

The background of the slide features a complex network diagram. It consists of numerous nodes of varying sizes, some colored in dark blue, light blue, and grey, connected by a web of thin, light grey lines. Some nodes are highlighted with larger, concentric circles. The overall aesthetic is technical and modern.

ANALISI DELLE PRESTAZIONI DI CLASSIFICATORI NELLA PREDIZIONE DELLA DIFETTOSITÀ DEL CODICE

Diana Pasquali – 0306320 – A.A 2021/2022

OUTLINE

- Introduzione
- Progettazione
- Risultati
- Discussione dei risultati
- Conclusioni

INTRODUZIONE

Obiettivo: illustrare l'andamento delle prestazioni di tre classificatori nel predire la difettosità di classi Java, al variare al variare delle tecniche di Balancing applicate.

Obiettivo 2: illustrare l'andamento delle prestazioni dei tre classificatori nel predire la difettosità di classi Java, applicando Feature Selection e valutare la rilevanza delle feature rispetto al modello in esame.

- I classificatori sono addestrati su un dataset che rappresenta la difettosità delle classi Java nei due progetti Apache/BookKeeper e Apache/Syncope.
- I classificatori considerati sono IBk, NaiveBayes e RandomForest.
- Le tecniche di Balancing considerate sono Oversampling, Undersampling e SMOTE.
- La tecnica di Feature Selection valutata è BestFirst (Filtro) con Backward Search.

PROGETTAZIONE

- Prima di procedere all'addestramento dei classificatori, è necessario costruire il dataset da utilizzare.
- Una volta costruito il dataset, sono stati rimossi i dati relativi all'ultimo 50% delle release in modo da ridurre l'effetto di Snoring.
- Il dataset costituito da differenti colonne che specificano:
 - Una **versione**
 - Un **file** java
 - 16 **metriche** (correlate alla presenza o meno di bug nel file e nella versione considerati)
 - Un'**etichetta** "Yes/No" per specificare se il file è risultato o meno difettoso all'interno della versione.

PROGETTAZIONE - METRICHE

➤ Le principali metriche considerate sono le seguenti:

Size: lo storico dimensione del file in termini di LOC (cross-release).

- Un valore alto di linee di codice, provocano un rischio maggiore di avere bug.

LOC_Touched: la somma delle LOC aggiunte e rimosse nell'arco delle revisioni sulla singola versione (intra-release).

- Un numero alto indica che sono cambiate molte LOC del file nella versione e allora c'è più possibilità di generare bugs.

NFix: il numero di revisioni al file che risolvono dei bug issues di Jira nella singola versione (intra-release).

- Un numero alto di commit che cercano di risolvere issues, potrebbe significare una difficile risoluzione dei problemi nel codice oppure una propensione del file ad essere difettoso.

PROGETTAZIONE - METRICHE

NR: il numero di revisioni al file nella singola versione (intra-release).

- Maggiori sono i commit al file, e maggiori sono le possibilità di introdurre bug.

NAuth: il numero di autori totali che hanno collaborato al file nella versione (intra-release).

- Con un valore elevato, aumentano le probabilità di generare difetti nel codice a causa di incomprensioni e delle differenti intenzioni tra i collaboratori.

LOC_Added: il numero di LOC aggiunte (intra-release).

- Simile alla Size. Più LOC sono aggiunte al file e maggiore è la possibilità di avere introdotto bug.

Churn: è la somma sulle revisioni di LOC added – deleted per la singola versione (intra-release).

- Un valore elevato rappresenta il numero di linee di LOC effettivo che potrebbe contribuire ad introdurre un bug nella classe.

PROGETTAZIONE - METRICHE

ChgSetSize: è il numero di files che sono stati modificati da commit insieme al file specificato (intra-release).

- Più sono i file modificati insieme in un certo commit e più sarà difficile identificare la natura dei bug eventualmente generato.

Age: è l'età della classe calcolata in termini di settimane (cross-release).

- Una classe con valore di età maggiore risulta più stabile in termini di difettosità e meno propensa ad avere bug.

WeightedAge: è l'età della classe pesata sul numero di LOC_touched (cross-release).

- Si considerano contemporaneamente l'età della classe e il numero totali di LOC aggiunte o rimosse.

Sono anche state considerate le seguenti: **MAX_LOC_Added, AVG_LOC_Added; MAX_Churn, AVG_Churn; MAX_ChgSetSize AVG_ChgSetSize.**

PROGETTAZIONE - MISURAZIONI



- Le metriche relative al codice dei progetti sono state raccolte usando il programma **lsw2-ProjectBugsAnalysis**.
- Usando il file di configurazione è possibile specificare il progetto e la percentuale delle release da considerare una volta prodotto il dataset finale.
- Per recuperare le informazioni sul progetto sono stati usati:
 - **Git**: per ottenere le informazioni relative ai commit.
 - **Jira**: per ottenere le informazioni relative ai tickets e alle versioni.
- Le LOC aggiunte ed eliminate per ogni file vengono raccolte ripercorrendo la storia dei commit ed infine tramite le funzionalità di FeatureCalculator è possibile calcolare le metriche finali.
- Tutti i risultati prodotti sono raccolti in istanze di tipo Record identificate da versione e nome del file e le cui informazioni complessive vengono infine scritte su un file CSV.

PROGETTAZIONE — BUGGYNESS



- La **BUGGYNESS** è una metrica che specifica se una classe è difettosa o meno in una certa release.
- Per capire se una classe è difettosa o meno in una release è stato applicato il seguente algoritmo:
 1. Si raccolgono tutti i ticket relativi al progetto in esame che riguardano bug, sfruttando l'API di **Jira**.
 2. Per ogni ticket si recuperano i valori di AV.
 3. Si identificano i commit del progetto che fanno riferimento ad almeno uno di questi ticket.
 4. Si assegnano ad ogni classe Java modificata dal commit che fa riferimento ad un certo ticket i valori di AV dello stesso. Si stima perciò che la classe sarà difettosa per tutte le AV.
- Per ogni ticket, si controlla che le AV siano consistenti con le FV e pertanto vengono scartati i ticket per cui FV sia precedente a tutte le AV.
- Se per il ticket non sono specificate AV allora viene applicato il metodo **Proportion**.

PROGETTAZIONE – PROPORTION

- Ogni bug ha un ciclo di vita con una specifica **proporzione** definita da OV, FV, IV ed in particolare è possibile calcolarla tramite la formula:

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

- Per ogni ticket, il valore di p è stato calcolato applicando la tecnica di proportion **MovingWindow**, ossia considerando la media dei valori di p per l'1% dei ticket precedenti.
- In caso di AV mancanti o inconsistenti per un certo ticket, è possibile calcolare il valore di IV utilizzando il valore di p di quello stesso ticket.
- In caso non ci siano ticket precedenti, allora non viene assegnato un valore di default a p , ma si applica il metodo di proportion **ColdStart**, ottenendo il valore medio dei valori medi di p per altri 16 progetti Apache.

PROGETTAZIONE - EVALUATION



- Per valutare la difettosità del progetto nel corso delle release, è stata utilizzata la tecnica di validazione **WalkForward**, dal momento che i dati sono strettamente legati ad aspetti temporali.
 - Ad ogni iterazione, il dataset iniziale viene diviso in training set e testing set, considerando una release come testing e tutte le precedenti vengono incluse nel training.
- Per applicare i classificatori e le varie tecniche di FeatureSelection, Balancing e SensitiveClassification è stata utilizzata l'API messa a disposizione da **Weka**.
- E' stata implementata inoltre una classe Analyzer che permette di creare una pipeline di tecniche da valutare successivamente usando Weka.
- L'effettiva valutazione della pipeline di tecniche applicate, avviene nel metodo **evaluation()** della classe Pipeline.
 - Il metodo istanzia un oggetto Evaluation di Weka che esegue la valutazione del modello costruito.

DISCUSSIONE RISULTATI



- Per analizzare le metriche ottenute dalla fase di valutazione è stato utilizzato il programma **Jmp** per realizzare dei grafici che sintetizzassero l'evoluzione di tali metriche.
- Per valutare la rilevanza delle features è stata utilizzata la GUI di Weka.
- Per mitigare l'effetto di **Snoring** è stato scartato il 50% delle versioni più recenti dal dataset, prima di essere valutato dal classificatore.
 - E' più facile che ci siano falsi negativi nelle ultime release. Perciò per evitare di ottenere risultati falsati, sono state valutate solo le prime release di ogni progetto.
- Le per ogni classificatore e tecnica di F.Selection e Balancing si sono analizzate:
 - ✓ **AUC**
 - ✓ **Kappa**
 - ✓ **Precision**
 - ✓ **Recall**

DISCUSSIONE RISULTATI - FEATURES

The screenshot shows the Weka Explorer window with the 'Select attributes' tab active. The 'Attribute Evaluator' is set to 'CfsSubsetEval -P 1 -E 1'. The 'Search Method' is set to 'GreedyStepwise -T -1.7976931348623157E308 -N -1 -num-slots 1'. The 'Attribute Selection Mode' is set to 'Use full training set'. The 'Attribute selection output' window shows the results of the search method, including the 'Merit of best subset found' and the 'Selected attributes'.

Attribute Evaluator
Choose CfsSubsetEval -P 1 -E 1

Search Method
Choose GreedyStepwise -T -1.7976931348623157E308 -N -1 -num-slots 1

Attribute Selection Mode
☒ Use full training set
☐ Cross-validation Folds 10 Seed 1

(Nom) Buggy

Start Stop

Result list (right-click for options)
13:41:01 - BestFirst + CfsSubsetEval
13:41:18 - GreedyStepwise + CfsSubsetEval

Attribute selection output

```
WeightedAge
Buggy
Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===

Search Method:
  Greedy Stepwise (forwards).
  Start set: no attributes
  Merit of best subset found: 0.221

Attribute Subset Evaluator (supervised, Class (nominal): 17 Buggy):
  CFS Subset Evaluator
  Including locally predictive attributes

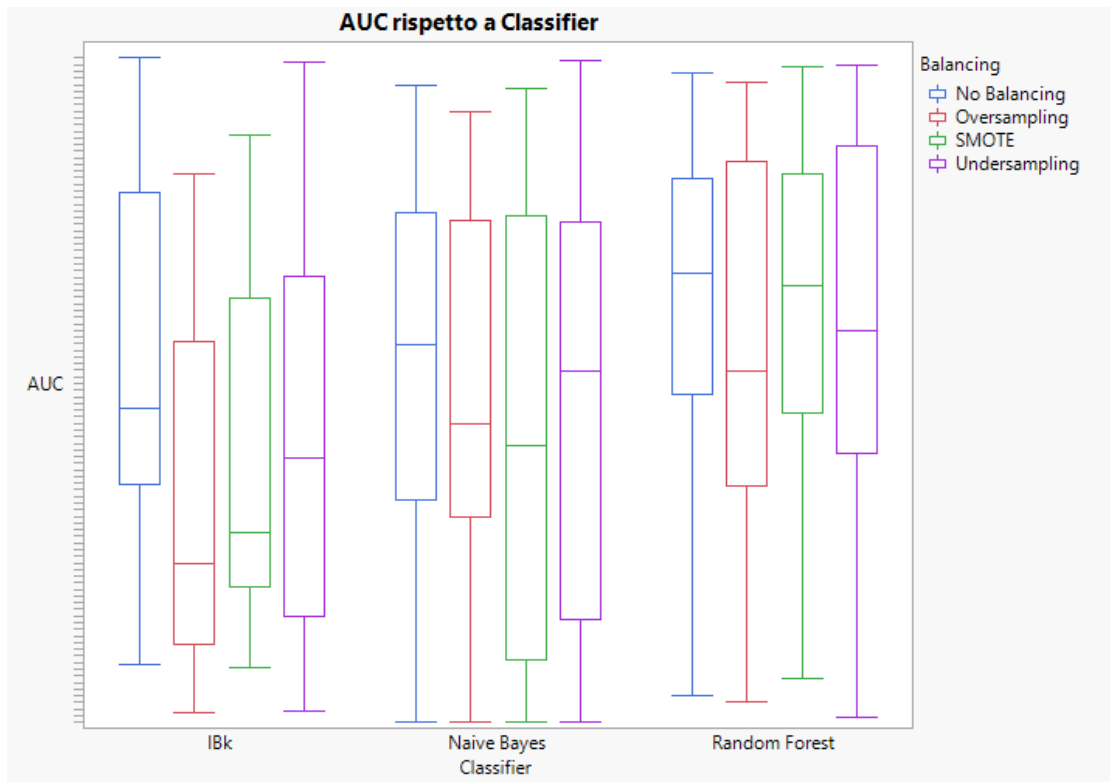
Selected attributes: 1,4,13,14,16 : 5
  Size
  NFix
  MAX_ChgSet
  AVG_ChgSet
  WeightedAge
```

➤ Applicando la Feature Selection, è possibile identificare quali sono gli attributi che hanno una rilevanza maggiore rispetto alla variabile di classificazione (Buggy).

➤ Sia applicando BestFirst, sia applicando GreedyStepwise, si ottiene che le metriche più rilevanti sulla base del dataset fornito sono:

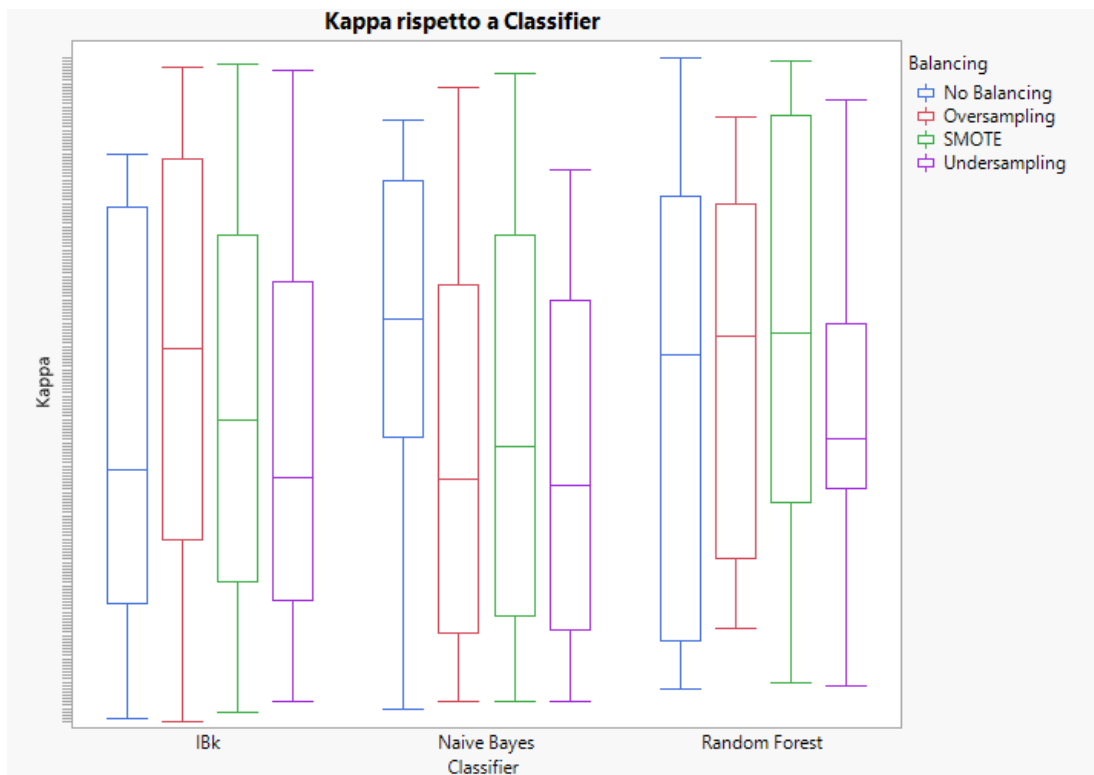
- ✓ Size
- ✓ NFix
- ✓ MAX_ChgSet
- ✓ AVG_ChgSet
- ✓ WeightedAge

DISCUSSIONE RISULTATI – AUC



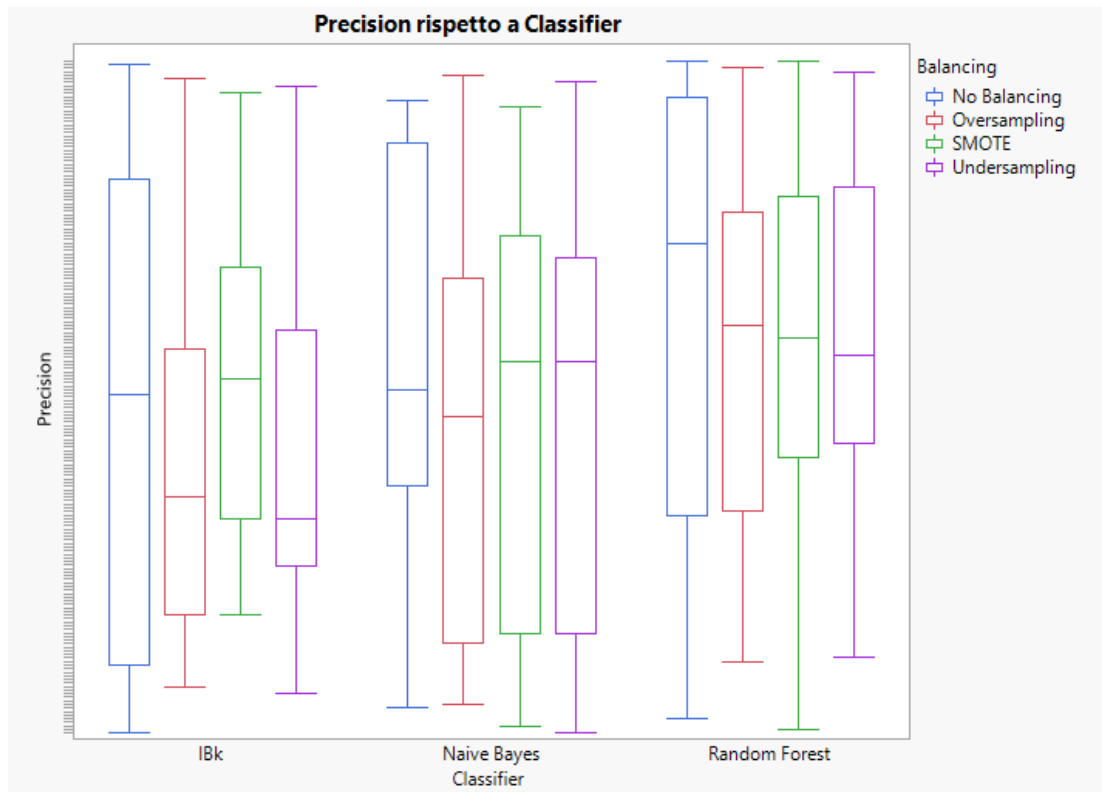
- Si considerano i dati del progetto BookKeeper.
- La tecnica **Undersampling** è quella che peggiora di meno le prestazioni dei classificatori IBk e NaiveBayes.
- La tecnica **Oversampling** invece, peggiora notevolmente le prestazioni di tutti i classificatori rispetto a quando non vengono applicate tecniche di bilanciamento.
- La tecnica **SMOTE** peggiora le prestazioni di tutti i classificatori, maggiormente in IBk e NaiveBayes.

DISCUSSIONE RISULTATI — KAPPA



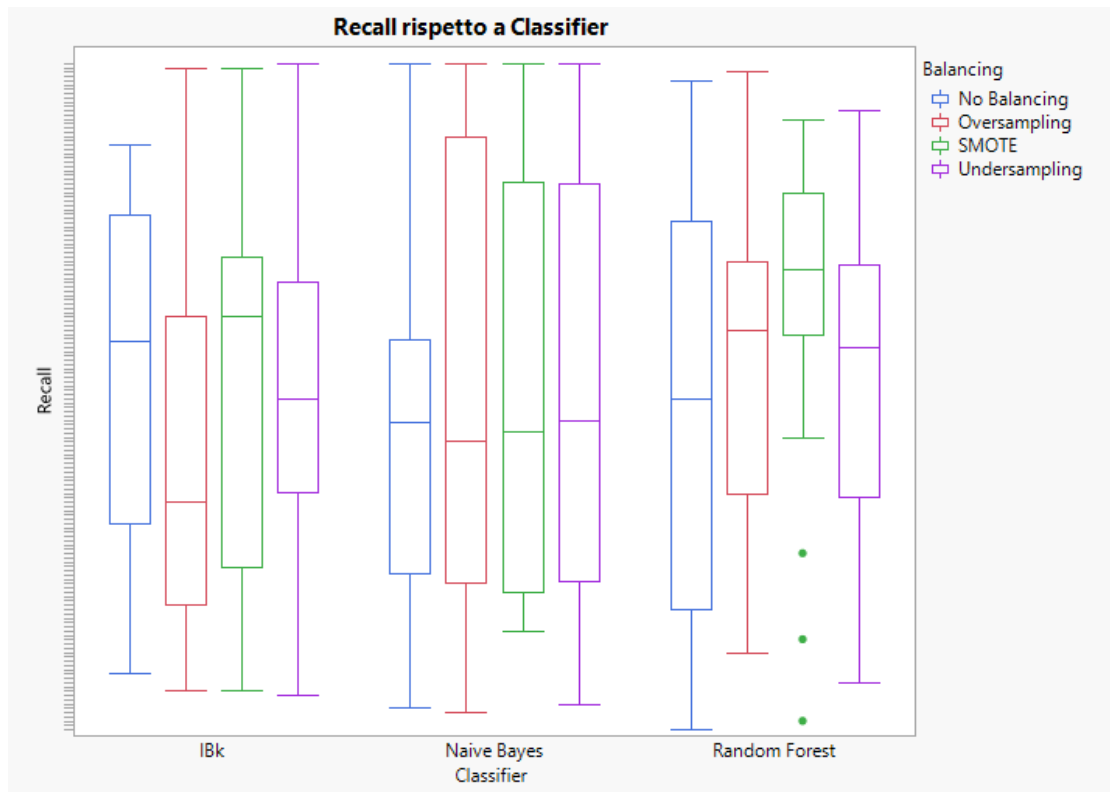
- Si considerano i dati del progetto BookKeeper.
- Le prestazioni di **NaiveBayes** peggiorano parecchio rispetto ad un classificatore “dummy”, applicando qualsiasi tecnica di bilanciamento sul dataset in input.
- Le prestazioni di **RandomForest** invece, migliorano leggermente applicando il bilanciamento con SMOTE e peggiorano applicando Undersampling.
- Nel caso di **IBk**, applicare Oversampling migliora parecchio le prestazioni, mentre Undersampling sembra influire di meno.

DISCUSSIONE RISULTATI — PRECISION



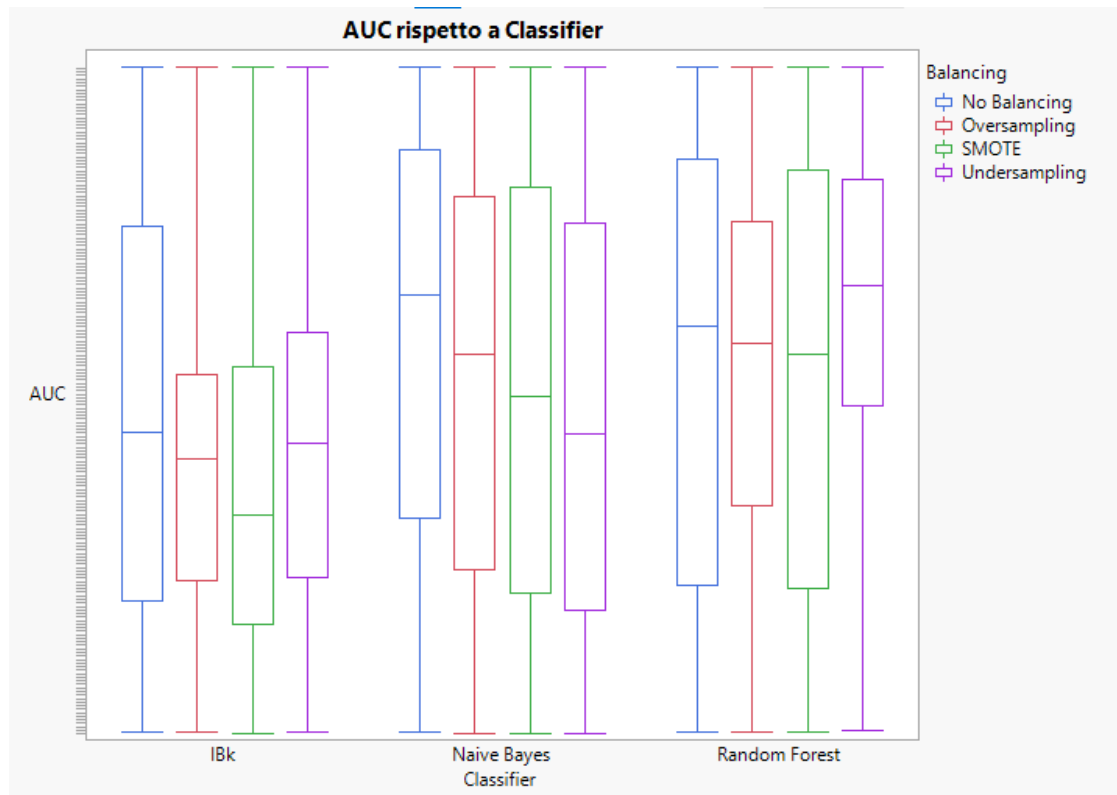
- Si considerano i dati del progetto BookKeeper.
- SMOTE e Undersampling aumentano leggermente la Precision nel caso di NaiveBayes.
- La Precision peggiora notevolmente applicando **Undersampling** ai classificatori RandomForest e IBk.
 - Il maggior peggioramento si ha nel caso di IBk
- Il bilanciamento ha in generale un impatto minore sulla Precision valutata con NaiveBayes rispetto agli altri classificatori.
- SMOTE aumenta leggermente la Precision nel caso di IBk.

DISCUSSIONE RISULTATI — RECALL



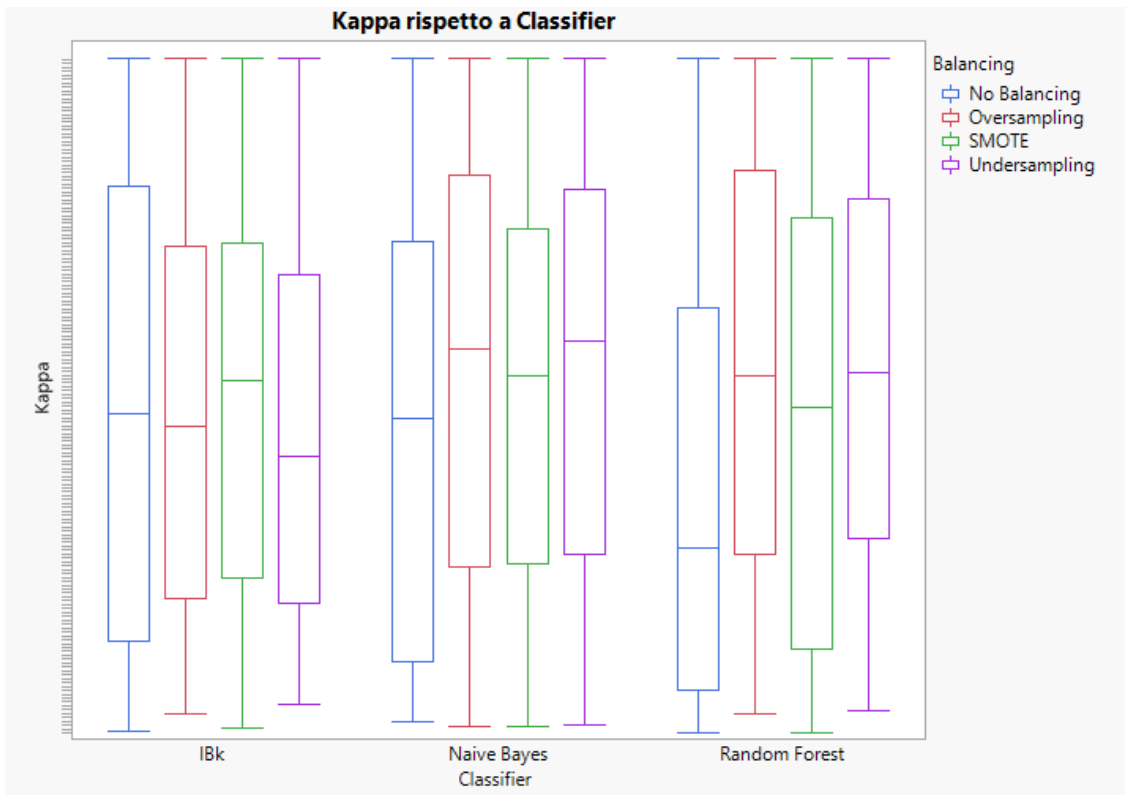
- Si considerano i dati del progetto BookKeeper.
- Nel caso di **RandomForest**, tutte e tre le tecniche aumentano il numero di positivi individuati e quindi la Recall.
 - Non sono hanno valori mediani più alti, ma anche la variabilità della distribuzione è minore.
- Nel caso di IBk e RandomForest, si ottengono più positivi individuati applicando **SMOTE**, sebbene la sua influenza sia più evidente per il secondo.
- In IBk applicare Oversampling o Undersampling riduce il numero di positivi individuati.

DISCUSSIONE RISULTATI – AUC



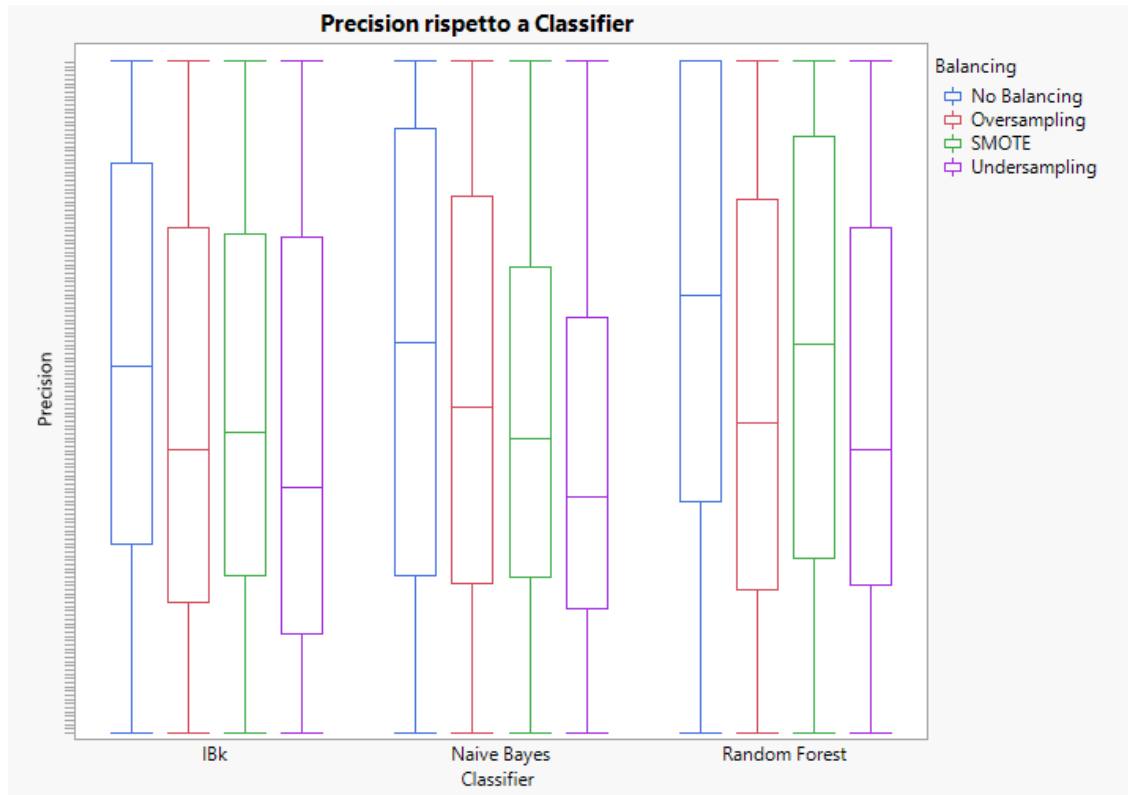
- Si considerano i dati del progetto Syncope.
- Sia per **IBk** che per **NaiveBayes**, applicare tecniche di bilanciamento peggiora in tutti i casi le prestazioni.
- Invece per quanto riguarda **RandomForest** si ottiene un leggero peggioramento del valore di AUC con Oversampling e SMOTE, mentre si ottiene un miglioramento applicando Undersampling.
- Il peggioramento maggiore si ottiene con NaiveBayes applicando Undersampling.

DISCUSSIONE RISULTATI — KAPPA



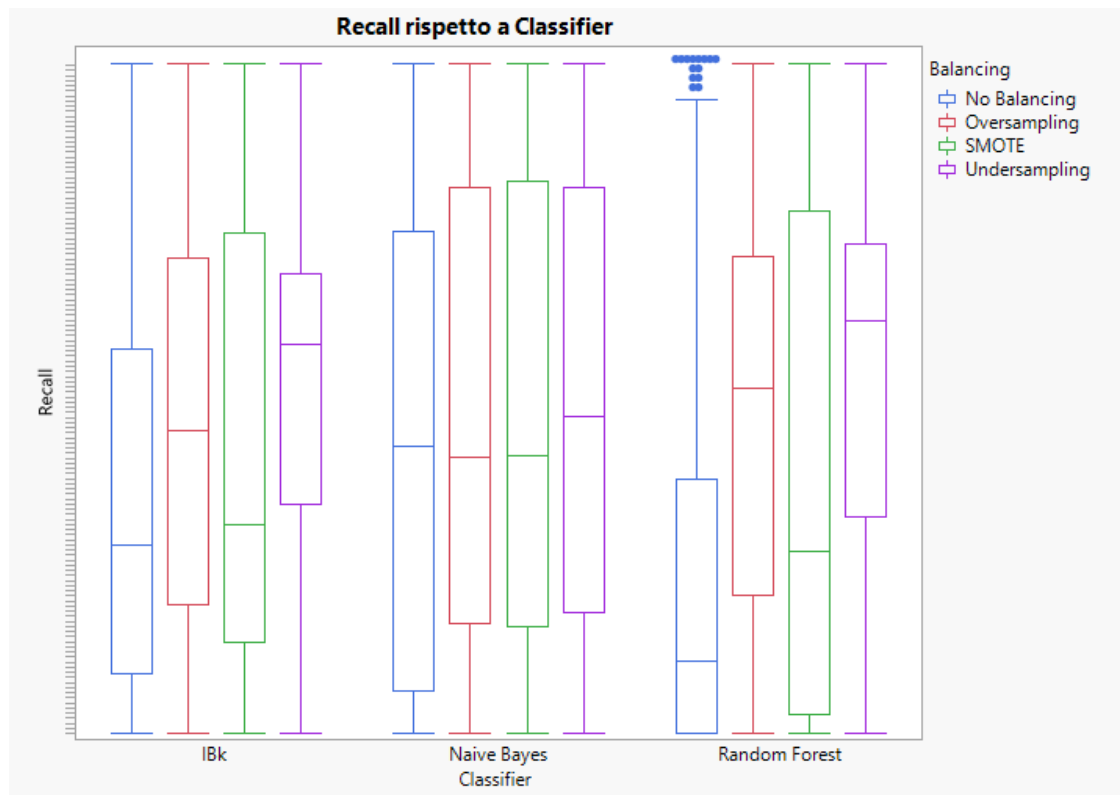
- Si considerano i dati del progetto Syncope.
- In questo caso applicando le tre tecniche di bilanciamento del dataset si ottengono aumenti delle prestazioni dei tre classificatori rispetto al classificatore “dummy”.
 - In maniera inferiore per IBk.
- Per **NaiveBayes** e **RandomForest**, tutte e tre le tecniche di bilanciamento migliorano notevolmente .
- Per **IBk** l'unico miglioramento si ottiene applicando **SMOTE**.

DISCUSSIONE RISULTATI — PRECISION



- Si considerino i dati del progetto Syncope.
- In questo caso il valore di Precision diminuisce applicando qualsiasi tecnica di bilanciamento del dataset per tutti i classificatori considerati.
- La tecnica che provoca il peggioramento maggiore del valore della Precision è **Undersampling** per tutti e tre i classificatori.
- La tecnica con la Precision maggiore invece è **SMOTE** per IBk e RandomForest; **Oversampling** per NaiveBayes.

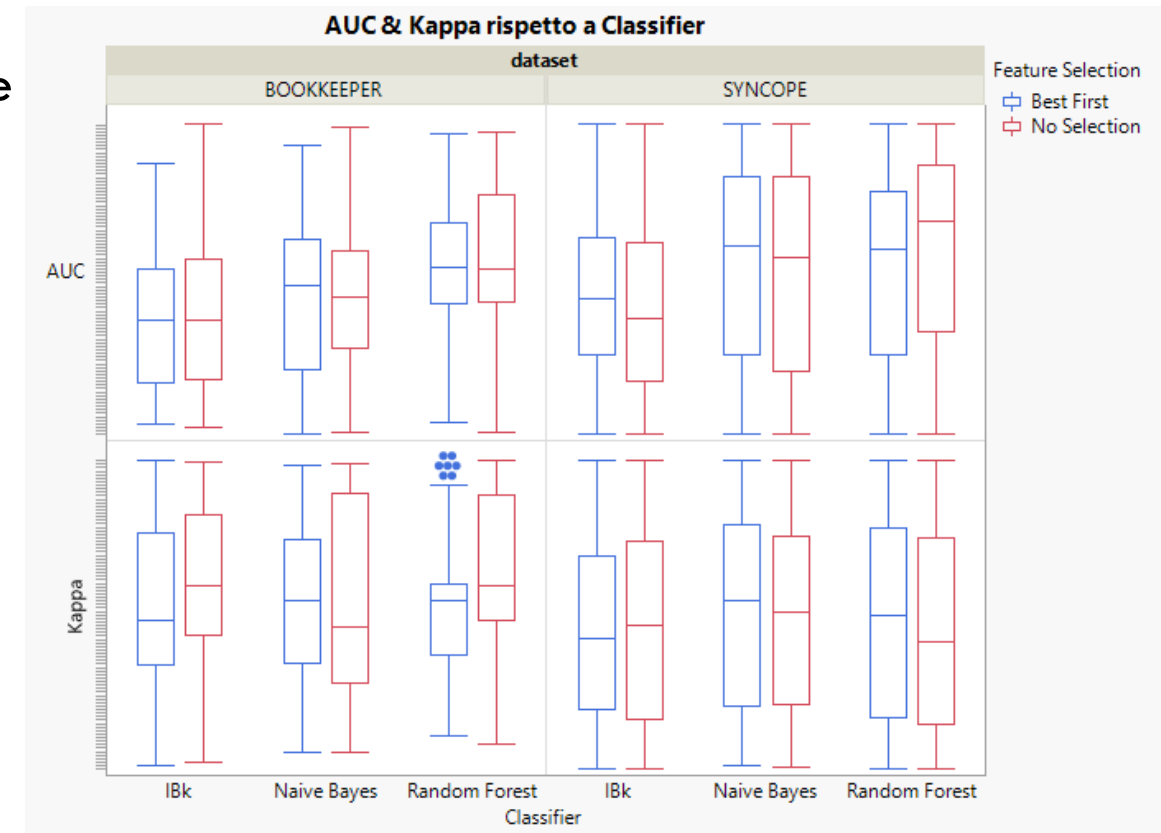
DISCUSSIONE RISULTATI — RECALL



- Si considerano i dati del progetto Syncope.
- Sia per **RandomForest** che per **IBk**, applicare tecniche di bilanciamento aumenta notevolmente il valore di Recall.
- Per **NaiveBayes** invece applicare qualsiasi tecnica di bilanciamento, tende a non influire più di tanto sul valore di Recall.
- Per quanto riguarda la Recall, le tecniche di Undersampling e Oversampling sembrano comportarsi generalmente meglio rispetto a SMOTE, per tutti e tre i classificatori.

CONCLUSIONI – FEATURE SELECTION

- Tra le 16 feature calcolate nel dataset dato in input al modello, solamente **5** sono risultate rilevanti al fine di predire la difettosità delle classi.
 - E' interessante notare come sia stata selezionata la **WeightedAge**, rimuovendo invece gli attributi di LOC_Touched e Age. Infatti, WeightedAge include al suo interno informazioni relative ad entrambe le metriche, pertanto risulta maggiormente significativa.
- In alcuni casi applicare la selezione delle feature migliora abbastanza le prestazioni.
 - Applicando BestFirst sul dataset di Syncope, si ottengono miglioramenti per NaiveBayes e per IBk.
- Il classificatore che migliora sempre applicando BestFirst per entrambi i dataset è NaiveBayes. Questo potrebbe essere dovuto al fatto che NaiveBayes assume che gli attributi del dataset siano il più possibile scorrelati tra loro.



CONCLUSIONI — BILANCIAMENTO

- Tra le tecniche di bilanciamento, non ce n'è una che è nettamente migliore delle altre, ma dipende intrinsecamente dal dataset di partenza e dal classificatore considerato.
- Su Syncope, si ottiene un numero maggiore di positivi individuati applicando **Undersampling**, mentre su BookKeeper si ottengono più positivi applicando **SMOTE**.
- Confrontando i valori di Recall senza applicare le tecniche di bilanciamento è evidente come essa sia maggiore sul dataset di BookKeeper, poiché esso è meno sbilanciato.
 - Buggy BookKeeper: 38%
 - Buggy Syncope: 5%
- Per questo motivo, il bilanciamento ha un impatto mediamente maggiore su Syncope rispetto a BookKeeper a prescindere dal classificatore.
- Il classificatore su cui mediamente non ha impatto il bilanciamento è **NaiveBayes**. Questo può essere dovuto al fatto che esso è un classificatore ad approccio generativo e dunque si presuppone che i dati siano generati seguendo lo stesso processo di generazione.

CONCLUSIONI - CLASSIFICATORI

- Tra i diversi classificatori, non ce n'è uno che ha performance migliori degli altri, ma anche in questo caso i risultati variano in base al dataset iniziale e alle varie tecniche applicate.
- Il classificatore che in generale ha le performance più basse è proprio **NaiveBayes**, lo stesso classificatore su cui hanno meno impatto le tecniche di bilanciamento, nonostante la sostanziale differenza di bilanciamento dei dataset iniziali.
- Performance leggermente migliori si ottengono con RandomForest, il quale a differenza degli altri classificatori rientra nella categoria dei **bagging classifiers**.
- Dal momento che è stato usato WalkForward come tecnica di validazione, è possibile che ci siano iterazioni particolarmente sfortunate che pesano negativamente sulle prestazioni. Generando molteplici alberi di decisione su cui mediare il risultato finale delle prestazioni, RandomForest riesce a mitigare i risultati negativi aumentando le performance generali.

GRAZIE PER L'ATTENZIONE!



Link alla repository Github:

lsw2-ProjectBugsDataset: [Diana0422/lsw2-ProjectBugsDataset \(github.com\)](https://github.com/Diana0422/lsw2-ProjectBugsDataset)

lsw2-ProjectBugsAnalysis: [Diana0422/lsw2-ProjectBugsAnalysis \(github.com\)](https://github.com/Diana0422/lsw2-ProjectBugsAnalysis)



Link a SonarCloud:

lsw2-ProjectBugsDataset: [lsw2-ProjectBugsDataset - Diana0422 \(sonarcloud.io\)](https://sonarcloud.io/lsw2-ProjectBugsDataset-Diana0422)

lsw2-ProjectBugsAnalysis: [lsw2-ProjectBugsAnalysis - Diana0422 \(sonarcloud.io\)](https://sonarcloud.io/lsw2-ProjectBugsAnalysis-Diana0422)