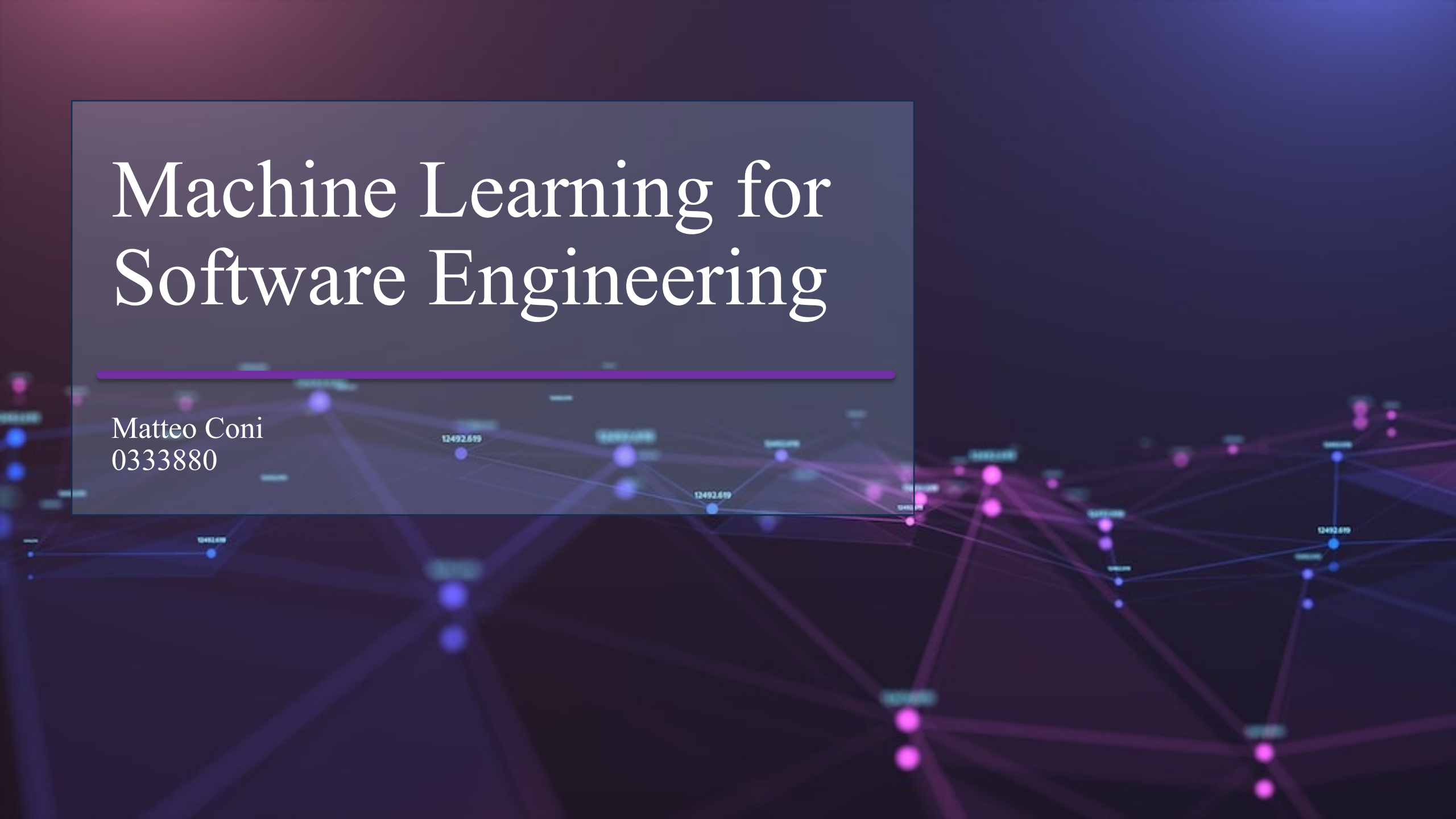
Machine Learning for Software Engineering

Matteo Coni
0333880



AGENDA

- **Contesto**
- **Obiettivi**
- **Metodologia**
- **Risultati ed Analisi**
- **Link Utili**

CONTESTO

Qualsiasi progetto nell'ambito dell'Ingegneria del Software necessita di una fase di test: quest'ultima però è costosa in termini ore-uomo e non sempre può essere effettuata efficacemente

INOLTRE

I BUG all'interno del software costano alle aziende circa $2.84 trilioni di dollari

CONTESTO (2)

È possibile predirre le classi che contengono bug tramite

MACHINE LEARNING

Uso di **modelli** e **classificatori** già esistenti

Testing Set
Training Set

Riduzione dell'**effort** legato alla fase di testing senza compromettere l'efficacia

OBIETTIVI

Valutare le prestazioni dei classificatori **IBK, NAIVE BAYES e RANDOM FOREST** nella predizione della bugginess delle classi nei progetti **APACHE BOOKKEEPER e ZOOKEEPER**

Confronto delle prestazione dei modelli al variare delle metriche e delle tecniche applicate

- **Feature Selection**
- **Sampling**
- **Cost Sensitivity**

METODOLOGIA

- Recupero dei **ticket** fixati ed etichettati «BUG» su **JIRA**
- Recupero dei **file per ogni release** e dei vari commit relativi ai bug individuati attraverso **GIT**
- **Proportion**
- Scelta e calcolo delle **metriche**
- **Training** Set VS **Testing** Set
- **Walk Forward**

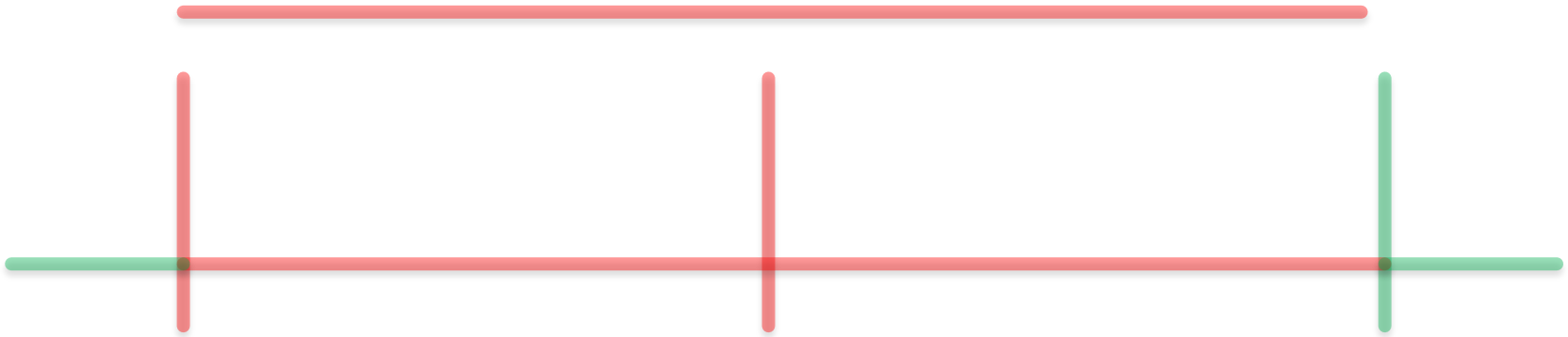
METODOLOGIA: JIRA

Come individuo le coppie (classe, release) buggy?

Ogni bug ha un ciclo di vita caratterizzato da IV, OV e FV: informazioni ricavate dall'Issue Tracking System JIRA

Jira Software

AV: Affected Version
Da IV (inclusa) a FV (esclusa)



IV: Injected Version
Release in cui il bug è stato introdotto

OV: Opening Version
Release in cui il bug è stato rilevato a seguito di una failure nel sistema

FV: Fixed Version
Release in cui il bug è stato eliminato

METODOLOGIA: Proportion

Come individuo le coppie (classe, release) buggy?

Ogni ticket JIRA contiene sempre le informazioni su quali siano la **OV** e la **FV**, mentre non tutti gli issue aperti riportano l'**IV**: è possibile stimarla attraverso una tecnica chiamata **PROPORTION.**

Vengono scartati ticket senza FV, con OV > FV, IV > OV e IV = OV = FV.

$$IV = FV - (FV - OV) \cdot p$$

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

METODOLOGIA (2): Proportion

Come individuo le coppie (classe, release) buggy?

COLD START

Variante utilizzata nel caso i bug con IV disponibile siano meno di 5. In questo caso la costante di proporzionalità viene calcolata a partire dai bug di altri progetti Apache, calcolandone la mediana, e rispettando l'ordine temporale.

INCREMENTAL

La costante di proporzionalità viene calcolata come la media di tutti i «p» dei bug aventi Injection Version e riferiti temporalmente prima del bug a cui si sta applicando proportion.

METODOLOGIA: GIT

Recupero file e commit per ogni release

- I file java relativi ad ogni progetto sono stati ottenuti attraverso **GIT**. Inoltre, per ogni bug, sono stati ricavati tutti i **commit** relativi ad esso. Si è cosi concluso il **labeling** completo delle classi, definendo in quale release ciascuna di esse fosse buggy e in quale no.

- Per eliminare il fenomeno dello **snoring,** che consiste nell'avere coppie (classe, release) non buggy ma che in realtà hanno difetti non emersi (bug dormienti), è opportuno **scartare** tutti i dati relativi alla **seconda metà delle release**: si assume quindi la prima metà più stabile.

METODOLOGIA: Metriche

Quali metriche* sono state considerate?

- **LOC**: numero di linee di codice
- **NR**: numero di revisioni
- **AUTHORS:** numero di autori dei commit
- **LOC TOUCHED:** somma delle LOC aggiunte e di quelle rimosse
- **LOC ADDED:** somma delle LOC aggiunte
- **MAX LOC ADDED:** numero di LOC massimo aggiunte in un commit
- **AVG LOC ADDED:** media di LOC aggiunte
- **CHURN:** somma di |LOC aggiunte – LOC rimosse|
- **MAX CHURN:** valore massimo del churn in una revisione
- **AVG CHURN:** valore medio del churn

*Tutte le metriche sono state calcolate **«intra-release»**

METODOLOGIA: Walk Forward

Valutazione dei classificatori

Run \ Release	1	2	3	4	5
1	Testing				
2	Training	Testing			
3	Training	Training	Testing		
4	Training	Training	Training	Testing	
5	Training	Training	Training	Training	Testing

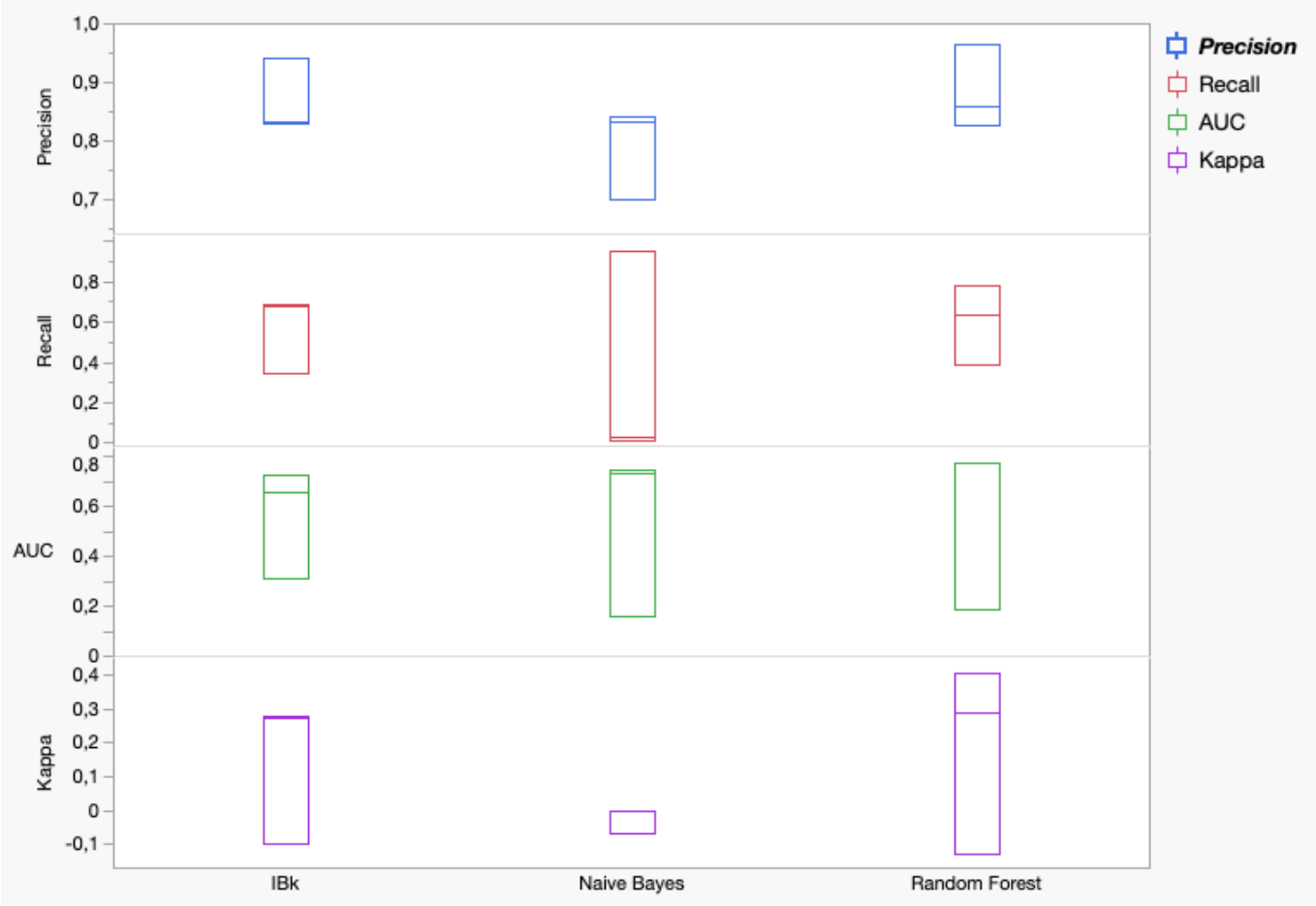
Per stabilire quale classificatore abbia le prestazioni migliori, è necessario valutarli: per fare ciò è stata usata la tecnica **Walk Forward**.

È una tecnica di validazione Time-Series, tiene cioè conto dell'ordine temporale dei dati: **non è quindi possibile utilizzare informazioni future per predire il passato.**

Ad ogni iterazione, per il training set k, viene effettuato nuovamente il calcolo delle IV calcolando proportion solamente sui bug con FV < k+1

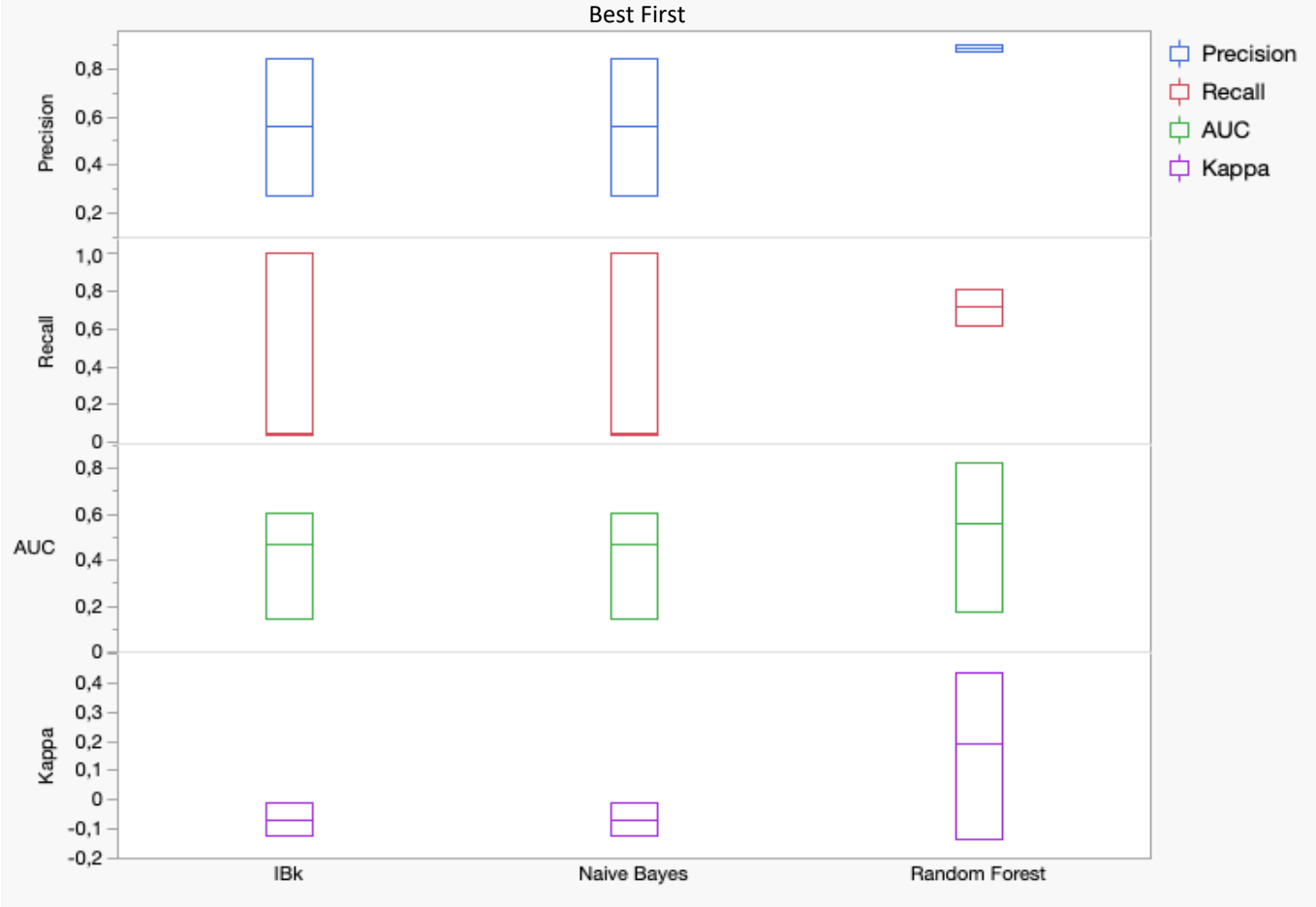
RISULTATI: BookKeeper

Nessun filtro applicato



Senza applicare nessun filtro, si nota che, tranne che nel caso della Recall, il classificatore **Random Forest** sembra comportarsi leggermente meglio. La quantità di dati è limitata a causa delle poche release disponibili.

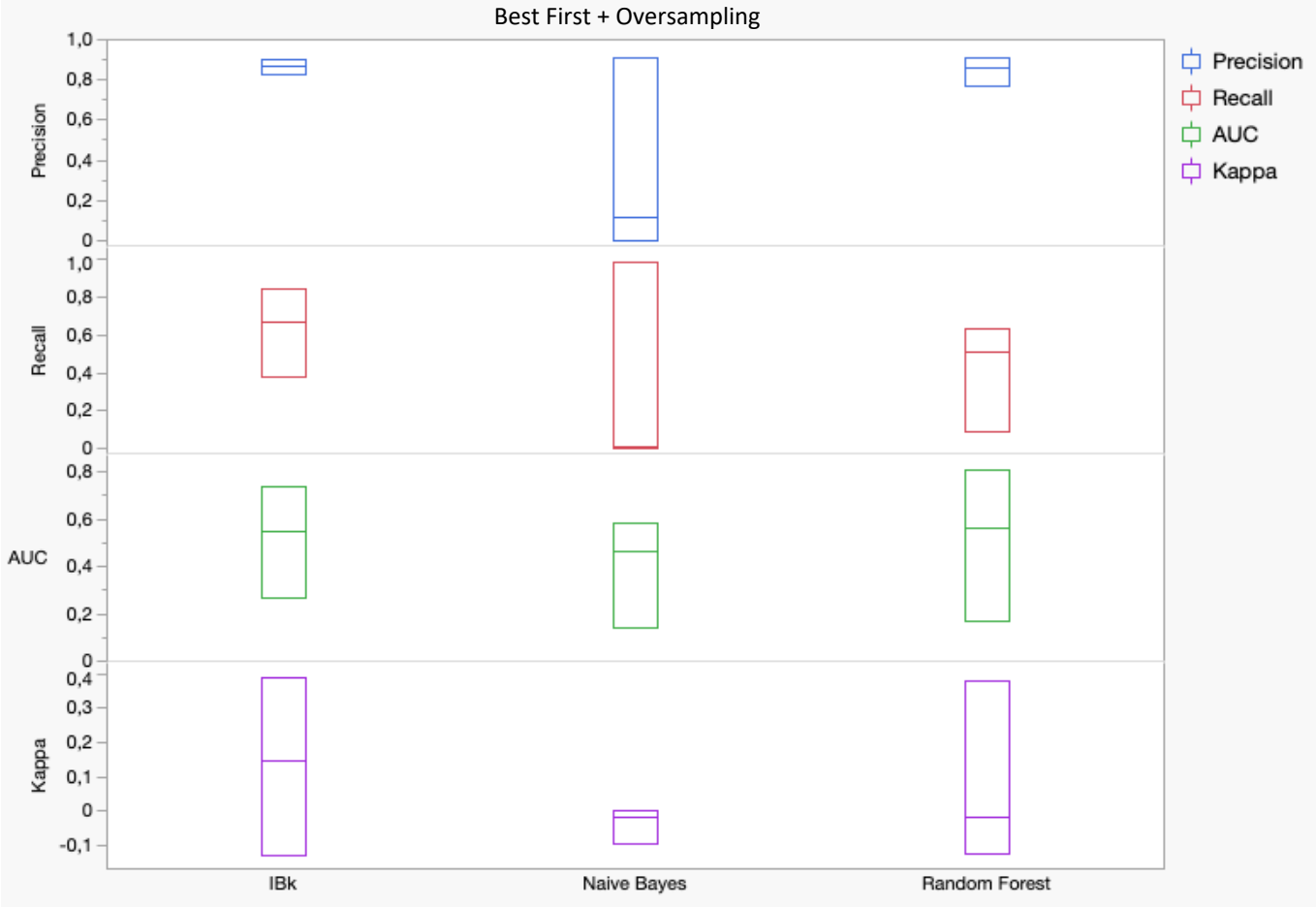
RISULTATI: BookKeeper



Solo Feature Selection (Best First)

Applicando soltanto feature selection (Best First), si nota che, anche qui tranne che nel caso della Recall, **il classificatore Random Forest** sembra comportarsi meglio. La quantità di dati è limitata a causa delle poche release disponibili.

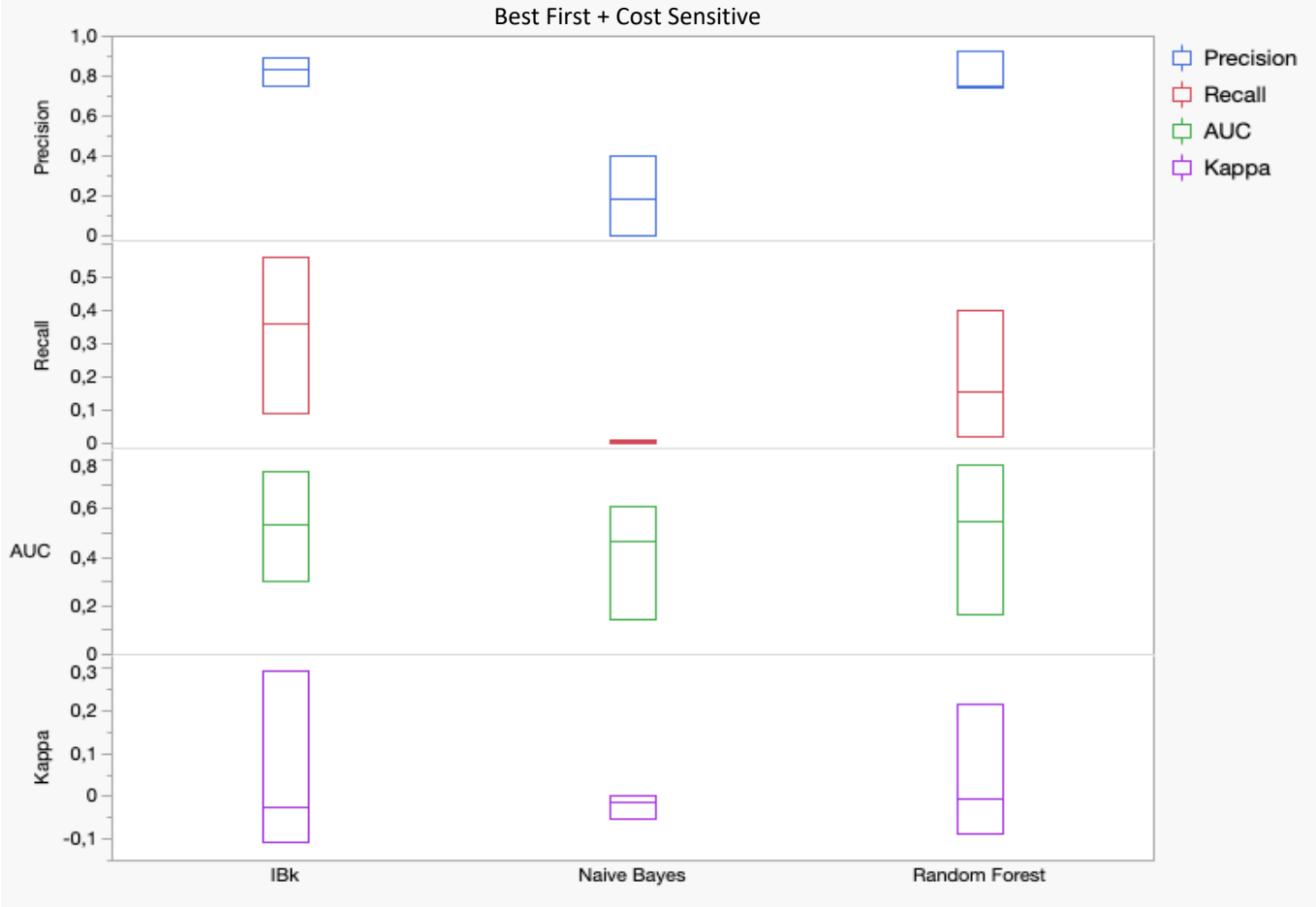
RISULTATI: BookKeeper



Feature Selection (Best First)
+
Oversampling

Applicando **feature selection e oversampling** , si nota che il classificatore **IBK** è quello che sembra comportarsi meglio. Si comporta similmente, tranne che per la Recall, Random Forest. La quantità di dati è limitata a causa delle poche release disponibili.

RISULTATI: BookKeeper

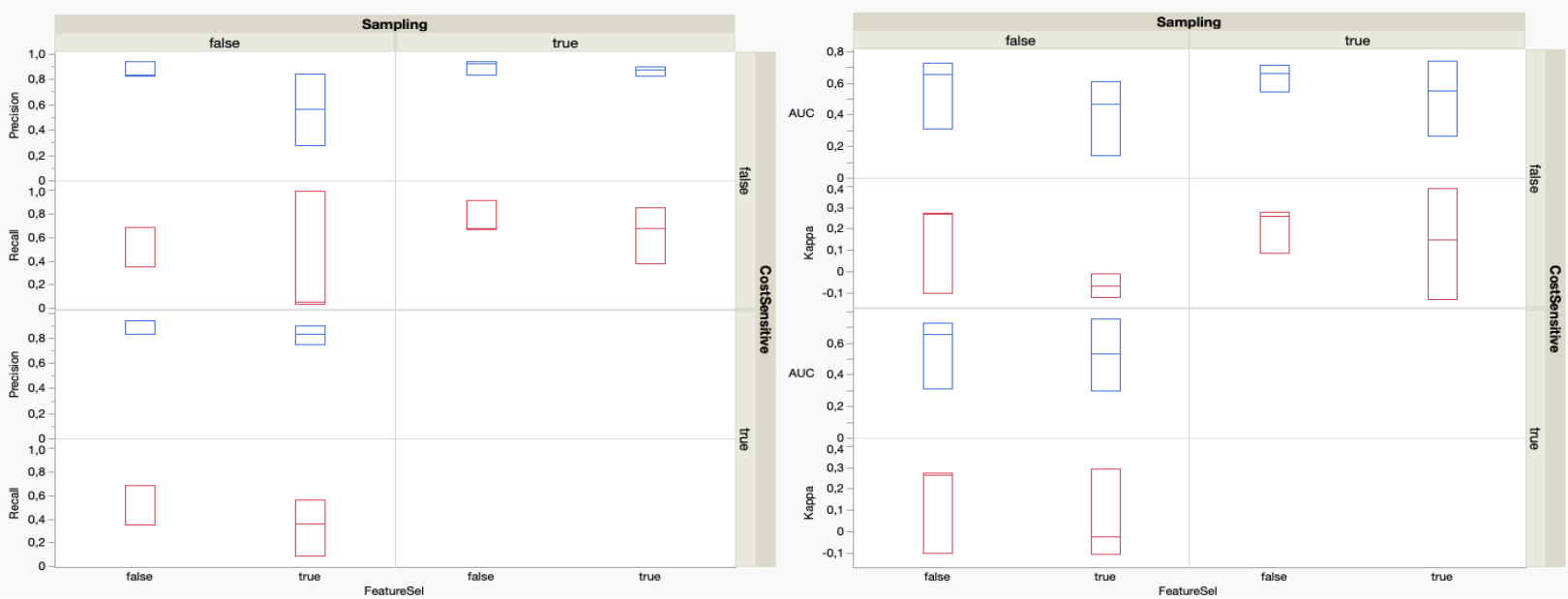


Feature Selection (Best First)
+
Cost Sensitive (CFN = 10*CFP)

Applicando **feature selection e cost sensitive**, si nota che il classificatore **IBK** è quello che sembra comportarsi meglio nuovamente. La quantità di dati è limitata a causa delle poche release disponibili.

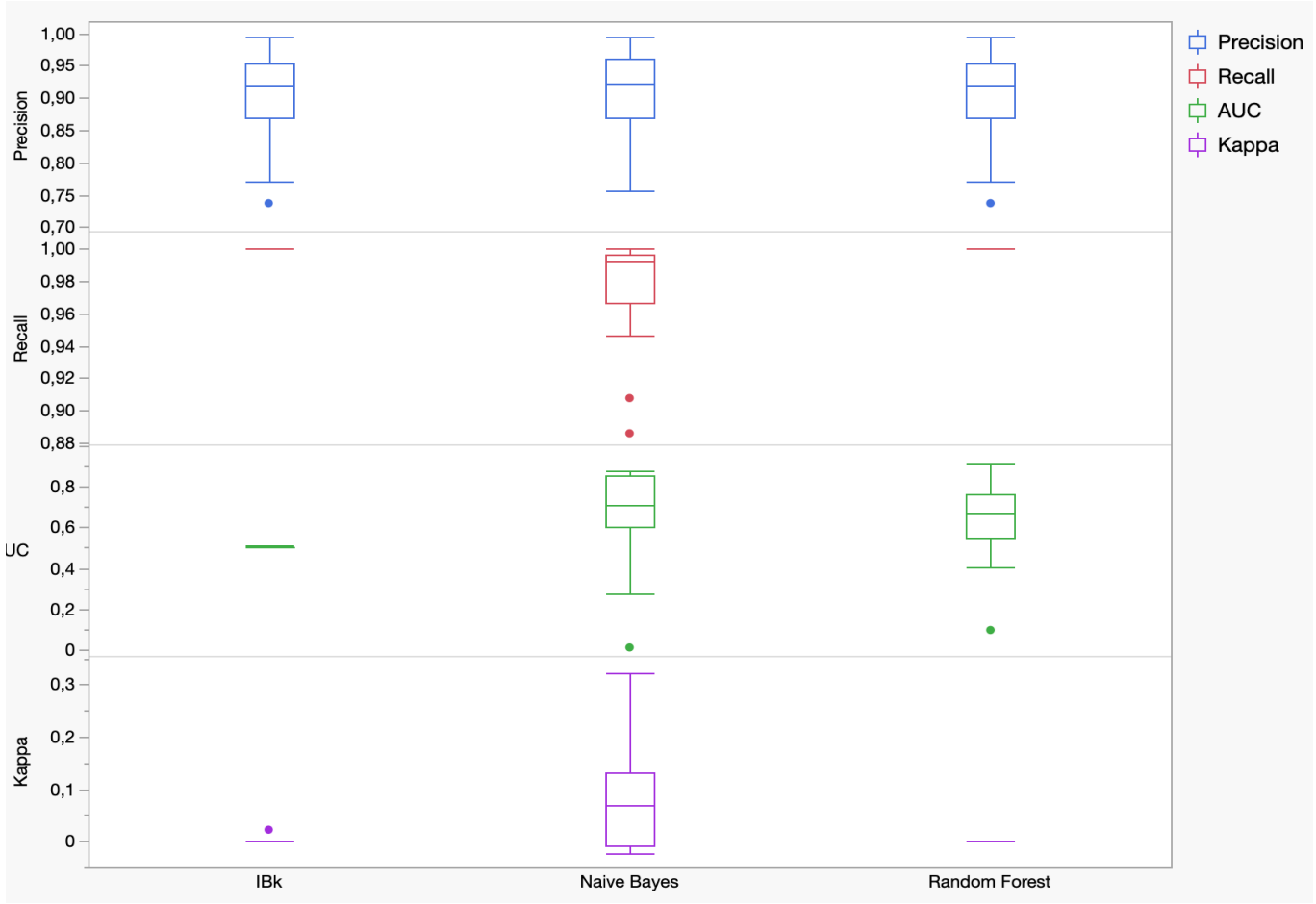
RISULTATI: BookKeeper

Dati i risultati precedenti, è possibile affermare che IBK si comporta leggermente meglio rispetto agli altri classificatori. Osservando i grafichi sotto, si può notare come, per quanto riguarda **Precision e Recall**, si ottengono dei risultati migliori **senza utilizzare nessun filtro** oppure con **OVERSAMPLING**. Analizzando anche **AUC e KAPPA**, si conferma quanto detto, ottenendo risultati leggermente migliori di nuovo con il bilanciamento.



È doveroso notare che si hanno risultati migliori con la tecnica **oversampling** attiva: questo può derivare dal fatto che, avendo un dataset relativamente piccolo e uno sbilanciamento importante tra i campioni, l'aumento dei dati ha un impatto maggiore. In più non viene scartato nessun dato.

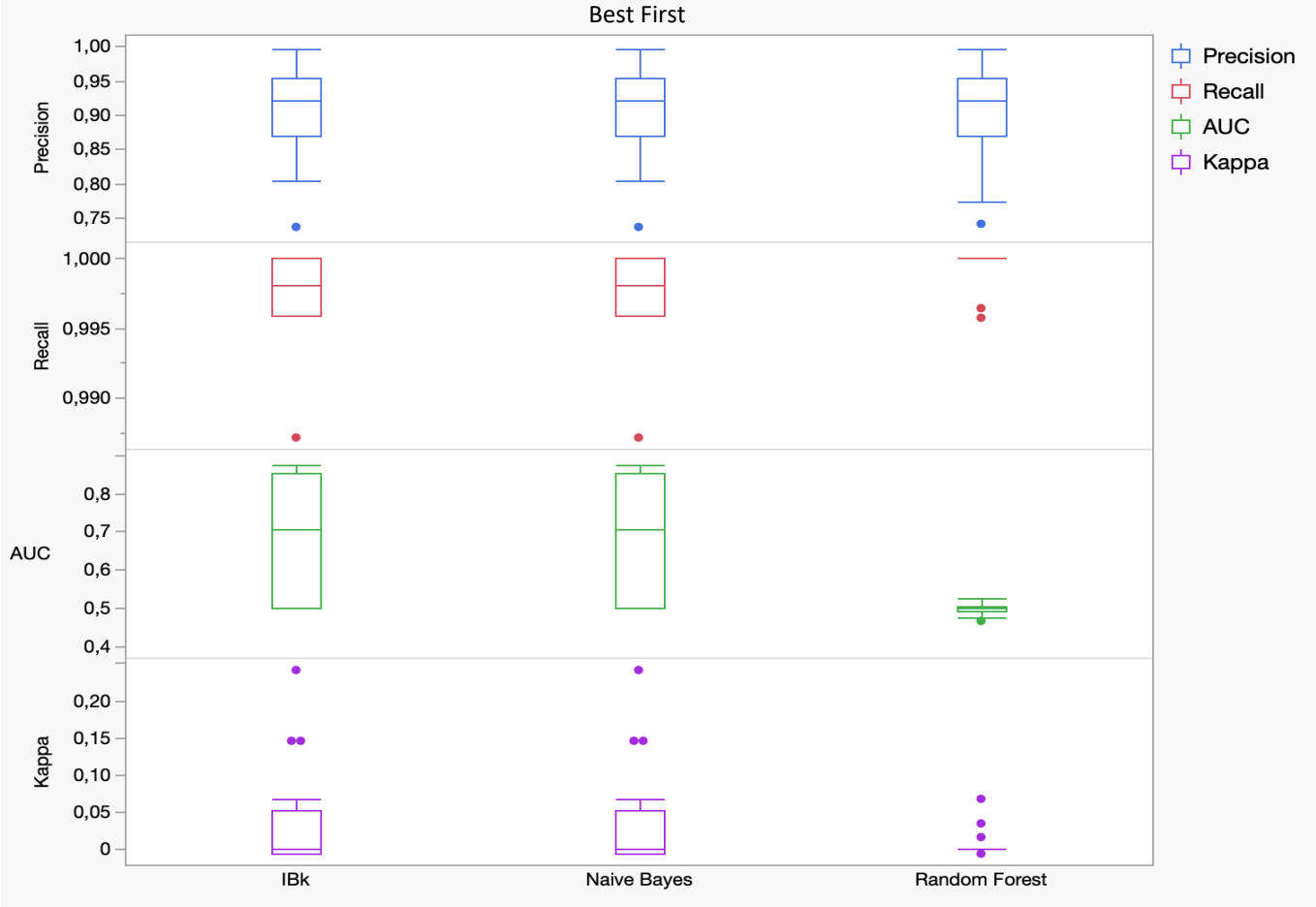
RISULTATI: ZooKeeper



Nessun filtro applicato

Senza applicare nessun filtro, si nota che la precision è simile in tutti e tre i classificatori. La **recall** sembra più alta in **IBK** e **Random Forest**, ma essendo la scala molto stretta, anche i valori ottenuti da **Naive Bayes** possono essere considerati buoni. L'AUC e Kappa sono risultate più alte con **Naive Bayes**.

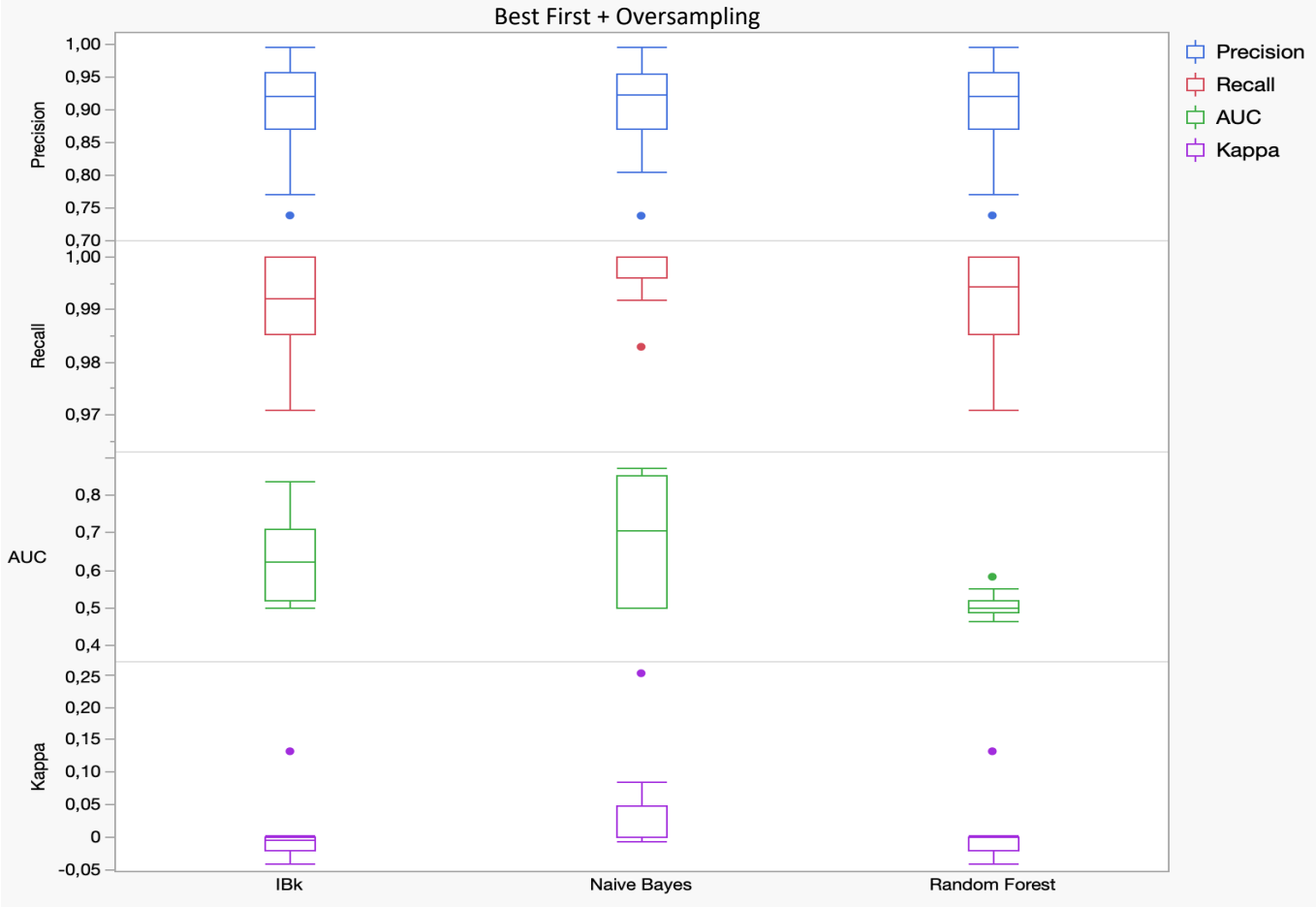
RISULTATI: ZooKeeper



Solo Feature Selection (Best First)

Applicando soltanto **feature selection (Best First)**, si nota che la precision e la recall mantengono dei valori molto alti in tutti e tre i casi, mentre l'AUC scende nel caso di Random Forest. I risultati sono leggermente migliori rispetto al caso senza filtri, con **IBK e Naive Bayes** al primo posto.

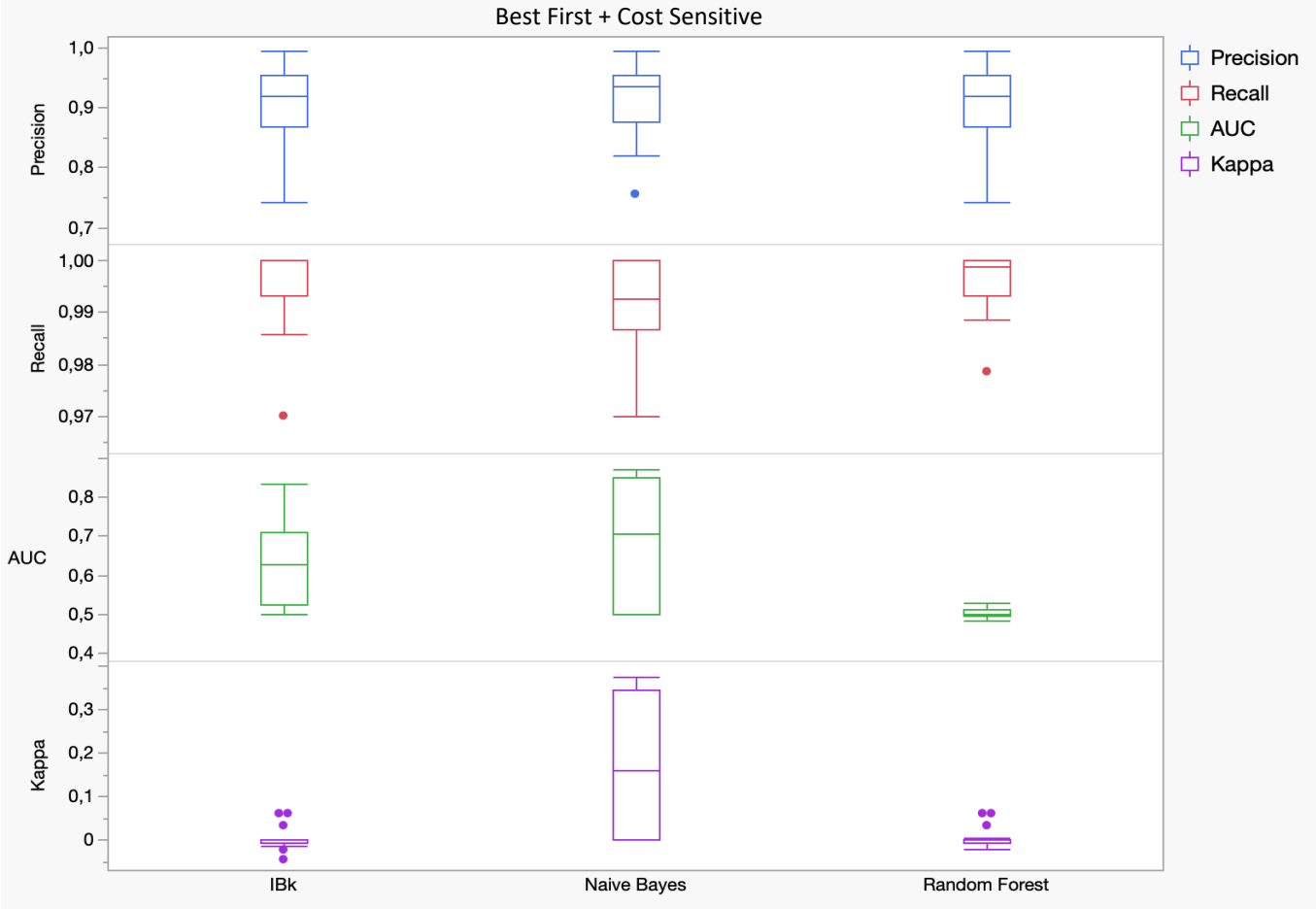
RISULTATI: ZooKeeper



Feature Selection (Best First) + Oversampling

Applicando **feature selection e oversampling** , si nota che il classificatore **Naive Bayes** si comporta mediante meglio in tutte le metriche: con esso si ottiene una **Recall e un'AUC** leggermente più alta, rimanendo però molto simile al caso precedente con solo Feature Selection.

RISULTATI: ZooKeeper

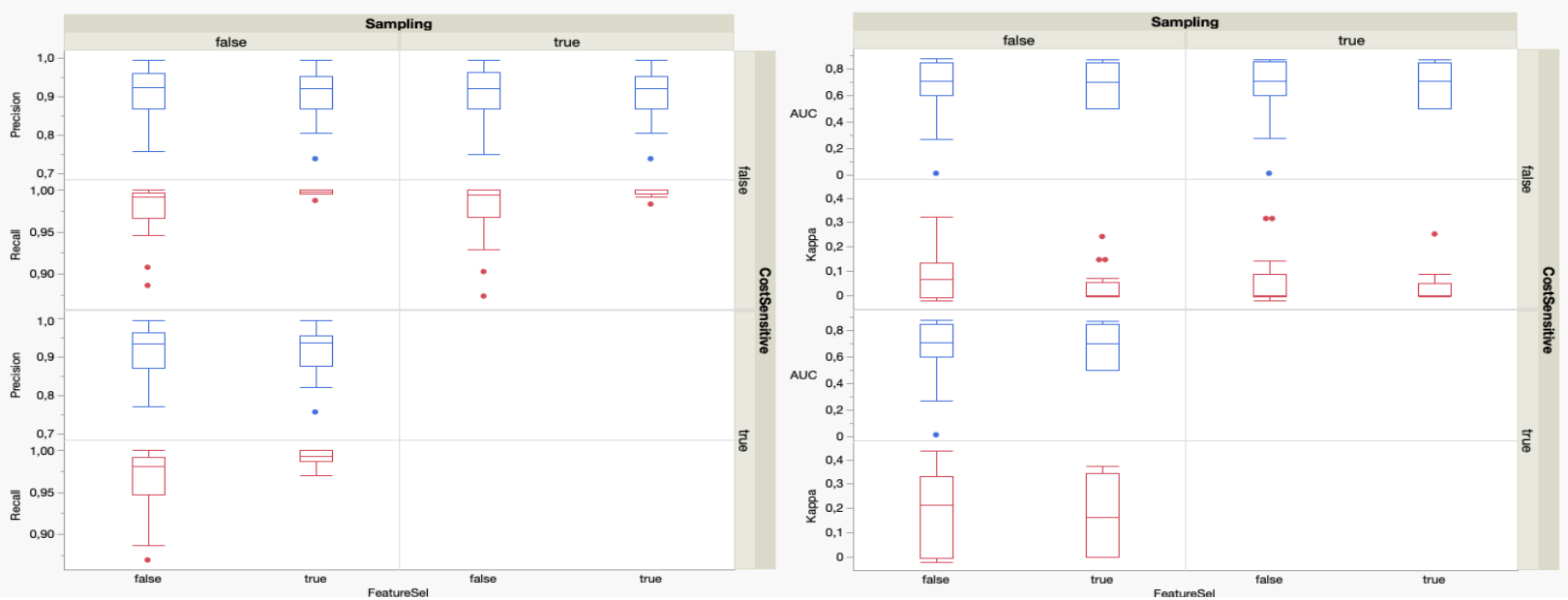


Feature Selection (Best First) + Cost Sensitive (CFN = 10*CFP)

Applicando feature selection e cost sensitive, si nota che il classificatore che da risultati migliori, anche se non eccessivamente, è di nuovo **Naive Bayes**. Peggiora leggermente la Recall, ma migliora nettamente il valore di Kappa.

RISULTATI: ZooKeeper

Dati i risultati precedenti, è possibile affermare che **Naive Bayes** si comporta leggermente meglio rispetto agli altri classificatori. Osservando i grafichi sotto, si può notare come, per tutte e quattro le metriche, si ottengono dei risultati decisamente migliori quando viene applicata la **Feature Selection (Best First).** Non si osservano miglioramenti con l'utilizzo di **Oversampling**. Anche con **Cost Sensitive e Best First**, il classificatore non riporta modifiche rilevanti nei dati, se non un leggero miglioramento del valore di Kappa.



Si può quindi considerare **Naive Bayes** il classificatore che fornisce risultati più accurati, in particolar modo con l'utilizzo di **Feature Selection:** questo può verificare empiricamente il concetto del «**less is more**», eliminando alcune metriche ridondanti tra loro. Si considerano così meno informazioni, ma con validità maggiore.

LINK UTILI

- **GITHUB:**

  https://github.com/matteo-coni/ProgettoISW2_ML

- **SONARCLOUD:**

  https://sonarcloud.io/summary/overall?id=matteo-coni_ProgettoISW2_ML

GRAZIE PER L'ATTENZIONE