TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Machine Learning for Software Engineering

Michele Tosi – matricola 0327862

Agenda

- Contesto e obiettivo
- Metodologia:
  - Individuazione delle coppie (classe, release) buggy
  - Costruzione del dataset per i classificatori
  - Metriche considerate
  - Valutazione dei classificatori
  - Classificatori e tecniche di utilizzo confrontate
- Risultati e conclusioni
- Minacce alla validità
- Links a GitHub e Sonarcloud

Contesto

- Ogni progetto di ingegneria del software include il testing per **identificare** e **correggere** i bug.
- L'attività di testing è **onerosa** e **costosa**, quindi, non può essere effettuata in modo esaustivo.
- **Problema:** individuare le porzioni del progetto che più verosimilmente contengono bug su cui incentrare i test.
- **Idea:** sfruttare le informazioni del passato riguardanti le classi caratterizzate da bug per predire quali classi in futuro potranno averne.

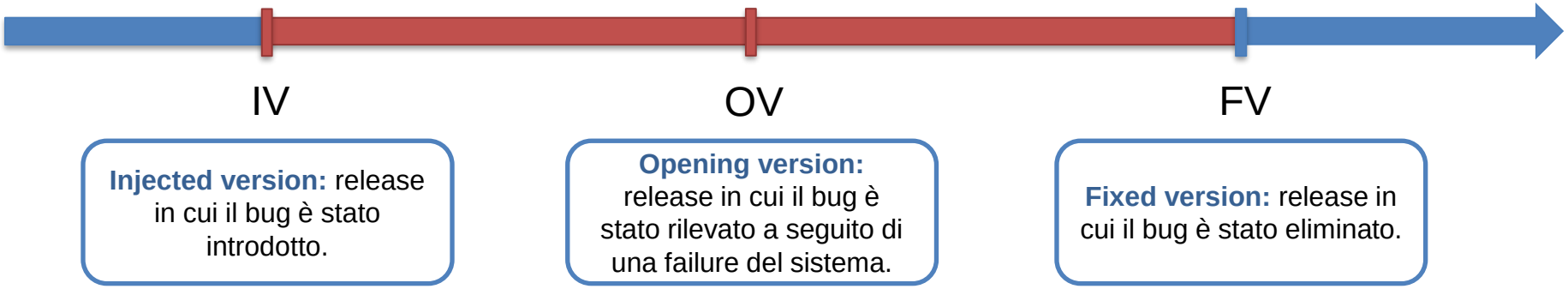
Obiettivo

- Rendere il processo di testing più **efficiente** e **mirato**.
- Dopo aver individuato quali classi sono state buggy e in quali release, stabilire quale classificatore effettua le predizioni migliori e con quali tecniche di utilizzo.
  - Classificatori considerati: **Random**, **Forest Naive Bayes** e **IBk**.
  - Tecniche di utilizzo considerate: **feature selection**, il **sampling**, il **cost sensivity** e alcune loro combinazioni.

Metodologia:

Individuazione coppie (classi, release) buggy

- **Idea:** ogni bug ha un ciclo di vita.



- Le classi affette dal bug sono quelle comprese tra l'injected version (inclusa) e la fixed version (esclusa).
- Le informazioni su IV, OV e FV sono state recuperate tramite le issue presenti su Jira.
- **Problema:** non tutti i ticket in Jira hanno la injected version.

Metodologia:

Individuazione coppie (classi, release) buggy (2)

- **Idea:** supporre proporzionalità tra l'arco di tempo che trascorre da quando un bug è rilevato a quando viene risolto e l'arco di tempo che trascorre da quando un bug viene introdotto a quando viene risolto.

- **Proportion:** tecnica utilizzata per stimare l'injected version dei bug, si basa sull'utilizzo della costante di proporzionalità p calcolata sui bug per cui IV è nota.

$$P = \frac{FV - IV}{FV - OV}$$

$$IV = FV - (FV - OV) \times P$$

Metodologia:

Individuazione coppie (classi, release) buggy (3)

- Si sono utilizzate le seguenti varianti di proportion:
  - **Cold-Start:**
    - proportion calcolata a partire dai ticket di altri progetti simili e non da quelli del progetto in esame.
    - Utilizzato nel caso in cui i ticket a disposizione sono considerati troppo pochi (meno della taglia della moving window).
  - **Moving-Window:**
    - proportion calcolata su un numero limitato di difetti dello stesso progetto (rilassa l'assunzione di incremental secondo cui p rimane mediamente costante durante tutto il progetto).
    - **Problema:** scelta della dimensione della finestra.

Metodologia:
Costruzione del dataset

- Per ridurre il fenomeno dello **snoring** sono state eliminati tutti i dati relativi alla seconda metà delle release.

- A ciascuna coppia (classe, release) vengono assegnate alcune **metriche** che verranno sfruttate dai classificatori per effettuare le predizioni.

- A ciascuna coppia (classe, release) viene assegnata una label booleana buggy o non buggy (a seconda se nella release la classe era buggy).

Metodologia:

Metriche considerate

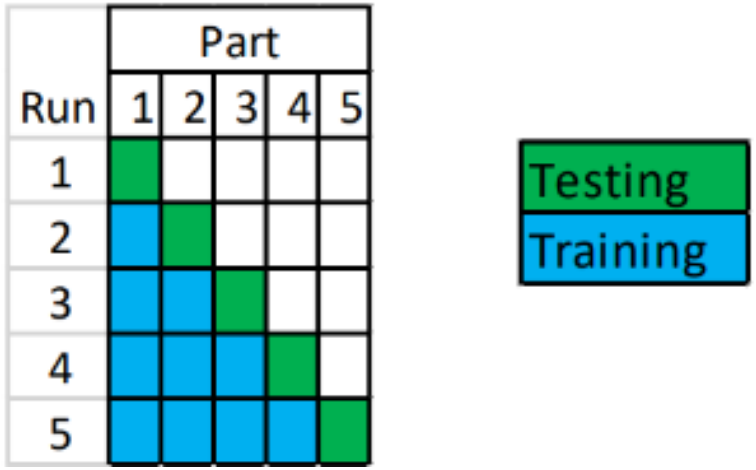
Nome	Descrizione
Size	Numero di linee di codice.
LOC added*	Somma delle linee di codice aggiunte sulle revisioni relative alle release.
Churn*	Somma di |LOC aggiunte – LOC rimosse| sulle revisioni di una release.
NR	Numero di revisioni all'interno della singola release.
Nauth	Numero di autori.
Change set size*	Numero di file committati insieme nelle revisioni relative alla release.

Delle metriche segnate con '*' sono stati considerati anche i valori massimo e medio, non sono stati inseriti nella tabella per migliorare la leggibilità.

Metodologia:

Valutazione dei classificatori

- **Reminder:** l'obiettivo è stabilire quale classificatore ha le prestazioni migliori e con quali tecniche di utilizzo.
- È necessario effettuare una **valutazione** dei classificatori.
- La tecnica utilizzata è il **Walk Forward**.



Metodologia:
Valutazione dei classificatori

- **Walk Forward:** è una tecnica di validazione **time-series**, tiene conto dell'ordine temporale dei dati (non si possono utilizzare nel training set informazioni future).

- Il dataset viene diviso per release ordinate cronologicamente:

  – Costruzione del **training set**: con le prime 'k' release viene eseguito il labeling esclusivamente con le informazioni **disponibili fino a quel momento**.

  – Costruzione del **testing set**: per ogni iterazione conterrà le informazioni sulle classi della release k+1-sima su cui andranno fatte le predizioni che avranno il labeling in base a **tutte le informazioni disponibili**.

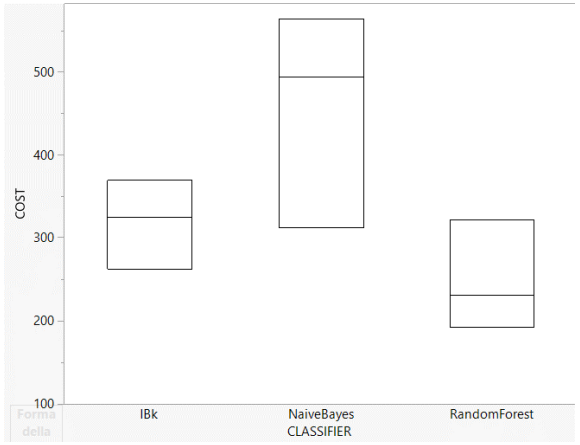
Metodologia:

Classificatori e tecniche di utilizzo

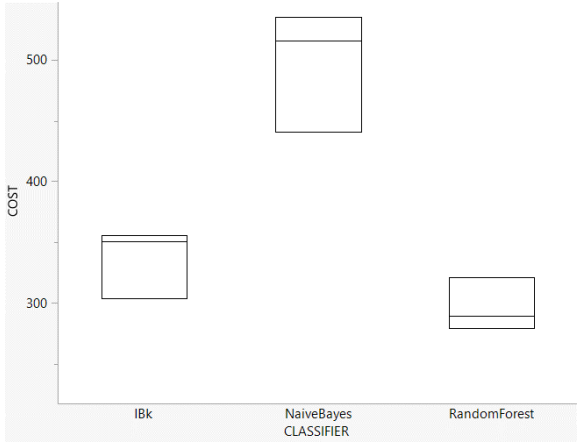
- Verranno considerate tutte le combinazioni tra i classificatori e le tecniche di utilizzo seguenti:
- **Classificatori**:
  - Random Forest
  - Naive Bayes
  - IBk
- **Tecniche di utilizzo**:
  - Nessun filtro
  - Solo feature selection (greedy backward search)
  - Feature selection (greedy backward search) + balancing (undersampling)
  - Feature selection + sensitive learing (CFN=10*CFP)

Risultati:

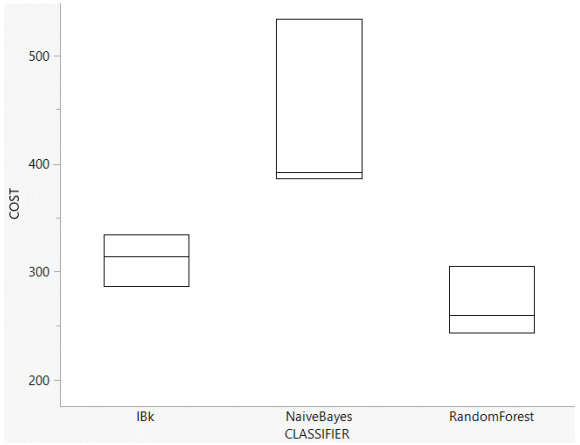
Bookkeeper – confronto costi



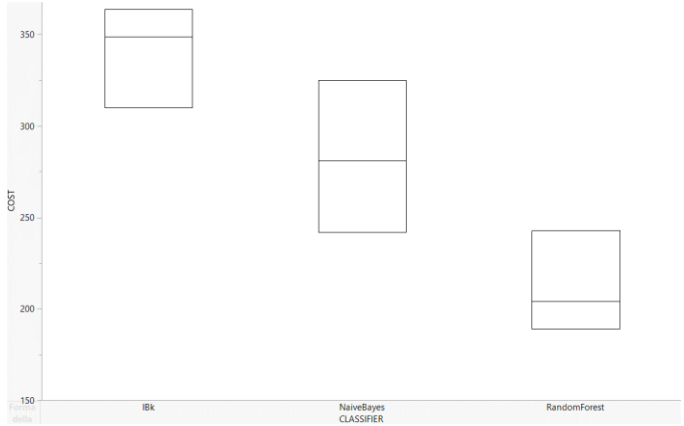
Nessun filtro



Feature selection



Feature selection e balancing

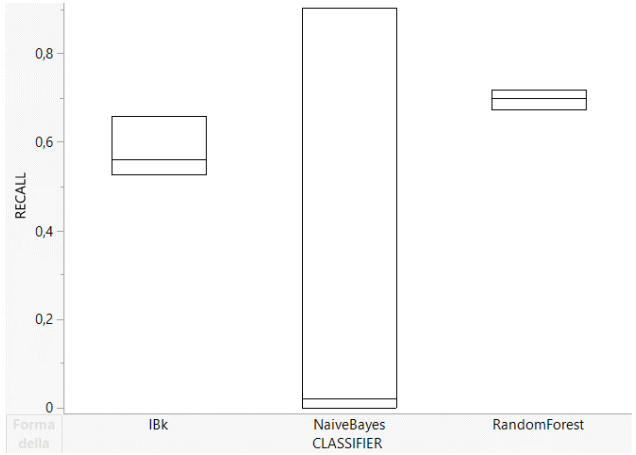


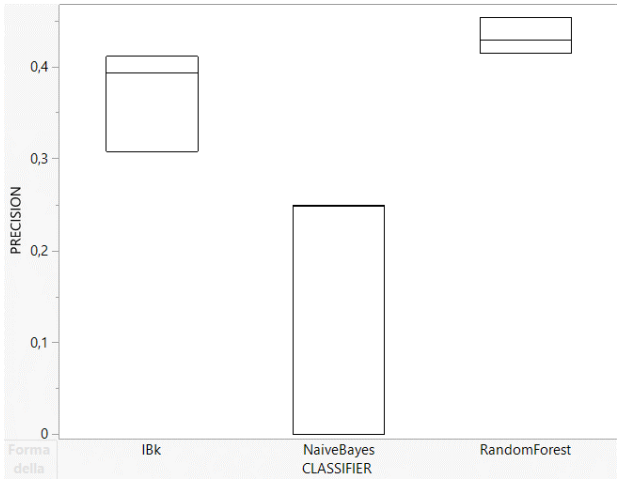
Feature selection e sensitive learning

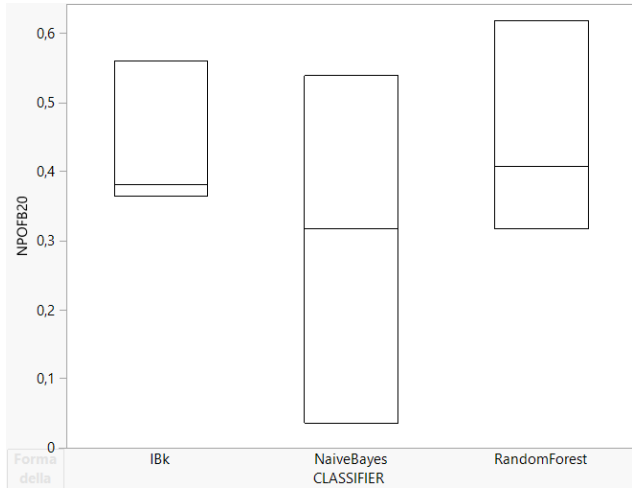
Random forest risulta essere il classificatore con il costo minore.

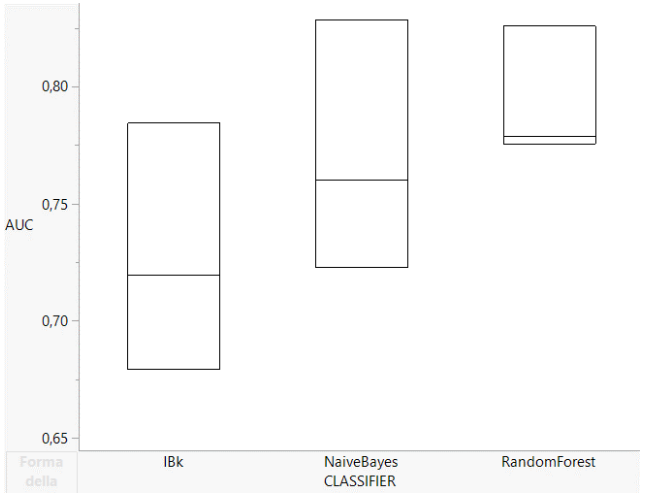
Risultati:

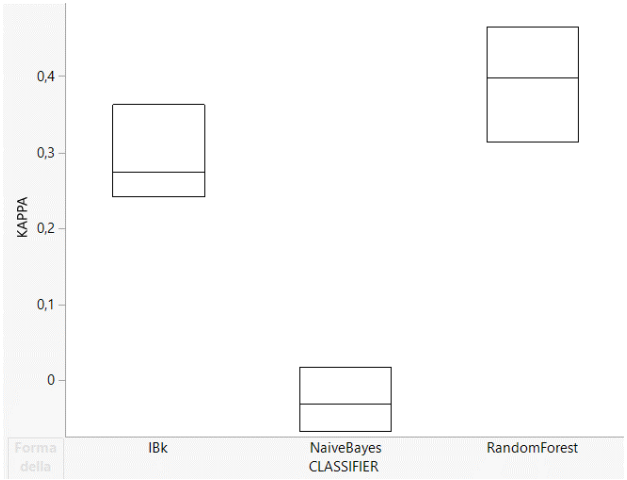
Bookkeeper senza filtri applicati ai classificatori







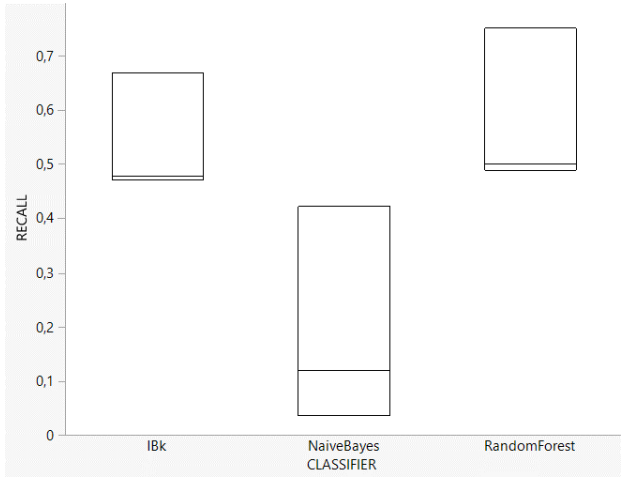


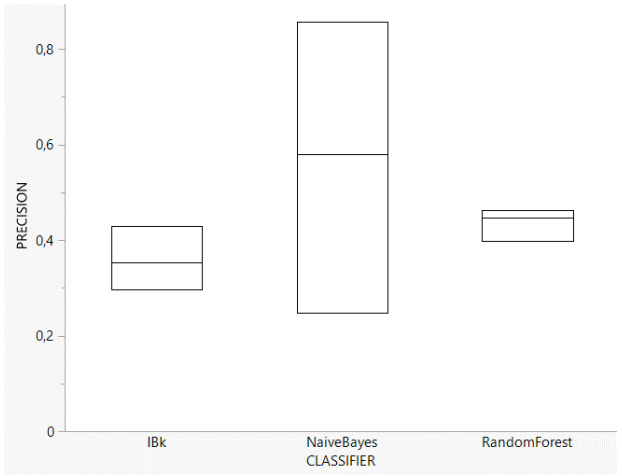


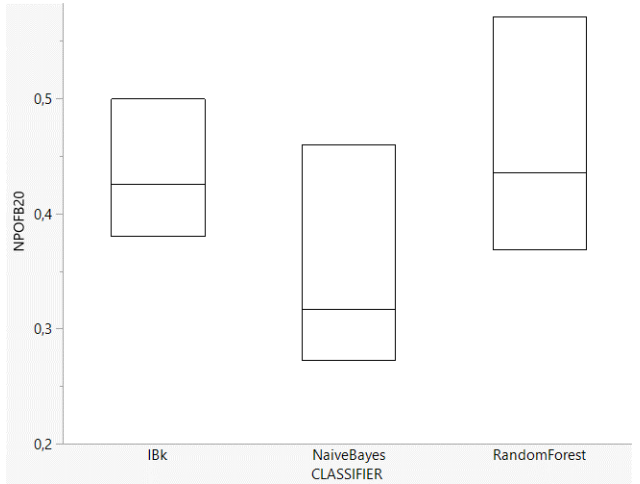
Nel complesso Random Forest sembra comportarsi meglio.

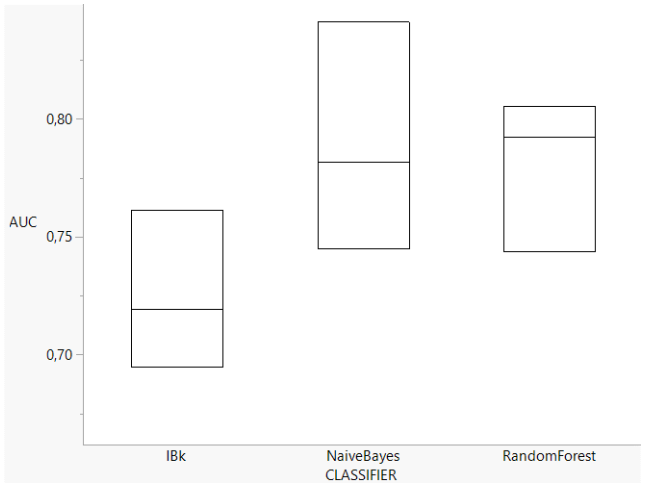
Risultati:

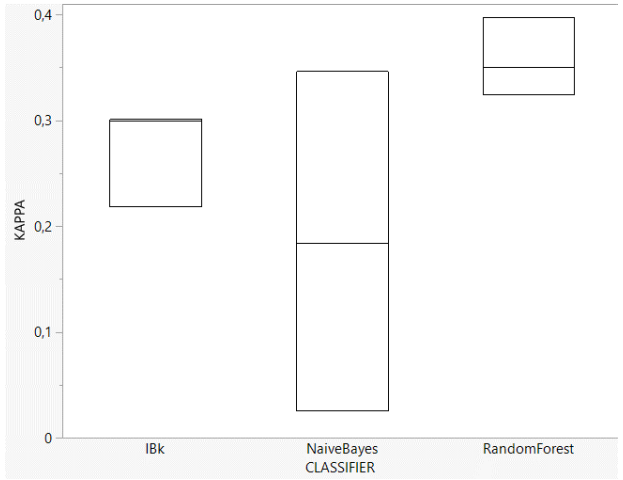
Bookkeeper feature selection







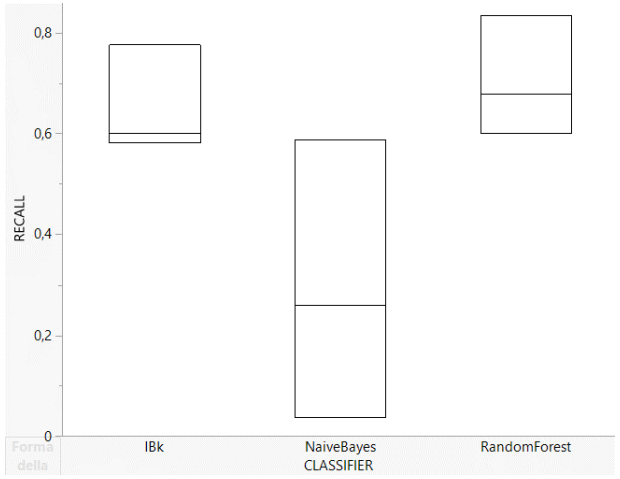


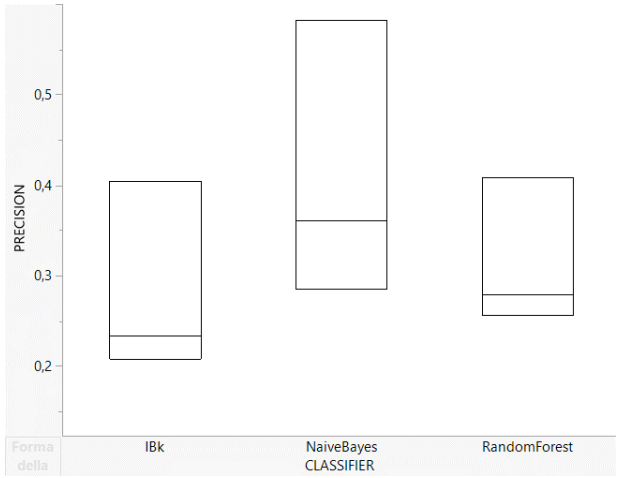


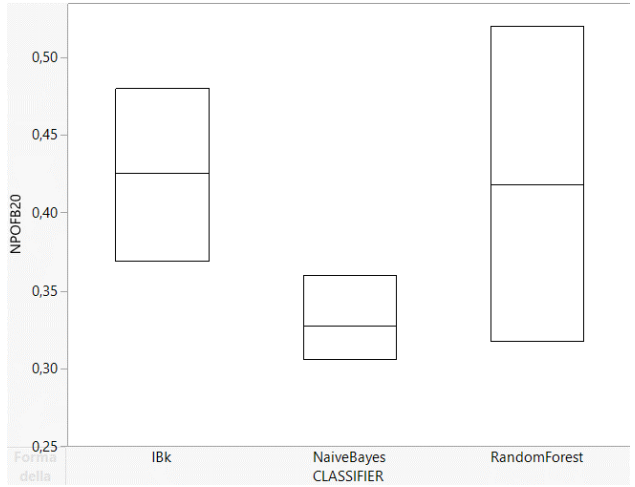
Diminuendo il numero di feature considerate recall e precision diminuiscono leggermente.

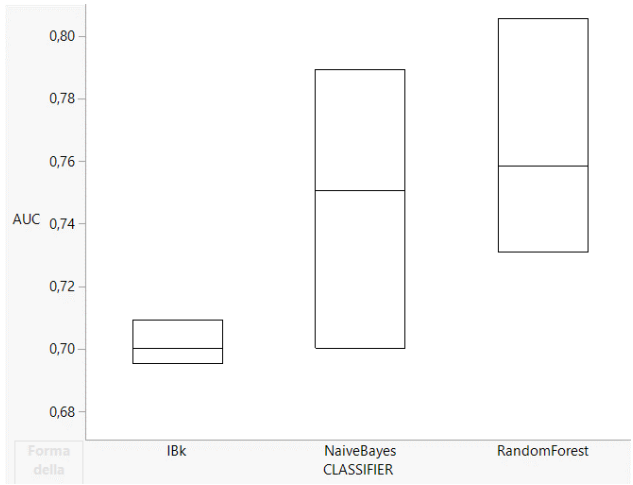
Risultati:

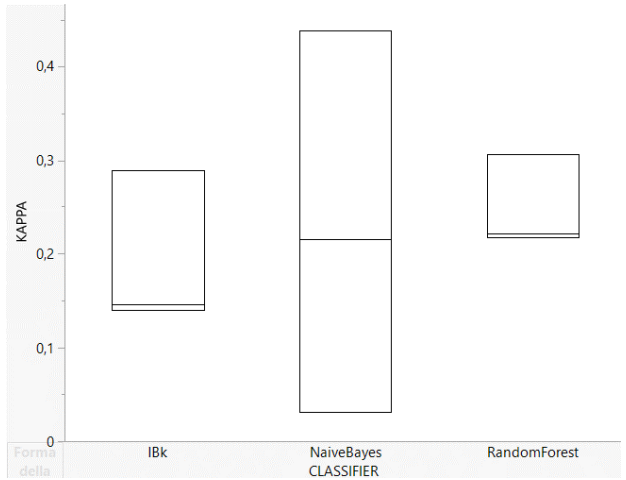
Bookkeeper feature selection e balancing







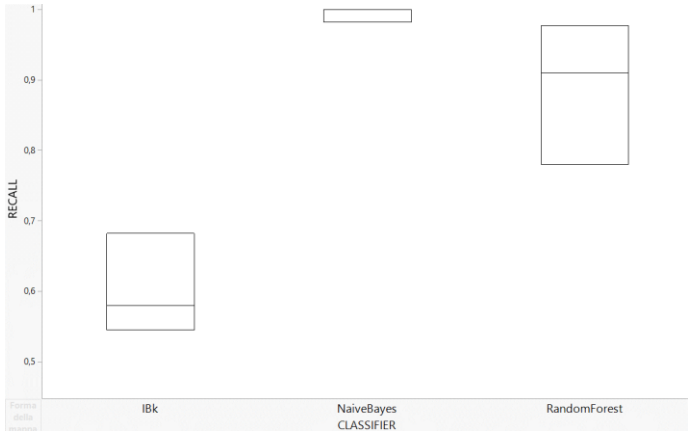


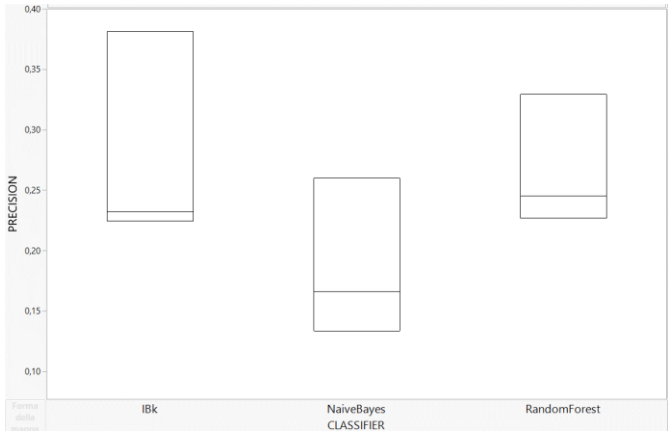


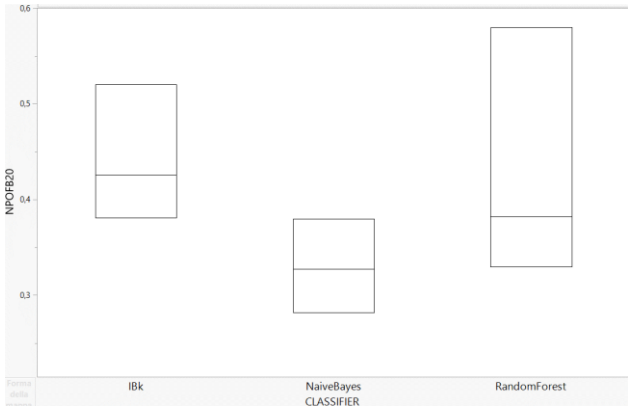
Attraverso il balancing aumenta la recall come da attese poiché aumenta la percentuale di positivi nel training set.

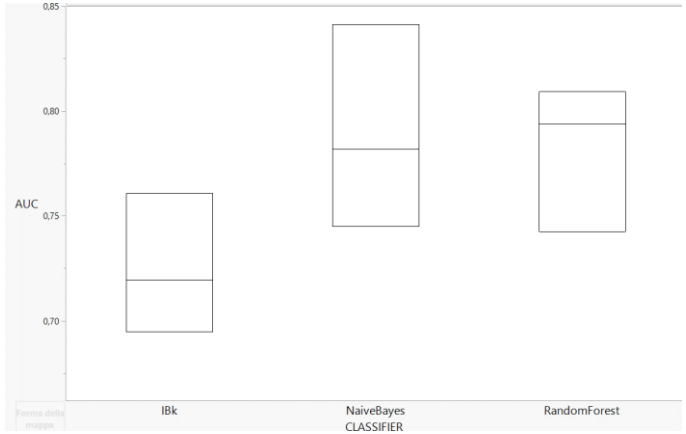
Risultati:

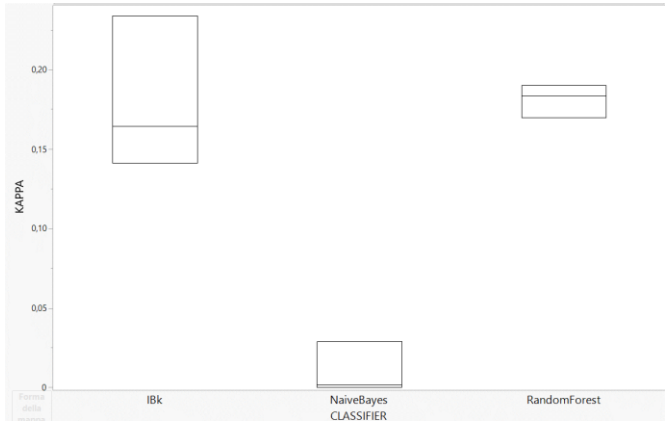
Bookkeeper feature selection e sensitive learning







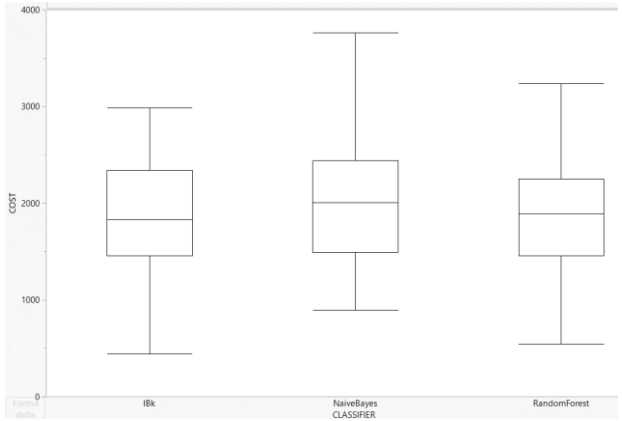




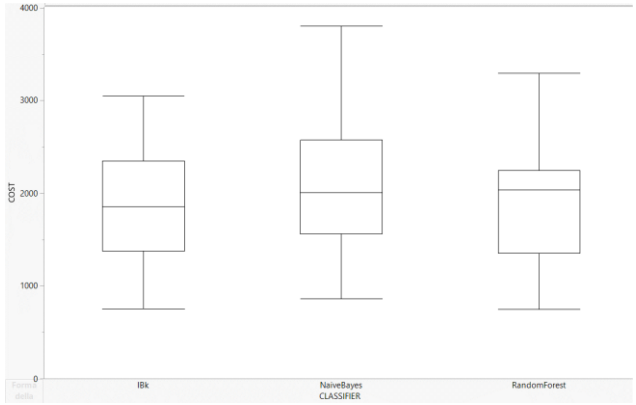
Penalizzando maggiormente FN rispetto a FP aumenta la recall e diminuisce la precision come da attese.

Risultati:

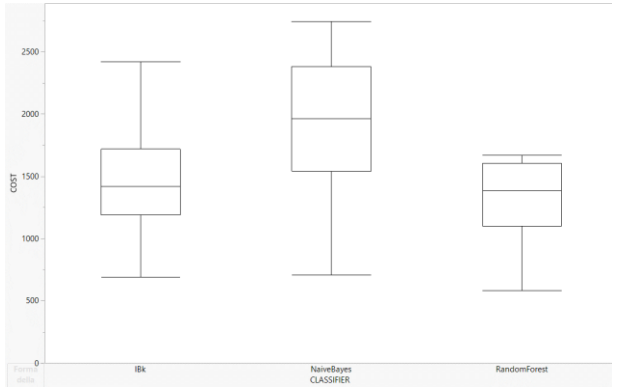
Storm – confronto costi



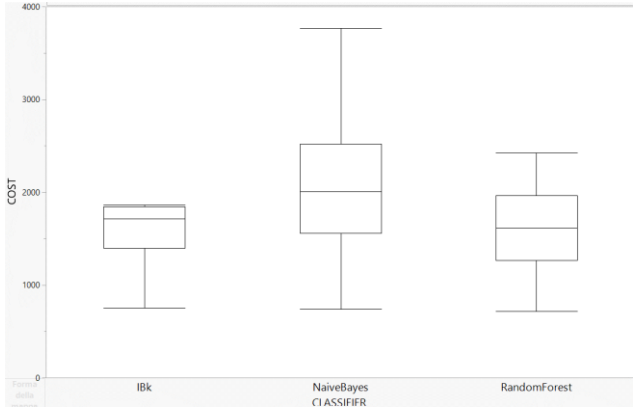
Nessun filtro



Feature selection



Feature selection e balancing

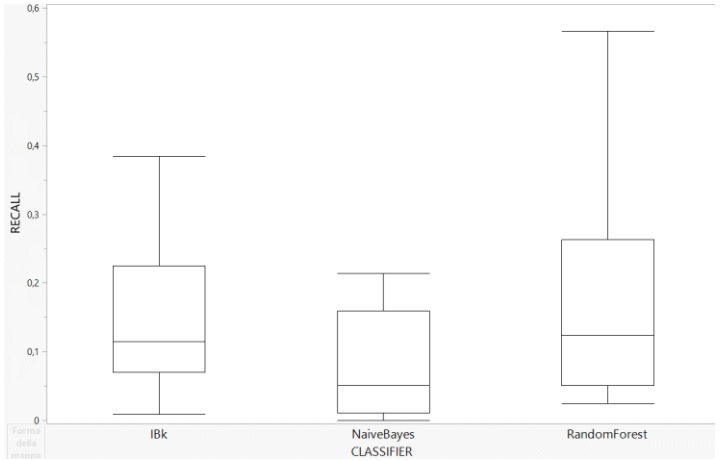


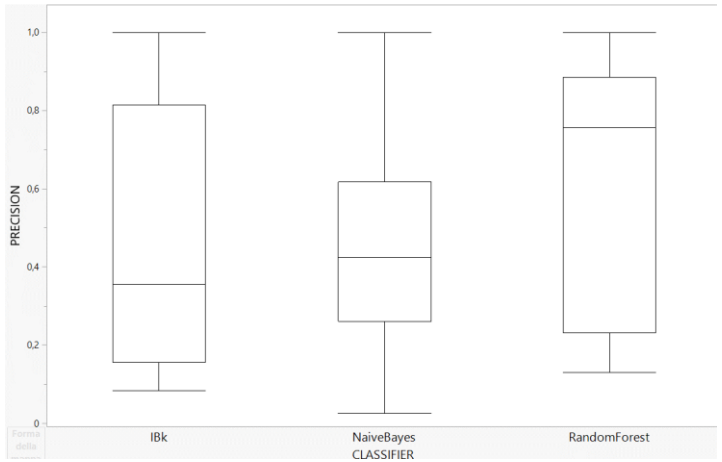
Feature selection e sensitive learning

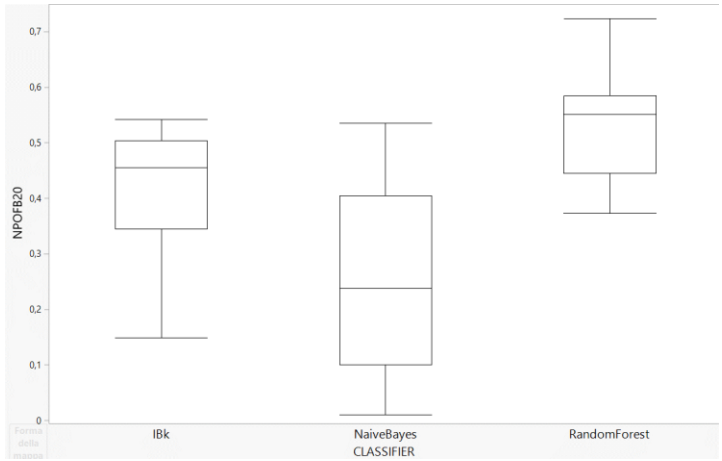
I costi dei classificatori risultano essere simili, soprattutto se si prendono in considerazione IBk e Random Forest.

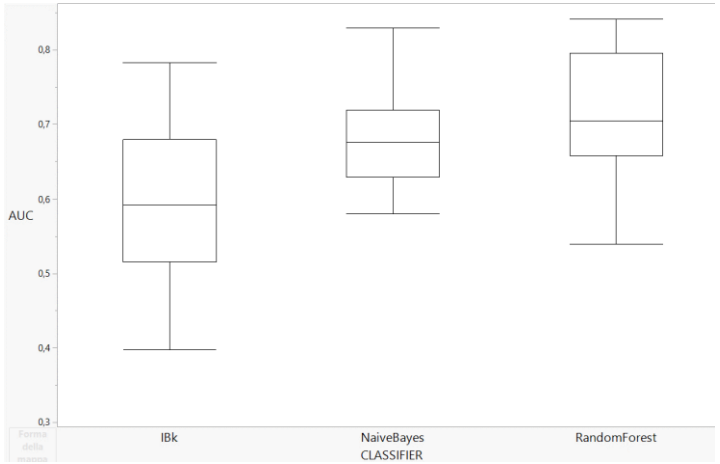
Risultati:

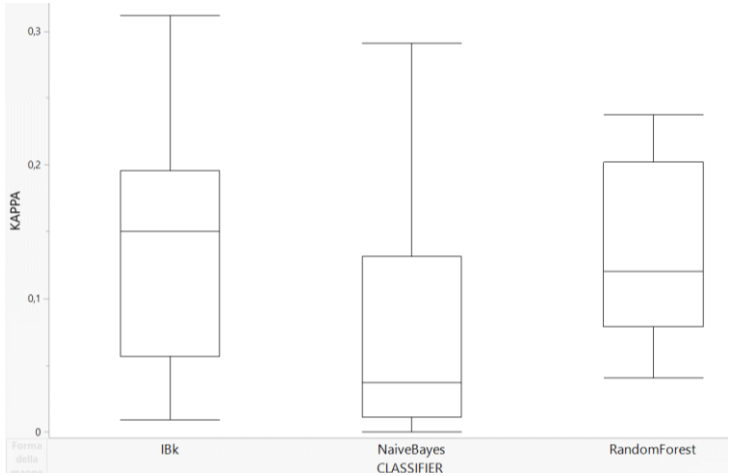
Storm senza filtri applicati ai classificatori







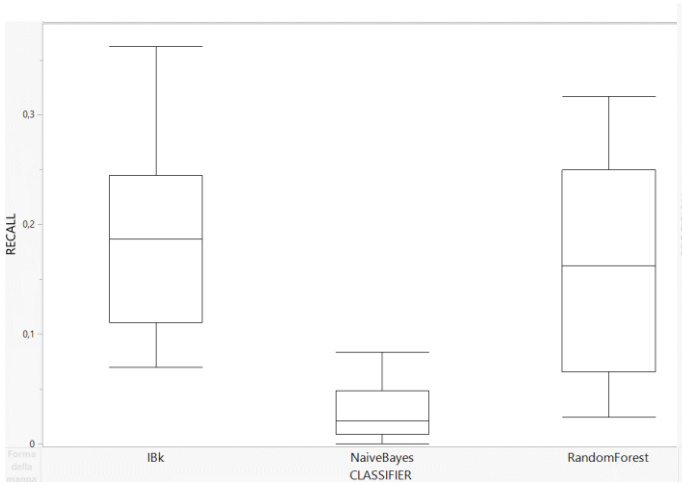


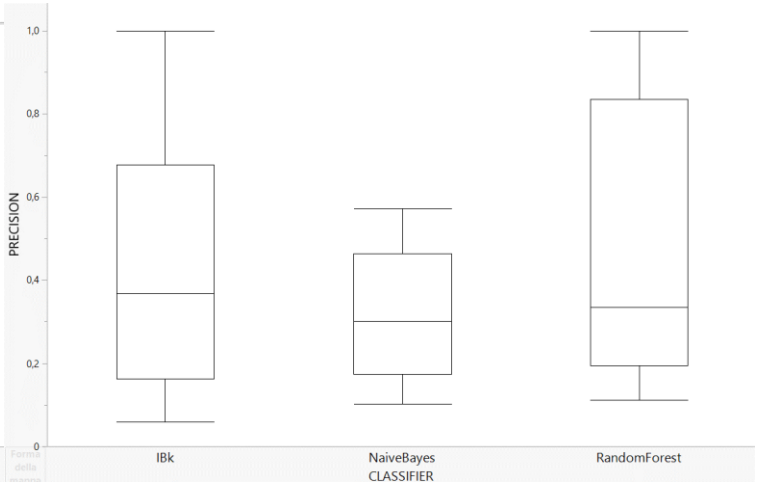


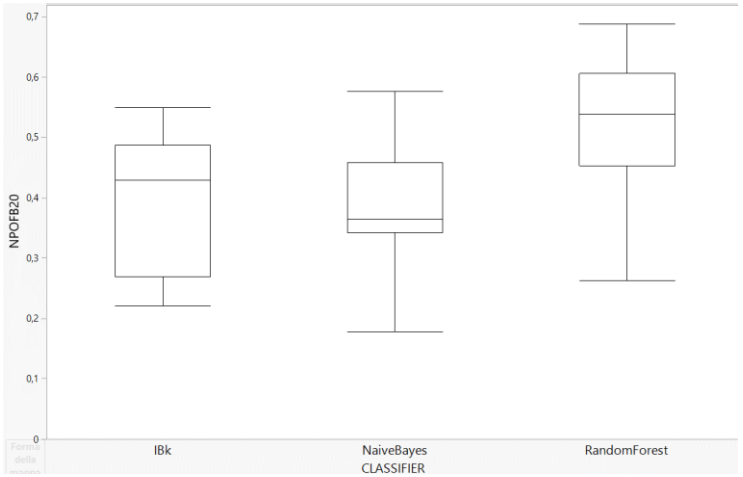
I valori ottenuti sono simili, Random Forest si comporta leggermente meglio.

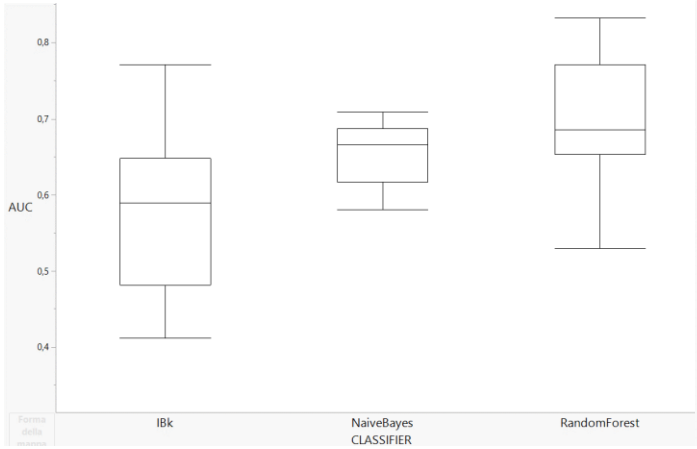
Risultati:

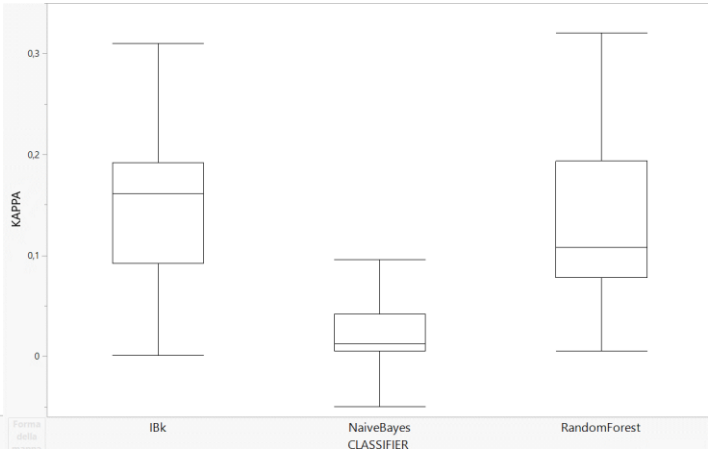
Storm feature selection







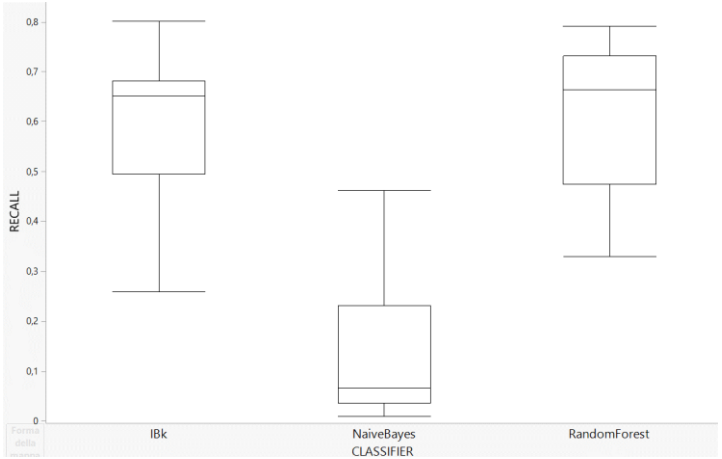


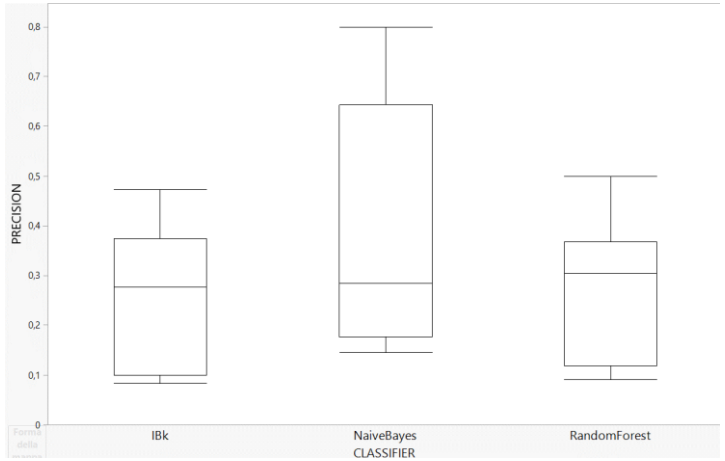


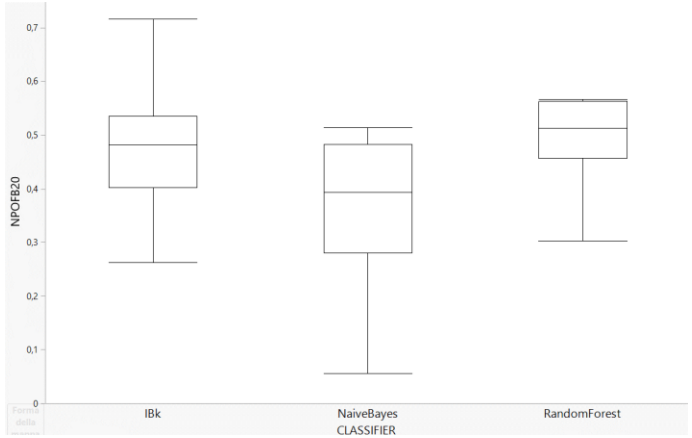
Diminuendo il numero di feature considerate recall e precision diminuiscono leggermente.

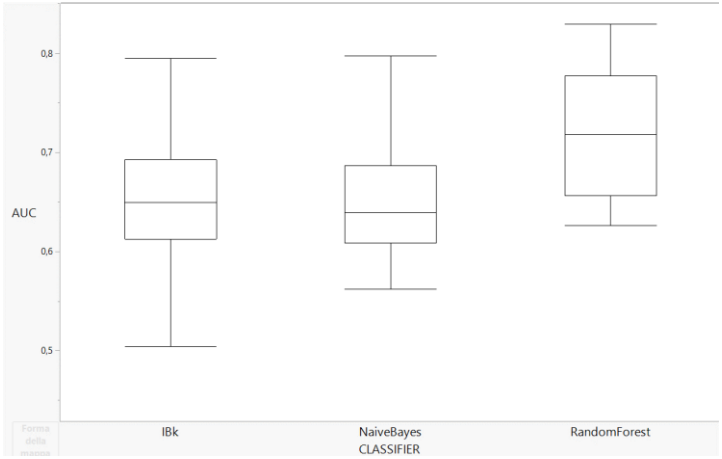
Risultati:

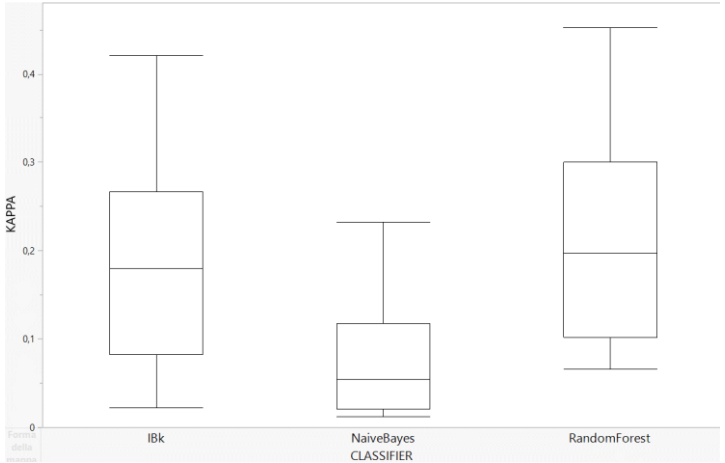
Storm feature selection e balancing







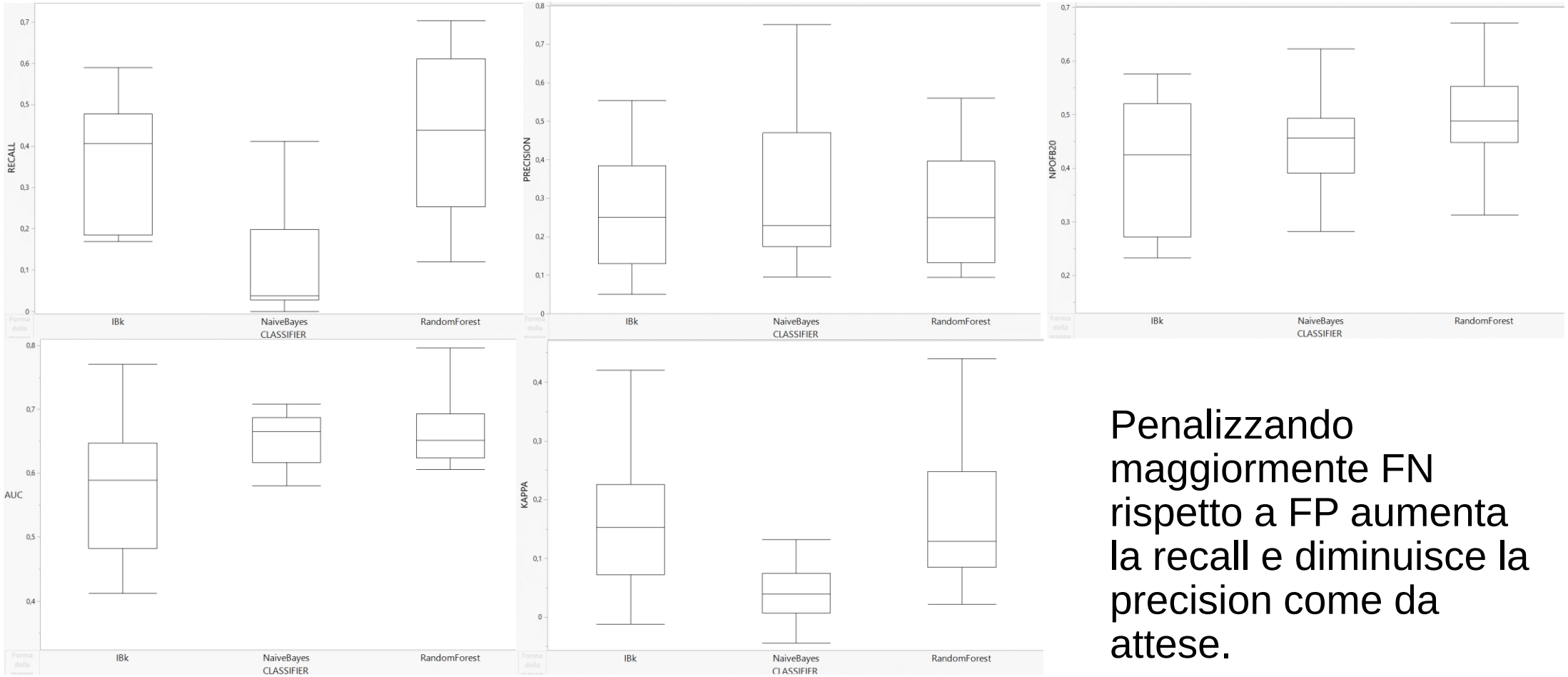




Attraverso il balancing aumenta la recall come da attese poiché aumenta la percentuale di positivi nel training set.

Risultati:

Storm feature selection e sensitive learning



Penalizzando maggiormente FN rispetto a FP aumenta la recall e diminuisce la precision come da attese.

Conclusioni

- Non è possibile eleggere una configurazione migliore in assoluto:
  - **Random Forest** con **feature selection** e **sensitive learning** sembra essere la configurazione migliore per Bookkeeper.
  - **Random Forest** con **feature selection** e **balancing**, **IBk** con **feature selection** e **sensitive learning** sembrano essere le configurazioni migliori per Storm.
- I risultati ottenuti sono coerenti con quanto atteso dalla teoria (es. con **cost sensitive** aumenta la **recall**).
- Le considerazioni formulate per i due progetti sembrano essere compatibili, per cui i risultati possono essere generalizzati per altri progetti.

Minacce alla validità

- In Jira non vengono considerate release senza data.
- Come OV e FV vengono prese le release con data uguale o superiore alla data presente nei campi ‘created’ e ‘resolutionDate’ dei ticket Jira.
- IV è assunto come primo fra gli AV se presenti nei ticket Jira.
- Seguendo il ‘Proportion Paper’ si è presa come threshold per il Cold Start il valore 5 utilizzando come ticket per il calcolo di Proportion solo quelli con gli AV.
- Per il calcolo di Proportion con Cold Start si è ipotizzato che gli altri progetti Apache siano simili a quelli studiati.
- Non sono stati considerati i ticket senza commit associati.
- Si sono assunte come vere le informazioni presenti su Jira come ‘resolutionDate’.

Link

https://github.com/MicheleTosi/BugFinder

https://sonarcloud.io/summary/new_code?id=MicheleTosi_BugFinder