Machine Learning for Software Engineering

Marcello Mario – Matricola: 0333376

Agenda

- Introduzione
  - Contesto e obiettivo dello studio
- Metodologia
  - Progettazione e costruzione del dataset
  - Metriche considerate
  - Metodologia di valutazione dei classificatori
- Risultati della misurazione
- Analisi dei risultati
- Link utili

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 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 per identificarle?

Contesto

- Per rispondere alla domanda precedente è opportuno avere un quadro completo di quali classi sono state caratterizzate da bug nel passato, e all'interno di quali release del progetto.
- Con queste informazioni, è possibile sfruttare gli strumenti di Machine Learning per predire quali classi possono, attualmente, 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 **Openjpa**), aver individuato quali classi sono state buggy e in quali release, e aver considerato tre classificatori (**Random Forest**, **Naive Bayes** e **IBk**), lo scopo finale sarà quello di stabilire quale classificatore effettua le predizioni migliori e con quali tecniche di utilizzo.

Le tecniche di utilizzo considerate sono la **Feature Selection**, il **Sampling**, la **Cost-Sensitivity Classification** ed alcune loro combinazioni.

Metodologia: come individuare le classi buggy?

- Ogni bug ha un ciclo di vita.

- Le classi interessate dal bug risultano essere difettose dall'Injected Version inclusa fino alla Fixed Version esclusa.
- L'insieme di release che va dalla Injected Version alla Fixed Version è detto **Affected Version (AV)**

Metodologia: come individuare le classi buggy?

- I due progetti considerati utilizzano il sistema di issue tracking Jira, che riporta informazioni riguardanti tutti i bug scoperti nel sistema e la loro risoluzione in dei ticket.
- I valori di OV e di FV sono sempre riportati dai ticket Jira, dal momento che indicano la creazione e la risoluzione del ticket stesso.

Problema: non tutti i ticket Jira contengono l'indicazione relativa alla Injected Version, che è però necessaria per il labeling delle classi.

Metodologia: come individuare le classi buggy?

- **Idea:** è possibile assumere che esista una proporzionalità fissa tra l'intervallo di tempo che trascorre da quando un bug viene rilevato a quando viene risolto (OV, FV) e tra l'intervallo di tempo che trascorre da quando un bug viene introdotto a quando viene risolto (IV, FV).
- La tecnica per stimare l'Injected Version dei bug è detta **Proportion**:
  - Per tutti i ticket con IV nota viene definita una costante di proporzionalità p:

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

  - Per tutti gli altri ticket il valore di IV viene calcolato nel seguente modo:

$$IV = FV - (FV - \mathrm{OV}) \cdot p$$

Metodologia: come individuare le classi buggy?

- Esistono diverse varianti per il calcolo di p
- Per questo progetto è stata scelta la variante **cold-start**:
  - Per un certo numero k di progetti simili viene calcolato un valore p_i, $\forall\ i \in \{1, \ldots, k\}$, che rappresenta il valore medio di proportion per ogni progetto considerato.
  - Il valore di p utilizzato nel progetto di interesse è la mediana di tutti i p_i
- In questo caso i progetti considerati per il calcolo di p_i sono alcuni progetti sviluppati dalla stessa Apache:
  - Avro
  - Storm
  - Zookeeper
  - Syncope
  - Tajo
  - OpenJPA (verrebbe escluso qualora il progetto di interesse fosse OpenJPA stesso)
  - Bookkeeper (verrebbe escluso qualora il progetto di interesse fosse Bookkeeper stesso)

Metodologia: come individuare le classi buggy?

- Una volta ottenuto il valore di IV per ogni ticket, è possibile effettuare il labeling delle classi, attraverso l'analisi dei commit associati ai ticket.
- Per ogni ticket:
  - Vengono individuati i commit associati ai ticket;
  - Vengono individuate le classi modificate in ogni commit associato ai ticket
  - Ciascuna delle classi modificate è buggy se appartiene ad una release che è compresa nell'intervallo [IV, FV)

Metodologia: alcune assunzioni

- Per implementare il meccanismo di calcolo del valore di proportion e di labeling delle classi, si è reso necessario adottare alcune assunzioni:
  - Per evitare il problema dello snoring (bug presenti ma non ancora scoperti) è stata scartata la seconda metà delle release poiché è molto probabile che i bug presenti nella prima metà siano stati scoperti e risolti.
  - Nel calcolo di proportion, in caso di OV = FV, è stato considerato FV – OV = 1
    - Per evitare denominatori uguali a 0;
    - Per la realistica assunzione che non è detto che un bug scoperto e risolto nella stessa release sia stato introdotto nel sistema effettivamente in quella release; per questo motivo si può assumere che sia stato introdotto almeno nella release precedente.

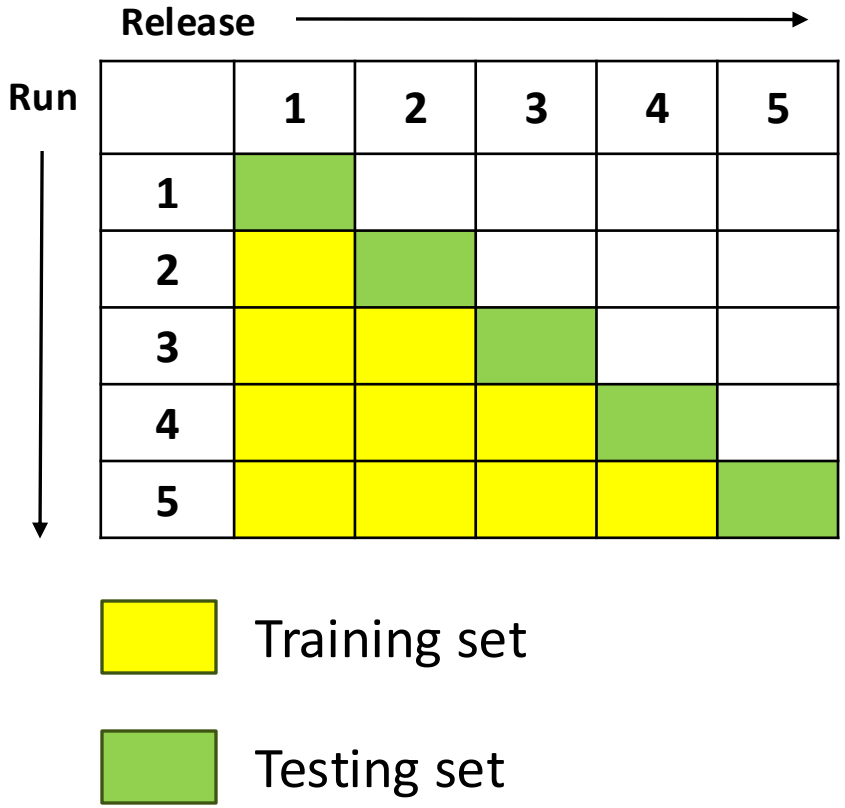
Metodologia: costruzione del dataset

- Oltre alla buggyness sono state considerate altre metriche per poter generare buoni dataset per il training dei classificatori.
- L'introduzione di ulteriori metriche è necessaria per fornire maggiori informazioni riguardo la classe ai classificatori, in modo da renderli più accurati
- Il dataset così ottenuto contiene come feature le informazioni riguardanti la classe (nome, release), le varie metriche aggiuntive ed il valore di buggyness da predire

Metodologia: costruzione del dataset

Nome	Descrizione
Size	Numero di linee di codice
LOC Added	Somma delle linee di codice aggiunte sulle revisioni
LOC Touched	Somma delle linee di codice aggiunte ed eliminate 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
NR	Numero di revisioni
Churn	Differenza in modulo tra linee di codice aggiunte ed eliminate sulle revisioni
Max Churn	Valore massimo del churn in una singola revisione
Average Churn	Media dei churn sulle revisioni
Number of Authors	Numero di autori

Metodologia: valutazione dei classificatori

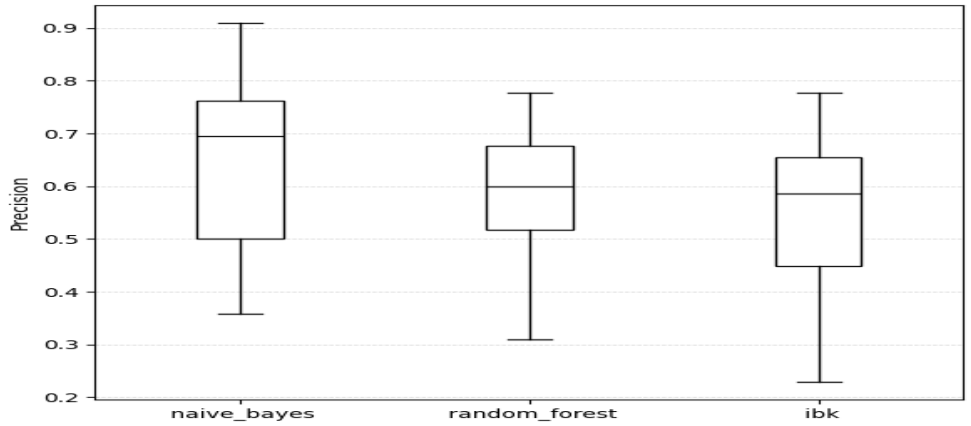


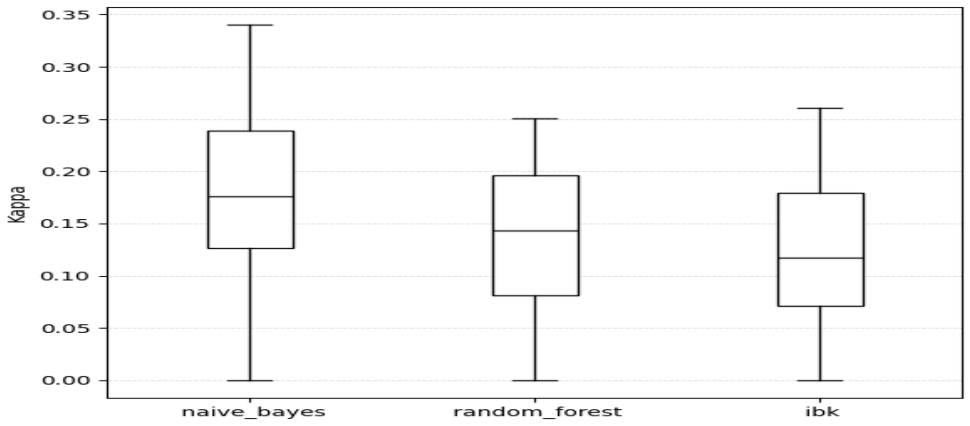
- Per la valutazione dei classificatori è stata scelta la tecnica del **walk-forward**
- È una tecnica **time-series**, per cui tiene conto dell'ordine temporale dei dati
- Il training set viene ottenuto considerando solo i dati disponibili fino alla release considerata
- La valutazione finale è calcolata come la media di tutte le iterazioni

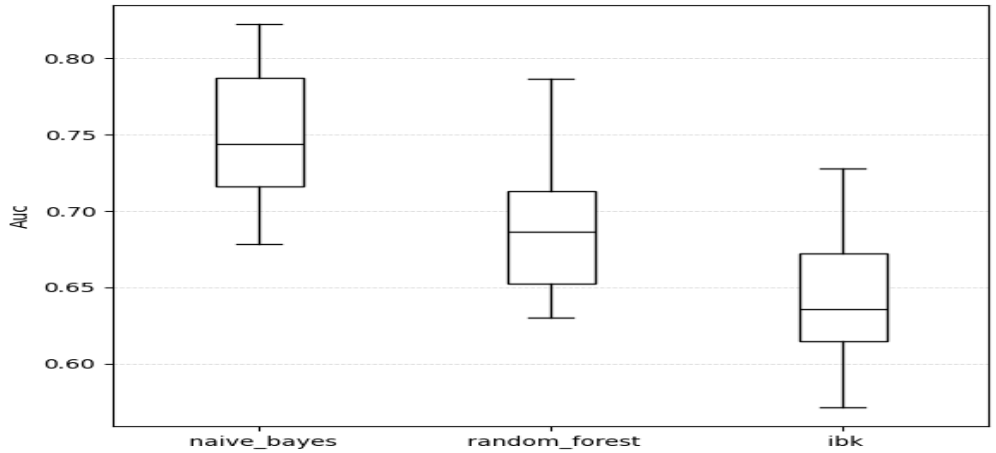
Metodologia: classificatori utilizzati

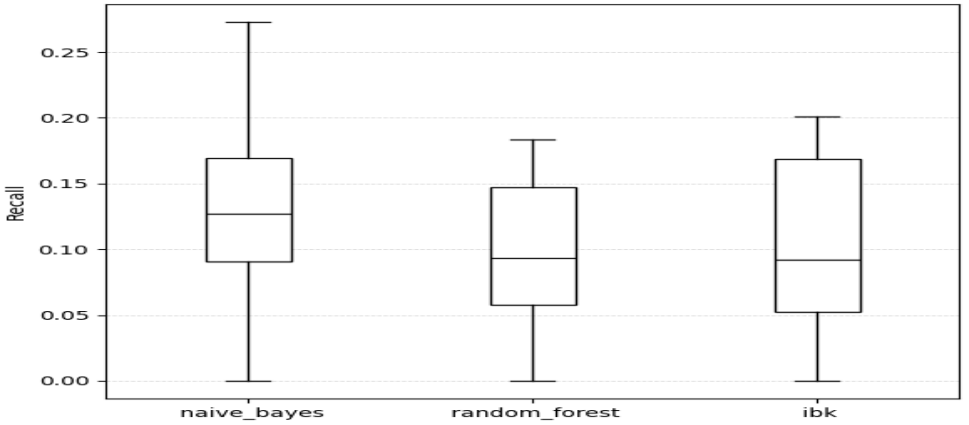
- I classificatori utilizzati
  - Naive Bayes
  - Random Forest
  - IBk
- Tecniche utilizzate con i classificatori
  - Senza nessun filtro
  - Feature selection (best first)
  - Feature selection (best first) + Sampling (undersampling)
  - Feature selection (best first) + Cost-Sensitive Classification (CFN = 10*CFP)

Risultati: BookKeeper, senza filtri



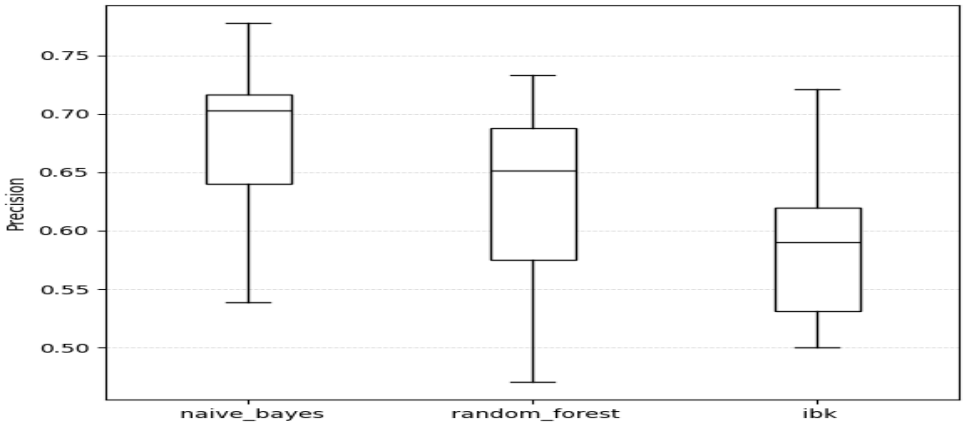


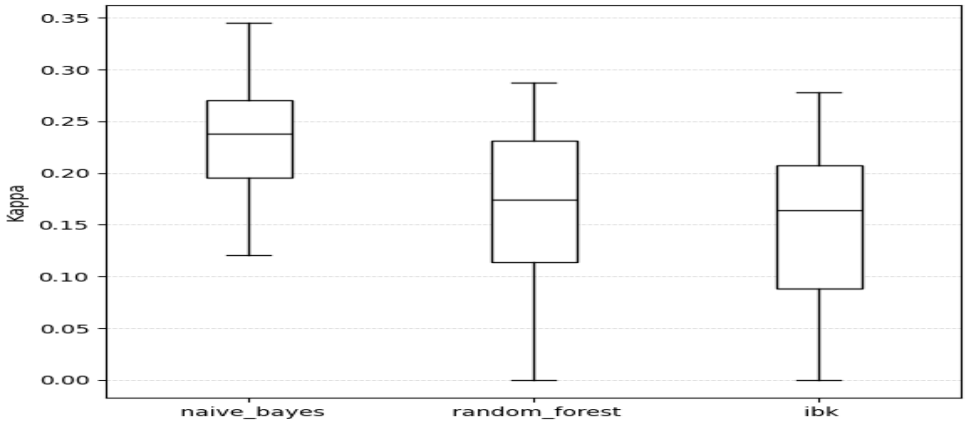


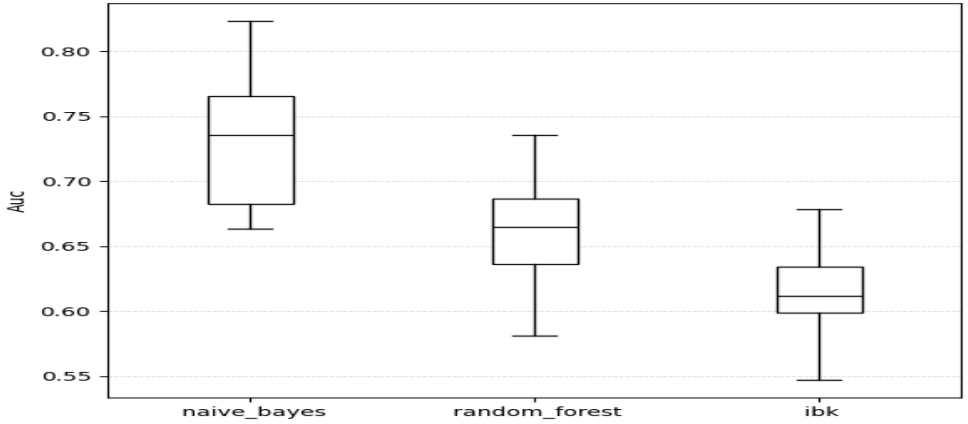


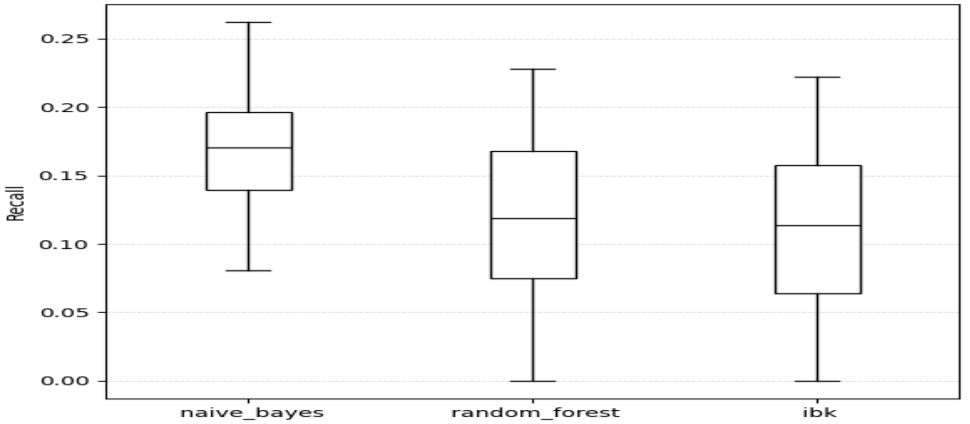
Nel complesso Naive Bayes sembra comportarsi meglio

Risultati: BookKeeper con feature selection



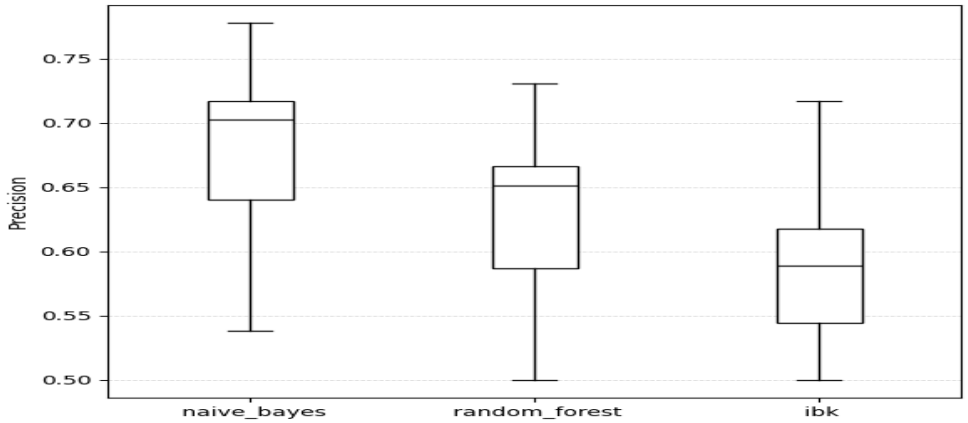


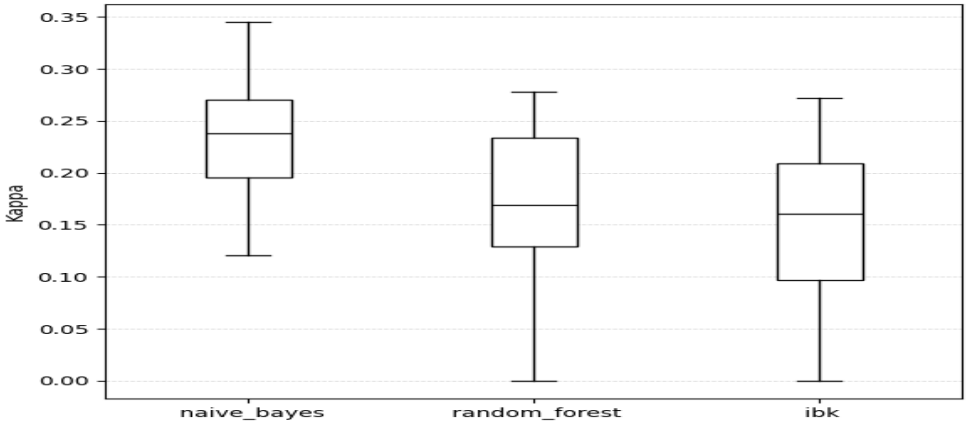


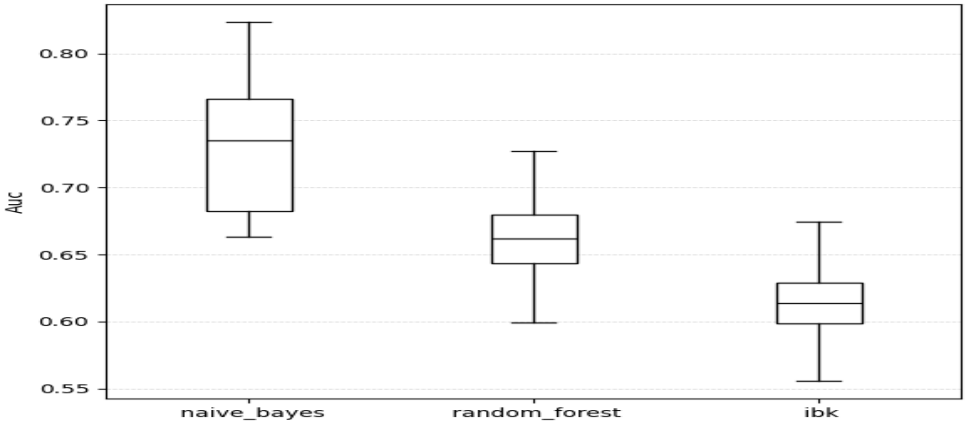


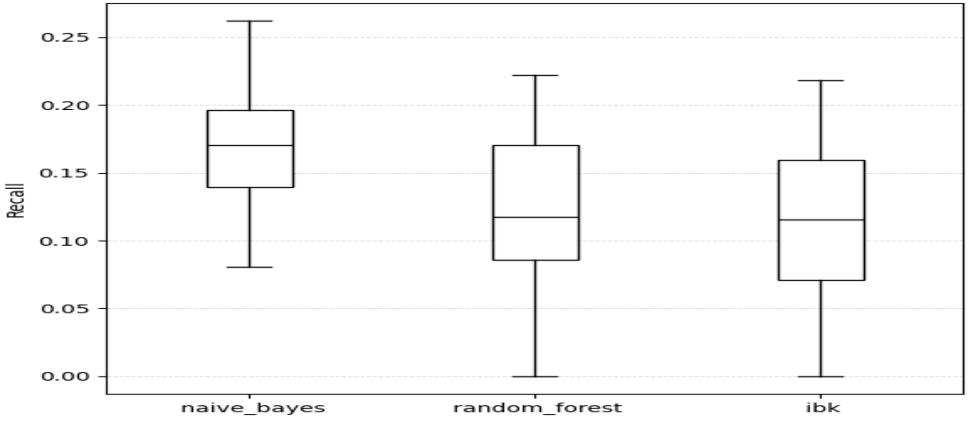
Nel complesso Naive Bayes sembra comportarsi meglio

Risultati: BookKeeper, feature selection + undersampling



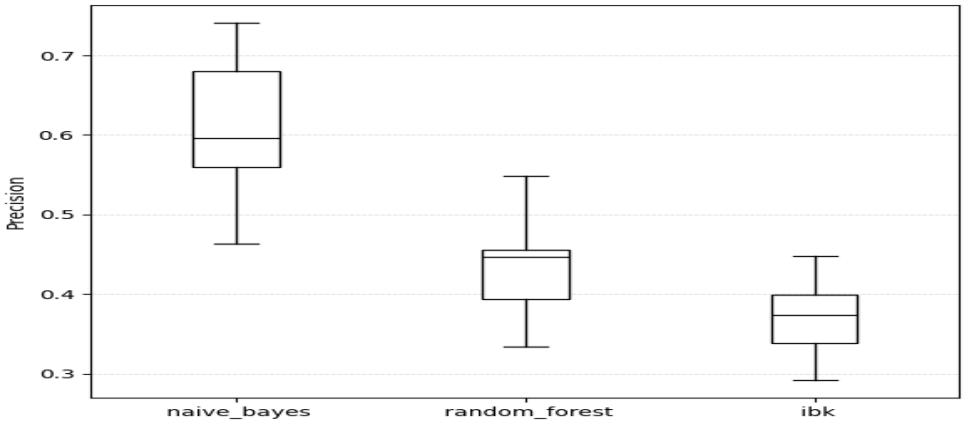


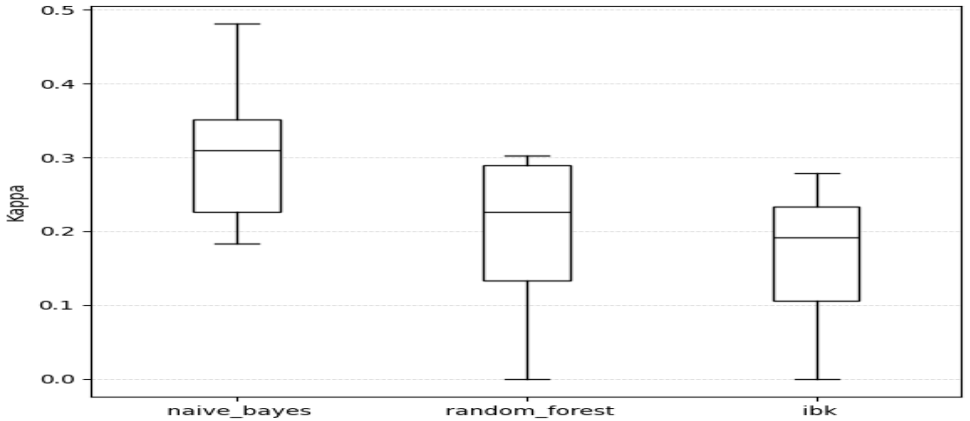


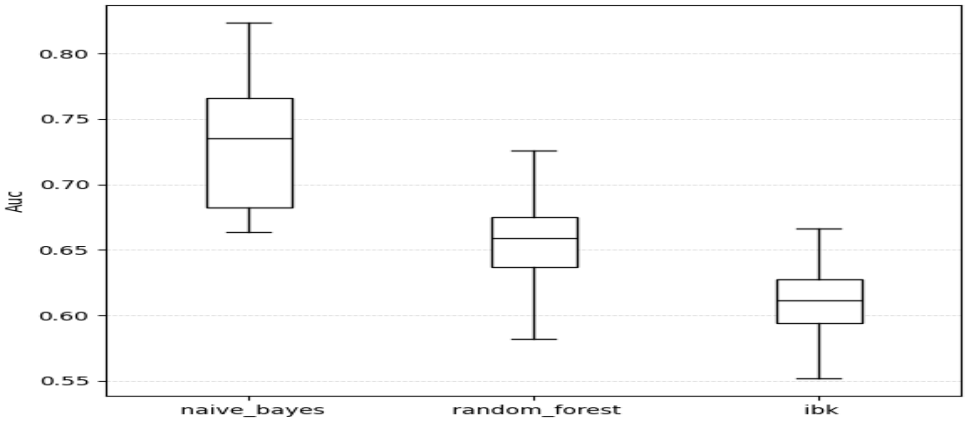


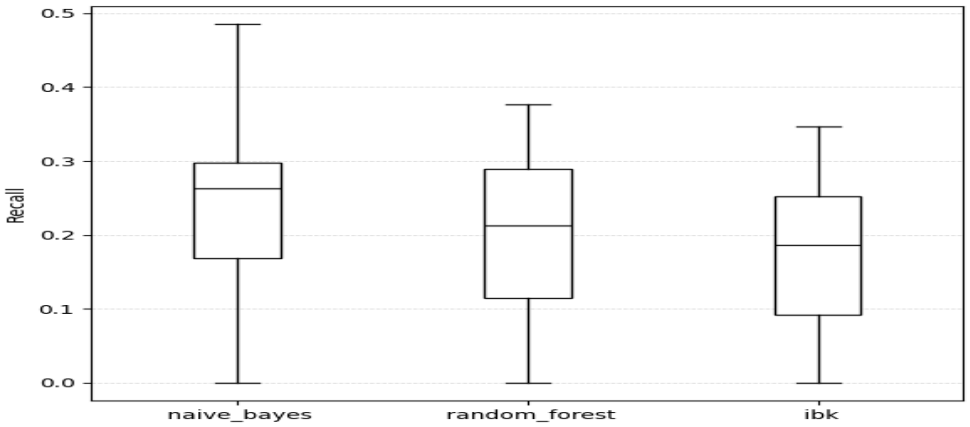
Nel complesso Naive Bayes sembra comportarsi meglio

Risultati: BookKeeper, feature selection + CSC







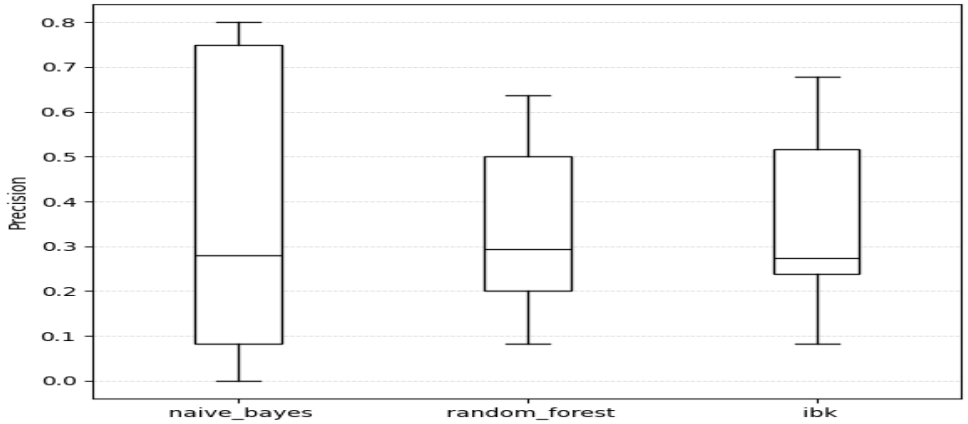


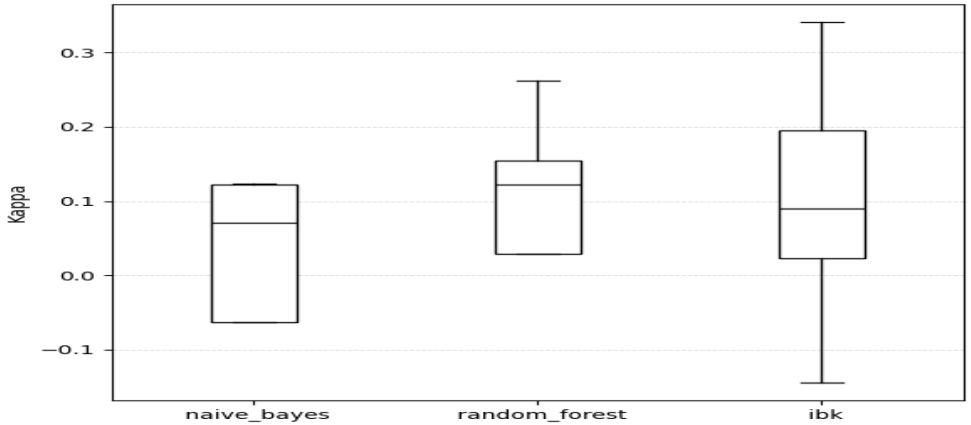
Nel complesso Naive Bayes sembra comportarsi meglio

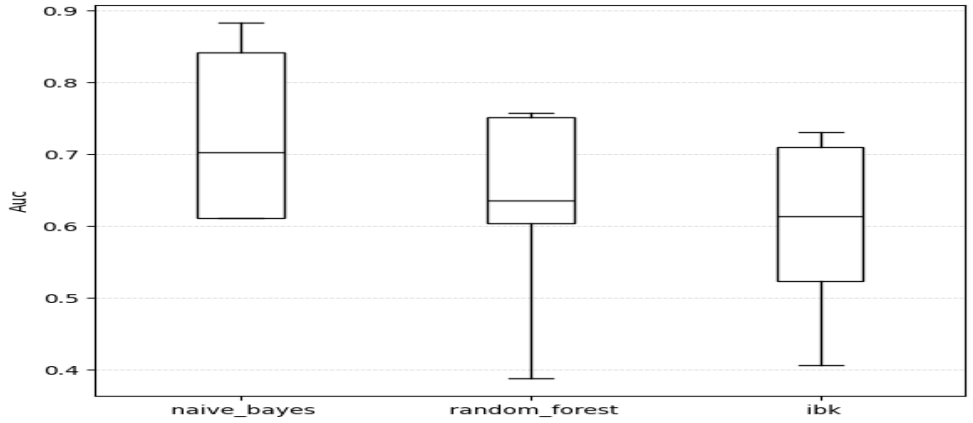
Risultati: BookKeeper, conclusioni finali

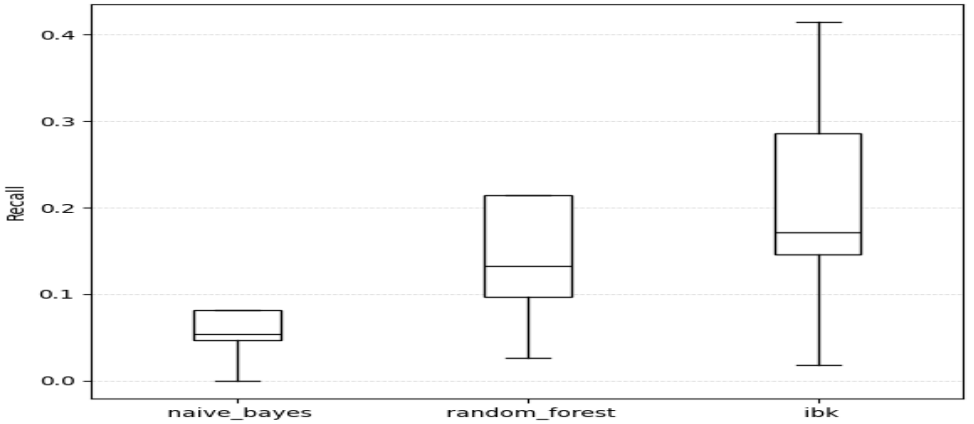
- Andando ad analizzare il valore mediano massimo di ognuna delle metriche di valutazione è possibile notare che:
  - La precision più elevata si ottiene tramite Naive Bayes nei casi senza fs, con fs e fs + undersampling
  - Il kappa più elevato si ottiene con Naive Bayes fs + CSC
  - La recall più elevata si ottiene con Naive Bayes fs + CSC
  - L'AUC più elevato si ottiene con Naive Bayes, ognuna delle sue configurazioni porta a valori molto simili
- Il classificatore dominante è dunque **Naive Bayes**
- La configurazione migliore è **Naive Bayes + fs + CSC** poichè offre i valori più alti di recall, AUC e Kappa. La precision, pur non essendo il massimo assoluto, rimane comunque su buoni livelli.

Risultati: OpenJPA , senza filtri



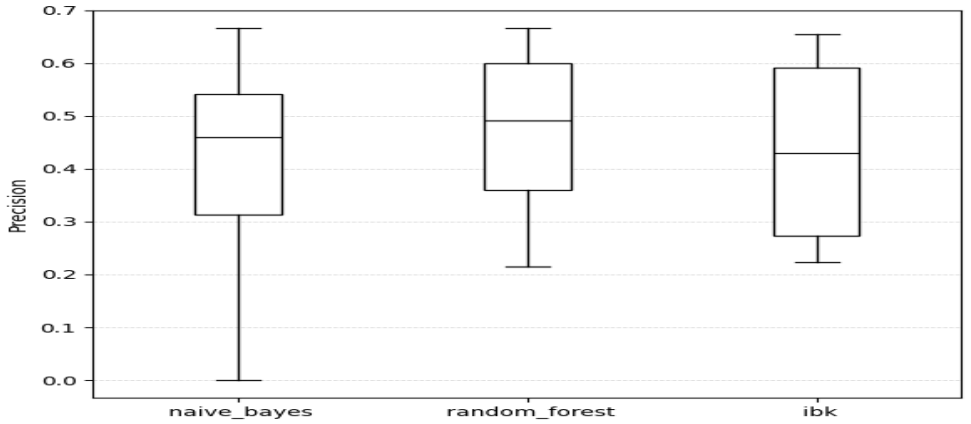


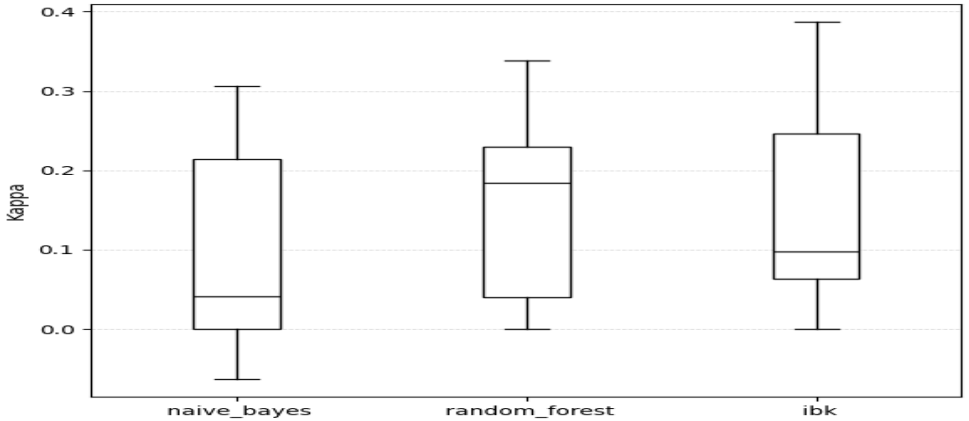


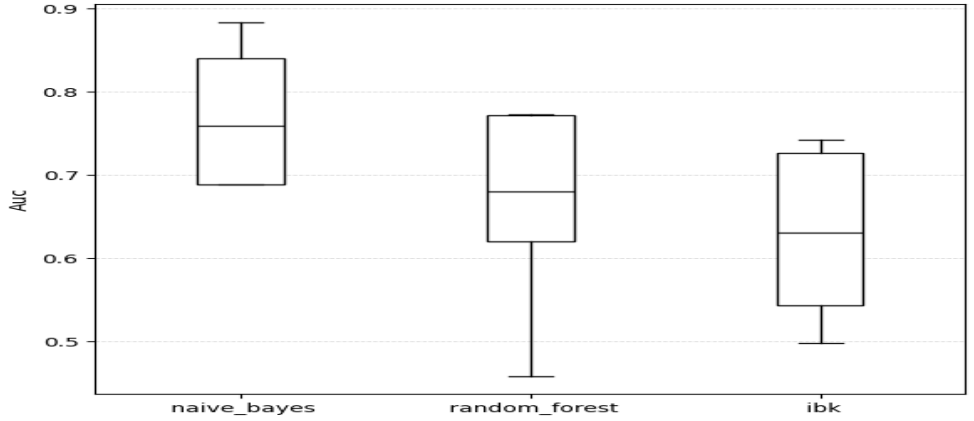


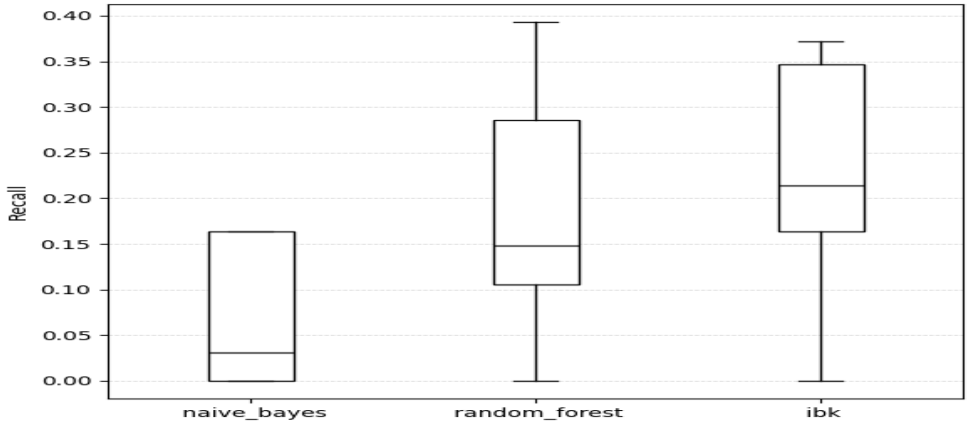
Nel complesso Random Forest sembra comportarsi meglio

Risultati: OpenJPA con feature selection



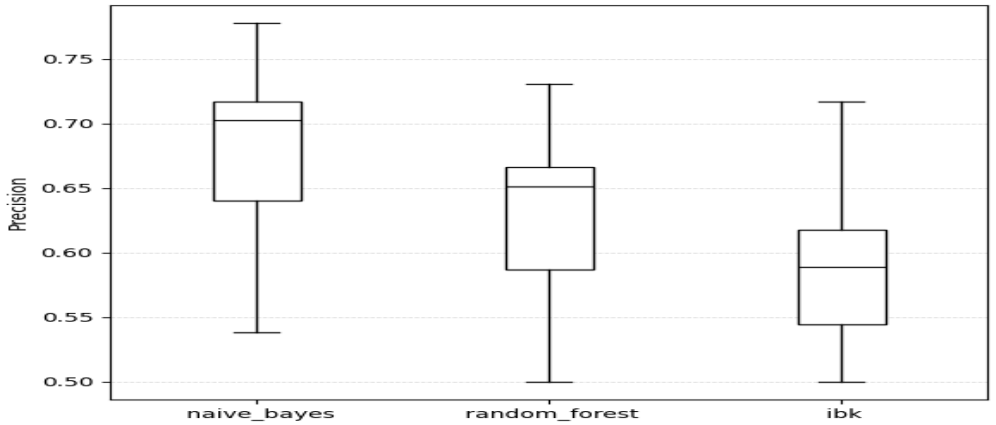


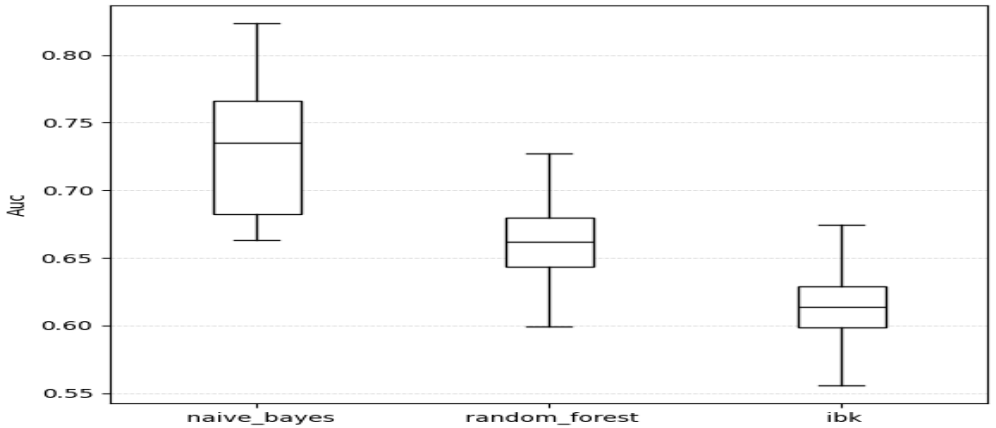


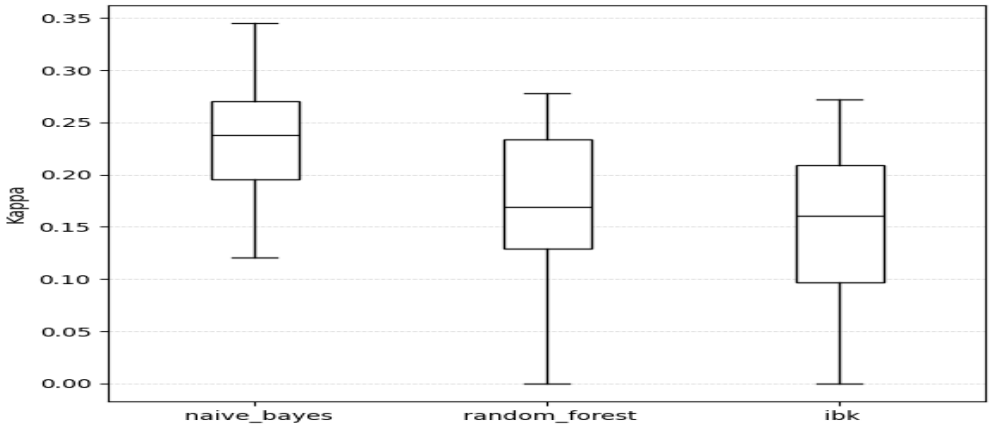


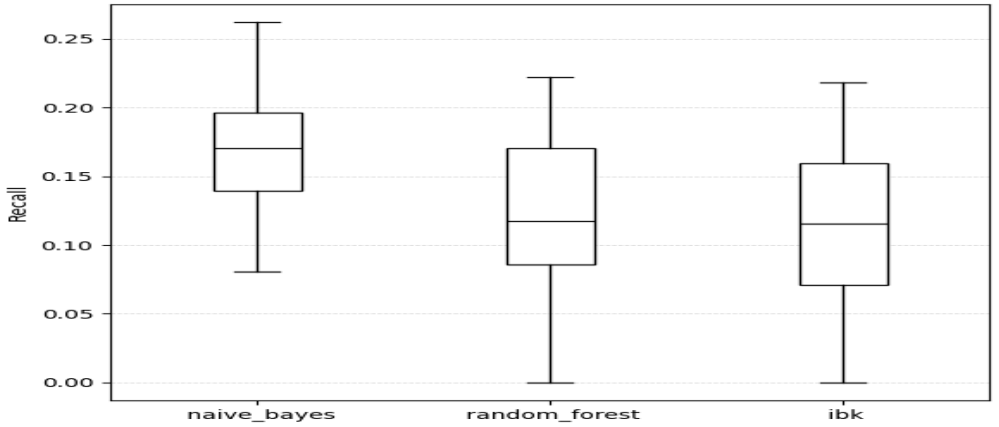
Nel complesso Random Forest sembra comportarsi meglio

Risultati: OpenJPA , feature selection + undersampling



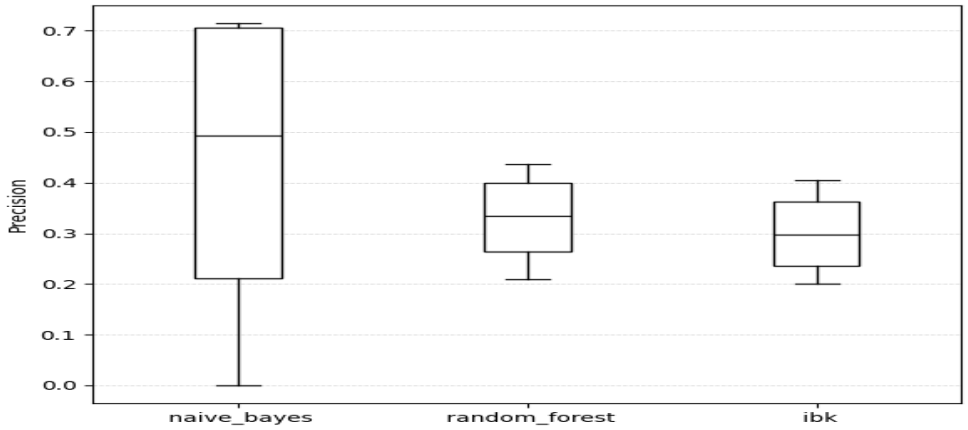


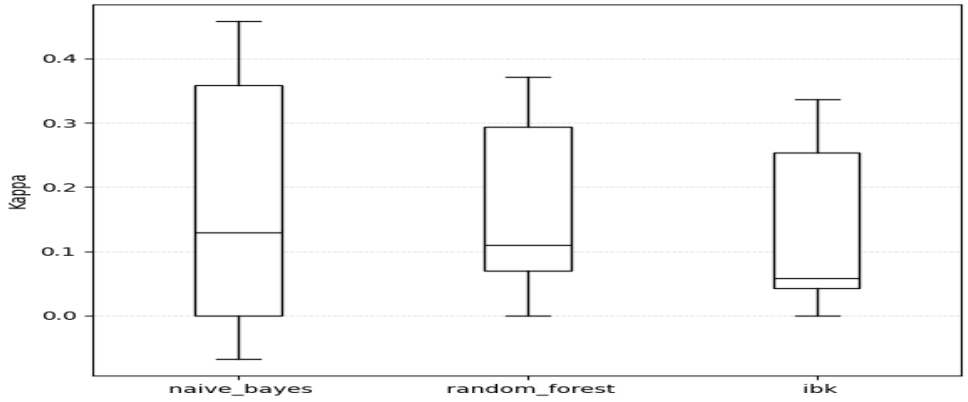


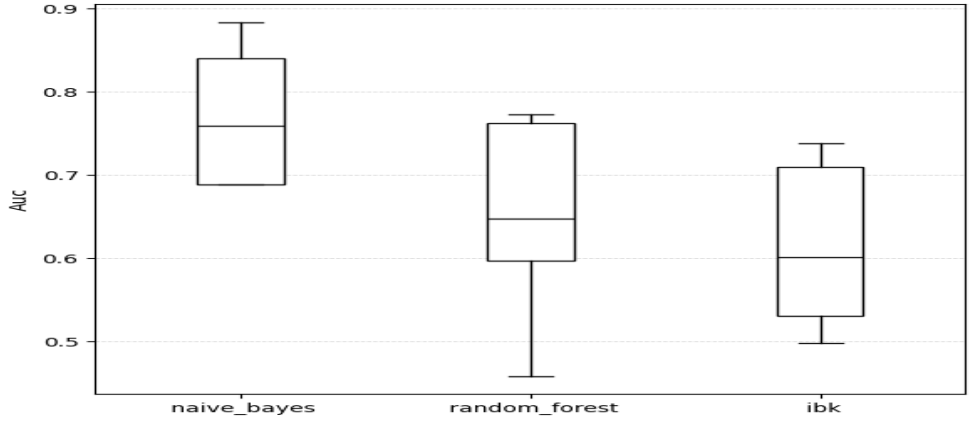


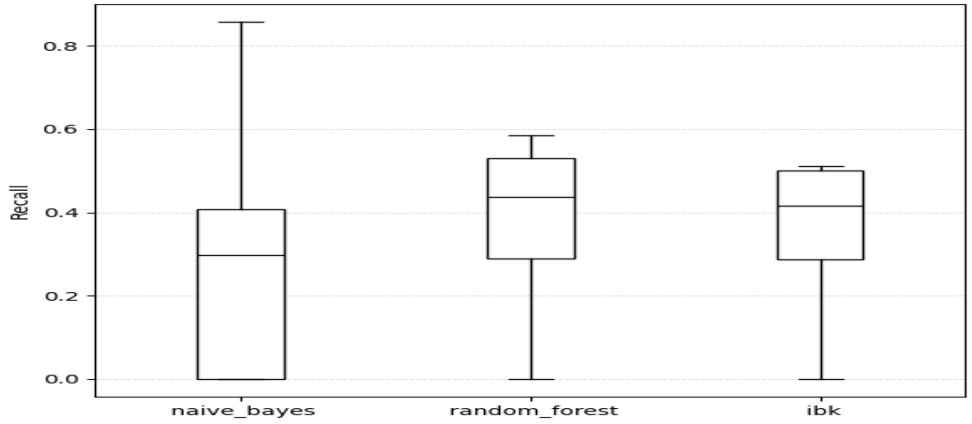
Nel complesso Naive Bayes sembra comportarsi meglio

Risultati: OpenJPA, feature selection + CSC









Nel complesso Naive Bayes sembra comportarsi meglio

Risultati: OpenJPA, conclusioni finali

- Andando ad analizzare il valore mediano massimo di ognuna delle metriche di valutazione è possibile notare che:
  - La precision più elevata si ottiene tramite Naive Bayes con fs + undersampling
  - Il kappa più elevato si ottiene con Naive Bayes fs + undersampling
  - La recall più elevata si ottiene con Random Forest fs + CSC
  - L'AUC più elevato si ottiene con Naive Bayes fs + undersampling
- Il classificatore dominante è dunque **Naive Bayes**
- La configurazione migliore risulta essere **Naive Bayes + FS + undersampling** poichè offre i valori più alti di precision, AUC e Kappa. Il recall, pur non essendo il massimo assoluto, rimane comunque su buoni livelli.

Link utili

- **Github:** https://github.com/MarioMarcello98/ProgettoISW2Falessi
- **SonarCloud:** https://sonarcloud.io/summary/new_code?id=MarioMarcello98_ProgettoISW2Falessi