere la conoscenza di questo dato per sapere quando il bug è stato introdotto, e quindi da quale release il file va marcato come defective.
La **data di creazione** del ticket viene usata per *computare* l'**Opening Version**, valore necessario per l'applicazione del **Proportion**.
Un ulteriore dato necessario per computare l'intervallo di influenza di un determinato bug è la **fixed version** riportata come campo dei ticket su JIRA.
Quando non presente l'**affected version** nei ticket raccolti su JIRA viene usato il metodo **Proportion**:
la variante implementata è il **Proportion Moving Window** che utilizza l'ultimo 1% dei dati esaminati come finestra da cui estrapolare i valori necessari per il calcolo del proportion da associare al ticket in esame che ne è privo. Poiché non è stato specificato se la quantità fosse un valore fisso o meno è stato implementato in due modi differenti:
-considerando l'1% dei ticket come la percentuale sui ticket totali che si andranno ad esaminare ed avendo cosi una finestra di dimensione fissata
-considerando l'1% dei ticket come la percentuale dei ticket esaminati fino a quel momento: cosi facendo la grandezza della finestra cresce all'aumentare del numero di ticket in esame.
Purtroppo per motivi di tempo non si è riusciti a testare la seconda implementazione del **Proportion** che però sarebbe risultata interessante, in quanto il numero dei ticket considerati nel calcolo cresce proporzionalmente alla quantità di ticket visti, poiché avrebbe richiesto uno studio completo di tutti i valori in output con relativi grafici per poter visualizzare i cambiamenti rispetto all'altra implementazione.
In entrambe le implementazioni nelle fasi iniziali non si hanno sufficienti dati per poter utilizzare il **Proportion** perciò viene calcolato un **valore medio del Proportion** da applicare ai ticket in questa fase**,** secondo il seguente algoritmo:

-Si calcola il proportion da tutti i ticket completi di dati e **conformi** (IV <= OV < FV) secondo la formula:

$$P = (FV - IV) / (FV - OV)$$

-Si memorizza ognuno di questi valori in un unico array
-Si calcola il valor medio dell'array

Quando il numero di **ticket** analizzato è *sufficientemente* rappresentativo del ciclo di vita dei bug nel progetto considerato, allora vi è il passaggio a **Moving Window** in cui il calcolo di **P** viene questa volta effettuato solo tra l'ultimo 1% dei ticket visti.
In entrambi gli scenari, una volta effettuato il computo di P, lo si usa quando si è in presenza di **ticket** incompleti (in particolare **privi di IV**) secondo la seguente formula:

$$IV = FV - (FV - OV) * P$$

Nel caso di **ticket non conformi**, si evita di *computare la proporzione* per usarla nel **Proportion Moving Window** e si distinguono 2 casistiche:

-il **ticket** pur essendo **non conforme** è **completo** di dati e se ne può calcolare l'AV = [IV, FV): in questo caso viene utilizzato per marcare i file toccati da questo come defective nelle release presenti nell'intervallo AV.

-il **ticket** è **non conforme** ed **incompleto** di dati, in questo caso viene scartato.

Come per la prima deliverable, tutte le librerie necessarie sono state **embedded** nel sorgente per garantire massima **portabilità** ed è stata effettuata l'implementazione di un **logger** per tenere traccia delle operazioni effettuate.
L'**output** principale è un **file .csv** (terminante con la dicitura "Bugginess") contenente per ogni **file** di ogni **versione** le **metriche** e la **difettosità**, espressa come valore **binario** (yes/no).

Di seguito un estratto del file csv di output:

Release	Filename	Size	LocTouche	NR	NFix	NAuth	LocAdded	MaxLocAdd	AvgLocAdd	Churn	MaxChurn	AvgChurn	ChgSetSize	MaxChgSe	AvgChgSet	Bugginess
1	src\main\java\org\apache\bookkeeper\benchmark\MySqlClient.java	121	333	4	0	2	235	137	2.0	137	137	1.0	287	278	138.0	no
1	src\main\java\org\apache\bookkeeper\benchmark\TestClient.java	215	436	5	1	2	340	252	3.0	244	252	2.0	287	278	111.0	no
1	src\main\java\org\apache\bookkeeper\bookie\Bookie.java	950	1466	15	5	3	1246	545	12.0	1026	545	10.0	316	278	50.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\BookieException.java	78	150	4	0	3	121	81	1.0	92	81	1.0	289	278	135.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\BufferedChannel.java	170	210	5	2	2	195	168	2.0	180	168	2.0	285	278	103.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\EntryLogger.java	441	795	10	4	3	633	487	6.0	471	487	4.0	314	278	70.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\FileInfo.java	185	216	6	1	3	210	124	2.0	204	124	2.0	288	278	104.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\LedgerCache.java	516	691	8	3	3	621	536	6.0	551	536	5.0	313	278	87.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\LedgerDescriptor.java	132	187	6	0	3	168	133	2.0	149	133	1.0	300	278	113.0	yes
1	src\main\java\org\apache\bookkeeper\bookie\LedgerEntryPage.java	152	171	4	0	3	164	151	2.0	157	151	1.0	298	278	139.0	no
1	src\main\java\org\apache\bookkeeper\bookie\MarkerFileChannel.java	124	147	2	0	1	147	147	1.0	147	147	1.0	279	278	169.0	no
1	src\main\java\org\apache\bookkeeper\client\AsyncCallback.java	123	401	6	0	3	269	126	3.0	137	126	1.0	293	278	92.0	no
1	src\main\java\org\apache\bookkeeper\client\BKException.java	241	289	5	0	3	280	249	3.0	271	249	3.0	289	278	109.0	yes
1	src\main\java\org\apache\bookkeeper\client\BookKeeper.java	495	1532	10	1	3	1040	410	10.0	548	410	5.0	319	278	75.0	yes
1	src\main\java\org\apache\bookkeeper\client\BookKeeperAdmin.java	694	1134	6	2	2	931	714	9.0	728	714	7.0	135	95	38.0	yes
1	src\main\java\org\apache\bookkeeper\client\BookieWatcher.java	174	219	5	1	2	212	204	2.0	205	204	2.0	288	278	123.0	no
1	src\main\java\org\apache\bookkeeper\client\CRC32DigestManager.java	42	56	3	0	2	53	50	1.0	50	50	0.0	285	278	170.0	no
1	src\main\java\org\apache\bookkeeper\client\DigestManager.java	148	229	6	0	3	207	184	2.0	185	184	2.0	288	278	105.0	yes
1	src\main\java\org\apache\bookkeeper\client\DistributionSchedule.java	55	71	4	0	3	66	61	1.0	61	61	1.0	287	278	132.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerCreateOp.java	139	263	9	3	3	210	167	2.0	157	167	1.0	315	278	81.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerDeleteOp.java	71	103	5	1	3	92	80	1.0	81	80	1.0	305	278	128.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerEntry.java	71	182	4	0	2	133	83	1.0	84	83	1.0	288	278	151.0	no
1	src\main\java\org\apache\bookkeeper\client\LedgerHandle.java	674	2162	13	3	3	1462	547	14.0	762	547	7.0	319	278	59.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerMetadata.java	215	328	8	1	3	288	198	3.0	248	198	2.0	296	278	82.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerOpenOp.java	153	295	10	3	3	232	140	2.0	169	140	2.0	316	278	74.0	yes
1	src\main\java\org\apache\bookkeeper\client\LedgerRecoveryOp.java	167	252	7	1	3	224	178	2.0	196	178	2.0	292	278	93.0	yes
1	src\main\java\org\apache\bookkeeper\client\MacDigestManager.java	53	85	5	1	3	76	67	1.0	67	67	1.0	287	278	107.0	no
1	src\main\java\org\apache\bookkeeper\client\PendingAddOp.java	141	199	5	0	3	182	138	2.0	165	138	2.0	291	278	127.0	yes
1	src\main\java\org\apache\bookkeeper\client\PendingReadOp.java	144	213	5	0	3	193	170	2.0	173	170	2.0	291	278	127.0	yes
1	src\main\java\org\apache\bookkeeper\client\ReadLastConfirmedOp.java	91	151	6	2	2	127	101	1.0	103	101	1.0	185	172	51.0	yes
1	src\main\java\org\apache\bookkeeper\client\ReadOnlyLedgerHandle.java	70	78	1	0	1	78	78	1.0	78	78	1.0	28	28	28.0	no
1	src\main\java\org\apache\bookkeeper\client\RoundRobinDistributionSchedule	69	89	3	0	2	88	87	1.0	87	87	1.0	285	278	170.0	yes
1	src\main\java\org\apache\bookkeeper\client\SyncCounter.java	71	87	3	0	2	86	85	1.0	85	85	1.0	285	278	170.0	no
1	src\main\java\org\apache\bookkeeper\conf\AbstractConfiguration.java	93	107	2	1	1	106	61	1.0	105	61	1.0	63	44	40.0	yes
1	src\main\java\org\apache\bookkeeper\conf\ClientConfiguration.java	195	232	4	3	2	224	216	2.0	216	216	2.0	45	44	14.0	yes
1	src\main\java\org\apache\bookkeeper\conf\ServerConfiguration.java	341	417	4	3	2	397	377	4.0	377	377	4.0	45	44	14.0	yes
1	src\main\java\org\apache\bookkeeper\meta\AbstractZkLedgerManager.java	232	257	2	1	1	255	251	2.0	253	251	2.0	39	36	21.0	no

Figura 13 Esempio dataset per il progetto "Bookkeeper"

4.2.2 Applicazione di modelli di machine learning e predizione

La **seconda parte** dell'applicativo consiste nell'applicare ai dati raccolti differenti modelli di **Classificatori**, **Tecniche di Balancing** dei dati e **Tecniche di Sampling** dei dati e valutare quale combinazione di questi 3 elementi produca **stime più accurate**.
In particolare, come **evaluation tecnique** è stata scelta la **walk forward** piuttosto che la ***k-fold* cross validation,** poichè la prima è la scelta più adeguata quando si lavora con **dati sensibili temporalmente** poiché *preserva l'ordine dei dati* (**Time-series validation**), mentre la cross validation non tiene in considerazione questo fattore.
Per applicare il *walk forward* bisogna **dividere** il **dataset** in porzioni, dove il criterio scelto è la **versione** (o **release**): le singole unità che compongono il walk forward contengono la porzione del dataset relative a tutti i file contenuti in una specifica versione: il **walk k-esimo** contiene i file, con relative metriche e difettosità, appartenenti alla **versione k**, su cui si cerca di fare predizione (**test**) e si usano le precedenti **k-1** versioni per fare l'addestramento del modello(**training**).

Lo scopo è quello di provare ogni possibile combinazione facendo il **prodotto cartesiano** tra:

`{classifier, balancing tecnique, feature selection}`

Dove gli insiemi sopracitati sono costituiti dai seguenti elementi:

-Classifier: **Random Forest , Naive Bayes , IBK**

-Balancing: **No Sampling , Oversampling , Undersampling , Smote**

-Feature Selection: **No Selection , Best First**

Si ottiene così un totale di **24 combinazioni**.

Per le varie possibilità di **Feature Selection** e **Balancing** non è stato fatto un *tuning* ma i parametri sono stati impostati con i valori considerati di *default*.
Questo processo utilizza quindi i dataset estrapolati dalla parte precedente, e quindi viene eseguito utilizzando sia **BOOKKEEPER** sia **OPENJPA**.
I **dataset** vengono quindi **caricati** e **convertiti** in modo che l'API di Weka riesca a lavoraci:
le porzioni di dateset in file csv vengono quindi convertite in **arff** e poi trasformate in tipi di dato **Instances**.
Inoltre i dati, prima di essere processati, subiscono un **pre-processamento** dove viene impostato il **Class index** (l'attributo che su cui si intende effettuare le stime) e i dati vengono **normalizzati.**
La normalizzazione è un passo essenziale per garantire che ogni attributo contribuisca con **lo stesso peso** nelle fasi di **training** del modello, evitando quindi di avere *bias* causato dalla differenza di scala.
Successivamente alla fase di preprocessamento vengono **definiti** i **walk** e memorizzati in strutture dati pronti per essere utilizzati, decretando la fine della fase di inizializzazione.
Iterativamente per ogni walk, vengono **scelte** ed **eseguite** le tutte varie **combinazioni** di **modello** di **ML**, applicando gli eventuali **filtri** a *classificatori/attributi*, per poi fare il **training** e l'**evaluation** sulla porzione di testing del walk.
I risultati ottenuti vengono memorizzati in una struttura dati che viene poi esportata su di un file **.csv** (terminante con la dicitura "classifierMetrics").

Di seguito è riportato un esempio del dataset contenente le performance dei classificatori:

Dataset	TrainingReleases	%DataTraining	%DefectiveTraining	%DefectiveTesting	Classifier	Balancing	FeatureSel	TP	FP	TN	FN	Precision	Recall	AUC	Kappa
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	None	None	53.0	41.0	184.0	23.0	0.5638297872340425	0.6973684210526315	0.8180701754385965	0.4776855918876416
BookkeeperBugginess.arff	1	7	45	25	RandomForest	None	None	67.0	73.0	152.0	9.0	0.4785714285714286	0.881578947368421	0.8307017543859649	0.4356594110115236
BookkeeperBugginess.arff	1	7	45	25	IBk	None	None	63.0	87.0	138.0	13.0	0.42	0.8289473684210527	0.721140350877193	0.3344536328660505
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	UnderSampling	None	53.0	41.0	184.0	23.0	0.5638297872340425	0.6973684210526315	0.8161403508771929	0.4776855918876416
BookkeeperBugginess.arff	1	7	45	25	RandomForest	UnderSampling	None	67.0	78.0	147.0	9.0	0.46206896551724136	0.881578947368421	0.8309941520467836	0.4112767248937749
BookkeeperBugginess.arff	1	7	45	25	IBk	UnderSampling	None	68.0	91.0	134.0	8.0	0.4276729559748428	0.8947368421052632	0.7451461988304093	0.3600833208065973
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	OverSampling	None	52.0	39.0	186.0	24.0	0.5714285714285714	0.6842105263157895	0.8056432748538012	0.4795389048991353
BookkeeperBugginess.arff	1	7	45	25	RandomForest	OverSampling	None	66.0	78.0	147.0	10.0	0.4583333333333333	0.868421052631579	0.8119590643274853	0.4025083461156726
BookkeeperBugginess.arff	1	7	45	25	IBk	OverSampling	None	64.0	95.0	130.0	12.0	0.4025157232704403	0.8421052631578947	0.7376608187134502	0.3083728820838792
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	Smote	None	53.0	44.0	181.0	23.0	0.5463917525773195	0.6973684210526315	0.8088304093567251	0.4597497923866162
BookkeeperBugginess.arff	1	7	45	25	RandomForest	Smote	None	70.0	88.0	137.0	6.0	0.4430379746835443	0.9210526315789473	0.8197076023391813	0.3904519798354086
BookkeeperBugginess.arff	1	7	45	25	IBk	Smote	None	64.0	91.0	134.0	12.0	0.4129032258064516	0.8421052631578947	0.7188304093567252	0.3255965717517564
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	None	BestFirst	52.0	45.0	180.0	24.0	0.5360824742268041	0.6842105263157895	0.8001169590643274	0.4436229205175604
BookkeeperBugginess.arff	1	7	45	25	RandomForest	None	BestFirst	66.0	72.0	153.0	10.0	0.4782608695652174	0.868421052631579	0.816345029239766	0.4317878355357060
BookkeeperBugginess.arff	1	7	45	25	IBk	None	BestFirst	61.0	76.0	149.0	15.0	0.4452554744525547	0.8026315789473685	0.7306432748538012	0.367252650788884
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	UnderSampling	BestFirst	53.0	47.0	178.0	23.0	0.53	0.6973684210526315	0.8030701754385965	0.4422384582803896
BookkeeperBugginess.arff	1	7	45	25	RandomForest	UnderSampling	BestFirst	67.0	78.0	147.0	9.0	0.46206896551724136	0.881578947368421	0.8127192982456141	0.4112767248937749
BookkeeperBugginess.arff	1	7	45	25	IBk	UnderSampling	BestFirst	65.0	82.0	143.0	11.0	0.4421768707482993	0.8552631578947368	0.7435087719298246	0.3748632171303106
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	OverSampling	BestFirst	53.0	45.0	180.0	23.0	0.5408163265306123	0.6973684210526315	0.8065204678362573	0.4538662682106835
BookkeeperBugginess.arff	1	7	45	25	RandomForest	OverSampling	BestFirst	64.0	72.0	153.0	12.0	0.4705882352941176	0.8421052631578947	0.7927777777777778	0.4139082058414463
BookkeeperBugginess.arff	1	7	45	25	IBk	OverSampling	BestFirst	63.0	89.0	136.0	13.0	0.4144736842105263	0.8289473684210527	0.7321637426900585	0.3255865038221597
BookkeeperBugginess.arff	1	7	45	25	NaiveBayes	Smote	BestFirst	59.0	74.0	151.0	17.0	0.4436090225563909	0.7763157894736842	0.7967543859649123	0.3584194130185276
BookkeeperBugginess.arff	1	7	45	25	RandomForest	Smote	BestFirst	69.0	81.0	144.0	7.0	0.46	0.9078947368421053	0.8039181286549708	0.4143191969221244
BookkeeperBugginess.arff	1	7	45	25	IBk	Smote	BestFirst	61.0	76.0	149.0	15.0	0.4452554744525255476	0.8026315789473685	0.7306432748538012	0.367252650788884
BookkeeperBugginess.arff	2	17	34	53	NaiveBayes	None	None	33.0	53.0	149.0	204.0	0.3837209302325581	0.1392405063291139	0.3616472406734346	-0.1166945453465700
BookkeeperBugginess.arff	2	17	34	53	RandomForest	None	None	36.0	51.0	151.0	201.0	0.4137931034482758	0.1518987341772152	0.3637151689852529	-0.0953484227410443
BookkeeperBugginess.arff	2	17	34	53	IBk	None	None	54.0	68.0	134.0	183.0	0.4426229508196721	0.227848101265822	0.4414086978318084	-0.1043969811472041
BookkeeperBugginess.arff	2	17	34	53	NaiveBayes	UnderSampling	None	30.0	46.0	156.0	207.0	0.3947368421052631	0.1265822784810126	0.3754543175836571	-0.0955189726088199
BookkeeperBugginess.arff	2	17	34	53	RandomForest	UnderSampling	None	73.0	81.0	121.0	164.0	0.474025974025974	0.3080168776371308	0.3970317917867736	-0.0902354718052162
BookkeeperBugginess.arff	2	17	34	53	IBk	UnderSampling	None	82.0	91.0	111.0	155.0	0.4739884393063583	0.3459915611814346	0.4435810669745625	-0.1021145446381189
BookkeeperBugginess.arff	2	17	34	53	NaiveBayes	OverSampling	None	32.0	49.0	153.0	205.0	0.3950617283950617	0.1350210970464135	0.3767493837991394	-0.1017508497351987
BookkeeperBugginess.arff	2	17	34	53	RandomForest	OverSampling	None	70.0	76.0	126.0	167.0	0.4794520547945205	0.2953586497890295	0.4041024355600117	-0.0782751963450011
BookkeeperBugginess.arff	2	17	34	53	IBk	OverSampling	None	70.0	80.0	122.0	167.0	0.4666666666666667	0.2953586497890295	0.4538580440322513	-0.0975777636067333

Figura 14 Esempio dataset dei classificatori per "Bookkeeper"

4.3 Analisi Dati

Si è scelto di iniziare questa fase di analisi mostrando una panoramica di ogni progetto in esame attraverso dei grafici introduttivi.

Relativamente al progetto **BOOKKEEPER**:

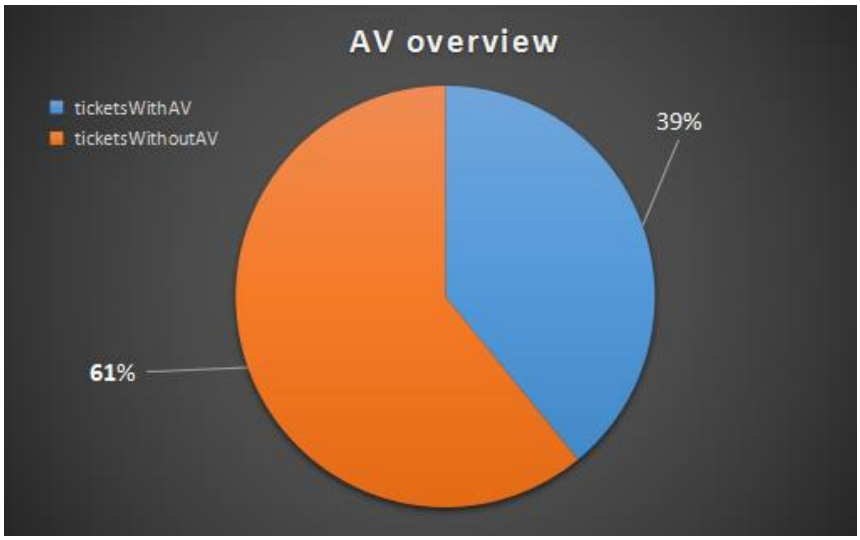

Figura 15 Overview delle percentuali di ticket dotati di AV nel progetto"Bookkeeper"

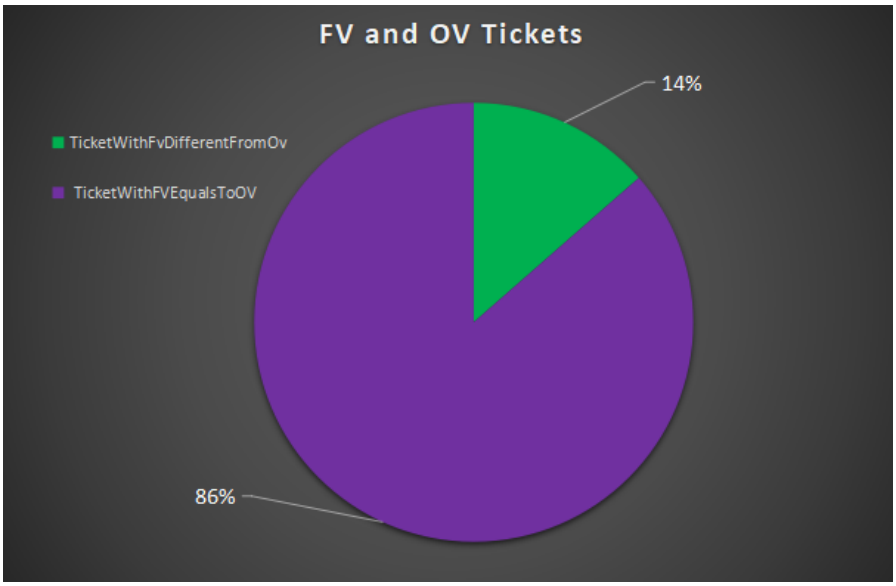

Figura 16 Overview delle percentuali di ticket con OV=FV nel progetto"Bookkeeper"

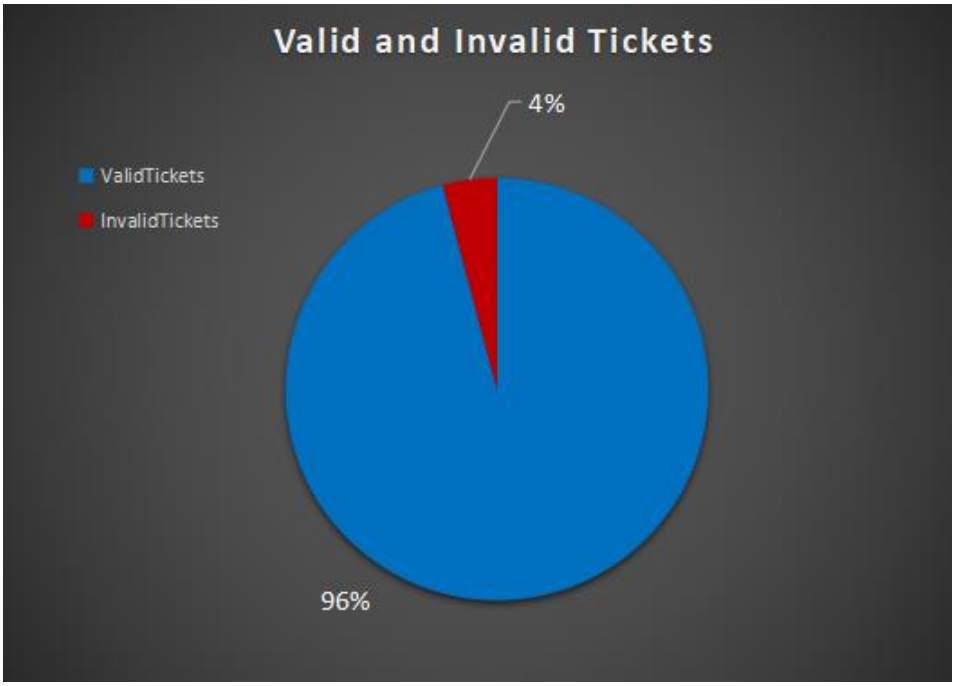

Figura 17 Overview delle percentuali di ticket validi/conformi nel progetto"Bookkeeper"

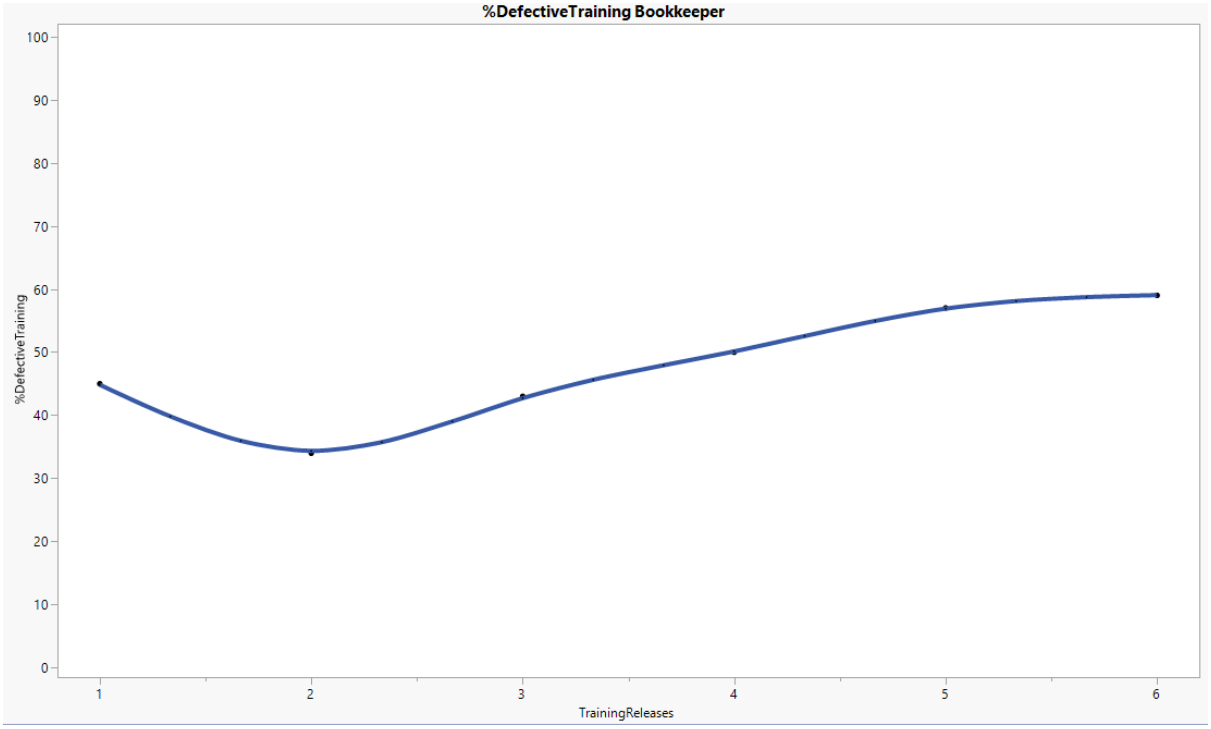

Figura 18 Percentuale dei dati di training difettosi in ogni walk di "Bookkeeper"

Relativamente al progetto **OPENJPA**:

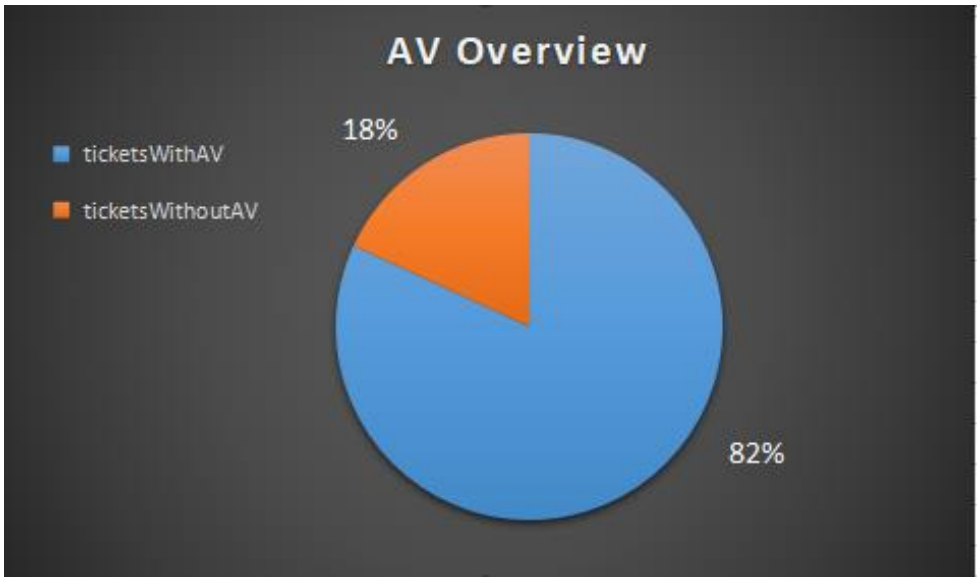

Figura 19 Overview delle percentuali di ticket dotati di AV nel progetto"OpenJpa"

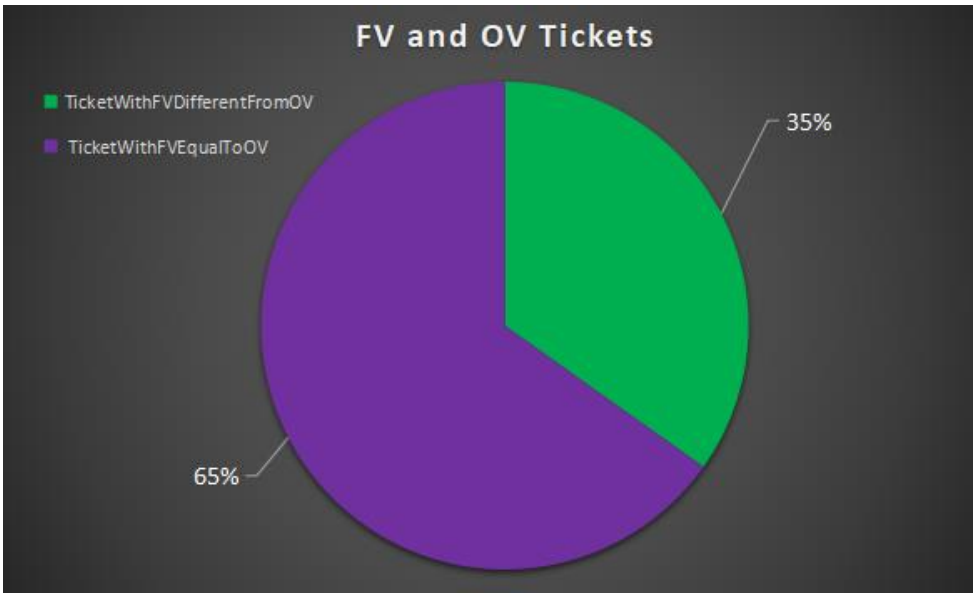

Figura 20 Overview delle percentuali di ticket con OV=FV nel progetto"OpenJpa"

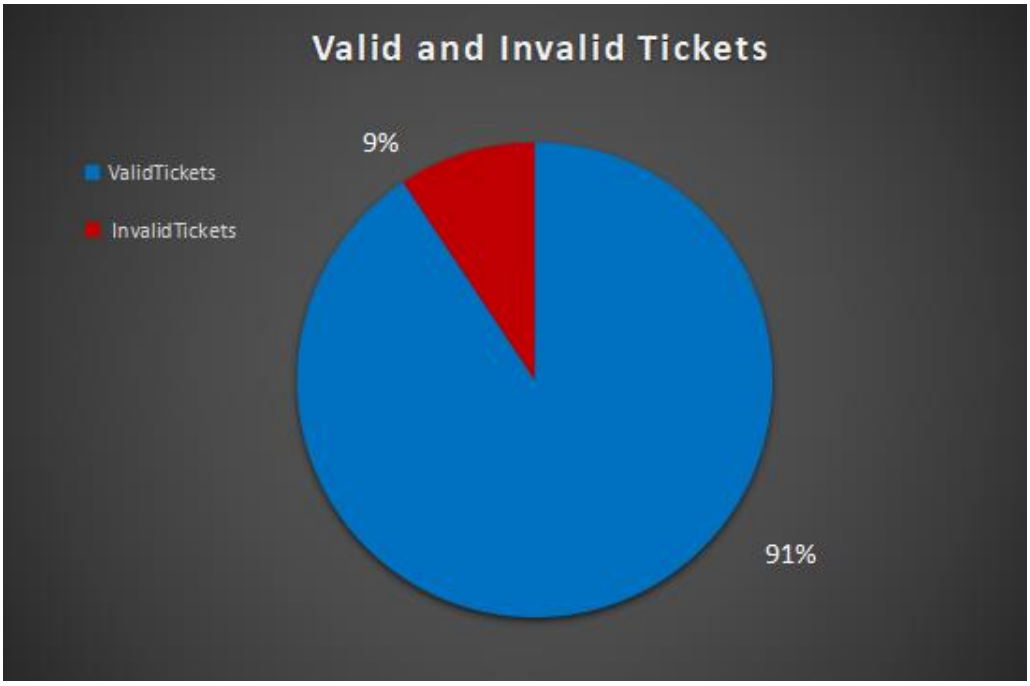

Figura 21 Overview delle percentuali di ticket validi/conformi nel progetto"OpenJpa"

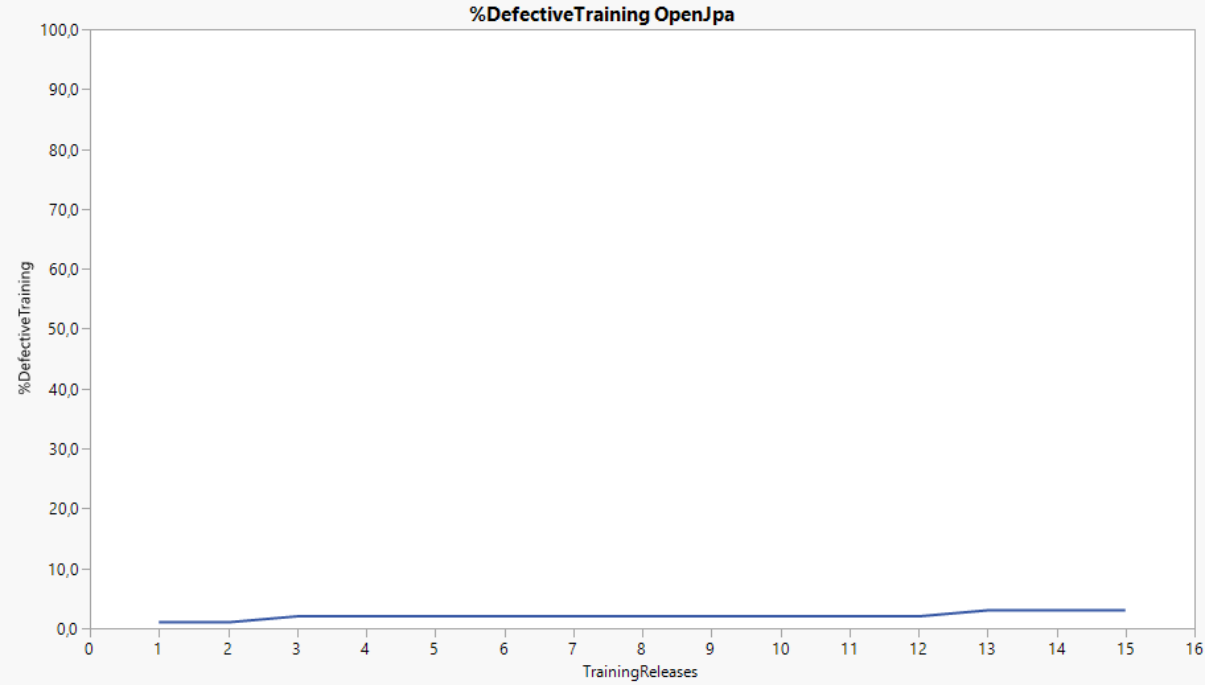

Figura 22 Percentuale dei dati di training difettosi in ogni walk di "OpenJpa"

Dai grafici soprariportati si può notare che **Bookkeeper** si presenta come un progetto molto **più instabile** e **meno maturo** di **OpenJpa**:

- Presenta una percentuale maggiore di ticket privi di AV (per cui Proportion si rende indispensabile)

- Presenta una percentuale maggiore di ticket in cui la OV coincide con la FV

- La percentuale di ticket validi e non è molto simile per entrambi i progetti invece

Come si può notrare **BookKeeper** ha un andamento della percentuale di difettosità dei dati di training molto più **instabile** e **crescente nel tempo**, anche se lo scarso numero di release non rendono possibile un'analisi più esaustiva sull'andamento nel lungo termine (infatti i dati raccolti relativamente ad OpenJpa sono molto piu corposi ed esaustivi rispetto quelli di Bookkeeper) .
OpenJpa invece, oltre che rimanere molto **stabile**, si tiene con valori di defectiveness molto bassi. Per quanto un valore di defectiveness basso sia un fattore positivo per il progetto in sé, rende però più difficile il **train** del modello: infatti potendo attingere a pochi esempi di **positività** è più difficile per il modello capire con una certa accuratezza quali sono gli attributi e i rispettivi valori che concorrono alla buggyness della classe.

BookKeeper, data la percentuale decisamente più alta di defectiveness permette un train più accurato rispetto a OpenJpa.

Nella parte successiva viene presentato uno studio focalizzato sul ricercare la migliore combinazione delle tre variabili del modello di ML per i classificatori misurati: **Precision, Recall, AUC** (o ROC Area) e **Kappa**.

Il software utilizzato è **JMP** che ha fornito grande agilità nella creazione dei grafici.
Per motivi di semplicità si è scelto di effettuare le comparazioni a parità di metrica (Precision, Recall, ecc.) così da evidenziare con maggiore facilità la scelta migliore per il classificatore preso in esame.

I progetti vengono analizzati separatamente.

Di seguito riportati i grafici relativi al progetto **BOOKKEEPER**:

PRECISION

Figura 23 Precision del progetto "Bookkeeper"

E' immediato osservare che quasi per ogni possibile combinazione, la **precisione** presenta una forte **varianza**, coprendo la maggior parte dei valori possibili. L'unico fattore su cui è possibile fare comparazione sono i vari **quantili** di **primo, secondo** e **terzo ordine**.
La **combinazione migliore** sembra essere {**Random Forest**, **Undersampling, No selection**} che presenta il tradeoff ottimo tra una variazione e la mediana: la prima è tra le più basse rispetto alle altre e una mediana del 92.73% è tra le migliori.
Per **nessun classificatore** si riesce a stabilire con assoluta certezza quale combinazione di Balancing e Feature selection fornisca una precisione migliore.
Per **Naive Bayes** si potrebbe suggerire la combinazione **Oversampling** e **Best First**, che nonostante abbia una mediana del 81.63% presenta forse la più bassa varianza tra tutte le combinazioni esaminate per la precision.
Mentre per **IBK** si è avuto un comportamento molto **simile** a **Random Forest** ed infatti anche qui la combinazione migliore è risultata **Undersampling** e **No selection**: una mediana del 92,53% ed una varianza molto simile alla migliore combinazione di Random Forest rendono **{IBK,Undersampling,No selection}** la **seconda migliore** terna per il valore precision.

RECALL

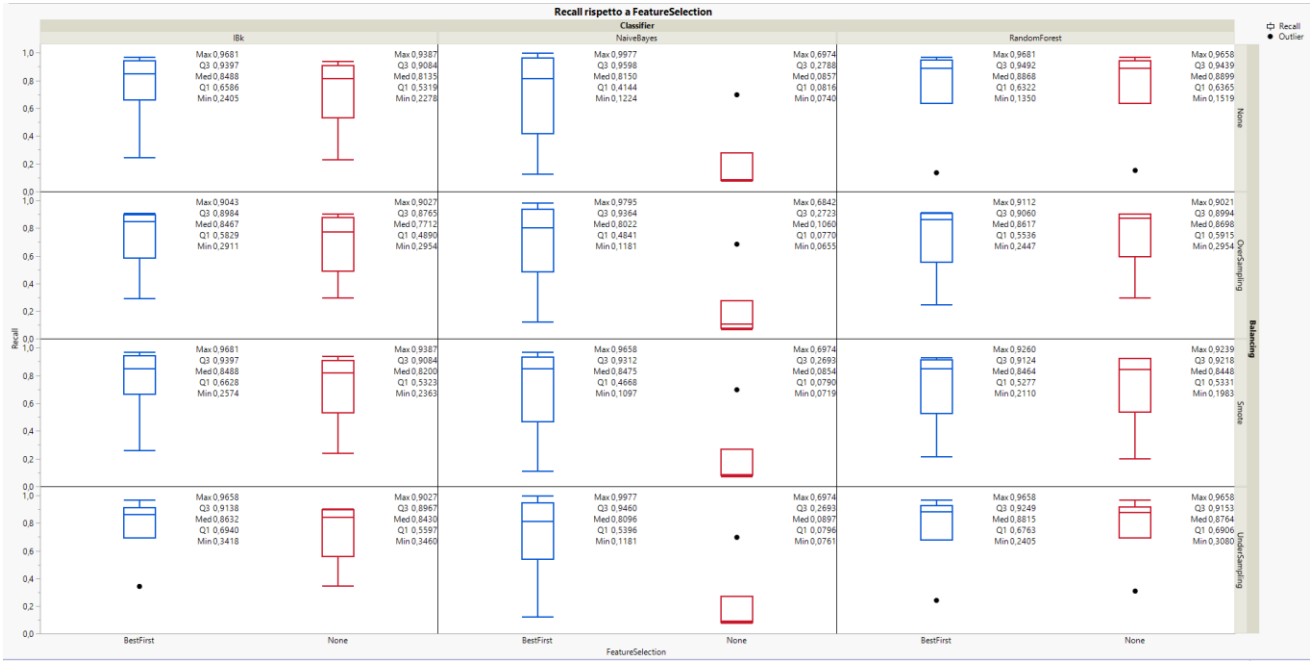

Figura 24 Recall del progetto "Bookkeeper"

La **recall** presenta rispetto alla *precision* cambiamenti più forti a seconda delle combinazioni.
La scelta che sembra essere migliore è quella di usare {**Random Forest, Undersampling, Best First**}, nonostante presenti un outlier dovuto alle fasi iniziali quando si avevano dati ridotti per l'addestramento, ha una variazione molto bassa ed una mediana del 88,15%.
Prestazioni simili ma ancora più facilmente riconoscibili si hanno per IBK la cui combinazione migliore risulta essere {**IBK, Undersampling, Best First**}, anche qui con la presenza di un outlier dato dagli addestramenti iniziali; la mediana è del 86,32%.
Per quanto riguarda **Naive Bayes** è molto difficile scegliere tra **Smote** ed **Undersampling** entrambi con feature selection **Best First,** in quanto il primo presenta mediana maggiore (84,75%) rispetto al secondo (80,96%) ma anche una varianza notevolmente più accentuata.
Si noti infine che gli **outlier** dei classificatori che hanno performato meglio (Random Forest ed IBK) sono relativi alle fasi iniziali di addestramento e quindi indicano un miglioramento nel tempo del classificatore, mentre gli outlier di Naive Bayes sono relativi agli ultimi addestramenti e quindi tendono ad indicare che quasi "casualmente" il solutore ha fatto bene in quello scenario e che quindi, come suggerisce il nome, Bayes è rimasto Naive (ingenuo) anche dopo l'apprendimento.

AUC

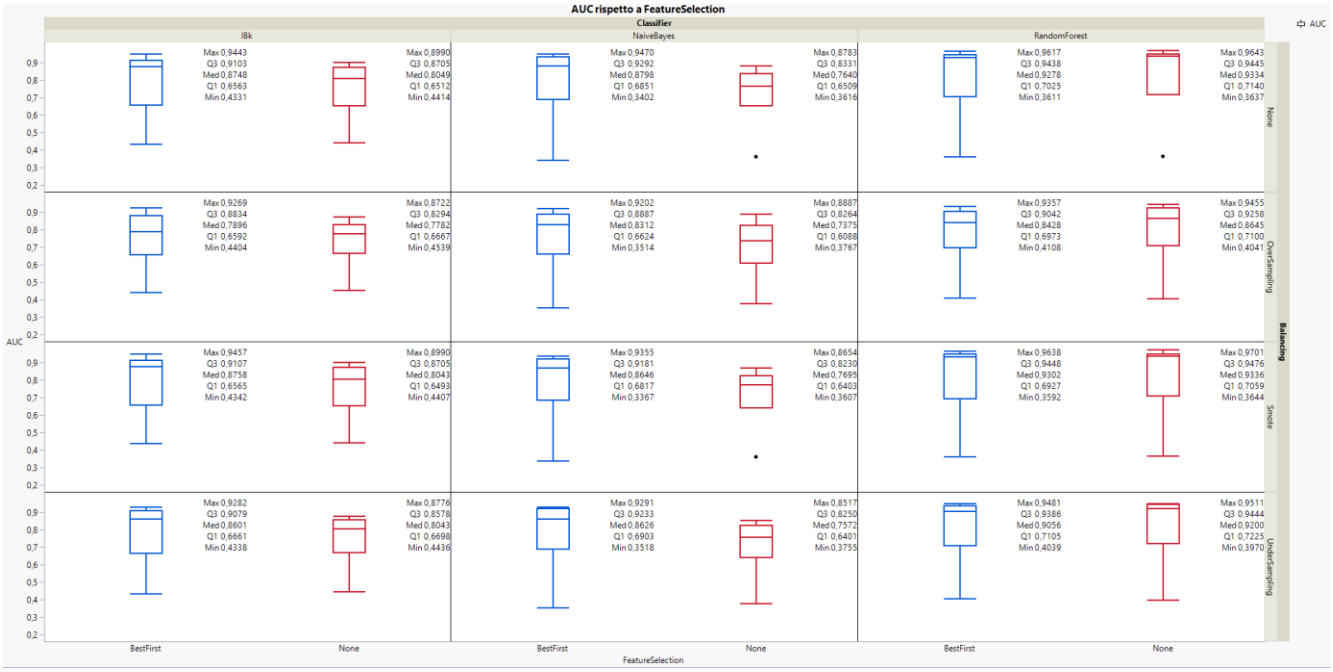

Figura 25 Auc del progetto "Bookkeeper"

Relativamente alla metrica **AUC** (area sotto la funzione ROC) è abbastanza immediato vedere come la combinazione di **{ Random forest** , **No sampling , No selection}** presenti risultati migliori e più stabili sia per la mediana (93,34%) che nella minore varianza possibile.
Per gli altri classificatori invece si hanno prestazioni molto simili delle varie combinazioni ma le migliori risultano essere : per **IBK** la migliore combinazione sembra essere **{ IBK** , **Smote , Best First}** con una mediana del 87,58% ; mentre per **Naive Bayes** la miglior soluzione risulta **{ Naive Bayes**, **No sampling , Best First}** con una mediana del 87,98%.
In questo scenario tutti gli outlier sono relativi alle fasi iniziali di addestramento e quindi tendono ad indicare un miglioramento generale delle prestazioni.

KAPPA

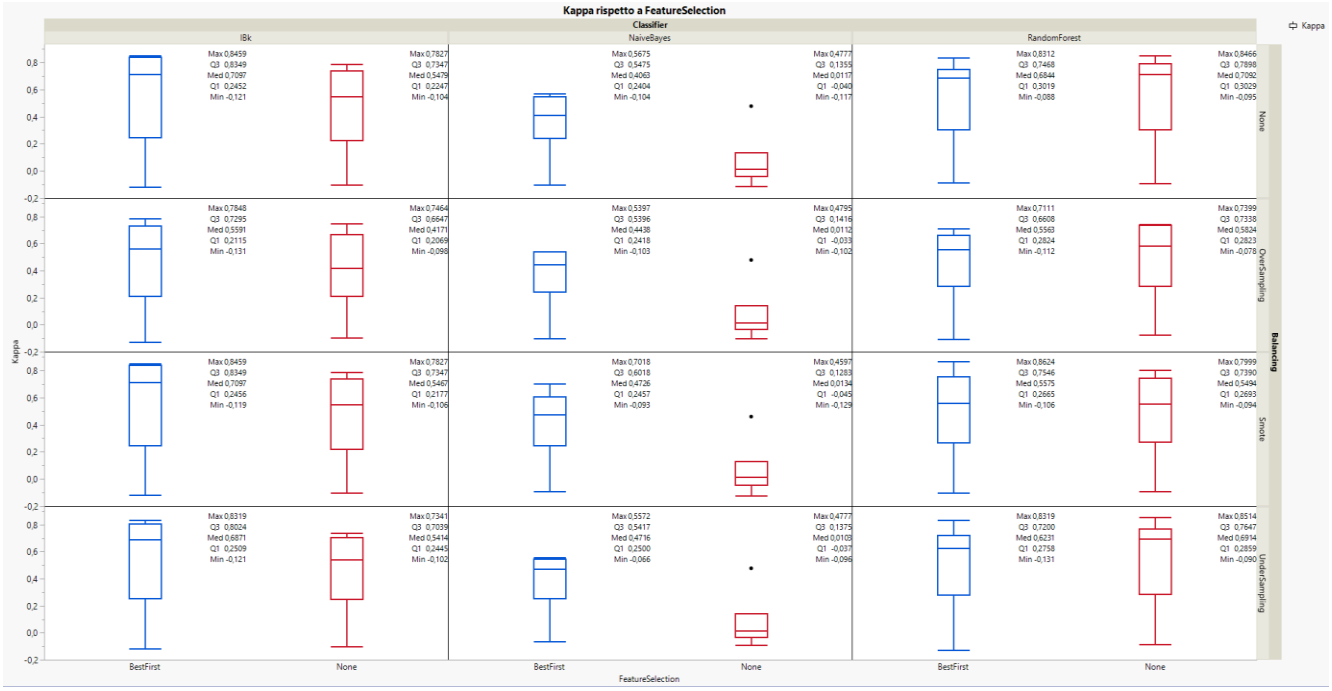

Figura 26 Kappa del progetto "Bookkeeper"

Come visibile dal grafico Naive Bayes non è stato in grado di performare bene se non in modo sporadico negli ultimi addestramenti (outliers), rimanendo comunque incapace di apprendere (non da risultati migliori di un classificatore banale che si limita a dividere a meta il dataset ed assegnare casualmente le label yes/no relative alla bugginess).

La **migliore prestazione** è stata ottenuto con **{IBK , Best First , No sampling}** con una mediana del 70,97%.

Le varianze risultano tutte molto elevate sia per IBK che per Random Forest.

La combinazione migliore per Random Forest è risultata essere invece **{Random Forest , No selection , No sampling}** con una mediana del 70,92%.

Per concludere utilizziamo il secondo grafico che da una visione d'insieme di tutte le metriche fino ad ora discusse:

Figura 27 Metriche totali del progetto "Bookkeeper"

Per concludere possiamo dire che :
-**{Random Forest , Undersampling, No selection}** sembra essere la scelta **migliore** per prestazioni relativamente al progetto Bookkeeper : la feature selection non ha giocato un ruolo chiave per questo solutore
-**{IBK, Undersampling , Best First}** ha anche performato molto bene, ma differentemente da Random Forest la feature selection qui è stata la chiave per ottenere il risultato migliore e ridurre la varianza delle metriche
-**Naive Bayes** è rimasto **naive** in questo scenario ed ha ottenuto risultati piu soddisfacenti in presenza di feature selection.

Di seguito viene presentata la **media tra tutti i walk** di **true/false positve/negative**:

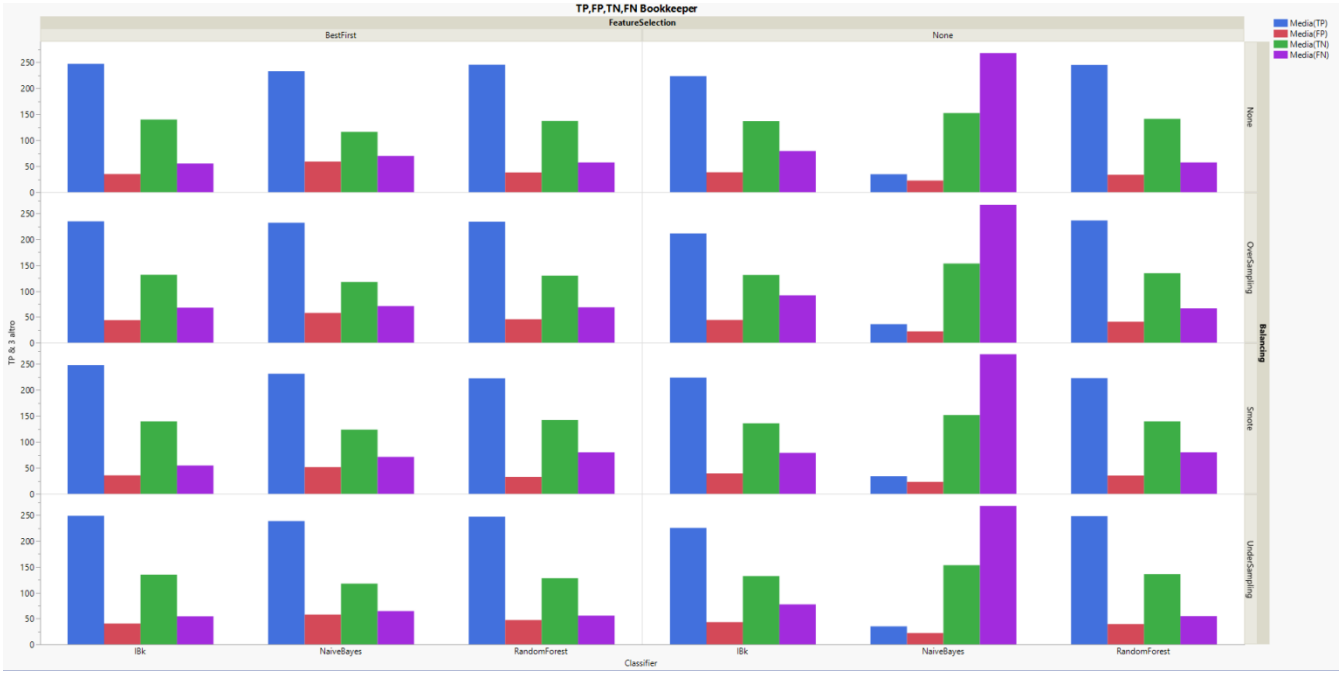

Figura 28 Overview delle performance dei classificatori in termini di TP,FP,TN,FN per il progetto "Bookkeeper"

L'obiettivo che ci si pone è quello di **massimizzare** l'output di **TP** e **TN** (ovvero classificare correttamente ciò che è defective come tale e saper riconoscere file non defective) e **minimizzare** il numero di predizioni errate , quindi **FP** e **FN**.

Come si puo vedere **Naive Bayes** privato della feature selection tende a classificare tutti i file come se non fossero difettosi portando a valori **elevati** dei **TN** per ovvi motivi e dei **FN** in quanto classifica come corretti anche tutti i file difettosi.

Per il resto i classificatori hanno andamenti simili sia in caso di feature selection che non ,ma come si puo vedre i **migliori rapporti** che tendono a massimizzare i TP e TN ed a minimizzare i FP e FN sembrano essere offerti per **Random Forest** da **No sampling** , **No selection** mentre per **IBk** da **Undersampling** ,**Best First.**

Passiamo ora all'analisi del progetto **OPENJPA**.

IN questo scenario si ha un discorso diametralmente opposto rispetto a Bookkeeper: infatti, sebbene abbiamo **molte più versioni**, e di conseguenza molti più walk (il che ci consente di avere un campione su cui effettuare l'analisi più ampio e quindi più significativo), il **rapporto** tra classi **defective e non,** come riportato nel grafico a inizio paragrafo, è **molto basso**.
Un rapporto così basso, mentre da un lato indica la qualità di OpenJpa, dall'altro rende meno accurato il modello di ML che opera sulle sue metriche: questo poiché la scarsa presenza di classi defective non permette di creare stime molto accurate su quali metriche e quali valori siano sintomo di buggyness.

PRECISION

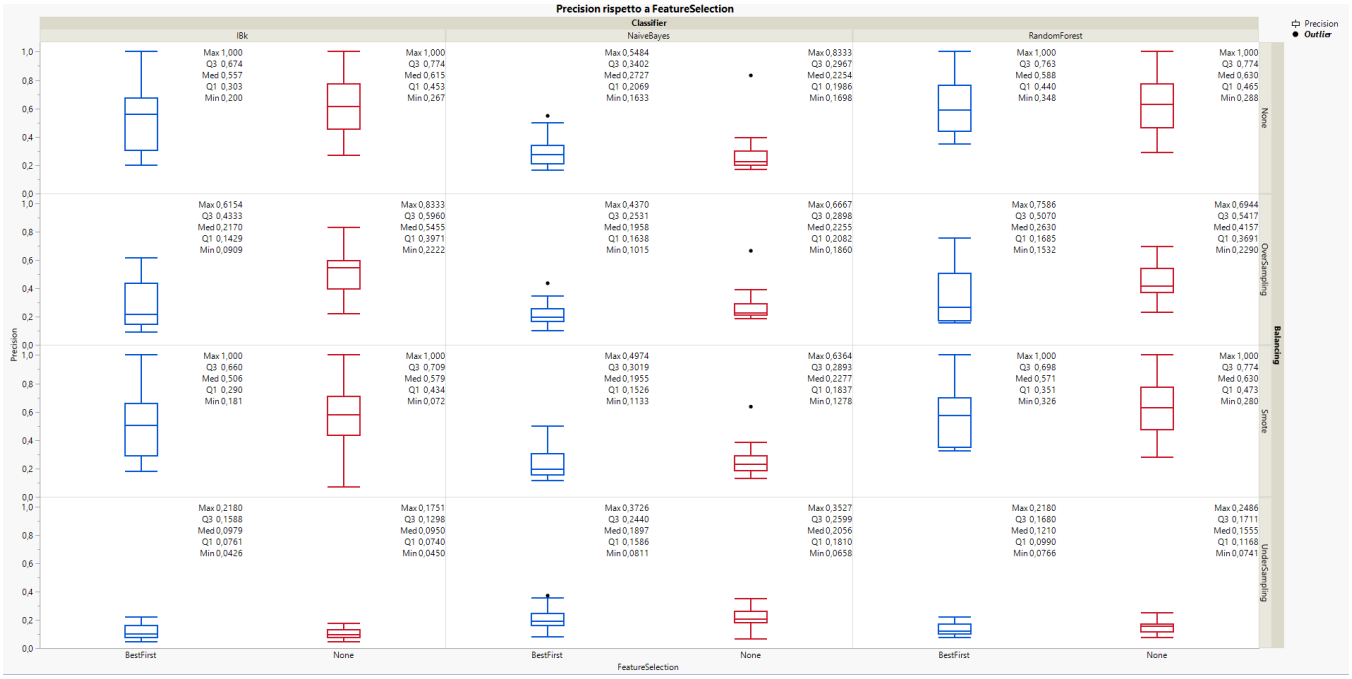

Figura 29 Precision del progetto "OpenJpa"

E' immediato osservare che tutti i classificatori presentano un'ampia variazione tra le performance.
Naive Bayes performa molto male in questo scenario: la combinazione che risulta essere lievemente migliore delle altre per questo classificatore è Best First, No balancing.
La **combinazione migliore** sembra essere {**Random Forest**, **Smote, No selection**} che presenta la mediana migliore tra tutte (63%) nonché una buona varianza.
Per **nessun classificatore** si riesce a stabilire con assoluta certezza quale combinazione di Balancing e Feature selection fornisca una precisione migliore.
Per **IBK** si è avuto un comportamento molto **simile** a **Random Forest** ma qui la combinazione migliore è risultata **No sampling** e **No selection** con una mediana del 61,5%: a confermare la **similitudine** nel comportamento **con Random Forest** è il dato che la seconda migliore combinazione per IBK risulta essere la stessa di Random Forest : **Smote, No selection** .
Per tutti i classificatori sembra che l'applicazione della feature selection conduca a risultati migliori in generale.

RECALL

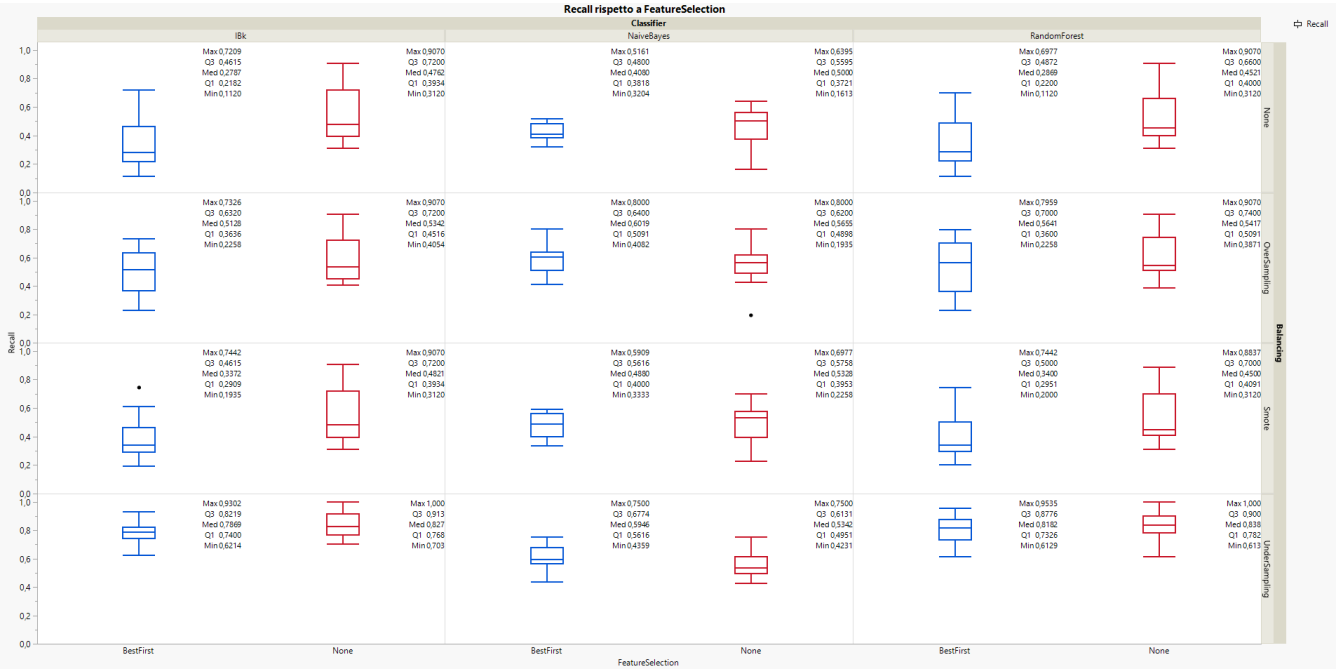

Figura 30 Recall del progetto "OpenJpa"

Nel caso della recall tutte le prestazioni migliori dei solutori si hanno avute in presenza di **Undersampling.**

La **combinazione migliore** risulta essere **{Random Forest, Undersampling , No selection}** con una mediana del 83,38%.

Per IBK si ha avuto un comportamento molto simile a Random Forest ed anche qui le migliori performance si sono registrate in presenza di **Undersampling , No selection** con una mediana del 82,7%.

Naive Bayes ha **performato** molto **male** rispetto agli altri due solutori, ma la miglior combinazione si ha avuta in presenza di **Undersampling , Best First** con una mediana del 59,46%.

AUC

Figura 31 AUC del progetto "OpenJpa"

Anche per questa metrica la **combinazione migliore** è stata ottenuta da **Random Forest** con **No sampling , No selection** con una mediana del 92,45%.

Per **IBK** si ha avuto un comportamento molto **simile** a **Random Forest** quando si applica **Best FIrst** ma molto **peggiore** in caso di **No selection** .Per questo classificatore la combinazione migliore si è avuta in presenza di **No sampling , Best first** con una mediana del 84,45%.

Naive Bayes ha **performato** molto **meglio** rispetto alle altre metriche e similmente a Random Forest sia icon che senza feature selection e la miglior combinazione si ha avuta in presenza di **Oversampling , No selection** con una mediana del 90,23%.

KAPPA

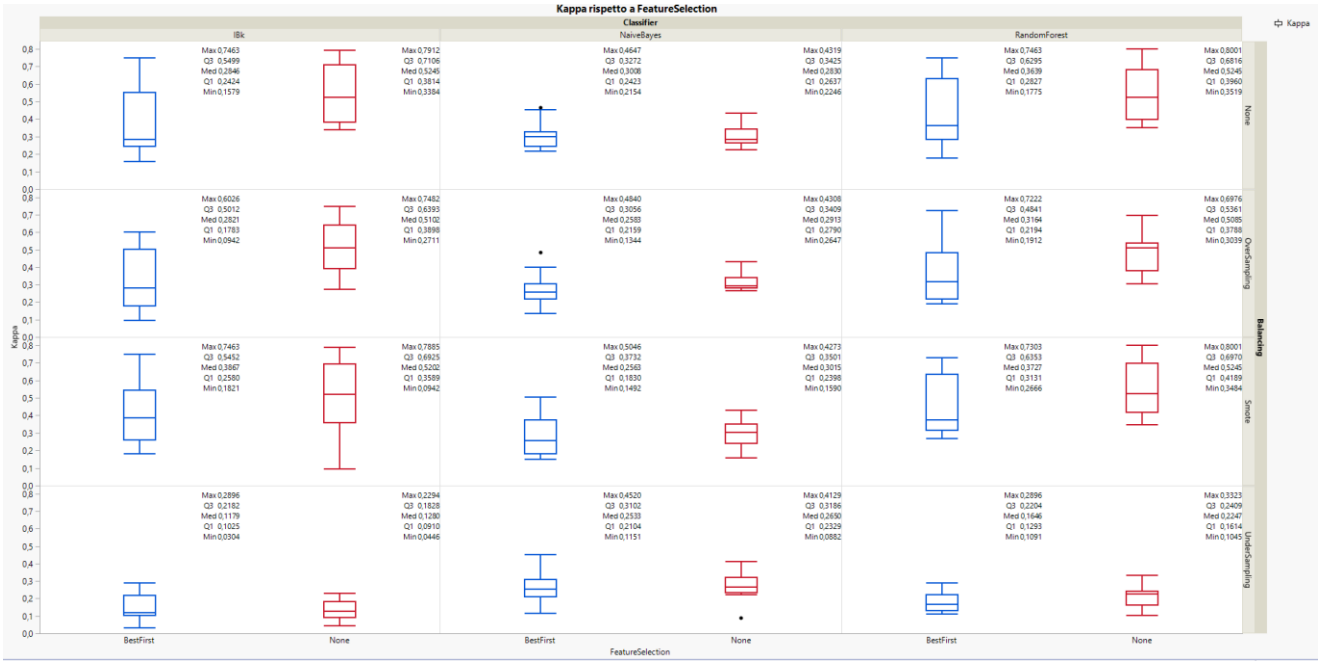

Figura 32 Kappa del progetto "OpenJpa"

I valori di kappa presentati nel grafico ci mostrano che i classificatori che **meglio** sono riusciti ad apprendere dal dataset sono stati **IBK** in combinazione con **No balancing** e **No feature selection** con una mediana del 52,45% e **Random Forest** con **Smote e No feature selection** con una mediana del 52,45%, con valori motlo simili tra loro .

Il classificatore che è risultato piu **dummy** è stato **Naive Bayes** la cui miglior prestazione è stata registrata con **Smote** e **No feature selection** con una mediana del 30,15%.

Per concludere utilizziamo il secondo grafico che da una visione d'insieme di tutte le metriche fino ad ora discusse:

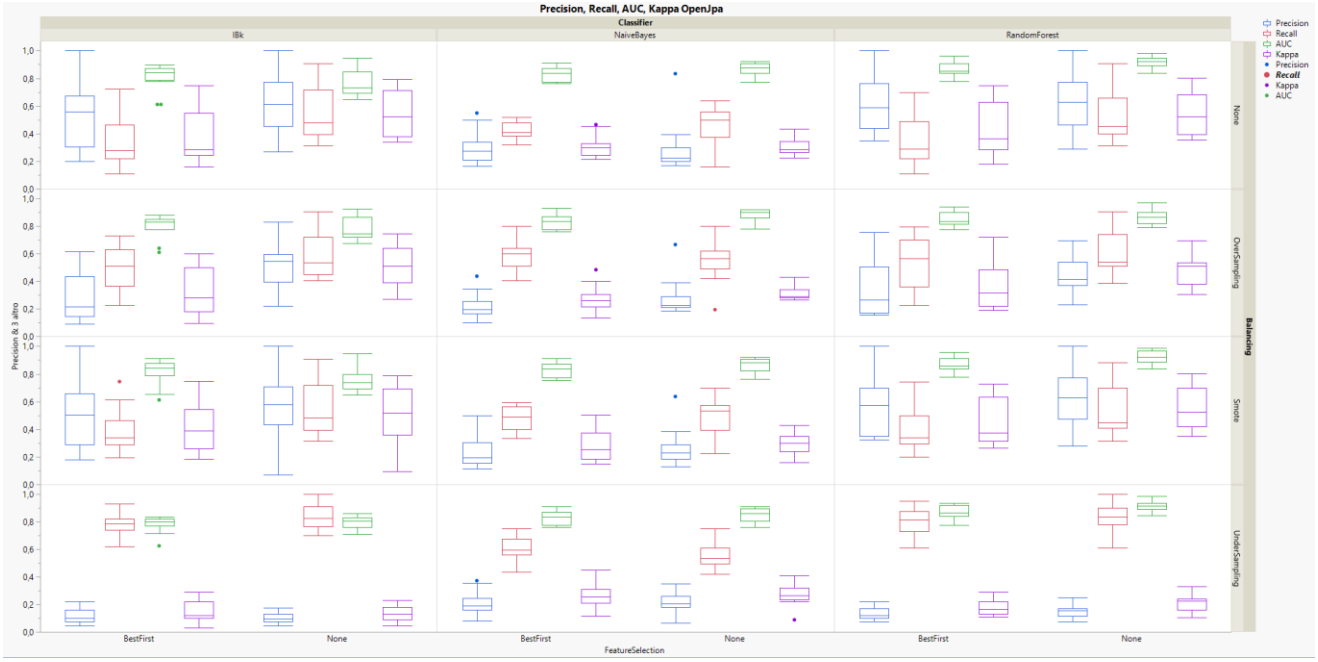

Figura 33 Metriche totali del progetto "OpenJpa"

Riassumendo possiamo dire che :
-**{Random Forest , Smote, No selection}** sembra essere la **miglior combinazione** che si possa ottenere per il progetto OpenJpa ma anche **{IBK, Smote , No selection }** ha performato molto bene ed in molto simile pichè sono le combinazioni che sono riuscite ad apprendere maggiormente dal dataset.
-**Naive Bayes** è rimasto **naive** anche in questo scenario ed inoltre non sembra che la feature selection abbia sortito grandi miglioramenti nelle prestazioni di questo classificatore.

Di seguito viene presentata la **media tra tutti i walk** di **true/false positve/negative**:

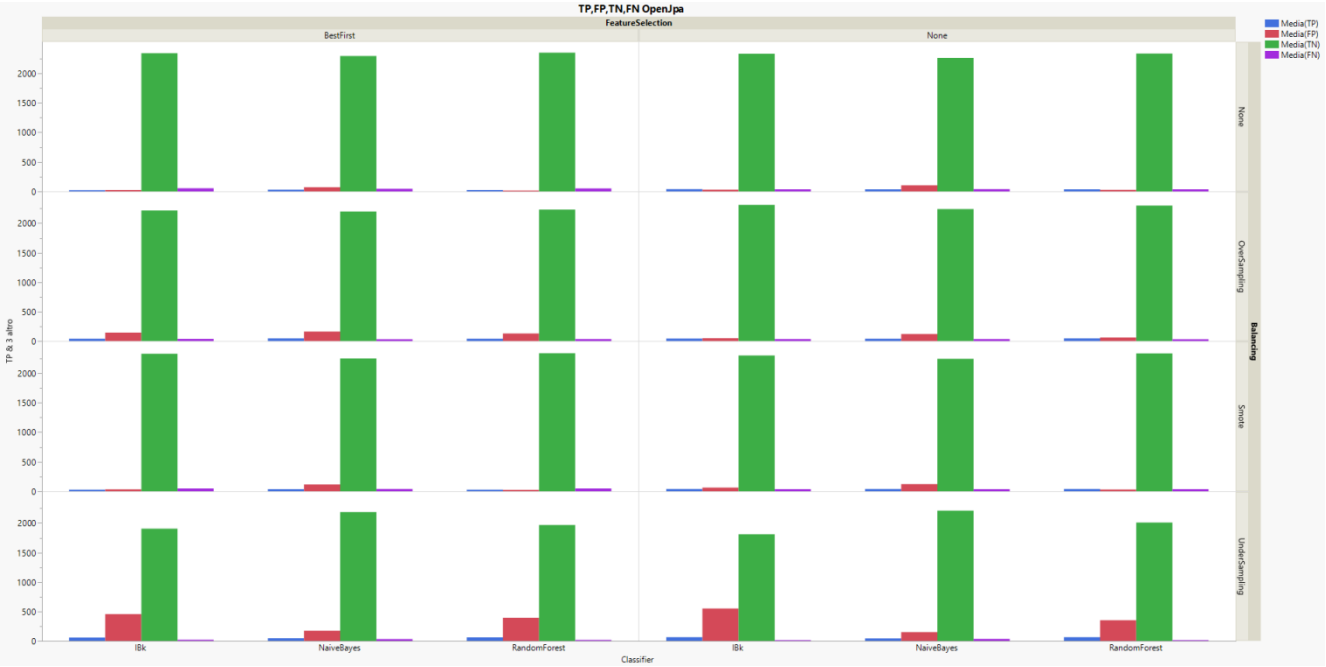

Figura 34 Overview delle performance dei classificatori in termini di TP,FP,TN,FN per il progetto "OpenJpa"

Il grafico conferma quanto anticipato all'inizio della trattazione: la **presenza di classi buggy** è **molto bassa** e ciò ha portata ad una maggiore difficoltà nell'addestramento dei classificatori.

Possiamo notare che **Undersampling** ha avuto le **performance peggiori** poiché tende a classificare le classi non difettose come tali e lo stesso vale per **Oversampling** anche se in modo meno accentuato.

Le **migliori prestazioni** ancora una volta si sono avute con le combinazioni di **{Random Forest, Smooth , No selection }**.