Deliverable 2

ML FOR SE

MARTINA SALVATI 0292307

Contenuto delle diapositive

Introduzione

- Obiettivi del progetto, Tecnologie utilizzate, Progetti open-source

Progettazione

- Introduzione alla progettazione

Milestone One
Costruzione del dataset

- Estrazione ticket da Jira
- Estrazione commit da git
- Mapping Jira/Git
- Metodo proportion con MW
- Features

Milestone Two
Applicazione degli algoritmi ML e analisi dati

- Creazione file ARFF
- Walk Forward
- Cost Sensitive classifier, Features Selection e Sampling
- Weka Evaluator

Risultati

- Bookkeeper
- Openjpa

Riferimenti

- Github
- SonarCloud

Introduzione

Obiettivi del progetto

Eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling, classificazioni sensibili al costo, e feature selection, sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni open-source

Tecnologie utilizzate

JIRA
servizio per monitoraggio di ticket e progetti

Weka
Machine Learning Software

Github
servizio di hosting dei progetti

Git
Version Control System

JMP
Per l'analisi statistica dei risultati

SonarCloud
servizio di sicurezza e qualità del codice basato sul cloud

Progetti open source

Bookkeeper

Openjpa

Introduzione alla progettazione

Per lo svolgimento del progetto è stato sviluppato un progetto in Java

- Le librerie utilizzate per lo svolgimento del progetto
  - **Jgit** per l'interazione con il repository Git
  - **Weka API** per Java per ML

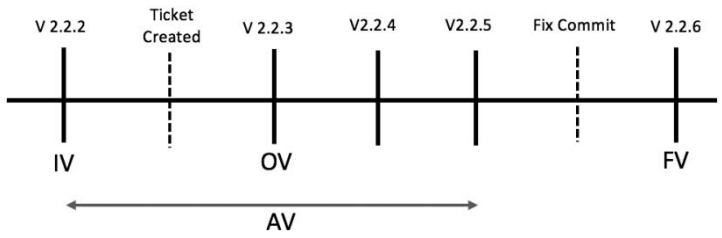
Il progetto in Java si divide in due milestone

- **Milestone 1** : Costruzione del dataset
  - Avviene la costruzione del dataset tramite merging delle informazioni estrapolate da Jira e Git
  - Dove possibile si acquisiscono le informazioni da Jira, altrimenti si utilizza la tecnica del *proportion* per stabilire se una classe è buggy o meno
  - Le informazioni raccolte si trasformano in un CSV
- **Milestone 2:** Applicazione degli algoritmi ML e analisi dati
  - Walk-Forward per lo splitting del dataset
  - Applicazione degli algoritmi ML : Cost Sensitive classifier, Features Selection e Sampling
  - Analisi risultati ottenuti e ricerca di pattern tramite JMP

MILESTONE ONE

Costruzione del dataset

Costruzione del dataset : Estrazione ticket da Jira



Recupero ciclo vita difetti da JIRA (se presente AV)

- Tramite Jira è stato possibile recuperare lo storico dei bug delle varie classi del progetto associata ai ticket.
  - **OV** : versione in cui è stato aperto il ticket – disponibile nei report Jira relativi ai difetti
  - **AV** : si trovano nell'intervallo [IV, FV) e rappresentano le versioni in cui la classe è stata affetta dal bug. Se presente nel ticket e *consistente*, viene recuperata.
  - **IV** : la prima versione dell'AV
  - **FV** : informazione estrapolata dalla data del *fix commit* estrapolato da Git. In particolare, è la release successiva alla data dell'ultimo commit che contiene nel commento l'ID del ticket.

Verifica delle informazioni

- Le informazioni estrapolate da JIRA vengono sempre controllate
- Se una informazione non è coerente, non viene presa in considerazione.
  Ad esempio:
  - Se presente AV si controlla che IV ≤ OV ; altrimenti si scarta
  - Una versione in cui non è presente la data di versioning non viene considerata
  - L'ultimo 50% delle versioni non viene considerato per diminuire il missing rate

Costruzione del dataset : Estrazione commit da Git

Recupero dei commit da git

- Vengono recuperati i *commit* da git e salvate le *informazioni necessarie* per le metriche tramite libreria JGIT
- Vengono filtrati i commit linkati a Jira tramite *Regular Expressions* nel commento

Recupero dello storico delle classi

- Per ogni classe, dai commit, viene recuperata il ciclo di vita della stessa attraverso i commit relativi alla classe (add, modify, delete)
- **Adding version** : versione in cui la classe è stata aggiunta nel progetto
- Queste informazioni sono necessarie per rendere l'informazione di ogni classe coerente, vengono interpolate le informazioni della classe con le informazioni generate dal metodo proportion

Assunzioni prese

- Una versione in cui non sono presenti commit non viene considerata
- Adding version >= IV

Tutte le informazioni estrapolate vengono salvate in file JSON

- Per il progetto è stato utilizzato il meccanismo di JSONPath per l'estrapolazione delle informazioni dai JSON creati

Mapping Jira/Git

Merging info da Jira e Git

- Le informazioni ricavate da Jira e Git vengono utilizzate per l'estrapolazione di informazioni necessarie alla costruzione del dataset

Estrazione della Fix Versions dal Fix Commit

- Con il merging delle info da Jira e Git viene prodotta la Fix versions, versione in cui è presente il commit che risolve il bug
- Vengono utilizzate le Regular Expression in Java per la ricerca dell'ID del ticket preso da Jira nel commento del commit
  - E' possibile che siano presenti più commit riferiti allo stesso ticket, per questo viene sempre considerato l'ultimo

Alcune assunzioni di progetto

- Se IV=FV allora AV=0 e quindi è come se non stessimo considerando il ticket in quanto i difetti post-rilascio non vengono considerati
- Se OV=1 allora IV=1 in quanto non può avvenire prima
- Se FV= OV allora IV=FV e quindi AV=0

Alcuni risultati ottenuti dal merging delle informazioni

Bookkeeper	
Linkage 1 (#CommitLinked/#Commit)	32%
Linkage 2 (Bug Fixed Ticket Jira/Total Ticket Git)	93%

Openjpa	
Linkage 1 (#CommitLinked/#Commit)	65%
Linkage 2 (Bug Fixed Ticket Jira/Total Ticket Git)	85%

Metodo proportion con MW

Calcolare IV

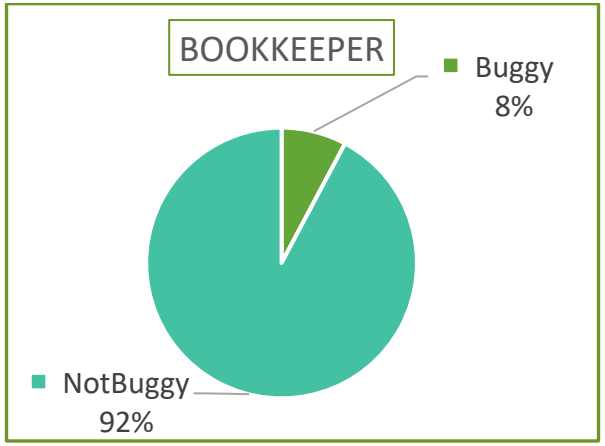
- Per il calcolo di IV i difetti sono stati ordinati in base alla data
- Per ogni difetto all'interno di un progetto, si è utilizzato la **P** media tra l'ultimo 1% dei difetti fissi
- $P = \frac{FV-IV}{FV-OV} \rightarrow IV = FV - (FV - OV) * P$
- Il calcolo di IV viene effettuato in maniera iterativa per ogni difetto riferito a una o più classi

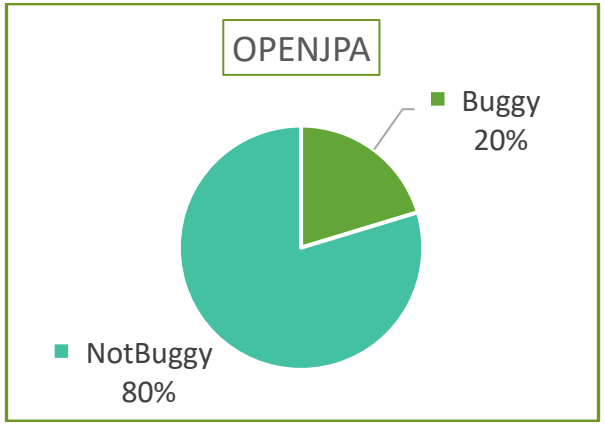
Metodo Moving Window

- Si è scelta la lunghezza della moving windov pari all'**1%** dei difetti precedenti
- Questo è stato fatto in quanto preso come compromesso tra la capacità di reagire ai cambiamenti e la capacità di assorbirli
- MW in Bookkeeper : *44*
- MW in Openjpa : *110*

Etichettare la classe difettosa

- Per ogni difetto è stata etichettata ogni versione della classe prima dell'IV come non buggy.
- Mentre viene etichettata ogni versione dalla IV inclusa alla FV esclusa come buggy.
- La FV è etichettata come non buggy.





Features

Link ai file generati dalla MilestoneOne

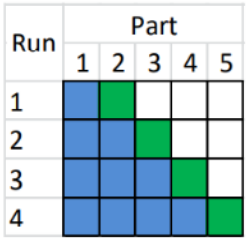
- BOOKKEEPER
- OPENJPA

1. **NR** : NUMERO DI REVISIONI
2. **NAUTH** : NUMERO AUTORI NELLA DETERMINATA RELEASE
3. **CHURN** : SOMMA DURANTE LE REVISIONI DI (added-delete) LOC
4. **MAX CHURN** : MASSIMO CHURN DURANTE LE REVISIONI
5. **AVERAGE CHURN** : CHURN MEDIO DURANTE LE REVISIONI
6. **LOC ADDED** : SOMMA DELLE OPERAZIONI DI ADDED DURANTE LE REVISIONI
7. **MAX LOC ADDED**: MASSIMO LOC ADDED DURANTE LE REVISIONI
8. **AVG LOC ADDED** : MEDIA LOC ADDED DURANTE LE REVISIONI
9. **LOC TOUCHED** : SOMMA DI LOC (added+delete) DURANTE LE REVISIONI
10. **AGE:** ETA' DELLE VERSIONI IN SETTIMANE
11. **WEIGHTED AGE:** $\frac{\sum_{i=1}^{N} Age(i) \times addedLOC(i)}{\sum_{i=1}^{N} addedLOC(i)}$
12. **SIZE :** SOMMA LINEE DI CODICE (LOC)

MILESTONE TWO

Applicazione degli algoritmi ML e analisi dati

Analisi del dataset : Creazione file ARFF e Walk Forward



(a) Walk-forward

Creazione file ARFF

- Il primo step è la conversion del CSV ottenuto dalla milestone ONE in file ARFF
- Il file ARFF è necessario per l'analisi dati con Weka
- Dando in input a weka il file ARFF è possibile l'applicazione degli algoritmi machine learning

Walk Forward

- Per la creazione del training e testing set è stata utilizzata la tecnica per serie temporali Walk-Forward
- Le parti vengono ordinate cronologicamente e, in ogni esecuzione, tutti i dati disponibili prima della parte da prevedere vengono utilizzati come set di training e la parte da prevedere viene utilizzata come set di test
- Il numero di esecuzioni è uguale a uno in meno rispetto al numero delle parti
- Nel nostro caso il numero delle parti è il numero di *versioni*

Applicazione algoritmi ML : Cost Sensitive classifier, Features Selection, Sampling e Classificatori :1

Feature selection
- No selection
- Best first

Balancing
- No sampling
- Oversampling
- Undersampling
- SMOTE

Cost Sensitive Classifier (CFN = 10*CFP)
- Sensitive Threshold Classifier
- Sensitive Learning Classifier
- No Cost Sensitive Classifier

Classificatori
- RandomForest
- NaiveBayes
- Ibk

- Sono usati diversi algoritmi ML per effettuare l'analisi del dataset ottenuto per la predizione della bugyness delle classi
- Per implementare i diversi algoritmi è stata usata la semplice API di WEKA per Java
- E' stata utilizzata la tecnica del FILTRO per l'applicazione degli algoritmi di Feature Selection e Balancing
- I metodi di filtro selezionano le funzionalità da un set di dati in modo indipendente per qualsiasi algoritmo di machine learning

```
//oversampling of the minority class
public static Resample oversampling(Instances data) {

    int numberOfBuggy= returnNumOfBuggyClass(data);
    double samplesizemajority = computeSampleSizePercentMajorityClass(data,numberOfBuggy);
    if(samplesizemajority<=0) {
        samplesizemajority = computeSampleSizePercentMajorityClass(data,returnNumOfNotBuggyClass(data));
    }
    final Resample filter = new Resample();
    filter.setBiasToUniformClass(1.0);
    try {
        filter.setInputFormat(data);
        filter.setNoReplacement(false);
        filter.setSampleSizePercent(samplesizemajority);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return filter;
}
```

Applicazione algoritmi ML : Cost Sensitive classifier, Features Selection, Sampling e Classificatori : 2

Feature selection
- No selection
- Best first

Balancing
- No sampling
- Oversampling
- Undersampling
- SMOTE

Cost Sensitive Classifier (CFN = 10*CFP)
- Sensitive Threshold Classifier
- Sensitive Learning Classifier
- No Cost Sensitive Classifier

Classificatori
- RandomForest
- NaiveBayes
- Ibk

Per l'applicazione del sampling sono state effettuate delle misurazioni:

- **OverSampling** : è stato necessaria la computazione in percentuale del numero di istanze della classe maggioritaria

```
public static double computeSampleSizePercentMajorityClass(Instances data, int numOfFirstClass) {
    int numOfSecondClass = data.numInstances() - numOfFirstClass;
    return 100 * ((double)(data.numInstances() +
            Math.abs(numOfFirstClass - numOfSecondClass))
            / data.numInstances());
}
```

- **Smote** : è stato necessaria la computazione di un valore in percentuale del numero di istanze di differenza della prima classe con la seconda

```
public static double computeSampleSizeSmote(Instances data, int numOfFirstClass) {
    int numOfSecondClass = data.numInstances() - numOfFirstClass;
    return Math.round(100 * ((double)(numOfSecondClass-numOfFirstClass)/numOfFirstClass));
}
```

Per l'applicazione del cost sensitive classifier è stata effettuata la creazione della matrice dei costi

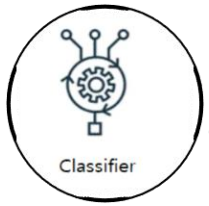
```
CostMatrix costMatrix = new CostMatrix(2);
costMatrix.setCell(0, 0, 0.0);
costMatrix.setCell(1, 0, weightFalsePositive);
costMatrix.setCell(0, 1, weightFalseNegative);
costMatrix.setCell(1, 1, 0.0);
return costMatrix;
```

Weka Evaluator

- Una volta effettuate tutte le combinazioni possibili del dataset è stato possibile effettuare la computazione tramite il Weka Evaluator delle metriche
- Questo ha permesso di generare un file contenente tutte le informazioni necessarie per rispondere alla domanda del progetto

TP	TF	FP	FN
TP RATE	FP RATE	PRECISION	RECALL
F MEASURE	AUC	KAPPA	ACCURACY

Quali tecniche di feature selection o balancing aumentano l'accuratezza dei classificatori?



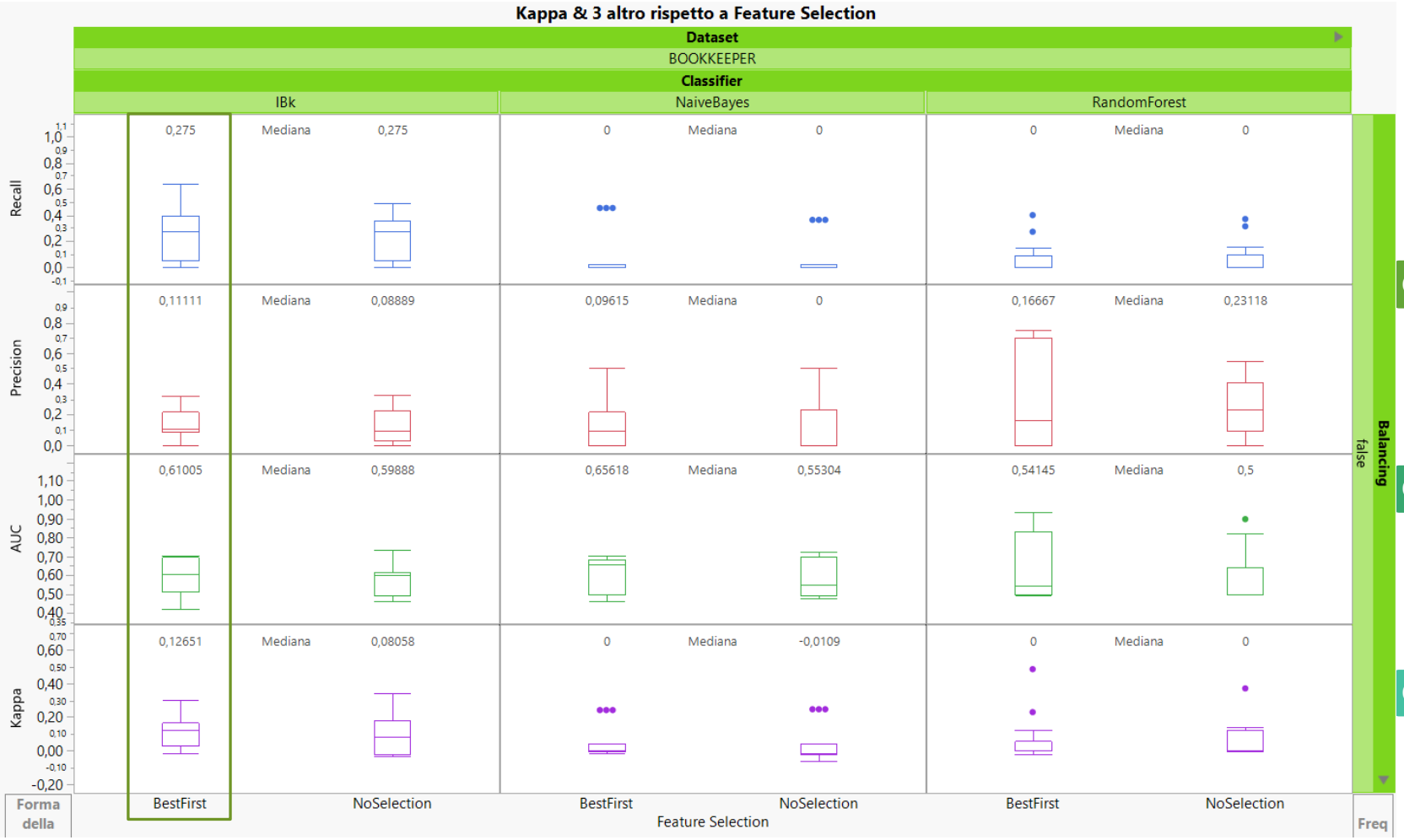
Per quali classificatori

Per quali dataset

?

Link al file :

Risultati Weka evaluator



Risultati 1 : Bookkeeper selection

Classificatore Ibk

- L'algoritmo di features selection BestFirst migliora l'accuratezza : precision, AUC e Kappa risultano essere migliorate nel caso in cui BestFirst è applicato. Mentre la recall risulta essere stabile per entrambi i casi. In conclusione, Ibk è migliore con BestFirst.

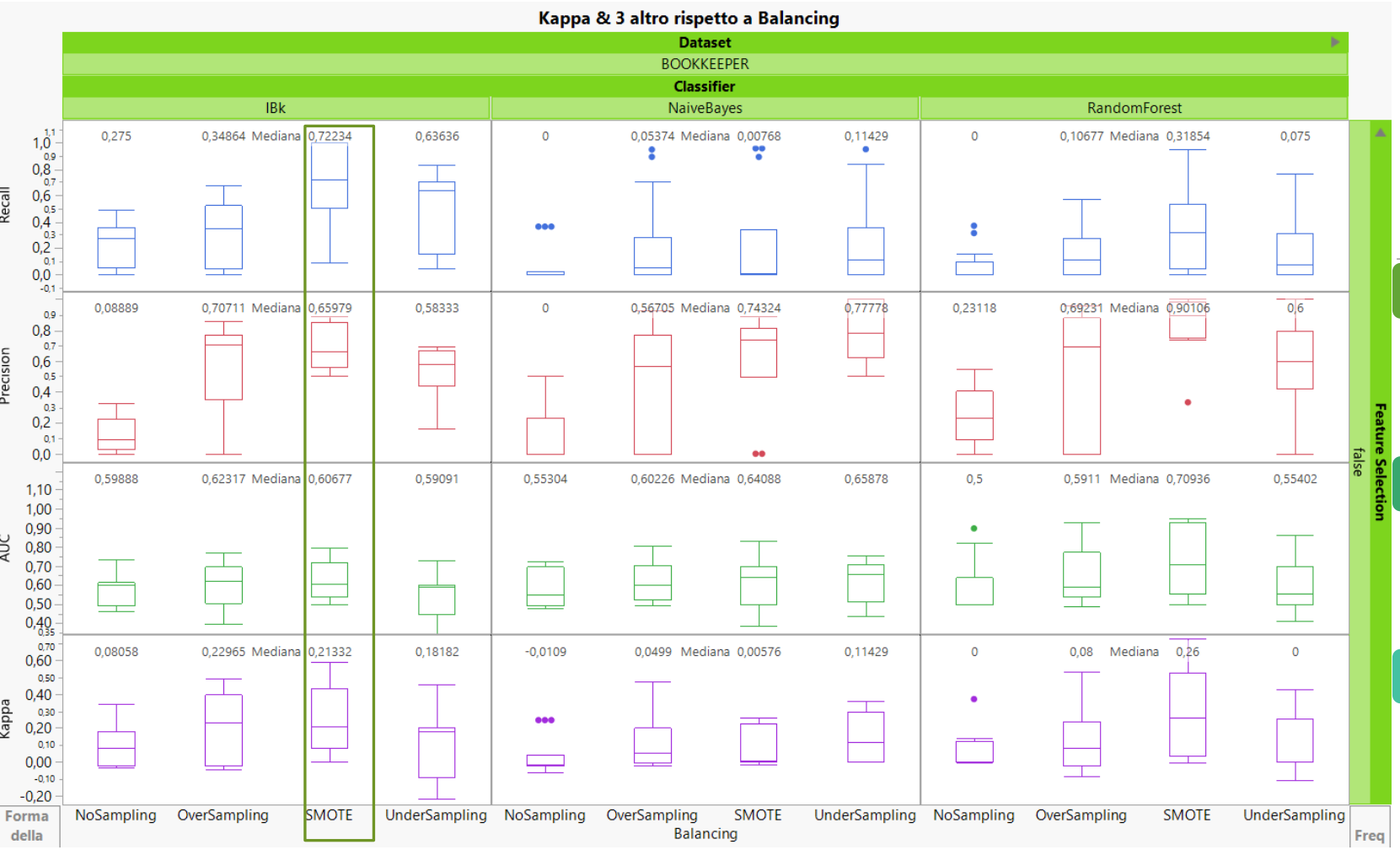
Classificatore NaiveBayes

- L'algoritmo di features selection BestFirst risulta migliorare l'accuratezza del classificatore NaiveBayes. I risultati risultano molto bassi, questo è dovuto al fatto che il sampling non è considerato. In altre considerazioni si vedrà il miglioramento con sampling.

Classificatore RandomForest

- L'algoritmo di feature selection BestFirst risulta migliorare l'accuratezza del classificatore RandomForest. Il caso senza sampling è uno dei peggiori per il dataset Bookkeeper.

Il dataset Bookkeeper risulta avere perfomance migliori con l'applicazione dell'algoritmo feature selection BestFirst nel caso in cui il sampling non è considerato. I risultati risultano molto negativi, questo ci suggerisce che il sampling probabilmente è fondamentale per questo dataset.



Risultati 2 : Bookkeeper sampling

Classificatore Ibk

- La tecnica di balancing **SMOTE** risulta migliorare in media le metriche di accuratezza del classificatore, mentre la non applicazione di alcun metodo di balancing ne peggiora le metriche.

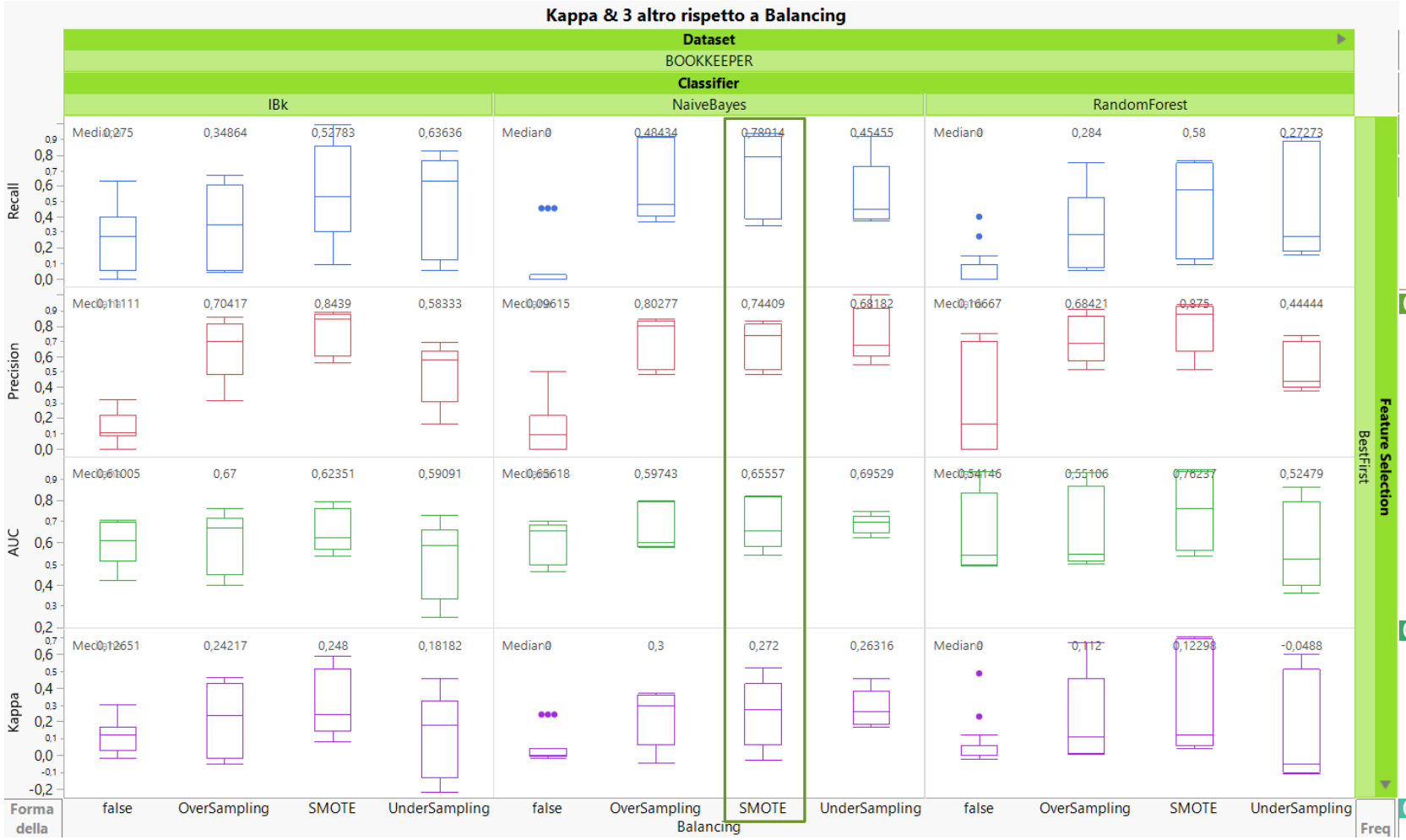
Classificatore NaiveBayes

- La tecnica di balancing **UnderSampling** risulta migliorare in media le metriche di accuratezza del classificatore, mentre la non applicazione di alcun metodo di balancing ne peggiora le metriche.

Classificatore RandomForest

- La tecnica di balancing **SMOTE** risulta migliorare in media le metriche di accuratezza del classificatore, mentre la non applicazione di alcun metodo di balancing ne peggiora le metriche

Il dataset Bookkeeper risulta avere perfomance migliori con l'applicazione del sampling (SMOTE) , al contrario senza l'applicazione del balancing le performance peggiorano. Si può notare come con il sampling il dataset Bookkeeper risulta migliorare rispetto alla slide precedente.



Risultati 3 : Bookkeeper sampling w selection

Classificatore Ibk

- I risultati confermano che la tecnica SMOTE abbia risultati migliori rispetto alle altre tecniche di sampling anche in presenta di features selection. L'applicazione di feature selection diminuisce la precision ma aumenta la recall. In generale, prendendo in considerazione F1 abbiamo un peggioramento del 0,9%. Dati I risultati ottenuti e I notevoli vantaggi dell'applicazione di feature selection (alleggerimento dataset, tempistiche) si può confermare che Ibk migliora con l'applicazione di SMOTE e BestFirst.

Classificatore NaiveBayes

- I risultati confermano che la tecnica BestFirst migliora l'accuratezza del classificatore NaiveBayes. La tecnica SMOTE ha risultati migliori rispetto UnderSampling, che sembrava la migliore senza la presenza di feature selection.

Classificatore RandomForest

- I risultati confermano che la tecnica SMOTE migliora l'accuratezza – insieme con l'applicazione di features selection.

Il dataset Bookkeeper migliora con l'applicazione del sampling SMOTE e features selection BestFirst. Il classificatore NaiveBayes con l'applicazione SMOTE e BestFirst risulta il migliore tra i classificatori.

Il dataset Openjpa non presenta risultati rilevanti con l'applicazione della sola tecnica di feature selection, ma sembra comportarsi meglio senza l'applicazione di alcuna tecnica.

Risultati 4 : Openjpa selection

Classificatore Ibk

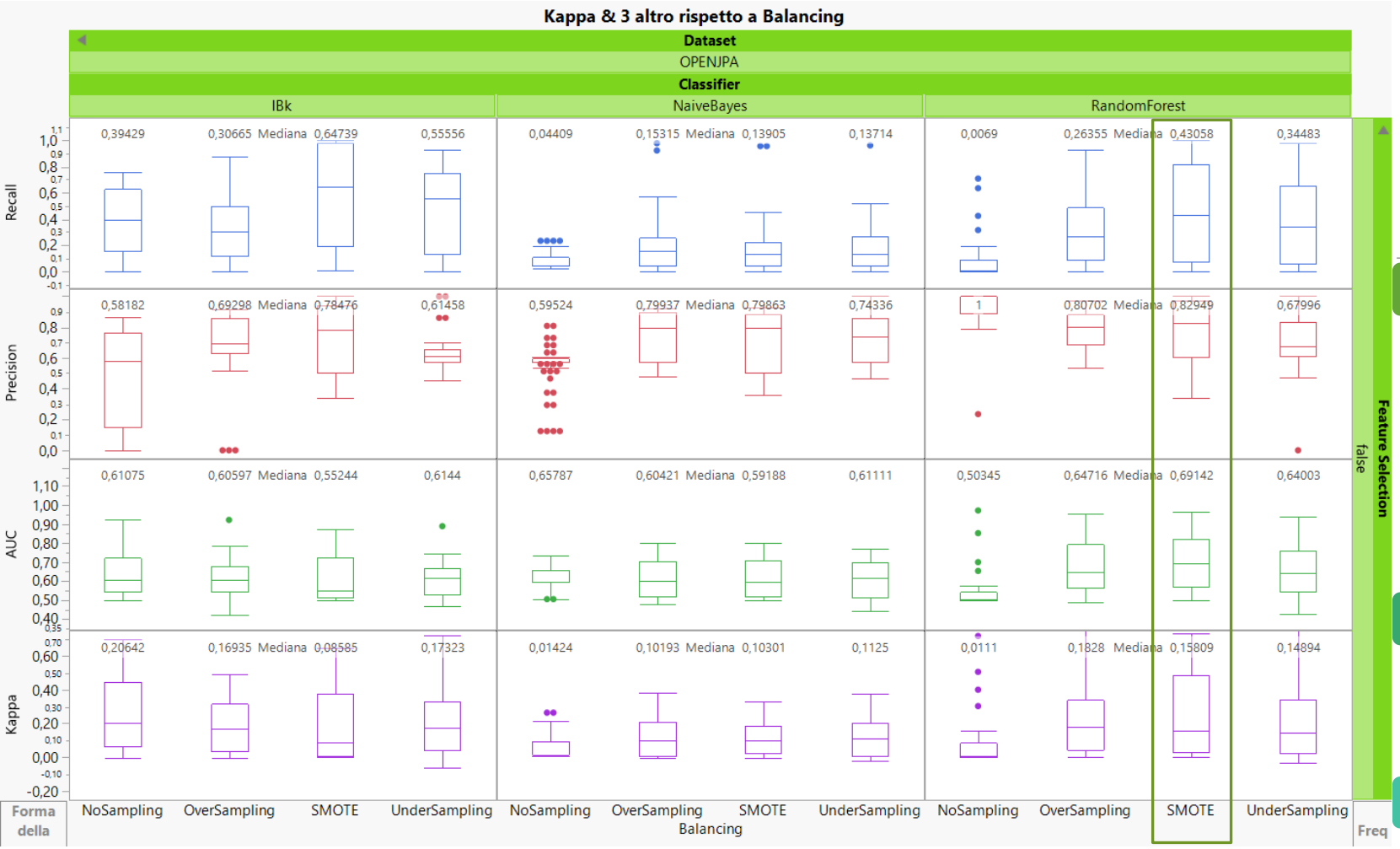
- Il classificatore Ibk migliora la Recall (del 21%) e Kappa e Precision(6/7%) in assenza di alcun algoritmo di feature selection. Dai risultati si evince che la non applicazione di feature selection migliori le prestazioni del classificatore Ibk.

Classificatore NaiveBayes

- Il classificatore NaiveBayes con l'applicazione di BestFirst peggiora AUC(10%) e Precision(6%), mentre migliora leggermente la Recall e Kappa. Dai risultati si evince che la non applicazione di feature selection migliora le prestazioni di NaiveBayes.

Classificatore RandomForest

- L'algoritmo RandomForest presenta dei risultati particolari. I valori di precision molto alti (0,8/1) mentre il valore di Recall molto bassi. Questo avviene in quanto il dataset risulta molto sbilanciato e il classificatore tende a classificare i valori come la classe maggioritaria. In generale, RandomForest sembra migliorare in presenza dell'algoritmo di feature selection BestFirst. I risultati ottenuti non sono soddisfacenti, questo fa intuire la necessità di utilizzare il sampling o altre tecniche per migliorare il dataset.



Risultati 5 : Openjpa sampling

Classificatore Ibk

- Il classificatore Ibk sembra migliorare le metriche di Recall e Precision con l'applicazione del sampling SMOTE. Mentre le metriche di AUC sono leggermente migliorate con la tecnica di UnderSampling. La metrica Kappa è leggermente migliore senza l'applicazione di alcuna tecnica di Sampling. Le tecnica di UnderSampling e Smote sembrano avere le prestazioni migliori con il classificatore Ibk : in particolare considerando I valori massimo UnderSampling ottiene un risultato migliore.

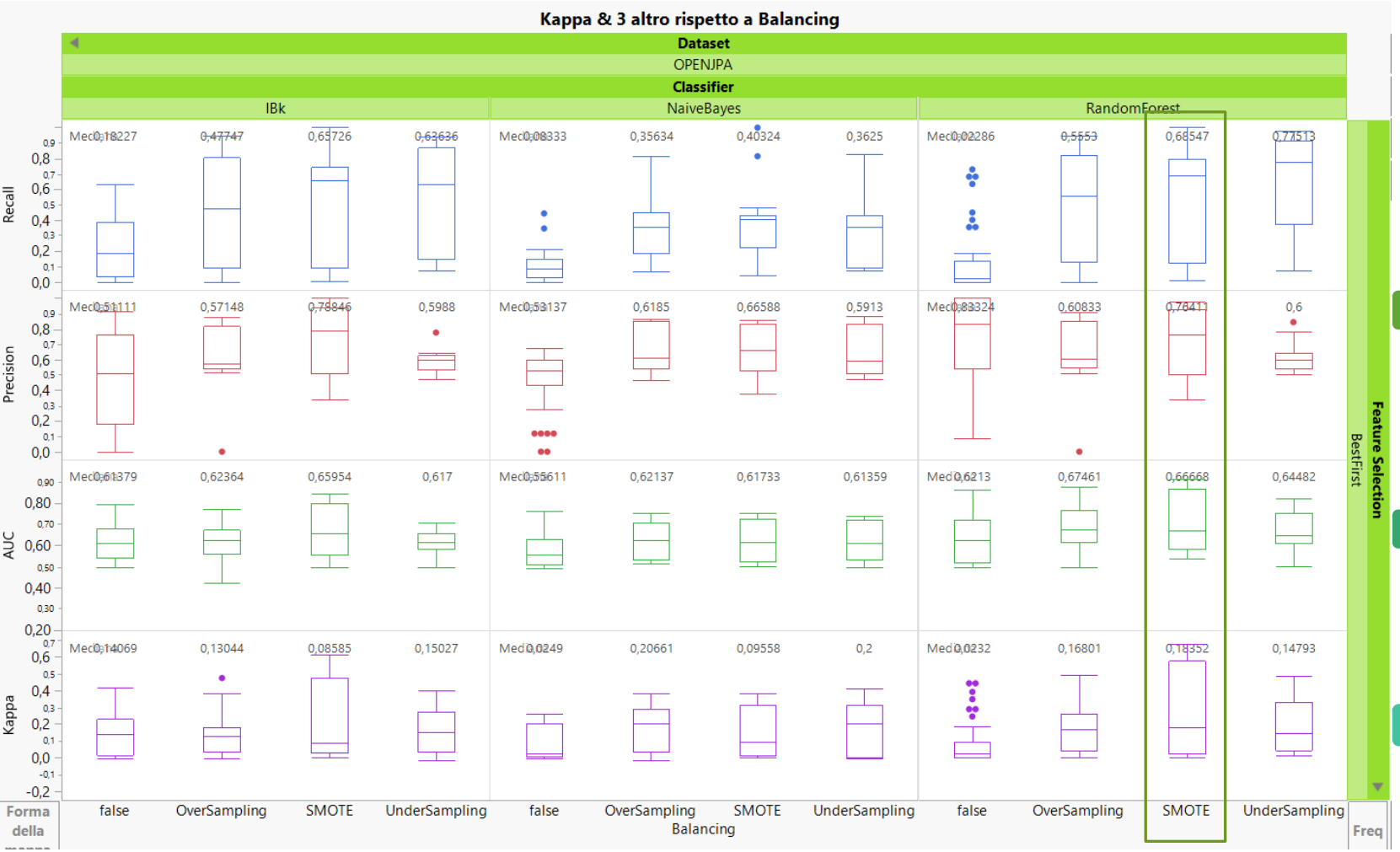
Classificatore NaiveBayes

- Il classificatore NaiveBayes in presenza di OverSampling (o similmente Smote) ottiene dei risultati migliori : Precision,Recall,AUC e Smote risultano sempre migliori rispetto alle altre applicazioni.

Classificatore RandomForest

- In presenza di sampling il classificatore RandomForest migliora notevolmente le prestazioni. In particolare, la tecnica di SMOTE migliora le metriche di Recall,Precision e AUC. Mentre la tecnica di OverSampling migliora leggermente (2%) la metrica Kappa.

Il dataset Openjpa risulta migliorare con l'applicazione del sampling. In particolare, confrontando i risultati (mediana,media e massimo) il classificatore RandomForest con sampling SMOTE ha risultati migliori.



Risultati 6 : Openjpa sampling w selection

Classificatore Ibk

- Il classificatore Ibk anche in presenza di feature selection sembra avere prestazioni migliori con l'applicazione della tecnica SMOTE. In particolare, confrontando con la slide precedente, si notano leggeri miglioramenti con feature selection

Classificatore NaiveBayes

- Il classificatore NaiveBayes anche in presenta di feature selection sembra avere prestazioni migliori con il sampling SMOTE e OverSampling. A differenza del risultato ottenuto senza feature selection, si notano miglioramenti con l'applicazione di BestFirst nel caso di sampling SMOTE

Classificatore RandomForest

- Il Classificatore RandomForest in presenta del sampling SMOTE e feature selection migliora la Recall del 20%, mentre diminuisce leggermente la precision rispetto al caso senza selection (7%). Nel caso di applicazione UnderSampling si nota un notevole miglioramento della recall (43%). In generale le metriche migliorano nel caso di applicazione sampling smote con feature selection.

Il dataset Openjpa ha prestazioni migliori in presenza di sampling e feature selection rispetto alle configurazioni precedenti.
In particolare, il classificatore che ottiene risultati migliori è RandomForest con tecnica SMOTE e best first.

Risultati 7 : Cost sensitive classifier Bookkeeper

Classificatore Ibk

- Il classificatore Ibk è l'unico dei classificatori che sembra ottenere risultati abbastanza stabili, confrontando la presenza o assenza del cost sensitive classifier.

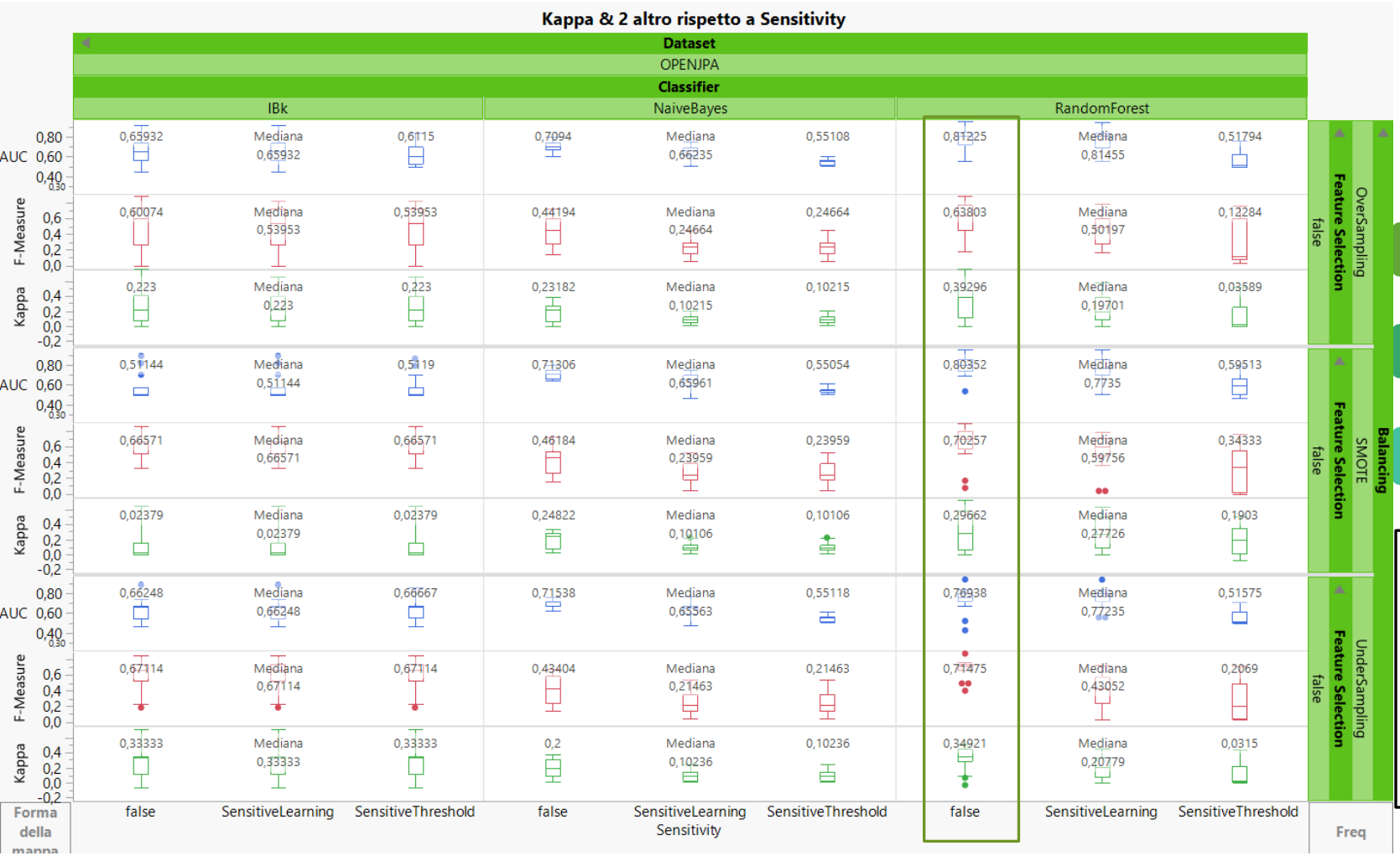
Classificatore NaiveBayes

- Il classificatore NaiveBayes non gestisce bene gli approcci cost sensitive classifier, soprattutto con threshold. Ma sembra gestire meglio gli approcci weighted. In generale, il classificatore ha risultati migliori in assenza del cost sensitive classifier.

Classificatore RandomForest

- Il Classificatore RandomForest non gestisce bene cost sensitive classifier; dai risultati le metriche risultano notevolmente più basse. In particolare, RandomForest si comporta come un classificatore O-R in presenta di cost sensitive classifier.

Dai risultati ottenuti, si può dire che in generale il dataset Bookkeeper presenta risultati migliori in assenza del cost sensitive classifier. Mentre si può affermare che i risultati migliori per Bookkeeper si hanno con l'applicazione del feature selection Best First e tecnica di sampling SMOTE, similmente per i classificatori Ibk e NaiveBayes.



Risultati 8 : Cost sensitive classifier Openjpa

Classificatore Ibk

- Risultati simili alla slide precedente

Classificatore NaiveBayes

- Risultati simili alla slide precedente

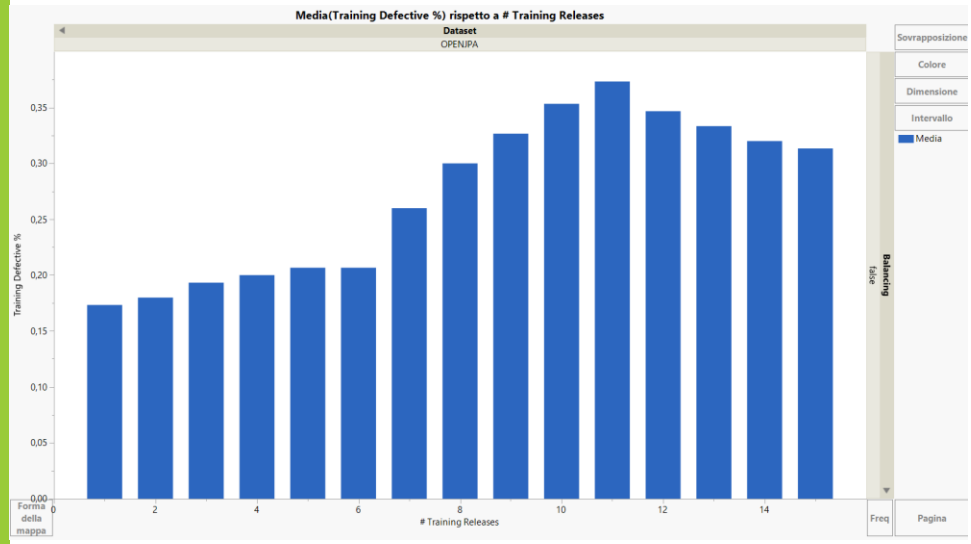
Classificatore RandomForest

- Risultati simili alla slide precedente

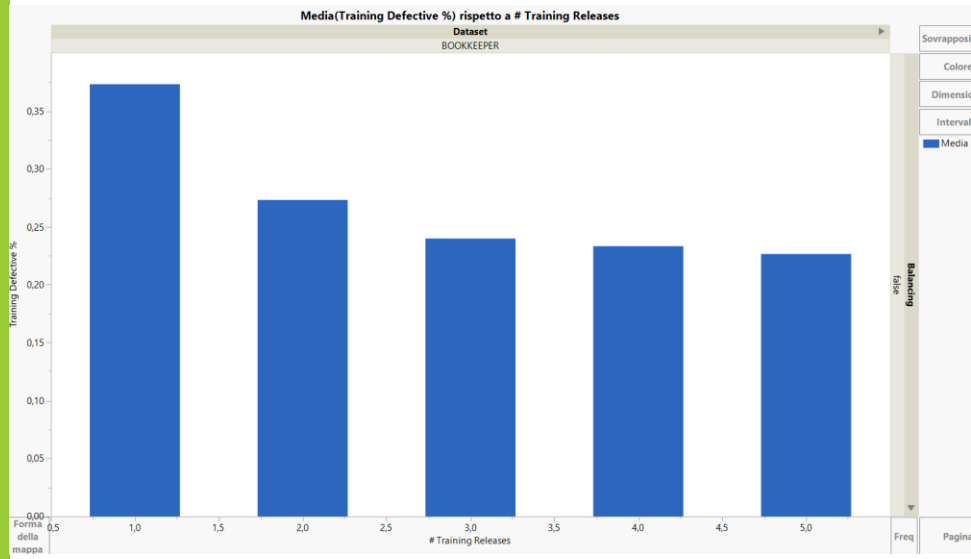
L'applicazione del cost sensitive classifier non permette ulteriori miglioramenti nel dataset. Dai risultati ottenuti, si evince che il dataset Openjpa ottiene migliori prestazioni con l'applicazione della sola tecnica di Sampling con il classificatore RandomForest.

Numero di buggyness all'aumentare del numero di release

- **Il numero di training defective aumenta all'aumentare nel numero di training releases per il caso di Openjpa, in particolare il culmine è in release 11.**
- **La stessa cosa non si può dire per Bookkeeper che presenta in media la stessa % di training defective. Mentre nella prima release c'è una % di training defective superiore alla media (c.a. 37%)**
- **Entrambi i progetti non superano mai una % di training defective superiore al 37%.**



Openjpa



Bookkeeper

Riferimenti

- SonarCloud :

https://sonarcloud.io/dashboard?id=msalvati97_DELIVERABLE_1_ISW2

- Github :

https://github.com/msalvati1997/DELIVERABLE_1_ISW2