# ReportISW2 - Modulo ML for Software Engineering

2019-2020

Lisa Trombetti 0290810

# SOMMARIO

# 1) DELIVERABLE 1

## 1. INTRODUZIONE

Lo scopo della prima deliverable è quello di analizzare i ticket di progetti open source della Apache Software Foundation per misurarne la stabilità del processo software. In particolare, in questa sezione andremo ad analizzare i ticket di tipo 'new features' del progetto TAJO. Tutto il codice a cui si farà riferimento può essere trovato al link 1.a. della sezione Links.

## 2. PROGETTAZIONE

Per iniziare è stato affrontato il problema di come recuperare le informazioni relative ai ticket ed ai commit ad essi associati. Per i primi è stata utilizzata l'api di Jira vista a lezione modificando opportunamente l'url della richiesta come segue:

*Type == "New Feature" AND (status == "Closed" OR status == "Resolved") AND Resolution == "Fixed"*

Per i commit invece è stata utilizzata l'api messa a disposizione da Github, la quale tuttavia necessita di un token di autorizzazione per essere utilizzata, in quanto senza di esso le limitazioni sul numero di richieste che è possibile effettuare in un ora risultano essere troppo stringenti.

Il progetto è stato strutturato per analizzare non solo i ticket di tipo 'new features', è infatti possibile specificare un valore diverso (es. 'Bug', 'Task', …) all'interno del file config.json. E' inoltre possibile modificare il nome dell'autore, il nome del progetto ed il token di Github.

Per associare i commit ai rispettivi ticket si è deciso di cercare all'interno del messaggio un id della forma 'TAJO-xxxx'. I formati più usati per riferirsi ad un ticket all'interno dei commit di TAJO sono riportati di seguito:

- TAJO-xxxx + ':'
- TAJO-xxxx + ']'
- TAJO-xxxx + ' '

Ad ogni ticket è stata inoltre associata una data di completamento calcolata prendendo in considerazione quella del commit più recente che lo riferisce.

Sono stati quindi generati due file formato csv contenenti le info ricavate nel modo sopracitato. Le informazioni dei ticket comprendono anche il numero di commit ad essi associati in modo da poter capire se gli sviluppatori di TAJO si siano attenuti o meno allo standard 1 commit per 1 ticket visto a lezione.

## 3. ANALISI

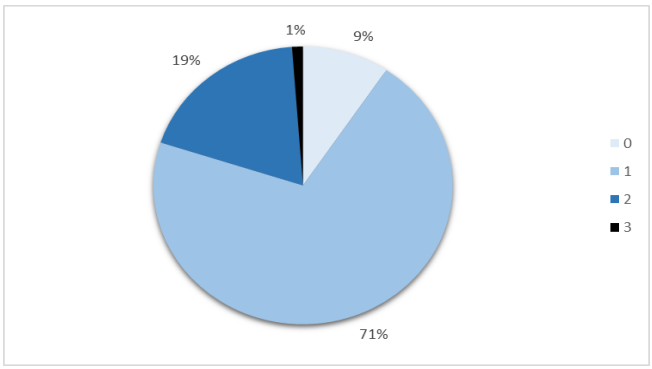


*Figura 1 - TAJO Commits per ticket*

Dalla *figura 1* possiamo vedere che nel 71% dei casi un solo commit è stato associato al ticket, come vorremmo che sia, ma esiste comunque una piccola percentuale di ticket, il 9%, a cui non è stato associato alcun commit.

Andiamo ora a vedere come il numero di ticket di tipo 'new features' si è evoluto nell'arco di tempo che va dall'anno 2011 a quello corrente 2020.

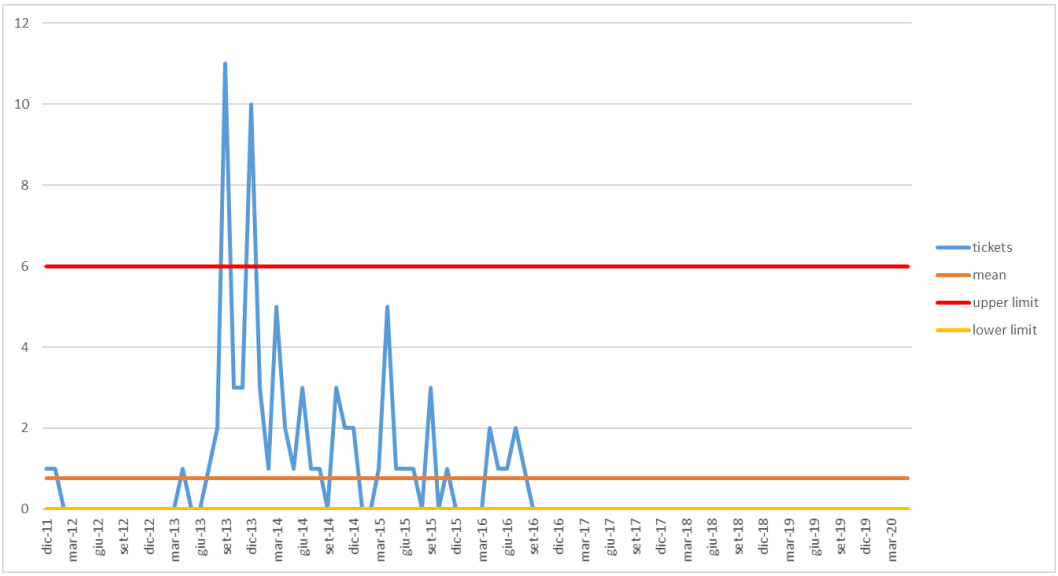


*Figura 2 – TAJO New Features*

Come possiamo vedere dalla *figura 2* la maggioranza dei commit relativi alle 'new features' si concentrano fra l'anno 2013 e 2016. A questo periodo corrispondono le release del sistema che vanno dalla prima Tajo 0.2.0 del 11-20-2013 all' ultima Tajo 0.11.3 del 18-05-2016.

Le linee rossa e gialla rappresentano i valori di upper e lower limit calcolati utilizzando le formule:

*Upper limit = MEAN + 3*STDDEV*

*Lower limit = MEAN - 3*STDDEV*

Nel calcolo del lower limit il valore ottenuto è -4,474 e si è quindi deciso di riportarlo a 0 poiché non avrebbe senso considerare un numero di ticket negativo.

Notando inoltre che l'attività del progetto sembra essersi notevolmente ridotta dopo l'anno 2016, si è deciso di provare a considerare solo il periodo che va dal dicembre 2011 al dicembre 2016. Il risultato è possibile osservarlo in *figura 3*.

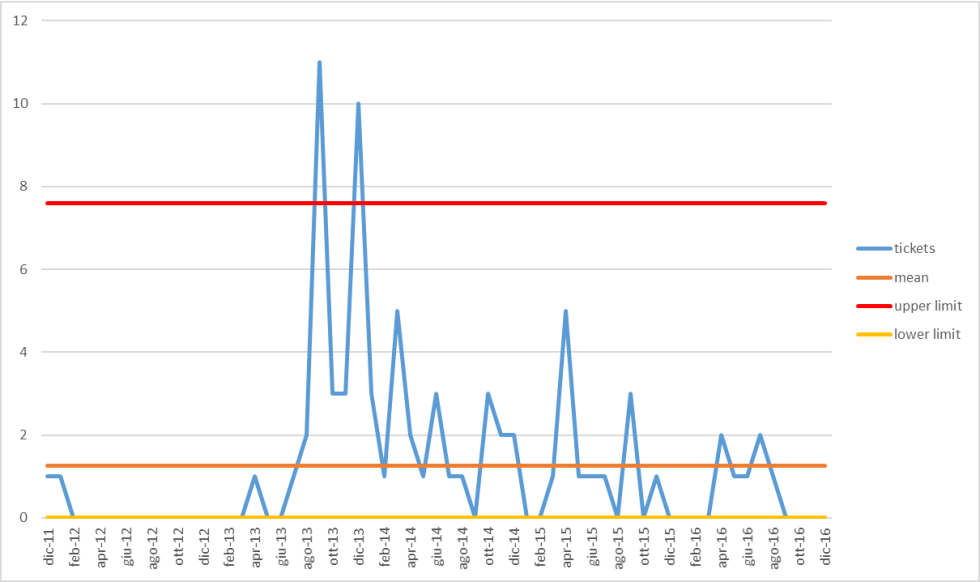


*Figura 3 - TAJO New Features (2011-2016)*

## 4. CONCLUSIONI

Possiamo quindi dire che **nella maggior parte dei casi gli autori si sono attenuti allo standard di un commit per ticket**. Inoltre, è anche possibile che la percentuale di ticket senza alcun commit sia minore rispetto a quella riscontrata poiché, come descritto nelle sezioni precedenti del report, questi sono stati individuati cercando nel messaggio "TAJO-xxxx" ma alcuni autori potrebbero aver dimenticato di riferirsi al ticket o utilizzato un formato diverso.

Per quanto riguarda invece le 'new features' possiamo notare che i due **picchi maggiori che eccedono il limite superiore corrispondono al periodo della prima release di sistema**, dove possiamo supporre che il numero di feature da introdurre fosse più alto. Infatti, già dalla seconda release nel marzo 2014 i valori risultano essere all'interno delle linee di controllo e vanno gradualmente diminuendo, fino all'ultima release di marzo 2016.

# 2) DELIVERABLE 2

## 1. INTRODUZIONE

Per la seconda deliverable del progetto era richiesto di eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di sampling e feature selection sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni opensource. In questo caso i due progetti presi in considerazione sono Bookkeeper e Storm della Apache Software Foundation. Il codice di riferimento può essere trovato nei link 1.a. e 1.b. della sezione Links.

## 2. PROGETTAZIONE

Come per la prima deliverable sono state utilizzate le informazioni sui ticket ed i commit dei due progetti, a cui vengono aggiunte quelle relative alle release e ai file associati ad ogni commit. Per ognuno dei progetti sono state considerate solo le prime metà delle release e le relative informazioni sono state acquisite tramite l'api di Jira.

Una volta raccolti i dati, utilizzando lo stesso procedimento proposto nella deliverable1, si è proceduto andando ad analizzare i ticket in base a:

- Fixed version assente: il ticket non verrà considerato
- Fixed version presente ma nessuna affected version: uso di proportion per calcolare le affected versions
- Fixed e Affected versions presenti: nessun cambiamento necessario

Il calcolo delle affected versions utilizzando proportion si basa sull'idea che ci deve essere un numero proporzionale di versioni necessarie all'individuazione di un difetto e alla sua successiva risoluzione. A questo scopo sono state usate le formule viste a lezione:

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

IV = FV - (FV - OV) * P

Per il calcolo del coefficiente P si è scelto di utilizzare una finestra scorrevole che tenesse conto della media sull'ultimo 1% di difetti risolti.

Dopo aver scartato i ticket sprovvisti di fixed version ed utilizzato proportion laddove necessario, sono stati considerati i singoli commit, andando a cercare le informazioni relative ai file modificati tramite l'api di Github. Tutti i file con estensione diversa da 'java' non sono stati considerati nelle fasi successive del progetto.

Per la creazione del dataset sono state scelte le seguenti nove metriche fra quelle viste a lezione:

- Size : linee di codice (loc)
- Loc touched : somma sulle revisioni dei loc aggiunti/eliminati
- Loc added : somma sulle revisioni dei loc aggiunti
- Max loc added : massimo numero di loc aggiunti
- Average loc added : media sul numero di loc aggiunti
- Number of authors : numero di autori
- Number of fix : numero di bug risolti
- Number of revisions : numero di revisioni
- Change set size : numero di file nello stesso commit

Una classe Record è stata creata per mantenere le informazioni sulle metriche, sul numero della release e sulla 'bugginess' del singolo file. Per verificare se un file sia o meno difettoso in una specifica release si è andati a considerare le affected versions del ticket associato al commit.

Dopo aver analizzato ogni file nelle release considerate, le informazioni presenti nei record sono state riportate su un file csv della forma "NOMEPROGETTOdataset.csv".

Per l'implementazione delle tecniche di sampling e feature selection sono state utilizzate le funzionalità di Weka tramite le api offerte da java. In particolare, sono state prese in considerazione:

- Evaluation technique:
  - Walk forward
- Feature selection:
  - No selection
  - Best first
- Balancing:
  - No sampling
  - Oversampling
  - Undersampling
  - SMOTE
- Classificatori:
  - RandomForest
  - NaiveBayes
  - Ibk

Utilizzando le informazioni presenti nel file csv del dataset sono stati creati due file di tipo arff, uno per il training ed uno per il testing, su cui sono state applicate tutte le possibili combinazioni di classificatori, tecniche di balancing e di feature selection. I risultati finali è possibile osservarli nel file "NOMEPROGETTO.csv" al link 1.b. della sezione Links.

## 3. ANALISI

Andiamo ora ad analizzare i risultati ottenuti per entrambi i progetti, BookKeeper e Storm, valutando gli effetti dell'uso di diverse tecniche di sampling e di feature selection.

### 3.1 BookKeeper

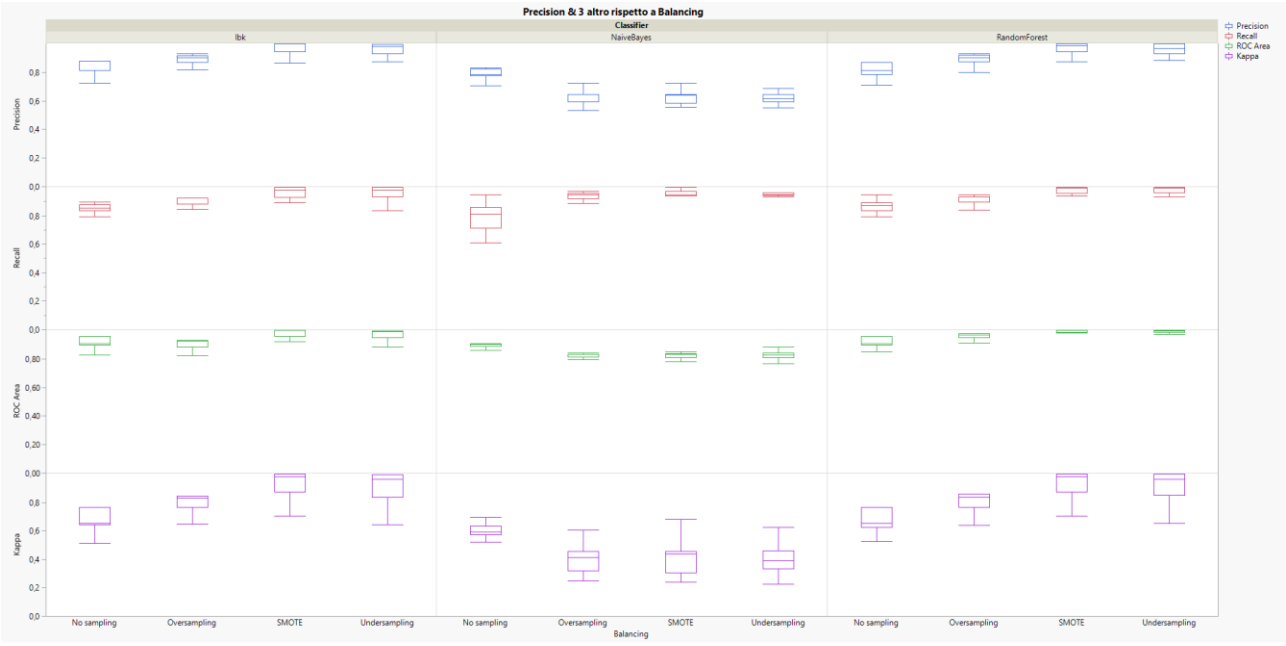


*Figura 4 - Differenze nelle tecniche di balancing in BookKeeper*

Come possiamo vedere dalla *figura 4*, le tecniche di sampling SMOTE e Undersampling sembrano prevalere rispetto alla altre per i classificatori Ibk e RandomForest, mentre nel caso di NaiveBayes nessuna delle tre tecniche considerate sembra andare a migliorare i valori ottenuti.

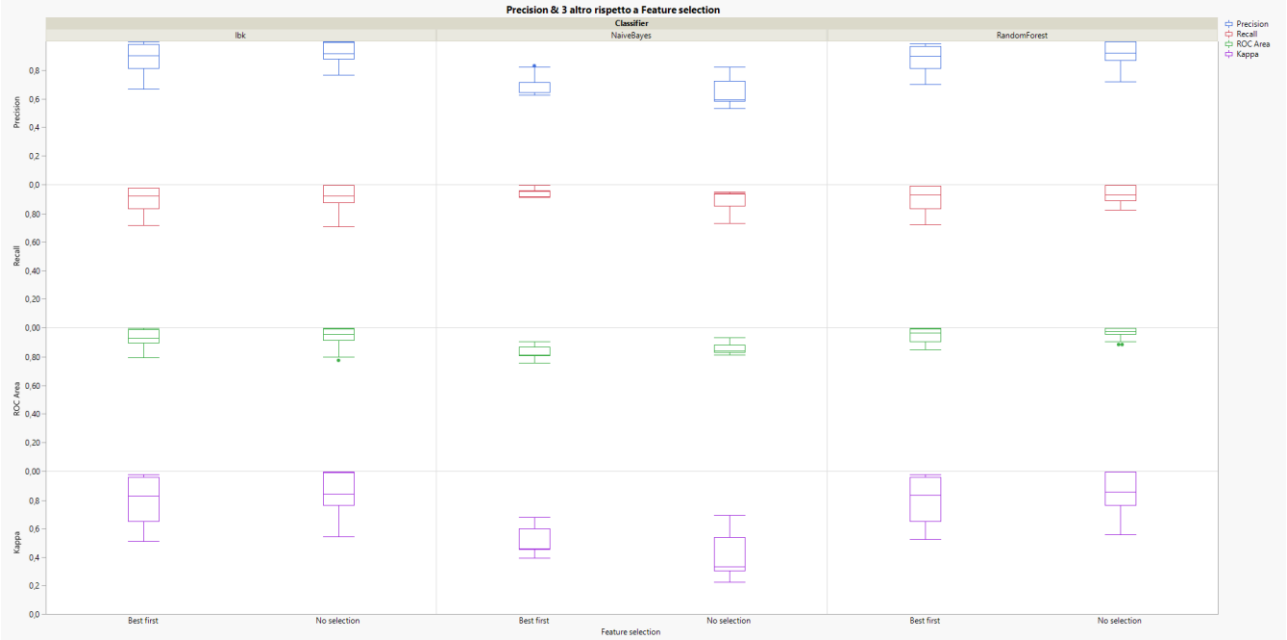


*Figura 5 - Differenze nelle tecniche di feature selection in BookKeeper*

Per quanto riguarda le tecniche di feature selection, possiamo vedere dalla *figura 5* che la tecnica Best first sembra essere utile nel caso di classificatore NaiveBayes, mentre non porta a risultati migliori rispetto alla no selection per gli altri tipi di classificatori.

### 3.2 Storm

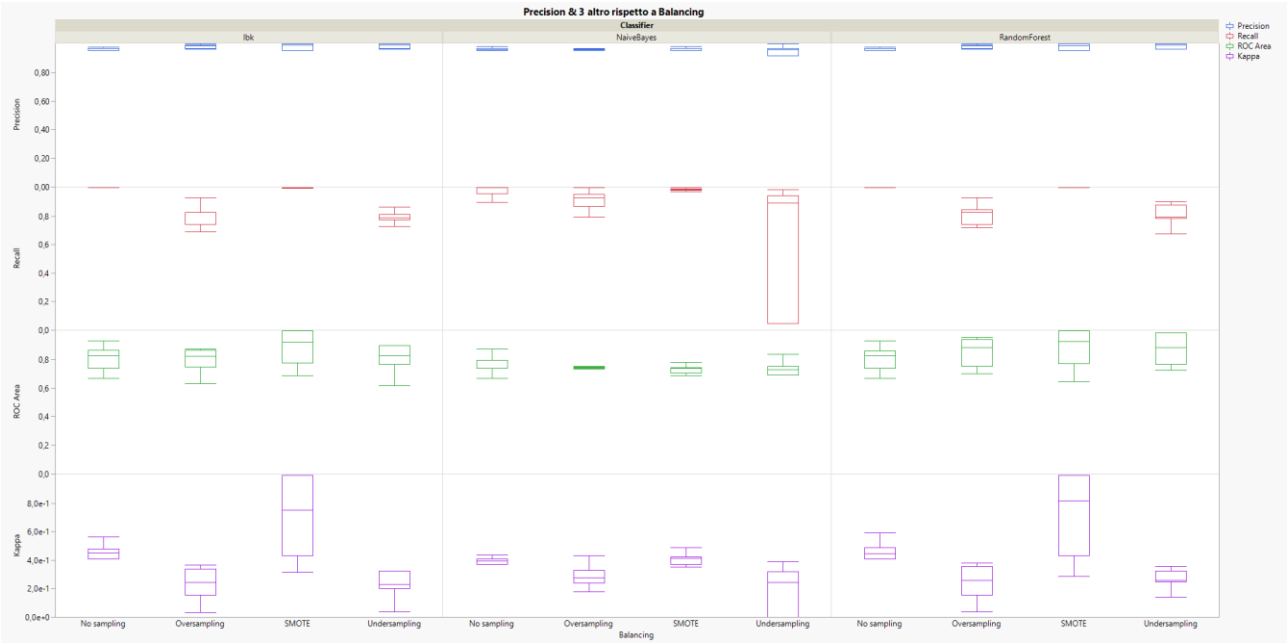


*Figura 6 - Differenze nelle tecniche di balancing in Storm*

Come possiamo notare dalla *figura 6*, in questo caso non ci sembra essere una migliore alternativa fra le tecniche di balancing per quanto riguarda la precision, mentre negli altri casi SMOTE presenta nuovamente i valori più alti.

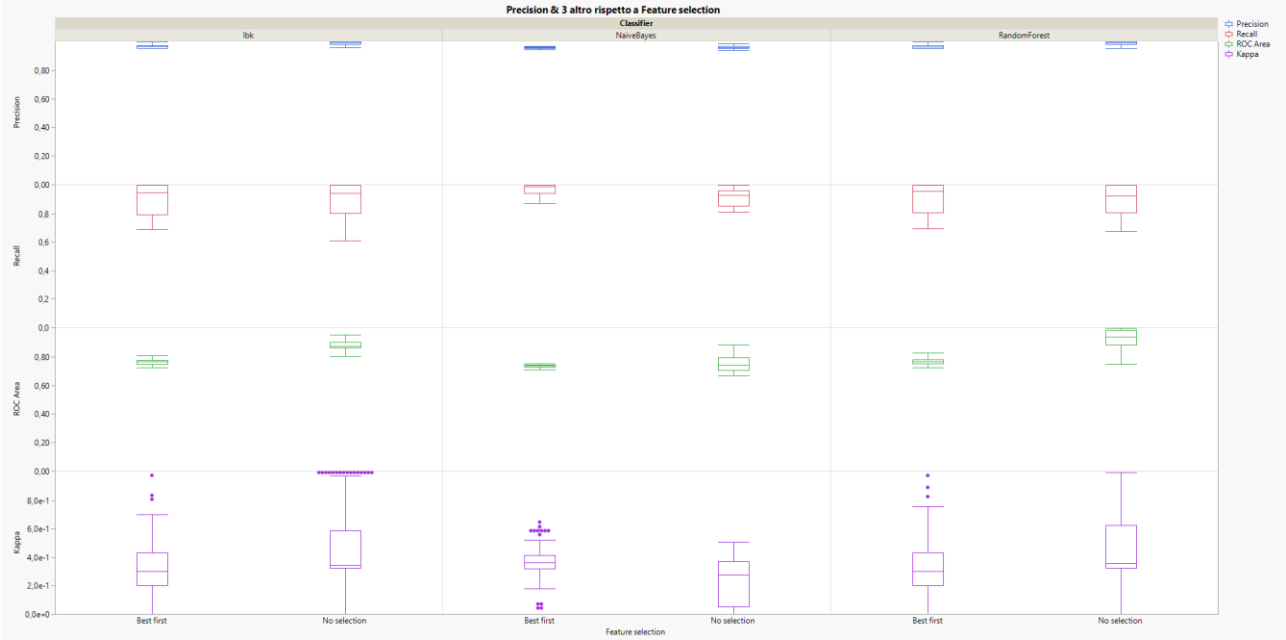


*Figura 7 - Differenze nelle tecniche di feature selection in Strom*

Anche in questo caso l'utilizzo della tecnica di feature selection Best first non ha portato a grandi miglioramenti rispetto al no selection. Da notare è la presenza di outliers per kappa, i cui valori ottenuti sono molto più bassi rispetto a quelli di BookKeeper.

### 4. CONCLUSIONI

In conclusione, possiamo dire che, nonostante i progetti considerati in questa seconda deliverable fossero molto diversi fra loro, in entrambi i casi **SMOTE è risultata essere la migliore fra le tecniche di balancing**. Per quanto riguarda invece la feature selection si è visto che l'introduzione della tecnica **Best first non porta a grandi miglioramenti** nell'accuratezza dei classificatori considerati.

## 3) LINKS

1. Github
    a. https://github.com/Lisa9601/ProgettoISW2
    b. https://github.com/Lisa9601/ProgettoISW2-weka
2. Travis
    a. https://travis-ci.org/github/Lisa9601/ProgettoISW2
    b. https://travis-ci.org/github/Lisa9601/ProgettoISW2-weka
3. SonarCloud
    a. https://sonarcloud.io/dashboard?id=lisa9601_ProgettoISW2
    b. https://sonarcloud.io/dashboard?id=lisa9601_ProgettoISW2-weka