Machine Learning per Software Engineering

Studio sulle performance di classificatori di Machine Learning applicati al Software Engineering per agevolare la fase di testing and bug prevention.

Matteo Ciccaglione 0315944

Indice

- Introduzione e motivazioni dello studio;
- Progettazione e raccolta dati;
- Analisi dei risultati;
- Conclusioni

Introduzione

- L’attività di testing è una fase fondamentale nel percorso di sviluppo di un progetto. L’individuazione di possibili bug in fase di testing permette di evitare failure del software in fase di produzione. Tuttavia richiede un effort elevato, ed un alto costo di risorse, per progetti di grandi dimensioni.

- L’applicazione del Machine Learning al Software Engineering fornisce strumenti per agevolare il lavoro del tester, come?

Introduzione

- Il Machine Learning applicato al SE ha l’obiettivo di fornire strumenti per l’individuazione delle classi con maggiore probabilità di essere “buggy” in futuro. L’idea è di utilizzare la storia del progetto (i dati del passato), o di progetti affini, per costruire dataset di training e testing da fornire ad un modello di Machine Learning.

- Il modello sarà poi in grado di effettuare una predizione sulle classi del progetto che ritiene possano nascondere dei bug.

Introduzione

- Il lavoro del tester viene così agevolato, in quanto può concentrare l’effort e le risorse a disposizione sulle classi individuate dal modello di ML.

- Questo studio si pone dunque l’obiettivo di analizzare diversi classificatori di Machine Learning, applicando ad ognuno molteplici tecniche, per individuare con quali tecniche e per quale classificatore si ottiene il risultato migliore, misurato in termini di Precision, Recall, AUC e Kappa.

Progettazione e raccolta dati

- Lo studio è stato sviluppato in riferimento a due progetti Apache ovvero:
  - BookKeeper;
  - Zookeeper;

- Per la costruzione dei dataset di training e testing, si è resa necessaria la raccolta di dati sulla storia passata di questi due progetti. Di che dati stiamo parlando?

- Con dati sulla storia passata ci riferiamo a tutte le informazioni che possono essere raccolte dai commit effettuati sul software di version control, in riferimento ad issue riportati sul software di issue tracking.

Progettazione e raccolta dati

- Per i progetti in analisi il software di VC è Git mentre il software di Issue tracking è Jira.

- Per effettuare il mining dei dati tramite questi due sistemi sopra riportati, è stato sviluppato del codice Java che:
  1)Interagisce con Jira per raccogliere tutti gli Issue di tipo Bug, con risoluzione Fixed e con stato Resolved oppure Closed;
  2)Per ogni Issue, interagisce con Git per individuare i commit che ne fanno riferimento (ovvero quei commit che riportano nel messaggio l'id dell'issue);
  3)Per ogni commit sono state raccolte ed aggregate (release per release) metriche da fornire al modello di ML.

Progettazione e raccolta dati

Metriche utilizzate nel progetto:

locTouched	Linee di codice toccate dal commit
locAdded	Linee di codice aggiunte dal commit
maxLocAdded	Il massimo valore di locAdded in quella release per quella classe
avgLocAdded	Il valor medio di locAdded in quella release per quella classe (calcolato rispetto al numero di revisioni)
churn	Somma tra il numero di righe di codice aggiunte e rimosse
maxChurn	Il massimo churn in quella release per quella classe
avgChurn	Il valor medio di churn in quella release per quella classe (calcolato rispetto al numero di revisioni)
nFix	Il numero di bug fixati che riguardavano quella classe in quella release
nAuthors	Il numero di autori che hanno toccato quella classe in quella release
nR	Il numero di revisioni che quella classe ha ricevuto in quella release

Progettazione e raccolta dati

- Una classe può essere etichettata come “buggy” in una determinata release se esiste un commit relativo ad un Issue in cui la release ne costituisce l’affected version.
  Per l’individuazione delle affected version si è fatto riferimento a quelle riportata dai developers su Jira (qualora disponibili e consistenti) o sono state calcolate applicando proportion incremental.

- Poiché il dataset non deve contenere unicamente le classi etichettate come buggy, sono stati raccolti ed analizzati anche i commit che non fanno riferimento ad alcun Issue.
  Una classe in una release potrebbe essere affetta da snoring, ovvero potrebbe presentare dei bug “dormienti”, che saranno individuati soltanto in release successive. Per limitare il problema dello snoring sono stati presi in considerazione tutti i bug riportati su Jira, ma sono state raccolte le metriche unicamente sulla prima metà delle release;

Progettazione e raccolta dati

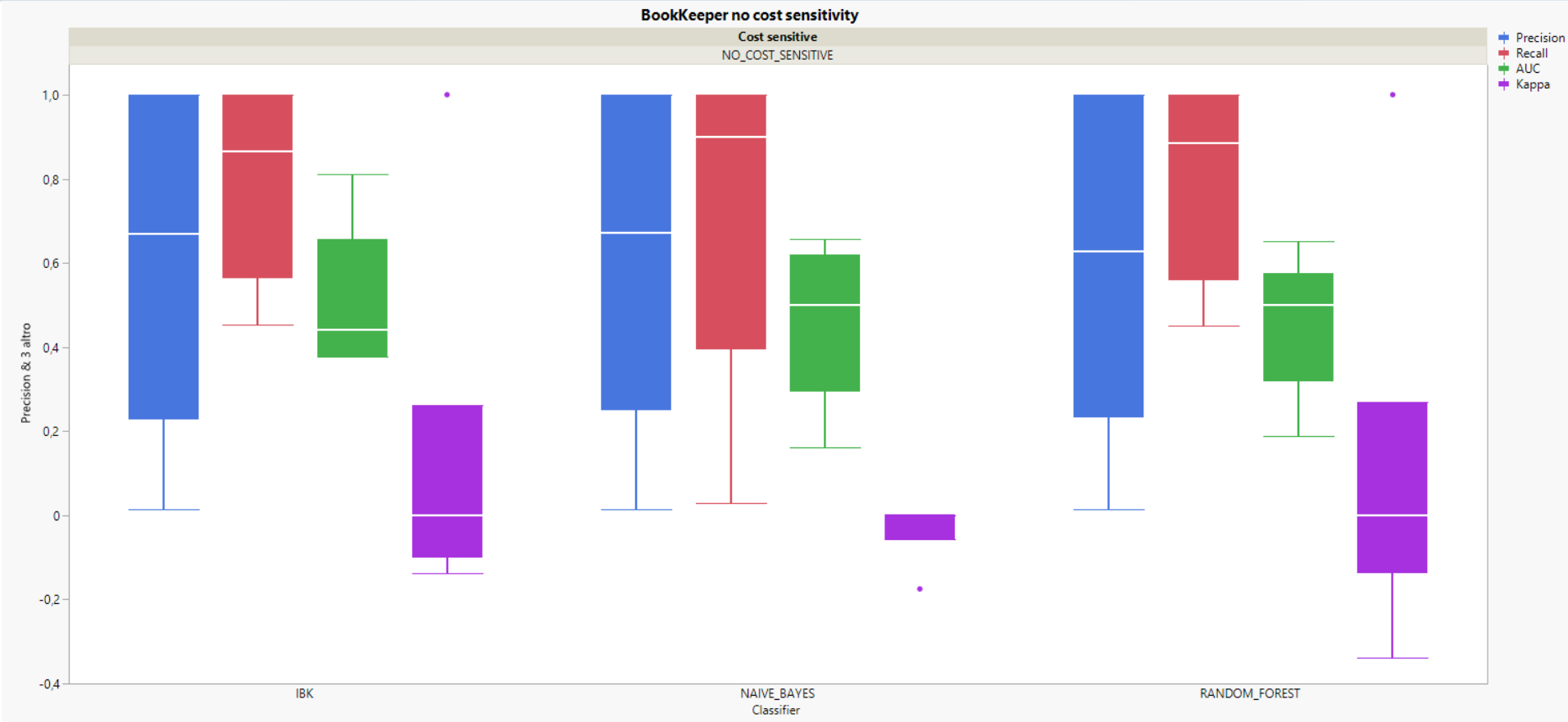
- Lo studio si preoccupa di analizzare la “bontà” di un classificatore, misurata attraverso metriche come AUC, Kappa, Recall e Precision, al variare delle tecniche applicate. Come tecnica di validazione è stata applicata WalkForward.

- I classificatori presi in considerazione sono NaiveBayes, RandomForest ed Ibk.
- Si è deciso di applicare feature selection con backward search per ridurre il numero di feature da utilizzare, undersampling per bilanciare il dataset che presenta un numero di istanze non buggy molto maggiore di quelle buggy, e tecniche di cost sensitivity.
- Per non complicare eccessivamente lo studio si è scelto di mantenere fisse le tecniche di feature selection ed undersampling, e di studiare il variare delle prestazioni (per ogni classificatore) al variare delle tecniche di cost sensitivity.
  A tal proposito si è deciso di variare tra questi possibili valori: Sensitive Threshold, Sensitive Learning e No cost sensitive.

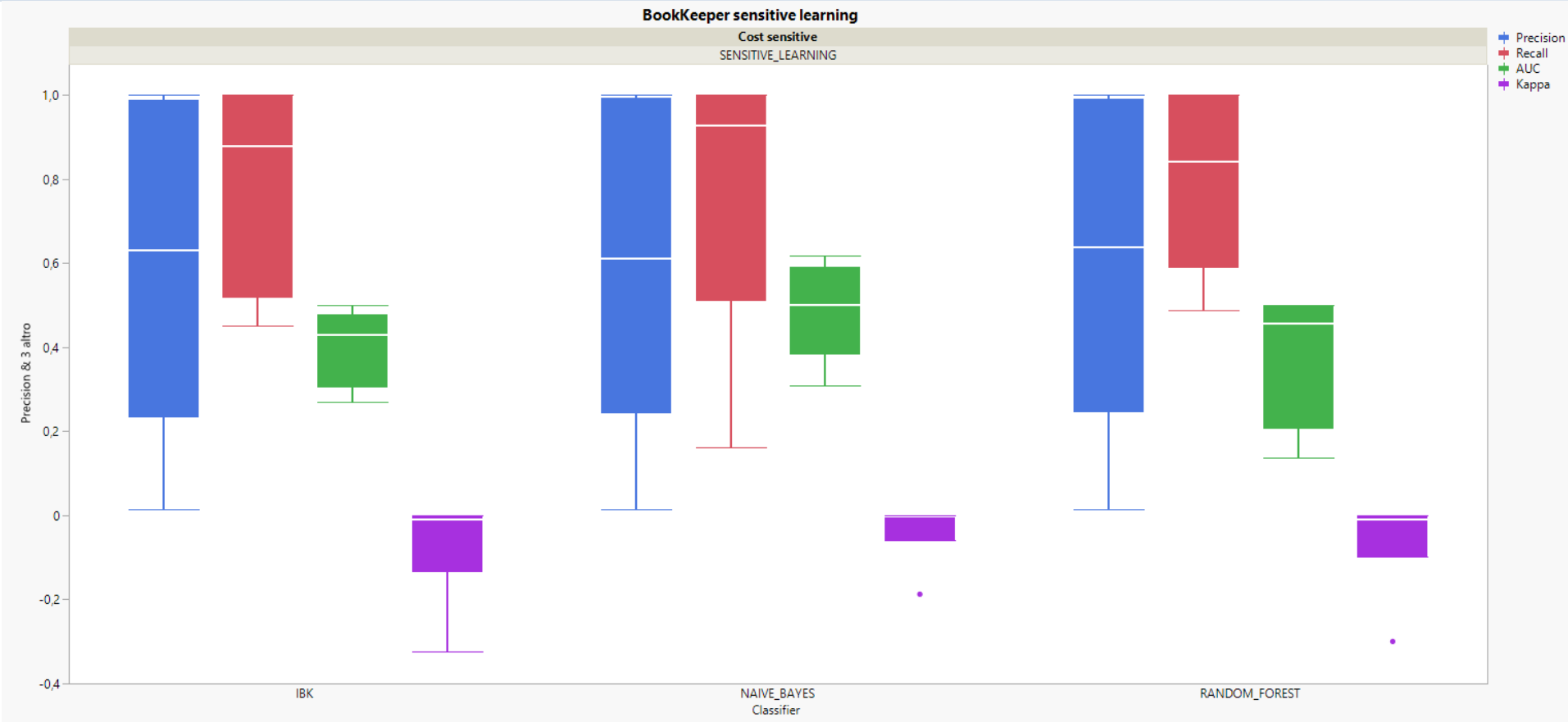
Analisi dei risultati: BookKeeper

- Il progetto Apache BookKeeper ha abbandonato l'utilizzo di Jira come software di Issue tracking, pertanto i dati raccolti fanno riferimento fino all'anno 2017.
- Per questo progetto sono state eseguite 6 run di WalkForward.
- Di seguito sono riportati dei box-plot che rappresentano, per ogni classificatore, la distribuzione per le metriche di Recall, AUC, Kappa e Precision.
- Ogni box-plot è relativo ad una differente tecnica di Cost Sensitivity applicata.
- Al termine dei grafici sono riportate delle considerazioni sui risultati dello studio.

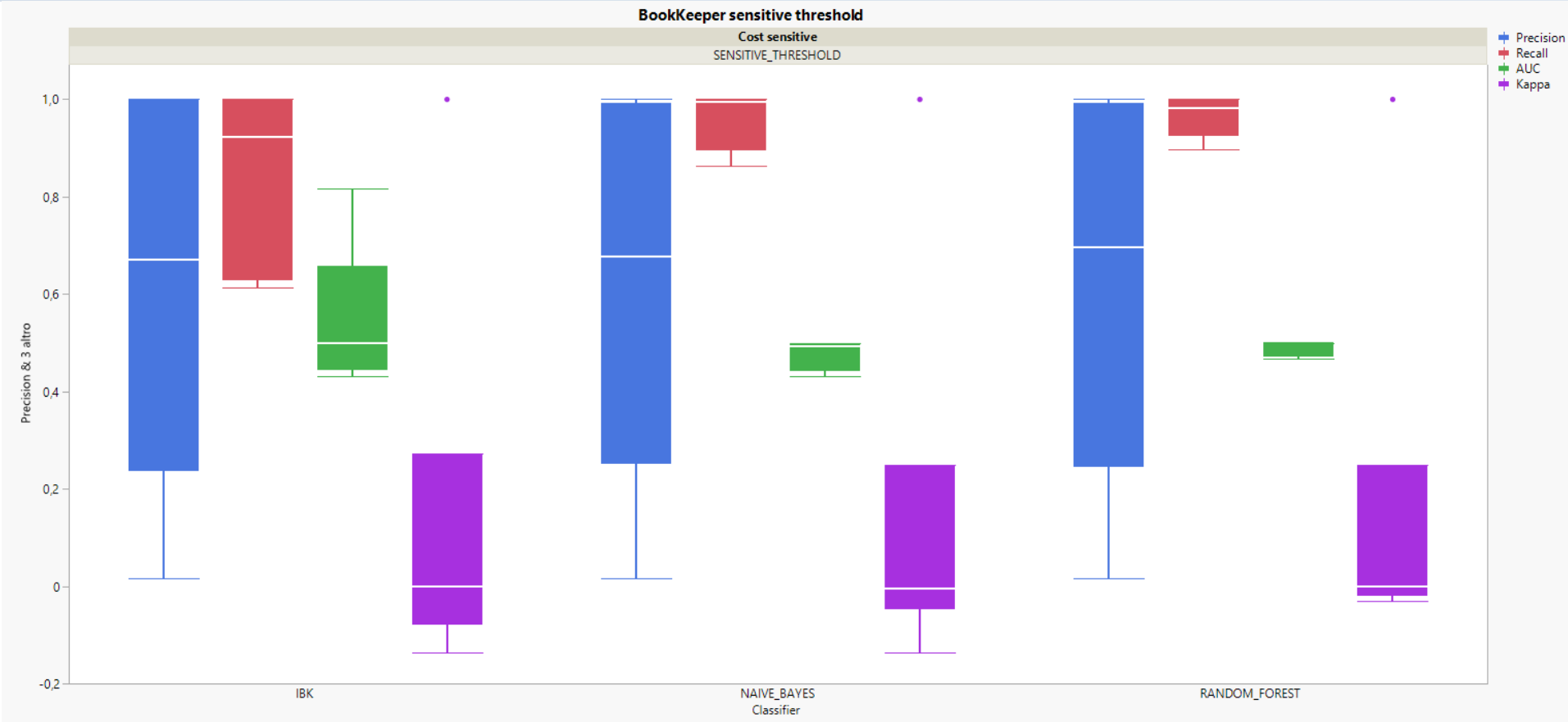
Analisi dei risultati: BookKeeper



Analisi dei risultati: BookKeeper



Analisi dei risultati: BookKeeper

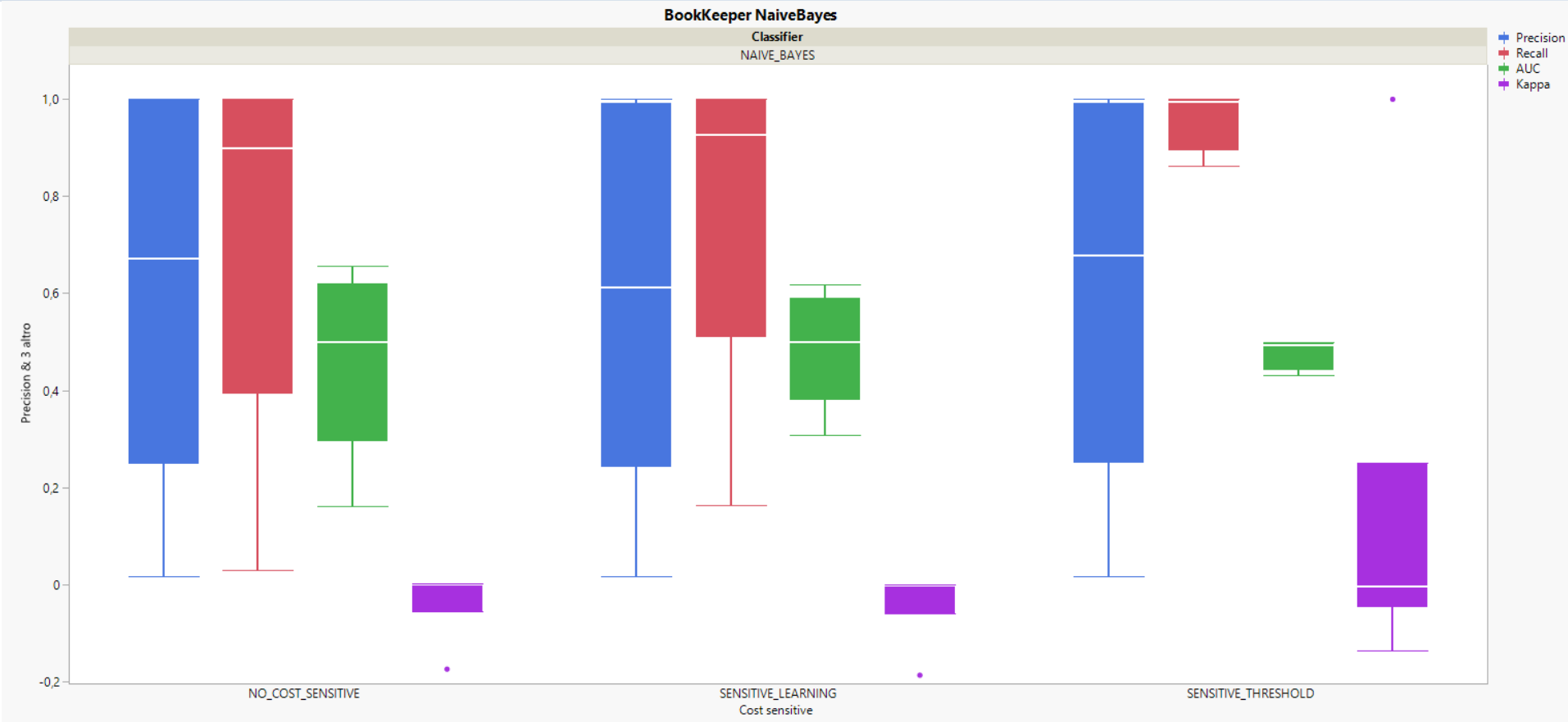


Analisi dei risultati: BookKeeper

- Dall’analisi effettuata si può notare come NaiveBayes sia il miglior classificatore da utilizzare in applicazione a questo progetto. Presenta in tutte e tre le configurazioni una precision ed una recall più alta ed una AUC con valor medio sullo 0.5;

- La migliore configurazione è con Cost Sensitive Threshold, in quanto presenta valor medio per Recall più alto ed ha una distribuzione per la recall più vicina al valor medio (bassa varianza). La precision ha un valor medio sullo 0.6. Alta recall e precision media è un buon compromesso per un classificatore che deve fornire una predizione corretta sulla difettosità di una classe. Infatti alta recall indica che quasi il 100% delle istanze buggy sono stati individuati correttamente dal classificatore, ed una precision sullo 0.6 indica che solo il 40% delle volte in cui il classificatore dice “positive” commette un errore. Inoltre anche il valore dell’AUC medio è sullo 0.5 con distribuzione molto vicina al valor medio. Questa configurazione ha anche una tra le migliori distribuzioni di Kappa (nel confronto). Di seguito un grafico che confronta, per NaiveBayes, le varie configurazioni.

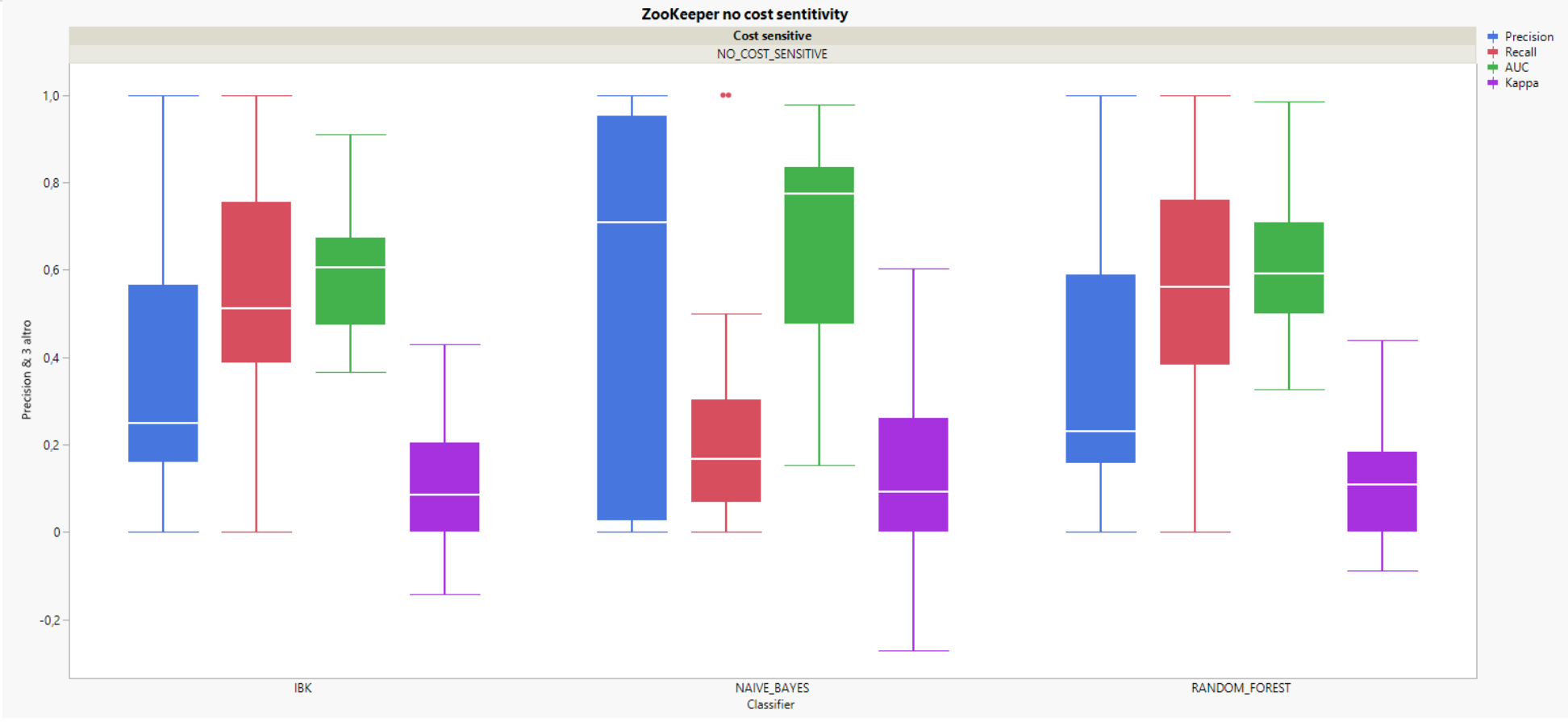
Analisi dei risultati: BookKeeper



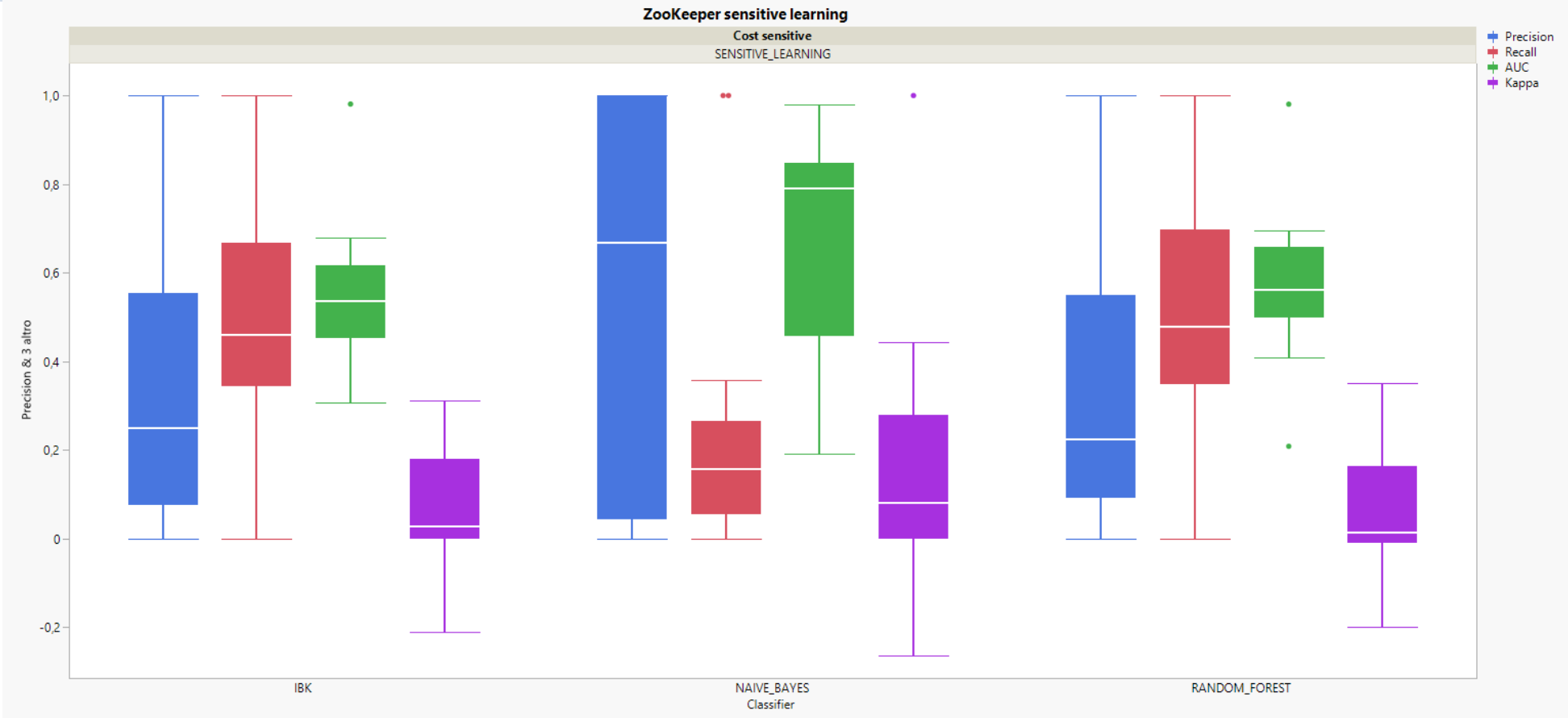
Analisi dei risultati: ZooKeeper

- Il progetto Apache ZooKeeper a differenza del precedente continua ad utilizzare Jira come software di Issue tracking, di conseguenza è stato possibile raccogliere un numero maggiore di dati, ed ottenere risultati più accurati.
- Per questo progetto sono state eseguite 22 run di WalkForward.
- Di seguito sono riportati dei box-plot che rappresentano, per ogni classificatore, la distribuzione per le metriche di Recall, AUC, Kappa e Precision.
- Ogni box-plot è relativo ad una differente tecnica di Cost Sensitivity applicata.
- Al termine dei grafici sono riportate delle considerazioni sui risultati dello studio.

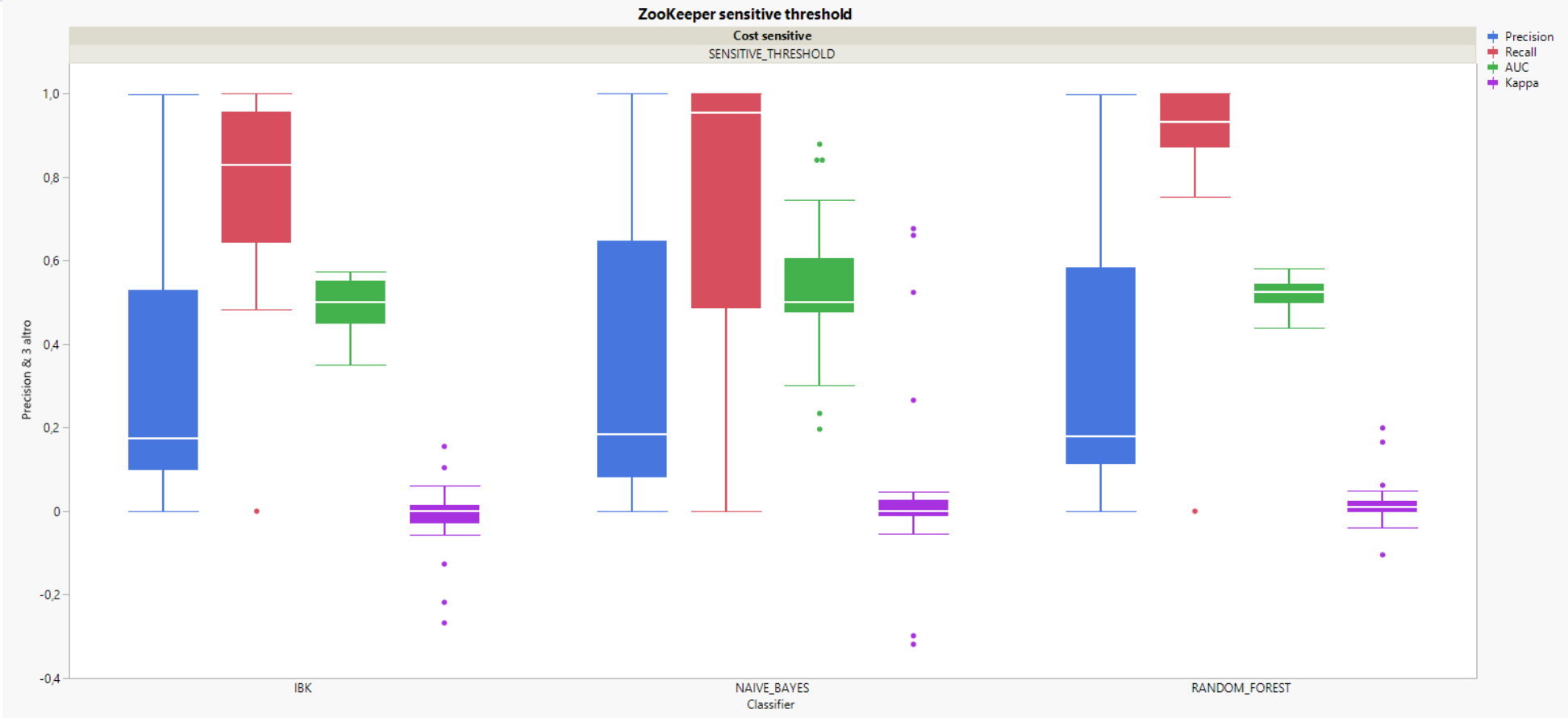
Analisi dei risultati: ZooKeeper



Analisi dei risultati: ZooKeeper



Analisi dei risultati: ZooKeeper



Analisi dei risultati: ZooKeeper

- I risultati ottenuti per ZooKeeper danno spazio ad un'ampia analisi. Consideriamo il comportamento dei classificatori Ibk e RandomForest rispetto alle diverse applicazioni di cost sensitivity, concentrandoci su Precision e Recall:
  1) Senza cost sensitivity presentano risultati molto simili, con una Recall con valor medio sullo 0.5-0.6 e una Precision con valor medio sullo 0.2 e distribuzione "bassa", con la maggior parte dei valori che arriva ad un massimo di 0.6;
  2) Con sensitive learning le prestazioni peggiorano di poco rispetto al caso precedente;
  3) Con sensitive threshold si osserva un incremento della Recall (che raggiunge un valor medio di 0.8 per Ibk e 0.9 per RandomForest), mentre la Precision rimane con una distribuzione simile;

- Per i risultati 1 e 2, siamo nella situazione dove il classificatore riesce ad individuare in media il 60% dei positivi, ma nel fare il labeling commette un errore l'80% delle volte, quindi l'output è inaffidabile (per il nostro caso di studio). Per il risultato 3 invece si ha che il classificatore riesce ad individuare l'80%-90% dei positivi, ma ancora con un tasso di errore elevato.

Analisi dei risultati: ZooKeeper

- Consideriamo ora i risultati per il classificatore NaiveBayes:
  1) Nello scenario no cost sensitivity presenta una precision molto alta con valor medio sullo 0.7 ma distribuzione ampia (alta varianza) e una recall decisamente bassa con valor medio sullo 0.2 e distribuzione vicina al valor medio;
  2) Nello scenario sensitive learning non si osservano miglioramenti;
  3) Nello scenario sensitive threshold si ottiene una recall decisamente alta con valor medio vicino ad 1 ma distribuzione ampia. La precision diminuisce con valor medio 0.2 , mentre AUC e Kappa diminuiscono rispetto ai casi precedenti.

- I risultati 1 e 2 indicano che, nell'effettuare il labeling il classificatore commette un errore solo il 30% delle volte, ma riesce ad individuare solo il 20% dei positivi. Quindi il classificatore è affidabile, ma è titubante nel classificare come positivo, al punto da individuarne solo 2 su 10. Per il nostro scenario applicativo, in cui vogliamo predire la difettosità di una classe, un risultato del genere esclude 8 classi buggy su 10. Il risultato 3 invece indica che il classificatore riesce ad individuare circa il 100% dei positivi, ma commette un errore l'80% delle volte in cui classifica come positivo.

Analisi dei risultati: ZooKeeper

- Alla luce di quanto detto tiriamo le conclusioni su questo studio. Non esiste un classificatore migliore in assoluto, nello scenario in cui si preferisce avere un'alta precisione (quindi pochi falsi positivi e poco effort aggiuntivo per i tester) il classificatore migliore è NaiveBayes con sensitive learning, mentre nel caso in cui si voglia individuare la quasi totalità dei positivi, perdendone in precisione, il classificatore migliore è RandomForest con sensitive threshold;

- Infatti dovendo scegliere tra NaiveBayes sensitive threshold e RandomForest sensitive threshold, il caso migliore è proprio il secondo in quanto ha un valor medio della Recall di poco più basso, ma una distribuzione più vicina al valor medio (bassa varianza) e una AUC e Kappa migliore;

- Non esiste in questo caso la soluzione migliore.

Conclusioni

- Dall’analisi dei risultati è emerso come il classificatore NaiveBayes si comporti bene per entrambi i progetti, risultando anche il classificatore migliore per BookKeeper;

- L’applicazione della tecnica di sensitive threshold porta ad un guadagno alto in Recall, che per l’applicazione del ML al SE è un buon risultato. Avere Recall alta significa riuscire ad individuare la quasi totalità delle classi potenzialmente buggy, e prevenire così problemi futuri;

- I risultati hanno mostrato come un incremento in Recall può portare ad una perdita di Precision. Questo risultato era prevedibile, se la Recall è alta allora riesco ad individuare un alto numero di positivi, ma per farlo devo classificare più istanze come positive e finisco per commettere un errore più alto abbassando la Precision. Il risultato ideale sarebbe avere una Precision ed una Recall alta, ma lo studio ha mostrato come questo risultato sia non raggiungibile.

Links

- GitHub

  https://github.com/matteociccaglione/GitLogTest

- SonarCloud

  https://sonarcloud.io/summary/overall?id=matteociccaglione_GitLogTest