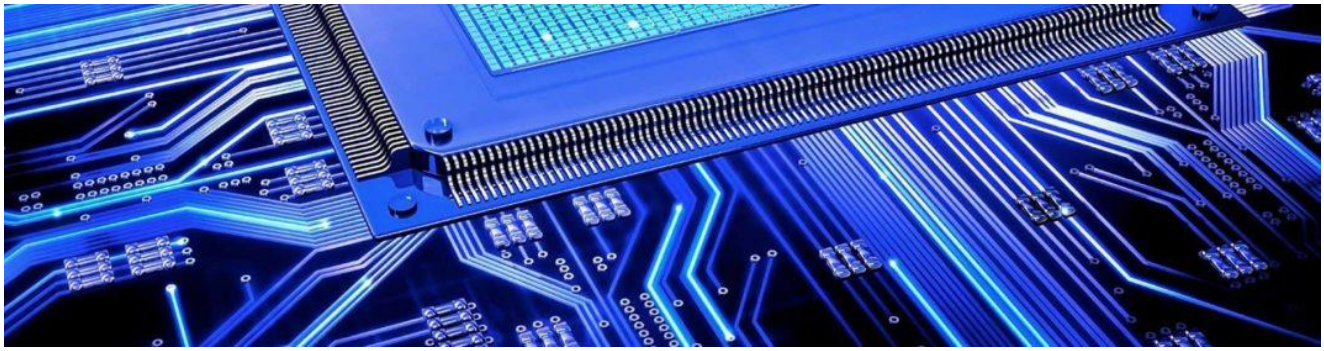
Relazione Progetto ISW2

Process Control Chart & Analisi Classificatori



Università di Roma TorVergata

Tummolo Gabriele 0283629

Il presente documento descrive il lavoro fatto per il completamento delle Milestone assegnate dal prof. Falessi D. durante il corso di Ingegneria del Software 2 presso l'Università degli Studi di Roma Tor Vergata, anno accademico 2019/2020.

Il progetto consiste in due deliverables principali. La deriverable 1 è composta da un'unica Milestone, mentre la seconda di compone in tre Milestone.

Deliverable One

Questa prima deliverable consiste nell'identificare ed analizzare la stabilità di un attributo del progetto "Apache Tajo" in particolare identificando lo sviluppo di New Features, quindi misurando il numero di features rilasciate in ogni mese nel periodo di attività del progetto, identificando i periodi di maggiore o minore produttività.

Queste informazioni sono state prese ed elaborate tramite codice Java:

- La repository in Jira contenente i ticker del progetto Apache Tajo (https://issues.apache.org/jira/projects/TAJO/issues)
- La repository del progetto Apache Tajo su github (http://github.com/apache/tajo)

Ottenere i dati

Il recupero i dati dalle repository è stato fatto attraverso REST API. Tramite query specifiche i dati necessari sono stati restituiti in formato JSON, che dopo averne studiato la struttura si è proceduto all'estrazione di tutte le informazioni necessarie per poter portare a termine questo primo passaggio. In primo luogo sono state

scaricate le informazioni inerenti alla prima repository. In Figura 1 è rappresentata la query specifica fatta che ha restituito le informazioni inerenti ai ticket New feature con lo stato “Resolution=fixed”.

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
        + projName + "%22AND%22issueType%22=%22New%20Feature%22AND%22resolution%22=%22fixed%22&fields=key,resolutiondate,versions,created&startAt="
        + i.toString() + "&maxResults=" + j.toString();
```

Figura 1: Query per ottenere informazioni da Jira

Ottenuto il file JSON , per ogni elemento è stato selezionato il campo key. Successivamente si è proceduto a scaricare le informazioni inerenti alla repository 2.

```
String url1 = "https://api.github.com/repos/apache/tajo/commits?page=";
String url2 = "&per_page=100";
```

Figura 2 Query per ottenere i commit da Github

Quindi per poter individuare il numero di commit effettuati in un determinato periodo (ANNO-MESE) sono stati analizzati in particolare i campi date (che indica la data del commit) e il campo message per individuare se presente il Ticket specifico [NomeTicket].

Elaborazione dati

Ottenuti tutti i dati (ticket e informazioni di Github) queste sono state incrociate per poter calcolare il numero di commit effettuati in un determinato periodo (Anno-mese); in particolare dopo aver salvato le informazioni di Jira in un ArrayList<Object> si è proceduto individuando nel JSON di Github la presenza di relativi commit inerenti ai specifici ticket contando il numero di commit . Dopo aver effettuato questo calcolo le informazioni ottenute sono state salvate in un file csv per poi graficarne il risultato e analizzare i dati.

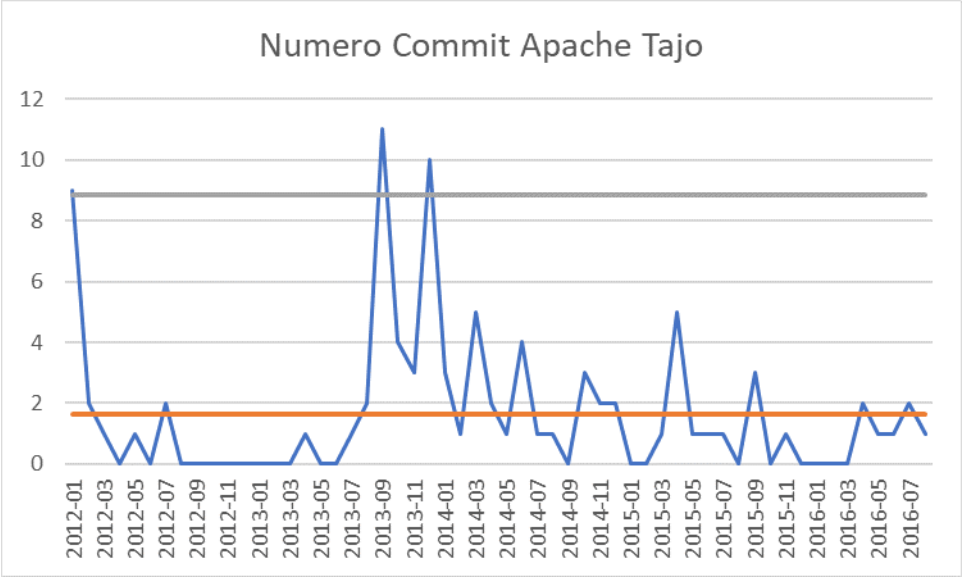


Figura 3 New Feature Tajo

Analisi dei Risultati

Il grafico rappresenta il numero di commit effettuati in un determinato periodo dell'anno, in arancione è stata rappresentata la media e in grigio il limite superiore [*media +(3*devianza standard)*], il limite inferiore [*media-(3*devianza standard)*] non compare in quanto negativo.

Il periodo analizzato è di circa 4 anni, con il periodo di massima attività nel settembre del 2013, il periodo di minore attività va dall'agosto del 2012 a luglio del 2013. Si può notare che per quasi tutto il periodo analizzato i valori sono sopra la media.

Conclusioni

Dalle osservazioni riguardanti il periodo analizzato si può concludere che il progetto è stato molto attivo nel periodo che va dalla fine del 2013 a circa metà del 2014, per poi diminuire nel tempo fino all'ultima data analizzata che ha un numero di commit inferiore alla media. Vista l'ultima data analizzata si può concludere che il progetto sia concluso o semplicemente non ci sono nuovi sviluppi. Successivamente c'è stata una integrazione del progetto con Sonarcloud per poi ridurre il numero di code smell rispetto alla prima versione del codice in Java.

Deliverable Two

Lo scopo di questa deliverable è quello di effettuare una stima dei migliori modelli predittivi per poter avere un informazioni preventiva sulla possibilità che una determinata classe di un progetto possa avere o meno un bug. I progetti che sono stati analizzati sono ***"Apache Bookkeeper"*** & ***"Apache Tajo"*** .Per poter effettuare questo tipo di studio è stato necessario creare un file csv contenente le informazioni relative ad alcune metriche delle classi. Il file è stato successivamente diviso in treaning set e testing set, il primo utilizzato per addestrare il modello di previsione e il secondo per poter effettuare la classificazione dei dati (quindi verificare tramite l'apposito classificatore la presenza o meno del BUG), il modello cercherà quindi di classificare i dati e successivamente verrà verificata la correttezza della classificazione attraverso differenti metriche. Per generare i set viene usata la tecnica di validazione walk forward (in quanto i dati dovrebbero essere dipendenti dal tempo). Sono state utilizzate anche delle tecniche di feature selection (filter) e di sampling (oversampling/undersampling e SMOTE). I classificatori che sono stati utilizzati:

- Random Forest
- Naive Bayes
- IBk

Questo lavoro è stato realizzato utilizzando WEKA, le metriche utilizzate per verificare quale tecnica e/o quale classificatore si comporta meglio sono state: Precision ,Recall ,Kappa , AUC.

- **Precision**: che indica quanto è accurato il modello ad individuare i positivi senza prendere falsi positivi [**TP/(TP + FP)]**
- **Recall**: indica quanto il modello è accurato nel trovare i veri positivi [**TP/(TP+FN)**]
- **Kappa**: quante volte il modello è stato più accurato rispetto a uno dummy
- **AUC**: probabilità che un'istanza casualmente scelta positiva sia classificata al di sopra di una casualmente scelta negativa.

TP = True Positive

FP= False Positive

FN= False Negative

I dati per poter costruire le informazioni dei 2 progetti sono stati presi dai seguenti link (Jira e Github):

- Le repository in Jira contente i ticket dei progetti (https://issues.apache.org/jira/projects/TAJO/issues) (https://issues.apache.org/jira/projects/BOOKKEEPER/issues)
- Le repositori di Github contente le informazioni delle varie classi (http://github.com/apache/tajo) (http://github.com/apache/bookkeeper)

Ottenere i dati

I dati sono stati ottenuti tramite REST API ottenendo file JSON che sono stati analizzati che sono stati salvati successivamente in file locali (Es. BookkeeperJira.json), questo è stato necessario in quanto il numero di richieste che possono essere fatte in un ora è limitato, quindi si è preferito in primo luogo scaricare tutte le informazioni necessarie per il calcolo delle metriche e poi procedere con il calcolo effettivo.

```
String url1 = "https://api.github.com/repos/apache/"+projectName+"/commits";
String url2 = "?page=";
String url3 = "&per_page=100";
```

Figura 4 Query per ottenere i dati da Github

Dal file JSON contiene le informazioni inerenti alle versioni quali:

- **TicketID:** l'id del ticket di tipo Bug presenti in Jira che sono stati risolti.
- **Fixed Version (FV):** la versione del progetto in cui è stato risolto il bug.
- **Affected Version (AV):** ovvero le versioni che sono affette da quel tipo di Bug.
- **Created:** data di creazione del ticket su Jira, necessario per capire qual è l'Observed Version del bug, ovvero la release in cui è stato scoperto.

Le AV se presenti possono essere più di una, per questo viene presa in considerazione la più vecchia versione in quanto questa è **l'Injected Version (IV),** cioè la prima versione del progetto nel quale è stato introdotto il bug.

Con le informazioni di Github incrociandole con quelle di Jira si riescono ad ottenere le informazioni per ogni commit, quali ad esempio il file modificato con il numero di righe cancellate, modificate, aggiunte ecc.

Elaborazione dati

Una volta studiata la struttura dati dei 2 file JSON si è proceduto con il calcolo delle metriche, in particolare sono state calcolate:

- **Size**: la dimensione in linee di codice del file considerato
- **LOC_touched**: il numero di linee di codice che sono state aggiunte o cancellate durante l'intera release
- **LOC_added**: il numero di righe di codice che sono state aggiunte durante l'intera release

- **MAX_LOC_Added**: il numero massimo di linee di codice che sono state aggiunte in un unico commit
- **AVG_LOC_Added**: il numero medio di linee di codice che sono state aggiunte durante l'intera release
- **Churn**: il numero di linee di codice che sono state aggiunte effettivamente durante l'intera release (aggiunte meno eliminate), questo valore può essere negativo nel caso in cui il numero di linee di codice eliminate è maggiore del numero di linee di codice aggiunte durante l'intera release
- **MAX_Churn**: il numero massimo di linee di codice che sono state aggiunte effettivamente durante l'intera release.
- **AVG_Churn**: il numero medio di linee di codice che sono state aggiunte effettivamente durante l'intera release.
- **NR**: numero di revisioni.

In particolare per poter calcolare la size sono stati scaricati le varie classi in formato codificato che successivamente sono stati decodificati per poter contare effettivamente il numero di linee di codice.

Per calcolare l'ultima metrica "Buggy" sono state sfruttate le informazioni presenti sul file JSON Jira, in particolare AV e FV che hanno permesso di identificare e attribuire l'attributo Buggy alla classe considerata. Come detto in precedenza le AV possono essere no presenti, per questo è stata utilizzata una tecnica che permette di stimare l'AV, questa tecnica è chiamata **Incremental Proportion** che si basa su una certa proporzionalità tra AV-OV e OV-FV:

1. Man mano che si analizzano i ticket che hanno AV andiamo ad utilizzare la seguente formula per calcolare il parametro **P = (FV-IV)/(FV-OV)**
2. Viene calcolata una media su P per ogni classe analizzata
3. Quando una classe non ha la IV (cioè la più vecchia AV), viene predetta utilizzando la formula **$IV_{predected}$ = FV –(FV-OV)*P**

WEKA TESTING

Ottenute tutte le informazioni precedenti l'obiettivo è quello di addestrare modelli che ci permettono di classificare delle classi di testing come contenenti bug o no. Per fare ciò in primo luogo bisogna dividere i dati in Training set e Testing set. La tecnica utilizzata per dividere i dati in questi due set è chiamata **Walk forward**, consiste utilizzare i dati precedenti per stimare i dati attuali, quindi l'insieme dei dati viene diviso in parti porzioni, ed a ogni passo la porzione di dati viene utilizzata come Testing set mentre tutte le porzioni precedenti vengono utilizzate per addestrare il modello; questa tecnica viene utilizzata perché tiene conto del fatto che le versioni possono avere metriche dipendenti dal tempo, in modo tale da non mettere mai in training delle versioni successive a quelle del testing. In questo caso i dati sono stati divisi per versione.

Infine per migliorare la classificazione è stata utilizzata la tecnica di **Sampling**, questa tecnica permette di bilanciare un dataset che in partenza risulta sbilanciato. In particolare vengono utilizzate tre tecniche:

- Under sampling: consiste nell'eliminare alcune istanze di dati maggioritari, scegliendo in modo random un numero di elementi portandolo al pari con il numero di elementi minoritari

- Over sampling: consiste nell'incrementare il numero di elementi minoritari nel dataset per portarlo allo stesso livello del numero di dati maggioritari, questo incremento viene fatto aggiungendo più volte gli stessi elementi minoritari
- SMOTE: è un meccanismo di Oversampling che crea elementi sintentici della classe minoritaria, quindi crea dati non reali utilizzando delle tecniche di stima.

In seguito vengono mostrati i risultati ottenuti per entrambi i progetti Apache:

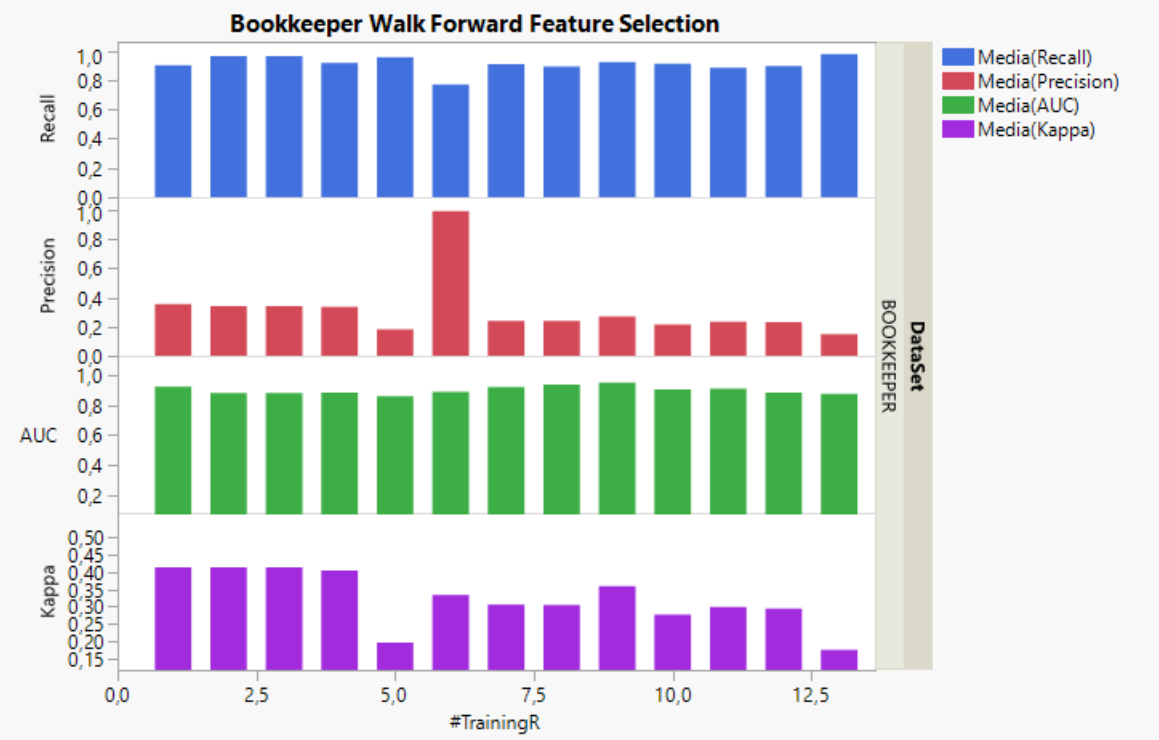


Figura 5 Bookkeeper WF Feature Selection

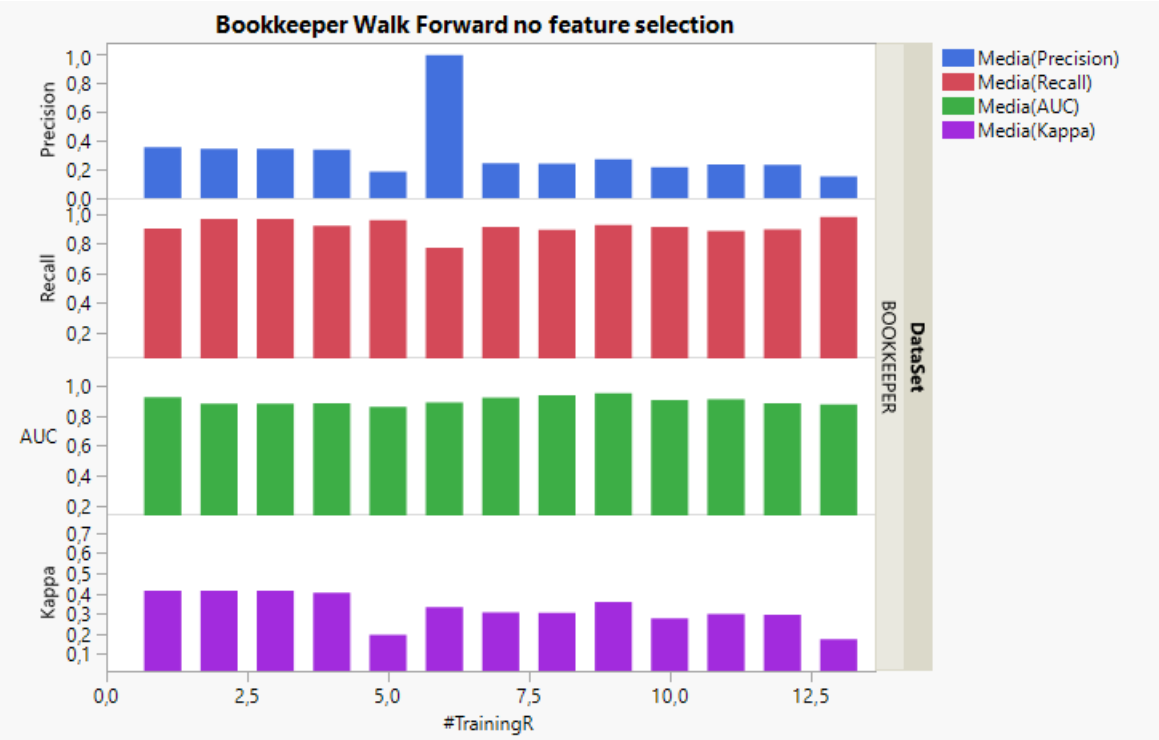


Figura 6 Bookkeeper Walk Forward Feature Selection

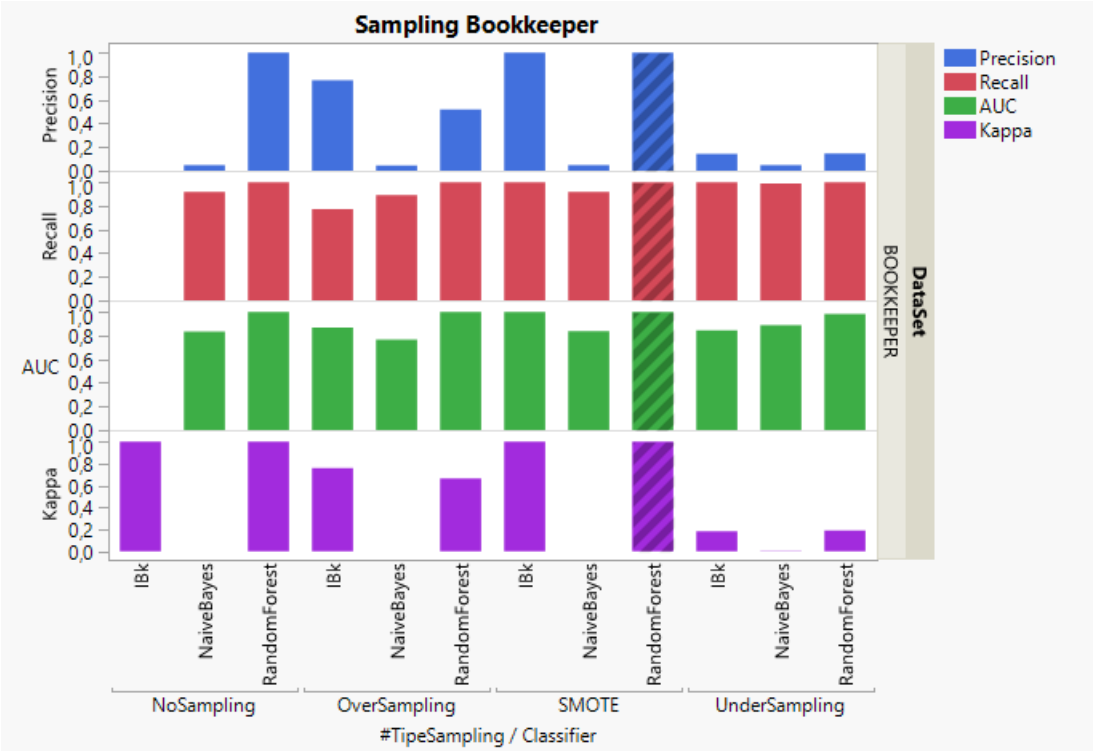


Figura 7 Bookkeeper Sampling

Analisi e risultati Bookkeeper

In generale le 4 metriche utilizzate (Recall, Precision, AUC, Kappa) più sono alte, migliore è il modello utilizzato. Questi grafici ci permetto di individuare quale tecnica/classificatore è migliore per creare un modello predittivo.

In particolare:

- *Sampling :* si può notare che per quanto riguarda i progetto di Bookkeeper la tecnica di sampling migliore risulta essere SMOTE, mentre il peggior risulta essere UnderSampling questo ci fa capire che eliminare dati in questo caso non ha aiutato i vari classificatori (in questo caso il training e i testing set sono stati partizionati 66/33)

- *Classificatore:* per quanto riguarda i classificatori si può notare facilmente che il peggiore di tutti è NaiveBayes che ha risultati peggiori soprattutto nel calcolo di Kappa, mentre il migliore è RandomForest.

- *Feature Selection:* feature selection è stato utilizzato con l'utilizzo di Walk Forward, però si può notare che l'utilizzo di feature selection non ha portato notevoli miglioramenti, inoltre si può notare che i valori più alti si trovano intorno a metà delle release, ma si dovrebbe avere una crescita con l'aumento delle release.

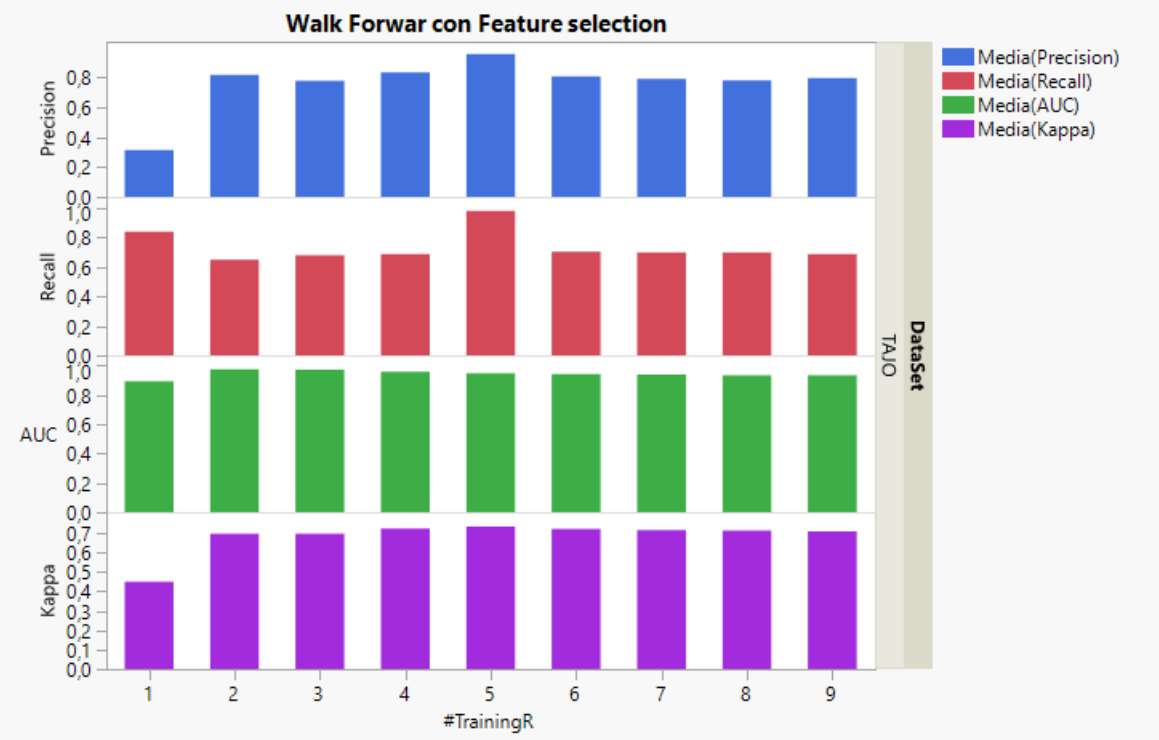


Figura 8 Tajo WF Feature Selection

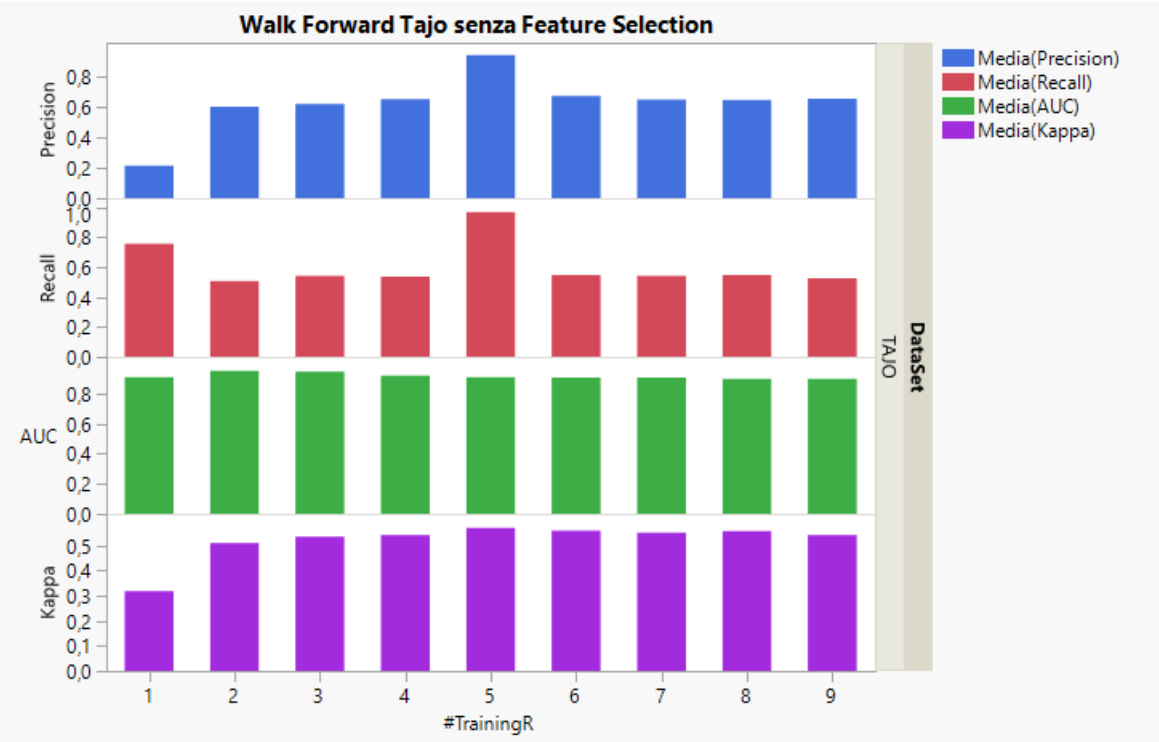


Figura 9 Tajo WF No Feature Selection

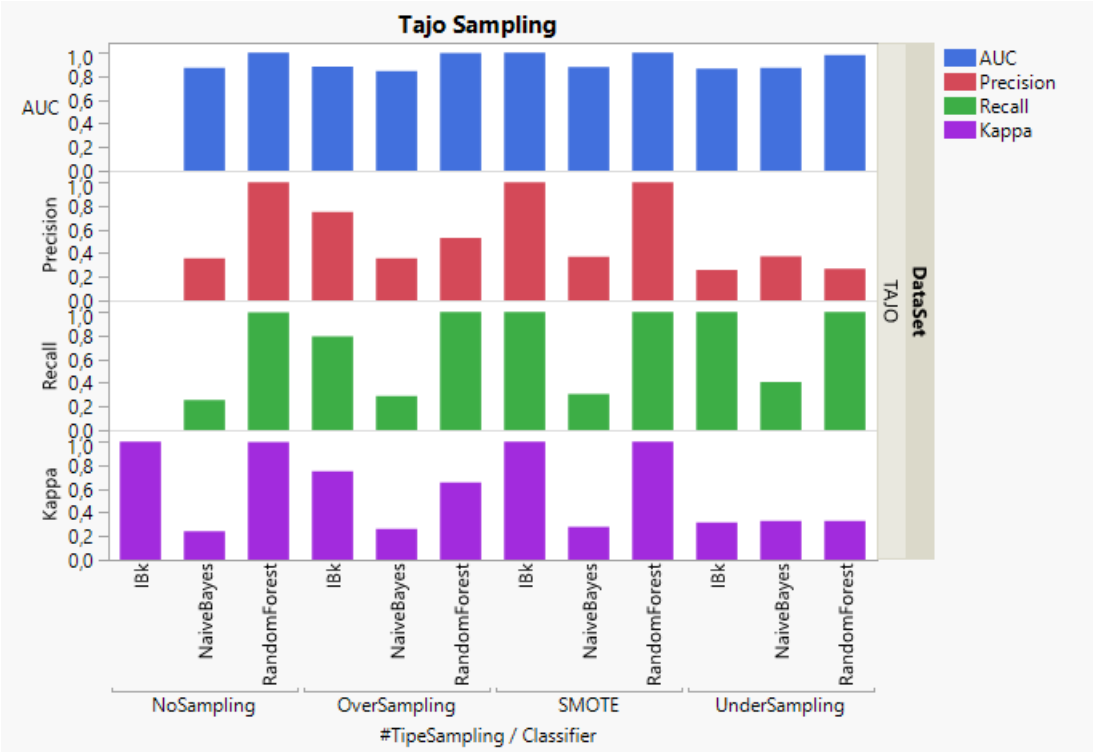


Figura 10 Tajo Sampling

Analisi e risultati Tajo

Anche in questo caso per ottenere il modello/classificatore migliore bisogna identificare i valori più alti, perche è sintomo appunto di un modello migliore

In particolare:

- *Sampling :* si può notare che per quanto riguarda i progetto di Tajo la tecnica di sampling migliore risulta essere SMOTE, mentre il peggior risulta essere UnderSampling questo ci fa capire che eliminare dati in questo caso non ha aiutato i vari classificatori (in questo caso il training e i testing set sono stati partizionati 66/33), possiamo dire questo solo per quanto riguarda il calcolo della Precision e di Kappa, perché le altre gli altri valori risultano soddisfacenti
- *Classificatore:* per quanto riguarda i classificatori si può notare facilmente che il peggiore di tutti è NaiveBayes che ha risultati peggiori soprattutto nel calcolo di Kappa, mentre il migliore è RandomForest, e confrontando con i risultati di Bookkeeper potremmo dire che la bontà di questi classificatori è simile.
- *Feature Selection:* feature selection è stato utilizzato con l'utilizzo di Walk Forward, però si può notare che l'utilizzo di feature selection non ha portato notevoli miglioramenti, inoltre si può notare che i valori più alti si trovano intorno a metà delle release, ma si dovrebbe avere una crescita con l'aumento delle release.

Infine le varie Deliverable sono state integrate con Travis Ci e con Sonarcloud, in particolare utilizzando quest ultimo è stato utilizzato per eliminare i code smell inizialmente introdotti.

Links:

- Deliverable 1:
  - Github: https://github.com/brielino/AnalisiTicket
  - SonarCloud: https://sonarcloud.io/dashboard?id=brielino_AnalisiTicket
- Deliverable 2:
  - Github: https://github.com/brielino/Deliverable2
  - SonarCloud: https://sonarcloud.io/dashboard?id=brielino_Deliverable2