Machine Learning for Software Engineering

MATTEO FANFARILLO – MATRICOLA: 0316179

Agenda

- Contesto e obiettivo
- Metologia:
  - ✓ Come individuare le coppie (classe, release) buggy?
  - ✓ Costruzione del dataset per i classificatori
  - ✓ Metriche considerate
  - ✓ Evaluation dei classificatori
  - ✓ Classificatori e tecniche di utilizzo da confrontare
- Risultati e conclusioni
- Link al repository GitHub del progetto + link SonarCloud

Contesto

- Qualunque progetto nell'ambito dell'Ingegneria del Software prevede un'attività di testing che mira a far emergere e correggere eventuali bug del software.

- Tuttavia l'attività di testing è costosa in termini ore-uomo, per cui non può essere effettuata in modo esaustivo.

- È necessario individuare un sottoinsieme di classi (o di file) del progetto su cui incentrare i test.

Problema: affinché l'attività di testing non perda di efficacia, è necessario individuare le classi che più verosimilmente contengono dei bug. Quale può essere un modo semplice per identificarle?

Contesto (2)

- Per rispondere alla domanda precedente, è opportuno avere un quadro completo di quali classi sono state caratterizzate da bug andati in produzione nel passato, e all'interno di quali release del progetto.

- Con queste informazioni, è possibile sfruttare gli strumenti del Machine Learning per predire quali sono le classi che attualmente possono contenere dei difetti.

- Non è necessario costruire un classificatore apposito per questo scopo. Piuttosto conviene utilizzare i classificatori e le tecniche già esistenti per effettuare la predizione.

Obiettivo

- Dopo aver preso due progetti Apache (BookKeeper e Avro), aver individuato quali loro classi sono state buggy e in quali release, e aver considerato tre classificatori (Random Forest, Naive Bayes e IBk), il nostro scopo è stabilire quale classificatore effettua le predizioni migliori e con quali tecniche di utilizzo.

Le tecniche di utilizzo di un classificatore che andremo a considerare sono la **feature selection**, il **sampling**, la **cost senstivity** e alcune loro combinazioni.

Metodologia: come individuare le coppie (classe, release) buggy?

- **Idea:** ciascun bug è caratterizzato da un ciclo di vita.

Injected version: release in cui il bug è stato introdotto.

Opening version: release in cui il bug è stato rilevato a seguito di una failure del sistema.

Fixed version: release in cui il bug è stato eliminato.

- Come suggeriscono i colori del disegno, le classi direttamente interessate dal bug risultano difettose dall'injected version (inclusa) alla fixed version (esclusa).

- Le informazioni su quali siano la opening version e la fixed version sono sempre disponibili su Jira, poiché gli istanti in cui il bug viene rilevato e risolto sono sempre noti. D'altra parte, non tutti gli issue aperti su Jira sono contrassegnati da un'injected version e, in effetti, non è banale stabilire quale sia.

Metodologia: come individuare le coppie (classe, release) buggy? (2)

Problema: come si può stimare l'injected version di un bug nel momento in cui non è riportato in Jira?

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

- La tecnica per stimare l'injected version dei bug è detta **Proportion**, in cui può essere definita una costante di proporzionalità **p** nel seguente modo:

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

- **p** può essere calcolato per tutti quei bug in cui l'injected version è nota.

Metodologia: come individuare le coppie (classe, release) buggy? (3)

- È possibile sfruttare il valore di p e la formula introdotta nella slide precedente per stimare l'injected version dei bug, ove non nota.

- Si possono sfruttare diverse varianti di Proportion, ciascuna delle quali prevede ipotesi differenti. Nel nostro caso, sono state effettuate le seguenti ipotesi:
  - ✓ È possibile assumere p pressoché costante durante tutto lo sviluppo dei software, per cui sono stati calcolati i valori p_1, p_2,…, p_n di tutti i bug con IV disponibile e, a partire da loro, è stato ricavato un unico p come valor medio.
  - ✓ Nel caso in cui i bug con IV disponibile siano meno di 5, i dati a disposizione sono considerati troppo pochi: in tal caso, è preferibile adottare un approccio Cold Start, in cui i valori p_1, p_2,…,p_n vengono calcolati a partire dai bug degli altri progetti Apache e non dai bug del progetto sotto esame.

- A valle di tali considerazioni, la variante di Proportion utilizzata è **Incremental Train-Test**.

Metodologia: costruzione del dataset per i classificatori

- A questo punto abbiamo individuato le injected version di tutti i bug e, quindi, abbiamo completato il labeling delle classi definendo in quali release ciascuna di esse fosse buggy e in quali no. Per eliminare il fenomeno dello snoring - che consiste nell'avere coppie (classe, release) risultanti non buggy mentre in realtà hanno dei difetti non emersi – è opportuno eliminare tutti i dati relativi alla seconda metà delle release e lavorare solo con i restanti.

- È inoltre necessario assegnare a ciascuna coppia (classe, release) alcune caratteristiche (dette metriche), che verranno sfruttate dai classificatori per effettuare il training dei dati ed essere in grado di fare delle predizioni.
  Dopodiché, ciascuna istanza del dataset da dare in pasto ai classificatori sarà relativa a una coppia (classe, release) e sarà composta da delle feature (o attributi), che consisteranno:
  - ✓ Nelle metriche.
  - ✓ Nel valore da predire, che non è altro che un booleano che indica se la coppia (classe, release) è buggy o meno.

Metodologia: metriche considerate

Nome	Descrizione
Size	Numero di linee di codice.
NR	Numero di revisioni all'interno della singola release.
N_auth	Numero di autori.
LOC_added	Somma delle linee di codice aggiunte sulle revisioni relative alla release.
MAX_LOC_added	Numero massimo delle linee di codice aggiunte in una singola revisione.
AVG_LOC_added	Media delle linee di codice aggiunte sulle revisioni relative alla release.
Churn	Somma di |linee di codice aggiunte – linee di codice rimosse| sulle revisioni relative alla release.
MAX_churn	Valore massimo del churn in una singola revisione.
AVG_churn	Media dei churn sulle revisioni relative alla release.

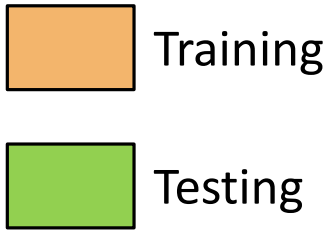
Metodologia: evaluation dei classificatori

- **Reminder:** il nostro obiettivo è stabilire quale classificatore ha le prestazioni migliori e con quali tecniche di utilizzo.

- Di conseguenza, è necessario effettuare una **valutazione** dei classificatori.

- Per farlo, si possono seguire diverse tecniche: noi utilizzeremo **Walk Forward**.

Run \ Release	1	2	3	4	5
1	Testing				
2	Training	Testing			
3	Training	Training	Testing		
4	Training	Training	Training	Testing	
5	Training	Training	Training	Training	Testing



Metodologia: evaluation dei classificatori (2)

- **Attenzione:** Walk Forward è una tecnica di validazione **time-series**, tiene conto cioè dell'ordine temporale dei dati: è impensabile che all'interno del training set vengano sfruttate delle informazioni future rispetto al training set stesso.

- In pratica, se le release 1, 2,…, k appartengono al training set, mentre la release k+1 appartiene al testing set, allora, per i dati che appartengono al training set:
  - ✓ Il calcolo dell'injected version dei bug va rifatto da capo, prendendo come costante di proporzionalità p la media dei p_1, p_2,…, p_h dei soli issue con IV disponibile e FV < k+1.
  - ✓ Anche il labeling delle classi va rifatto da capo, prendendo in considerazione esclusivamente i bug con fixed version < k+1.

- Solo il testing set utilizza una porzione del dataset calcolato inizialmente sfruttando tutti i dati disponibili.

Metodologia: evaluation dei classificatori (3)

- Riassumendo:

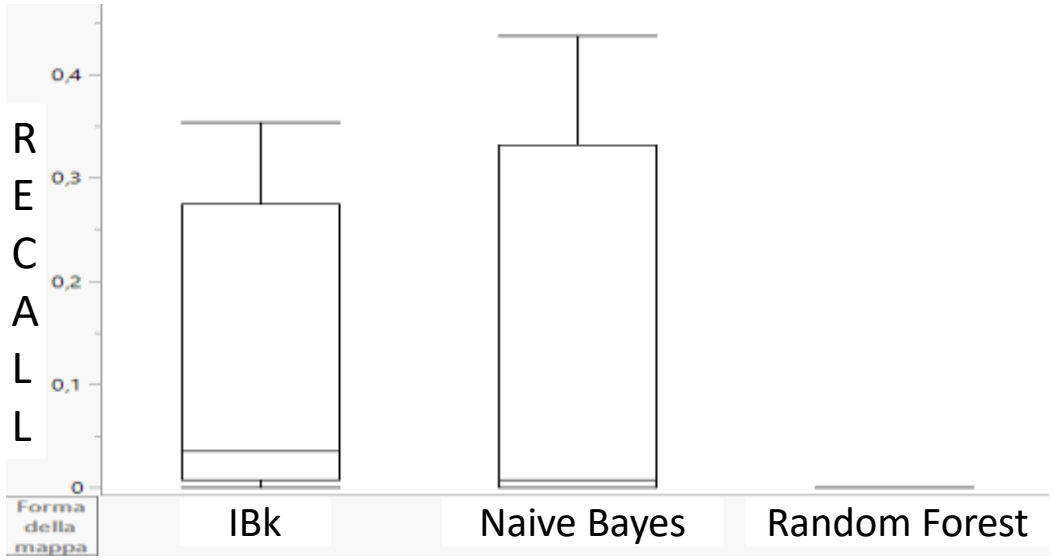
Training set	**Testing set**
✓ Deve essere il più fedele possibile all'utilizzo reale del classificatore. ✓ Non può sfruttare i dati del futuro. ✓ È affetto da snoring.	✓ Deve essere il più fedele possibile alla realtà: idealmente, può anche essere costruito tramite un oracolo. ✓ Deve sfruttare tutti i dati che si hanno a disposizione. ✓ Non è affetto da snoring.

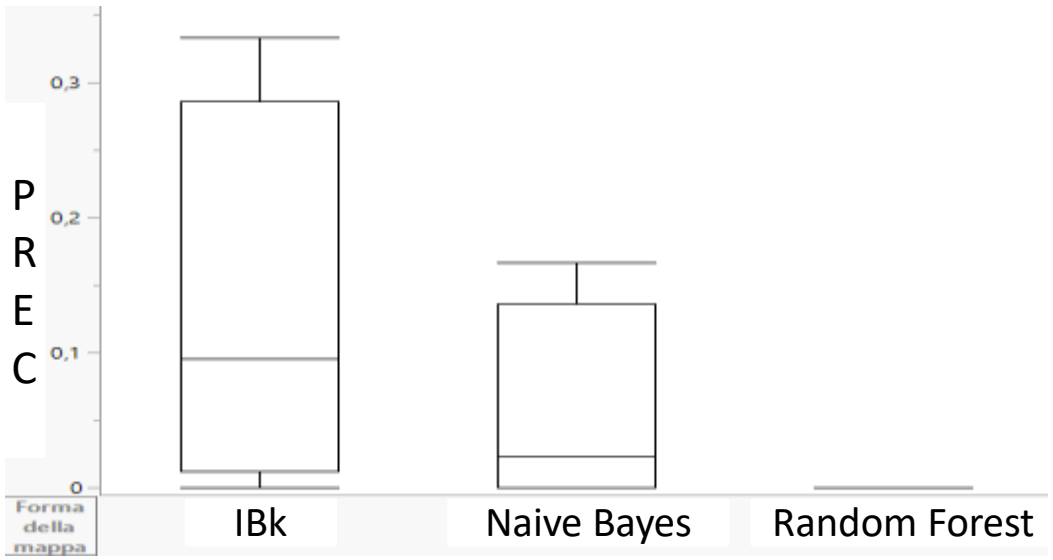
Metodologia: classificatori e tecniche di utilizzo da confrontare

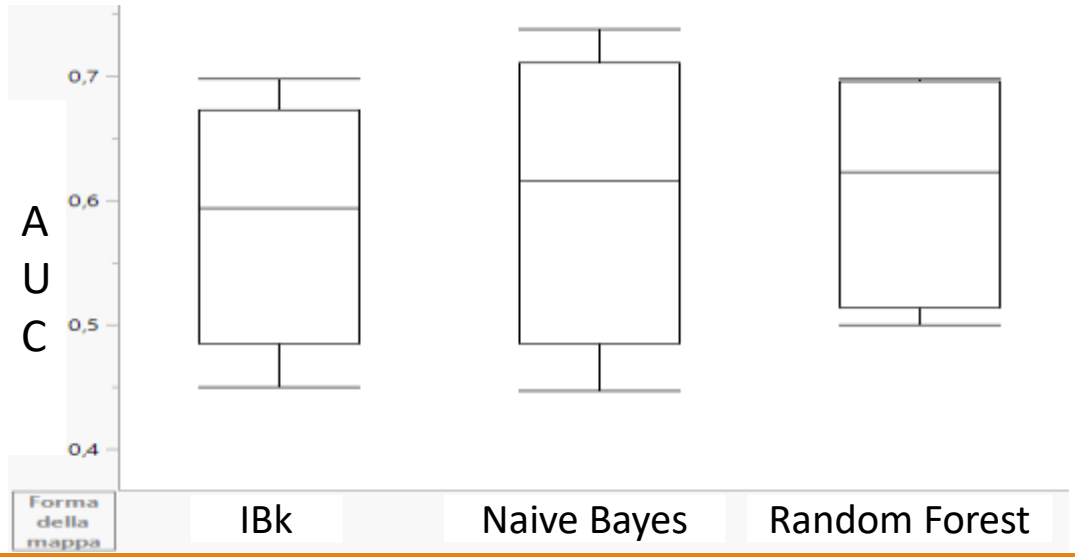
- Verranno considerate tutte le combinazioni tra i classificatori e le tecniche di utilizzo riportate in tabella:

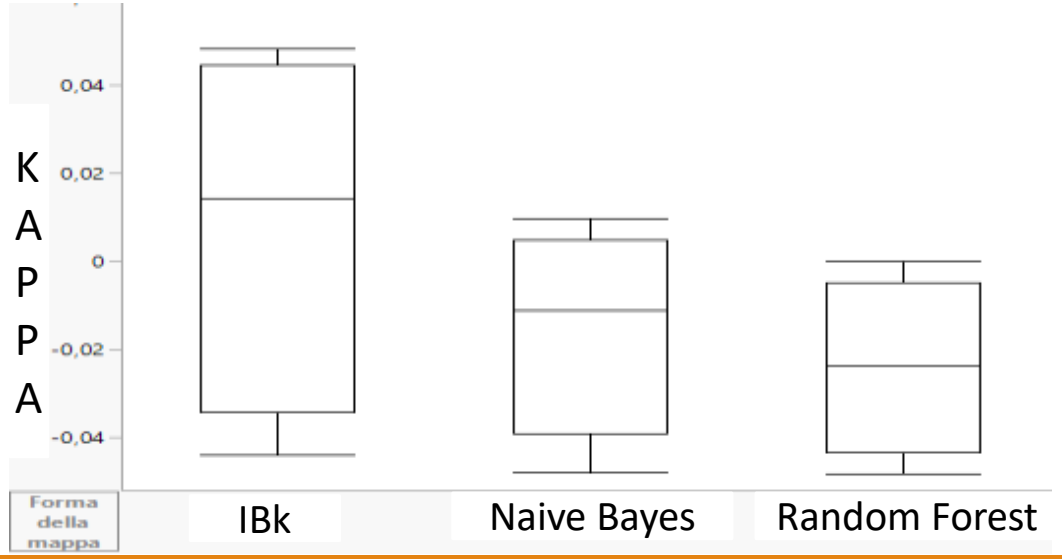
Classificatori	**Tecniche di utilizzo**
✓ Random Forest ✓ Naive Bayes ✓ IBk	✓ Nessun filtro ✓ Solo feature selection (greedy backward search) ✓ Feature selection (greedy backward search) + balancing (undersampling) ✓ Feature selection + sensitive learning (CFN = 10*CFP)

Risultati: BookKeeper – nessun filtro applicato ai classificatori



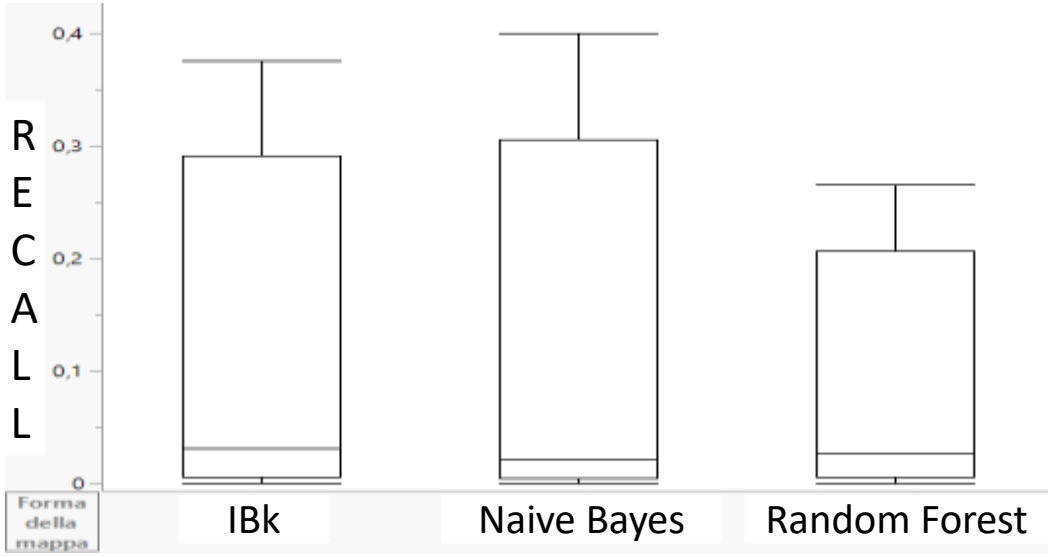


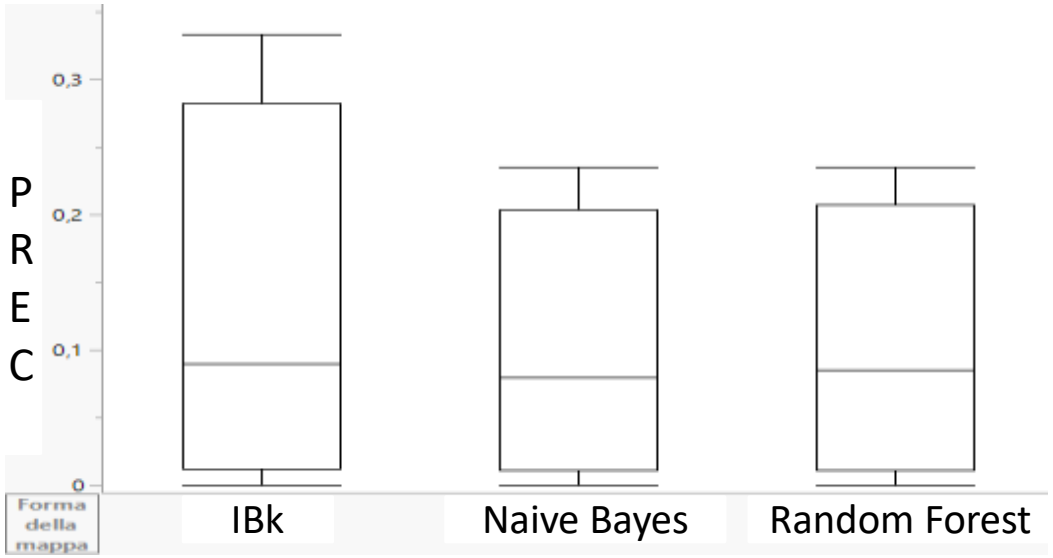


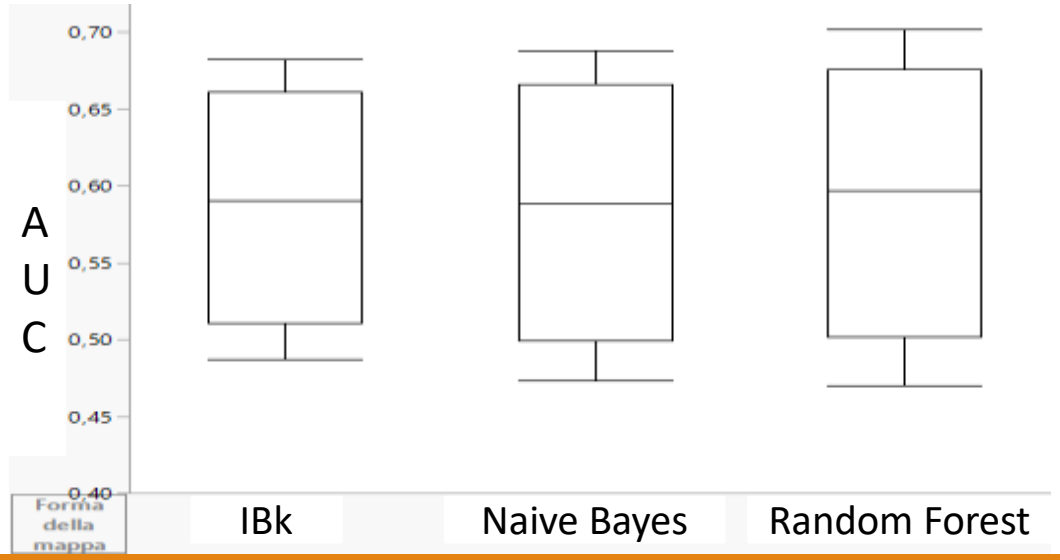


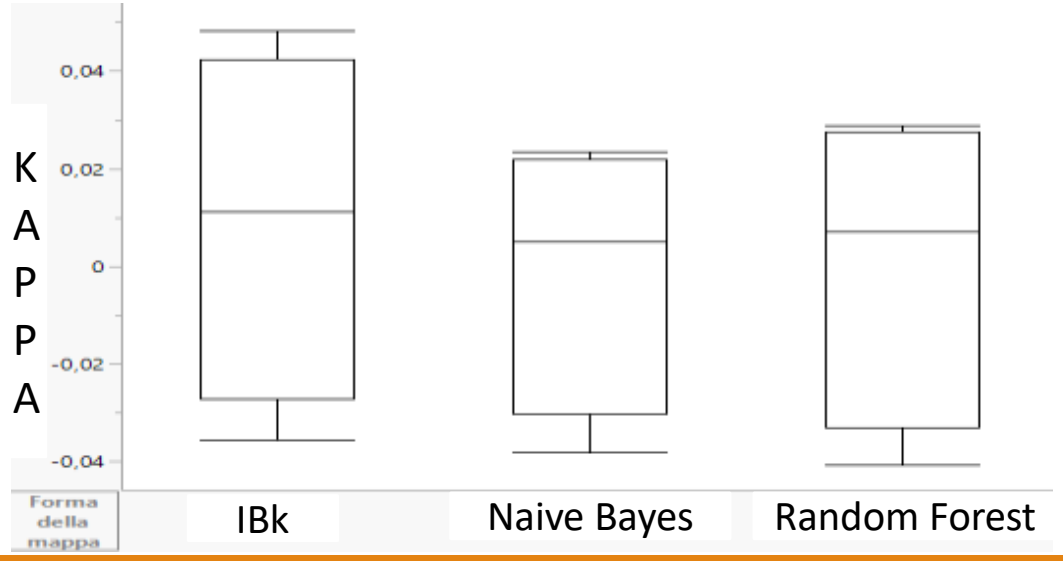
Nel complesso IBk sembra comportarsi meglio.

Risultati: BookKeeper – solo feature selection



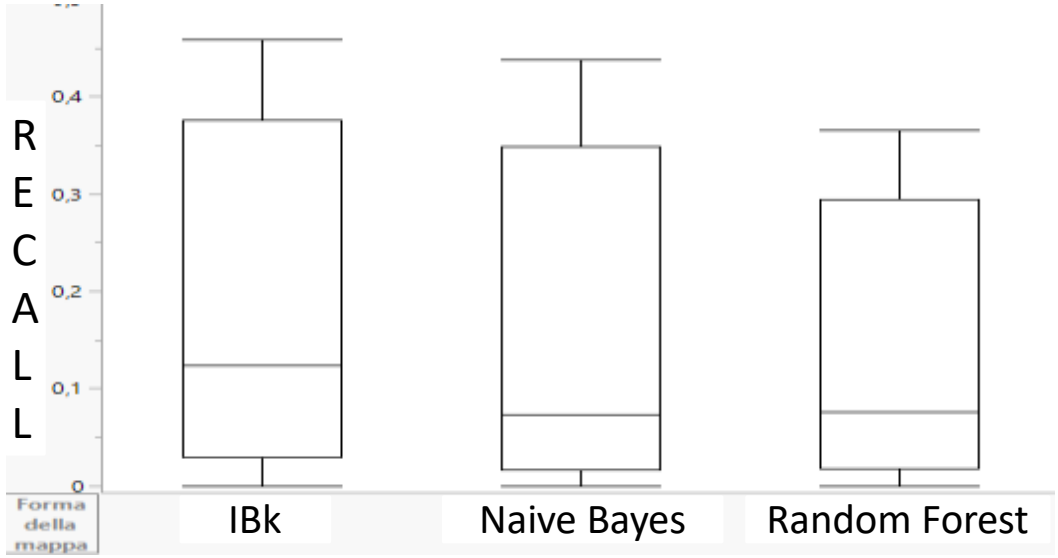


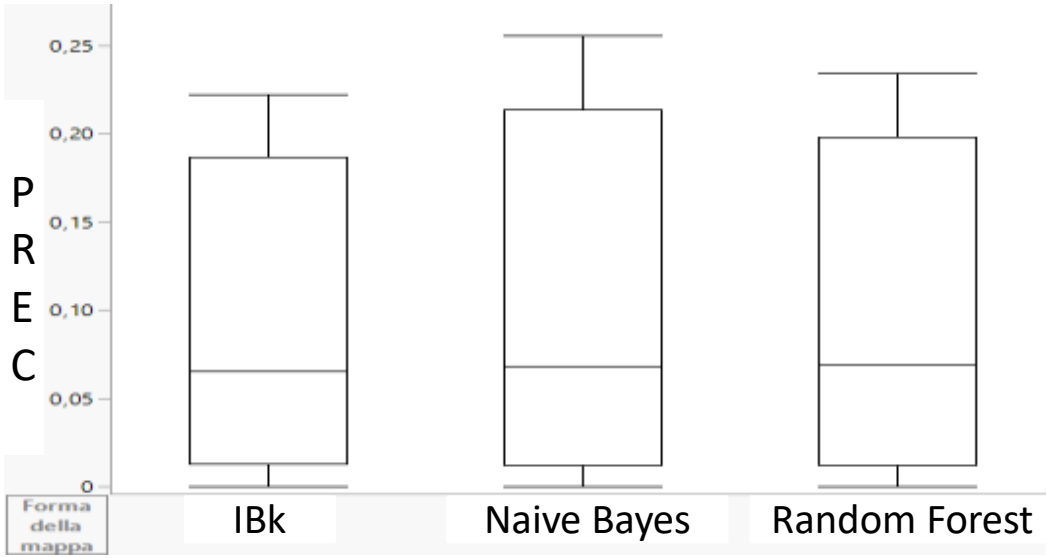


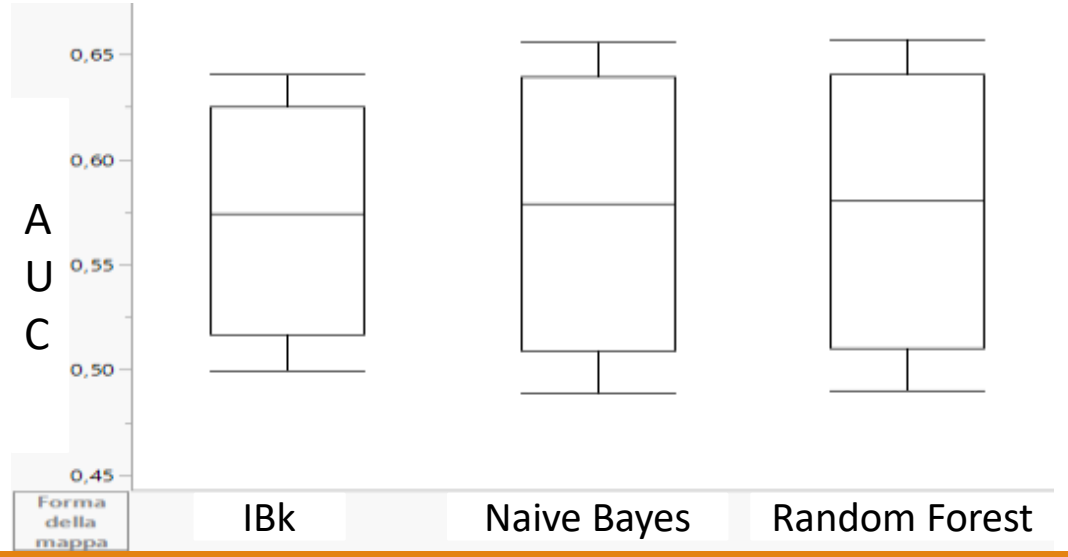


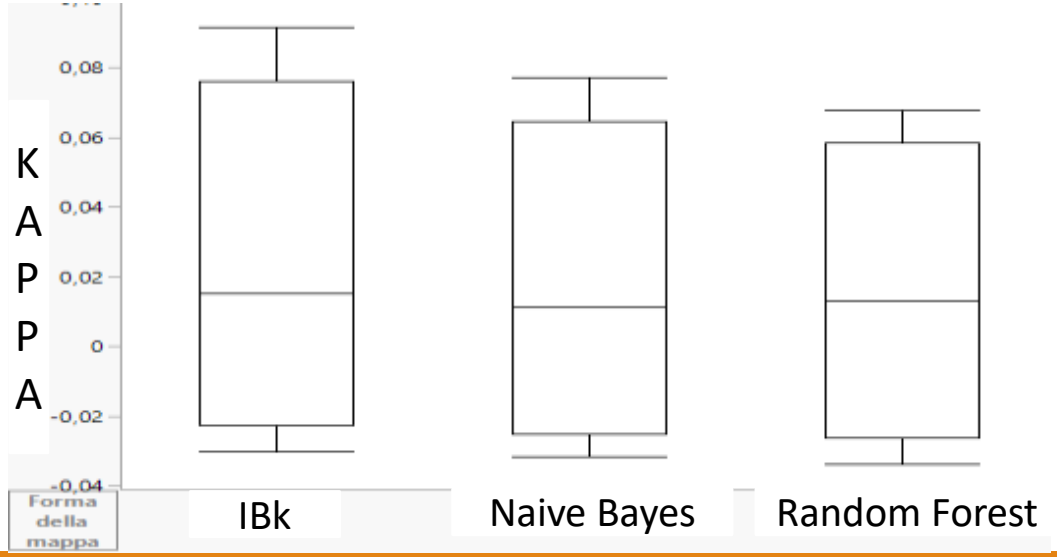
Nel complesso IBk sembra comportarsi meglio.

Risultati: BookKeeper – feature selection & undersampling



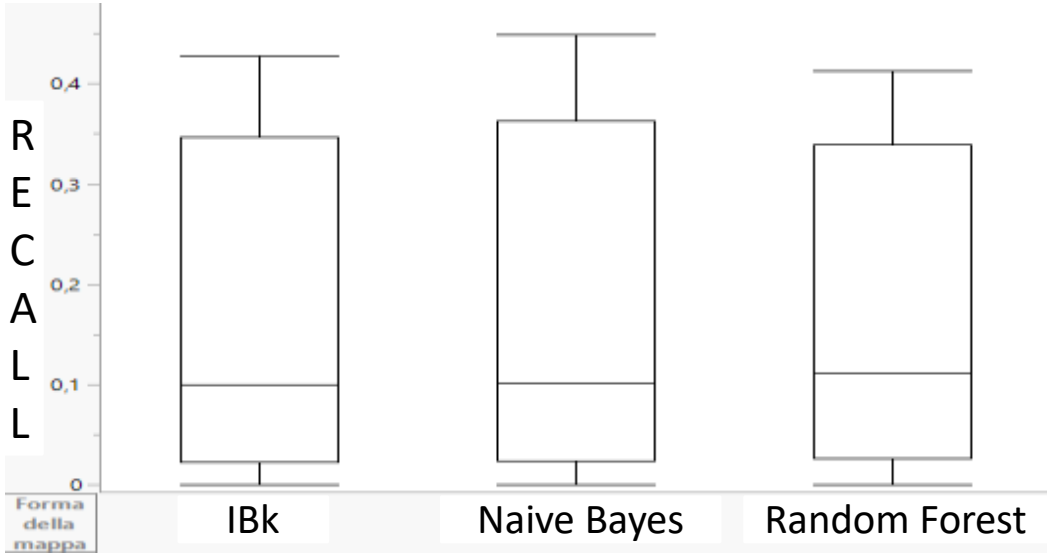


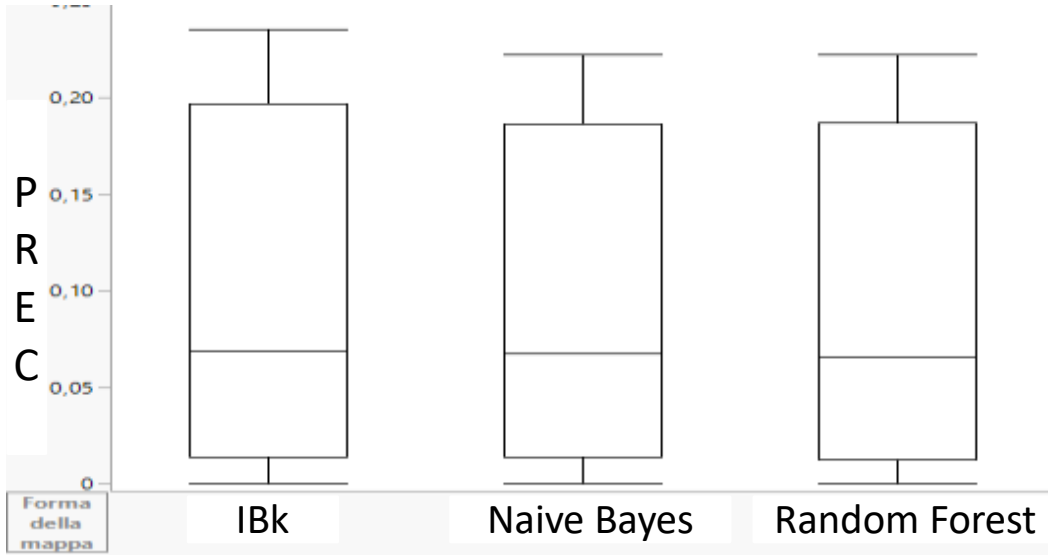


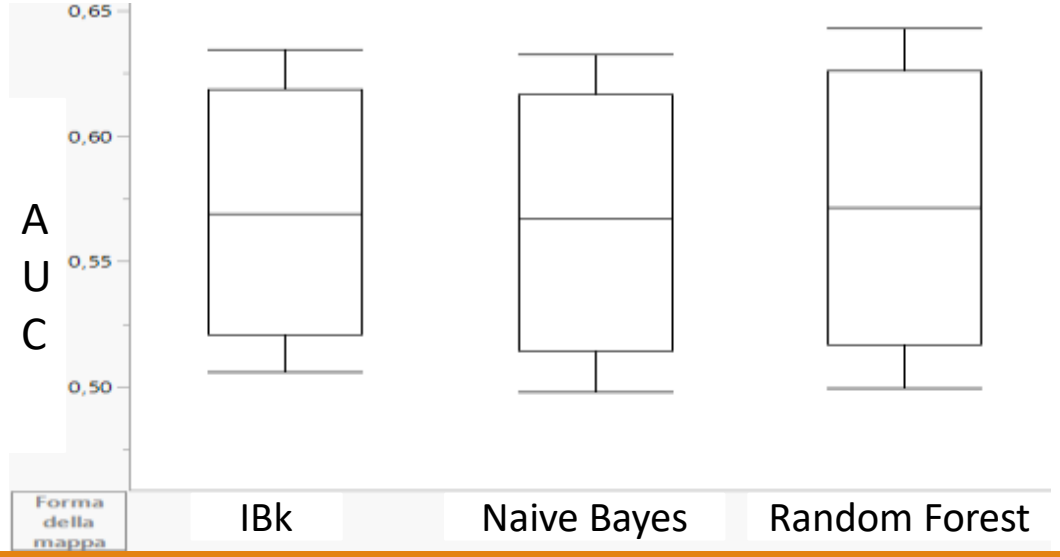


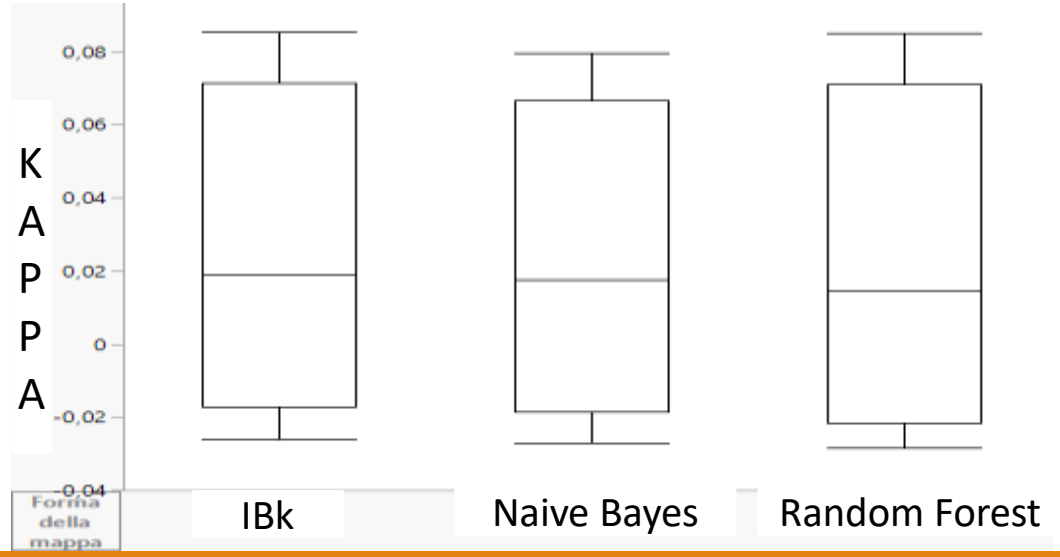
Nel complesso IBk sembra comportarsi meglio.

Risultati: BookKeeper – feature selection & sensitive learning









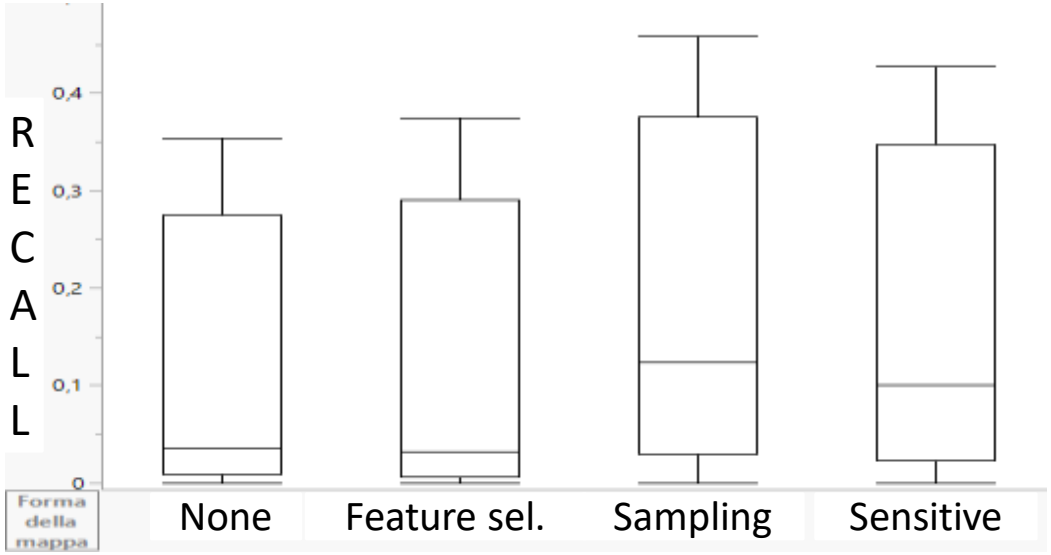
Nel complesso i tre classificatori presentano dei risultati molto simili.

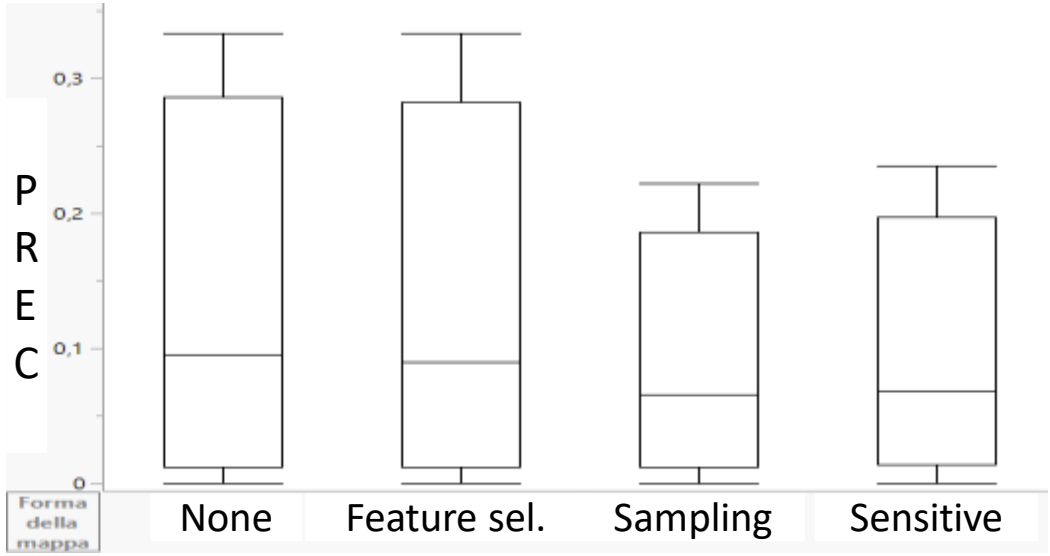
Risultati: BookKeeper – prime considerazioni

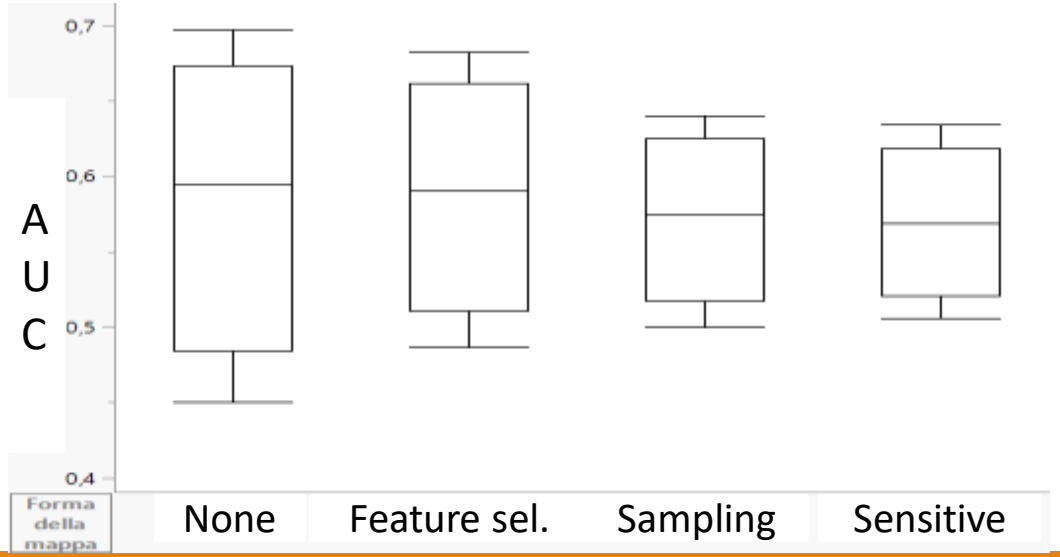
- In base ai box-plot mostrati nelle slide precedenti sembra che, in ogni caso, il classificatore IBk assuma un comportamento migliore (o comunque non peggiore) degli altri due.

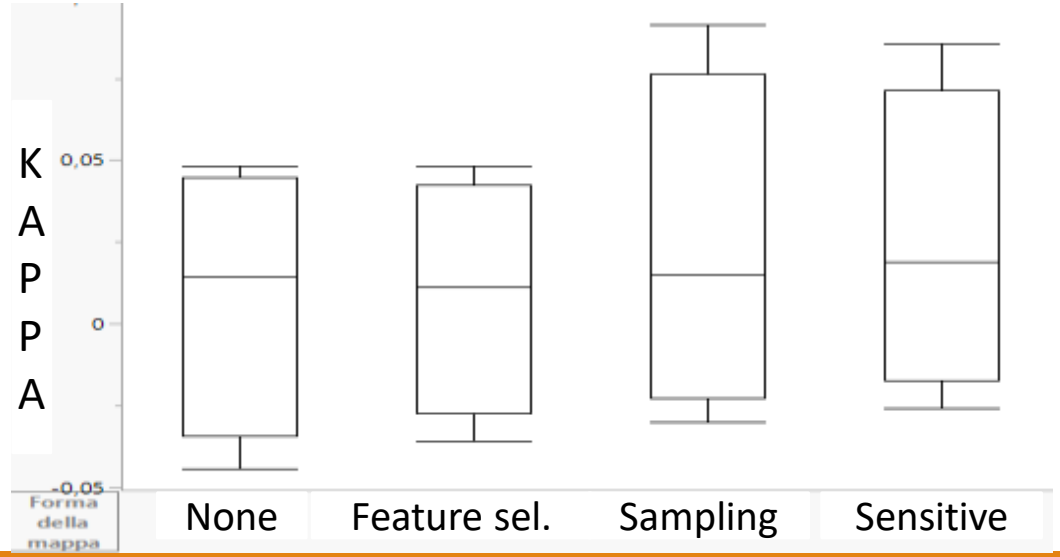
- Per capire dunque quale sia la combinazione (classificatore, tecnica di utilizzo) migliore, confronteremo le distribuzioni delle varie metriche (recall, precision, AUC, kappa) sulle diverse tecniche di utilizzo per il classificatore IBk.

Risultati: BookKeeper – classificatore IBk









Nel complesso l'undersampling e il sensitive learning risultano convenienti da usare assieme alla feature selection.

Risultati: BookKeeper – conclusioni

- L’utilizzo di nessun filtro e la feature selection da sola danno una precision più alta. Dall’altro lato, le combinazioni feature selection + undersampling e feature selection + sensitive learing mostrano una recall e un kappa più elevati.

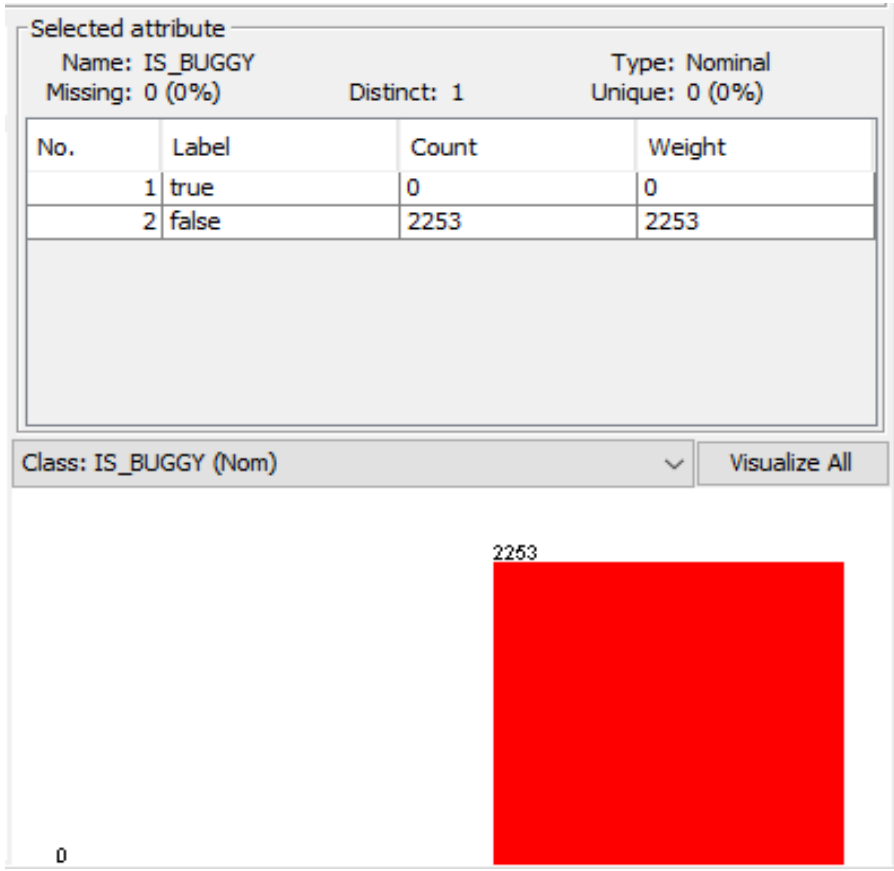
- Questo vuol dire che:
  - ✓ L’utilizzo di nessun filtro e la feature selection da sola danno meno falsi positivi ma più falsi negativi.
  - ✓ Nel complesso, le combinazioni feature selection + undersampling e feature selection + sensitive learing dimostrano di saper effettuare delle predizioni migliori (grazie al valore di kappa statisticamente più elevato).

- Poiché un falso negativo identifica un errore più grave di un falso positivo, l’uso di undersampling o di sensitive learning assieme alla feature selection, col classificatore IBk, rappresenta la soluzione migliore per effettuare delle predizioni sulla buggyness delle classi di BookKeeper.

Risultati: Avro

- Purtroppo in Avro non è possibile effettuare un'analisi dei risultati come in BookKeeper poiché il training set di tutte le iterazioni del Walk Forward presenta esclusivamente istanze non buggy, il che rende impossibili le predizioni.

La figura (estratta da Weka) mostra come all'ultima iterazione del Walk Forward (che sarebbe la più aggiornata) ci siano 2253 istanze non buggy e 0 istanze buggy.



Risultati: Avro (2)

- Questo fenomeno dimostra che Avro è un progetto altamente caratterizzato dallo snoring: in particolare, tutti i bug che hanno interessato le prime release del progetto sono stati individuati e poi risolti solo nelle ultime versioni.

- In conclusione, per tale progetto, l'injected version, la opening version e la fixed version dei vari bug tendono a essere molto distanti tra loro.

Link

- **GitHub:** Fanfarillo/isw2-deliverable (github.com)
- **SonarCloud:** isw2_deliverable - Fanfarillo (sonarcloud.io)