

Machine Learning For Software Engineering

Marco Lorenzini – 0353515



Agenda

01

Introduzione

02

Obiettivo

03

Progettazione

04

Risultati

05

**Assunzioni &
Minacce alla validità**

06

Conclusioni

01

Introduzione – cosa è il testing?

Il testing è un processo di convalida del software effettuato attraverso metodi sistematici (white-box, black-box) per individuare incongruenze tra specifica formale, implementazione e comportamento atteso.

Ma quanto costa?

Tempo di sviluppo



20-40% del tempo di sviluppo è dedicato al testing

Costo medio annuo



23% del budget annuale IT di un'organizzazione è dedicato al testing

02

Obiettivo

Ottimizzare il testing identificando classi ad alto rischio di bug tramite l'utilizzo di Modelli di **Machine Learning**.

Questo ci permette di capire come è meglio indirizzare budget e tempo per lo sviluppo di casi di testing.

Lo studio si concentra sull'analisi dei risultati ottenuti su due progetti open-source della Apache Software Foundation: **BookKeeper** e **Avro**.



03

Progettazione – Raccolta dei Dati

Assunzione principale: il **futuro** è simile al passato.



Tramite **Jira** estraiamo i **ticket** riferiti a **bug** risolti.



Tramite **Git** estraiamo i **commit** relativi ai **ticket** estratti.

Ora sappiamo quali **classi** sono state modificate

03

Progettazione – Individuare i bug

Idea principale: ogni **bug** ha un ciclo di vita rappresentabile come:



Injected Version (IV): versione in cui il bug viene introdotto in una classe

Opening Version (OV): versione in cui il bug viene individuato

Fixed Version (FV): versione in cui il bug viene risolto

03

Progettazione – Proportion

Punto di partenza: una classe è difettosa dalla **IV** fino alla **FV** (esclusa).

Mentre la **OV** e la **FV** sono facilmente individuabili dai ticket di **Jira**, questo non vale per l'**IV** che spesso non è definita, come comportarci?

Idea: il tempo che intercorre tra **FV** e **OV** è proporzionale al tempo tra **FV** e **IV**.
Più il bug è stato difficile da individuare, più sarà il tempo necessario per risolverlo.

Proportion:

$$p = \frac{FV - IV}{FV - OV} \Rightarrow IV = FV - p * (FV - OV)$$

03

Progettazione – Proportion

Utilizziamo, in base alla situazione, due diverse tecniche:

Proportion Increment

- Quando?

La usiamo se il numero di ticket con IV a nostra disposizione è ≥ 5 .

- Come?

Calcolo **p** sfruttando i ticket raccolti fino a quel momento.

Proportion Cold Start:

- Quando?

La usiamo se il numero di Ticket con IV a nostra disposizione è < 5 .

- Come?

Calcolo **p** basandomi su altri progetti Apache e lo imposto pari alla mediana dei valori.

03

Progettazione – Metriche

LOC: Numero di righe di codice.

Authors Number: Numero di autori della classe.

Number of Revisions: Numero di revisioni.

Touched Loc: Somma di righe aggiunte e rimosse nelle revisioni.

Total Added Lines: Numero di righe aggiunte nelle revisioni.

Average Added Lines: Numero medio di righe aggiunte nelle revisioni.

Max Added Lines: Numero massimo di righe aggiunte nelle revisioni.

Churn: Valore assoluto della differenza tra linee aggiunte e rimosse.

Max Churn: Churn massimo nelle revisioni.

Number Fix: Numero di bug fixati.

Cyclomatic Complexity*: Numero di cammini indipendenti nella classe.

Average Churn: Churn medio nelle revisioni.

Days Between Commits*: Numero di giorni tra un commit e l'altro.

* = Metriche Personalizzate

03

Progettazione – Training e Testing Set

Nel **training set** la buggyness delle classi viene determinata utilizzando solo i ticket fino alla release considerata. Le classi vengono etichettate come buggy se sono state modificate in commit associati a bug con una **injected version** precedente. Più **realistico** possibile, affetto da **snoring**.

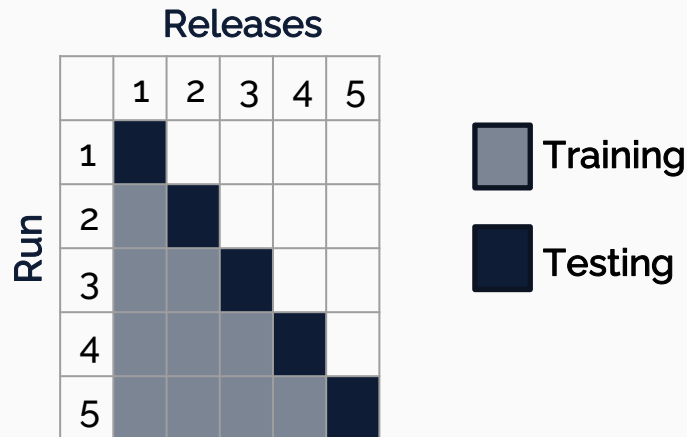
Nel **testing**, invece, considero tutti i ticket disponibili, includendo anche informazioni su bug corretti in futuro. Più **preciso** possibile, non affetto da **snoring**.

03 Progettazione – Walk Forward

All'interno dei dati è presente la componente temporale, vogliamo quindi preservare il loro ordine cronologico. Come?

Walk Forward: tecnica time-series.

Per minimizzare lo **snoring**,
utilizziamo la prima metà del dataset.



03

Progettazione – Modelli e tecniche

I modelli utilizzati sono:

- Naive Bayes
- lbk
- Random Forest

Le tecniche, applicate ad ogni iterazione del **Walk Forward** sono:

- Cost Sensitive Classifier
- Feature Selection con Best First Backward
- Balancing con SMOTE

Le metriche utilizzate per la valutazione dei modelli sono:

- Precision
- Recall
- Npofb20
- AUC
- Kappa

04

Risultati – Feature Selection

Vediamo gli attributi selezionati per entrambi i progetti:

Bookkeeper

```
Selected attributes: 1,2,11,13 : 4
LOC
authors_number
cyclomatic_complexity
days_between_commits
```

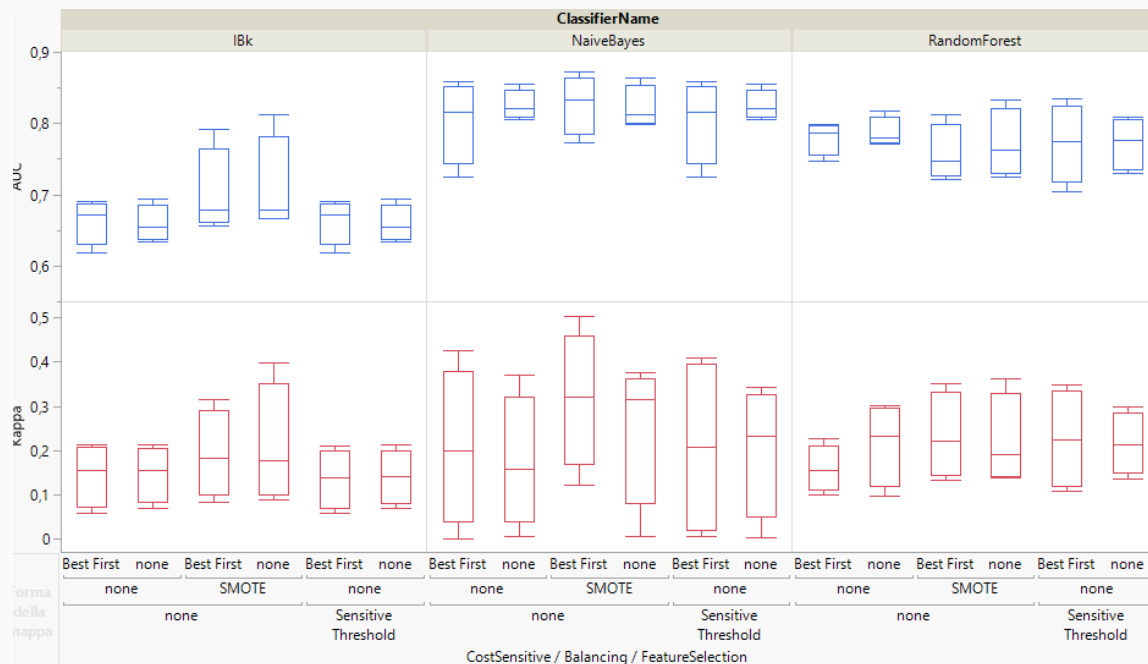
Avro

```
Selected attributes: 1,3,10,11,13 : 5
LOC
revisions_number
number_fix
cyclomatic_complexity
days_between_commits
```

In entrambi i casi, le metriche personalizzate **Complessità Ciclomatica** e **Days Between Commit** vengono selezionate.

04

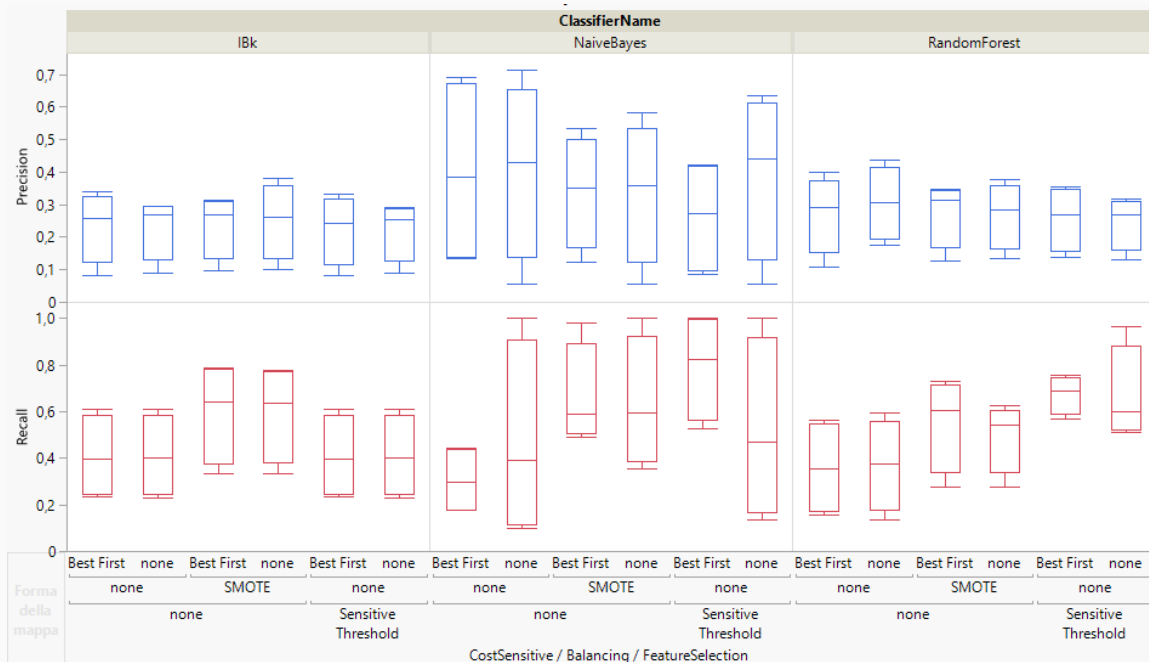
Risultati Bookkeeper – AUC & Kappa



Sia per AUC che per Kappa, Naive Bayes risulta essere il modello con i risultati migliori, Random Forest, soprattutto per Kappa risulta il più stabile.

04

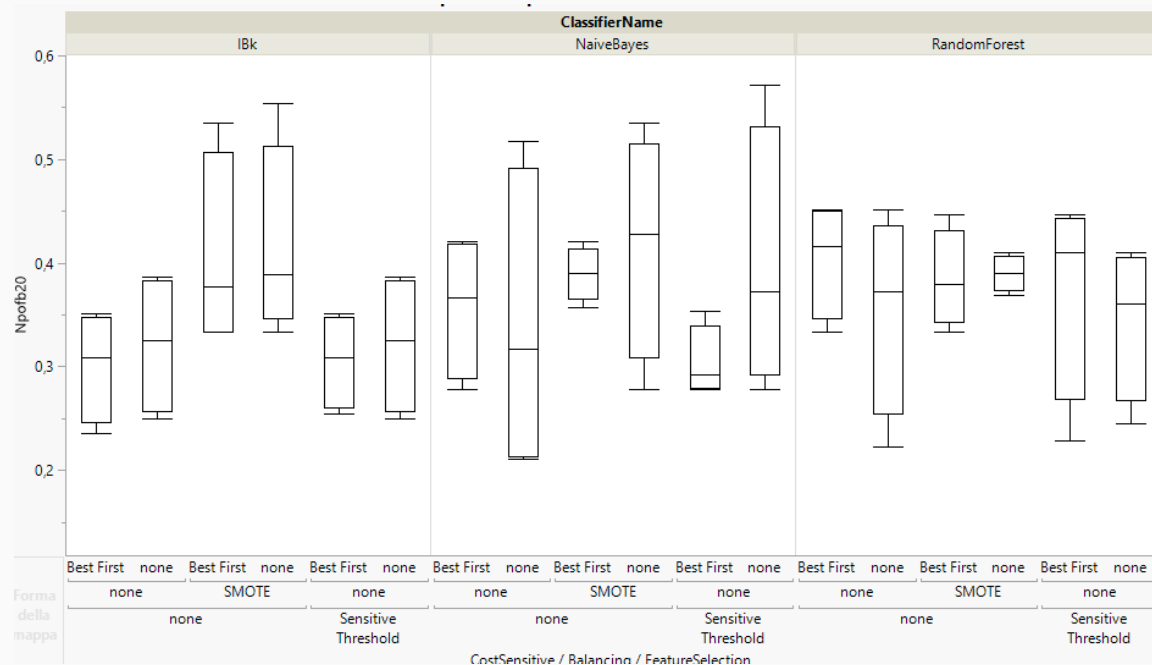
Risultati Bookkeeper – Precision & Recall



Sia per la **Precision** che per la **Recall** il miglior modello risulta essere **Naive Bayes**.

04

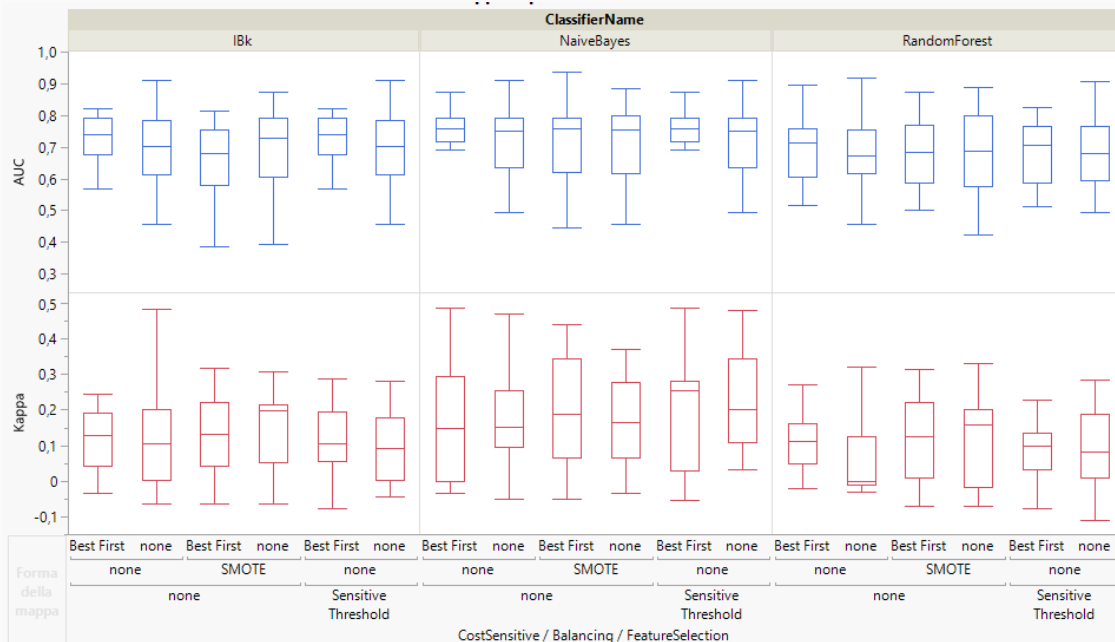
Risultati Bookkeeper – nPofB20



I risultati migliori vengono raggiunti da **Naive Bayes**, ma **Random Forest** sembra essere il più stabile.

04

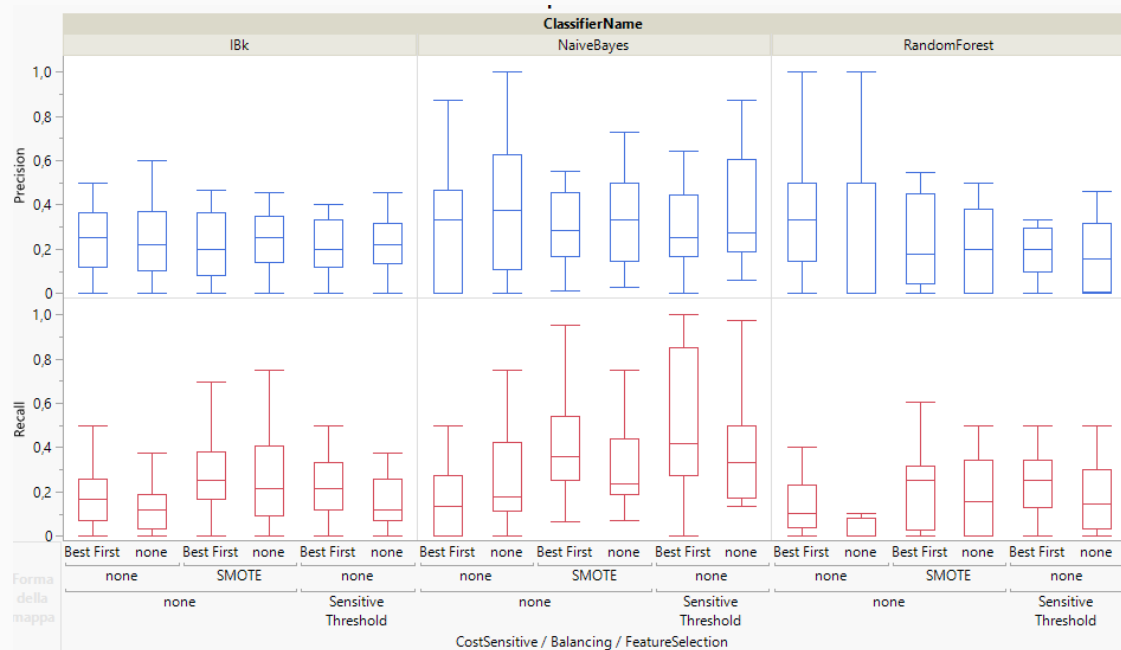
Risultati Avro – AUC & Kappa



I risultati migliori vengono raggiunti, sia per AUC che per Kappa, da Naive Bayes.

04

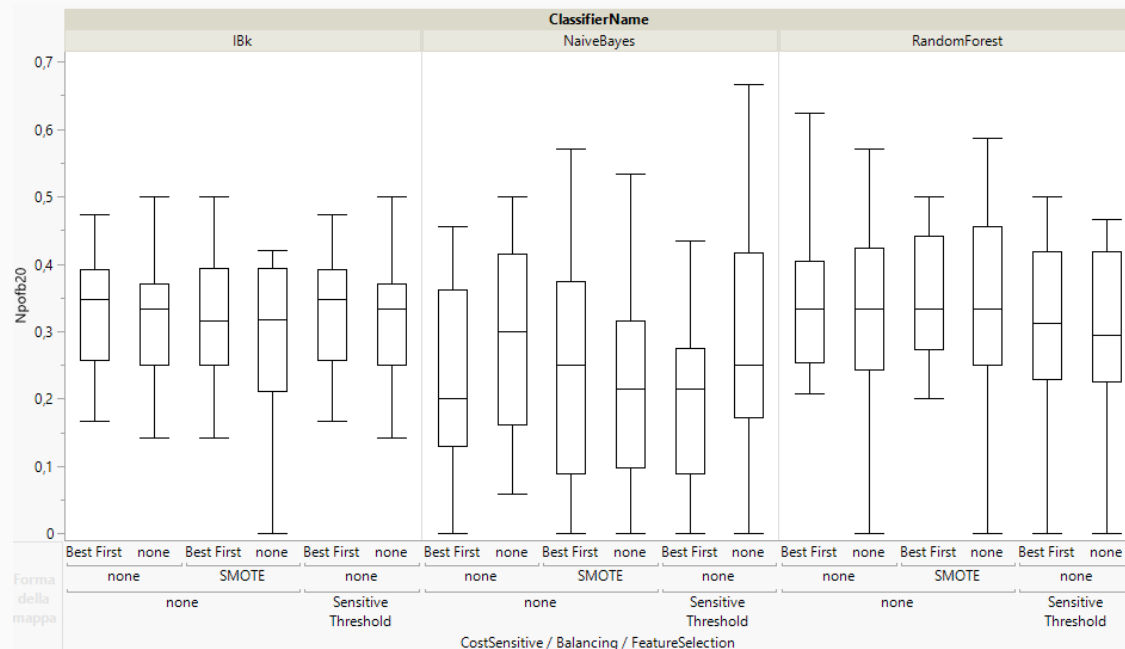
Risultati Avro – Precision & Recall



I risultati migliori, sia per **Precision** che per **Recall**, vengono raggiunti da **Naive Bayes**.

04

Risultati Avro – nPofB20



Il picco massimo viene raggiunto da **Naive Bayes**, a livello di mediana invece **Random Forest** mostra i risultati migliori

04

Risultati – Conclusioni

Per Bookkeeper, **Naive Bayes** risulta essere il modello migliore, soprattutto con l'applicazione delle tecniche **SMOTE** e **Feature Selection**.

Anche per Avro , il modello migliore risulta essere **Naive Bayes** con l'applicazione di **SMOTE** e **Feature Selection**.

05

Assunzioni & Minacce alla validità

1. La data di apertura e chiusura del **ticket**, prelevata da **Jira**, potrebbe non essere accurata.
2. I ticket che etichettiamo come **buggy**, potrebbero non esserlo.
3. Le varie tecniche di proportion sono tecniche **conservative**, possono quindi sottostimare quelle che sono le performance dei vari classificatori.
4. I ticket considerati per proportion potrebbero non essere validi

06 Conclusioni - Fonti

1. Paper Proportion: Leveraging the Defects Life Cycle to Label Affected Versions and Defective Classes, *Bailey Vandehei, Daniel Alencar Da Costa, Davide Falessi*.
2. Paper Snoring: The Impact of Dormant Defects on Defect Prediction: A Study of 19 Apache Projects, *Davide Falessi, AalokAhluwalia, Massimiliano Di Penta*.
3. Tempo di sviluppo testing: <https://intersog.com/blog/development/software-testing-percent-of-software-development-costs/>
4. Costo medio annuo testing: <https://www.globalapptesting.com/blog/software-testing-cost>

06

Conclusioni - Link



https://sonarcloud.io/project/overview?id=MarcoLor01_DatasetBuilderISW2



<https://github.com/MarcoLor01/DatasetBuilderISW2>



**Grazie per
l'attenzione!**