SOFTWARE ENGINEERING

Applicazione di tecniche di Machine Learning per l'ingegneria del software

Andrea Andreoli – 0350012

Roadmap

Introduzione

Metodologia

Risultati

Conclusioni

Introduzione

L'**ingegneria del software** è una disciplina informatica che si occupa dei processi produttivi e delle metodologie di sviluppo finalizzate alla realizzazione di sistemi software

Tra i principali settori di questo vastissimo campo vi è quello dello **sviluppo software**, che si compone di varie attività, quali:

- Progettazione e Architettura del sistema
- Programmazione
- Test e debugging

Introduzione
contesto

Abbiamo detto che nello sviluppo software una delle attività fondamentali è quella di **test e debugging**

I bug all'interno di un software equivalgono ad ingenti perdite di soldi

Siamo quindi interessati a garantire che il software che sviluppiamo sia di qualità

Problema: Come rendere più efficiente possibile l'attività di debugging?

Introduzione
obiettivo

L’obiettivo di questo studio è quello di rendere l’attività di debugging più **agevole**, **efficiente** e **meno costosa**

Come farlo?

Attraverso l’utilizzo di tecniche di **Machine Learning** con lo scopo di predire ed esporre i futuri difetti software

Lo studio è stato condotto sui progetti **Apache BookKeeper** e **Apache OpenJPA** (scritti in **Java**)

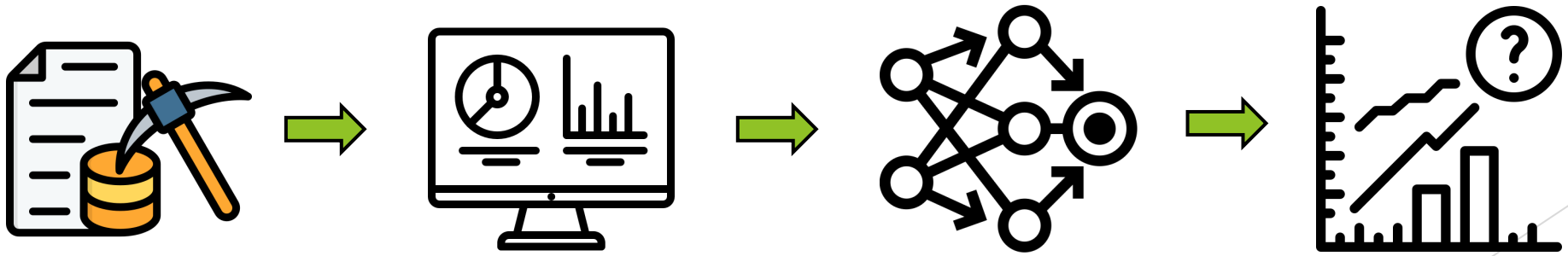
Ciò che ci preme sapere è: quali sono le classi che hanno più probabilità di essere **buggy**?

Metodologia
JIRA e Git - 1

Per poter effettuare delle predizioni abbiamo bisogno di raccogliere dati da fornire ai modelli di **Machine Learning**

Il primo passo è quello di fare data mining sui seguenti software:

- **JIRA**
- **Git**



Metodologia
JIRA e Git - 2

JIRA è un software progettato per il monitoraggio di ticket e progetti (**bug tracking**)

Informazioni che ci interessa recuperare da JIRA:

1. Lista delle release associate a ciascun progetto
2. Lista dei ticket di tipo **bug** associati a ciascun progetto

Nota. In un ticket vengono specificate molte informazioni rilevanti per il nostro studio, ossia: fix version, affected versions, opening version

Metodologia
JIRA e Git - 3

Git è un software per il controllo di versione distribuito (**version control**)

Informazioni che ci interessa recuperare da Git:

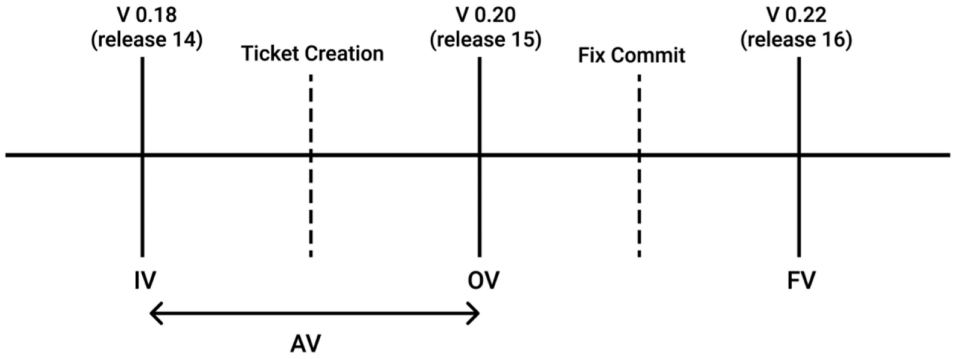
1. Lista dei commit associati a ciascun progetto
2. Classi che vengono toccate da ogni commit in ogni release di ciascun progetto

Metodologia
Proportion - 1

Non tutti i ticket hanno le affected versions disponibili o consistenti, per cui occorre fare il labeling mediante la tecnica della **Proportion**

L'idea chiave è che vi è una **proporzionalità** (*P*) fra i difetti di uno stesso progetto, tra il numero di affected versions fra IV e FV, e il numero di versioni fra OV e FV

Sappiamo che le AV sono nell'intervallo $[IV, FV)$ e per ogni ticket è nota la sua FV, il problema di determinare le AV si traduce nella stima della IV



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

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

Metodologia
Proportion – 2

Vi sono tre metodologie maggiormente conosciute per sfruttare la Proportion:

- **Proportion_Incremental**
- **Proportion_ColdStart**
- **Proportion_MovingWindow**

In questo studio si è deciso di implementare il metodo **Proportion_Incremental** in quanto rappresenta un approccio semplice, lineare e sfrutta la maggior quantità di informazioni disponibili ad un dato istante di tempo.

La **P_Increment** viene calcolata come la media tra tutte le *P* dei difetti passati dello stesso progetto, nel caso in cui la disponibilità di informazioni è limitata allora si utilizza la **P_ColdStart**, per cui è stato necessario implementare anche questo metodo

Metodologia
Metriche

Con il mining effettuato su JIRA e Git possiamo produrre delle metriche (**feature**) per costruire il dataset su cui addestrare i modelli di Machine Learning per effettuare le predizioni

In tabella sono riportate le metriche scelte per questo studio

Metrica	Descrizione
Size (LOC)	Linee di codice
LOC Touched	Somma delle linee di codice aggiunte ed eliminate sulle revisioni
NR	Numero di revisioni
Nfix	Numero di difetti fixati
Nauth	Numero di autori
LOC Added	Somma delle linee di codice aggiunte sulle revisioni
Max LOC Added	Numero massimo di linee di codice aggiunte in una singola revisione
Average LOC Added	Numero medio di linee di codice aggiunte sulle revisioni
Churn	Differenza in modulo tra linee di codice aggiunte ed eliminate sulle revisioni
Max Churn	Valore massimo del churn in una singola revisione
Age	Anni trascorsi dal rilascio della release

Metodologia
Buggyness di una classe

Il fine ultimo di questo studio è quello di predire se una classe può essere buggy o meno

Nella costruzione del dataset, oltre alle feature precedentemente elencate, occorre aggiungere un'ultima colonna in cui viene specificato il valore da predire, ossia, una feature che indica se la classe è **buggy** oppure **non buggy**

Metodologia
Training set e testing set - 1

Il dataset ricavato a seguito dell'operazione di mining viene suddiviso in: **training set** e **testing set**

Il **training set** viene utilizzato per effettuare l'addestramento dei modelli predittivi, quindi, è importante che non siano presenti errori che vanno a riversarsi nelle predizioni, mentre, il **testing set** viene utilizzato per la valutazione del modello

Se sono disponibili molti dati (etichettati) si può realizzare uno split dei dati in training e testing set (ad esempio: $\frac{2}{3}$ - $\frac{1}{3}$)

In generale non sono disponibili molti dati (etichettati), ma sono presenti in quantità limitata, perciò occorre sfruttare delle tecniche più sofisticate

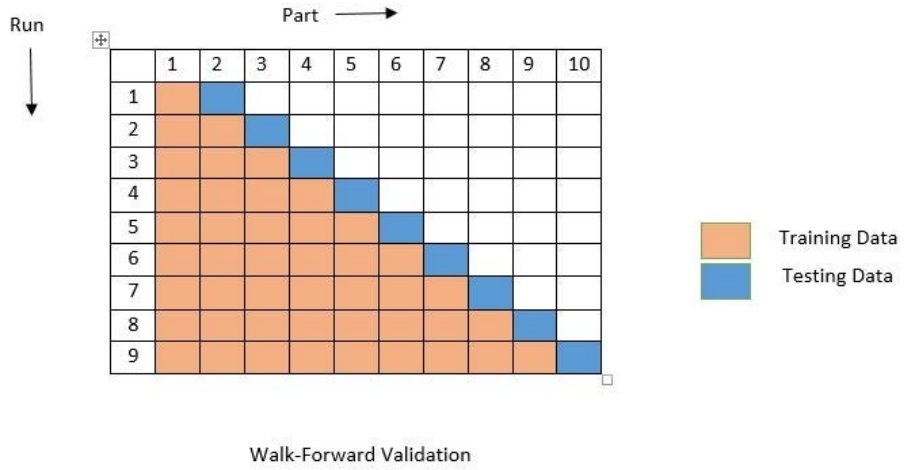
Metodologia
Training set e testing set - 2

Per la validazione è stata utilizzata la **walk-forward** una tecnica **time-series**

Il dataset viene diviso in parti piccole e ordinate cronologicamente

Tutti i dati disponibili fino alla parte da predire sono usati come training set e la parte da predire è usata come testing set

L'**accuratezza** del modello è calcolata come la media di tutte le run effettuate



Walk-Forward Validation

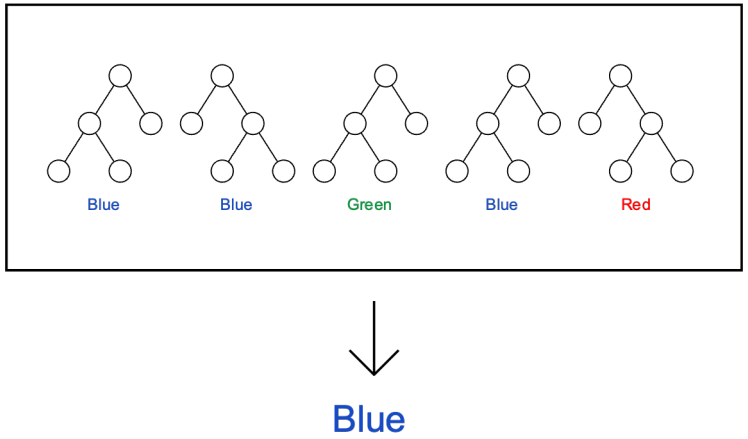
Metodologia
Classificatori utilizzati

Per lo studio dei risultati sono stati utilizzati tre classificatori:

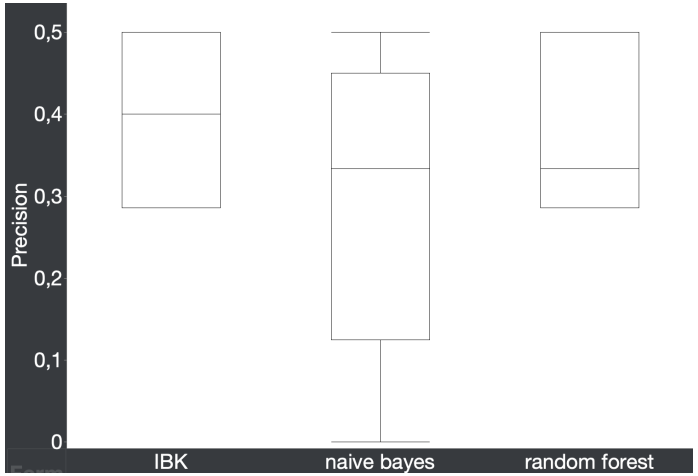
- Random Forest
- Naive Bayes
- IBK

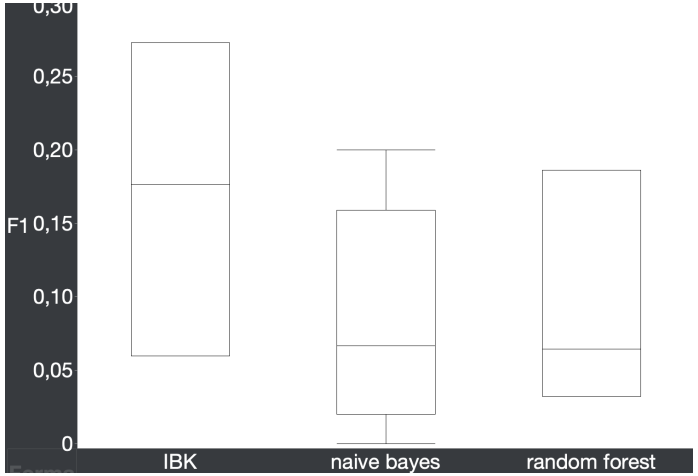
Tecniche utilizzate con i classificatori:

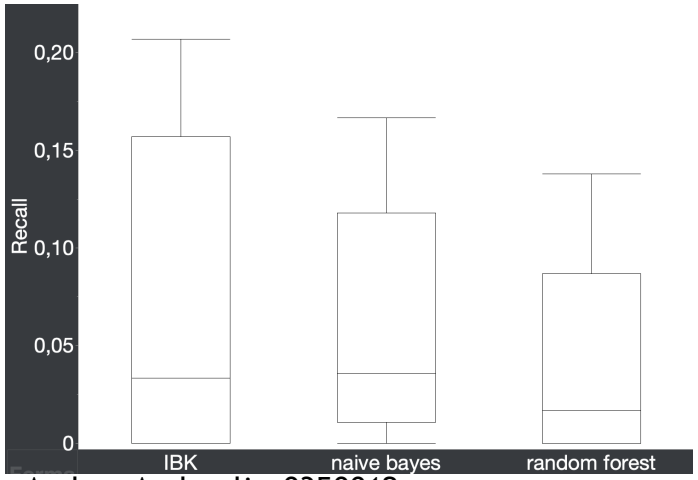
- Nessuna tecnica applicata
- Feature selection best first
- Sensitive learning ($CFN = 10 \cdot CFP$)
- SMOTE come balancing

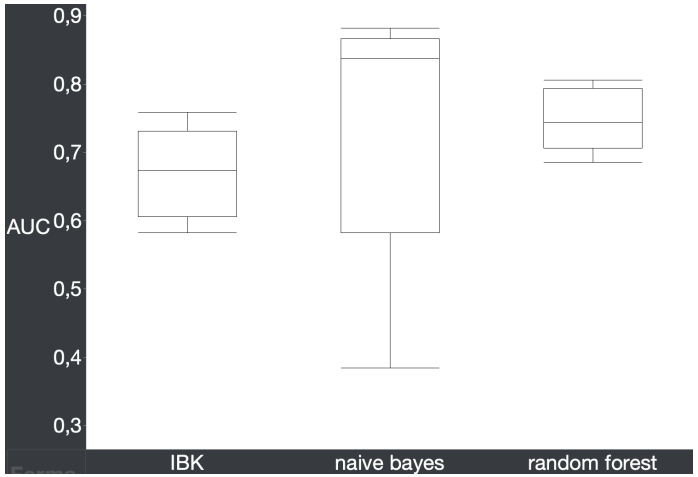


Risultati (BookKeeper)
Classificatori base

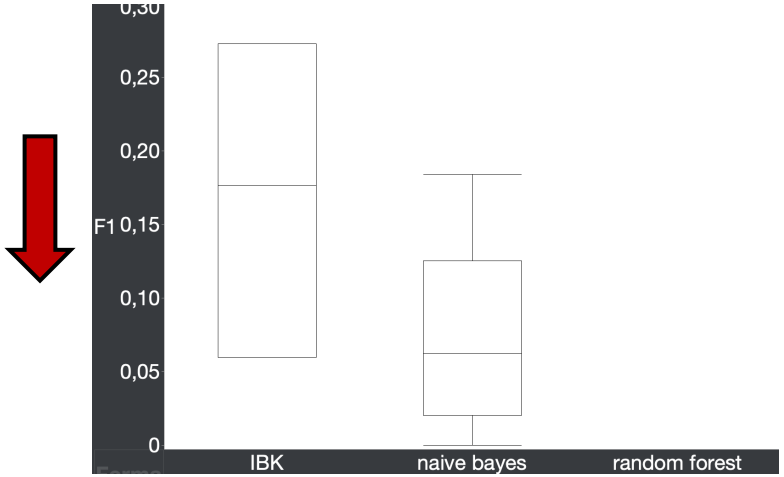




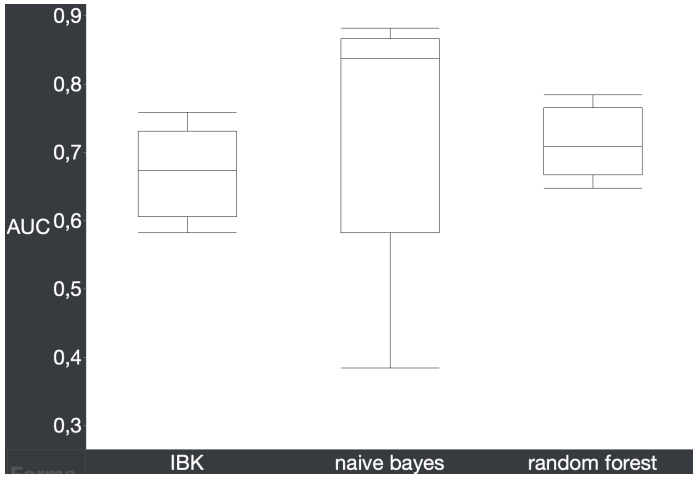




Risultati (BookKeeper)
Classificatori con sensitive learning



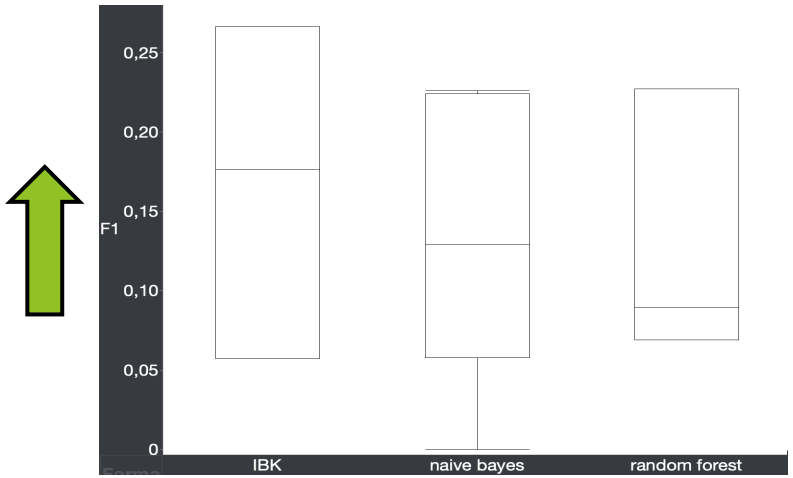
=



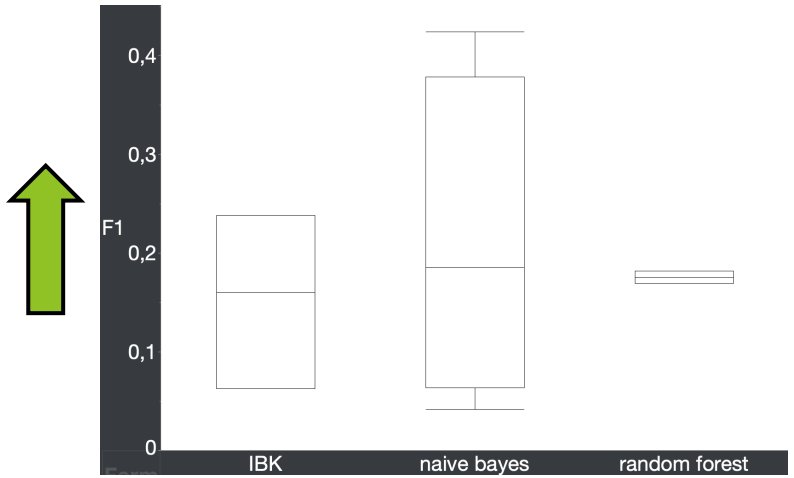
Notare che solo **IBK** ha
un valore di **AUC** non invariato
rispetto al classificatore base

Risultati (BookKeeper)

Classificatori con sampling



Classificatori con feature selection



Notare che solo **IBK** ha un valore di **F1** invariato rispetto al classificatore base

Risultati (BookKeeper)
NPofB20

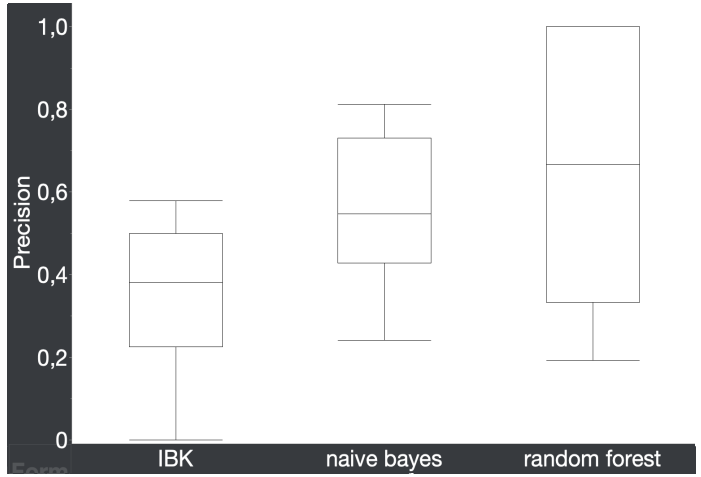
PofB: effort-aware metric definita come la proporzione delle entità identificate analizzando il primo $x\%$ della code base ordinate secondo le relative probabilità di essere buggy

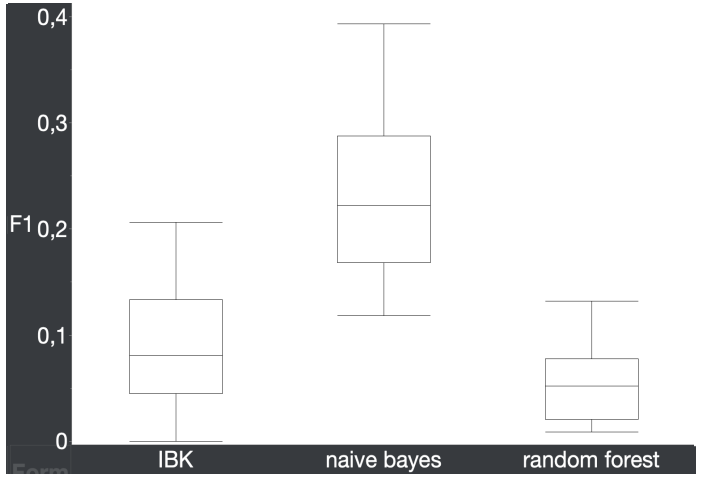
I valori medi ottenuti su ciascun classificatore per **NPofB20** (ossia la PofB normalizzata) dalle 5 run eseguite sono i seguenti

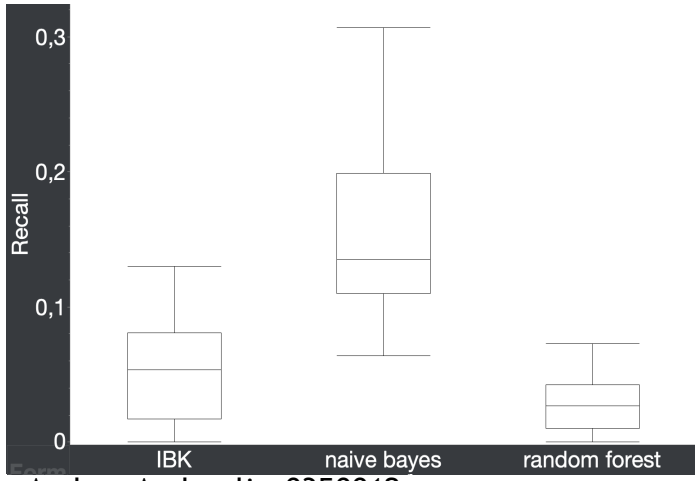
- IBK: 0.284 = 28%
- Naive Bayes: 0.422 = 42%
- Random Forest: 0.454 = 45%

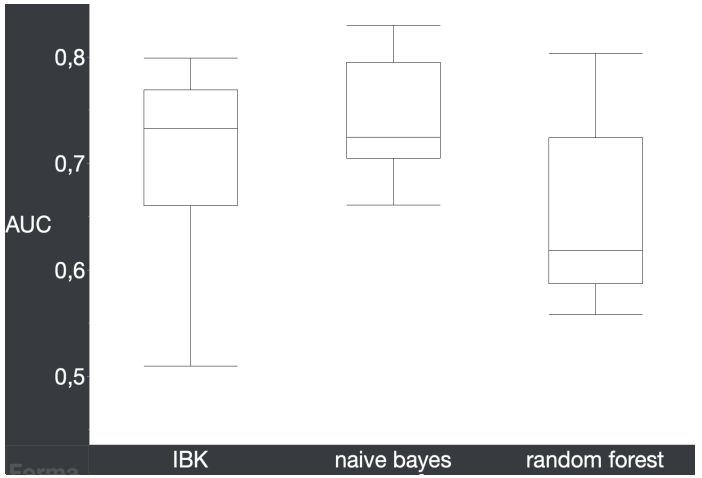
Random Forest sembra essere il migliore rispetto a questa EAM

Risultati (OpenJPA)
Classificatori base

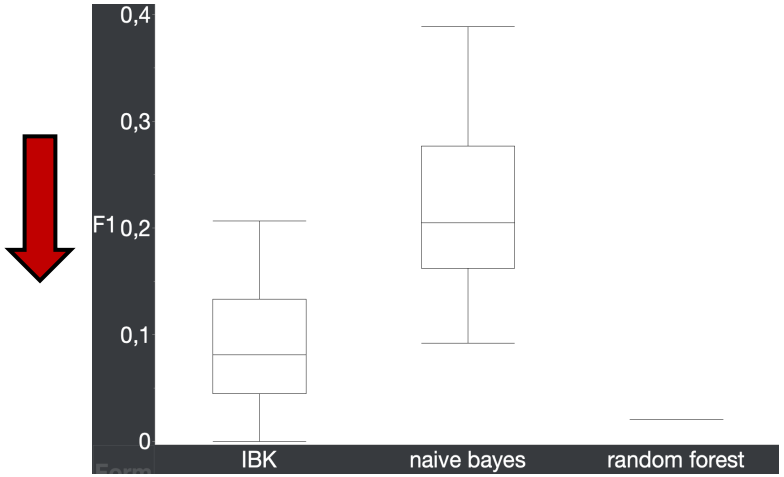


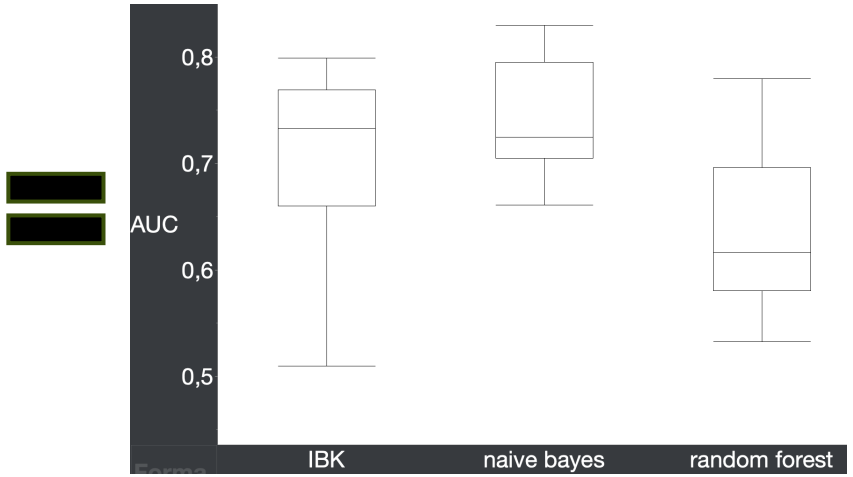






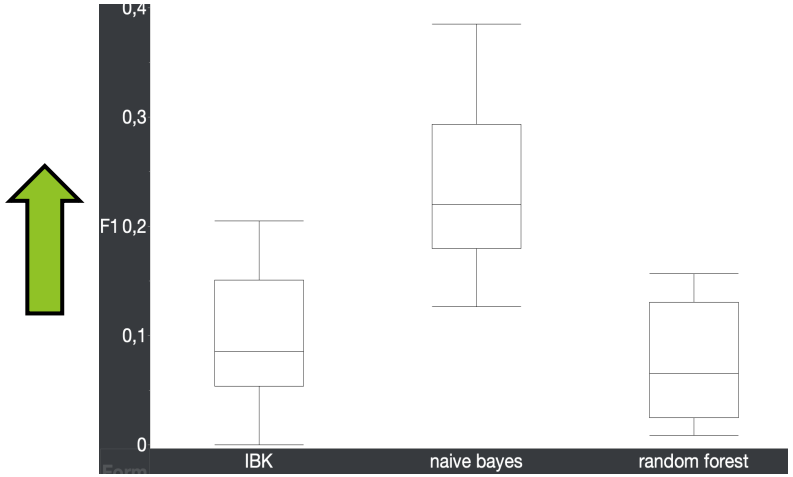
Risultati (OpenJPA)
Classificatori con sensitive learning





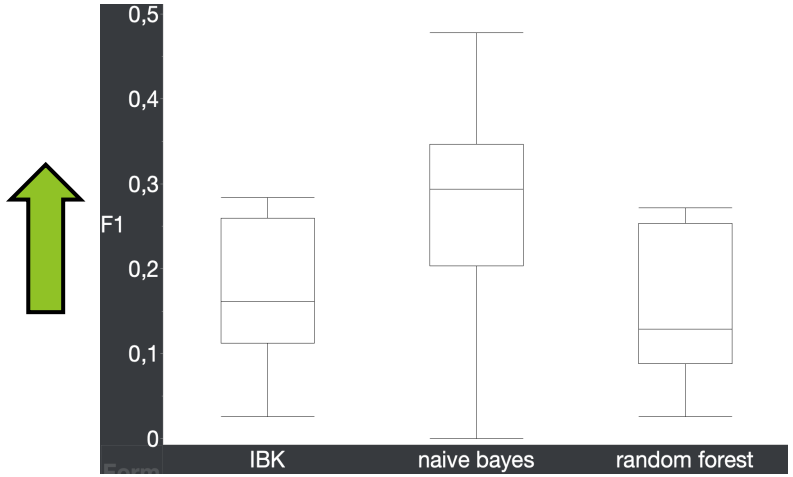
Risultati (OpenJPA)

Classificatori con sampling



Notare che con l'applicazione del sampling si hanno dei miglioramenti marginali

Classificatori con feature selection



Risultati (OpenJPA)
NPofB20

PofB: effort-aware metric definita come la proporzione delle entità identificate analizzando il primo $x\%$ della code base ordinate secondo le relative probabilità di essere buggy

I valori medi ottenuti su ciascun classificatore per **NPofB20** (ossia la PofB normalizzata) dalle 17 run eseguite sono i seguenti

- IBK: 0.240 = 24%
- Naive Bayes: 0.168 = 17%
- Random Forest: 0.246 = 25%

Random Forest sembra essere il migliore rispetto a questa EAM

Conclusioni

Come precedentemente visto dai risultati ottenuti **non** si ha un classificatore nettamente migliore rispetto ad un altro

Né tantomeno esiste una tecnica da applicare migliore rispetto a un'altra

La scelta del classificatore e della tecnica da applicare può variare in base a:

- Tipologia di problema da affrontare
- Dataset a disposizione
- Feature che compongono il dataset
- Istanze di training disponibili

In generale ci sono molte variabili in gioco, quindi **non** esiste una scelta giusta o sbagliata!

Link

GitHub

Link repository GitHub del progetto:
https://github.com/Andrea041/ISW2-Project

Sonarcloud

Link di Sonarcloud:
https://sonarcloud.io/project/overview?id=Andrea041_Software-Engineering-2-Project