Ingegneria del software 2

Progetto A.A. 2019/2020

Machine Learning for Software Engineering

0287567

Paolo Melissari

Table of Contents

1. DELIVERABLE 1

INTRODUZIONE

Lo scopo di questo deliverable è quello di costruire un processo control chart per analizzare la qualità dello sviluppo software, attività necessaria per le aziende con CMMI4 (dove le misure vengono utilizzate per analizzare quantitativamente le performance del progetto) e CMMI5 (dove le misure vengono utilizzate anche per fare Defect Prevention). In questo caso, è stato sviluppato un process control chart per monitorare la presenza di bug nel progetto Mahout al variare del tempo. Per farlo, si è seguito questo workflow:

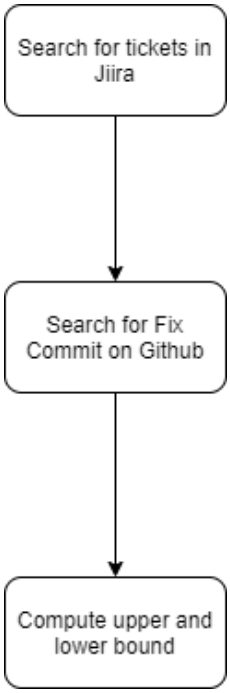


Figura 1Workflow

Per prima cosa quindi, sono stati considerati tutti i ticket che hanno issueType = Bug, che hanno status = closed o status = resolved e che hanno resolution = fixed, ovvero si richiedono le informazioni di tutti quei ticket che sono stati chiusi e che sono associati a dei bug fixati. In Figura 2 è mostrato lo snippet utilizzato per ottenere i dati:

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
       + projName + "%22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR"
       + "%22status%22=%22resolved%22)AND%22resolution%22=%22fixed%22&fields=key,resolutiondate,versions,created&startAt="
       + i.toString() + "&maxResults=" + j.toString();
JSONObject json = readJsonFromUrl(url);
```

Figura 2. Snippet per la lista di ticket

A questo punto, si prendono tutte le commit del progetto tramite le API di Github e si cerca in ognuno di essi la presenza del nome di uno dei ticket (ad es. “Mahout-145”), come mostrato in Figura 3:

```
for (i = 0;  i<total; i++) {
    JSONObject key = jsonArray.getJSONObject(i%1000);
    String commitMessage = key.getJSONObject("commit").get("message").toString();
    //farlo con un metodo
    if(commitMessage.length() < projName.length()+ threshold) {
       break;
    }

    String ticketMessage = null;
    int[] resultArray = findStartEnd(projName,commitMessage);
    int start = resultArray[0];
    int end = resultArray[1];

    String date = key.getJSONObject("commit").getJSONObject("committer").getString("date");

    if (start<end) {

        ticketMessage = commitMessage.substring(start,end); // prendo tutto finchè non trovo ] o :
        String formattedDate = checkEsistence(ticketsID, ticketMessage,date);

        logger.log(Level.INFO, ticketMessage);
        logger.log(Level.INFO, formattedDate);
        logger.log(Level.INFO, "---------------");
        if(formattedDate != null && mapDate.containsKey(formattedDate)) {
          mapDate.put(formattedDate, mapDate.get(formattedDate)+1);
        }
        else if (formattedDate != null) {
          mapDate.put(formattedDate, 1);
        }
```

Figura 3.

Per ogni commit trovata, ne viene considerato il mese e l’anno e viene aumentato di 1 il conteggio dei bug presenti nel progetto in quel mese di quell’anno. Il risultato finale è un file csv, di cui viene mostrato una parte nella figura 4:

2008/04	5
2008/10	1
2008/11	3
2009/04	1
2009/06	1
2009/07	2
2009/09	2
2009/10	6
2009/12	4
2010/01	1
2010/02	7
2010/03	2
2010/04	9
2010/05	2
2010/06	4
2010/07	14
2010/08	19
2010/09	22
2010/10	7
2010/11	2
2010/12	2
2011/01	15
2011/02	5
2011/03	7
2011/04	12

Figura 4.

Per determinare l'upper bound e il lower bound, ci si basa sul fatto che in una distribuzione normale, il 99% dei valori risiede nell'intervallo [-3*Standard Deviation*media, 3*Standard Deviation*media]. Se un processo è maturo, tutti i valori delle misurazioni dovranno risiedere in questo intervallo.

Il process control chart è mostrato in Figura 5:

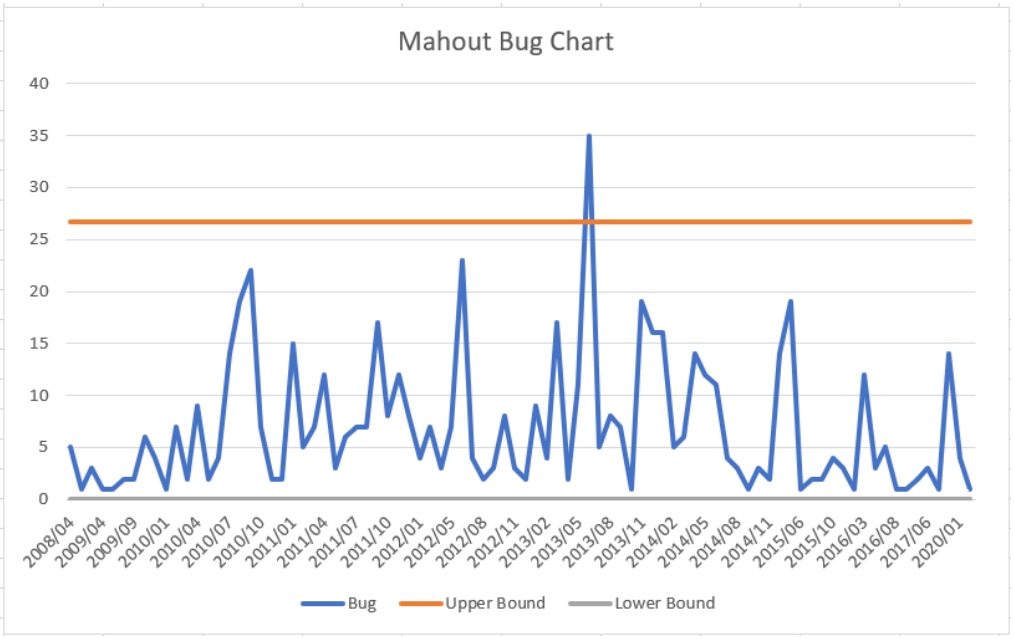


Figura 5. Process Control Chart

Come si può vedere, nel 2013/06 si eccede il valore dell'upper bound. A questo punto, in un'azienda con CMMI5 si parte da questa informazione per capire il motivo del superamento del valore limite e prendere le misure necessarie per prevenire che avvenga.

2. DELIVERABLE 2

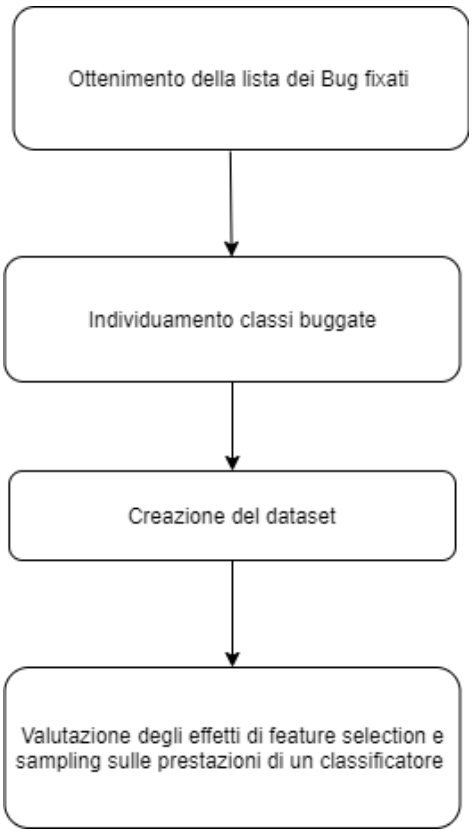
Introduzione

L'obiettivo del deliverable è quello di applicare e misurare gli effetti delle tecniche di sampling e feature selection su dei modelli di classificazione utilizzati nella predizione di bug nel software.

Per tale scopo sono stati scelti i file Java di due progetti Apache, BOOKKEEPER e OPENJPA, sui quali sono stati raccolti dei dati da cui partire per l'analisi dell'efficacia di queste tecniche.

Dato che secondo studi empirici la maggior parte dei bug presenti in un progetto appartiene all'ultimo 20% delle release, è molto probabile che se nel modello si considerano i file appartenenti a queste versioni si avrà un missing rate molto alto (ovvero un'alta percentuale di classi identificate come non difettive, che in realtà non sono difettive). Per questo motivo si è scelto di non considerare l'ultimo 50% delle release di un progetto in modo che, sempre secondo questi studi, si possa avere un missing rate pari a circa l'1%, valore ritenuto accetabile.

Il workflow seguito è il seguente:



Progettazione

Per prima cosa, si è scelto un metodo per definire se una classe fosse definibile come "Buggata" oppure no.

In tal senso è possibile sfruttare Jiira, un sistema di tracking per progetti software che offre la possibilità di visionare lo storico dei bug che sono stati fixati (o che devono ancora essere fixati) associando ad ognuno di essi un ID, la versione in cui quel bug è stato trovato (Opening Version) e se disponibili anche la versione in cui quel bug è stato fixato (Fixed Version) e in quali versioni era presente quel bug (Affected Versions).

Dunque in prima analisi, si potrebbero prendere da Jiira tutte le informazioni relative ad un certo bug fixato (in particolare l'insieme delle Affected Versions) e definire come buggata nelle versioni appartenenti alle Affected Versions ogni classe che appartiene alla fix commit di quel bug,ovvero la lista di file che hanno risolto quel bug. A tal proposito, è stato utilizzata l'API di Jiira che permette di ottenere queste informazioni, come mostrato dal seguente snippet:

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
        + projName + "%22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR"
        + "%22status%22=%22resolved%22)AND%22resolution%22=%22fixed%22&fields=key,resolutiondate,versions,created&startAt="
        + i.toString() + "&maxResults=" + j.toString();
JSONObject json = JiiraUtils.readJsonFromUrl(url);
```

Figura 6. Composizione dell'URL per ottenere le informazioni sui bug

Come mostrato in Figura 6, nell'URL si richiedono tutti i ticket che hanno issueType = Bug, che hanno status = closed o status = resolved e che hanno resolution = fixed, ovvero si richiedono le informazioni di tutti quei ticket che sono stati chiusi e che sono associati a dei bug fixati.

Tuttavia, dato che spesso le informazioni relative alle Affected Versions non sono sempre affidabili (bug con Affected Versions posteriore alla Opening Version) e a volte non sono neanche presenti, utilizzare unicamente il dato delle Affected Versions rischia di farci sottostimare il numero di classi buggate all'interno di un progetto.

Per risolvere questo problema, si è utilizzato il metodo Proportion per stimare le Affected Versions di un certo bug qualora queste non fossero presenti in Jiira. Questo metodo infatti, parte dall'assunzione che la proporzione tra il tempo passato tra la scoperta di un bug e la sua risoluzione è proporzionale al tempo in cui il bug è stato presente nel sistema. In base a questo quindi, è possibile affermare che il tempo tra Injected Version (versione in cui il bug è stato introdotto) e Opening Version (versione in cui il bug è stato trovato) sia proporzionale a quello tra Opening Version e Fixed Version. In questo modo possiamo dire che $IV = FV - (FV - OV) * P$ e quindi potremo stimare che le Affected Versions sono tutte quelle versioni che appartengono all'insieme [IV,FV). Quello che rimane da stabilire è il valore della costante P.

Tra le varie configurazioni possibili per il calcolo di P, quella scelta in questo caso è stata quella incrementale, ovvero quella in cui la costante di proporzione viene calcolata come media tra le costanti di proporzione di tutti i bug trovati fino a quel momento.

I motivi di tale scelta sono:

- Semplicità di implementazione.
- Con il metodo moving window, se non si hanno un buon numero di bug fixati, si rischia di computare P basandosi solo su uno o due bug e questo potrebbe portare ad un valore di P non significativo.
- Con il metodo cold start, si rischia che nel computo di P vengano considerati progetti troppo diversi dal progetto considerato e quindi la stima delle Affected Versions non sia corretta.

In Figura 7 è riportato lo snippet relativo all'applicazione del metodo per calcolare IV come descritto precedentemente:

```
public static String proportion(String fixedVersion, String openingVersion){
    int injectedVersion =  (int) (Double.valueOf(fixedVersion) -((Double.valueOf(fixedVersion)-Double.valueOf(openingVersion))*p));

    if (injectedVersion <= 0)
        injectedVersion = 1;
    return String.valueOf(injectedVersion);
}
```

Figura 7. Applicazione del metodo proportion

Il calcolo di P invece viene fatto attraverso il metodo in Figura 8(richiamato per tutti i ticket di cui è possibile conoscere IV,OV ed FV):

```
public static void computeP(int fixedVersion, int openingVersion, int injectedVersion) {
    Integer numerator = Integer.valueOf(fixedVersion)-Integer.valueOf(injectedVersion);
    Integer denominator = Integer.valueOf(fixedVersion)-Integer.valueOf(openingVersion);
    double newP = 0.0;
    if(denominator != 0) {
        newP = numerator/denominator;
    }
    totalP += newP;
    count++;
    p = totalP/count;

}
```

Figura 8. Calcolo del parametro di proporzione

A questo punto, sono state scelte dei parametri che possono essere correlati alla presenza di un bug e che sono necessari per fare una predizione sulla presenza di un bug. I parametri scelti sono:

Size	LOC
NR	Numero di volte in cui la classe compare in una commit
NFix	Numero di volte in cui la classe compare in una fix-commit
ChgSetSize	Numero di file presenti nelle commit di cui la classe fa parte
MaxChgSetSize	Massimo tra i ChgSetSize
AvgChgSetSize	Media tra i ChgSetSize
Churn	Differenza tra LOC added e LOC deleted nella classe in una certa versione
MaxChurn	Massimo tra i Churn
AvgChurn	Media tra i Churn

A questo punto, è stato creato un dataset che per ogni riga indica il nome del file, la versione di appartenenza, le metriche indicate sopra e se è buggy o no.

FileName	Version Name	#Revision	#FixCommit	Size	Churn	MaxChurn	AvgChurn	ChgSetSize	MaxChgSetSize	AvgChgSetSize	Buggy
bookkeepe	1	0	0	137	0	0	0	0	0	0	yes
bookkeepe	1	8	0	244	496	123	62	44	8	5	yes
bookkeepe	1	80	44	1026	-1048	53	-13	748	26	9	yes
bookkeepe	1	8	4	92	232	53	29	184	26	23	yes
bookkeepe	1	0	0	180	0	0	0	0	0	0	no
bookkeepe	1	48	28	471	8	59	0	392	26	8	yes
bookkeepe	1	16	12	204	204	20	12	108	12	6	yes
bookkeepe	1	32	16	551	-2020	21	-63	312	20	9	yes
bookkeepe	1	20	8	149	-348	30	-17	176	12	8	yes

Figura 9. Il dataset

Il dataset è stato utilizzato per valutare la prestazione di 3 classificatori:

- KNN
- Naive Bayes
- Random Forest

Per ognuno di essi poi, è stato valutato l'impatto sulle prestazioni utilizzando un filtro per fare Feature Selection e utilizzando Oversampling, Undersampling e SMOTE come tecniche di sampling.

Per poter valutare le prestazioni di un classificatore, sono state scelte le seguenti metriche:

- **Precision**: $\frac{TP}{TP+FP}$, indica la percentuale di istanze indicate come positive, che sono veramente positive.
- **Recall**: $\frac{TP}{TP+FN}$, indica la percentuale di istanze positive correttamente individuate
- **F1 measure**: $2*\frac{Precision*Recall}{Precision+Recall}$, media armonica tra precision e recall
- **Area Under ROC Curve (AUC-ROC):** area sottostante la ROC curve, ovvero una curva che ha sull'asse y il True Positive Rate(=recall) $\frac{TP}{TP+FN}$ e sull'asse x il False Positive Rate $\frac{FP}{FP+TN}$, calcolati al variare del classification threshold
- **Area Under Precision Recall Curve (AUPRC):** area sottostante la curva ha sull'asse y la Precision e sull'asse x la Recall, calcolati al variare del classification threshold
- **Kappa**: indica quanto il classificatore è più accurato rispetto ad un classificatore random

L'aggiunta della metrica AUPRC è dovuta al fatto che quest'ultima dà risultati più significativi rispetto la AUC-ROC in tutte le situazioni in cui si vogliono fare predizioni facendo training su un dataset che ha grande disparità tra istanze classificate come positive e istanze classificate come negative. Infatti utilizzando la AUPRC si pone l'attenzione sulla classe minoritaria (grazie all'utilizzo della recall) mentre la AUC-ROC pone uguale attenzione ad entrambe le classi, con il risultato di essere insensibili ad eventuali errori di classificazione della classe minoritaria (bilanciati dal gran numero di classificazioni corrette nella classe maggioritaria). Ad esempio si consideri la situazione di Figura 10, nel quale viene preso un dataset con classi bilanciate e poi viene reso sbilanciato aggiungendo 640 casi nella classe "No Disease":

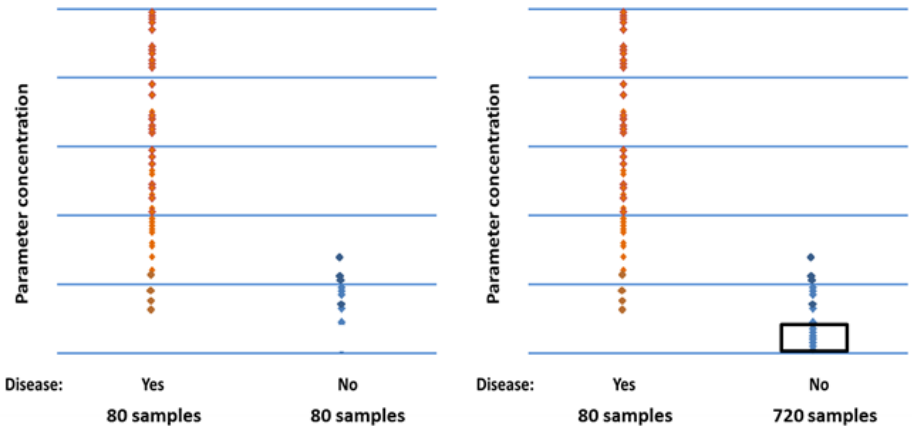


Figura 10 Dataset

Nella Figura 11, viene mostrato il comportamento della ROC Curve con questi due dataset:

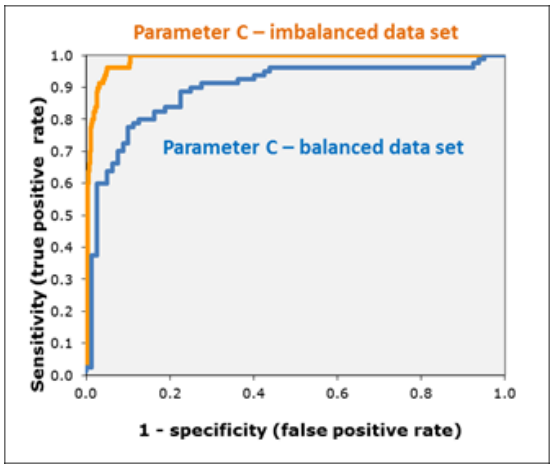


Figura 101. ROC Curve con un dataset sbilanciato

Come è possibile notare, un dataset sbilanciato dà una curva che si avvicina maggiormente al punto (1,0) rispetto alla curva del dataset bilanciato e quindi l'area sotto questa curva è maggiore. Nella Figura 12, viene mostrata la curva Precision-Recall:

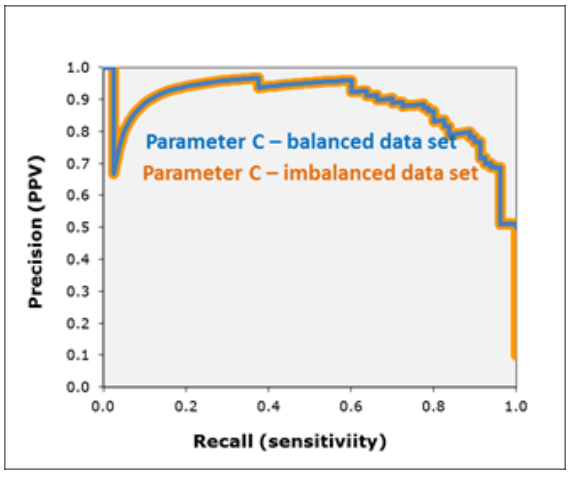


Figura 112. Precision-Recall curve con un dataset sbilanciato

In questo caso le due curve sono praticamente identiche, a parte il punto con recall = 1 nel quale il dataset bilanciato ha valore di precision = 0,5 mentre quello non bilanciato ha valore 0,1. Il risultato è che in entrambi i casi l'area sotto la curva è molto simile.

Credits: https://acutecaretesting.org/en/articles/precision-recall-curves-what-are-they-and-how-are-they-used

Per queste ragioni, durante l'analisi dei risultati verrà dato maggior peso ai valori di AUPRC rispetto a quelli di AUC-ROC.

RISULTATI

In Figura 12 e 13 vengono mostrati i risultati ottenuti per il progetto BOOKKEEPER, nei quali per ogni classificatore vengono confrontate le prestazioni delle tecniche di balancing :

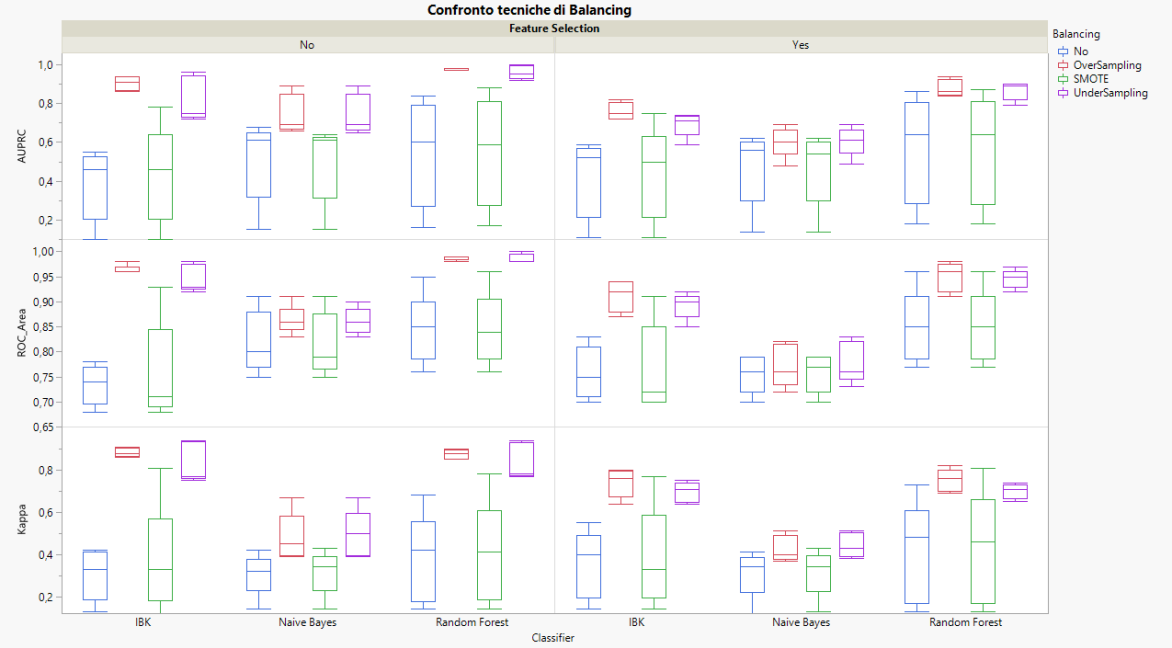


Figura 12

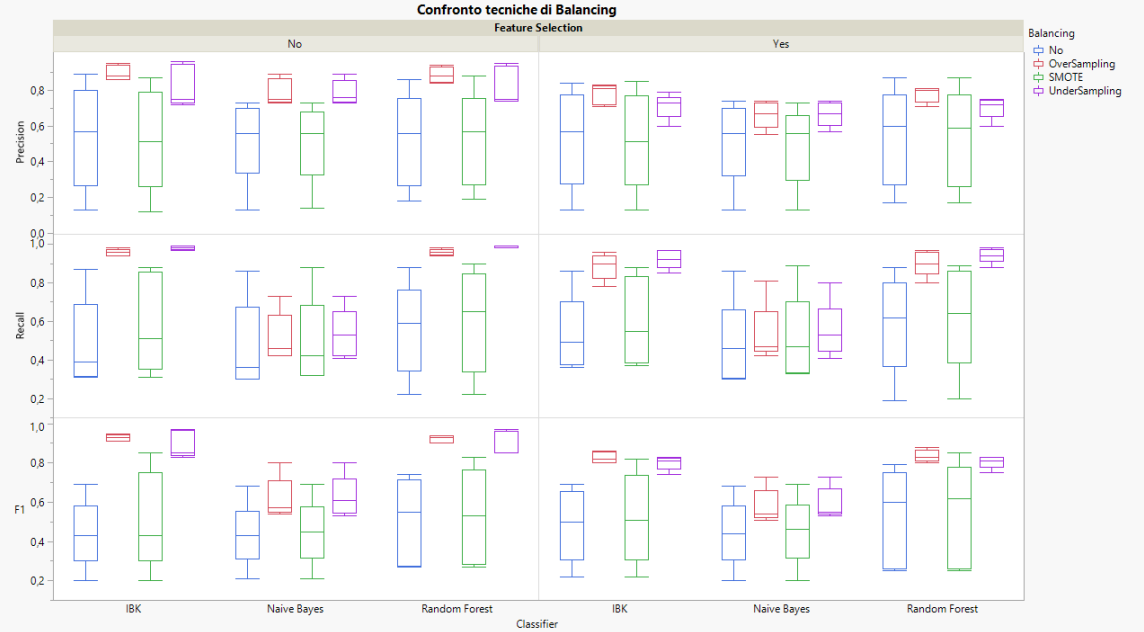


Figura 13

Da questi grafici è possibile fare le seguenti considerazioni:

- In assenza di Feature Selection, le tecniche di OverSampling ed UnderSampling hanno in generale prestazioni migliori su ogni metrica e su ogni classificatore rispetto a SMOTE e No-Sampling.
- In presenza di Feature Selection, OverSampling ha prestazioni leggermente migliori su ogni metrica rispetto ad UnderSampling. SMOTE e No-Sampling continuano ad avere prestazioni inferiori ad OverSampling e Undersampling.
- Utilizzare OverSampling ed UnderSampling, a prescindere dall'utilizzo o meno di tecniche di Feature Selection, porta ad un notevole incremento di prestazioni nei classificatori IBK e Random Forest, mentre per Naïve Bayes l'incremento di prestazioni è significativo solo in assenza di Feature Selection.
- Le prestazioni di Naïve Bayes secondo la metrica AUC-ROC sono simili anche al variare delle tecniche di Balancing

In Figura 14 e 15 vengono mostrati i risultati ottenuti per il progetto BOOKKEEPER, nei quali per ogni classificatore vengono confrontate le prestazioni delle tecniche di Feature Selection :

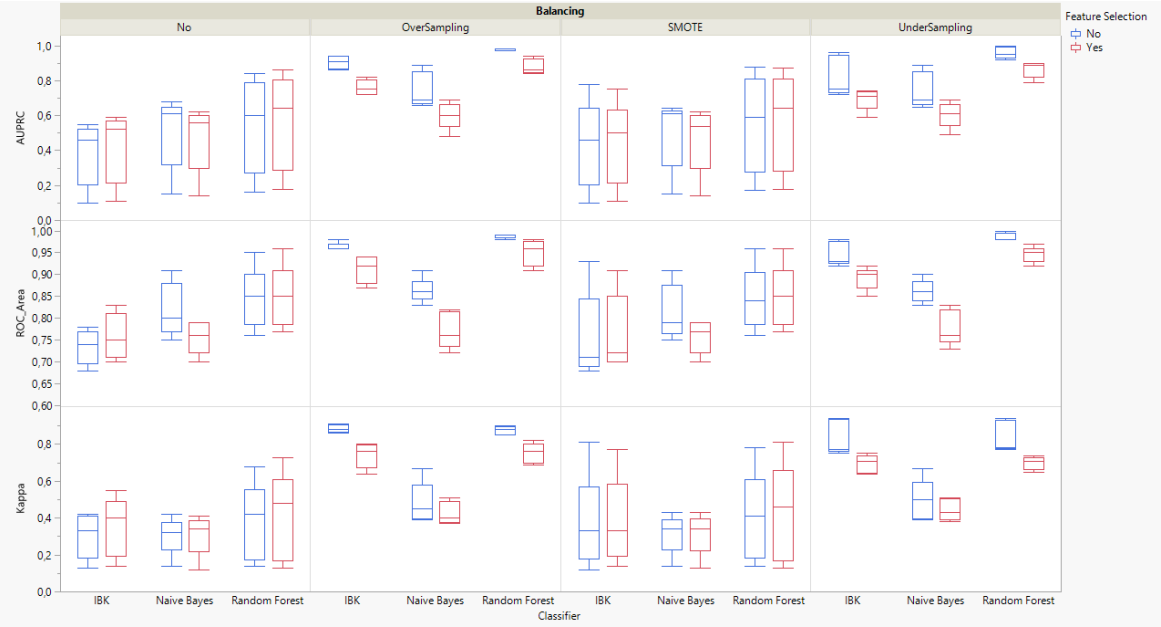


Figura 14

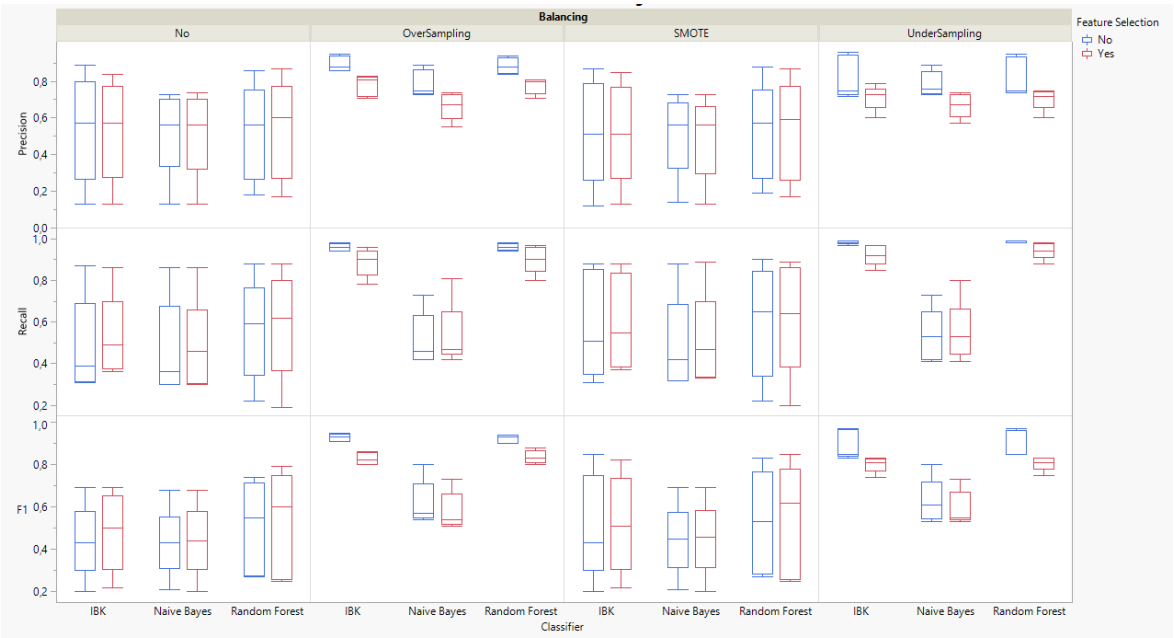


Figura 15

Da questi grafici è possibile fare le seguenti considerazioni:

- Utilizzare tecniche di Feature Selection senza utilizzare contemporaneamente tecniche di Balancing, non porta a miglioramenti delle prestazioni per nessun classificatore.
- Per i classificatori IBK e Random Forest, l'utilizzo di OverSampling è più efficace senza l'utilizzo di tecniche di feature selection.
- Per il classificatore Naïve Bayes l'utilizzo di tecniche di Feature Selection unito ad OverSampling porta a miglioramenti di prestazione secondo le metriche F1, Recall e Kappa, in quanto portano risultati simili rispetto al non utilizzo di Feature Selection, ma con minore varianza. Per le altre metriche, le prestazioni sono peggiori rispetto al non di Feature Selection.
- L'utilizzo di Feature Selection combinato a SMOTE, non comporta nessun miglioramento significativo per nessuna metrica e per nessun classificatore rispetto ad utilizzare SMOTE senza Feature Selection
- L'utilizzo di Feature Selection combinato ad UnderSampling , non comporta nessun miglioramento significativo per nessuna metrica e per nessun classificatore rispetto ad utilizzare UnderSampling senza Feature Selection

A seguito di queste considerazioni, si è quindi giunti a questi risultati per BOOKKEEPER:

- Per il classificatore IBK, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è OverSampling + No Feature Selection.
- Per il classificatore Naïve Bayes, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è UnderSampling + No Feature Selection.
- Per il classificatore Random Forest, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è OverSampling + No Feature Selection. La scelta di questa combinazione rispetto a UnderSampling + No Feature Selection è dovuta al fatto che quest'ultima ha portato risultati peggiori per la metrica AUPRC, mentre per tutte le altre metriche i risultati sono sostanzialmente identici e, come spiegato nella sezione precedente, si è preferito dare maggiore importanza alle configurazioni che portano ad un aumento della AUPRC.

In figura 16 e 17 vengono mostrati i risultati ottenuti per il progetto OPENJPA, nei quali per ogni classificatore vengono confrontate le prestazioni delle tecniche di balancing :

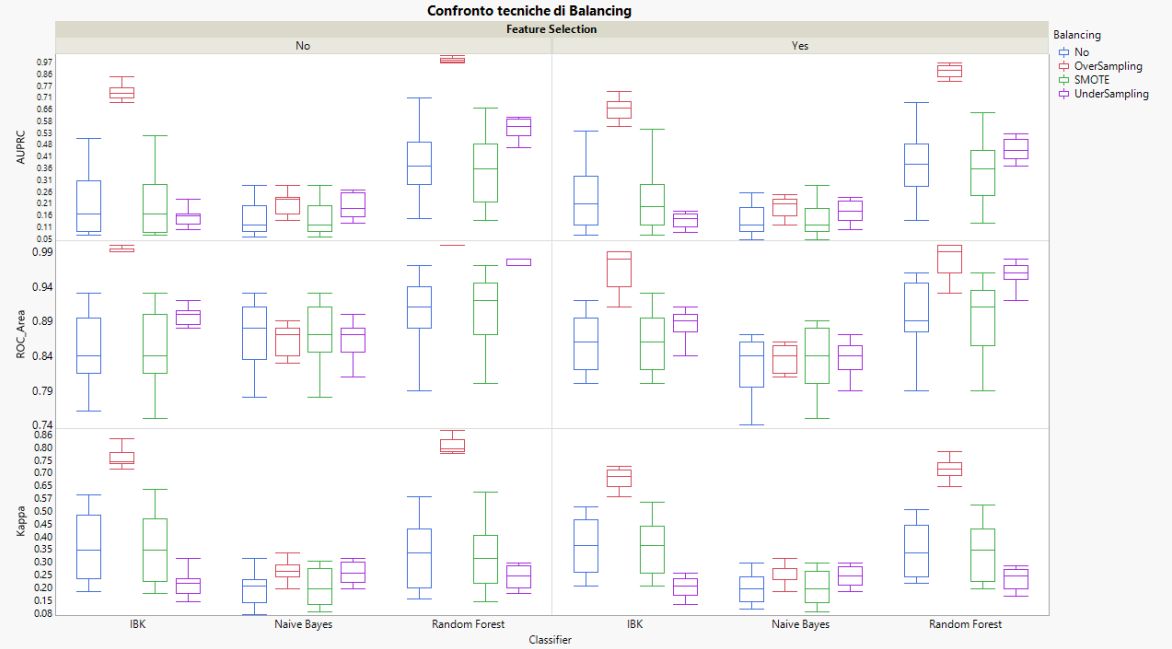


Figura 16

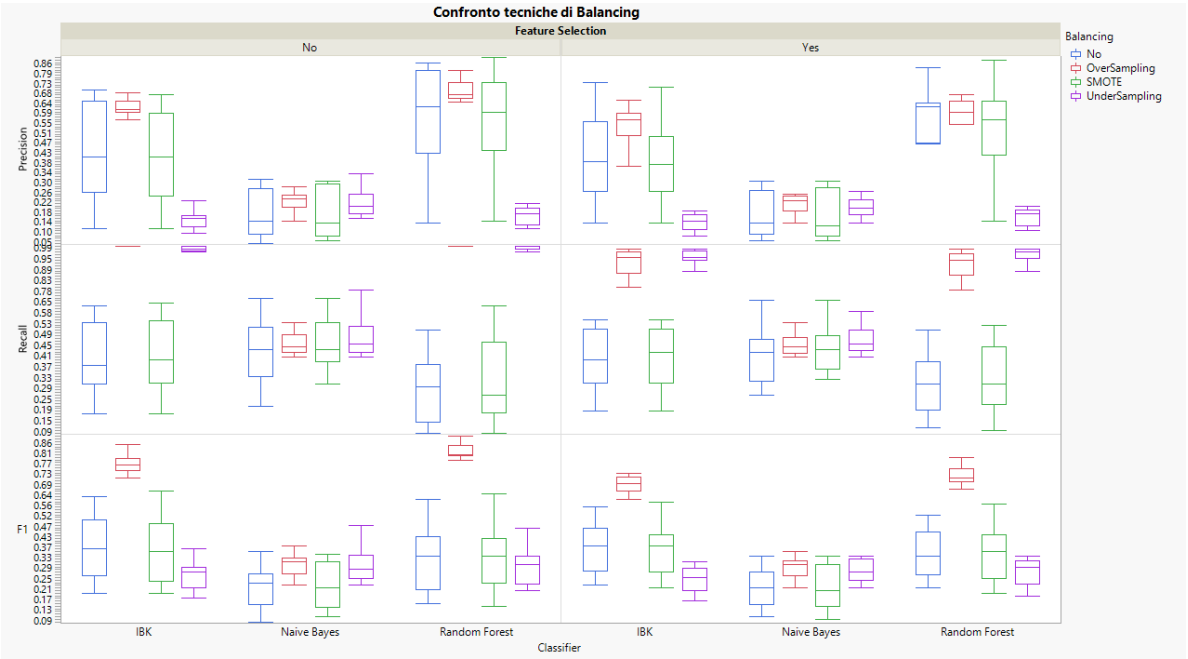


Figura 17

Da questi grafici è possible fare le seguenti considerazioni:

- In assenza di tecniche di Feature Selection, l'utilizzo di OverSampling comporta ad un grande miglioramento di prestazioni secondo tutte le metriche per i classificatori IBK e Random Forest.
- In assenza di tecniche di Feature Selection, l'utilizzo di tecniche di Balancing (in particolare OverSampling ed UnderSampling) porta ad un buon miglioramento secondo tutte le metriche per il classificatore Naïve Bayes.
- In presenza di tecniche di Feature Selection, l'utilizzo di OverSampling per i tutti e 3 i classificatori porta ad un miglioramento delle prestazioni che è però più lieve rispetto al non utilizzo di Feature Selection.
- In tutti i casi, l'utilizzo di SMOTE non ha portato nessun miglioramento significativo rispetto al non utilizzare alcuna tecnica di Sampling.

In Figura 18 e 19 vengono mostrati i risultati ottenuti per il progetto OPENJPA, nei quali per ogni classificatore vengono confrontate le prestazioni delle tecniche di Feature Selection :

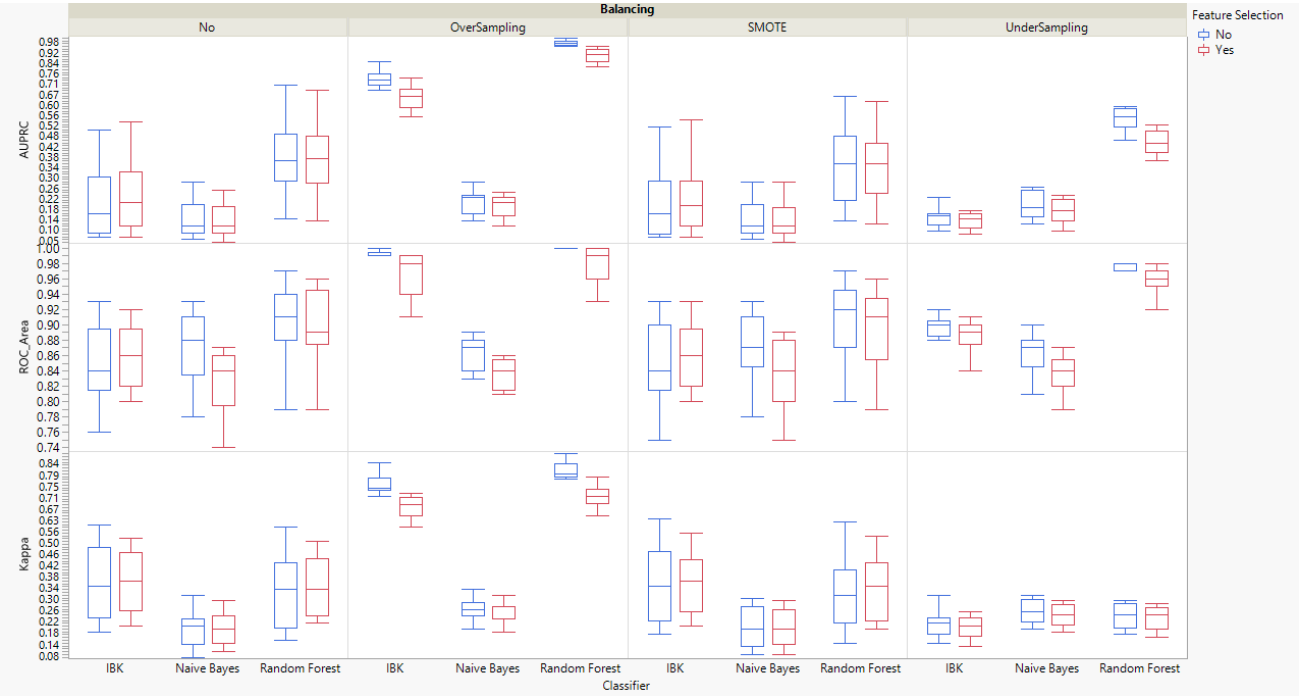


Figura 18

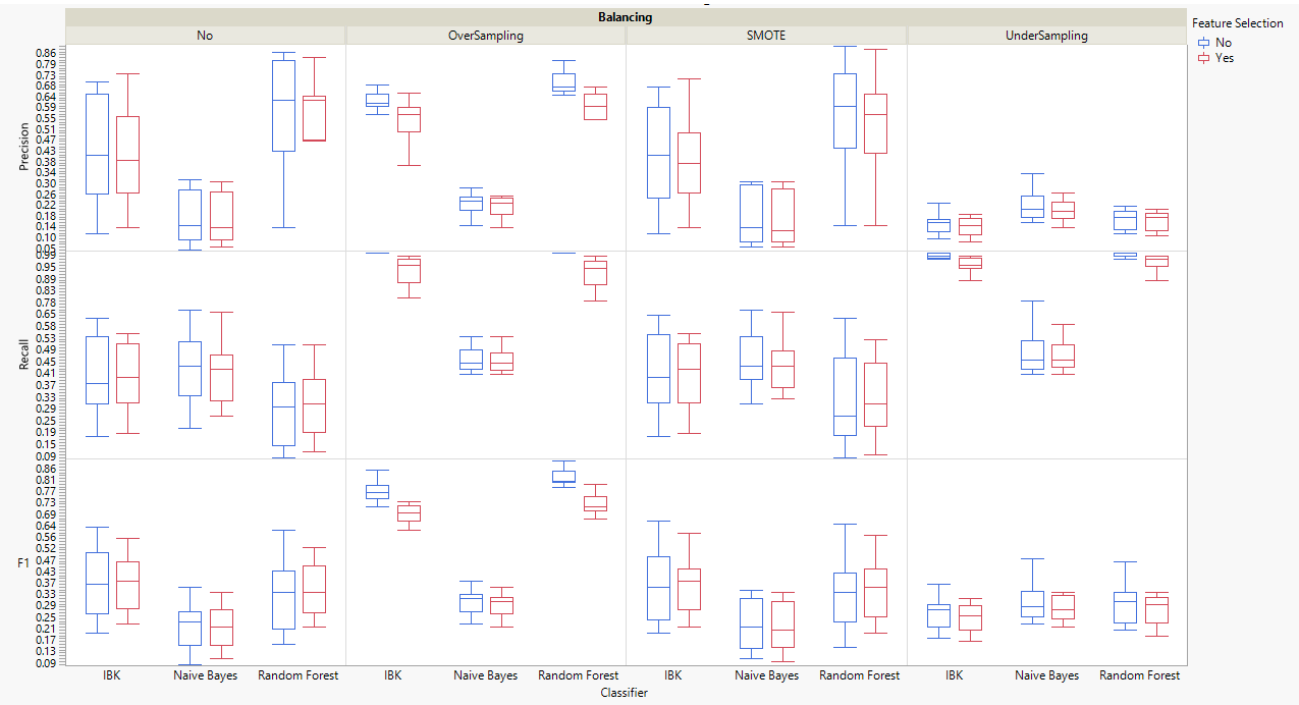


Figura 19

Da questi grafici è possible fare le seguenti considerazioni:

- Utilizzare tecniche di Feature Selection senza utilizzare contemporaneamente tecniche di Balancing, non fa variare di molto le prestazioni dei classificatori.
- Per i classificatori IBK e Random Forest, l'utilizzo di OverSampling è più efficace senza l'utilizzo di tecniche di feature selection.
- Per il classificatore Naïve Bayes l'utilizzo di tecniche di Feature Selection unito ad OverSampling , non fa variare di molto le prestazioni.
- L'utilizzo di Feature Selection combinato a SMOTE, non comporta nessun miglioramento significativo per nessuna metrica e per nessun classificatore rispetto ad utilizzare SMOTE senza Feature Selection
- L'utilizzo di Feature Selection combinato ad UnderSampling , non comporta nessun miglioramento significativo per nessuna metrica e per nessun classificatore rispetto ad utilizzare UnderSampling senza Feature Selection

A seguito di queste considerazioni, si è quindi giunti a questi risultati per OPENJPA:

- Per il classificatore IBK, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è OverSampling + No Feature Selection.
- Per il classificatore Naïve Bayes, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è OverSampling + No Feature Selection. Anche in questo caso, nonostante utilizzare

UnderSampling + No Feature Selection abbia portato a risultati simili rispetto ad OverSampling + No Feature Selection, si è preferita quest'ultima poichè ha dato risultati migliori secondo la metrica AUPRC.

- Per il classificatore Random Forest, la combinazione migliore di tecniche di Feature Selection e tecniche Balancing è OverSampling + No Feature Selection.

A conferma del fatto che la metrica AUC-ROC può portare a considerazioni sulla bontà di un classificatore fuorvianti rispetto ad AUPRC, vengono riportati confronti tra i valori medi assunti tra le due metriche nei progetti BOOKKEEPER (figura 20) e OPENJPA (figura 21):

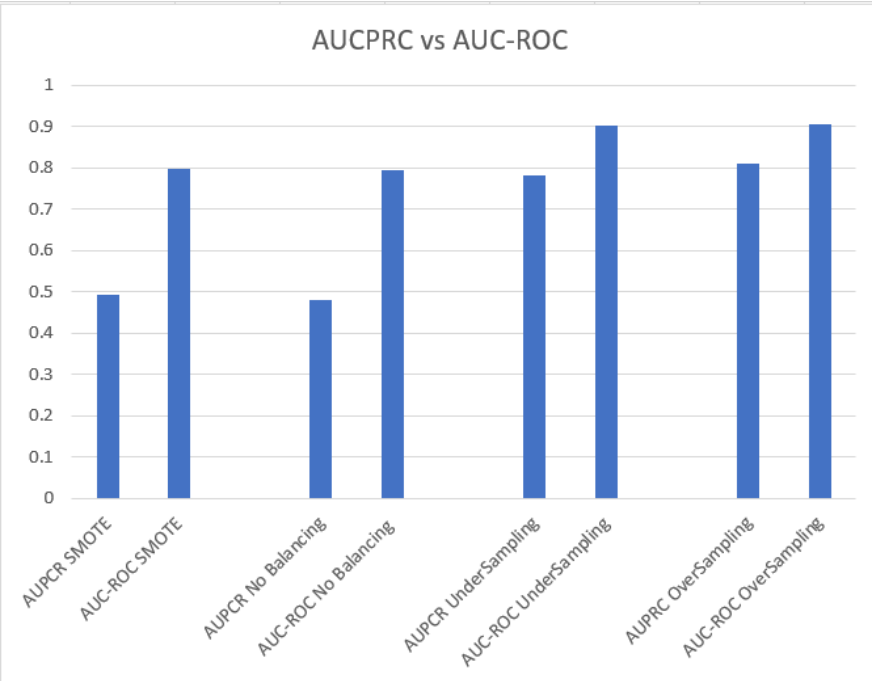


Figura 20.Confronto Bookkeeper

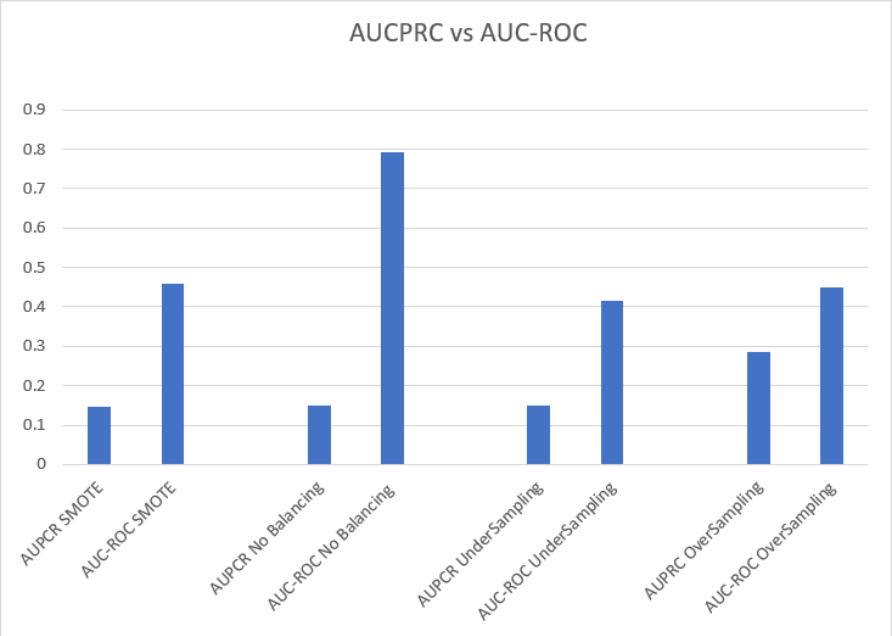


Figura 21. Confronto OpenJPA

Come si può notare, in situazioni in cui il training set è sbilanciato, il classificatore ha valori medi di AUC-ROC che sono superiori ai valori medi di AUPRC (a volte anche di molto).

Per questo motivo dunque, nei casi in cui più di una combinazione tra tecniche di sampling e tecniche di balancing davano risultati simili su tutte le altre metriche, si è deciso di scegliere quella con AUPRC maggiore.