REPORT DELIVERABLE 2

Nome : Chiacchia Matteo
Matricola : 0300177
Email : matteoch99@gmail.com

Repository ***Github***: https://github.com/chiacchius/Deliverable_2
SonarCloud: https://sonarcloud.io/dashboard?id=chiacchius_Deliverable_2

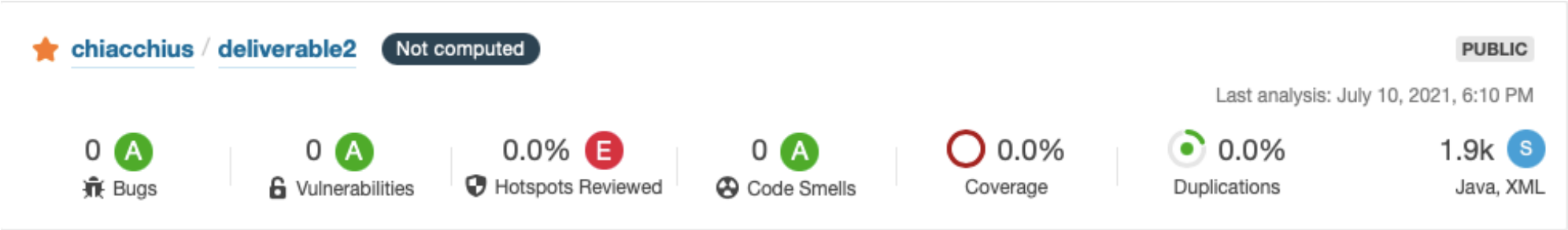
PREDIZIONE DELLA DEFECTIVENESS DI FILE SORGENTE

Note preliminari

- Descrizione del processo di *analisi* e *studio* effettuato su un progetto *open source* di **Apache**
- Risultati ottenuti tramite *automazione* fornita mediante *codice Java*, di conseguenza ottenibili per un progetto qualsiasi.
- Applicazione *hostata* sulla piattaforma di ***GitHub***
- Analisi del codice effettuato tramite ***SonarCloud***
- Utilizzo di strumenti di **Ticket Tracking** e **Machine Learning** per fornire risultati **quantitativi** e **statisticamente** significativi

Tecnologie e software utilizzati

- **Java 15.0.1** come *enviroment* di sviluppo
- **Eclipse IDE** come IDE
- **Github** e **Jira** per il raccoglimento di informazioni
- **Github** e **git** come sistema di **Remote Repository** e **Versioning Control**
- **SonarCloud** come strumento di analisi del codice
- Librerie **JGit.jar, json.jar** e **weka.jar** rispettivamente per connessione con *Github*, *parsing* di file *JSON* e per *evaluation* di modelli di **ML**
- **JMP** per la creazione dei grafici di analisi



SonarCloud values

Descrizione iniziale

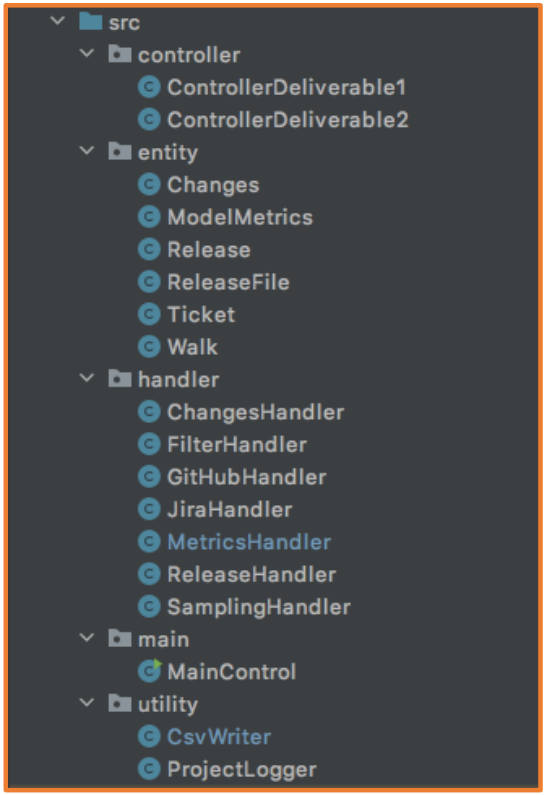
- **Eseguire uno studio empirico,** basato su modelli di **Machine Learning,** al fine di identificare le **Classi defective** di un progetto.
  - Utilizzo di tecniche come ***Feature selection***, ***Sampling*** e ***Sensitive***
    - Offerte dalle *API* di ***WEKA***
- Determinare quale combinazione di queste tecniche produce risultati migliori per ogni classificatore
- Scopo: determinare classi potenzialmente *defective* in modo tale da utilizzarle più di altre per effettuare *testing* esaustivo.
- Progetti analizzati : **BOOKKEEPER** & **ZOOKEEPER**

Progettazione

- Due *Milestone*
  - ***Milestone* 1: Acquisizione** delle metriche e della **defectiveness** dei file .java del progetto mediante **JIRA** e **GitHub**
  - **Milestone 2: Analisi** dei dati ottenuti successivamente all'elaborazione del dataset ottenuto nel primo *Milestone*, con tecniche di **Machine Learning**
- *Milestone* 1
  - Obiettivo: determinare se una specifica classe è **Defective** su una determinata **Release**
  - Acquisire metriche della classe (*size*, numero di revisioni ecc…)
  - Rimozione del **50%** delle ***release*** più recente.
    - Prima di poter definire un file buggy è necessario accorgersi della presenza di un bug.
    - Spesso il tempo passato tra introduzione e rivelazione del bug è non trascurabile (**class snoring**), portando a un'incertezza
    - **Missing rate** del *10%* quando si vanno ad eliminare il *50% delle releases*

Progettazione

- Milestone 2
  - utilizzare le informazioni ottenute nel primo *Milestone* per fare valutazioni su quale modello di *Machine Learning* produca risultati **più accurati**
  - 3 **classificatori utilizzati :** *Random Forest, Ibk, NaiveBayes*
  - Obiettivo: Confrontare tecniche di **Machine Learning** e verificare quale sia la migliore per il nostro caso
- File sorgenti **organizzati** in ***packages*** a seconda dello scopo.
- Codice pensato per avere **un'alta modularità e riusabilità.**
  - Possibilità di utilizzarlo per effettuare lo studio su qualunque progetto ***Apache***
  - Possibilità di riutilizzare le componenti.

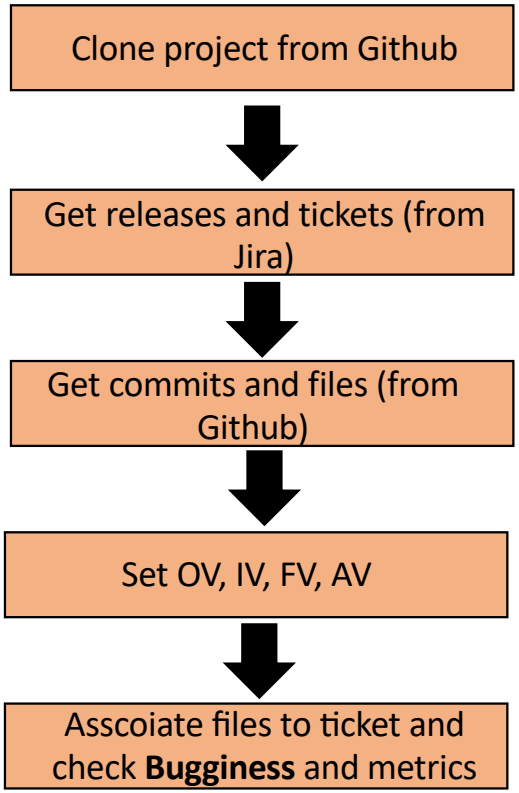


Analisi del sorgente – Milestone 1

- Ottenere le **metriche** e la **bugginess** di ogni **classe** di ogni **release.**
- **Release** raccolte da **JIRA**
  - Tramite un'interrogazione a: https://issues.apache.org/jira/rest/api/2/project/***projectName***
  - Restituzione di un file ***JSON*** che viene scansionato per ottenere le *Release*, con le varie informazioni, e salvarle su istanze della classe ***Release.java***
- **Metriche** computate usando il *log* dei commit forniti da **Github** (tramite l'interfaccia *JGit*)
  - Si effettua inizialmente il ***clone*** della *repository*
  - Utilizzati unicamente i commit del **master Branch**, scartati quelli degli altri cicli di sviluppo
  - L'applicativo calcola tutte le metriche di tutte le *Release*, scartando poi, in fase di creazione dell'output, la seconda metà di queste.

Analisi del sorgente – Milestone 1

- Ottenimento dei file .java da associare ad ogni *Release*
  - Presi dall'ultimo *commit* prima del rilascio di una versione.
  - Si ottiene il *File Tree* dal *commit* e si ricavano i *path* dei file.
- Per ogni file si crea un'istanza della classe ***ReleaseFile.java***
  - **Metriche** calcolate : LOC, LOCAdded, AvgLocAdded, MaxLocAdded, LocTouched, Churn, maxChurn, avgChurn, numAuthors, numRevision

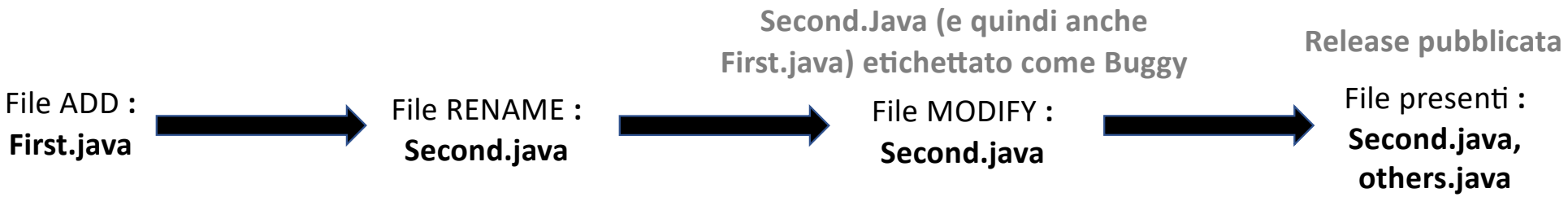


Calcolo della bugginess

- **Bugginess** calcolata mediante l'impiego di ticket su **JIRA**
- Per ogni ticket si prelevano:
  - **Id del ticket**: utilizzato per trovare i *commit* di *fix*
  - **Creation date:** per trovare la *Opening Version*
- I *commit* relativi ai ticket sono utilizzati per determinare i file '*toccati*'(grazie alle ***DiffEntry*** dell'interfaccia ***JGit***), che saranno quindi *defective.*
  - Ogni ticket ha : OV, IV, FV, AV
- Caso particolare: ***Injected Version***
  - Caso 1: Si preleva il vettore delle AV da *Jira* e, se esiste e il primo elemento ha id di versione minore o uguale alla OV, si considera questa come IV.
  - Caso 2 : ***Proportion*** con *Moving Window* (1% di release precedenti con *IV* presa da Jira)
- ***Fixed Version:*** Release in cui appare l'ultimo *commit* relativo al ticket
  - Si assume sempre attendibile.
  - Se non si trovano *commit* relativi a un ticket di *Jira* si scarta il ticket
- ***Affected Versions:*** **[Injected Version, Fixed Version)**

Gestione delle DiffEntry

- Le ***DiffEntry*** tengono traccia dei cambiamenti portati dai *commit*.
  - 5 tipi : **ADD, MODIFY, DELETE, RENAME, COPY**
- Idea alla base : ogni *DiffEntry,* di tipo **MODIFY** o **DELETE,** tiene traccia di un cambiamento effettuato per 'risolvere' il bug del ticket su cui si sta lavorando. I *Files* soggetti a questi cambiamenti sono etichettati come ***defective.***
- Caso particolare : **RENAME**
  - Ogni oggetto *ReleaseFile* tiene traccia di tutti i *path* che un *file* può avere avuto (lista di alias)
  - Se un file è *buggy* con un ***path***, ovviamente lo è anche col ***path* precedente/successivo**.



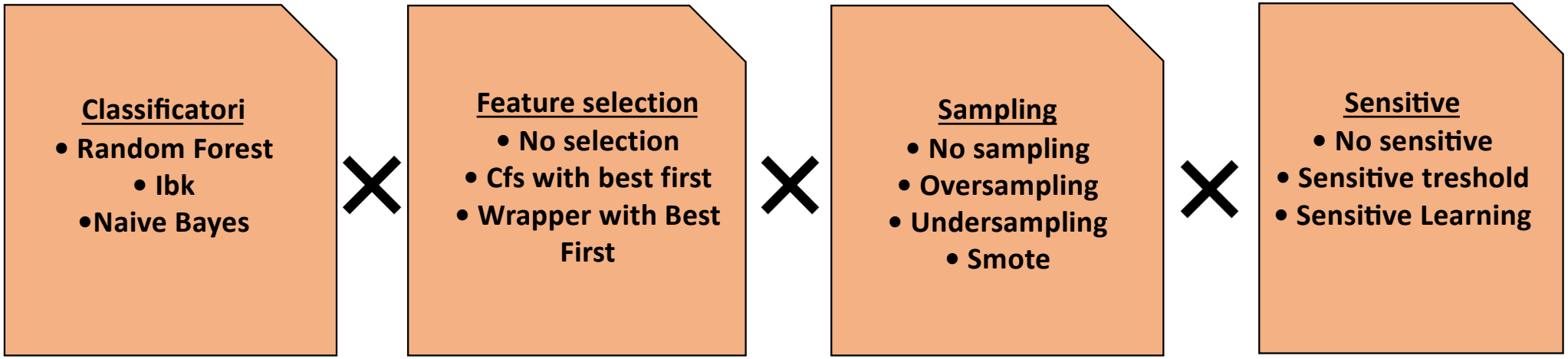
Proportion

- **Proportion** con **Moving Window** (1%).
- Si salvano, inizialmente, le *Injected Versions* di tutti i ticket per i quali sono disponibili da Jira.
  - IV utilizzate per il calcolo del *proportion* effettuato successivamente
- *P = (FV – IV) / (FV – OV)*
  - La *P* utilizzata sarà la mediana delle *P* calcolate sull'1% dei ticket precedenti.
  - Se FV = OV denominatore uguale a 0 → si assume denominatore uguale a 1
- IV predetto*: IV = FV – (FV – OV) * P*
  - Se IV compreso tra prima *Release* e OV incluse → OK
  - Se IV > OV → IV = OV
  - Se IV negativo → IV = prima *Release*

Analisi del sorgente – Milestone 2

- Più piccola e più semplice da un punto di vista algoritmico del *Milestone* 1
- ***Evaluation tecnique*** utilizzata : ***walk-forward***
  - Preserva l'ordine temporale dei dati
- Si divide il *dataset* in due porzioni (*training set* e *testing test*) che rappresentano il ***Walk-forward*** (classe ***Walk.java***)
  - Criterio di divisione : **versione**
  - ***Walk k-esimo*** contiene le *Instances* di ***training set*** (release [1;k-1]) e ***testing set*** (release k)
- Si crea ogni combinazione possibile tra **Classificatori**, **feature selection**, tecniche di **sampling** e tipi di **sensitive**
- Come valore positivo è stato scelto il valore di *defective*

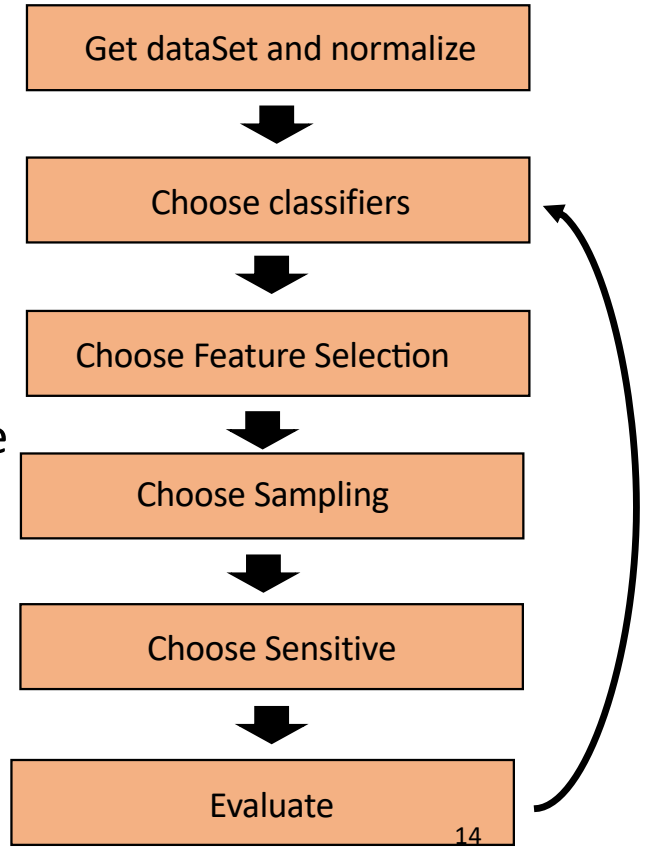
Modelli di machine learning



Prodotto cartesiano tra **classificatori**, **Feature selection**, **Sampling** e **Sensitive** per ottenere ogni tipo di combinazione

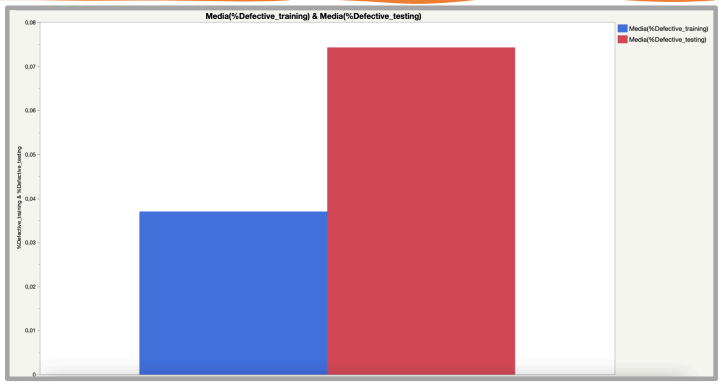
Flusso di operazioni

- Si utilizza il *dataset* creato nel *Milestone* 1
  - Esportazione del .csv in .arff per permettere a *Weka* di lavorarci
- Dati ***preprocessati*** e ***normalizzati***
  - Si imposta il *classIndex*, cioè l'attributo su cui si faranno le stime
  - Si normalizza per garantire che ogni attributo abbia lo stesso peso nella fase di *training* del modello
  - Si evita il *bias* causato dalla differenza di scala
- Per ogni ***Walk*** vengono eseguite tutte le combinazioni analizzate precedentemente
- Si effettua l'*evaluation* per ogni combinazione per poi salvare i risultati in una struttura dati (***ModelMetrics.java***) utilizzata per creare il file *.csv* finale
  - Se un solo valore calcolato non è rappresentabile (es. NaN) si scarta il ***Walk*** per evitare di compromettere l'analisi

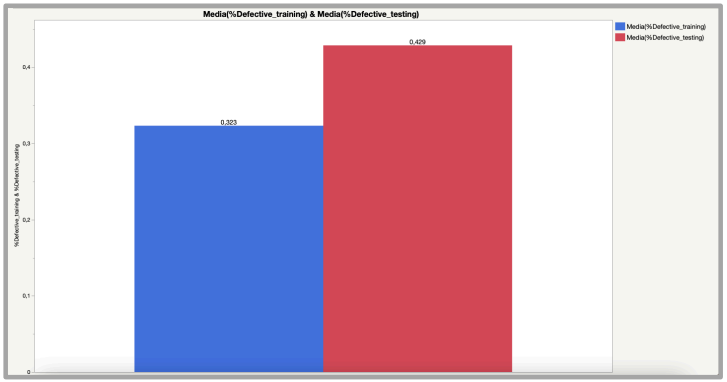


Analisi dei dati

- Studio focalizzato nel cercare la migliore combinazione delle 4 variabili del modello
  - ***Precision***, ***Recall***, **AUC** (o ***ROC Area***) e ***Kappa***
- Alcune osservazioni
  - Dati più bilanciati in *Bookkeeper* (circa 50% file *defective*) rispetto a *Zookeeper* (circa il 7%)
  - Numero di ***walk*** di *Zookeeper* molto più alto grazie a una maggiore grandezza del ***Dataset***
    - Ma meno accurato a causa dell'enorme divario di percentuale di *bugginess*
- Comparazioni effettuate a parità di metrica (*Precision*, *Recall* ecc.)
- Software utilizzato per la creazione dei grafici : ***JMP***



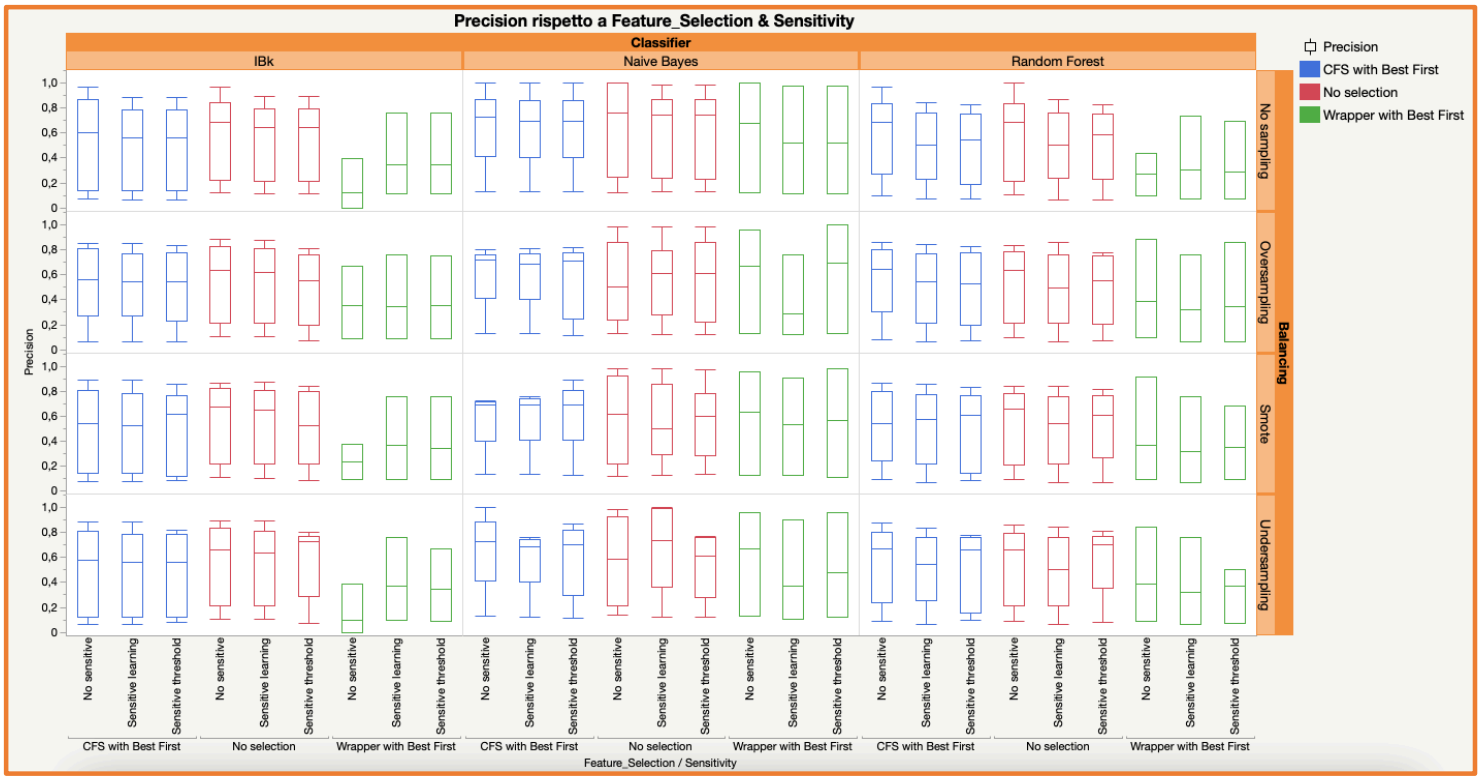
Percentuale bugginess in training e testing set BOOKKEEPER



Percentuale bugginess in training e testing set ZOOKEEPER

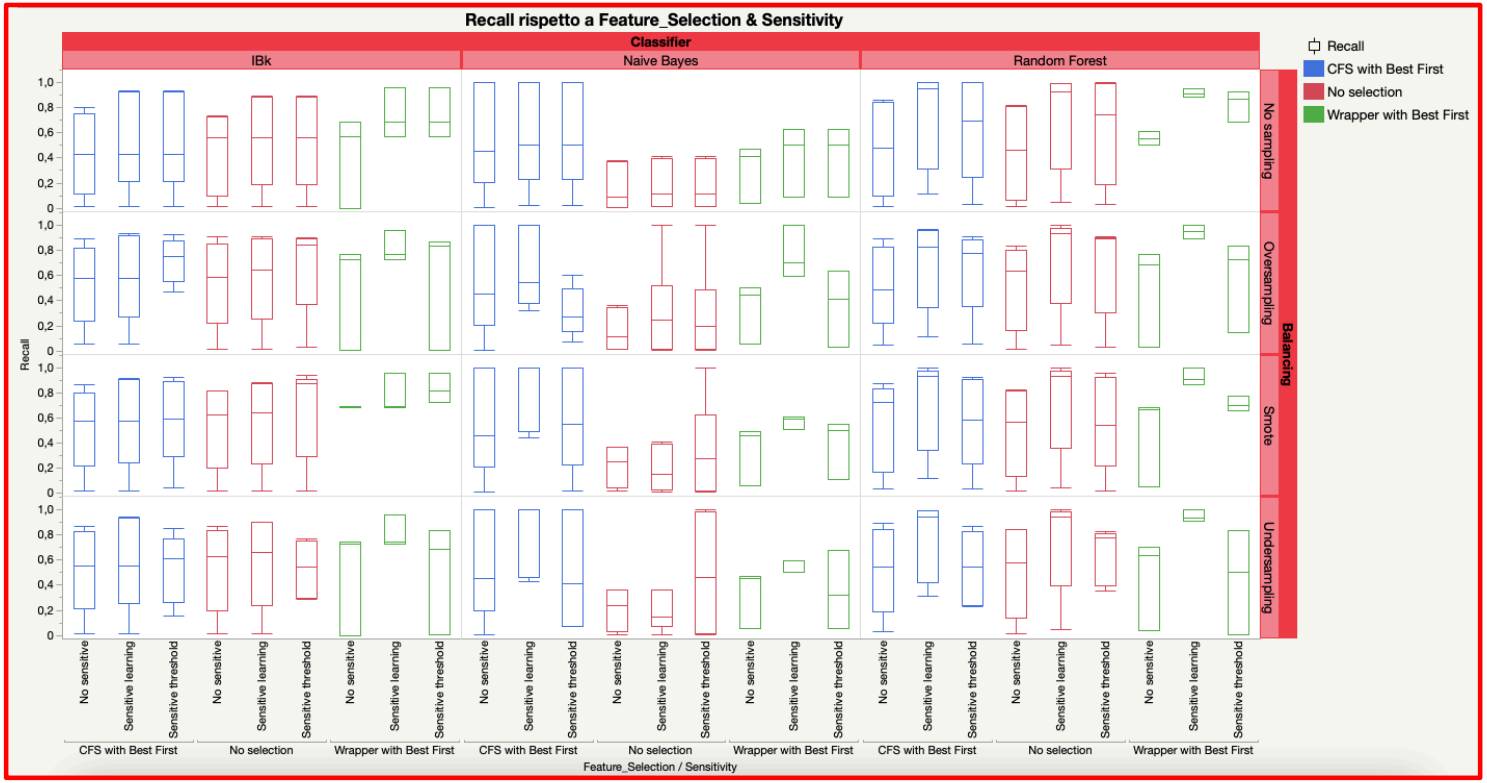
Bookkeeper Precision

- Valori con *varianza* alta
- **Mediane** più alte in ***Naive Bayes*** e **No Sampling**
  - Ma *varianza* elevata
- Varianze più basse in **Ibk, Wrapper con BF** e **No Sensitive**
  - Ma valori bassi



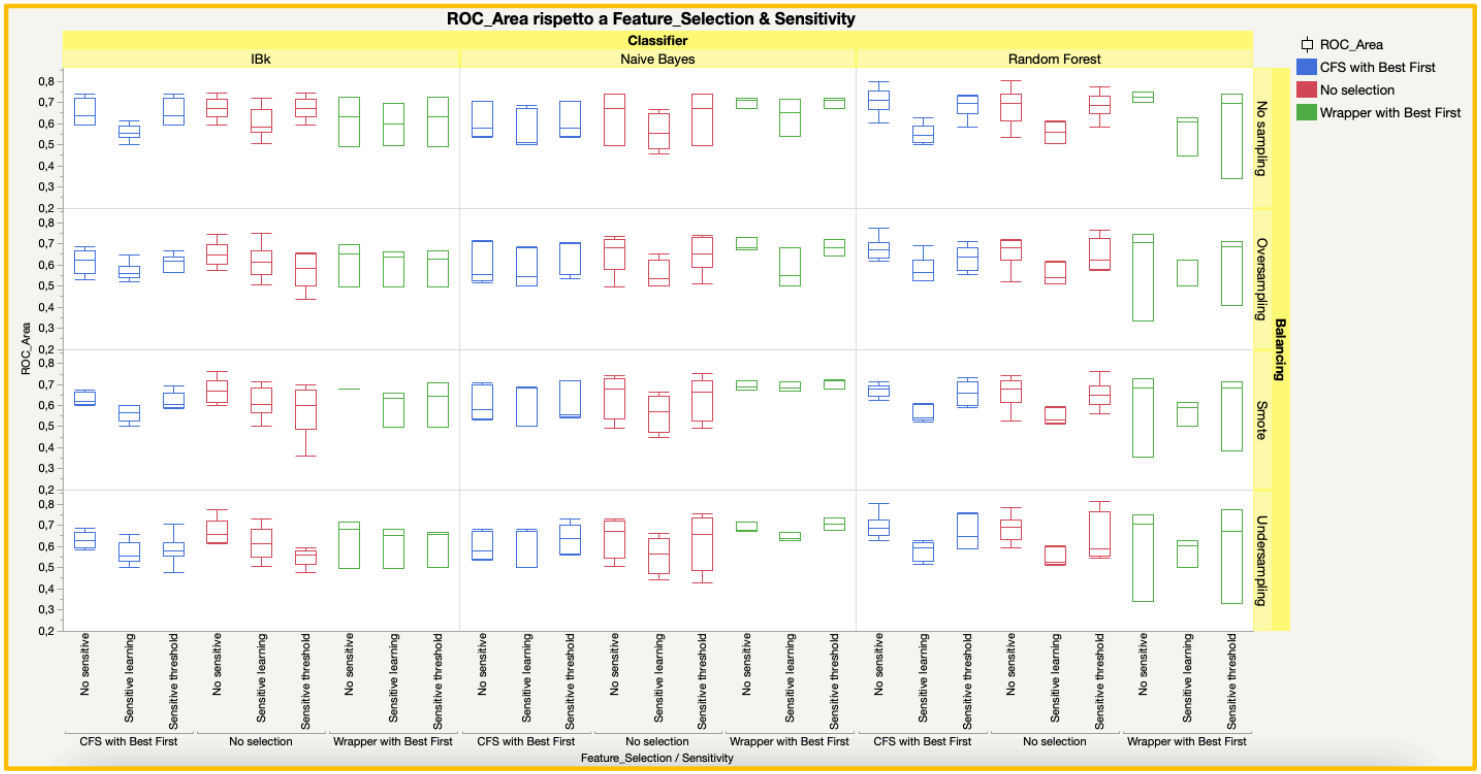
Bookkeeper Recall

- Migliori valori con {***Random forest, wrapper con BF, undersampling, Sensitive Learning}***
  - ***Mediana*** con valore alto e poca varianza
- Con ***Ibk*** i migliori risultati sono con ***Sensitive Learning*** e ***Wrapper con BF***
  - Anche ***Naive Bayes*** ma solo con ***Smote*** e ***Undersampling***



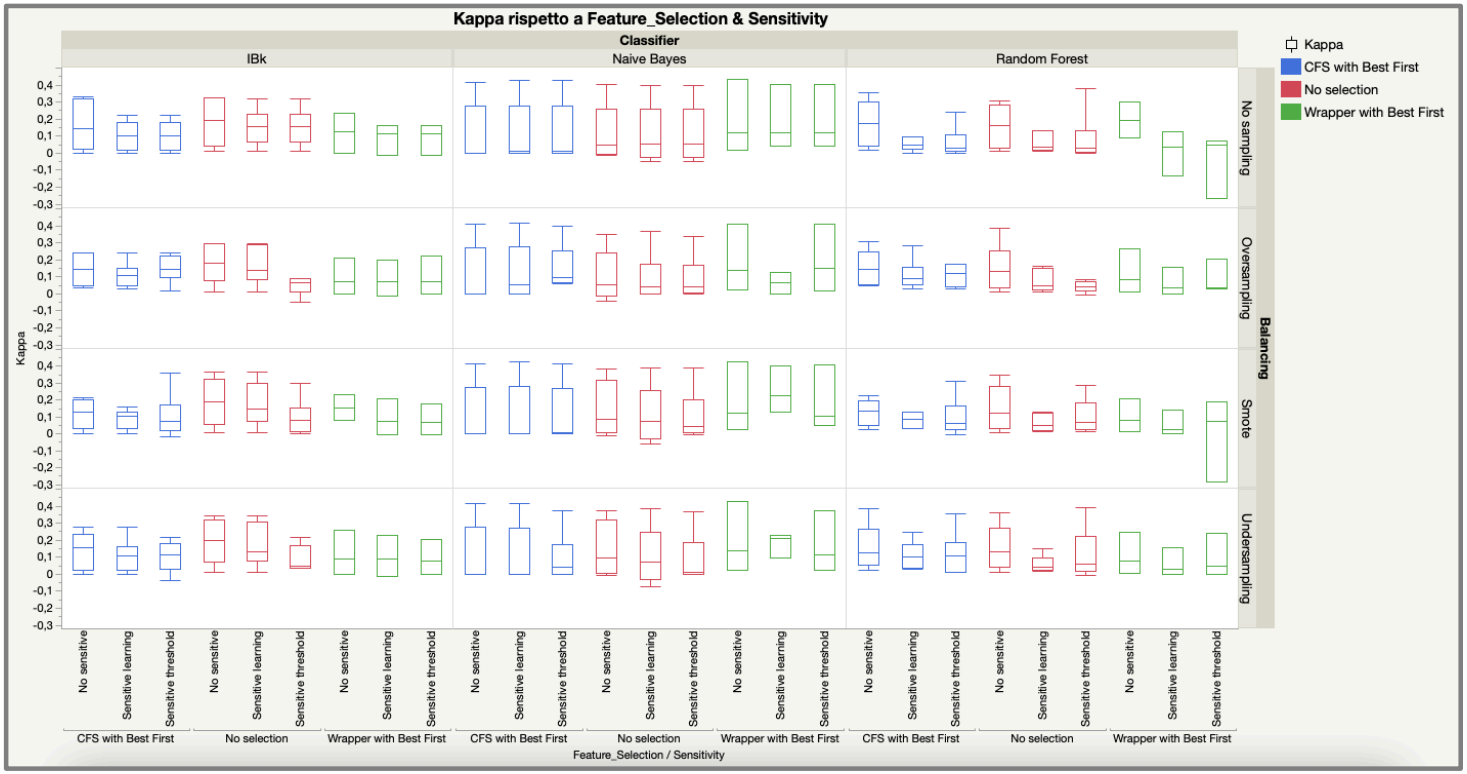
Bookkeeper ROC_Area

- Valore con, in generale, poca **varianza**
  - ***Eccetto Random Forest, Wrapper con BF e No sensitive*** (eccetto con ***No sampling***)
- Migliori valori con {***Random Forest***, ***Wrapper con BF***, ***No sampling***, ***No Sensitive***} e {***Ibk***, ***Smote***, ***Wrapper con BF***, ***No Sensitive***}
  - Non sono i valori più alti ma comunque soddisfacenti e con poca varianza



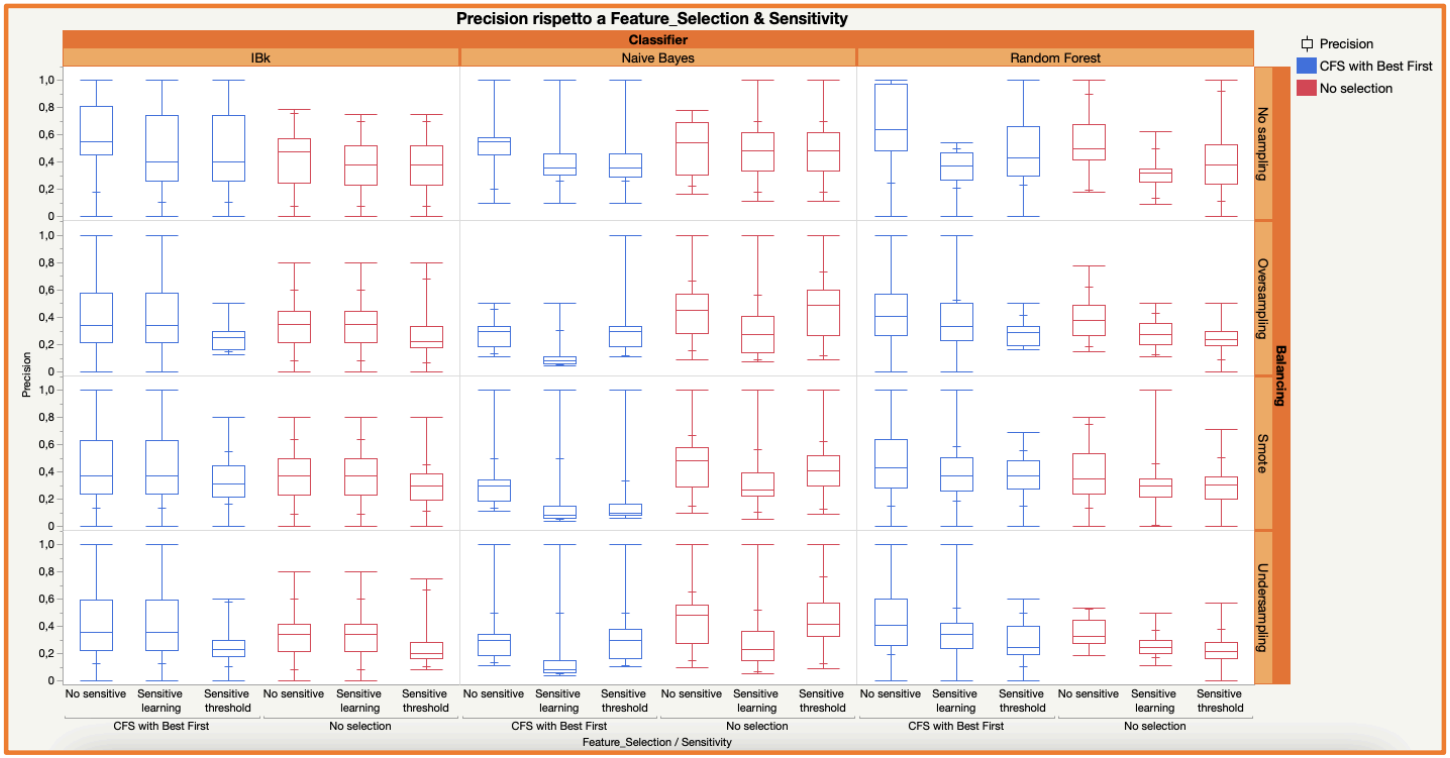
Bookkeeper Kappa

- Valori, in generale, simili tra loro
- Mediane non eccessivamente sopra lo 0
  - Poco guadagno rispetto a un classificatore *dummy*
  - Alcuni casi con valori negativi
- Migliori valori con {***Random Forest***, ***No sampling***, ***Wrapper con BF***, ***No Sensitive***}



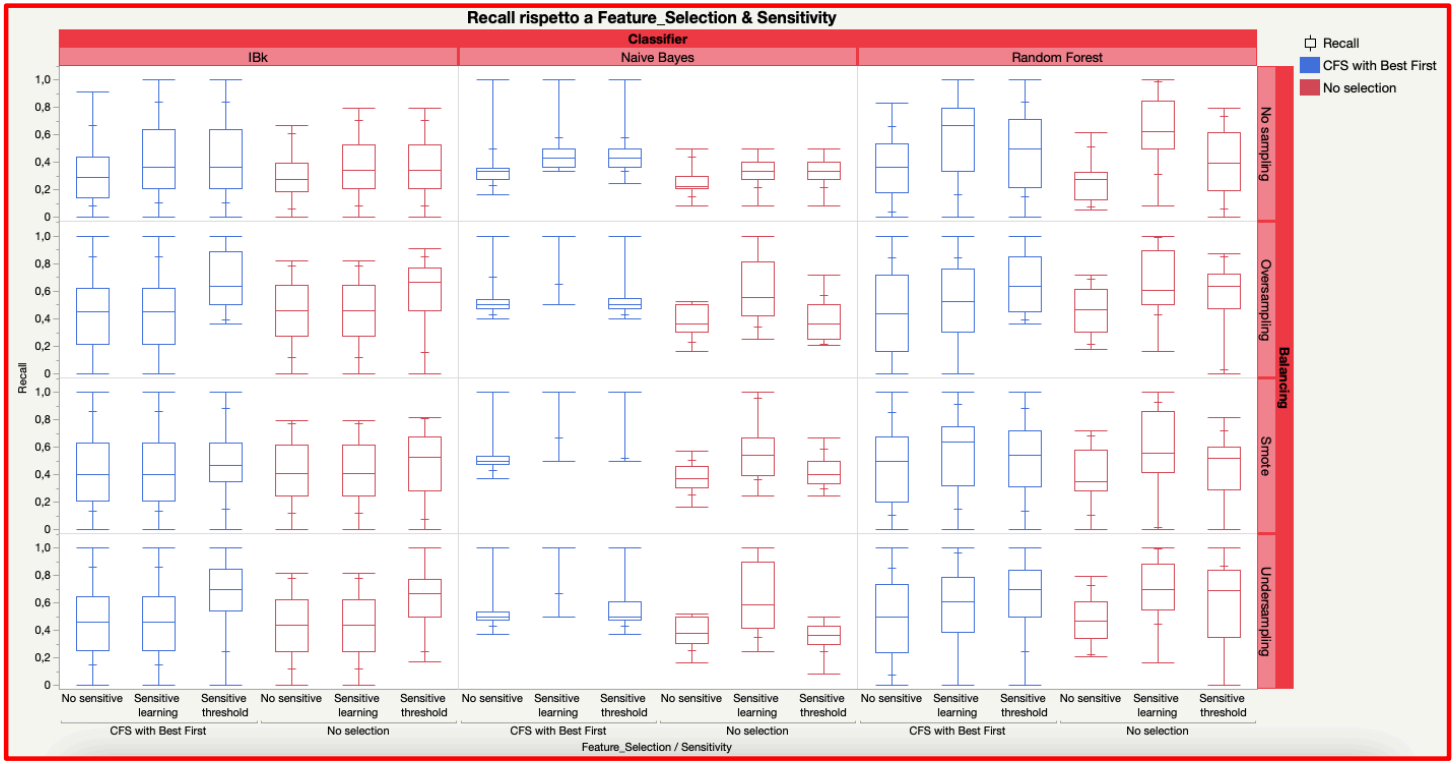
Zookeeper Precision

- Valori relativamente bassi (mediana quasi sempre sotto 0.5)
- Migliori valori con {***Random Forest***, ***CFS con Bf***, **No sampling**, **No sensitive**}
  - Soprattuto considerando il quantile di terzo grado



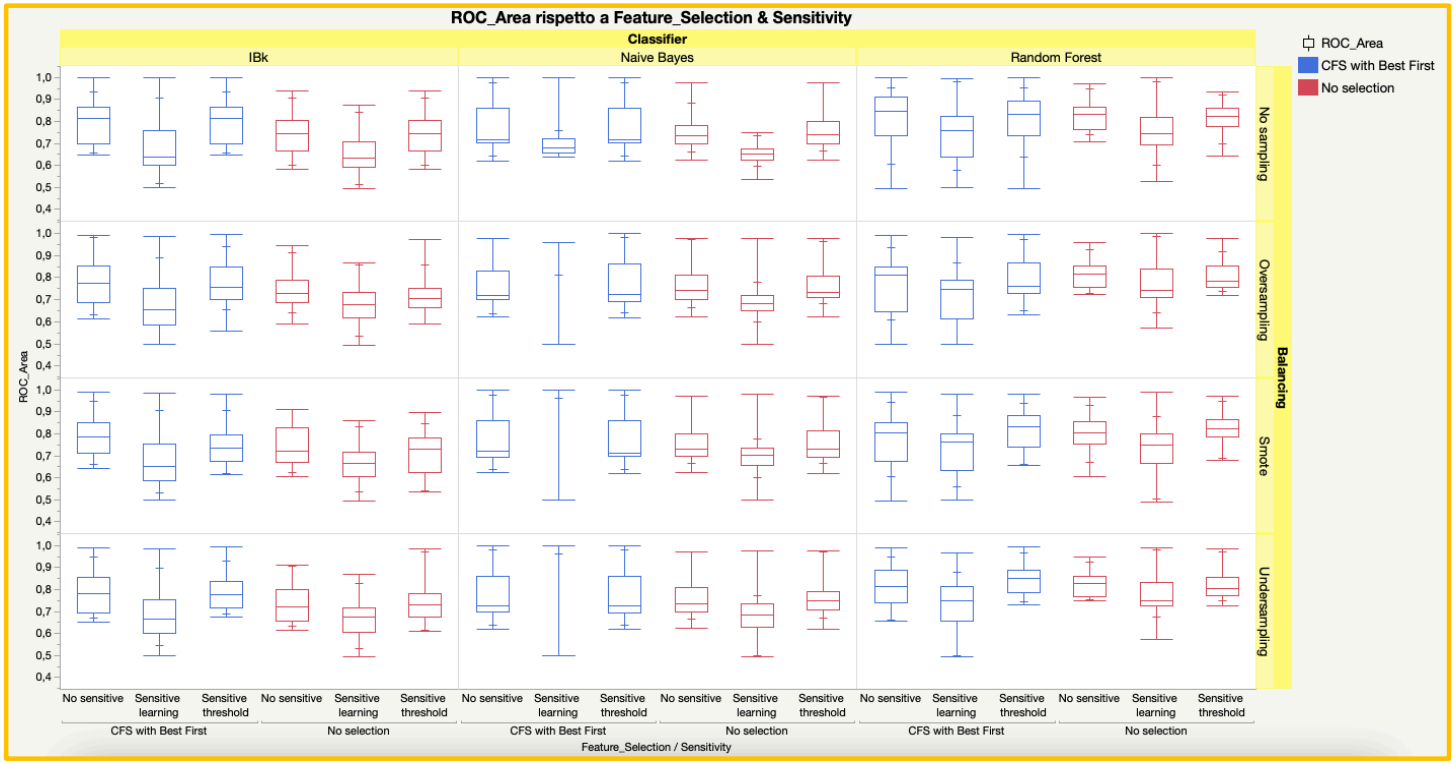
Zookeeper Recall

- **Varianze** non troppo elevate ma **mediane** basse
- ***Naive Bayes*** classificatore con varianze minori
- ***Random forest*** con ***Cfs con BF*** e ***Sensitive*** ha le mediane più alte
  - Migliori risultati con {***Random forest, Cfs con BF, Smote, Sensitive threshold***}
  - Anche {***IBk, Cfs con BF, Oversampling, Sensitive threshold***} ha buoni valori



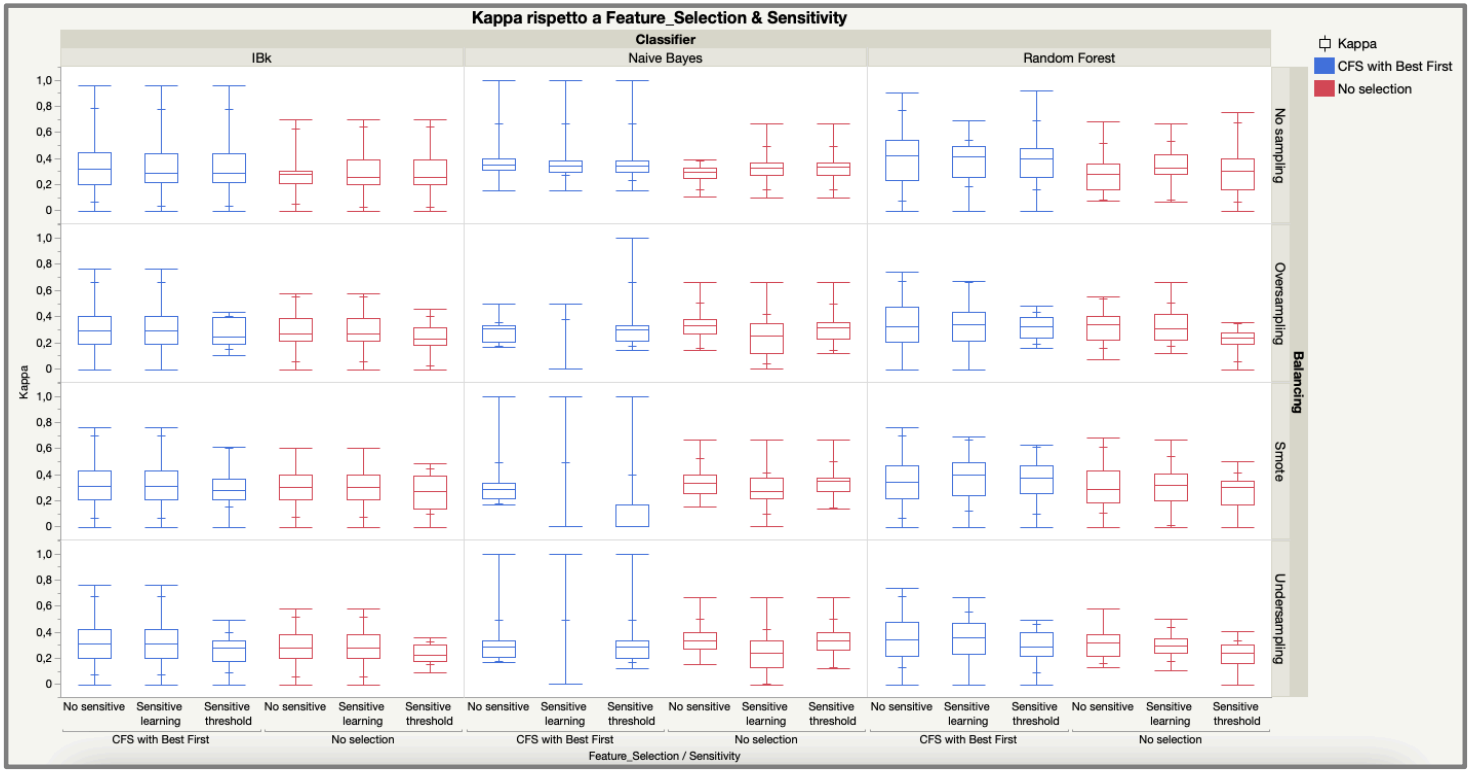
Zookeeper ROC_Area

- Come ***Bookkeeper*** si ha una **varianza** bassa
- Miglior combinazione : {***Random forest, No feature, No sampling , Sensitive threshold***}
- ***Ibk*** e ***Naive Bayes*** si comportano in maniera simile



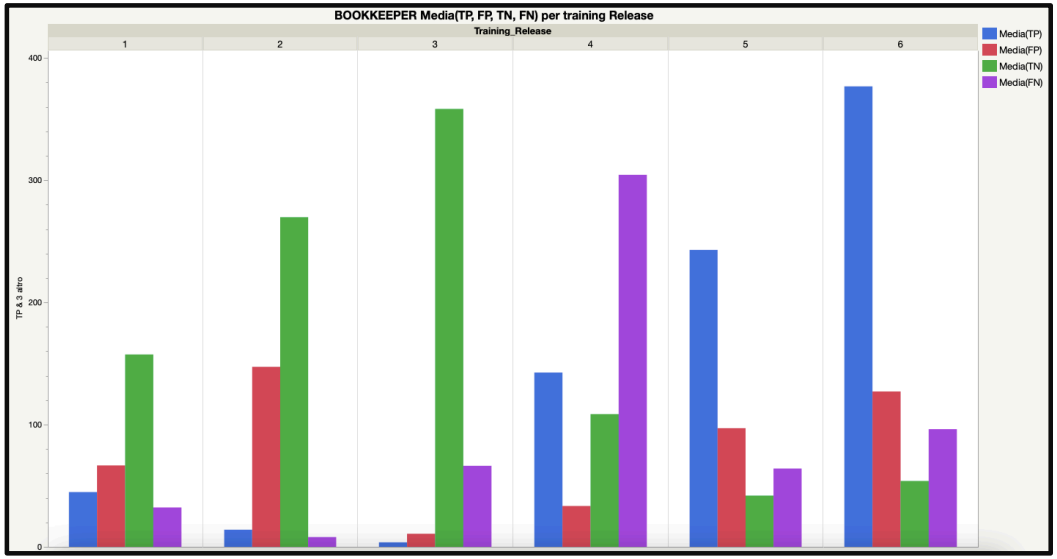
Zookeeper Kappa

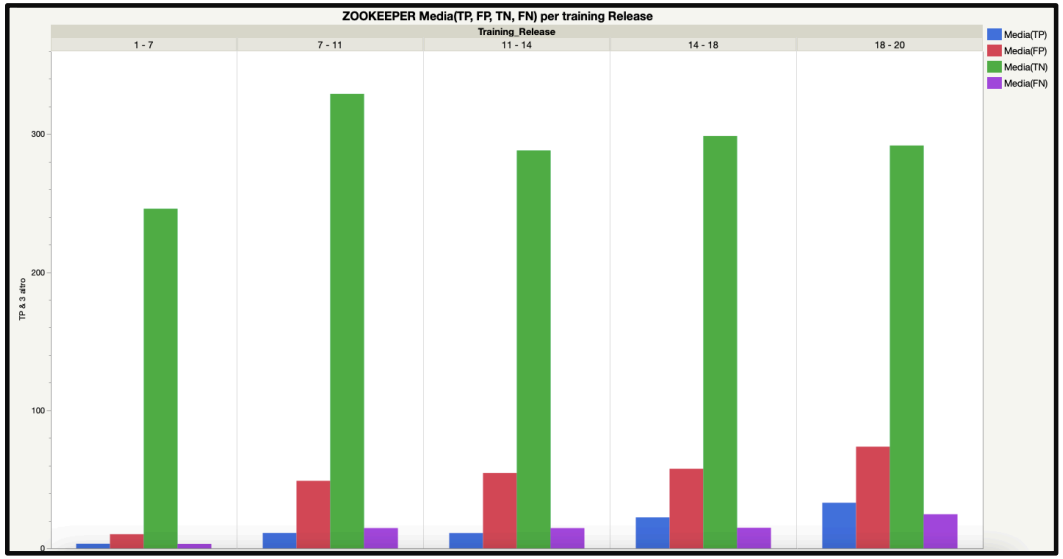
- Valori molto simili tra loro
- Mediana più alta con {***Random Forest***, ***CFS con Bf***, **No sampling**, **No sensitive**}
- ***Naive Bayes*** buono con ***No Feature***
- ***Varianze*** migliori con ***No Feature*** ma valori più alti con ***CFS con BF***



Altri grafici

- ***Media*** dei **TP, FP, TN, FN** con varie *training Release*
- *Bookkeeper* ha un numero di ***TP*** crescenti nel tempo
  - Elevata percentuale di classi *defective*
- Zookeeper ha un numero elevato di ***TN***
  - Progetto con alta qualità e poche classi *defective*





Conclusioni

- Classificatore più accurato***: Random Forest*** (specialmente abbinato a ***CFS*** o ***Wrapper***)
  - ***Ibk*** ha risultati spesso analoghi (con ***CFS*** e ***No Sampling*** e ***Sensitive Learning***)
  - ***Naive Bayes*** non è la scelta da considerare nella maggior parte dei casi (la *Feature selection* migliora un minimo i valori)
- L'utilizzo della ***Feature Selection*** migliora i dati
  - ***Wrapper con BF*** non presente in *Zookeeper* perché utilizza come attributi unicamente 'numero di *Release*' e *'Bugginess*' –> walk scartati
- Le tecniche di **Sampling** spesso peggiorano i risultati
- La ***sensitive*** aiuta ad avere risultati migliori, specialmente se abbinata a ***Random Forest***
- ***Smote*** non si è particolarmente distinto anche se ha una elevata complessità e tempi di esecuzione