DELIVERABLE 2: PREDIRE LA DIFETTOSITÀ DI CODICE SOFTWARE

Martina De Maio

Studentessa Laurea Magistrale in Ingegneria Informatica,

Università degli Studi di Roma “Tor Vergata”, Roma, Italia

Matricola: 0296447

Agenda

Il lavoro è organizzato secondo la seguente scaletta:

1. Introduzione
2. Ciclo di vita di un difetto
3. Metodologia
4. Risultati
5. Conclusione
6. Links

Introduzione

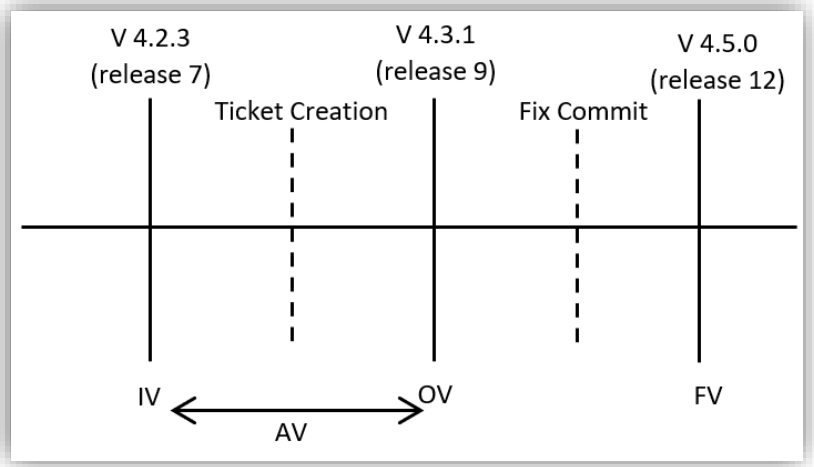
- Il modo in cui i difetti vengono introdotti nel codice e l'enorme volume di difetti nel software sono in genere al di là delle capacità e delle risorse della maggior parte dei team di sviluppo. A causa di questo problema, i ricercatori hanno esplorato approcci di machine learning per prevedere
  - se è probabile che si verifichi un difetto in un modulo software
  - quanti difetti post- deployment possono verificarsi.
- La previsione dei difetti consente di identificare gli artefatti software che potrebbero avere un'alta difettosità e, di conseguenza, agli sviluppatori di concentrarsi su artefatti specifici. Ciò consente di ridurre il costo necessario al testing del codice, concentrando le risorse per il testing su alcune classi piuttosto che su altre.
- La seguente presentazione ha lo scopo di fornire una descrizione del processo di analisi e studio effettuato per il Deliverable 2, il cui obiettivo è quello di eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di ***sampling***, ***feature selection*** e ***cost sensitive*** , offerte dall'API di ***Weka***, sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni open-source. In particolare, i progetti analizzati sono Apache ***BookKeeper*** e Apache ***Syncope***.
- Nel corso del lavoro saranno identificate le classi di un progetto più predisposte alla difettosità e, successivamente, utilizzati modelli di ***Machine Learning*** per predire su quali file sia più probabile l'insorgenza di ***bug*** nel codice sorgente, sulla base del valore di metriche calcolate sui file.

Introduzione

- L'obiettivo finale di tale studio è quello di identificare quali tecniche di feature selection, balancing o sensitivity aumentano l'accuratezza dei classificatori, e per quali classificatori e quali dataset.
- Le varie tecniche utilizzate nella fase di predizione sono:
  - ***Walk forward*** come tecnica di valutazione.
  - ***No selection*** / ***best first*** come feature selection.
  - ***No sampling*** / ***oversampling*** / ***undersampling*** / ***SMOTE*** come balancing
  - ***No cost sensitive*** / ***Sensitive Threshold*** / ***Sensitive Learning*** (CFN = 10 * CFP)
  - ***RandomForest*** / ***NaiveBayes*** / ***Ibk*** come classificatori.
- Il lavoro si concentra sui metodi automatizzati per la creazione di un dataset per predire quali classi sono affette da bug e in quali release per ogni progetto analizzato.
- Per identificare le origini di un difetto è stato fatto uso delle ***Affected Versions*** (AV) disponibili nei reports sui difetti forniti dal sistema di issue trackers ***JIRA***.
- Nelle slide successive verranno introdotte le nozioni di base che verranno poi implementate nelle varie fasi progettuali.

Ciclo di vita di un difetto

- Per etichettare le classi come difettose o meno si è sfruttato il ciclo di vita di un difetto.
- La figura illustra i termini chiave utilizzando il difetto «BOOKKEEPER-852» come esempio. Il difetto viene prima iniettato nel codice alla ***Injected Version*** (***IV***), ovvero la versione V4.2.3.
- Successivamente, viene osservata una failure e viene creato un ***defect report*** per descrivere il difetto. La versione relativa alla creazione del defect report è la ***Opening Version***(***OV***), ovvero la versione V4.3.1.



- In seguito, in una data versione futura, il difetto viene corretto mediante modifiche eseguite in una o più classi. La versione correlata alla correzione del difetto è la ***Fixed Version*** (***FV***), ovvero la versione V4.5.0.
- Un ***Affected Version*** (***AV***) è qualsiasi versione affetta dal difetto. Pertanto, le AV nel nostro esempio sono quelle nell'intervallo [IV, FV), ovvero le versioni V4.2.3 e V4.3.1. La versione V4.5.0 non è un AV poiché contiene la correzione e quindi non è affetta dal difetto.

Ciclo di vita di un difetto

- L'OV è disponibile in tutti i report relativi ai difetti su JIRA, poiché è generato dall'issue tracker al momento della creazione del report.

- L'FV invece è la release successiva alla data dell'ultimo commit associato al difetto, ossia il commit che risolve il difetto e che contiene l'ID del bug nel suo log.

- Le AV invece non sono sempre disponibili, e, se lo sono, non è detto che esse siano affidabili. La lista delle AV è coerente quando il primo AV si verifica prima dell'OV. La logica infatti è che il difetto deve aver interessato una release che si è verificata almeno nel momento in cui è stato creato il report sul difetto. Cioè, un difetto non può essere stato iniettato nel codice dopo aver osservato la relativa failure.

- Per quei difetti per cui non si è riuscito a recuperare la lista delle AV, si introduce un metodo per recuperare tali AV. Poiché le AV sono nell'intervallo [IV, FV), e poiché conosciamo sempre FV, il recupero delle AV di un bug si traduce in realtà nella ***stima del suo IV***.

Stimare IV tramite Proportion

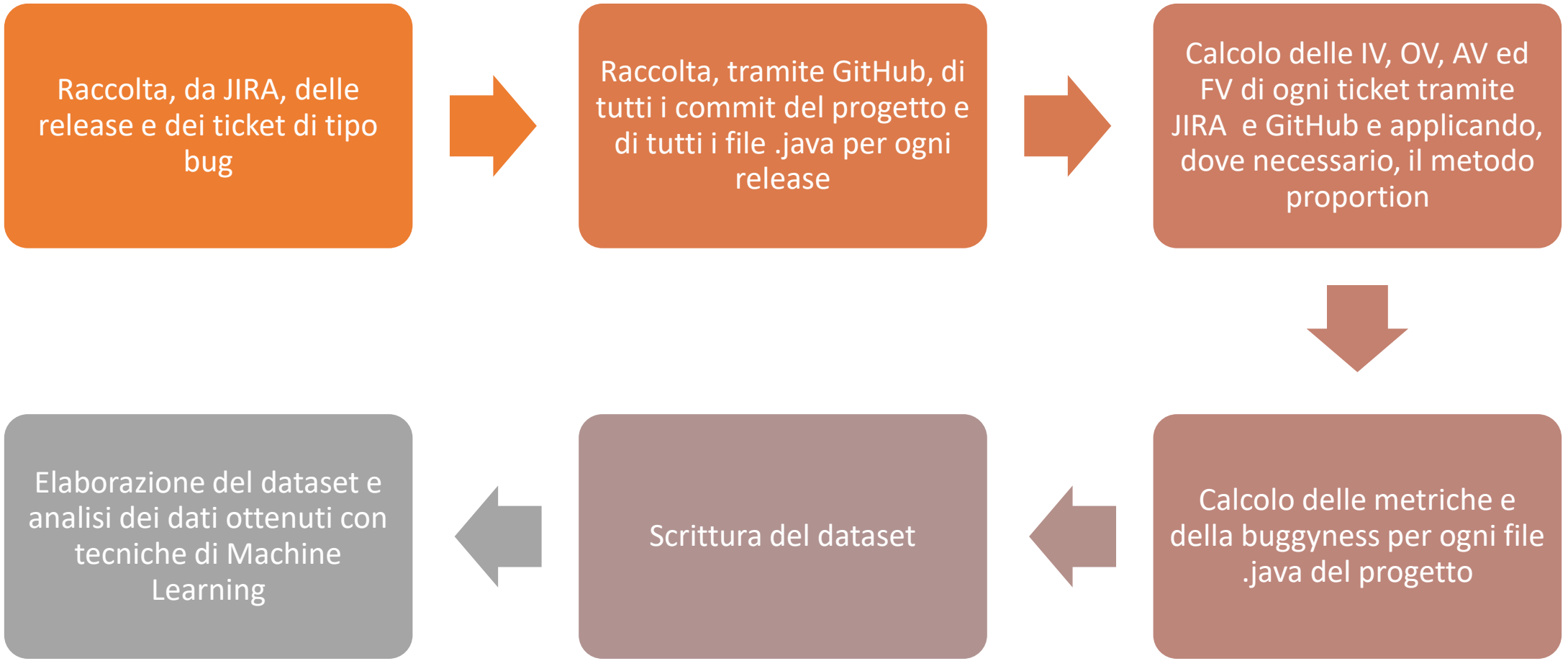
- Per stimare l'IV di un difetto si è utilizzato il metodo ***Proportion***. Tale metodo assume una proporzione ***P*** stabile, tra i difetti dello stesso progetto, tra il numero di Affected Versions tra IV e FV ed il numero di versioni tra OV e FV.
- La logica è che difetti di uno stesso progetto potrebbero avere un ciclo di vita stabile e coerente tra loro. I difetti richiedono un certo numero di release per essere individuati e un altro numero per essere risolti. La nostra intuizione è che la proporzione tra questi numeri sia in qualche modo stabile tra i difetti dello stesso progetto.
- Essendo FV e OV noti per ogni difetto, l'idea è di calcolare P sui difetti precedenti e di utilizzarlo per il calcolo delle IV per quei difetti i cui le AV non sono disponibili o non sono coerenti.

Si definisce P come : $P = (FV - IV)/(FV - OV)$.

E si calcola IV come : $IV = FV - (FV - OV) * P$

- Tra le varie configurazioni esistenti per il calcolo di *P* si è scelta quella del ***moving window***, che consiste nel calcolare *P* come la media dell'1% dei difetti risolti precedenti al ticket in questione.

Workflow fasi progettuali



Fase 1: Raccolta dei dati da JIRA

- Per ottenere le release del progetto si utilizza l'API di JIRA. In particolare, si effettua una query tramite l'url :

  https://issues.apache.org/jira/rest/api/2/project/projName

  la quale restituisce un file ***.json*** che viene analizzato e da cui vengono raccolte le informazioni riguardo l'ID della release, il nome, e la data. Una volta raccolte, ad ogni release viene associato un indice, il quale rende il lavoro successivo per proportion più semplice, e vengono ordinate cronologicamente.

- L'ultimo 50 % delle release viene inoltre rimosso, in quanto si è dimostrato che in questo modo il Missing rate si abbassa al 10%, portando a risultati più accurati nella predizione. Infatti, le release più recenti si sono dimostrate avere un numero di classi difettose minore rispetto a quelle più datate, in quanto è più probabile che un bug non sia ancora stato scoperto.

- Tramite JIRA, sistema di tracking per progetti software che consente di visionare lo storico dei bug che sono stati risolti o che devono ancora essere risolti, si possono raccogliere anche i ticket del progetto attraverso la query :

  *https://issues.apache.org/jira/rest/api/2/search?jql=project%20%3D%20**projName**%20AND%20issue type%20%3D%20**Bug**%20AND%20status%20in%20(**Resolved**%2C%20**Closed**)%20AND%20resolution %20%3D%20**Fixed**%20ORDER%20BY%20updated%20DESC&fields=key*

  che recupera tutti i ticket con issueType = " *Bug*", status = "*closed*" o "*resolved*", resolution = " *fixed* ".

Fase 1: Raccolta dei dati da JIRA

Questa query produce un JSONObject, il quale viene analizzato e parsato per ricavare le informazioni necessarie, come indicato a lato, a titolo d'esempio, per il ticket "BOOKKEEPER-1105".

Per ogni ticket viene quindi prelevato l'***ID***, le ***Affected Versions*** se presenti e la ***creation date,*** presi rispettivamente tramite i campi "key", "versions" e "created" del JSONObject.

- L'ID dei ticket viene utilizzato per trovare quei commit che sono stati effettuati per risolvere il bug citato nel ticket, ossia quelli aventi nel messaggio proprio l'ID del ticket.
- La ***data di creazione*** del ticket viene usata per computare ***l'Opening Version***, valore necessario per l'applicazione del Proportion.
- Non sempre le Affected Versions sono riportate, e, se riportate, non sempre sono consistenti. Comunque, per i ticket con AV riportate si è settata come IV la prima release appartenente alle Affected Versions.


```
{"expand":"operations,versionedRepresentations,editmeta,changelog,renderedFields",

"self":"https://issues.apache.org/jira/rest/api/2/issue/13093560",

"id":"13093560",

"fields":{
        "versions":[
                {"archived":false,
                "releaseDate":"2017-08-10",
                "name":"4.5.0",
                "self":"https://issues.apache.org/jira/rest/api/2/version/12335340",
                "description":"Release 4.5.0",
                "id":"12335340",
                "released":true}],

        "resolutiondate":"2017-08-16T01:42:23.000+0000",
        "created":"2017-08-09T18:25:48.000+0000"},
        "key":"BOOKKEEPER-1105"}
```

Fase 2: Raccolta dei commit da GitHub

- Dopo aver effettuato il clone della repository, tramite l'API di ***JGit*** viene recuperato il log dei commit forniti da Github.

- Scorrendo tale log, sono stati recuperati tutti i file .java per ogni release presenti nella repository del progetto, ripercorrendo il ***tree*** di ogni commit, che contiene tutti i file presenti nel progetto nel momento in cui è stato effettuato il commit. Ad ogni release viene quindi assegnata una lista di files .java.

- I file java, nel corso del ciclo di vita del progetto, possono essere stati ***rinominati*** o cambiati di path, sia tra una release e l'altra, sia nella stessa release. Analizzando i singoli commit in ordine cronologico, è possibile trovare quali classi sono state rinominate e assegnare a ogni file una lista di file *alias* contenenti le varie rinominazioni della classe stessa. Per gestire i rename si sono analizzate le ***diffEntry*** per ogni commit e si è cercato il caso in cui l'operazione sul file fosse una RENAME.

Fase 3: Calcolo delle IV, OV, AV ed FV di ogni ticket e applicazione di Proportion

- Una volta raccolti tutti i commit, si associa ad ogni ticket una lista di commit aventi nel log l'ID del tale ticket, e si setta come Fixed Version FV la release successiva alla data dell'ultimo commit trovato, poiché vuol dire che quello è stato l'ultimo commit effettuato per risolvere il bug, e quindi si presume che nella release successiva a tale commit il bug non sia più presente in quanto è stato risolto.
- Eventuali ticket senza nessun commit associato sono stati eliminati.
- A questo punto, ogni ticket ha:
  - OV presa da JIRA
  - FV presa da GitHub
  - Eventualmente AV prese da JIRA, e quindi IV
- Come introdotto all'inizio della presentazione, nel caso in cui le AV non siano presenti oppure esse non siano consistenti, bisogna stimare il valore delle IV di ogni ticket tramite il metodo ***Proportion***.
  1. Per prima cosa si controllano le IV e AV prese da JIRA. Per i ticket aventi FV = 1 oppure OV = 1, si setta la rispettiva IV = 1 (per definizione IV <= OV).
  2. Ora, se IV != 0
     a. Se FV >IV && OV >= IV → si setta AV come [IV,FV),
     b. FV >= IV && OV < IV oppure FV < IV → i dati relativi alle IV, e quindi alle AV, sono sbagliati, quindi setto IV = 0 e svuoto la lista di AV, per poi andare a ricalcolare IV con Proportion
     c. FV = IV → vuol dire che non ci sono AV per quel ticket, poiché il ticket è stato introdotto e fixato nella stessa release.

Fase 3: Calcolo delle IV, OV, AV ed FV di ogni ticket e applicazione di Proportion

Si applica il meccanismo di Proportion a quei ticket che hanno IV = 0, per i quali non si è riusciti a ricavare una Injected Version.

Per l'applicazione di Proportion si è scelto di implementare due diversi metodi:

Metodo 1 : IV=FV se FV = OV

1. Si effettua un check iniziale: essendo Predicted IV = FV - (FV -OV) * P , se sono presenti ticket aventi FV =OV viene predicted IV = FV . Per questi ticket si setta quindi IV = FV. Tali ticket non verranno presi in considerazione nel calcolo di Proportion, ossia non vengono usati per il calcolo di P.
2. Usando il meccanismo di ***moving window***, per ogni ticket avente IV = 0 si calcola la P dell'1% dei difetti precedenti tramite la formula P = (FV -IV) / (FV -OV) e si stima la IV utilizzando la media, arrotondata in eccesso, tra le P trovate.

Metodo 2 : se FV = OV, ticket cancellato

1. Se sono presenti ticket aventi FV = OV, essi vengono rimossi dalla lista di ticket.
2. Si procede come sopra

Una volta ottenute tutte le IV, si settano le rispettive AV, appartenenti all'intervallo [IV,FV).

Fasi 4-5: Computazione di metriche e scrittura del dataset

Una volta applicato l'algoritmo Proportion per ogni ticket, si è calcolata la buggyness di ogni classe in tutte le release. Nel dataset finale la buggyness è stata indicata con il valore "Yes"/"No".

Tramite il meccanismo delle ***diffEntry***, ossia i cambiamenti che avvengono in un file tra un commit e quello precedente, per ogni file vengono calcolate le seguenti metriche:

Size	Numero di linee di codice LOC
LOC_added	Numero di LOC aggiunte nelle varie revisioni della release
MAX_LOC_Added	Valore massimo tra le LOC_added
AVG_LOC_Added	Valore medio tra le LOC_added
Churn	Somma, attraverso le revisioni, della differenza tra LOC aggiunte e cancellate
MAX_Churn	Valore massimo tra i churn

AVG_Churn	Valore medio tra i churn
NR	Numero di revisioni in cui la classe compare
NAUTH	Numero di autori
ChgSetSize	Numero di file committati insieme alla classe di interesse
MAX_ChgSet	Valore massimo tra i ChgSetSize
AVG_ChgSet	Valore medio tra i ChgSetSize

L'output finale è un file ***.csv*** contenente per ogni file di ogni release le metriche calcolate e la buggyness.

I dataset per entrambi i progetti si trovano nella cartella ***"\output\first part"***. Di seguito un estratto del dataset per BookKeeper:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	RELEASE	FILENAME	LOC	LOC_added	MAX_LOC_Added	AVG_LOC_Added	Churn	MAX_Churn	AVG_Churn	NR	NAUTH	ChgSetSize	MAX_ChgSet	AVG_ChgSet	BUGGYNESS
2	1	bookkeeper-benchmark/src/main/java/org/apache/b	124	235	137	58.75	137	137	34.25	4	2	475	207	118.75	No
3	1	bookkeeper-benchmark/src/main/java/org/apache/b	243	340	252	68	244	252	48.8	5	2	476	207	95.2	No

Fase 6: Machine Learning

- In questa fase viene utilizzato come input il dataset estrapolato nella fase precedente, per valutare la prestazione dei classificatori ***Random Forest***, ***Naive Bayes*** e ***Ibk***, per trovare se e quali tecniche di feature selection, balancing o sensitivity aumentano l'accuratezza di un determinato tipo di classificatore.
- A questo scopo è stata provata ogni possibile combinazione di *{classifier, balancing tecnique, feature selection, sensitivity}* ottenendo un totale di 3x2x4x3 = ***72 combinazioni***.
- La tecnica di valutazione utilizzata è quella del ***Walk Forward***, scelta necessaria per preservare l'ordine dei dati, avendo le istanze dei dati un legame temporale tra loro. Il dataset è stato diviso in porzioni a seconda del numero di release: le singole unità che compongono il walk forward, rappresentate dalla classe *Walk.java*, contengono la porzione del dataset relative a tutti i file contenuti in una specifica release.
- Per stimare la difettosità delle classi appartenenti ad una certa release, si utilizzano le ***release precedenti*** ad essa come ***training*** e la predizione viene effettuata sulla release corrente, utilizzata come ***testing***.

I dataset si trovano nella cartella ***"\output\second part".***

Risultati

La percentuale di classi buggy finale per i due progetti analizzati e per i due metodi di Proportion applicati è mostrata nella tabella a lato.

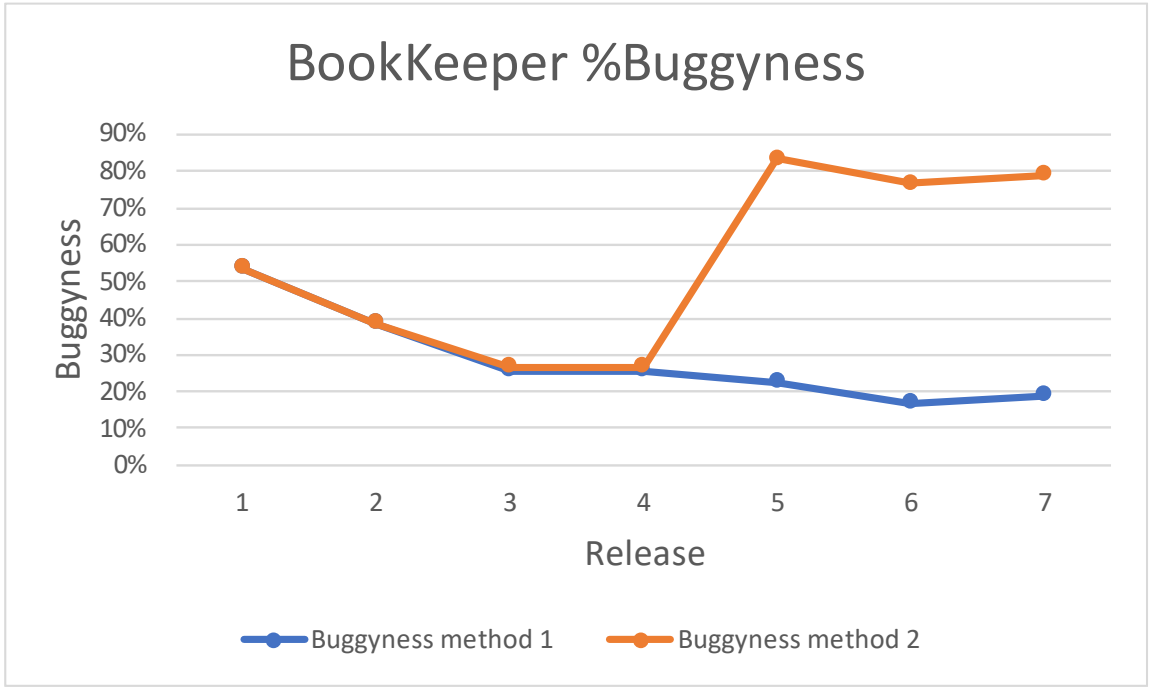
Come si può notare, per BookKeeper, con il metodo 1 la percentuale di classi buggy totale è nettamente inferiore rispetto all'applicazione del secondo metodo. Questo perché si è settato IV=FV per quei ticket aventi FV=OV, con una conseguente AV nulla. Infatti, essendo AV l'insieme delle release comprese tra [IV,FV), se IV=FV, non avrò AV per questi ticket, e quindi ci saranno più ticket aventi una lista di AV nulla, con conseguente diminuzione di classi difettose.

Per Syncope, invece, si può notare come la scelta di uno dei due metodi porta a risultati simili.

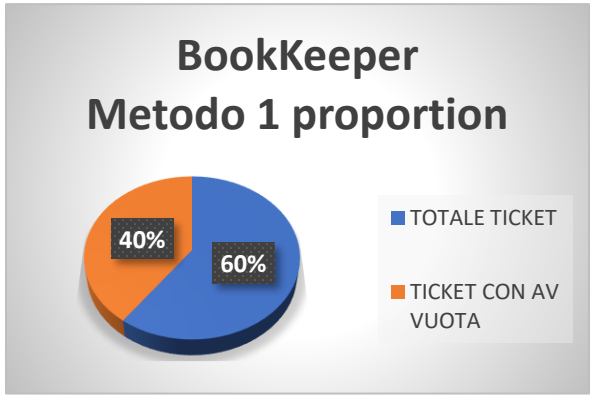
Proportion	BookKeeper	Syncope
Metodo 1	25%	42%
Metodo 2	59%	41%

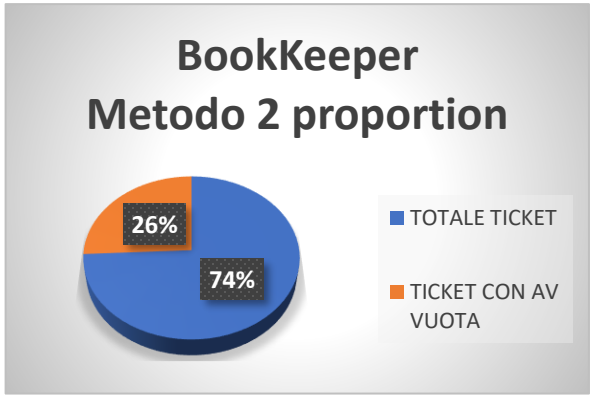
Risultati

- Nel grafico è stata riportata la percentuale delle classi difettose per ogni release per il progetto BookKeper analizzato, per entrambi i metodi di Proportion implementati.
- Si può notare una netta differenza tra i due metodi dalla release 4 in poi, in quanto le release release 5,6,7 sono risultate poco affette da bug utilizzando il metodo 1. Questo comportamento può essere giustificato dal fatto che, utilizzando il primo metodo, per le release 5,6,7, settando IV=FV si sono perse informazioni importanti, dato che tali release non erano quasi mai presenti nelle AV dei ticket. In basso si può osservare, per BookKeeper, la percentuale dei ticket non aventi AV rispetto al totale per i due metodi di proportion implementati.
- Invece, per il metodo 2, si ha un andamento molto più **instabile** e **crescente nel tempo**, anche se comunque lo scarso numero di release non rende possibile un'analisi più completa sull'andamento nel lungo termine.



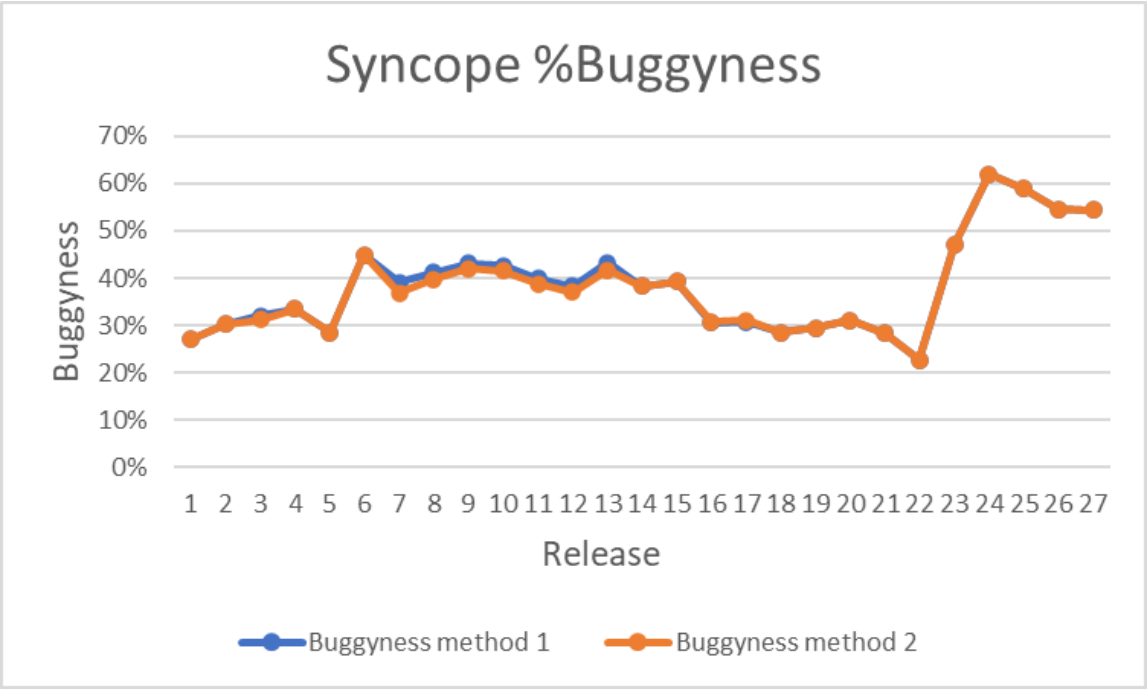
Proportion	TOTALE TICKET	TICKET CON AV VUOTA
Metodo 1	420	279
Metodo 2	216	75



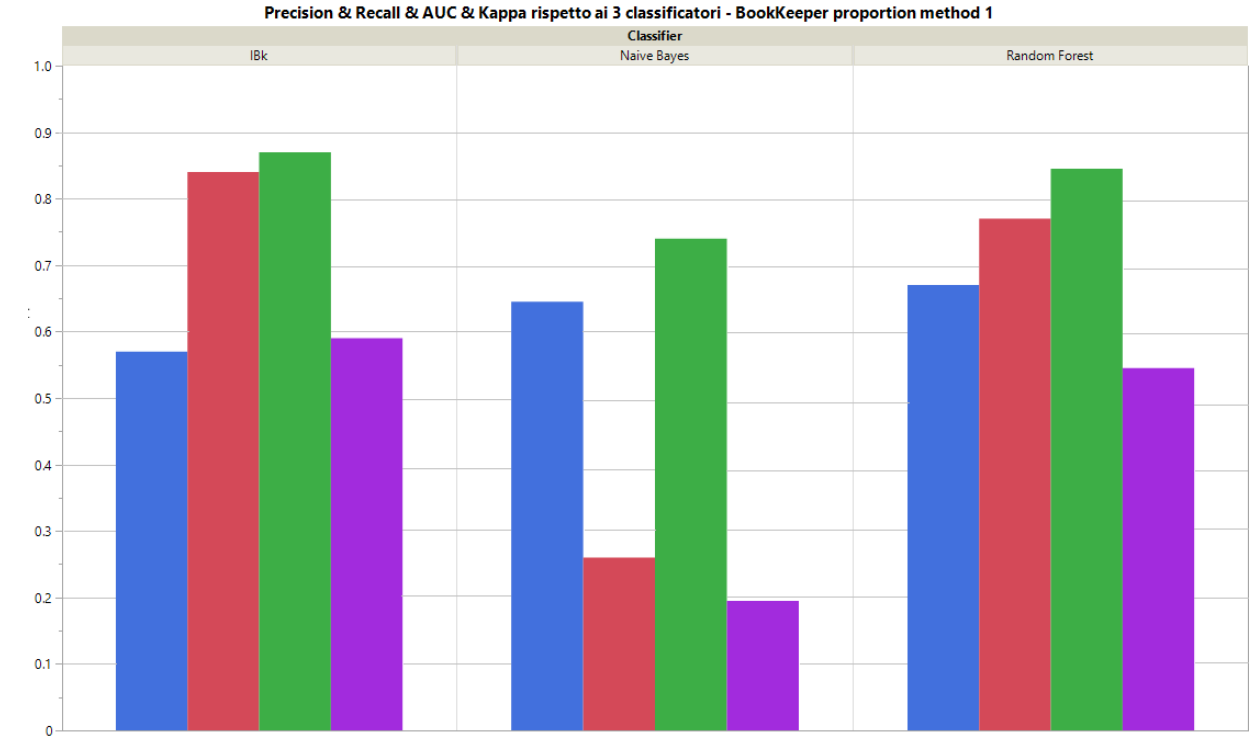


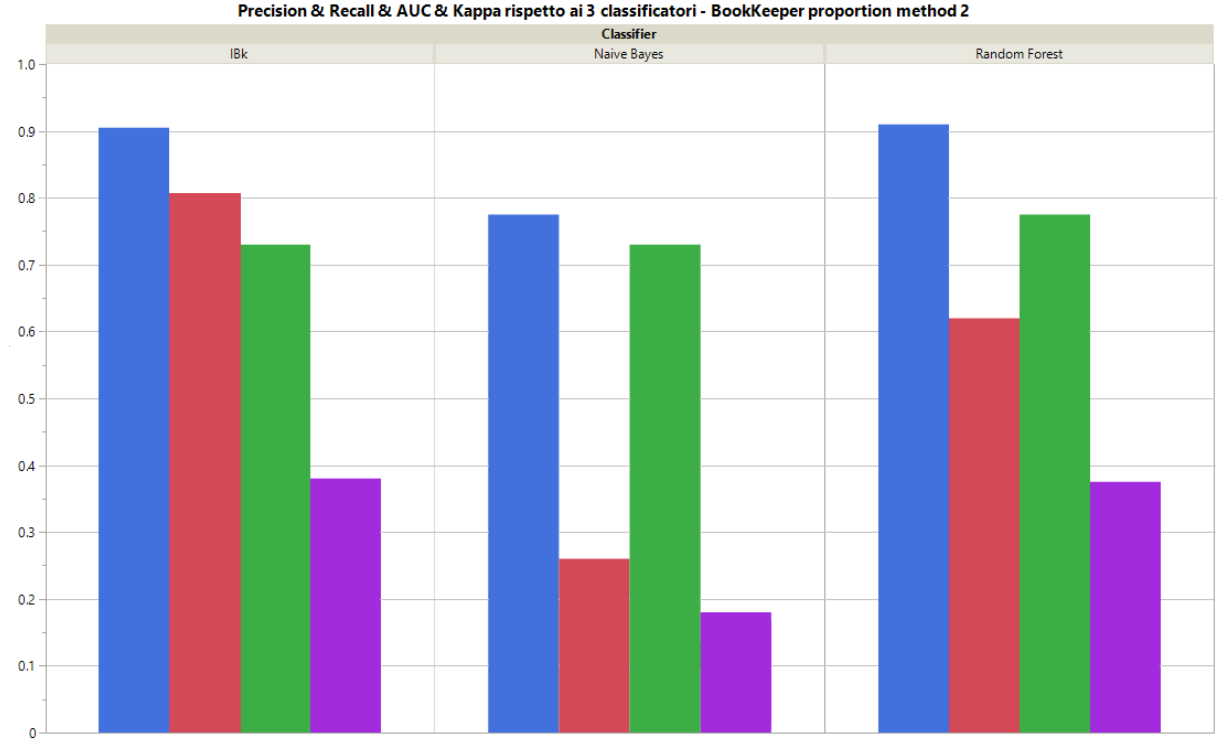
Risultati

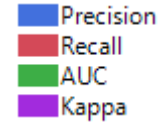
- Nel grafico è stata riportata la percentuale delle classi difettose per ogni release per il progetto Syncope analizzato, per entrambi i metodi di Proportion implementati.
- Si può notare come la scelta di uno dei due metodi di Proportion non influisce sul risultato finale, in quanto l'andamento osservato è praticamente identico nei due casi.
- Inoltre, osservando l'andamento per il metodo 2 di Proportion, rispetto a BookKeeper si osserva un comportamento più stabile fino alla release 22 con valori di picco significativamente più bassi.



Risultati



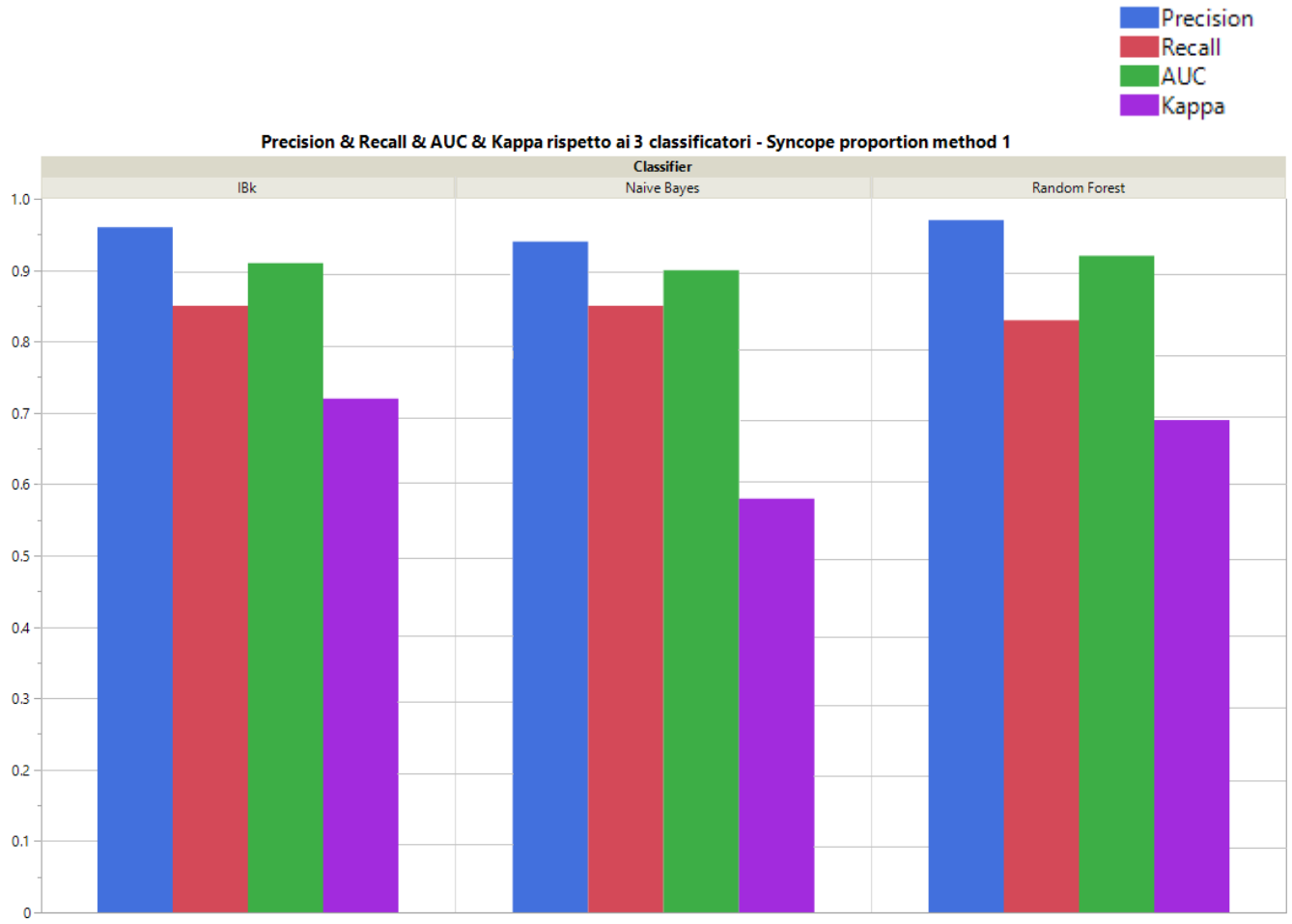




- Nelle figure vengono mostrati i risultati ottenuti per BookKeeper, per entrambi i metodi proportion implementati, al variare dei diversi classificatori.
- Il metodo 2 presenta valori di Precision nettamente più alti rispetto al metodo 1, per tutti i classificatori utilizzati, arrivando anche a valori di 0.9 per i classificatori ***IBK*** e ***Random Forest***. Il dataset creato con il metodo 2 infatti ha un numero molto più elevato di positivi rispetto a quello creato con il metodo 1, e valori di precision alti vanno a significare che nel metodo 2 sono stati commessi molti meno errori nel valutare le istanze come positive rispetto al metodo 1. Per quanto riguarda la Recall, invece, si osserva un peggioramento delle prestazioni nel metodo 2, in quanto i valori si abbassano leggermente.
- Per quanto riguarda i classificatori ***IBK*** e ***Naive Bayes*** si osserva che, a prescindere dal metodo di proportion implementato, il valore della Recall resta più o meno stabile, mentre si abbassa nel caso di ***Random Forest*** utilizzando il metodo 2.
- Il valore di AUC si abbassa utilizzando il metodo 2 per i classificatori ***IBK*** e ***Random Forest***, così come il valore di Kappa.
- Si osserva che il classificatore ***Naive Bayes*** è l'unico che presenta valori di Precision, Recall, AUC e Kappa molto simili tra i due metodi di proportion utilizzati.

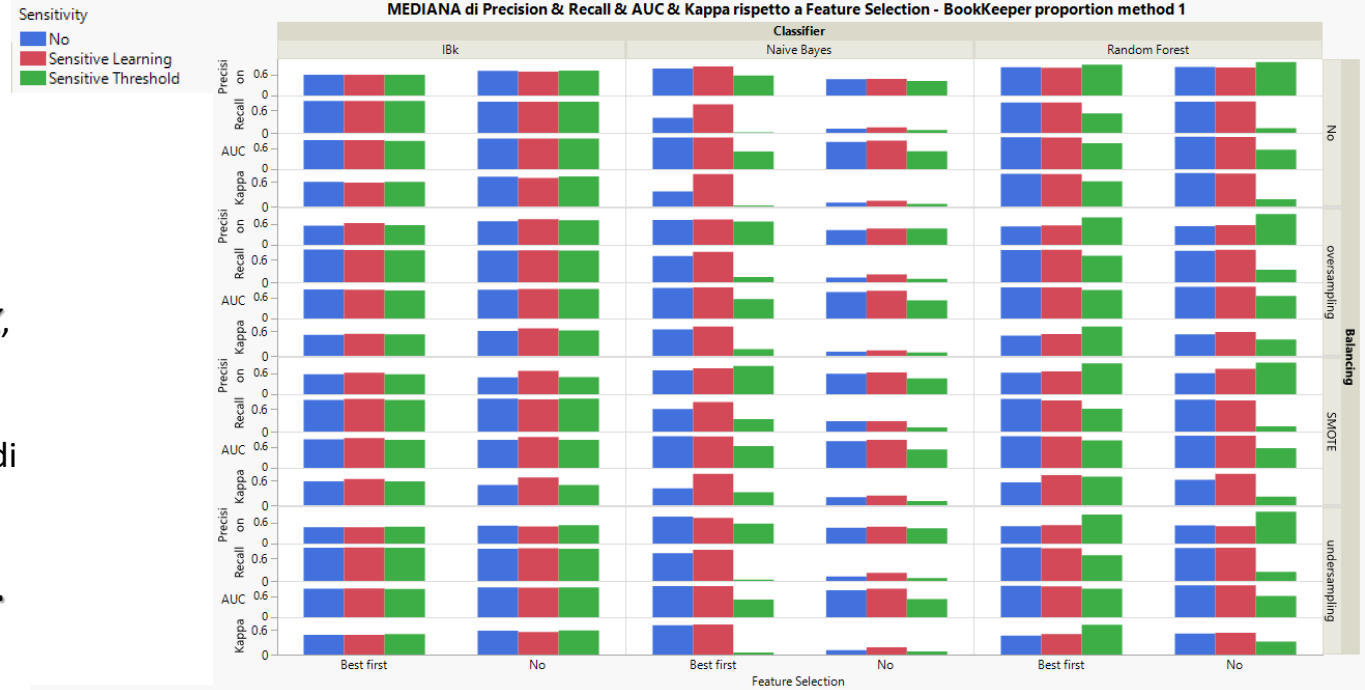
Risultati

- Per quanto riguarda Syncope, si è analizzato un solo metodo di proportion, in quanto i dataset sono pressoché identici per entrambi i metodi.
- Si può osservare come le prestazioni sono nettamente migliori rispetto a BookKeeper, in quanto si hanno, per tutti e 3 i classificatori, valori di Precision, Recall, AUC e Kappa molto elevati.
- Rispetto a BookKeeper, inoltre, i 3 classificatori si comportano più o meno tutti allo stesso modo, con valori delle metriche molto simili tra loro, tranne il caso di ***Naive Bayes*** in cui si ha un valore di Kappa leggermente inferiore agli altri classificatori.
- La Precision è molto alta in tutti e 3 i classificatori, stando a significare che sono state classificate correttamente molte istanze come positive, e anche i valori di Recall e AUC sono notevolmente elevati.
- Si osserva che il valore di Recall è praticamente identico per tutti e 3 i classificatori.

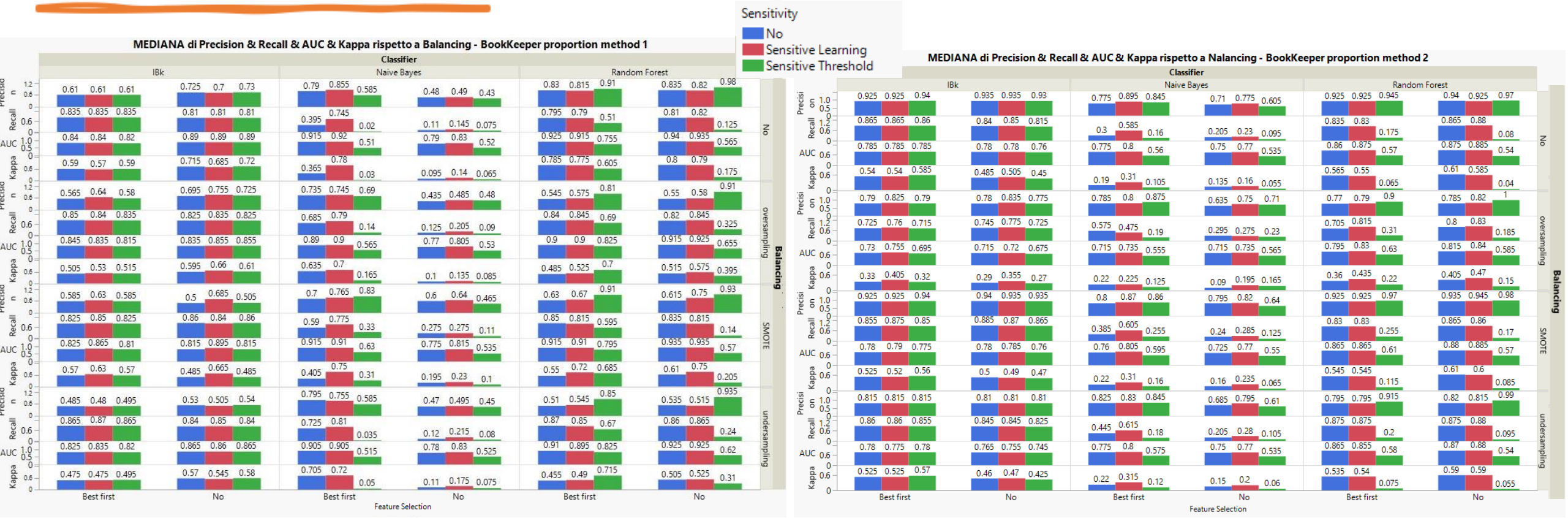


Risultati

- A fianco si mostrano i grafici di BookKeeper, utilizzando il metodo 1 di proportion, al variare delle tecniche di Feature Selection, Balancing e Sensitivity.
- Si può osservare che il classificatore che fornisce prestazioni migliori è ***IBK***, in quanto la mediana di Precision, Recall, AUC e Kappa è in media superiore rispetto agli altri classificatori, e in generale tali metriche hanno valori molto elevati utilizzando tale classificatore. Si osserva che l'utilizzo di tecniche di Balancing, Feature Selection e Sensitivity non cambia di molto le prestazioni di IBK. Per tale classificatore la combinazione migliore sembra essere **{No balancing, no Feature Selection, Sensitive Threshold}.**
- Comportamenti differenti invece ha il classificatore ***Naive Bayes***, per il quale si osservano prestazioni peggiori per ogni metrica utilizzando il Sensitive Threshold, indipendentemente dalle tecniche di Feature Selection e Balancing utilizzate. Per tale classificatore si osserva un leggero miglioramento delle prestazioni utilizzando Best first e SMOTE, indipendentemente dalla tecniche di Sensitivity utilizzate. Utilizzando il Sensitive Threshold si hanno valori di Recall e Kappa sempre molto bassi. Non applicare le tecniche di Feature Selection porta ad un peggioramento delle prestazioni soprattutto nei valori di Recall e Kappa, per tutte le tecniche di Balancing. Per tale classificatore la combinazione migliore sembra essere **{Best First, no Balancing, Sensitive Learning}.**
- ***Random Forest*** ha prestazioni leggermente migliori di Naive Bayes, anche se il non utilizzo di tecniche di Feature Selection affiancato all'uso di Sensitive Threshold anche in questo caso porta a un peggioramento. Per tale classificatore la combinazione migliore sembra essere **{Best First, no Balancing, Sensitive Learning}.**



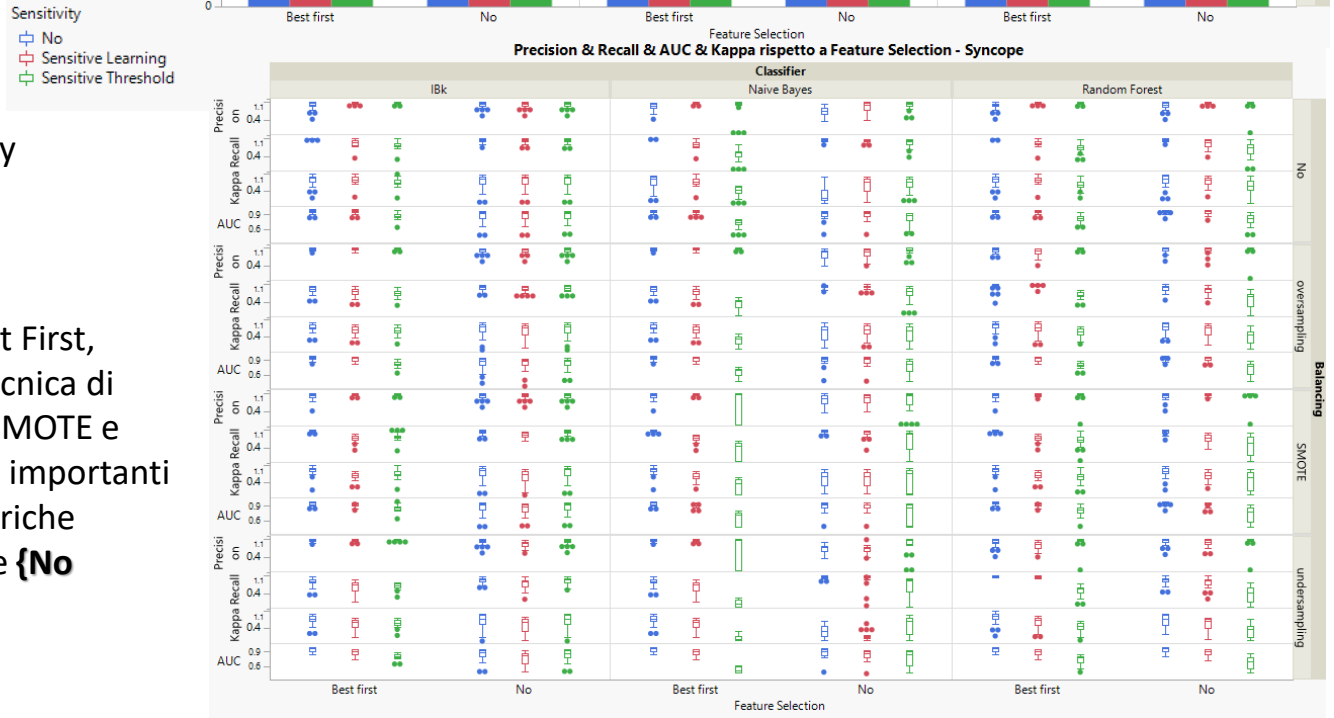
Risultati



- Mettendo a confronto i due metodi di proportion per BookKeeper, si osserva che, nel caso dell'utilizzo del metodo 2, si osserva una notevole differenza delle prestazioni del classificatore ***Random Forest*** nel caso dell'utilizzo di Best First e Sensitive Threshold. Si osserva infatti che, rispetto al metodo 1, i valori di Recall e Kappa sono di gran lunga inferiori.
- Per quanto riguarda ***IBK***, invece, nel grafico relativo al metodo 2 si osserva un miglioramento generale delle prestazioni, per qualunque tipo di Balancing, Feature Selection e Sensitivity. Si osserva in particolare come utilizzando SMOTE, no , Feature Selection e Sensitive Threshold si ha un notevole aumento di tutte le metriche.
- Nel caso di ***Naive Bayes*** invece, utilizzando Best First si hanno prestazioni migliori nel caso del metodo 1, mentre il metodo 2 si comporta meglio non utilizzando tecniche di Feature Selection.

Risultati

- Per quanto riguarda Syncope, si hanno prestazioni notevolmente migliori rispetto a BookKeeper per ogni classificatore, anche se si ha un notevole aumento di outliers, ossia valori anomali presenti nel dataset.
- Il classificatore peggiore sembra essere ***Naive Bayes***, in quanto si hanno valori di Precision, Recall, AUC e Kappa inferiori rispetto agli altri classificatori. In particolare si può osservare come l'utilizzo del Sensitive Threshold affiancato a Best First porta a valori molto bassi di Recall e Kappa, e in generale si osserva un abbassamento di tutte le metriche. Soprattutto utilizzando Undersampling si osserva un peggioramento delle prestazioni. Per quanto riguarda questo classificatore, la combinazione migliore sembra essere **{No Feature Selection, SMOTE, Sensitive Threshold}**
- ***IBK*** sembra essere il classificatore che si comporta in modo migliore, in quanto per tutte le metriche si osservano valori elevati e non si ha una grossa varianza nei dati. In generale si hanno prestazioni simili indipendentemente dalla tecnica di Feature Selection, Balancing e Sensitivity scelte. Per tale classificatore la combinazione migliore sembra essere **{No balancing, no Feature Selection}**, indipendentemente dal Sensitivity applicato.
- ***Random Forest*** si comporta in modo simile a IBK soprattutto utilizzando Best First, mentre si ha un peggioramento delle prestazioni non utilizzando nessuna tecnica di Feature Selection e utilizzando Sensitive Threshold, soprattutto nel caso di SMOTE e Undersampling. Si osserva come la scelta di balancing non porti a differenze importanti nelle prestazioni. In generale si hanno valori leggermente inferiori delle metriche rispetto a IBK. Per tale classificatore la combinazione migliore sembra essere **{No balancing, no Feature Selection, Sensitive Learning}.**



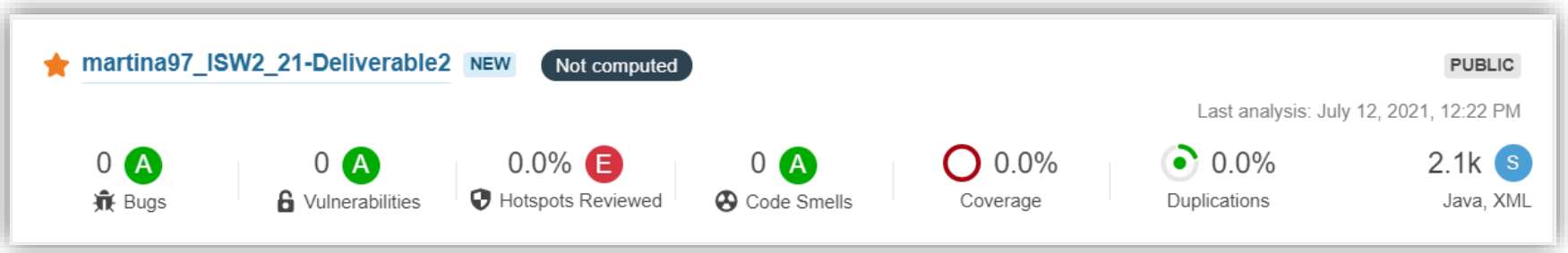
Conclusione

- In generale, non utilizzare tecniche di Balancing e di Feature Selection ha mostrato prestazioni migliori per tutti i classificatori e per tutti i dataset.
- Utilizzare Sensitive Threshold porta spesso a prestazioni scarse, in quanto i valori delle metriche si abbassano notevolmente.
- Sia per BookKeeper che per Syncope, il classificatore migliore è IBK, utilizzando **{No Balancing, No Feature Selection, Sensitive Threshold}.**

Links

- Link SonarCloud:

https://sonarcloud.io/dashboard?id=martina97_ISW2_21-Deliverable2



- Link GitHub:

https://github.com/martina97/ISW2_21-Deliverable2

- Per una visione migliore dei grafici si rimanda al seguente link:

https://github.com/martina97/ISW2_21-Deliverable2/tree/main/Immagini%20ML