ISW2

Machine Learning for Software Engineering

Università degli studi di Roma "Tor Vergata"

Lorenzo Valeriani

lorenzo.valeriani.459326@alumni.uniroma2.eu

0306361

Agenda

- Introduzione ed obiettivi
- Progettazione e scelte implementative
- Risultati e discussione
- Minacce alla validità

Introduzione ed obiettivi

Il **testing** è una parte fondamentale della produzione di un software. Più i progetti crescono, maggiore è il suo **costo** in termini economici e temporali.

- Indicare quali componenti è più probabile che contengano bug può aiutare ad **ottimizzare** le risorse investite nella fase di test.

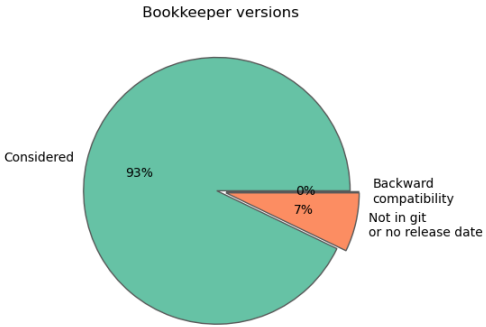
- Strumenti di **Machine Learning** permettono di sviluppare modelli predittivi a partire da dati **passati**. Il punto chiave è la corretta misurazione del software.

- È opportuno paragonare diversi **classificatori** e tecniche di **feature selection**, **sampling** e **cost sensitivity** al fine di trovare la combinazione che più si adatta al progetto su cui si sta lavorando.

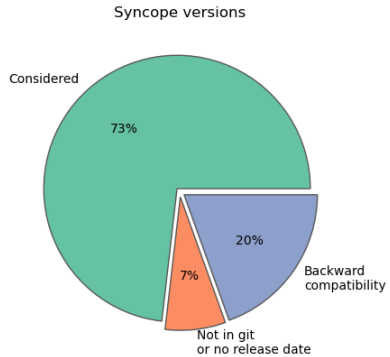
Progettazione e **scelte implementative**: Release

Le Release sono ottenute con la REST API di JIRA. Di quelle ritornate in formato **JSON** dall'API, sono considerate solo le release che hanno il campo "releaseDate" e sono presenti anche su git.

Dalle rimanenti sono ulteriormente rimosse le release pubblicate per retrocompatibilità (es. 1.0.8 rilasciata dopo la 1.1.0)



Tot = 14



Tot = 82

Progettazione e **scelte implementative**: Release

Per ogni release si tiene traccia sia della data di rilascio indicata da JIRA, usata per eseguire operazioni sui **ticket**, sia di quella calcolata tramite git, usata per le operazioni sui **commit**.

Per ridurre al minimo le conseguenze causate dal problema dello **snoring**, le release etichettate nella slide precedente come "considered" sono divise in due parti:

- Il primo 50%, meno influenzato dal suddetto problema, viene usato come dataset.
- Il secondo 50% viene usato solamente per tenere traccia di eventuali bugfix che riguardino anche le versioni usate come dataset.

Si arriva ad un totale di 6 release considerate per BookKeeper e di 30 per Syncope.

Progettazione e **scelte implementative**: Bug

Anche la lista dei **bug** è ottenuta tramite JIRA. Nello specifico, si chiedono all'API tutte le issue di tipo "Bug", con risoluzione "fixed" e stato "resolved" o "closed".

Per ogni bug si considerano come:

- **Opening Version** la prima versione successiva alla data di creazione del ticket.
- **Fixed Version** la prima versione successiva al fix commit.
- **Injected Version** la prima delle **Affected Version**, se presenti, o il valore calcolato con **Proportion.**

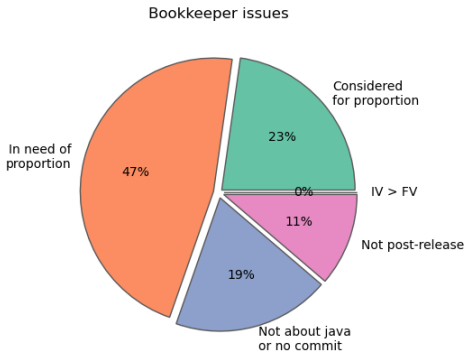
Si escludono le issue con IV = FV (non post-release), IV > FV e quelle che non hanno fix commit o che non riguardano file Java.

Progettazione e **scelte implementative**: Proportion

La variante di proportion utilizzata è **moving window** ovvero:

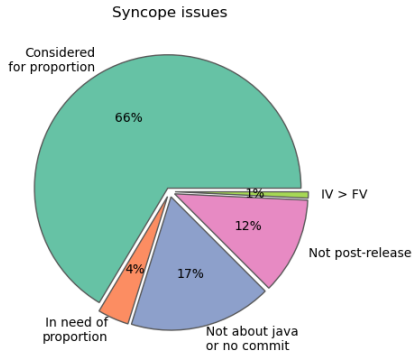
- Si calcola P come media dei rapporti (FV - IV) / (FV - OV) per i bug con AV note che cadono all'interno della moving window della issue corrente.
- Si calcola IV come FV – (FV – OV) ∗ P.

La dimensione della moving window è variabile e frutto di un trade-off tra quantità delle issue considerate e rilevanza dovuta alla vicinanza temporale. Si usa come size 1% di tutte le issue per Syncope e 2% per BookKeeper.



Tot = 435

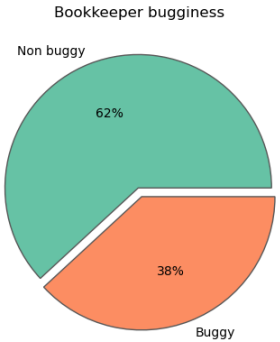
Per entrambi i progetti si ha sempre:
1 <= P <= 2

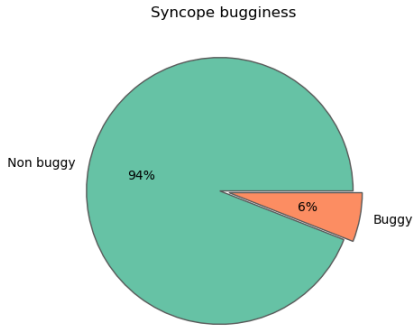


Tot = 793

Progettazione e **scelte implementative**: Bugginess

Il **Linkage**, ovvero la quantità di bug che hanno almeno un commit associato, è una misura di importanza fondamentale. Questo perché, oltre a stabilire la FV del bug tramite il fix commit, tutti i commit associati ad una data issue permettono di identificare i file affetti dal bug in questione. Tutti i file interessati dai vari commit che hanno nella descrizione la chiave della issue (es. SYNCOPE-42 o BOOKKEEPER-42) sono infatti considerati buggy dalla IV inclusa alla FV esclusa.





Progettazione e **scelte implementative**: Metriche

Il dataset contiene **16 metriche**:

Version	File Name	LOC	LOC_touched	NR	NFIX	NAuth	LOC_Added	MAX_LOC_added	AVG_LOC_added	Churn	MAX_Churn	AVG_Churn	ChgSetSize	MAX_ChgSet	AVG_ChgSet	Age	Weighted Age	Buggy

La maggior parte di queste è calcolata in modo semplice sfruttando l'eseguibile di git, invocato con il parametro --numstat. LOC si ottiene con un programma apposito (tokei), il quale è in grado di riconoscere anche commenti a linea singola, multilinea e molto altro.

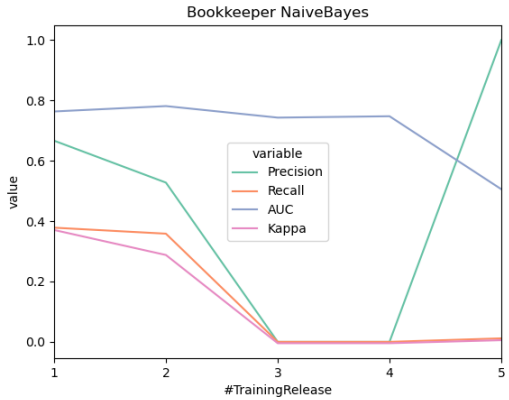
Progettazione e **scelte implementative**: Walk Forward

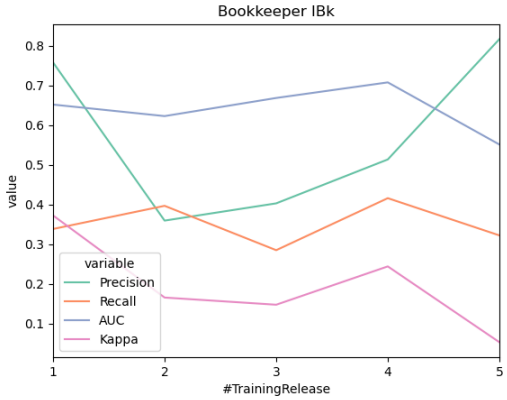
Si usa Walk Forward come tecnica di valutazione dei classificatori. Si mantiene l'ordinamento temporale delle release e si eseguono diverse run. Alla run n si considerano le prime n - 1 release come training set e la release n come testing set. Le release da n + 1 in poi non devono influenzare le iterazioni precedenti.

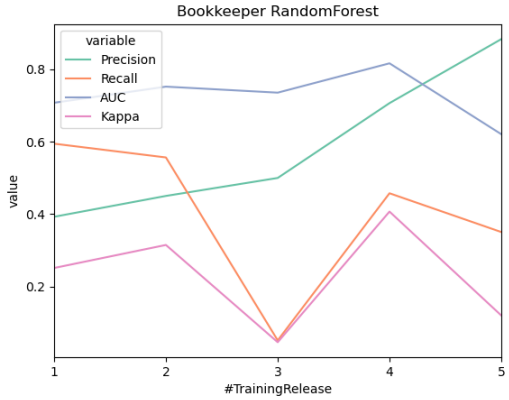
Si valutano:

- Precision
- Recall
- AUC
- Kappa

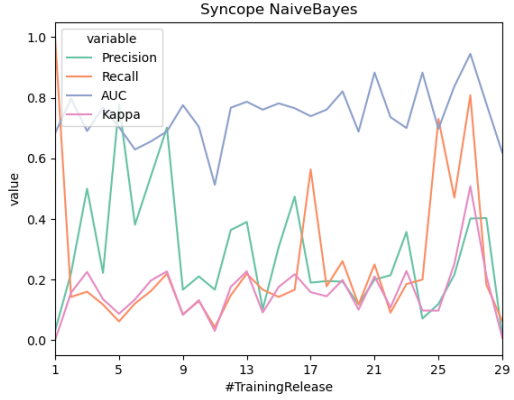
Risultati e **discussione**: Walk Forward

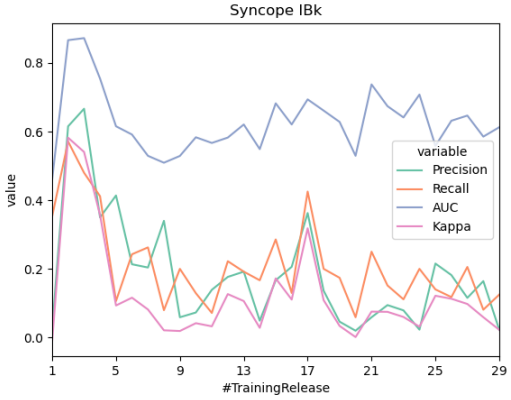


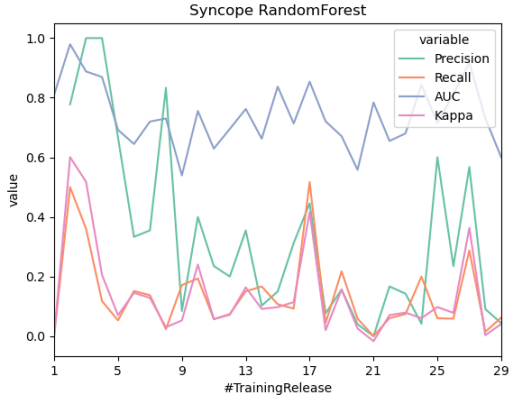




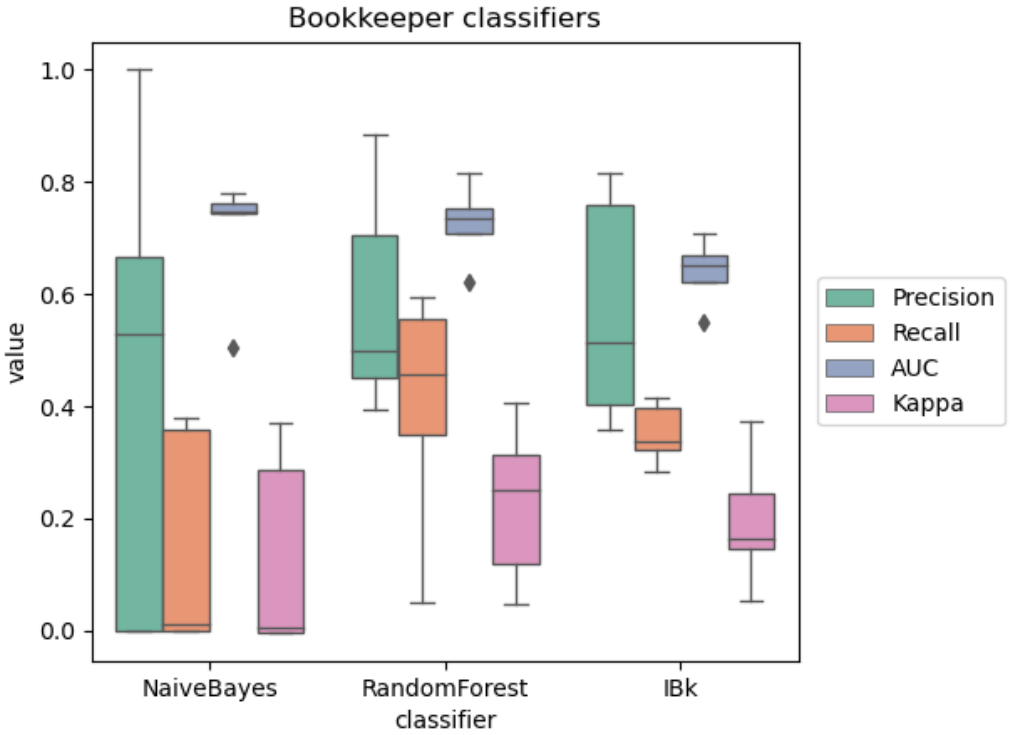
Andamento delle metriche durante Walk Forward





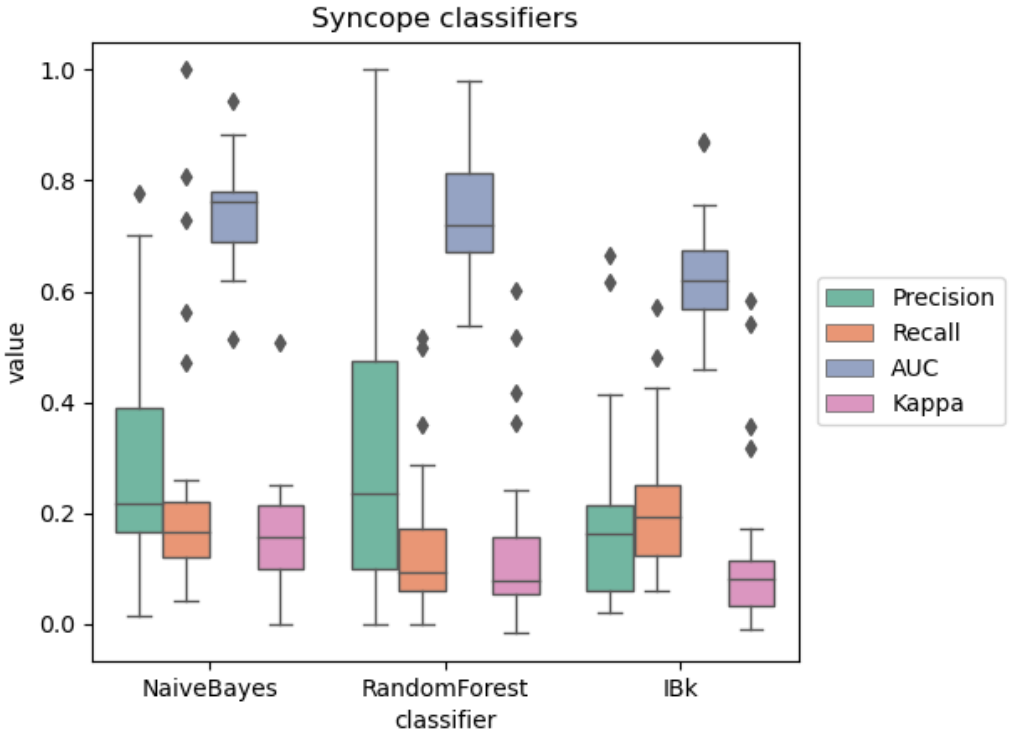


Risultati e **discussione**: BookKeeper Classifiers



Classifier	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NaiveBayes	0.438889	0.149693	0.708333	0.131023
Random Forest	0.586594	0.402279	0.726447	0.228152
IBk	0.569642	0.351156	0.639864	0.196114

Risultati e **discussione**: Syncope Classifiers



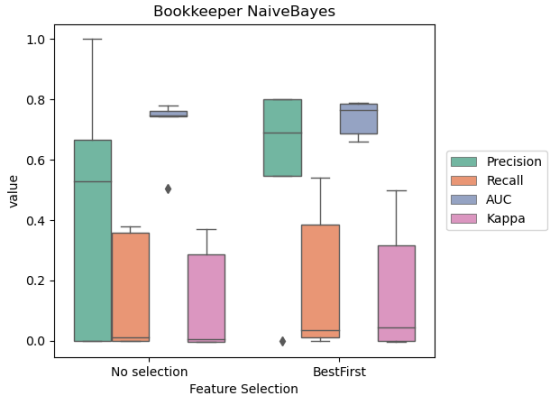
Classifier	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NaiveBayes	0.284560	0.247850	0.743377	0.159514
Random Forest	0.336172	0.138368	0.740437	0.137321
IBk	0.186441	0.211822	0.629130	0.121796

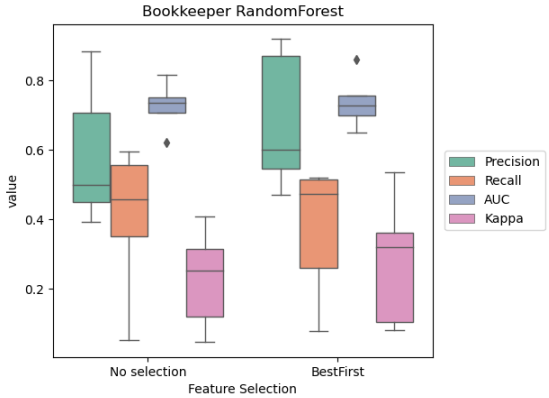
Risultati e **discussione**: Classificatori

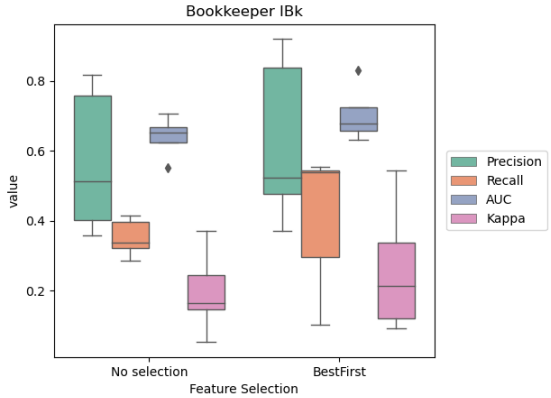
Per **BookKeeper** è semplice eleggere **Random Forest** come migliore in termini di media, 1Q, mediana e 3Q per tutte le metriche. IBk vince su Naive Bayes per tutte le metriche tranne AUC, per la quale quest'ultimo ha prestazioni simili a quelle di Random Forest.

Per **Syncope** la situazione è diversa: Random Forest e Naive Bayes hanno prestazioni simili, ma il primo vince per precision, il secondo per recall e kappa. Data la maggior importanza dei falsi negativi rispetto ai falsi positivi nel contesto in studio, è da preferire **Naive Bayes**. IBk perde in tutte le metriche tranne recall, per la quale mostra una mediana superiore rispetto agli altri classificatori. Per questo dataset, precision, recall e kappa sono inferiori rispetto a BookKeeper. Ciò suggerisce che il progetto sia più complicato da predire.

Risultati e **discussione**: BookKeeper Feature Selection





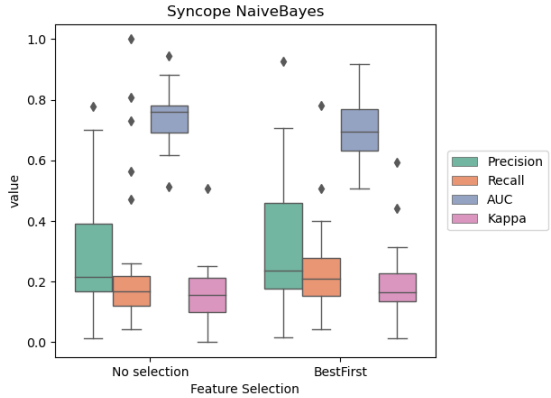


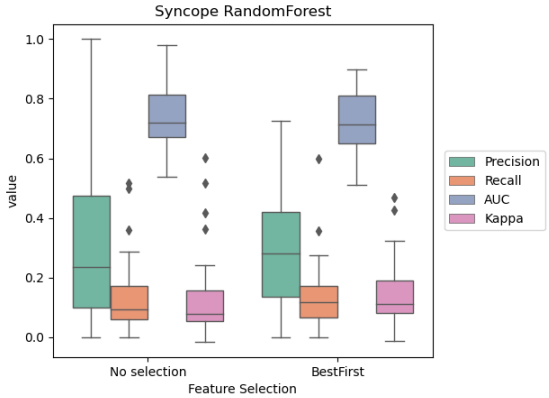
Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.438889	0.149693	0.708333	0.131023
BestFirst	0.567264	0.194682	0.737150	0.171124

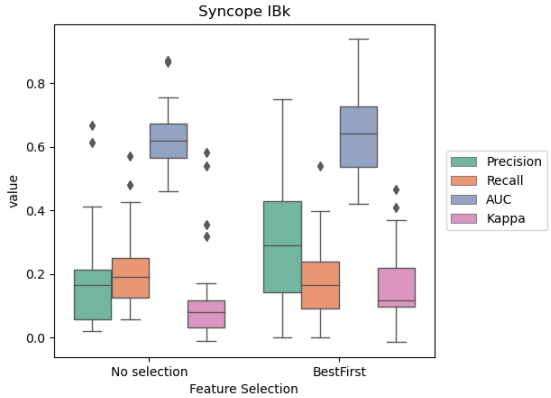
Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.586594	0.402279	0.726447	0.228152
BestFirst	0.680875	0.368876	0.739113	0.280971

Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.569642	0.351156	0.639864	0.196114
BestFirst	0.624898	0.406629	0.704041	0.261796

Risultati e **discussione**: Syncope Feature Selection







Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.284560	0.247850	0.743377	0.159514
BestFirst	0.294953	0.231582	0.706180	0.188336

Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.336172	0.138368	0.740437	0.137321
BestFirst	0.302062	0.136390	0.713387	0.141395

Selection	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSelection	0.186441	0.211822	0.629130	0.121796
BestFirst	0.295657	0.174957	0.645047	0.160612

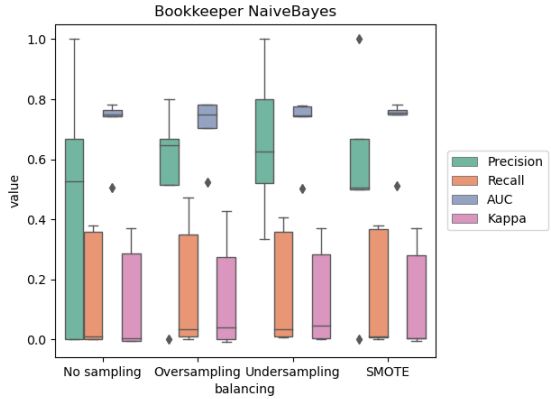
Risultati e **discussione**: Feature Selection

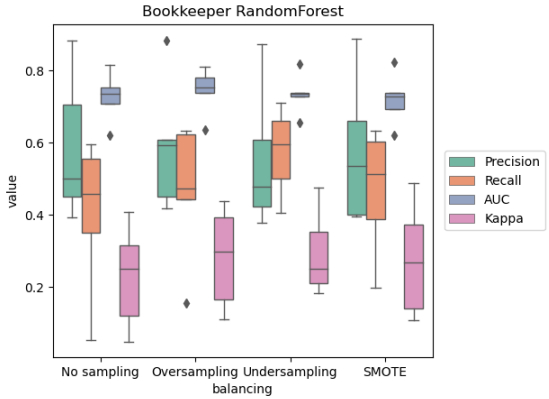
Per **BookKeeper** la Feature Selection migliora le metriche di tutti i classificatori, fatta eccezione per la recall di **Random Forest**, che cala leggermente. In questo caso, il paragone con gli altri classificatori non è più così netto: **IBk** può essere considerato una valida alternativa, visto il suo valore di recall. Probabilmente, il miglioramento è dovuto al fatto che 16 attributi sono un numero molto elevato e che molti di questi sono correlati. Il beneficio di questa tecnica non è quindi solo temporale in questo caso.

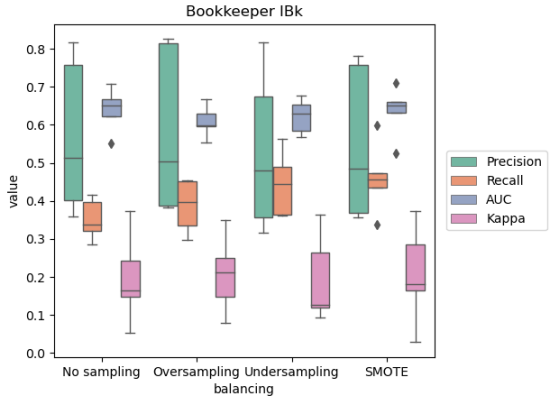
Gli attributi più considerati sono "LOC" > "NFix" > "ChgSetSize" > ... per Syncope
"LOC" > "LOC_Touched" > "NFix" > ... per BookKeeper

Per **Syncope**, generalmente, alcune metriche tendono ad aumentare, mentre altre diminuiscono. La recall, nello specifico, diminuisce sempre. Per **Naive Bayes** si riduce anche AUC, ma precision e Kappa aumentano. Si è davani quindi a un compromesso in cui si deve tener conto anche di una necessità di tempo minore per misurare il progetto ed eseguire il modello.

Risultati e **discussione**: BookKeeper Balancing





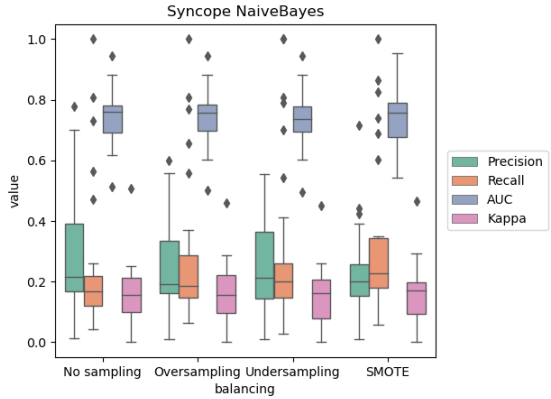


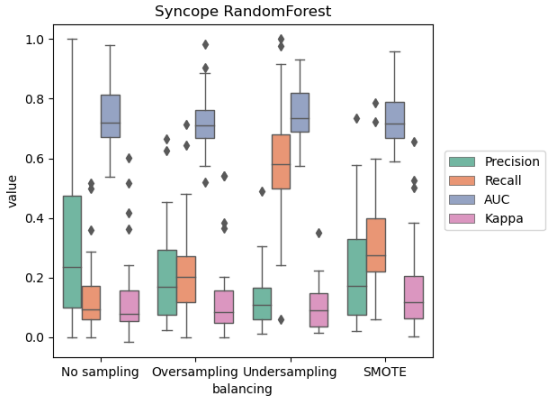
Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.438889	0.149693	0.708333	0.131023
Over-sampling	0.525741	0.173621	0.708358	0.147005
Under-sampling	0.655776	0.163403	0.709686	0.140968
SMOTE	0.534632	0.152988	0.711546	0.131346

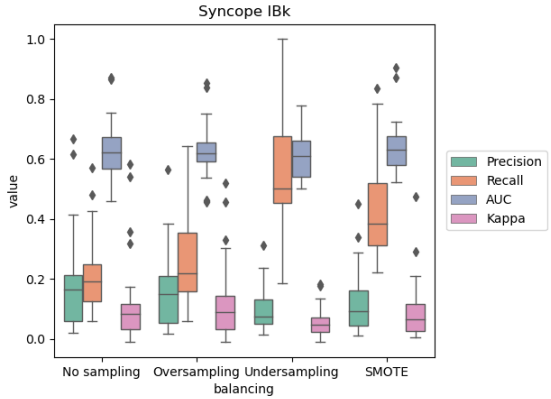
Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.586594	0.402279	0.726447	0.228152
Over-sampling	0.590689	0.465614	0.743431	0.280913
Under-sampling	0.552447	0.574572	0.734511	0.294720
SMOTE	0.576189	0.467554	0.721049	0.275750

Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.569642	0.351156	0.639864	0.196114
Over-sampling	0.582701	0.386828	0.609351	0.208098
Under-sampling	0.528367	0.444713	0.622640	0.193840
SMOTE	0.549948	0.459961	0.635521	0.206043

Risultati e **discussione**: Syncope Balancing







Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.284560	0.247850	0.743377	0.159514
Over-sampling	0.242354	0.283476	0.740145	0.160975
Under-sampling	0.237452	0.308753	0.736946	0.151323
SMOTE	0.224217	0.326480	0.742832	0.158085

Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.336172	0.138368	0.740437	0.137321
Over-sampling	0.211695	0.241279	0.727703	0.138238
Under-sampling	0.125667	0.589077	0.741385	0.101594
SMOTE	0.217512	0.333168	0.733307	0.172951

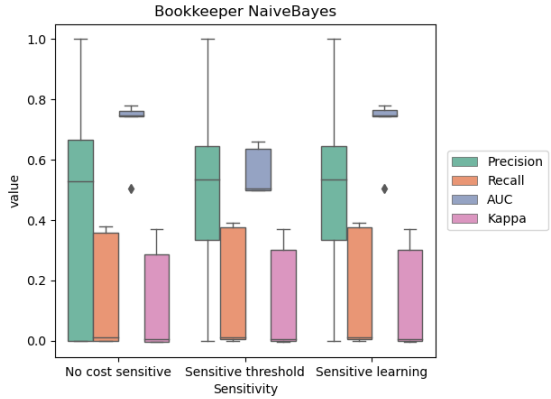
Balancing	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
NoSampling	0.186441	0.211822	0.629130	0.121796
Over-sampling	0.164290	0.270140	0.626038	0.119696
Under-sampling	0.095836	0.553552	0.616487	0.055905
SMOTE	0.123859	0.415566	0.641232	0.090700

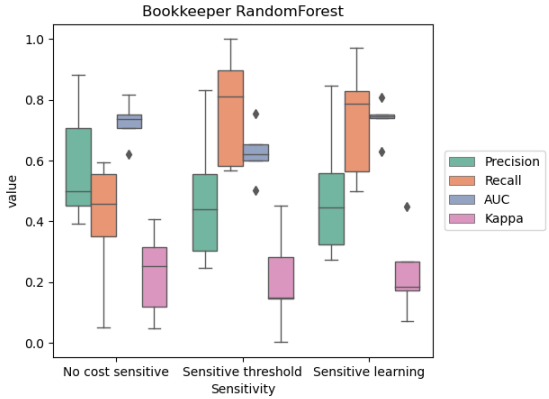
Risultati e **discussione**: Balancing

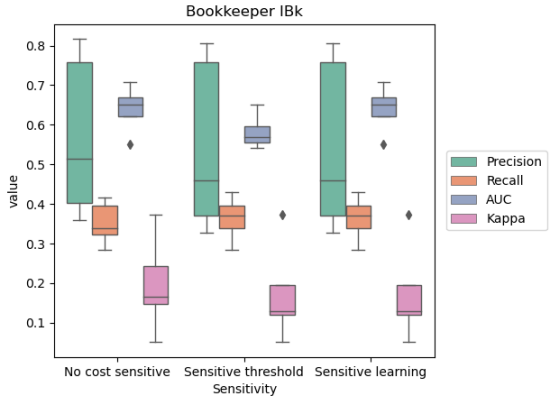
In accordo con la teoria, ogni metodologia di balancing produce un aumento di recall. Le altre metriche a volte aumentano, altre si riducono. Oversampling, contrariamente a quanto ipotizzabile, causa sempre un aumento di precision. Anche in questo caso, il classificatore che mostra i risultati generalmente preferibili in questo ambito è **Random Forest**. La configurazione con Oversampling mostra il valore più alto di AUC, mentre quella con Undersampling ha recall e kappa massimi. La precision più elevata è della configurazione con undersampling di Naive Bayes, ma le altre metriche sono tutte inferiori a quelle di Random Forest. È opportuno notare come per **BookKeeper** il dataset di base non sia eccessivamente sbilanciato.

Anche per **Syncope** il balancing porta ad un aumento di recall. Per questo progetto il dataset è più numeroso, ma molto sbilanciato. Generalmente, l'applicazione di Undersampling porta un aumento considerevole della recall, ma anche una diminuzione di kappa, che è già molto bassa come metrica. Sia Naive Bayes che Random Forest hanno buone performance. Per entrambi i classificatori SMOTE aumenta la recall più di Oversampling, ma per Naive Bayes causa anche una maggiore riduzione di precision. Il calo di precision è già un risultato dell'applicazione di tecniche di balancing, bisogna quindi decider quanto questo calo sia tollerabile.

Risultati e **discussione**: BookKeeper Cost Sensitivity





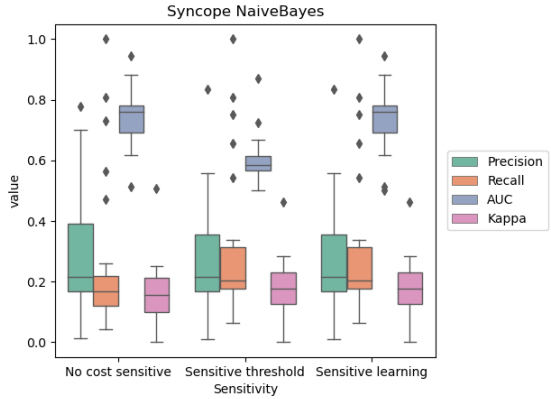


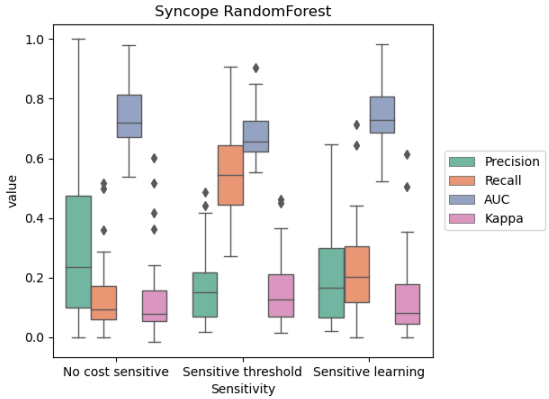
Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.438889	0.149693	0.708333	0.131023
Sensitive Threshold	0.502222	0.157577	0.560263	0.134867
Sensitive Learning	0.502222	0.157577	0.708374	0.134867

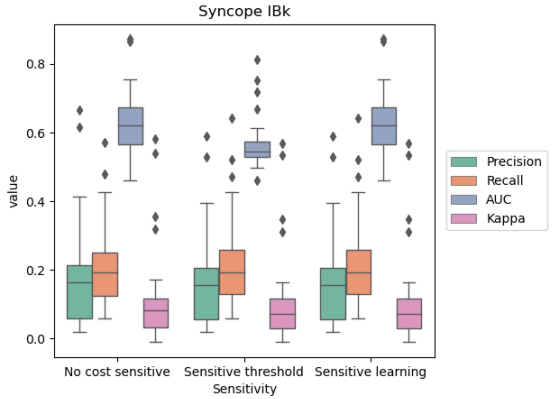
Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.586594	0.402279	0.726447	0.228152
Sensitive Threshold	0.475263	0.771532	0.626234	0.206418
Sensitive Learning	0.490493	0.731422	0.735162	0.229091

Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.569642	0.351156	0.639864	0.196114
Sensitive Threshold	0.544033	0.363828	0.582624	0.173689
Sensitive Learning	0.544033	0.363828	0.639853	0.173689

Risultati e **discussione**: Syncope Cost Sensitivity







Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.284560	0.247850	0.743377	0.159514
Sensitive Threshold	0.265106	0.290762	0.598379	0.174372
Sensitive Learning	0.265106	0.290762	0.737110	0.174372

Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.336172	0.138368	0.740437	0.137321
Sensitive Threshold	0.167523	0.555907	0.675961	0.160217
Sensitive Learning	0.214855	0.236808	0.739813	0.138584

Cost Sensitivity	AVG. Precision	AVG.Recall	AVG.AUC	AVG.Kappa
No Cost Sensitive	0.186441	0.211822	0.629130	0.121796
Sensitive Threshold	0.174378	0.221431	0.569060	0.118006
Sensitive Learning	0.174378	0.221431	0.629130	0.118006

Risultati e **discussione**: Cost Sensitivity

Come per Balancing, anche per Cost Sensitivity la recall aumenta sempre. In questo caso si paga con una diminuzione delle altre metriche, fatta eccezione per la precision di Naive Bayes che aumenta. Per Naive Bayes e IBk non c'è differenza tra Sensitive Learning e Sensitive Threshold. Per Random Forest Sensitive Learning mostra precision e kappa migliori di Sensitive Threshold, il quale vince per recall. Rispetto al balancing, la recall aumenta considerevolmente per Random Forest, ma non per gli altri classificatori. Le altre metriche sono generalmente inferiori.

Matrice dei costi

CTP = 0	CFN = 10
CFP = 1	CTN = 0

Analogamente, per **Syncope** la recall aumenta in genere al prezzo delle altre metriche**.** Sensitive Threshold equivale a Sensitive Learning per Naive Bayes e IBk. Con **Naive Bayes** la recall non aumenta tanto quanto nel caso di Balancing, ma le altre metriche non diminuiscono allo stesso modo. Per Random Forest con Sensitive Threshold la recall aumenta considerevolmente e anche kappa aumenta, ma la precision si dimezza.

Minacce alla Validità

Oltre ai problemi del linkage e delle release scartate esposti in precedenza, è possibile evidenziare altre problematiche:

- Non necessariamente una issue viene chiusa sempre con un commit contenente la key. In particolare, i progetti stanno abbandonando JIRA in favore di GitHub.
- Lo studio si concentra sui commit raggiungibili dalla branch master. Eventuali altre branch sono considerate solo in caso di merge su master. Anche i merge, però, possono causare problemi.
- I rename e i cambi di directory sono parzialmente gestiti. Si tiene traccia di entrambi, ma può capitare di perderne qualcuno. È possibile che git indichi queste operazioni come la rimozione di un file e l'aggiunta di uno nuovo. Con la perdita della storia di un file non si perdono solo le metriche, ma anche informazioni correlate alla bugginess.

Conclusioni

- Come spesso enfatizzato a lezione, non c'è una soluzione ottima, ma ci sono diverse scelte possibili frutto di compromessi (**no silver bullet**).
- I risultati sono per la maggior parte in accordo con la teoria.
- Si ha AUC > 1 / 2 e kappa > 0, quindi le feature sono indicative del problema studiato ed i classificatori si comportano meglio di uno randomico (es. ZeroR).
- Le diverse configurazioni sono sempre paragonate alle configurazioni base (no selection, no sampling, no sensitivity).

Link:
https://github.com/Lorenzoval/isw2_ml_for_se_deliverable
https://sonarcloud.io/summary/new_code?id=Lorenzoval_isw2_ml_for_se_deliverable