Machine Learning & Software Engineering

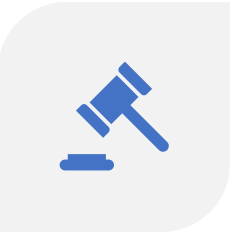
Matteo Ferretti – 0300049

Outline

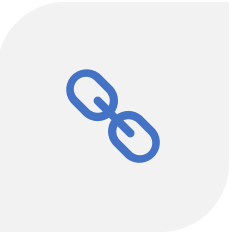
INTRODUZIONE

PROGETTAZIONE

RISULTATI



CONCLUSIONE



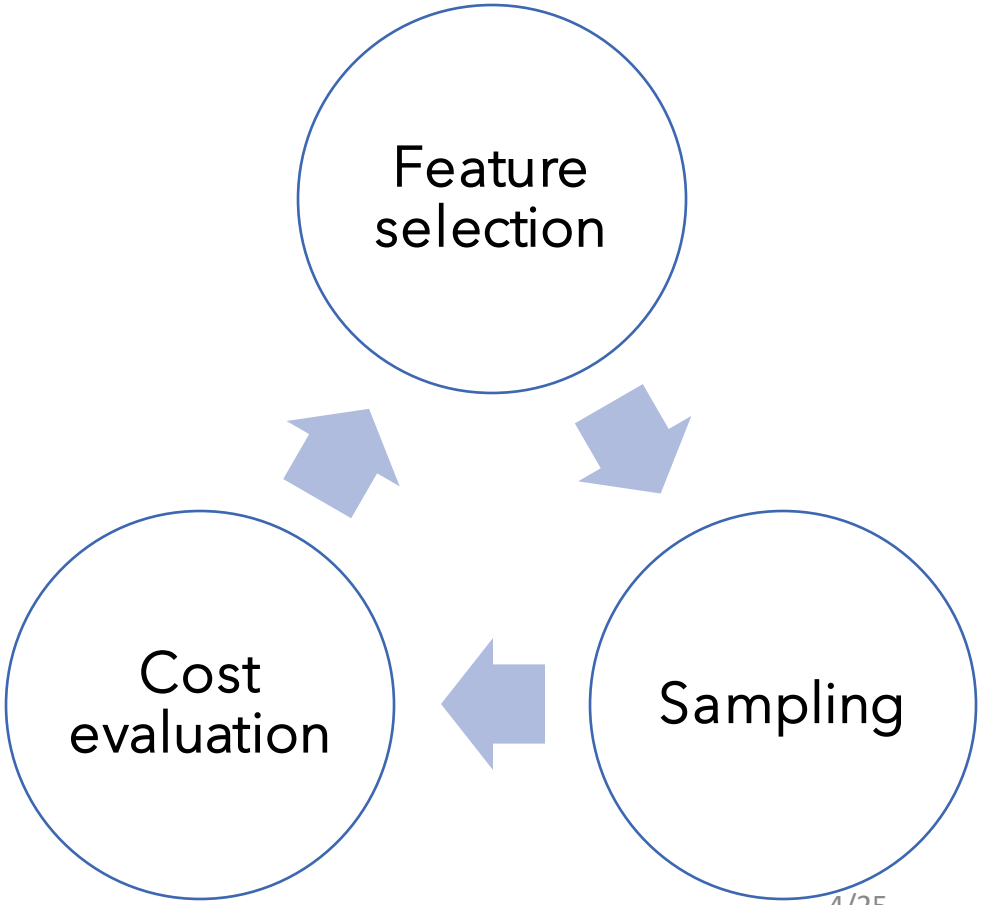
LINKS

Introduzione

- Nello sviluppo software, al fine di migliorare qualità e produttività si mettono in pratica tecniche di **software analytics**, attuate tramite tecniche di software engineering e machine learning.
- La **previsione dei difetti** consente di identificare gli artefatti software che potrebbero essere più o meno probabilmente difettosi così da permettere agli sviluppatori di concentrarsi su artefatti specifici.
- Ciò consente di **ridurre i costi** necessari al testing del codice, concentrando le risorse per il testing su alcune classi piuttosto che su altre.

Introduzione

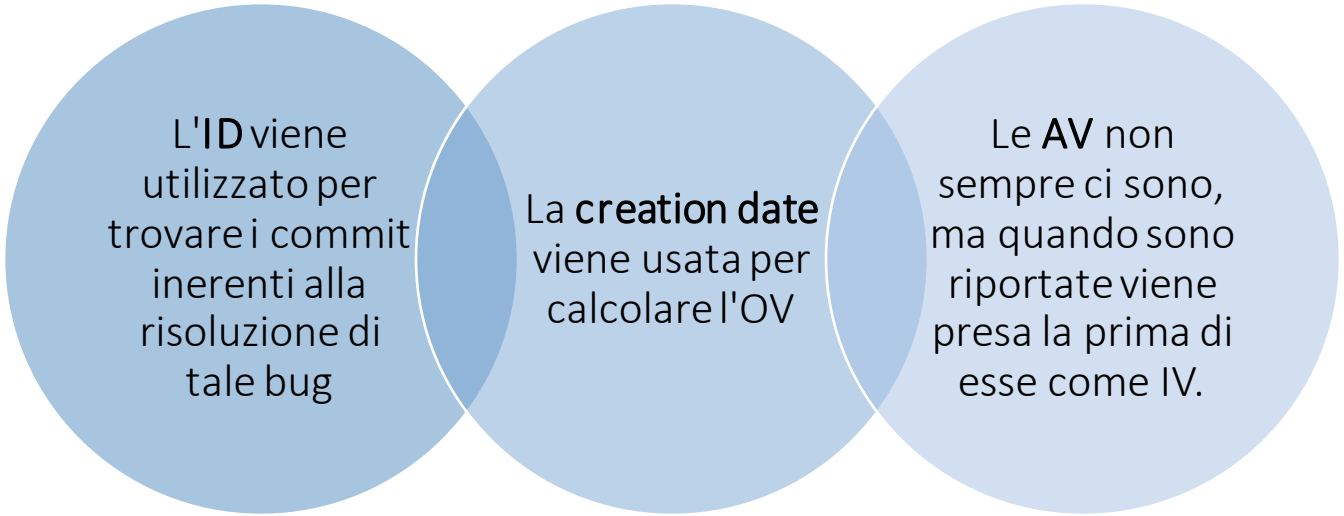
Lo scopo del progetto realizzato è quello di eseguire uno studio empirico finalizzato a misurare l'effetto di tecniche di **sampling**, **feature selection** e **cost evaluation** (offerte dall'API di **Weka**) sull'accuratezza di modelli predittivi della bugginess del codice di due applicazioni open-source: Apache **Avro** e Apache **BookKeeper**.



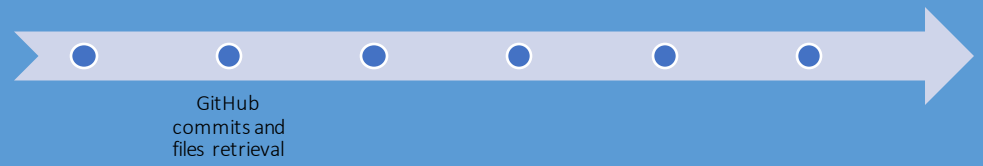
Progettazione

Progettazione

- Raccolta delle release esistenti (query all'URL *https://issues.apache.org/jira/rest/api/2/project/projName.* Essa restituisce un file JSON contenente le informazioni quali ID della release, nome, e data).
- Raccolta, tramite Jira, di tutti i ticket dei progetti.
- Per ogni ticket viene prelevato l'ID, le Affected Versions (se presenti e consistenti) e la creation date.



Progettazione



Clone della repository in locale

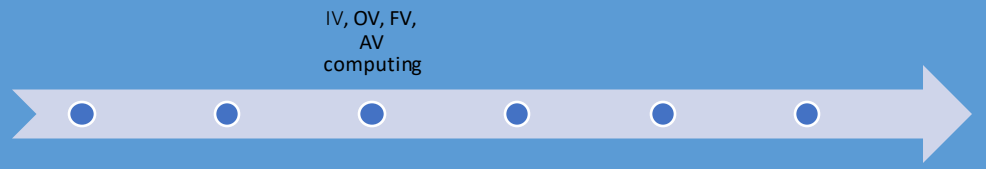
Recupero del log dei commit (API JGit)

Recupero dei file .java dal log, ripercorrendo il tree di ogni commit (struttura contiene tutti i file presenti nel progetto nel momento in cui è stato effettuato il commit)

Ad ogni release viene quindi assegnata una lista di files .java.

L'ultimo 50 % delle release viene inoltre rimosso. Le più recenti si sono dimostrate avere un numero di classi difettose minore rispetto a quelle più datate, in quanto è più probabile che un bug non sia ancora stato scoperto.

Progettazione



Una volta raccolti tutti i commit, si associa ad ogni ticket una lista di commit aventi nel log l'ID del tale ticket

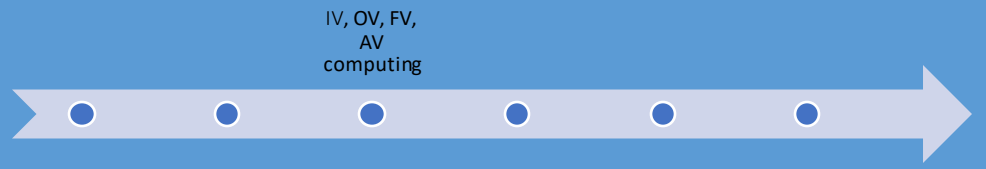
Si setta come FV la release successiva alla data dell'ultimo commit trovato.

Eventuali ticket senza nessun commit associato vengono eliminati.

A questo punto, ogni ticket ha OV (Jira), FV (Git), ed eventualmente AV ed IV (Jira).

Nel caso in cui le AV non siano presenti o consistenti, viene stimato il valore delle IV di ogni ticket tramite il metodo Proportion, per poi calcolare le AV.

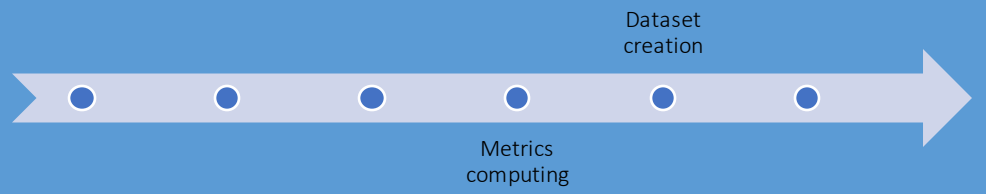
Progettazione



Il metodo di proportion che è stato applicato è **moving window**.

- ∀ Ticket:
- Se l'IV c'è, allora **P = (FV - IV) / (FV - OV).**
- Altrimenti, Predicted IV = FV - (FV - OV) * **P,** dove:
  - **P =** E[last 1% of resolved defects]
- Per i ticket che non hanno IV, P viene impostato inizialmente a 1.
- Spostando poi avanti la finestra man mano, P viene calcolato come la media dei P dell' ultimo 1% di ticket.

Progettazione



Tramite il meccanismo delle **diffEntry** (i cambiamenti che avvengono in un file tra due commit consecutivi) per ogni file vengono calcolate le seguenti metriche:

- LOC
- LOC_ADDED
- MAX_LOC_ADDED
- AVG_LOC_ADDED
- NR
- NFIX
- AUTHORS
- CHURN
- MAX_CHURN

I risultati vengono poi riportati in un file csv.

Progettazione

- Il dataset ottenuto dalla fase precedente viene utilizzato per valutare la prestazione dei classificatori
  - **Random Forest**
  - **Naive Bayes**
  - **Ibk**

  per trovare se e quali tecniche di feature selection, balancing o sensitivity aumentano l'accuratezza di un determinato tipo di classificatore.
- La tecnica di valutazione utilizzata è quella del **Walk Forward**, scelta necessaria per preservare l'ordine dei dati.
- Il dataset è stato diviso in porzioni a seconda del numero di release.

Progettazione

È stata provata ogni possibile combinazione di {feature selection, balancing, sensitivity, classifier}, come si può notare dalla logica racchiusa dal seguente pseudo codice:

```
for ["NO_FEATURE_SELECTION","BEST_FIRST"]
        for ["NO_SAMPLING","OVERSAMPLING","UNDERSAMPLING","SMOTE"]
                for ["NO_COST","SENSITIVE_THRESHOLD","SENSITIVE_LEARNING"]
                        for ["RANDOM_FOREST","NAIVE_BAYES","IBK"]
                                doWalkForward();
reportResults();
```

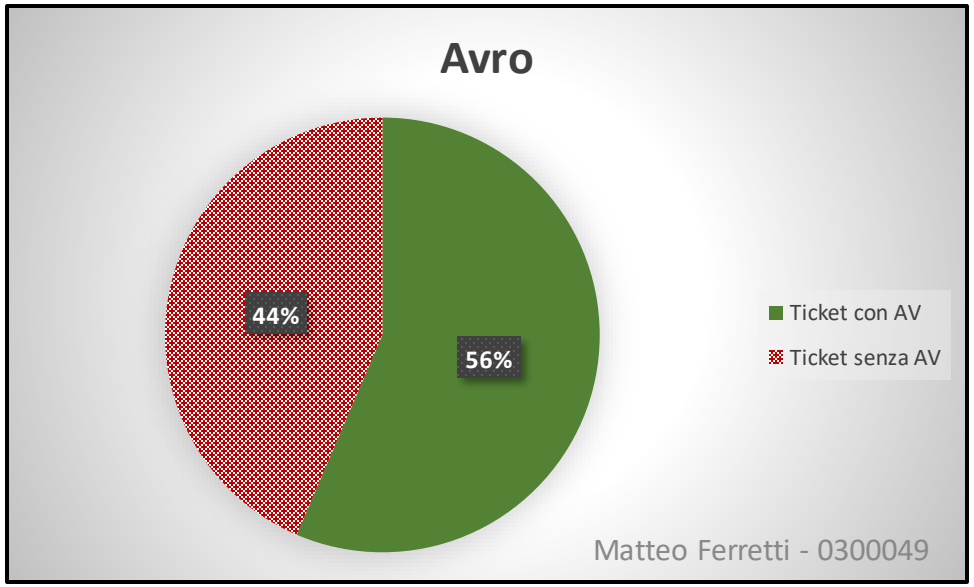
Risultati

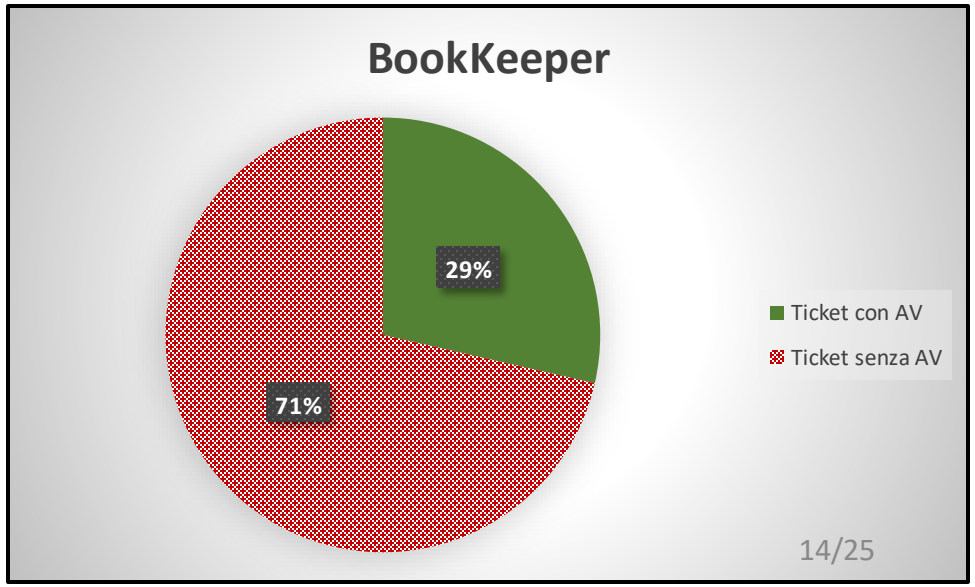

AV a disposizione
Bugginess (in generale)
Bugginess (nel tempo)
Confronto tra classificatori (Bookkeeper)
Confronto tra classificatori (Avro)
Analisi % defective in training

Risultati

La percentuale di **ticket senza affected version**, ovvero che non andranno a contribuire ai calcoli inerenti alla difettosità delle classi è riportata qui.

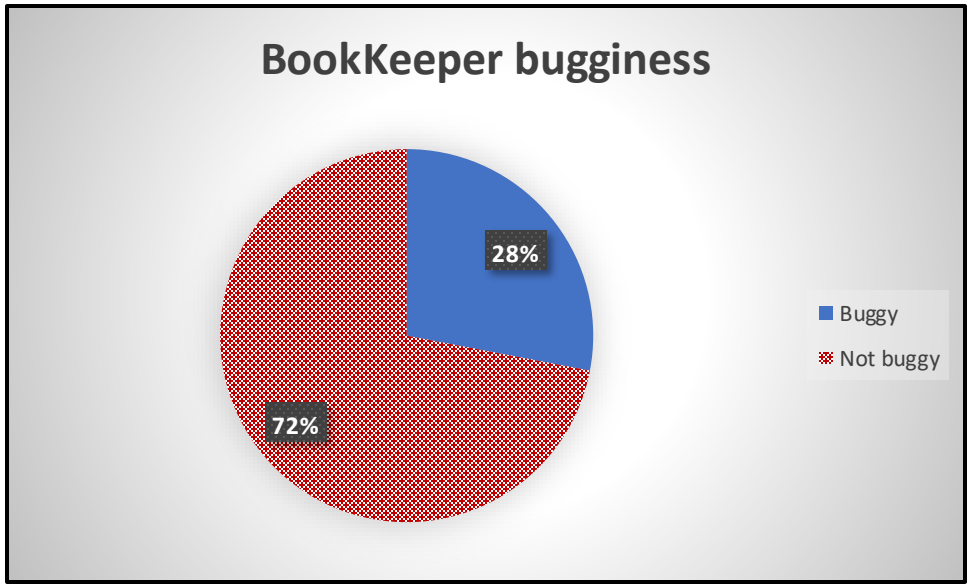
Affected versions				
	BookKeeper	Percentuale	**Avro**	Percentuale
Ticket totali	378	100,00	829	100
Ticket senza AV	270	71,43	361	43,55
Ticket con AV	108	28,57	468	56,45

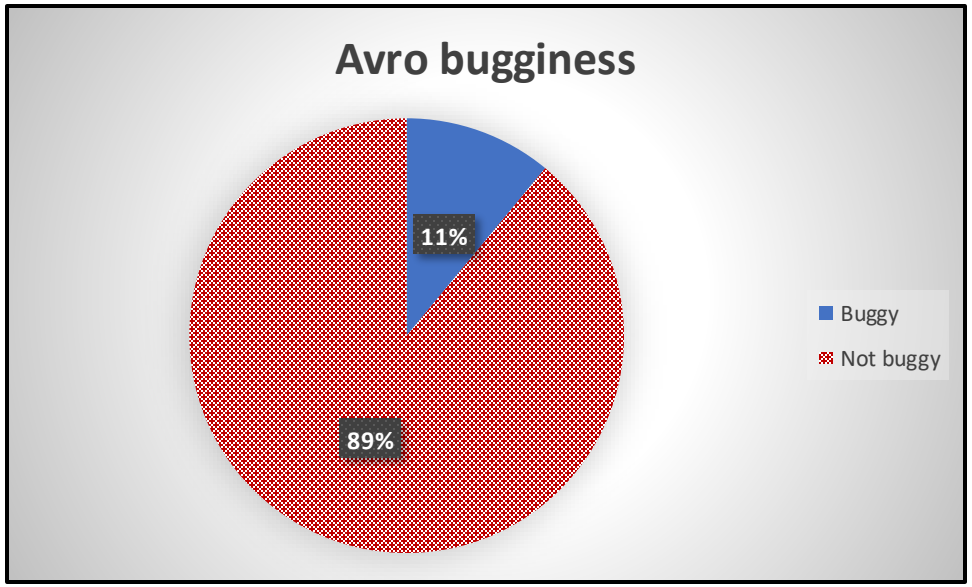




Risultati

La percentuale di classi buggy totale è relativamente bassa in entrambi i progetti dato che risulta molto spesso FV == OV. Nell'applicazione del metodo proportion, nel suddetto caso, la IV viene impostata pari alla FV (così da non impostare alcuna AV successivamente). Ciò comporta una diminuzione sostanziale di classi difettose.

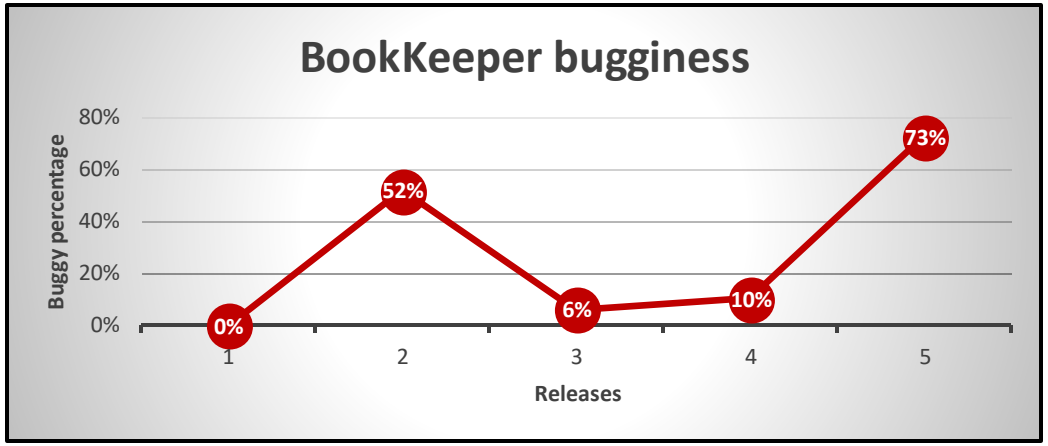


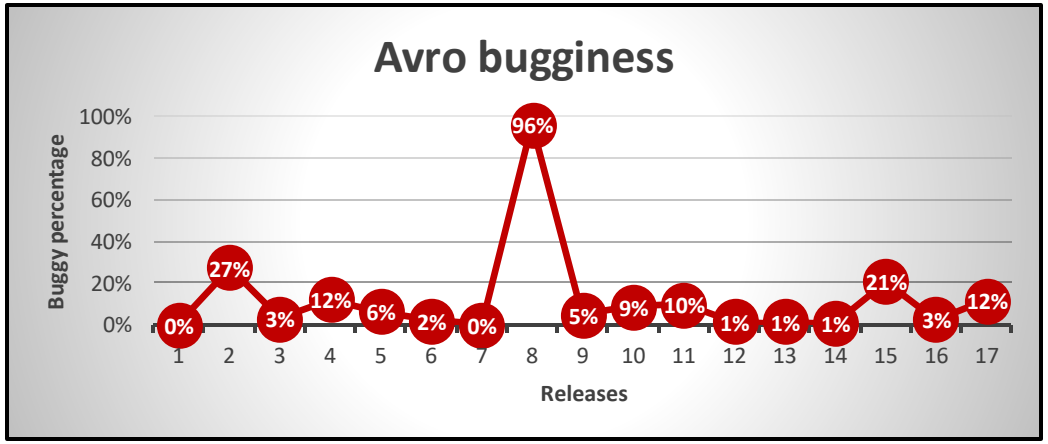


Risultati

La bugginess tende a restare bassa in quanto:

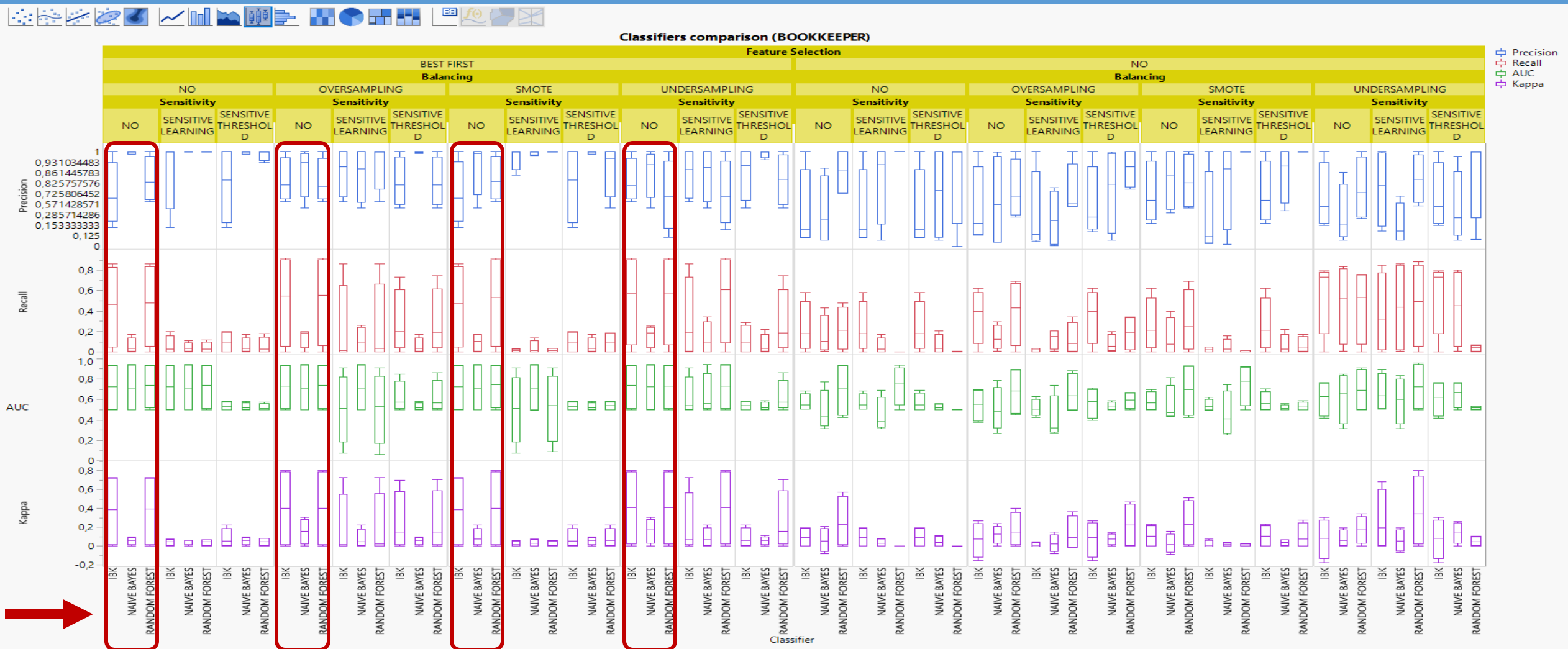
- Per BookKeeper, solo il 30% dei ticket iniziali da un contributo alla bugginess. Per Avro, circa la metà. Questo per via delle AV, o addirittura delle FV.
- Talvolta le classi cambiano nome o path da una versione all'altra, e questo fa in maniera tale che sfuggano al controllo sui modified o deleted, non risultando buggy quando in realtà potrebbero esserlo.
- Esistono versioni con pochissimi commit, il che riduce i dati a disposizione su cui calcolare la buggyness.





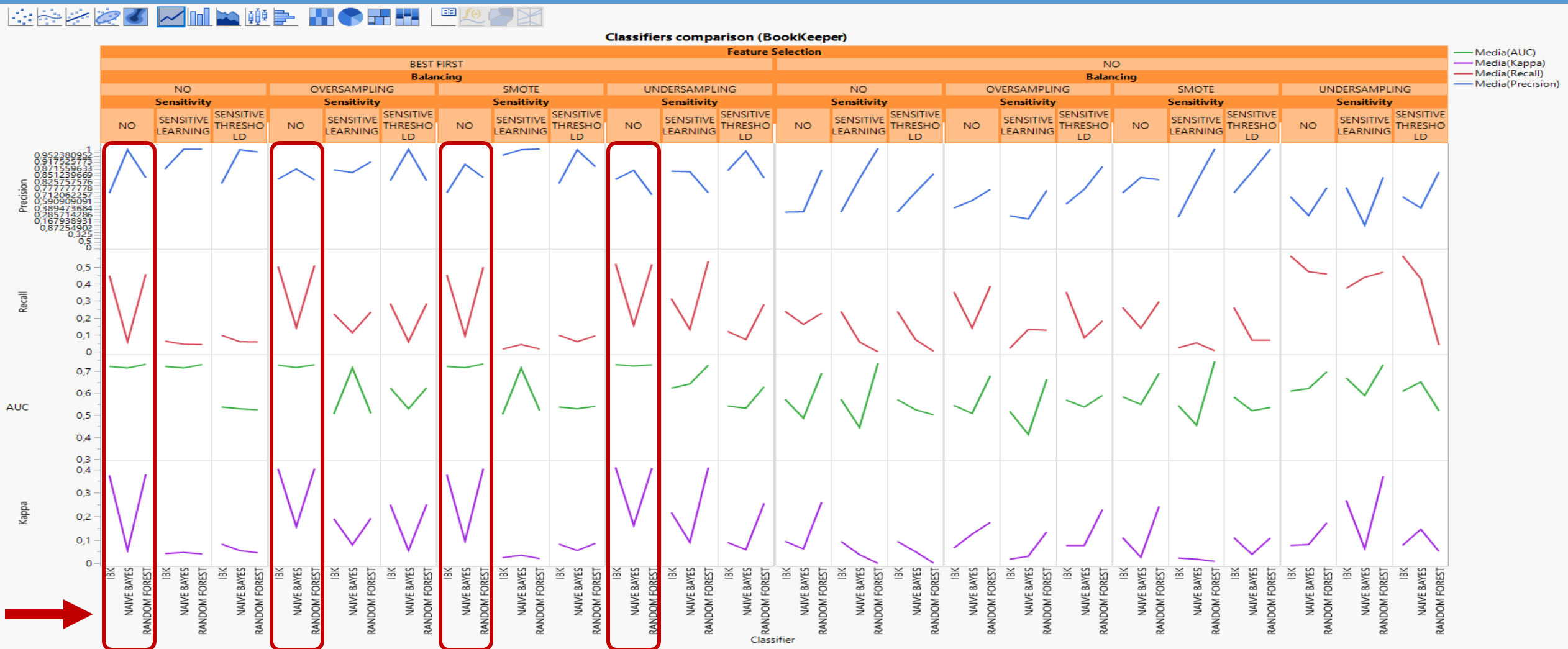
Risultati


Confronto tra i classificatori (Bookkeeper)


Classifiers comparison (BOOKKEEPER)
Feature Selection
BEST FIRST
NO
Balancing
NO
OVERSAMPLING
SMOTE
UNDERSAMPLING
Sensitivity
NO
SENSITIVE LEARNING
SENSITIVE THRESHOLD
Precision
Recall
AUC
Kappa
IBK
NAIVE BAYES
RANDOM FOREST
Classifier

Risultati


Confronto tra i classificatori (Bookkeeper)


Classifiers comparison (BookKeeper)
Feature Selection
BEST FIRST
NO
Balancing
NO
OVERSAMPLING
SMOTE
UNDERSAMPLING
Sensitivity
NO
SENSITIVE LEARNING
SENSITIVE THRESHOLD
Media(AUC)
Media(Kappa)
Media(Recall)
Media(Precision)
Precision
Recall
AUC
Kappa
IBK
NAIVE BAYES
RANDOM FOREST
Classifier

Risultati

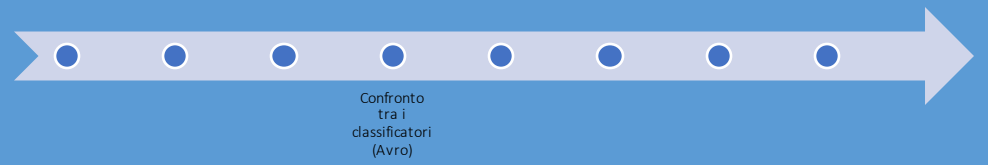
- Per BookKeeper, **applicare feature selection** tende ad alzare i valori di tutte le metriche.
- Risulta **svantaggioso applicare sensitivity,** in quanto tende ad alzare pochissimo la precision abbassando poco Kappa, sensibilmente l'AUC e moltissimo la recall.
- Il balancing non fornisce alcun contributo degno di nota, quindi sconveniente da applicare dato il costo.

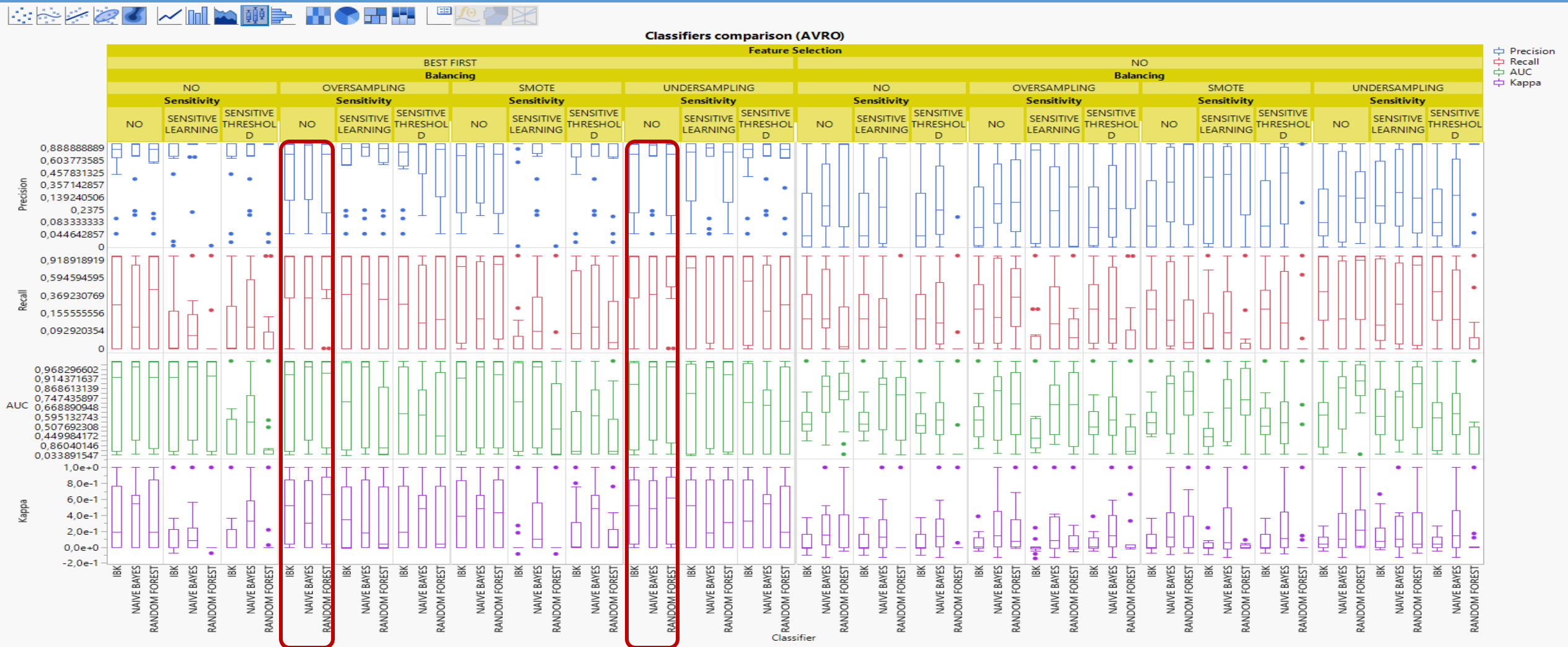
Tra i classificatori, **Random Forest e IBK** risultano essere i migliori e pressochè equivalenti.

Con feature selection e no sensitivity Random forest risulta più accurato nel caso di SMOTE o assenza di balancing, mentre IBK nel caso di undersampling. Nel caso di oversampling risultano identici.

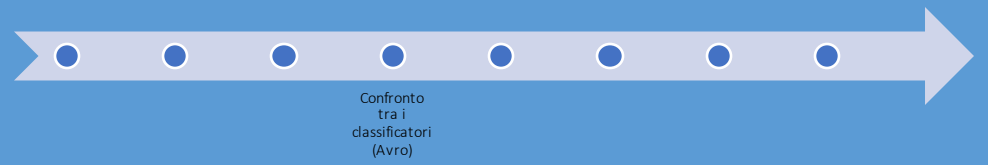
Il miglior classificatore è Random Forest con applicata solo la feature selection.

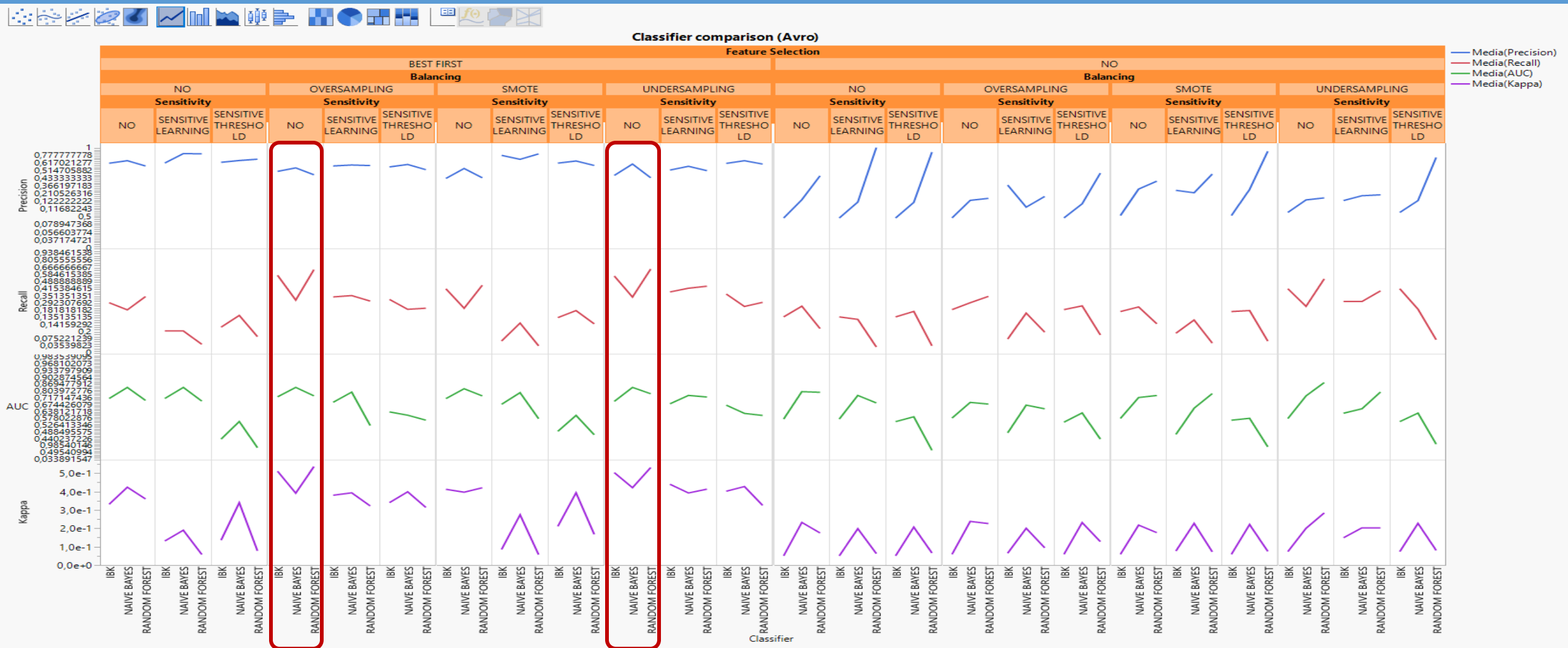
Risultati


Confronto
tra i
classificatori
(Avro)


Classifiers comparison (AVRO)
Feature Selection
BEST FIRST
NO
Balancing
NO
OVERSAMPLING
SMOTE
UNDERSAMPLING
Sensitivity
NO
SENSITIVE LEARNING
SENSITIVE THRESHOLD
Precision
Recall
AUC
Kappa
IBK
NAIVE BAYES
RANDOM FOREST
Classifier

Risultati


Confronto
tra i
classificatori
(Avro)


Classifier comparison (Avro)
Feature Selection
BEST FIRST
NO
Balancing
NO
OVERSAMPLING
SMOTE
UNDERSAMPLING
Sensitivity
NO
SENSITIVE LEARNING
SENSITIVE THRESHOLD
Precision
Recall
AUC
Kappa
IBK
NAIVE BAYES
RANDOM FOREST
Classifier
Media(Precision)
Media(Recall)
Media(AUC)
Media(Kappa)

Risultati

- Per Avro, **applicare feature selection** permette di alzare i valori di Kappa in maniera sostanziosa, il che migliora il comportamento di tutti i classificatori. Inoltre tende ad alzare la precision.
- Risulta **svantaggioso applicare sensitivity,** in quanto tende ad abbassare recall ed AUC.
- Il balancing di tipo **oversampling ed undersampling** permette di ottenere risultati sensibilmente migliori di recall a discapito di poca precision.

Tra i classificatori, **Random Forest e IBK** risultano i migliori; in generale il primo mantiene recall più alta ed accuracy più bassa del secondo, ma un'AUC maggiore.

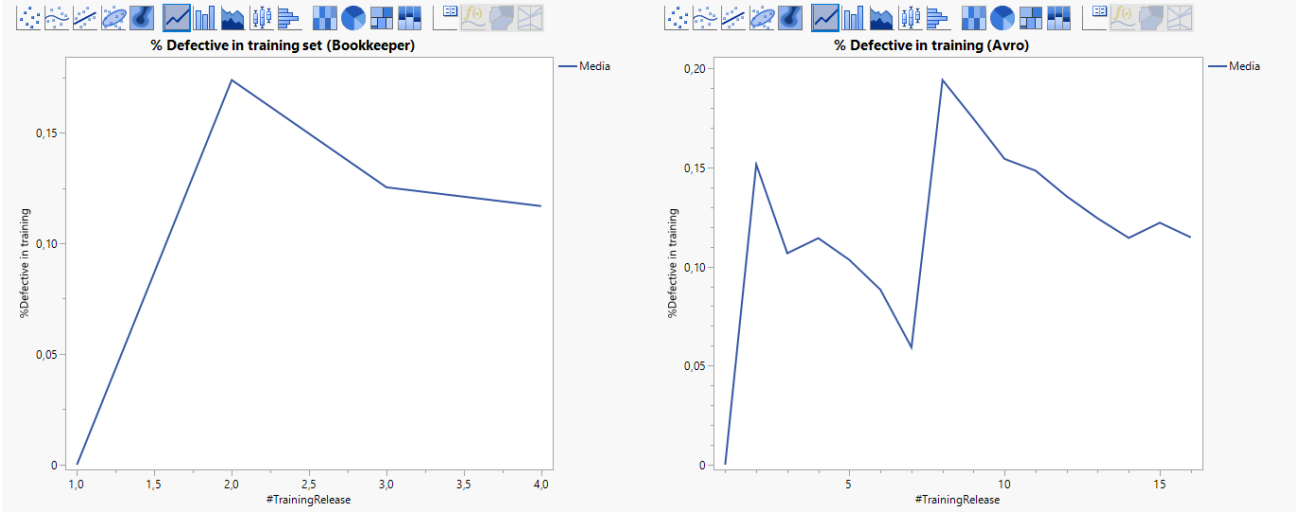
La miglior configurazione è quindi data da IBK/Random forest con feature selection, no sensitivity, oversampling/undersampling.

Risultati

Durante il Walk Forward, il numero di defective in training si comporta in maniera simile nei due progetti:

- Nel caso di Bookkeeper la tendenza è l'aumentare dei defective nei primi passi, per poi diminuire.
- Nel caso di Avro il comportamento è simile e tende anche a ripetersi.

Entrambi i progetti non superano mai una percentuale di training defective superiore al 20%.



Conclusione

In generale, in entrambi i progetti, **utilizzare feature selection** e **non utilizzare sensitivity** ha mostrato prestazioni migliori per tutti i classificatori.

- Utilizzare Sensitive Threshold e Sensitive Learning tende ad abbassare notevolmente i valori delle metriche per via del già notevole class imbalance del dataset iniziale.
- Utilizzare Feature selection permette sempre di ottenere un dataset su cui i classificatori ottengono prestazioni migliori.

Per BookKeeper non conviene applicare balancing, mentre per Avro conviene applicare oversampling ed undersampling.

Per entrambi il classificatore migliore sembra essere Random Forest con le suddette configurazioni.

Links

SonarCloud: https://sonarcloud.io/project/overview?id=IronMatt97_ISW2-2122

GitHub: https://github.com/IronMatt97/ISW2-2122