MACHINE LEARNING FOR SOFTWARE ENGINEERING

ANALYSIS OF DEFECTS PREDICTION

LUDOVICO DE SANTIS – 0320460

GENERAL INDEX

INTRODUCTION

- Software Testing is a core activity, especially in modern systems, it allows to avoid problems (e.g. bugs), improving the user experience and the success of the project in all areas.
- Testing all areas of the project is really expensive, recent studies focus on what to test.
- Reducing the test cost allows to reduce testing time, save money, having a better time to market and improving productivity in other areas of the project, bringing to a more competitive scenario.
- Basic Idea: use of classifiers to predict class defects, using Machine Learning techniques.

METRICS - SOME DEFINITIONS

- **Recall**

Percentage of how many positives the model guessed with respect to the real positives.

- **Precision**

Percentage of how many positives the model guessed are true positives.

- **AUC**

The ability of the model to correct classify the istances.

- **Kappa**

Behavior of the classifier with respect to a dummy.

GOALS

- The fundamental goals are to analyze and work on performances of classifiers used to predict the defectiveness of classes in open-source projects.
- In details, two projects have been analyzed:

  →Apache **BookKeeper**

  →Apache **Syncope**
- The classifiers used are:

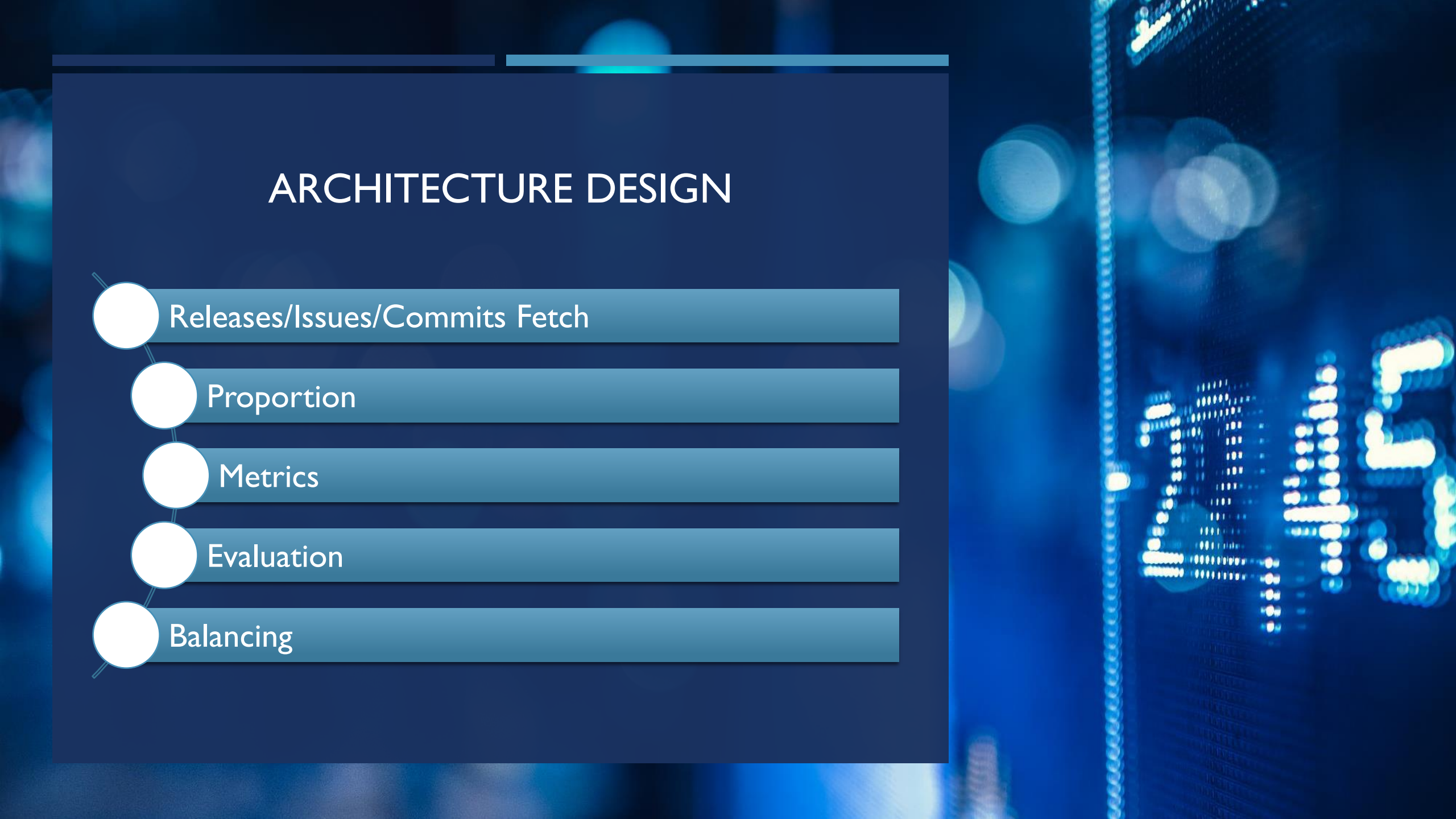
  →**Naïve Bayes**

  →**Ibk**

  →**Random Forest**

ARCHITECTURE DESIGN

- Releases/Issues/Commits Fetch
- Proportion
- Metrics
- Evaluation
- Balancing



RELEASES FETCH

- List of releases of the projects fetched from Jira, using API to obtain the necessary informations, like release names and their release dates.
- Ordering by time.
- The second half of releases have been discarded to limit **snoring**.

ISSUES FETCH

- List of issues of the projects fetched from Jira.

Type == “defect” AND (status == “Closed” OR status ==“Resolved”) AND Resolution ==“Fixed”.

- Extrapolated informations:

→Fix Version

→Opening Version

→Injected Version

PROPORTION

- **First Step**

Issues without Fix Version or inconsistent issues have been discarded.

- **Identification Step**

Identification of Issues without Injected Version.

- **Proportion, First Step**

Proportion with incremental approach to overcome this limit. In particular, in each release k, **P(k)** computed using informations of all issues of previous releases (1….,k-1) with the complete informations, following the alongside formula.

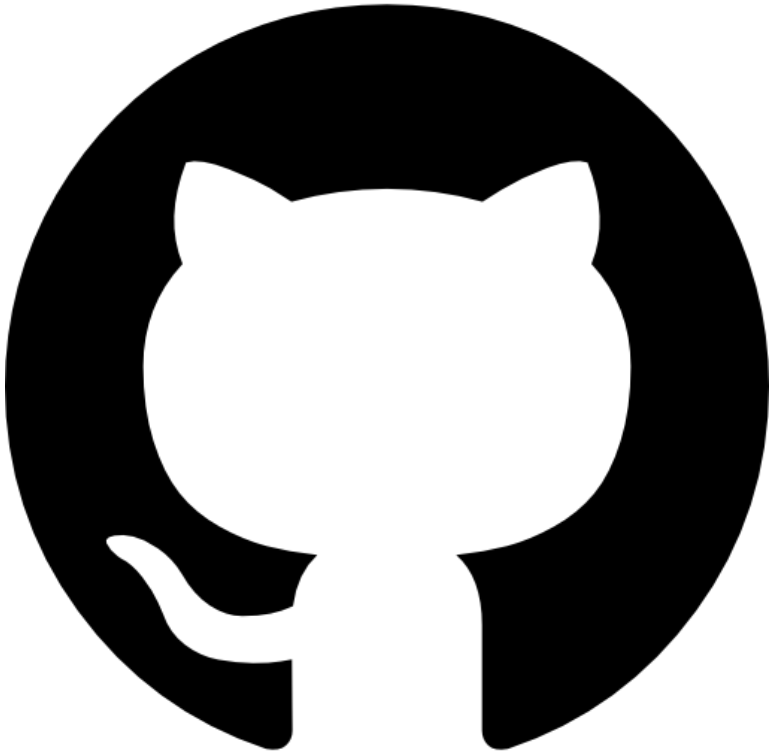
$$\frac{FV - IV}{FV - OV}$$

- **Proportion, Second Step**

Calculation of Injected Version for devoid issues, following the formula: **IV = FV – (FV – OV)*P**, with P as the average of the P(k)s.

COMMITS FETCH

- List of commits and files of the projects fetched from Github to obtain the necessary informations to calculate metrics for each class.
- In each bugfix commit is indicated the relative issue number, and it brings to labeling of defective classes from Injected Version until the previous release of the Fix Version.



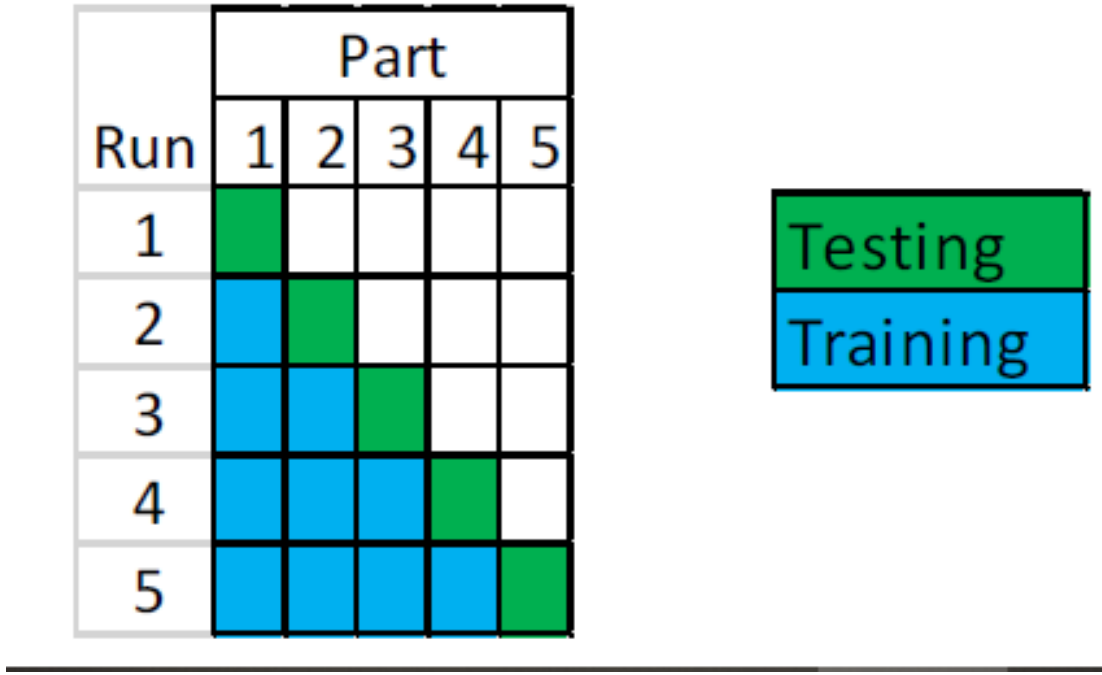
METRICS

Chosen metrics 

- Age
- Number of Revisions
- Number of Bugfix
- LOCs
- LOCs Touched
- LOCs Added
- Churn
- Average Churn
- Authors Number
- Average Change Set

EVALUATION

- The **Walk Forward** technique has been used for evaluation, based on time-series.



BALANCING

- To avoid unbalanced scenarios, the analysis through Weka have been executed analyzing the results of the classifiers in 4 different scenarios:

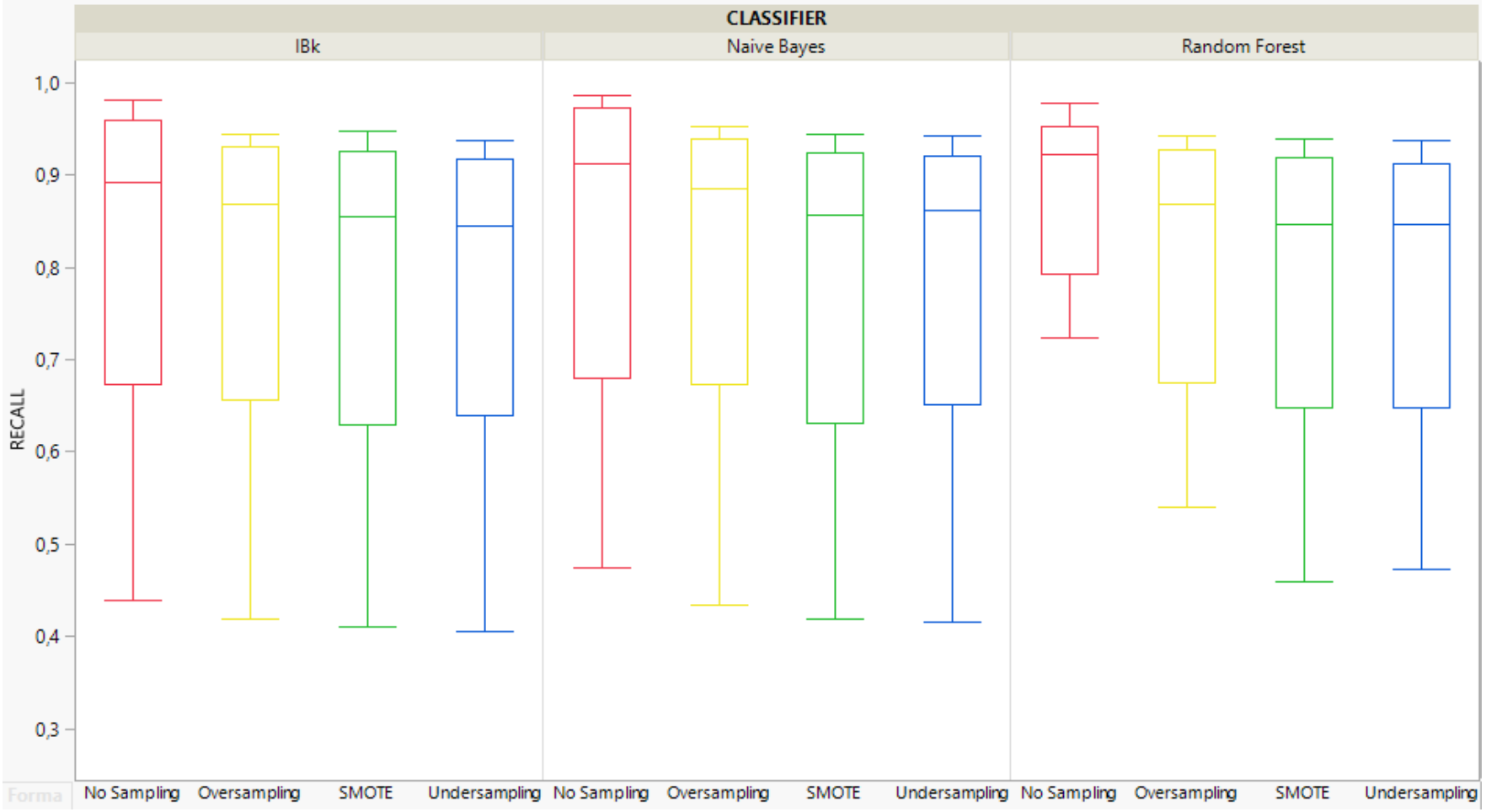
  →**No Sampling**

  →**OverSampling**

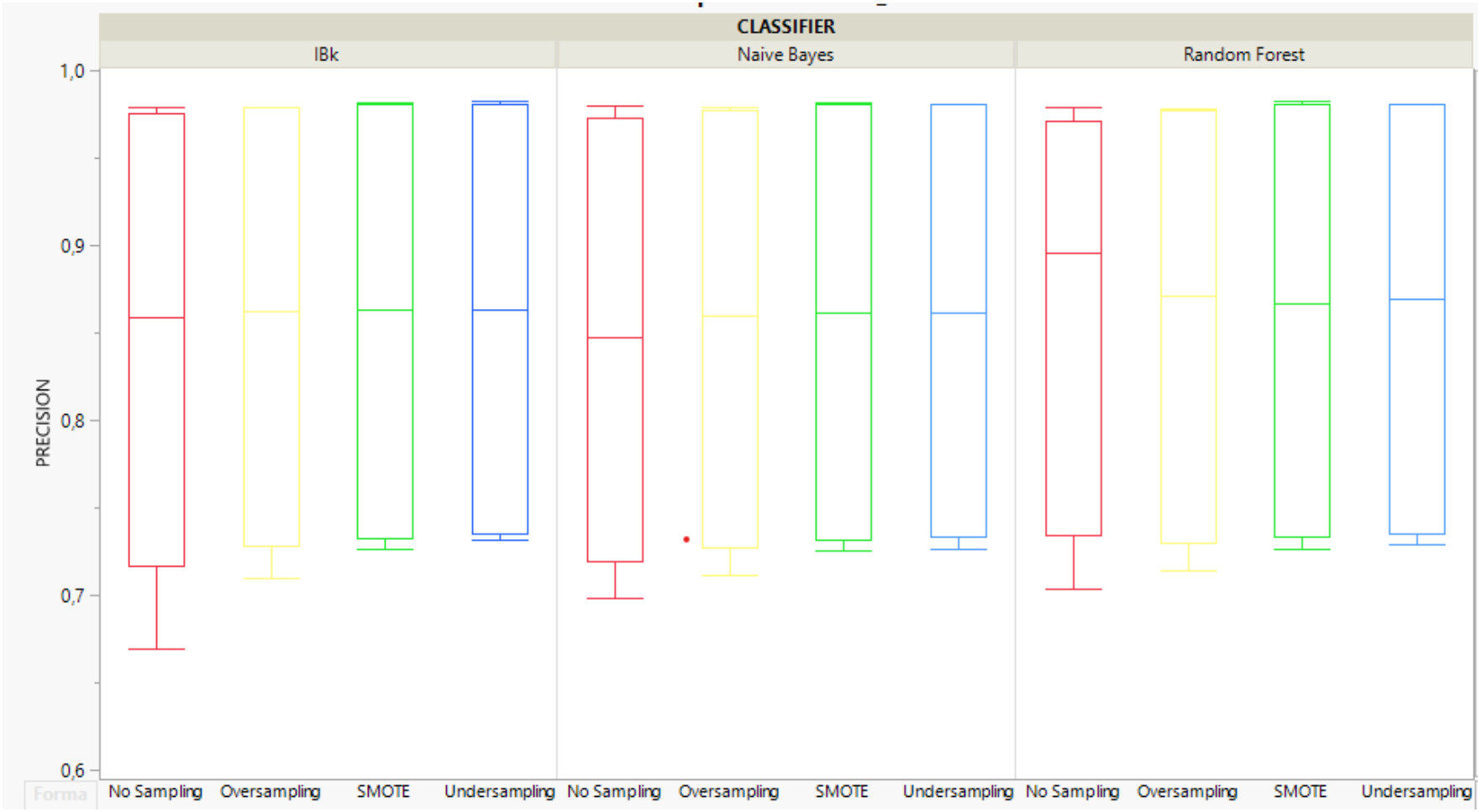
  →**UnderSampling**

  →**SMOTE**

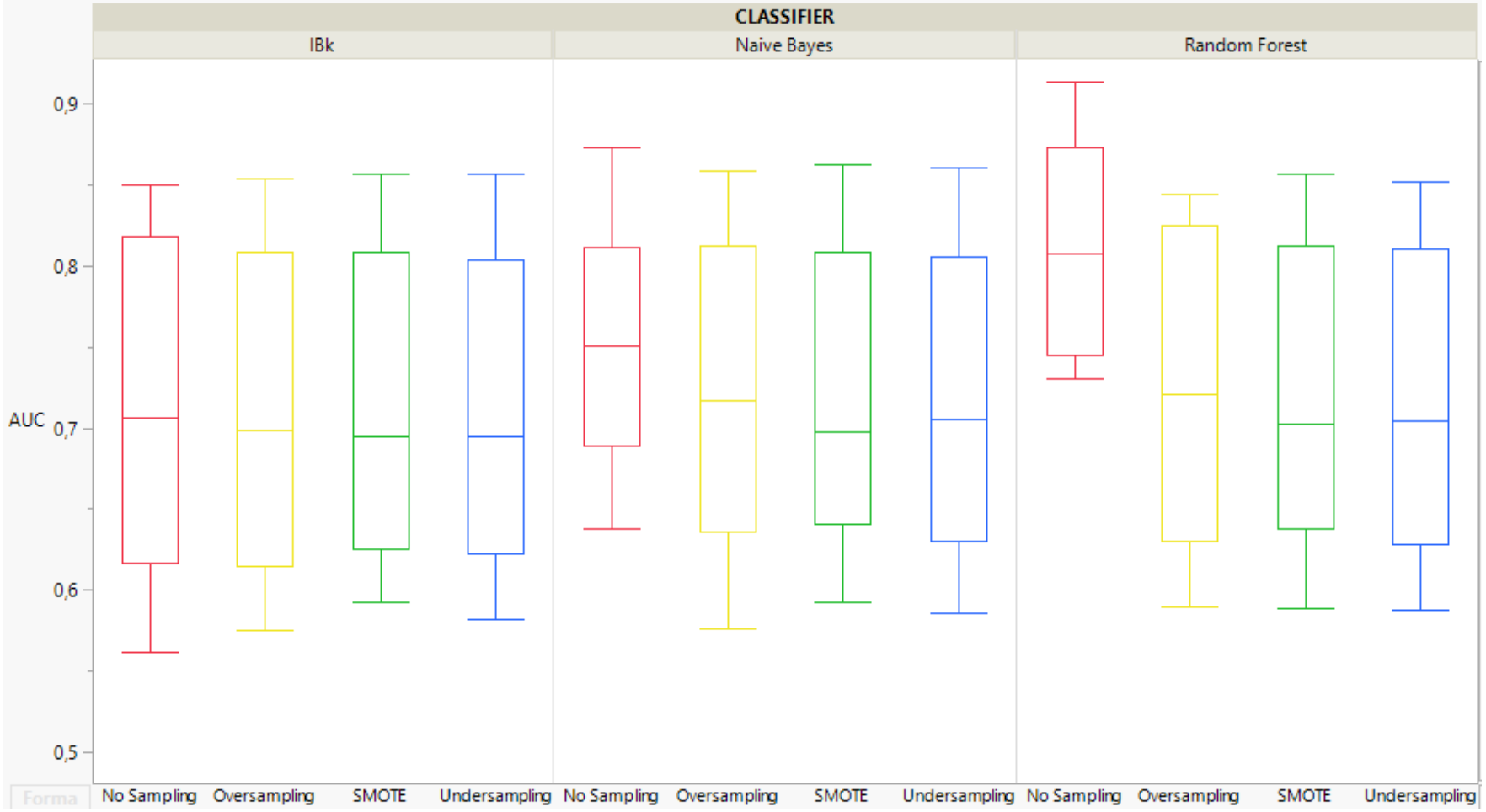
BOOKKEEPER - RECALL



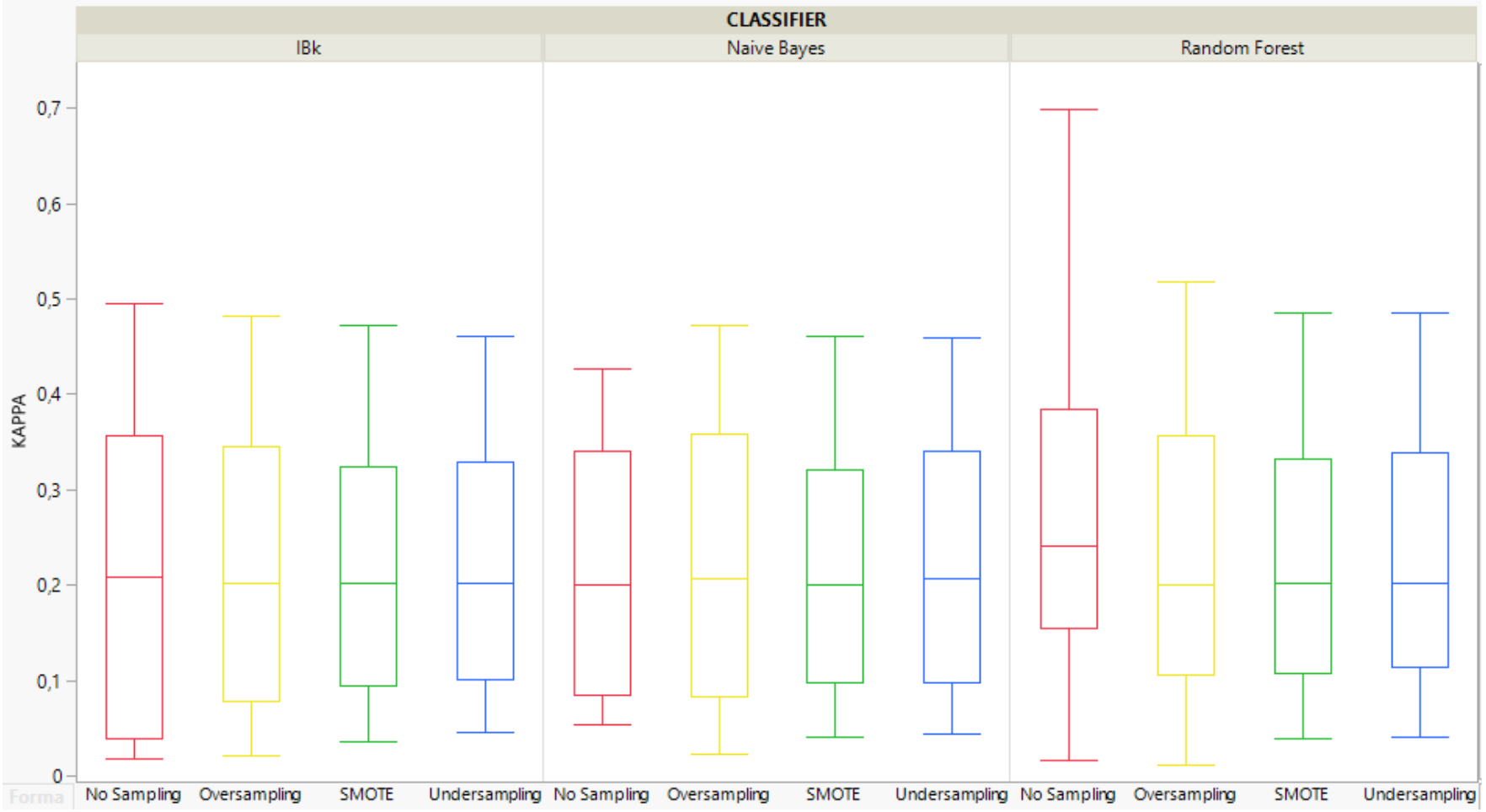
BOOKKEEPER - PRECISION



BOOKKEEPER - AUC



BOOKKEEPER - KAPPA



BOOKKEEPER - COMMENTS

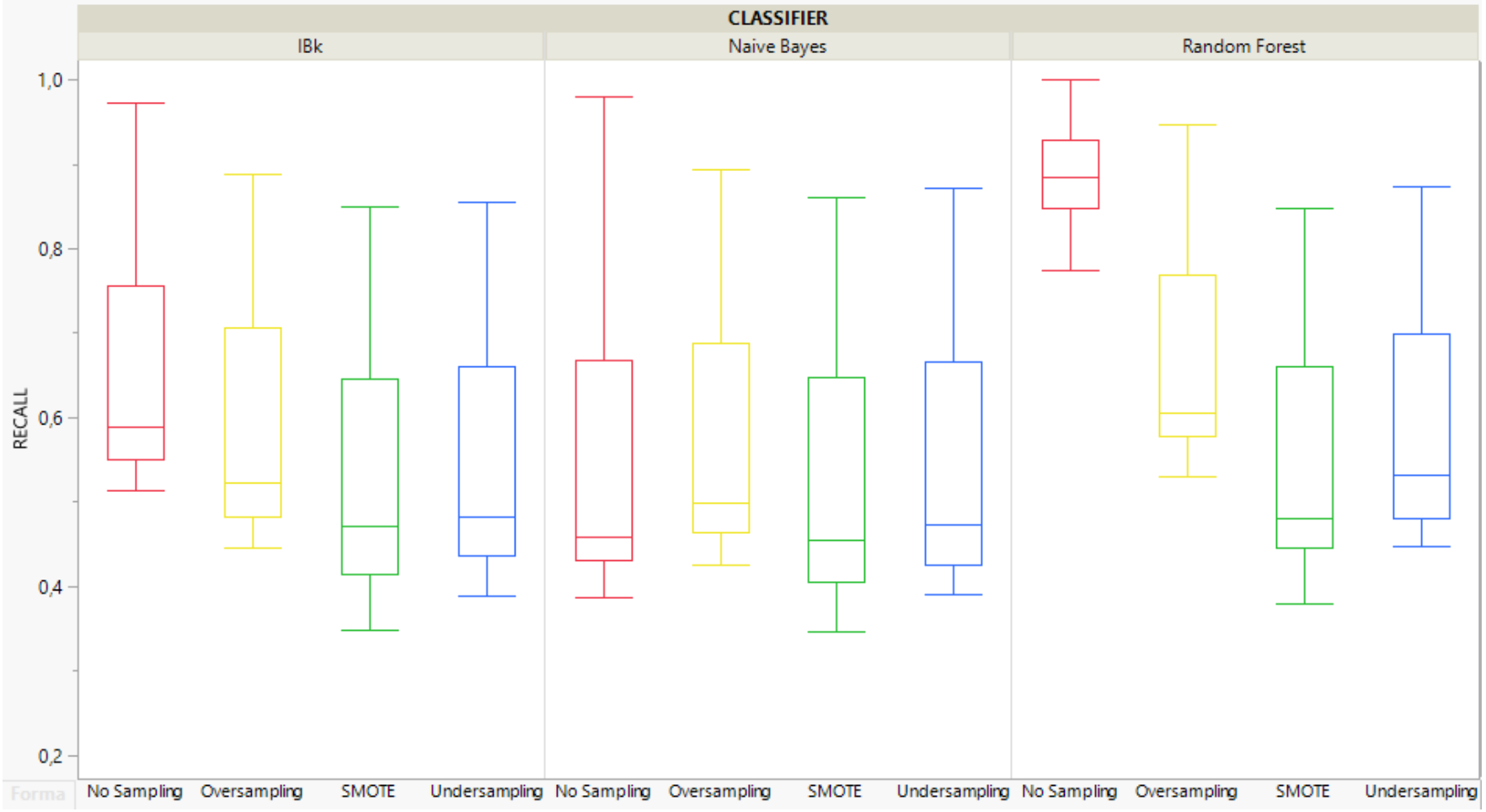
- There aren't major differences between classifiers, their behavior is similar.

- Balancing seems **not** to be the best choice, better performances without it, **why**?

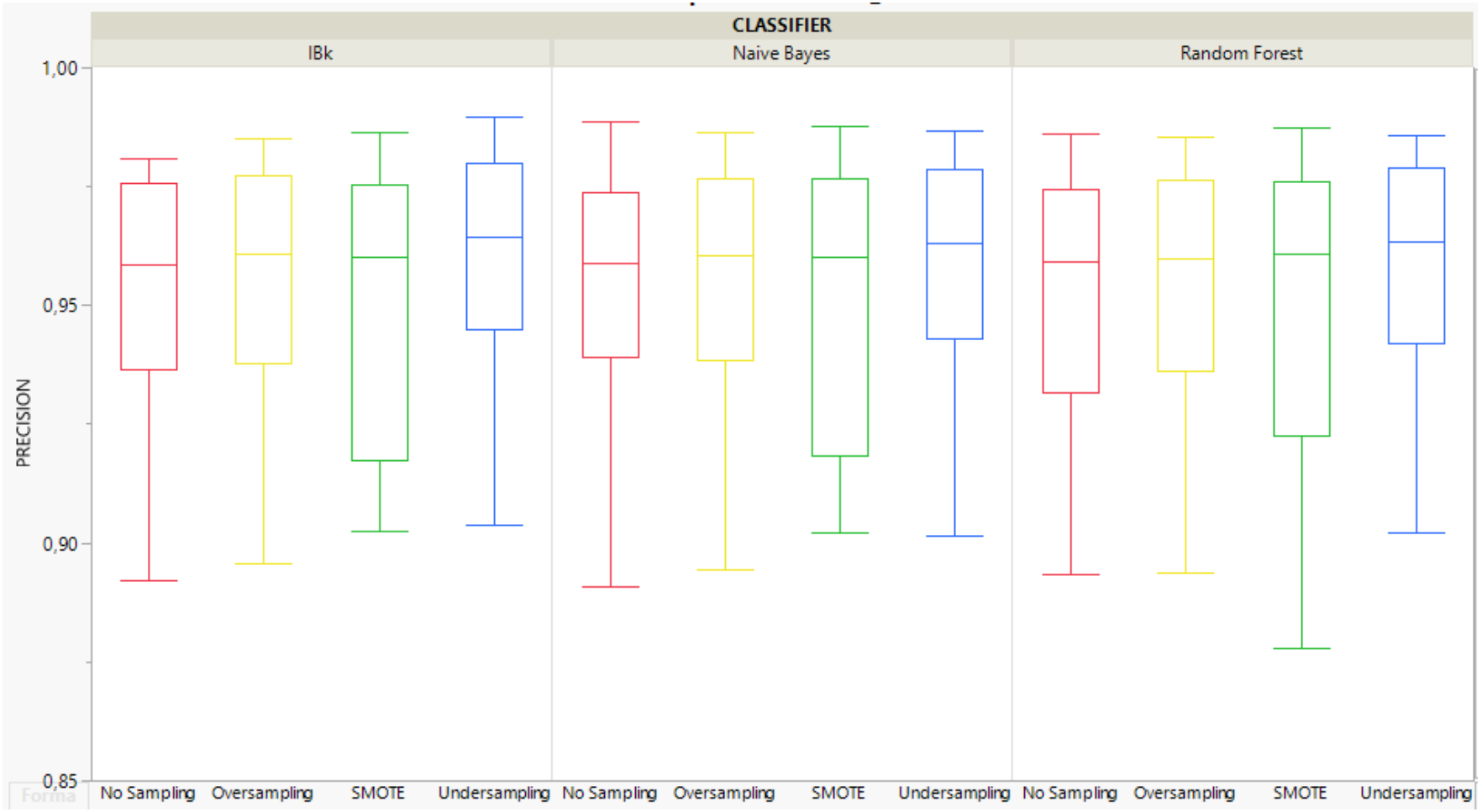
→A possible explanation lies on the heterogeneity of the data., bringing sampling to not help classifiers to correctly indentify classes. Plus, the bookkeeper dataset presents 20% of buggy classes, so it's a bit pre-balanced.

- An Utilization of this technique combined with feature selection and cost sensitive should improve the performances.

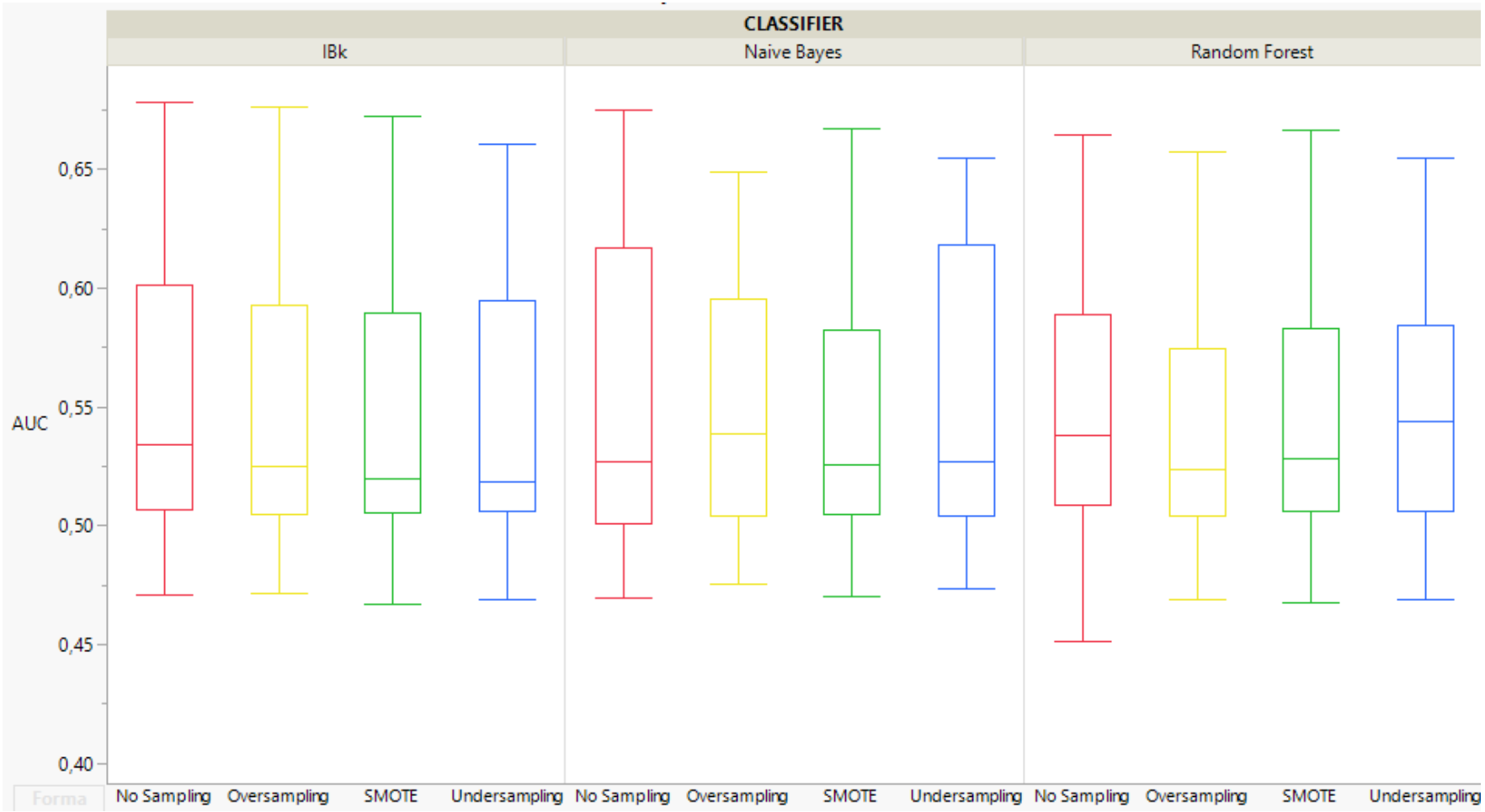
SYNCOPE - RECALL



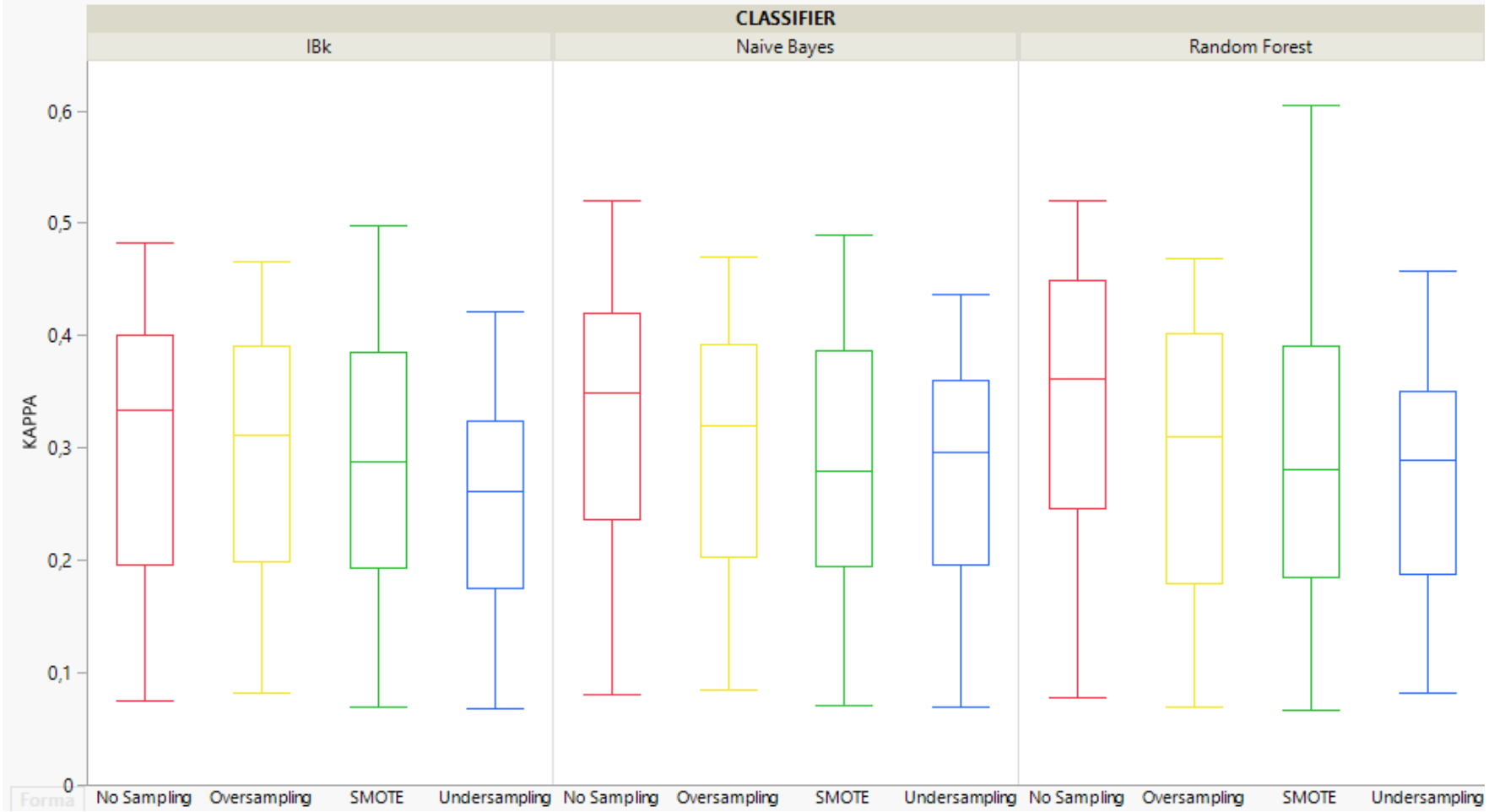
SYNCOPE - PRECISION



SYNCOPE - AUC


CLASSIFIER
IBk
Naive Bayes
Random Forest
AUC
0,65
0,60
0,55
0,50
0,45
0,40
No Sampling
Oversampling
SMOTE
Undersampling
No Sampling
Oversampling
SMOTE
Undersampling
No Sampling
Oversampling
SMOTE
Undersampling

SYNCOPE - KAPPA



SYNCOPE - COMMENTS

- There aren't major differences between classifiers, their behavior is similar
- The performances between scenarios with balancing or scenarios without balancing are quite similar.
- If we compare the three balancing techniques, we can denote better performances for the Oversampling technique. An Utilization of this technique combined with feature selection and cost sensitive should improve the performances.

CONCLUSIONS

- The classifiers present similar performances. The use of Walk Forward technique migth present some unlucky iterations, affecting the performances of the classifiers.

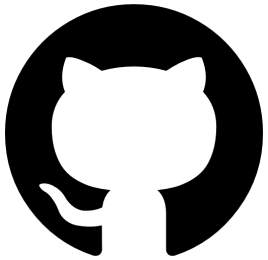
- In an enterprise environment, the use of this techniques can improve the performances of teams, improving the speed of development and the quality of work, bringing to better scenarios from all points of view. Finding and fixing bugs quickly leads us to save money, offer a better product and faster, increasing customer satisfaction, so increasing customer fidelity, which is fundamental in an environment with high rate of competitiveness.
- Time is a limited and highly valuable resource, optimizing it means having an important competitive advantage.

The bad news is time flies. The good news is you're the pilot!

THANK YOU!

Milestone I:

https://github.com/Ludovix9070/Milestone1



Milestone2:

https://github.com/Ludovix9070/Milestone2

Milestone I:

https://sonarcloud.io/summary/overall?id=Ludovix9070_Milestone1

Milestone2:

https://sonarcloud.io/summary/overall?id=Ludovix9070_Milestone2