VALERIO MORONI

UNIVERSITÀ DEGLI STUDI DI ROMA
“TOR VERGATA”

MATRICOLA: 0278403

DELIVERABLE DEL PROGETTO DI ISW2

MACHINE LEARNING FOR SOFTWARE ENGINEERING

AGENDA

Il lavoro è organizzato secondo la seguente scaletta:

1. Introduzione
2. Progettazione
3. Risultati
4. Conclusione
5. Links

INTRODUZIONE (1)

- Nello sviluppo software, tra gli aspetti determinanti, ci sono la qualità del software prodotto e del relativo testing.
- Per lo sviluppo di larghe applicazioni possono essere importanti degli strumenti di supporto all'individuazione di possibili classi contenenti bug, in modo da poter andare ad affinare l'attività di testing.
- L'analisi statistica offre la possibilità, partendo da un insieme di dati di riferimento, di predire comportamenti futuri con un determinato grado di accuratezza.
- Il **machine learning** aiuta in quest'operazione in quanto offre un supporto software all'utilizzo della statistica per l'analisi dei dati.
- Il supporto software è offerto da algoritmi e modelli predittivi che migliorano le proprie performance con l'esperienza accumulata su un determinato insieme di dati.
- L'obiettivo dello studio è stato quello di misurare l'effetto di tecniche di ***sampling*** e di ***feature selection*** sull'accuratezza di modelli predittivi di localizzazione di bug nel codice di larghe applicazioni open-source.
- In questo studio si sono analizzati due progetti open source di Apache SF: **BookKeeper** e **OpenJPA**.
- Nel corso del lavoro verranno identificate le classi di ciascun progetto più predisposte alla difettosità e, successivamente, verranno utilizzati modelli di machine learning con l'obiettivo di predire su quali file sia maggiore la predisposizione all'insorgenza di bug nel codice, sulla base dei valori di **metriche** calcolate su di essi.

INTRODUZIONE (2)

- Si cercherà, quindi, di identificare quali tecniche di **feature selection** e **balancing** aumentano l'accuratezza dei classificatori e quale tra questi ultimi produce il miglior risultato.
- Durante il progetto sono stati considerati:

  - **RandomForest**, **NaiveBayes, IBK** come classificatori.

  - **No Selection** e **Best First** come tecniche di feature selection.

  - **No Sampling, Undersampling, Oversampling, SMOTE** per il balancing.

  - **Walk Forward** come tecnica di valutazione.

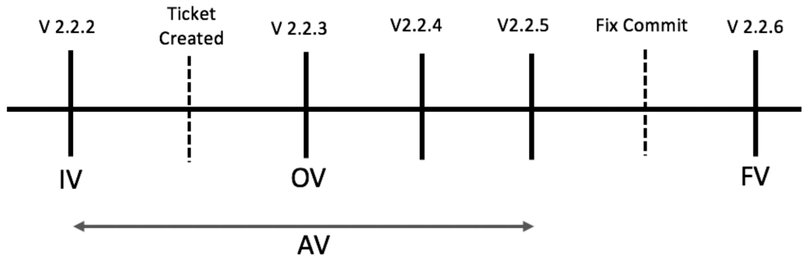
- Per la creazione del dataset, per segnalare una classe come buggy o meno, sono stati considerati i ticket presenti su **Jira** relativi ad un singolo progetto. Jira è un Issue Tracking System, ossia un software per il monitoraggio di ticket relativi ad issues.

PROGETTAZIONE (1) – INFORMAZIONI DA JIRA

Tra le informazioni che Jira fornisce sul singolo ticket, ci si è focalizzati su:

- **Creation Date:** è la data di creazione del ticket.
- **Resolution Date:** è la data di risoluzione del ticket.
- **Opening Version (OV)**: indica la versione rilasciata in seguito all'apertura del ticket.
- **Affected Version (AV):** indica l'insieme delle versioni affette dal bug.
- **Fixed Version (FV):** indica la versione rilasciata in seguito alla chiusura del ticket e, di conseguenza, alla correzione del difetto.

PROGETTAZIONE (2) – VERSIONI



- Le date di creazione e di risoluzione del ticket sono informazioni sempre disponibili su Jira, tramite le quali è possibile risalire ad *OV* ed *FV*.
- OV, infatti, viene generata dal sistema al momento della creazione del ticket.
- FV è la release successiva alla data dell'ultimo commit associato al difetto, ossia il commit che lo risolve e che contiene l'ID del bug nel suo log.
- Le AV, invece, non sono sempre disponibili ed, in caso fossero presenti, non è detto che siano affidabili.
- La lista delle AV è coerente quando la prima AV è precedente all'OV, ossia che l'IV non può essere uguale all'OV. Un bug, infatti, non può essere stato iniettato nel codice dopo aver osservato la relativa failure, ma deve aver interessato una release presente almeno nel momento in cui è stato creato il ticket.
- Per i ticket con AVs non affidabile, essendo queste ultime comprese tra IV e FV (esclusa) e disponendo sempre della FV, la stima del suo valore si traduce nella stima dell'IV.

PROGETTAZIONE (3) – PROPORTION

- Tra le varie strategie a disposizione per la stima di IV, in questo studio empirico è stato considerato l'approccio **proportion**.
- L'idea alla base è che ci sia una proporzione, all'interno dello stesso progetto, tra il numero di versioni necessarie per individuare il difetto ed il numero di versioni per il fix dello stesso.
- La proporzione **P** viene calcolata come $P = \frac{FV - IV}{FV - OV}$
- Il valore di IV si calcola come $IV = FV - (FV - OV) \cdot P$
- L'idea è di calcolare P sui difetti precedenti e di utilizzarlo per il computo delle IV per quei difetti dove le AV non sono disponibili o non sono coerenti.
- Tra le varie tecniche disponibili per il calcolo di P, si è scelto il metodo **increment**, che consiste nella media tra i valori di P dei ticket precedenti.

PROGETTAZIONE (4) – CREAZIONE DEL DATASET

Il workflow seguito per la creazione del dataset è stato il seguente:

1. Vengono memorizzati i ticket ottenuti da Jira
2. Calcolo/stimo i valori di OV, AV ed IV
3. Per i ticket validi (con OV diversa da IV e da FV, ossia con lista delle AV≠0) memorizzo i valori di IV, OV e FV
4. Per ogni commit della repository, controllo se il messaggio di commit contiene l'ID di qualche ticket
5. Per ogni ticket associato, per i file presenti all'interno del commit, setto le classi come buggy per tutte le versioni comprese tra IV e FV

PROGETTAZIONE (5) – ESTRAZIONE DATI DA JIRA

- Per ottenere le versioni di ciascun progetto, si usano le API di Jira, in particolare si effettua una query tramite il seguente url: https://issues.apache.org/jira/rest/api/2/project/ + **projectName**; Questo'ultimo rappresenta il progetto in esame.
- Il risultato, restituito in formato Json, viene analizzato: si prendono le versioni, in particolare il nome e la data, e gli si assegna un indice.
- L'ultima metà delle release viene rimossa, in quanto si è dimostrato che in questo modo il **missing rate** si abbassa al 10%, portando risultati più accurati nella predizione.
- Le release più recenti hanno, logicamente, un numero di classi difettose minore rispetto a quelle più datate, essendo maggiore la probabilità che un bug non sia ancora stato scoperto.
- Sfruttando Jira si possono prelevare anche i ticket di ciascun progetto. Si è usata la query: https://issues.apache.org/jira/rest/api/2/search?jql=project=%22**projectName**%22AND%22issueType%22=%22**Bug**%22AND(%22status%22=%22**closed**%22OR%22status%22=%22**resolved**%22)AND%22resolution%22=%22**fixed**%22&fields=key,versions,resolutiondate,created,fixVersions&startAt=
- La query recupera tutti i ticket con issueType = Bug, status = closed o resolved e resolution = fixed .

PROGETTAZIONE (6) – ESTRAZIONE DATI DA JIRA

- La query produce un JsonObject, che viene analizzato e parsato, cercando di estrapolarne le informazioni necessarie.
- Per ogni ticket viene prelevato l'**ID** (campo "key"), le **Affected Versions** (se presenti, campo "versions") e la **creation date** (campo "created").
- L'ID dei ticket viene utilizzato per trovare i commit effettuati aventi nel messaggio proprio l'ID del ticket, per risolvere il bug ad esso associato.
- La data di creazione del ticket viene utilizzata per calcolare l'**Opening Version**, necessaria per l'applicazione di Proportion, che è la prima versione rilasciata dopo la creazione del ticket.
- Nel caso in cui ci fosse l'AV tra le informazioni del singolo ticket, il valore dell' **Injected Version** si ottiene dalla versione con data di rilascio minore tra tutte le presenti all'interno della lista di Affected Version.
- Per il calcolo della **Fixed Version** viene utilizzata la variabile del ticket resolution date. Si prende l'indice della versione rilasciata dopo la data di risoluzione del ticket.


```
{
    "versions": [{
        "archived": false,
        "releaseDate": "2011-12-07",
        "name": "4.0.0",
        "self": "https://issues.apache.org/jira/rest/api/2/version/12317843",
        "description": "",
        "id": "12317843",
        "released": true
    }],
    "resolutiondate": "2012-04-28T08:12:00.000+0000",
    "created": "2011-08-26T08:09:50.000+0000",
    "fixVersions": [{
        "archived": false,
        "releaseDate": "2012-06-13",
        "name": "4.1.0",
        "self": "https://issues.apache.org/jira/rest/api/2/version/12319145",
        "id": "12319145",
        "released": true
    }]
}
```


Campo "fields" del JsonObject in output dalla query

PROGETTAZIONE (7) – STIMA INJECTED VERSION

Nel caso in cui l'AV non dovesse essere presente, viene effettuata la stima del valore di IV. Per la stima, è prima necessario il calcolo del valore di P per i ticket con IV nota. Per ogni ticket, quindi:

1. Si calcola il valore di $P = \frac{FV - IV}{FV - OV}$
2. Una volta calcolato, lo si associa al ticket
3. Successivamente, si procede alla stima del valore di IV, per i ticket sprovvisti di AV. Per il singolo ticket con indice n:
4. Viene calcolato il valore di P come media tra i ticket con AV presente ed indice compreso tra 0 ed n-1

    - Se non dovessero essere presenti ticket con AV nell'intervallo considerato, stimo IV con OV.

5. Con il valore di P trovato, viene effettuato il calcolo del valore di IV stimato: $IV = FV - (FV - OV) \cdot P$

    - Se l'IV stimata dovesse essere minore di 1 (per valori alti di P), l'IV viene impostata ad 1.

PROGETTAZIONE (8) – UTILIZZO DI GIT

- Inizialmente viene effettuato un clone della repository del progetto da Github, tramite l'API **JGit**, recuperandone tutti i file.
- Si selezionano tutti i file ".java" presenti, per ogni release.
- A questo punto viene fatta l'associazione delle misure, che avviene per la coppia: (**nomeClasseJava**, **NumeroVersione**).
- Viene creata una mappa con chiave la coppia appena presentata e come valore l'array di metriche considerate, inizialmente inizializzate tutte a 0.
- La libreria è stata anche utilizzata per effettuare l'iterazione sui singoli **commit** e l'ottenimento delle modifiche, con relativi file, tra due commit, utilizzando **DiffFormatter** e **DiffEntry**.
- Tramite le DiffEntry, ossia i cambiamenti che avvengono in un file tra un commit ed il precedente, si sono calcolate le metriche.

PROGETTAZIONE (9) – CALCOLO DELLE METRICHE

- **LOC Added**: servendosi di JGit, per ogni file (DiffEntry) all'interno del commit abbiamo a disposizione l'oggetto Edit, che contiene le informazioni sulle singole modifiche effettuate al singolo file. Per il calcolo delle linee di codice aggiunte, vengono considerate le modifiche di tipo Insert ed il numero di linee aggiunte.
- **Max Loc Added**: andando ad aggiornare le metriche relative ad un file, si effettua il controllo tra le nuove linee di codice aggiunte. La variabile relativa alla metrica, inizialmente impostata a 0, viene aggiornata quando necessario.
- **Average Loc Added**: per ogni file, viene calcolata come $\frac{LocAdded}{NumberOfRevisions}$
- **Loc Touched**: utilizzando la libreria JGit, come visto per le LOC Added, sono state considerate le linee di codice coinvolte da modifiche di tipo insert, delete e replace.
- **Number Of Revisions**: viene calcolato come valore incrementale, ogni volta che un file appare in un commit.
- **Number Of Bug Fix:** viene calcolato come il numero di ticket Jira associati al commit in questione.
- **Change Set Size:** la quantità di file presenti nel commit considerato viene calcolato tramite la size della lista di oggetti DiffEntry, presenti nella libreria JGit. Ciascun oggetto indica il singolo file all'interno del commit.
- **Max Change Set Size:** si effettua lo stesso ragionamento di MaxLocAdded. Viene impostata una variabile che si aggiorna di volta in volta.
- **Average Change Set Size:** per ogni file, viene calcolata come $\frac{ChangeSetSize}{NumberOfRevisions}$
- **Bugginess:** Viene impostata a true in due casi: se il file è presente all'interno di un commit con almeno un ticket Jira associato o se il file, in una versione differente, è associato ad un ticket Jira valido, con versione in [IV, FV).

PROGETTAZIONE (10) – MACHINE LEARNING

- In questa fase si prende in input il dataset costruito in precedenza e lo si sfrutta per valutare le prestazioni dei classificatori **Random Forest**, **Naive Bayes** ed **IBK**.
- Si cerca di capire se e quali tecniche di feature selection e balancing aumentano l'accuratezza di un determinato tipo di classificatore.
- Per la validazione del dataset è stata utilizzata la tecnica **Walk Forward**, data la dipendenza temporale delle istanze, per poterne preservare l'ordine.
- Con questa tecnica si eseguono n-1 run, con "n" numero di versioni all'interno del dataset considerato. Per ogni singola run, indicata con "j", vengono considerate tutte le versioni comprese tra 1 e j per il training set e la versione j+1 per il testing set.
- L'ultima iterazione corrisponde alle versioni da 1 a n-1 per il training set e la versione n per il testing set.

PROGETTAZIONE (11) – FEATURE SELECTION

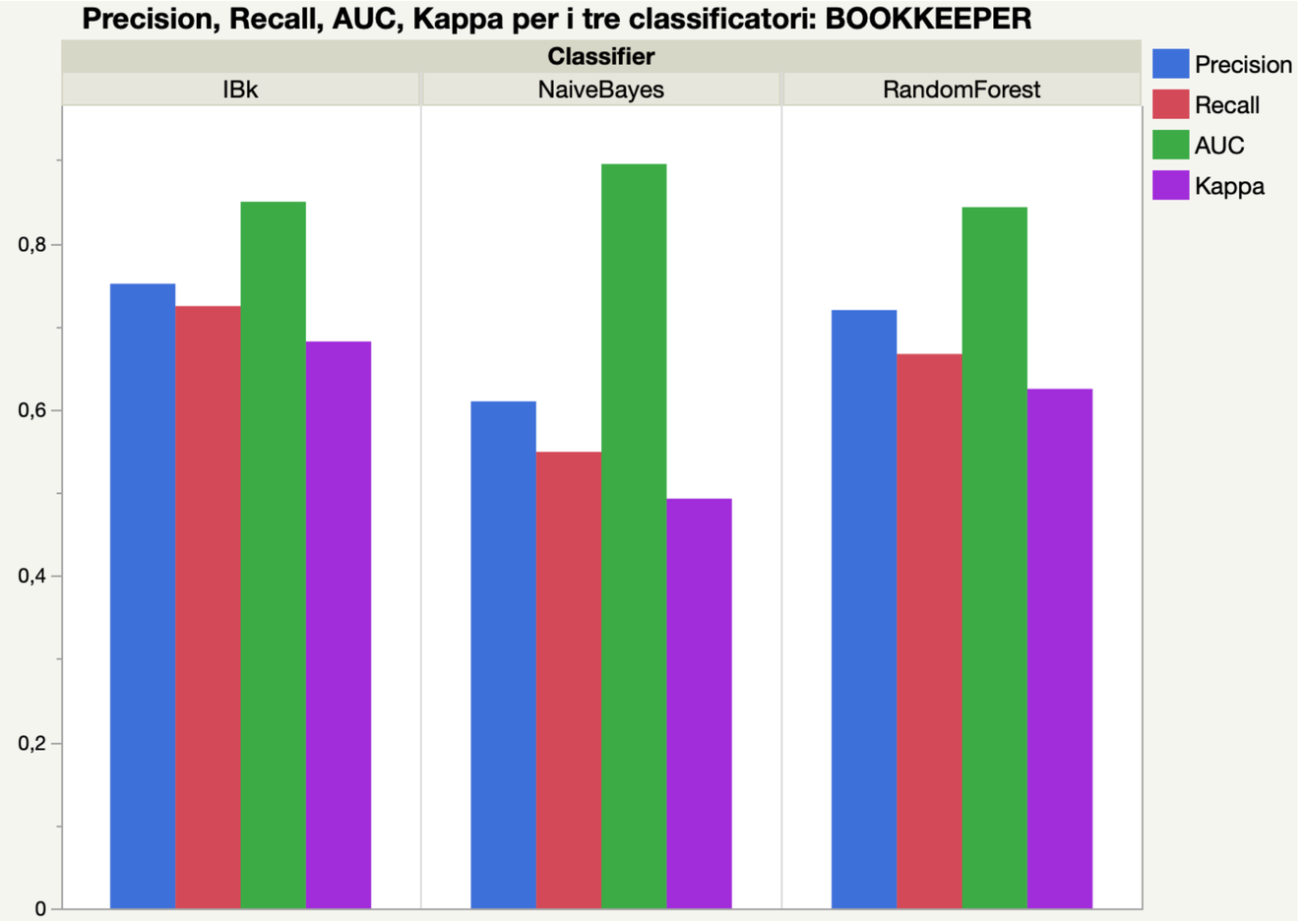
- La **feature selection** permette di ridurre il costo dell'apprendimento, riducendo il numero degli attributi non correlati.
- Offre delle performance migliori rispetto all'utilizzo dell'intero set di attributi.
- Per la valutazione della correlazione dei singoli attributi è stato utilizzato l'oggetto **CFSSubsetEval** offerto da **Weka**.
- L'idea alla base è quella della creazione di un subset contenente attributi che sono altamente correlati tra di loro.
- La ricerca e selezione degli attributi viene effettuata secondo **backward search**. Inizialmente vengono considerati tutti gli attributi del dataset, procedendo con la rimozione di un solo attributo per volta fin quando non si assiste ad un significativo decremento dell'accuratezza.

PROGETTAZIONE (12) – SAMPLING

- Dal dataset di partenza è possibile vedere come ci sia uno sbilanciamento tra il numero di istanze **buggy** e **non buggy**. Le prime, infatti, sono sensibilmente meno presenti delle seconde.
- Per evitare che i classificatori abbiano un'alta accuratezza sulle istanze del secondo tipo, ma bassa sul primo, vengono applicate differenti tecniche di **sampling** per andare ad effettuare un bilanciamento del training set.
- **Undersampling:** per bilanciare il numero delle classi, vengono estratte dalla classe maggioritaria un numero di istanze pari al numero della classe minoritaria. Per applicare l'undersampling si è utilizzato l'oggetto **Resample** offerto da Weka, andando a specificare le opzioni *-M 1.0*.
- **Oversampling:** per bilanciare il numero delle classi, vengono ripetute le istanze della classe minoritaria fin quando il loro numero non sarà uguale a quello della classe maggioritaria. Come per l'undersampling, si è utilizzato l'oggetto **Resample**, andando a specificare le opzioni *-B 1.0 -Z 2**percentuale_classe_maggioritaria.
- **SMOTE:** La tecnica SMOTE è un raffinamento dell'oversampling, in quanto l'estensione della classe minoritaria non avviene con la ripetizione delle istanze, bensì con la generazione di queste ultime in modo sintetico. Per l'ottenimento di questo risultato vengono utilizzati degli algoritmi basati sulle similitudini esistenti tra le istanze della classe minoritaria, per andare a generarne di nuove in numero necessario.

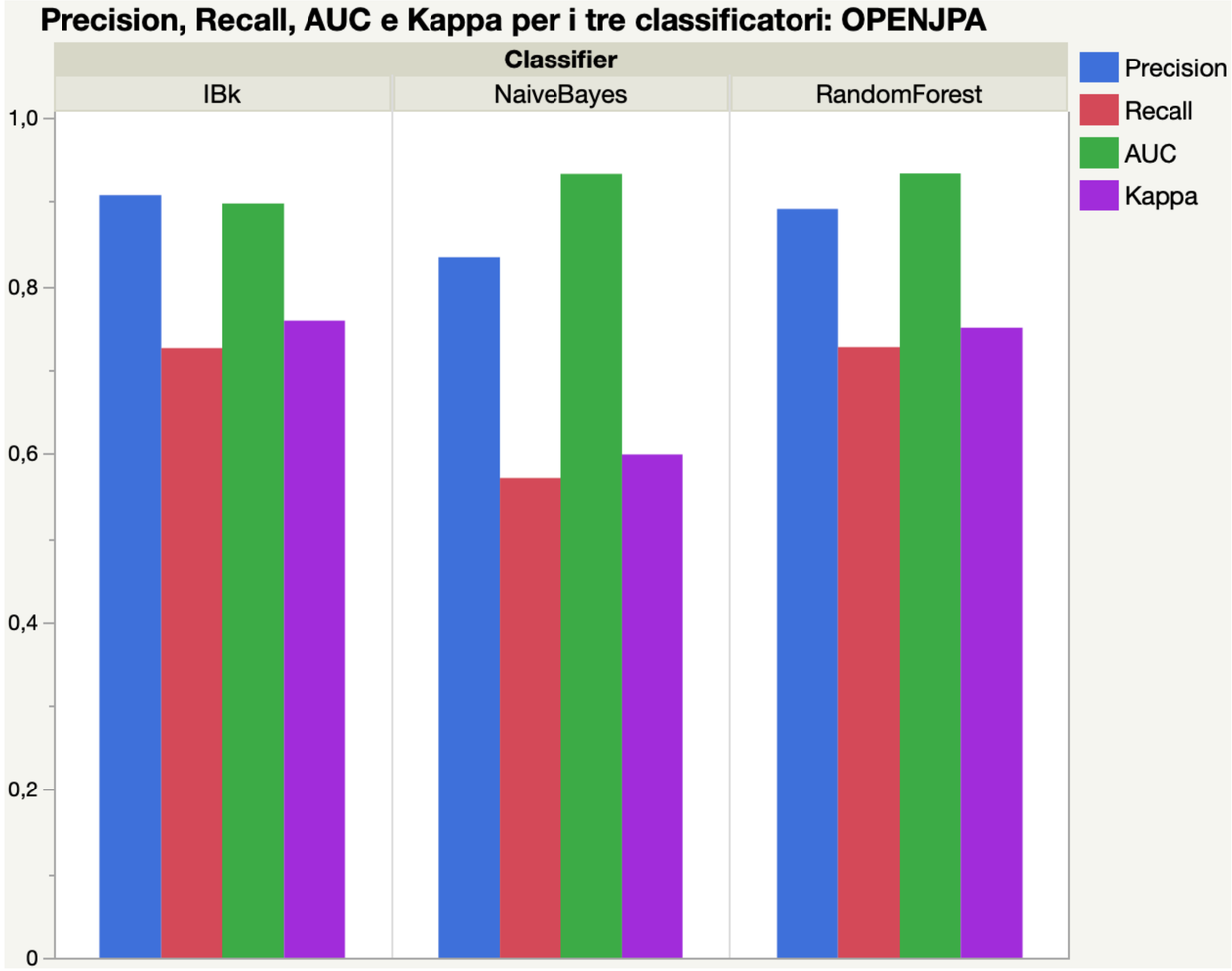
RISULTATI (1) – BOOKKEEPER

- Si può notare come il valore di AUC sia molto elevato per tutti e 3 i classificatori, per Naive Bayes supera addirittura 0.9
- Per tutte le altre metriche, però, Naive Bayes è il classificatore che si comporta in maniera peggiore.
- Molto simili tra loro IBK e Random Forest, con buoni valori di precision e recall, stando a significare che sono state classificate correttamente molte istanze come positive.
- IBK presenta, confrontandolo con Random Forest, valori leggermente migliori per ogni metrica, affermandosi come il classificatore che si comporta meglio per il progetto BookKeeper.



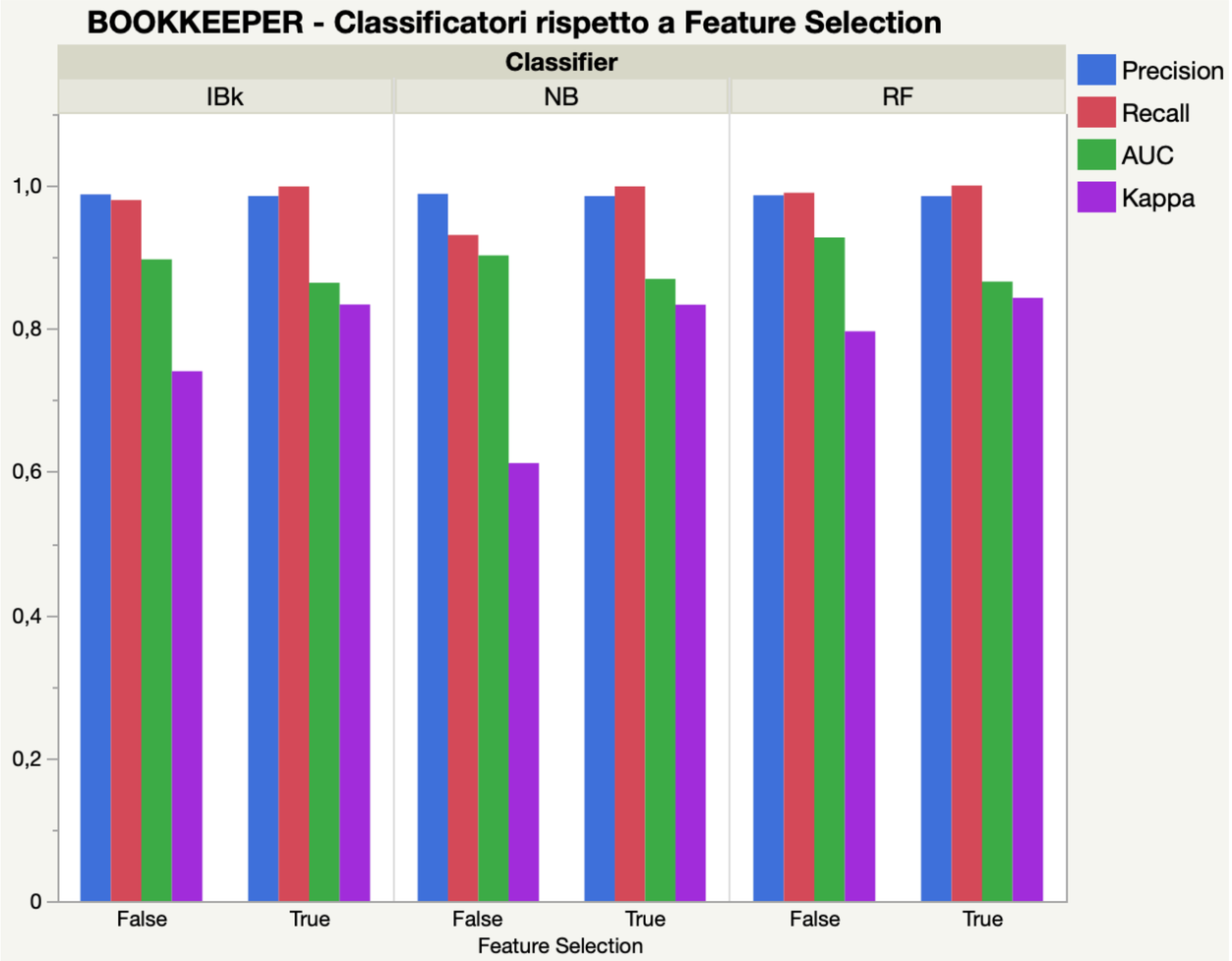
RISULTATI (2) – OPENJPA

- Anche qui si può notare come il valore di AUC sia molto elevato per tutti e 3 i classificatori, in particolare Naive Bayes e Random Forest presentano i valori più alti, che superano 0.9
- Naive Bayes ha il valore più basso per tutte le altre metriche, esattamente come avveniva per BookKeeper. Tra queste, la precision è sicuramente il valore migliore, indicando che il classificatore valuta correttamente le istanze come positive. In combinazione con la recall, però, che è la metrica con il valore più basso, si intuisce che il classificatore valuta correttamente le istanze come positive, ma con tanti falsi negativi in mezzo.
- IBk e Random Forest presentano comportamenti molto simili tra loro, con valori praticamente identici in termini di precision, recall e kappa.



RISULTATI (3) – BOOKKEEPER: FEATURE SELECTION

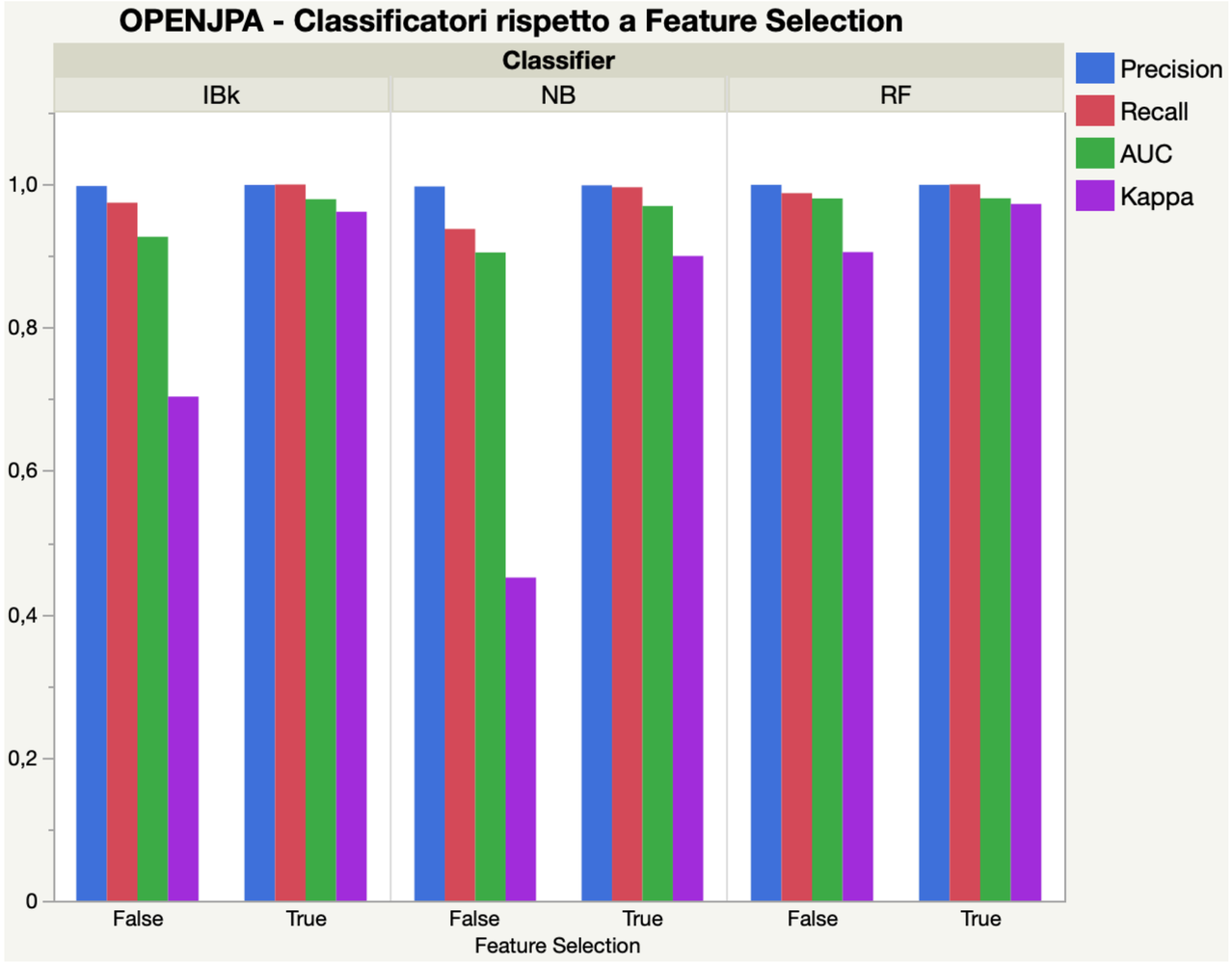
- Si può notare come, senza l'applicazione di Feature Selection, metriche come Kappa decrescono in maniera significativa, per tutti i classificatori meno Random Forest.
- La precision è una metrica che rimane pressoché invariata, mantenendo valori molto elevati anche senza feature selection.
- Con Best First si ha un miglioramento della recall, per ogni classificatore, ma si ha una diminuzione globale di AUC.
- La recall diventa la metrica più elevata con l'applicazione di Best First.
- È veramente complicato scegliere il classificatore che si comporta in maniera migliore.



True indica l'applicazione di Best First, false che non è presente alcuna tecnica di Feature Selection

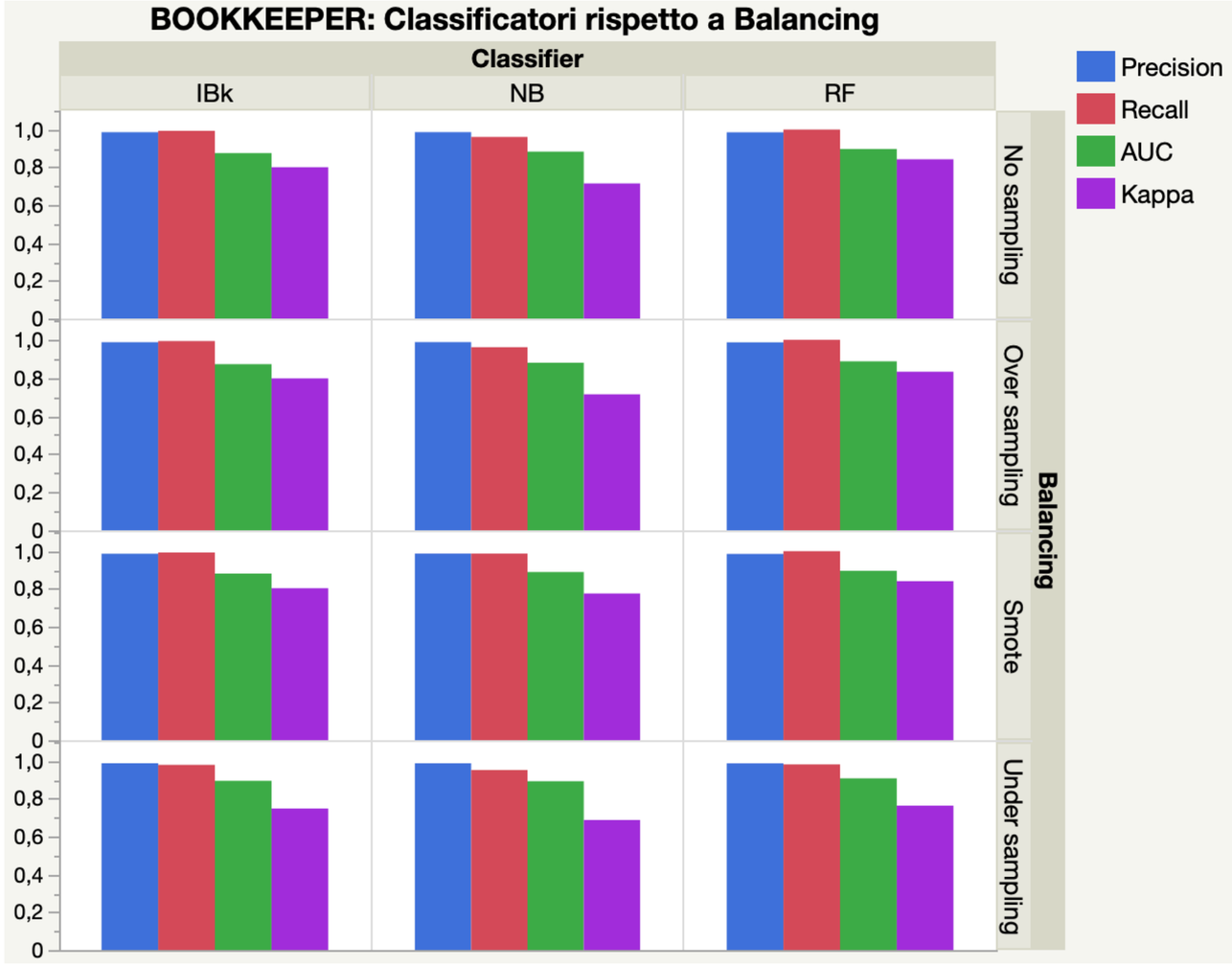
RISULTATI (4) – OPENJPA: FEATURE SELECTION

- Si può notare come l'applicazione di Best First porti ad un miglioramento complessivo di tutte le metriche.
- I miglioramenti più evidenti si notano in termini di AUC e Kappa, in particolare per Naive Bayes quest'ultima migliora sensibilmente.
- Precision e Recall mantengono sempre valori molto elevati. Quest'ultima migliora leggermente con Best First per Naive Bayes.
- Difficile scegliere se il classificatore che si comporta meglio sia IBk o Random Forest.



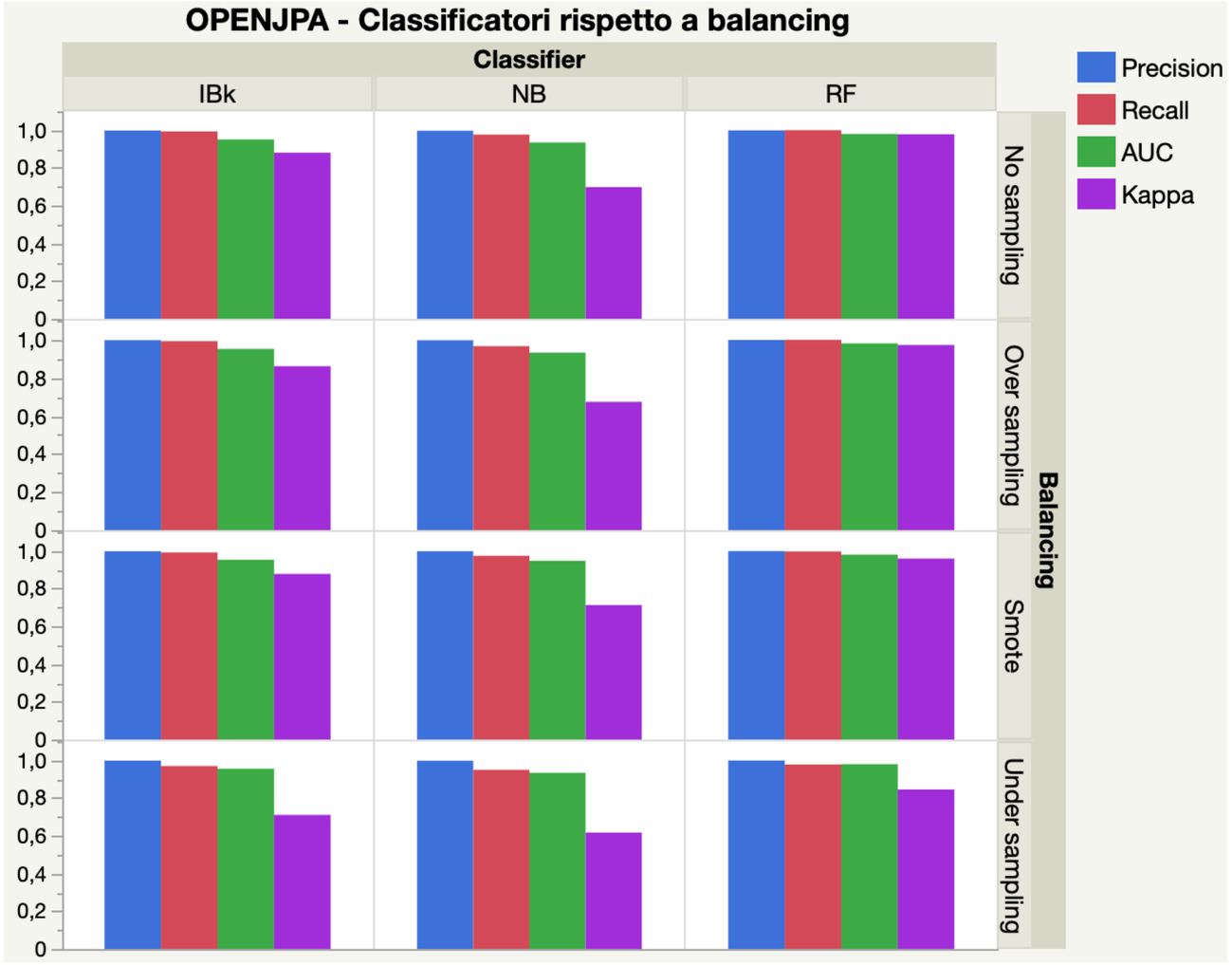
RISULTATI (5) – BOOKKEEPER: BALANCING

- L'applicazione delle tecniche di balancing porta risultati non troppo diversi per ciascuno dei classificatori.
- Precision e recall si mantengono molto elevate in ogni caso, raggiungendo i valori massimi per SMOTE, per tutti e 3 i classificatori.
- La tecnica peggiore, che porta ad una diminuzione importante dei valori di Kappa e AUC è l'undersampling.
- È veramente difficile scegliere la tecnica migliore: in ogni caso, il classificatore che tende a comportarsi peggio è Naive Bayes.



RISULTATI (6) – OPENJPA: BALANCING

- Si può subito osservare come Random Forest abbia il miglior comportamento con ogni tecnica, mostrando solo una diminuzione di kappa con l'undersampling.
- I valori di precision e recall continuano a rimanere elevatissimi, così come il valore di AUC.
- Naive Bayes continua a rimanere il classificatore più simile ad un dummy, avendo la metrica Kappa sempre discretamente bassa.
- È impossibile eleggere un candidato migliore, essendo le varie distribuzioni tutte molto simili tra loro.



CONCLUSIONI

- Dall'analisi effettuata è possibile evincere come non ci sia una configurazione migliore in assoluto.
- Nei grafici finali si è osservato come ogni metrica, per quasi ogni classificatore, presenti valori veramente molto alti.
- Si sono osservati miglioramenti per entrambi i progetti con l'applicazione di feature selection, così come con il balancing, ma in maniera meno evidente.

LINKS

- Github: https://github.com/valeriomoroni95/ISW2_Deliverable_Buggyness
- SonarCloud: https://sonarcloud.io/project/overview?id=valeriomoroni95_ISW2_Deliverable_Buggyness