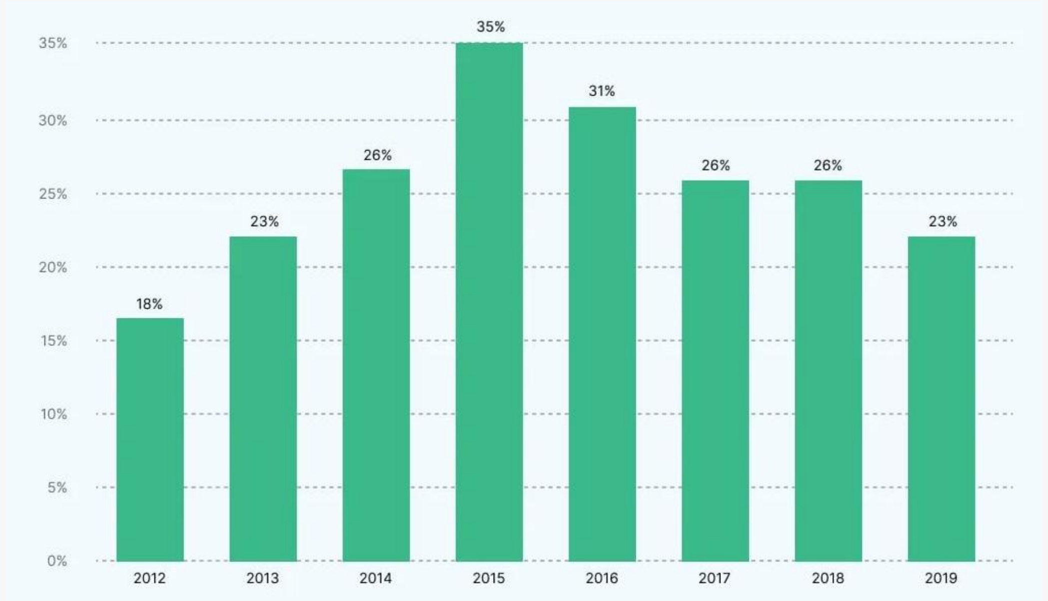
MACHINE LEARNING FOR SOFTWARE ENGINEERING AS TESTING SUPPORT

- Alessandro Finocchi - 0340543
- A.A. 2023/2024

AGENDA

INTRODUCTION



Proportion of budget allocated to quality assurance and testing as a percentage of IT

- Software testing expanses typically account around 18-25% of the project budget.
- Machine learning prediction models may **support** this process predicting classes that are more likely to be buggy.

GOAL

- We want to make the software testing process more **efficient**, ensuring **quality** and **reliability** in the results, at the same time optimizing **resources** and **budget**.
- Given a class, from it's characteristics we want to know if it will be buggy or not.
- Build a machine learning model trained with class characteristics that can answer this demand.
- Exploiting such a tool, the testing team can **focus** more precisely on those classes predicted to be buggy and test them more intensely.
- Use different models and find **the optimal configuration**.
- Empirical study on 2 open source project: **AVRO** and **BOOKKEEPER**.

ASSUMPTIONS

- **The past is similiar to the future**: the software development behavior will follow patterns that the machine learning model will be able to learn, without considering discontinuities.

- **Linear development model**: the developement process doesn’t carry out more version alongside each other.

- **Jira completeness**: the Jira tracking system comprises all the information about the project involved in the study, github will be used only for retrieving the repository.

SCRAPING

Jira → Issue Tracking System to retrieve releases and tickets

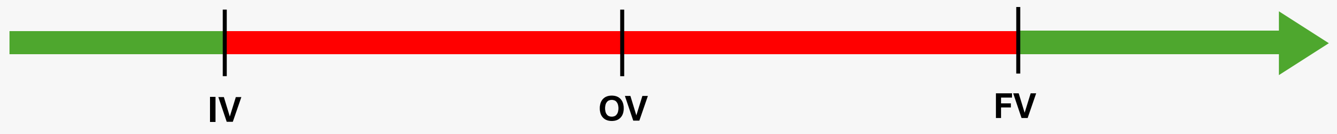
GitHub → Version Control System to store the repository and keep track of all commits that touch classes

$$\begin{tikzcd}\boxed{\text{Releases}} \arrow[r, dash, "1:m"] & \boxed{\text{Tickets}} \arrow[r, dash, "1:n"] & \boxed{\text{Commits}} \arrow[r, dash, "n:m"] & \boxed{\text{Classes}}\end{tikzcd}$$

METHODOLOGY

DEFECTS

- The idea starts from the **lifecycle** of a defect



- At the beginning the defect doesn't exist.
- At a certain version, not alweys known, the defect is introduced: we will call such version the **Injected Version** (**IV**).
- The defect is not discovered until the **Opening Version** (**OV**), the version when the corresponding ticket was created.
- The **Fixed Version** (**FV**) is the first release without the defect.

PROPORTION

- Sometimes it’s not known when a defect is actually introduced
- The **Proportion** methods find an approximation of the IV for a bug
- In this project we used a brand new approach, based on calculating the proportion value with all the tickets considered known until the current version

$$P = E_{t \in T_i}\left[\frac{t(FV) - t(IV)}{t(FV) - t(OV)}\right]$$

where T_i are all the tickets with an injected version until the current version.

- For the tickets without an injected version, it will be computed as

$$t(IV) = t(FV) - \big(t(FV) - t(OV)\big)\, P \qquad \forall t \in \overline{T_i}$$

where $\overline{T_i}$ are all the tickets without an injected version until the current version.

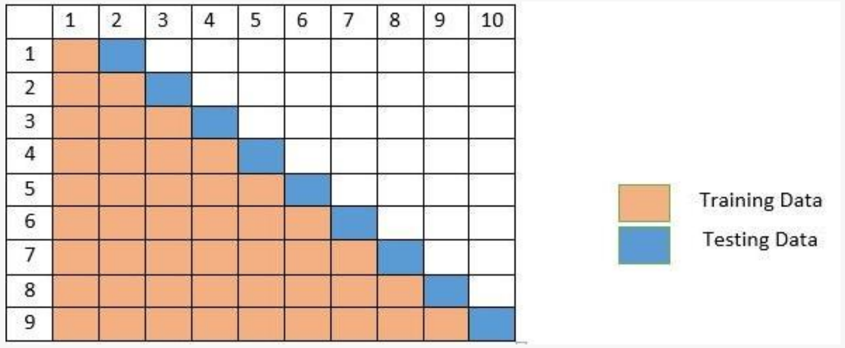
- This approach is the most non-conservative one because it uses all the information in our possesion (different from Incremental approach)

METRICS

Metrics	Description
Size	The lines of code of the class.
LOC_added	The lines of code added.
LOC_removed	The lines of code removed.
Churn	The difference between added and removed LOC.
N_revisions	Number of revisions.
N_defect_fixes	Number of defect fixes.
N_authors	Number of authors that worked on that class.
CBO	Coupling between objects: counts the number of dependencies that a class owns.
Fan-in	Counts the number of input dependencies a class has.
Fan-out	Counts the number of output dependencies a class has.
Public_methods_qty	Counts the number of public methods.
Is_buggy	The **label** to predict.

TRAINING

- The ML models we are going to use are **Supervised Learning** classificators: they need a dataset with the label to predict specified.
- The dataset needs to be splitted in a **training set** for learning and **testing set** for evaluating the performances of the model.
- Since our data is time-sensitive, we need to take into account its **temporal order**, thus we use the **walk-forward** strategy which is a time-series validation technique.
- To minimize snoring, only the first half releases are used in the dataset construction.

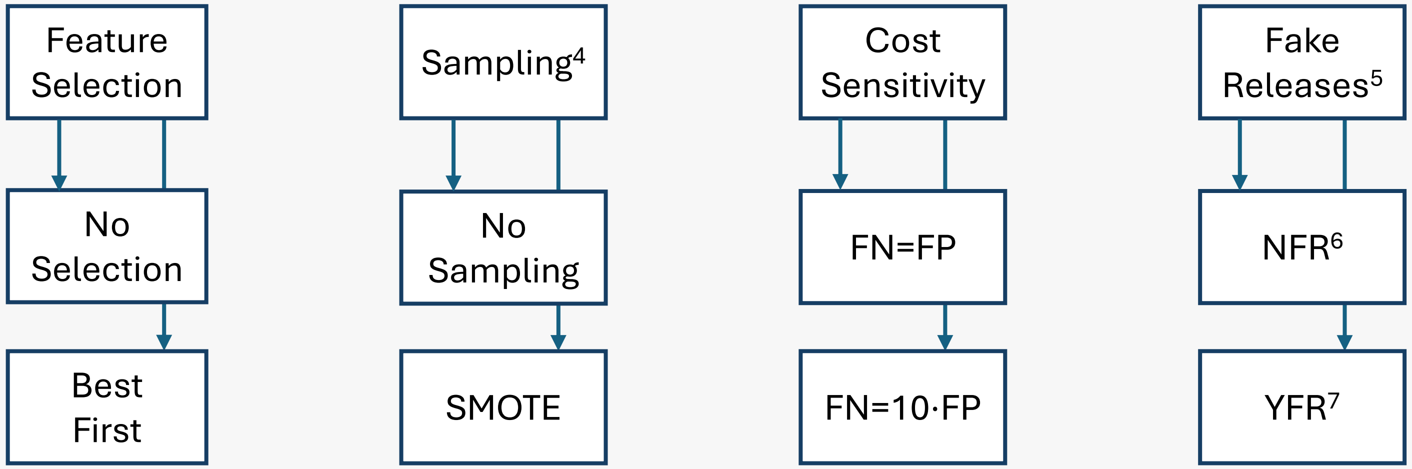


	1	2	3	4	5	6	7	8	9	10
1	🟧	🟦								
2	🟧	🟧	🟦							
3	🟧	🟧	🟧	🟦						
4	🟧	🟧	🟧	🟧	🟦					
5	🟧	🟧	🟧	🟧	🟧	🟦				
6	🟧	🟧	🟧	🟧	🟧	🟧	🟦			
7	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟦		
8	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟦	
9	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟦

🟧 Training Data
🟦 Testing Data

TRAINING

- For training models, different strategies have been taken into accounts:
  - **Feature Selection**: classifiers focus on attributes with the greater IG[1]
  - **Sampling:** permits to balance the instance type occurrences in the datasets
  - **Cost Sensitivity**: define what type of error is more important[2]
  - **Fake Releases**: a brand new strategy for the problem considered[3]



METHODOLOGY

- Scraping
- Defects
- Proportion
- Metrics
- Training
- Classifiers

1. Information Gain
2. We prefer not having FNs
3. Consider commits that are no part of any release to use more information
4. Studied only for Avro
5. Studied only for Bookkeeper
6. No Fake Releases
7. Yes Fake Releases

CLASSIFIERS

- Classifiers used in the process are

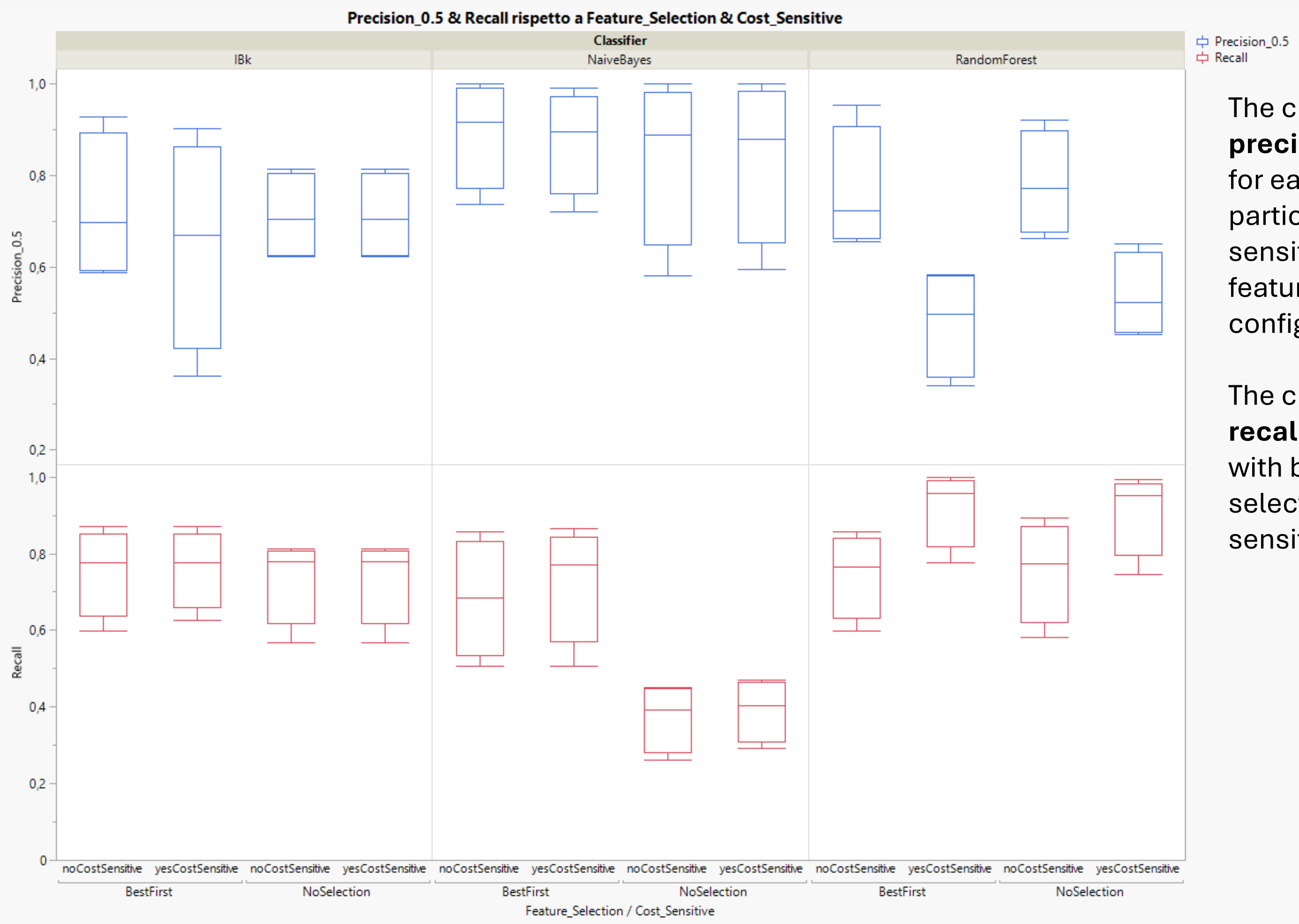
Naive Bayes	Random Forest	IBk

- They will be evaluated using the following metrics:

  - **Precision** = $\frac{TP}{TP+FP}$
  - **Recall** = $\frac{TP}{TP+FN}$
  - **NPofb20**: percentage of bugs that a developer can identify by inspecting the top 20% of lines of code, computed with the ACUME tool.
  - **AUC**: the probability that the model, given a randomly chosen positive and negative example, will rank the positive higher than the negative.

- To explore the results the **JMP** suite of computer programs for statistical analysis and machine learning has been used

PRECISION & RECALL



The classifier with highest **precision** is **Naïve Bayes** for each configuration, particularly the no cost sensitive with best first feature selection configuration.

The classifier with highest **recall** is **Random Forest**, with best first feature selection and with cost sensitivity.

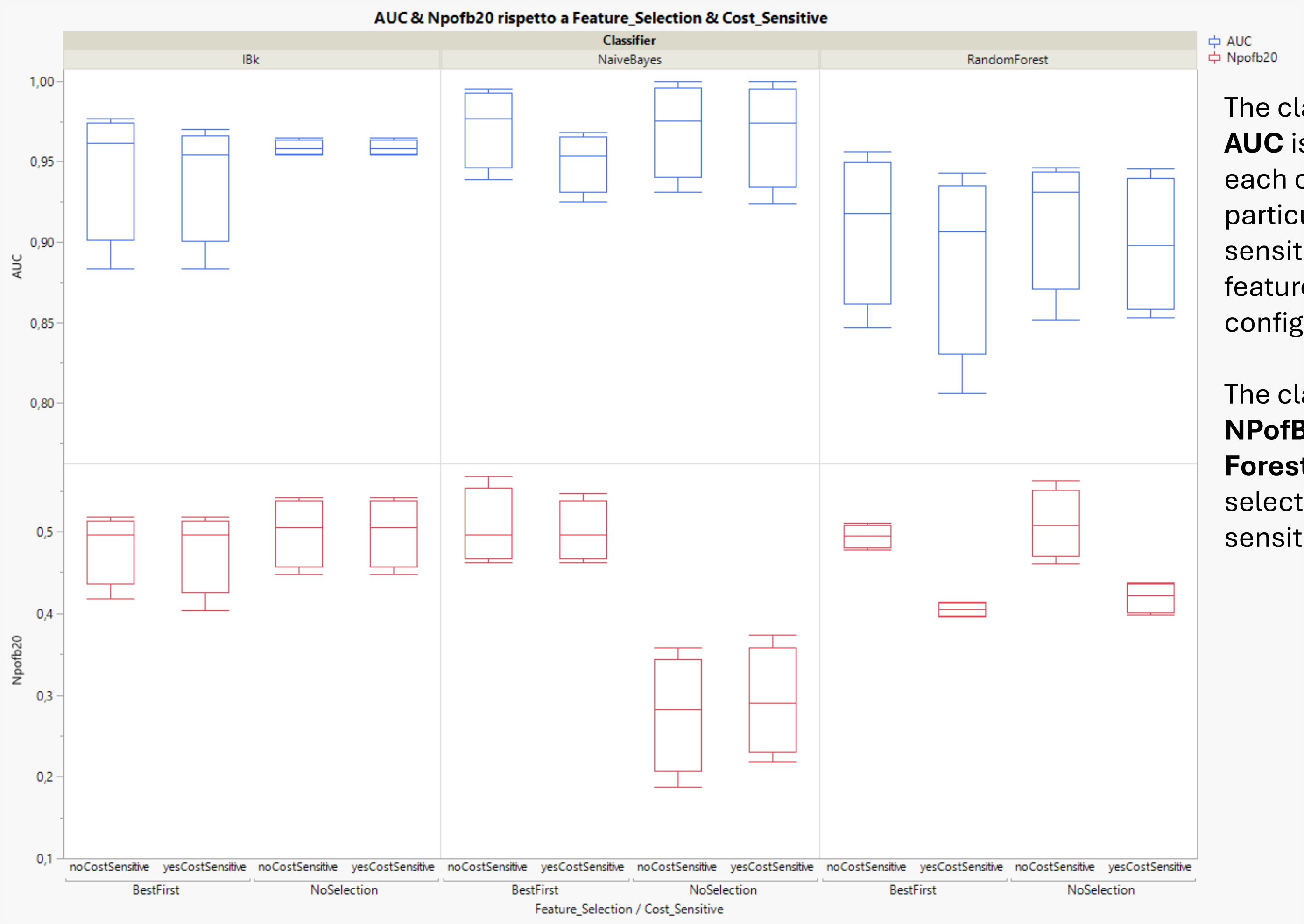
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

AUC & NPOFB20



The classifier with highest **AUC** is **Naïve Bayes** for each configuration, particularly the no cost sensitive with best first feature selection configuration.

The classifier with highest **NPofB20** is **Random Forest**, without feature selection and without cost sensitivity.

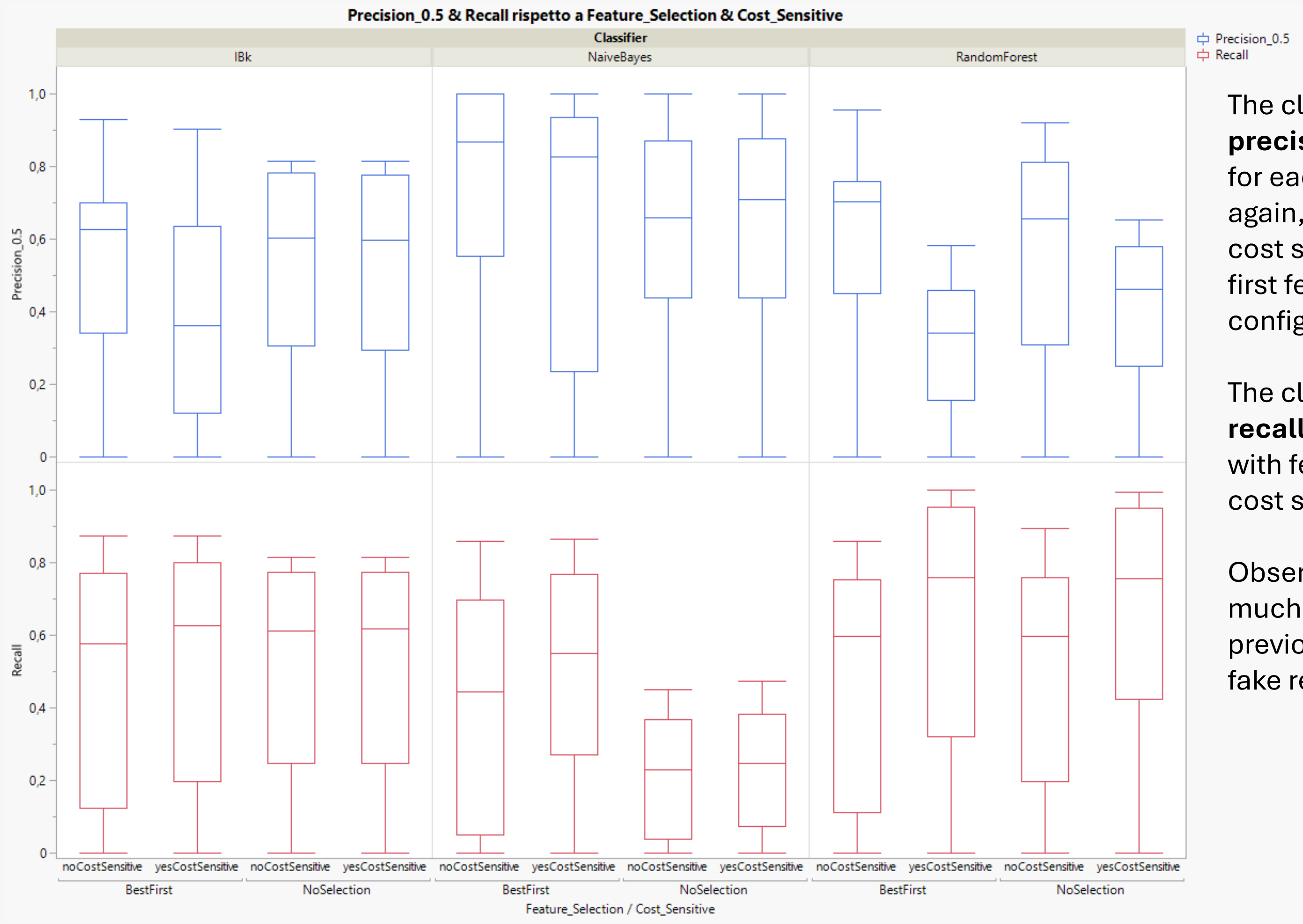
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

PRECISION & RECALL



The classifier with highest **precision** is **Naïve Bayes** for each configuration again, particularly the no cost sensitive and best first feature selection configuration.

The classifier with highest **recall** is **Random Forest**, with feature selection and cost sensitivity again.

Observe that dispersion is much higher than in the previous case, without fake releases.

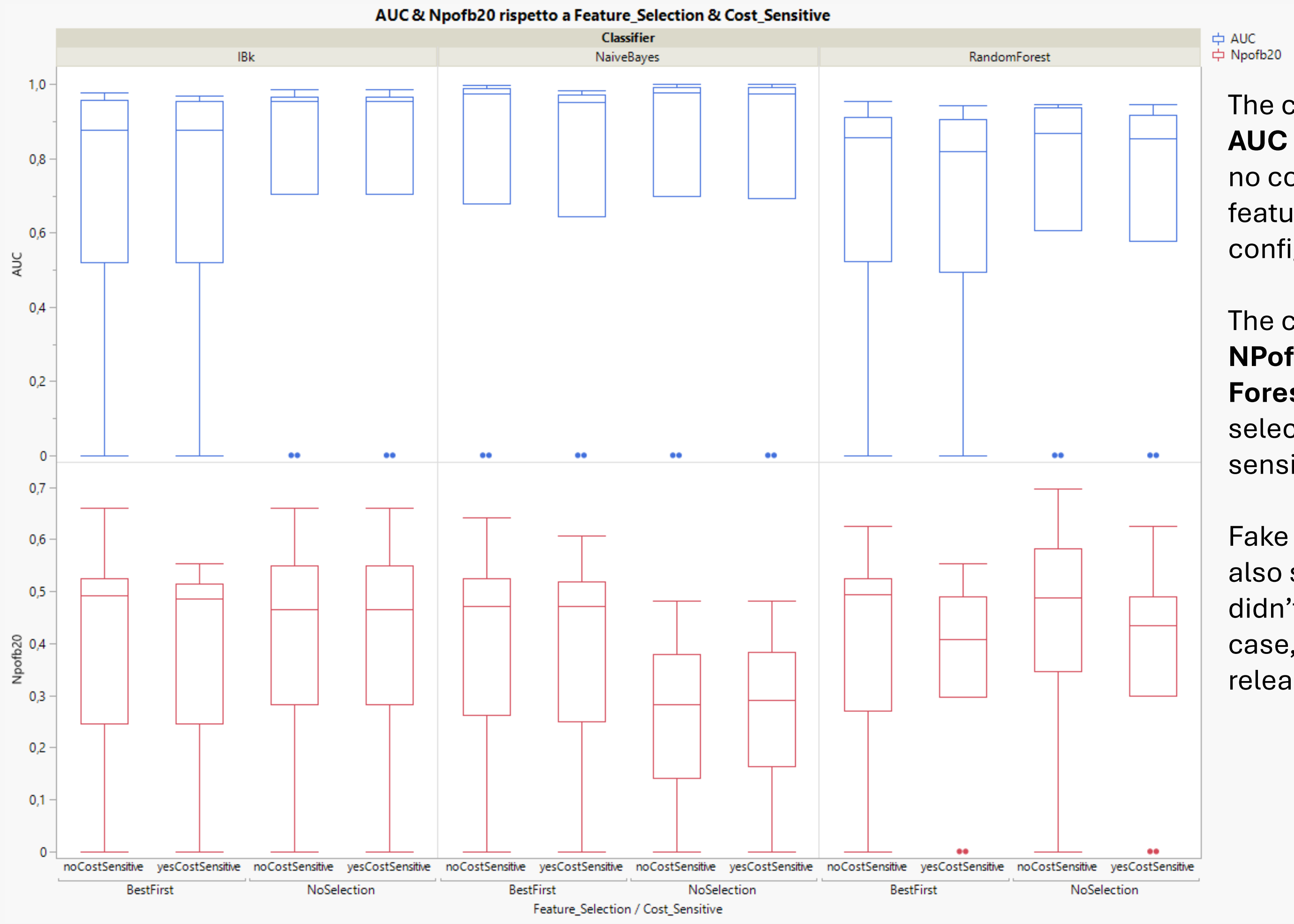
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

AUC & NPOFB20



The classifier with highest **AUC** is **Naïve Bayes** with no cost sensitive and no feature selection configuration.

The classifier with highest **NPofB20** is **Random Forest**, without feature selection and cost sensitivity.

Fake releases introduces also some outliers[1], which didn't exist in the previous case, without fake releases.

BOOKKEEPER RESULTS

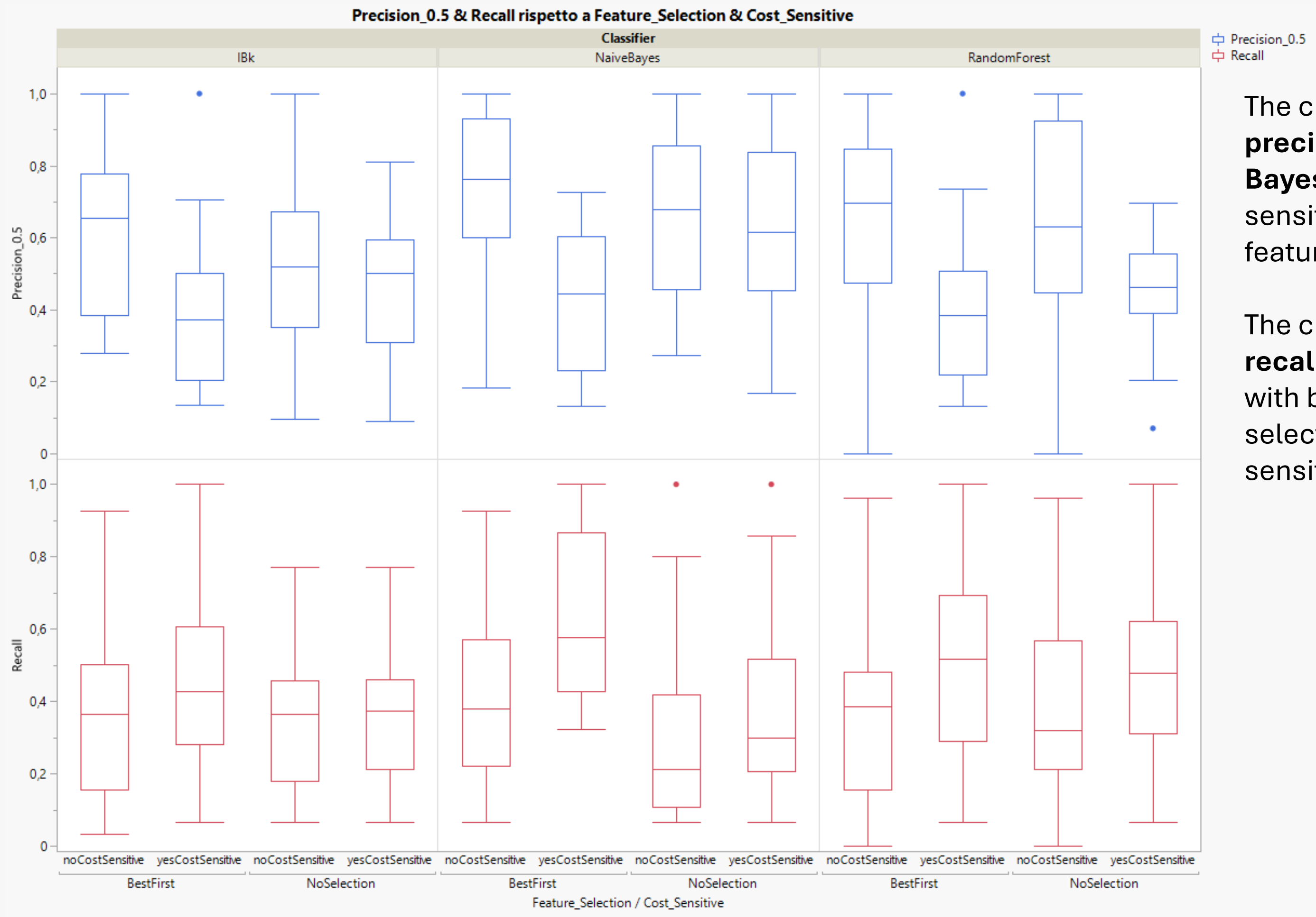
- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

1. A data point in a graph or a set of results which is very bigger of smaller than the next nearest data point

PRECISION & RECALL



The classifier with highest **precision** is still **Naïve Bayes** with no cost sensitivity and best first feature selection.

The classifier with highest **recall** is **Naïve Bayes**, with both best first feature selection and cost sensitivity.

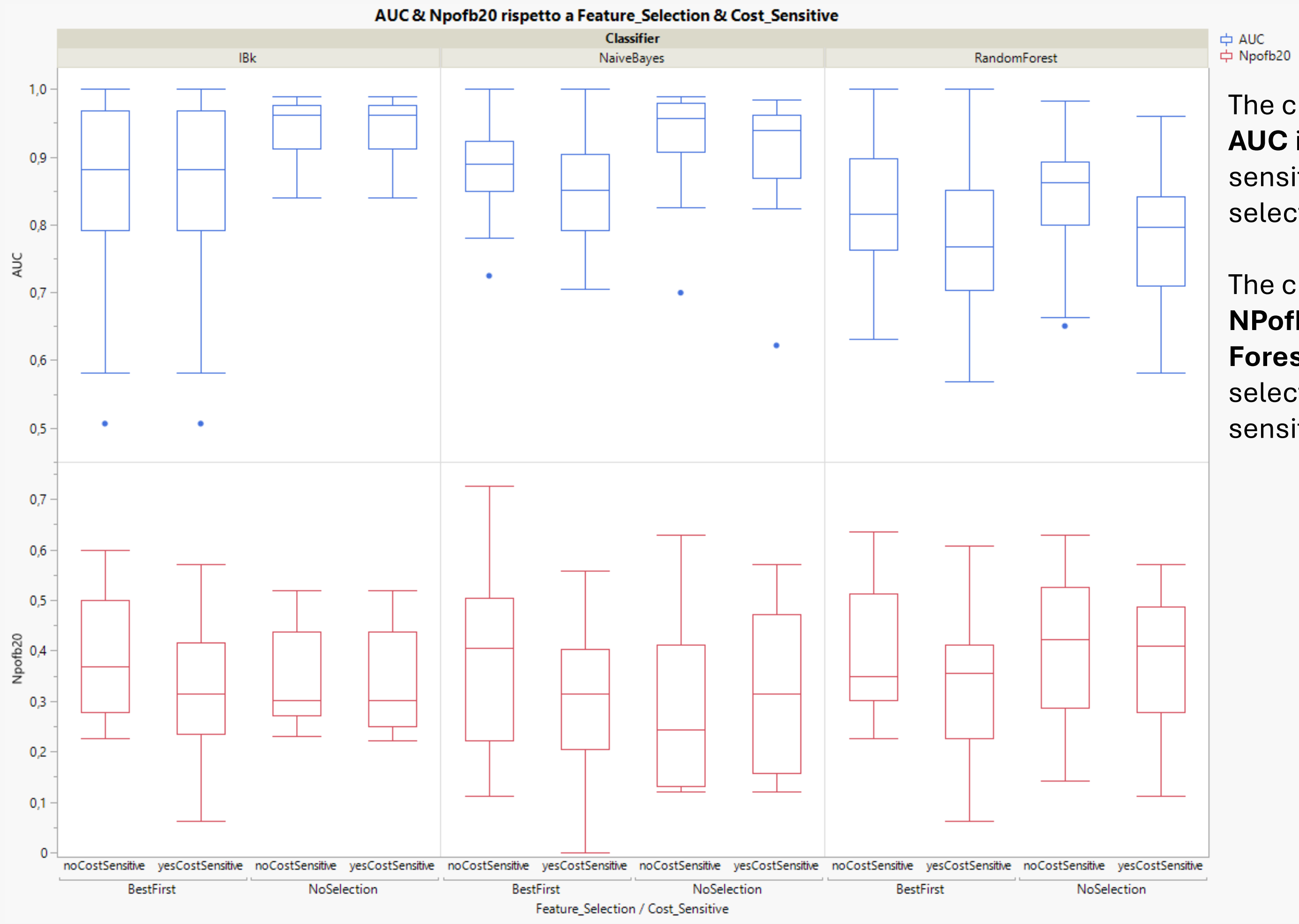
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

AUC & NPOFB20



The classifier with highest **AUC** is **IBk** with no cost sensitivity and no feature selection.

The classifier with highest **NPofB20** is **Random Forest**, without feature selection and cost sensitivity.

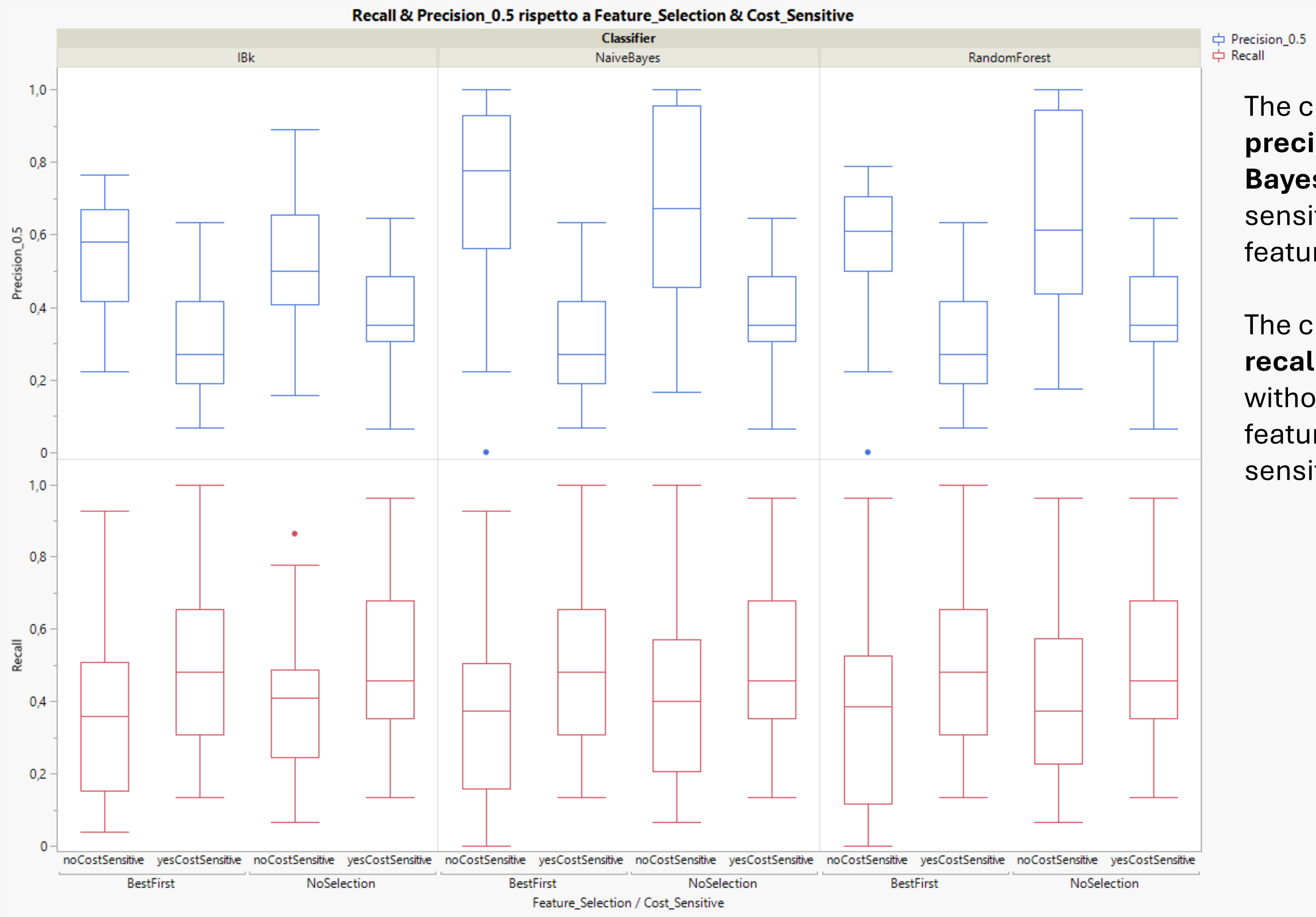
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

PRECISION & RECALL



The classifier with highest **precision** is still **Naïve Bayes** with no cost sensitivity and best first feature selection.

The classifier with highest **recall** is **Random Forest**, without both best first feature selection and cost sensitivity.

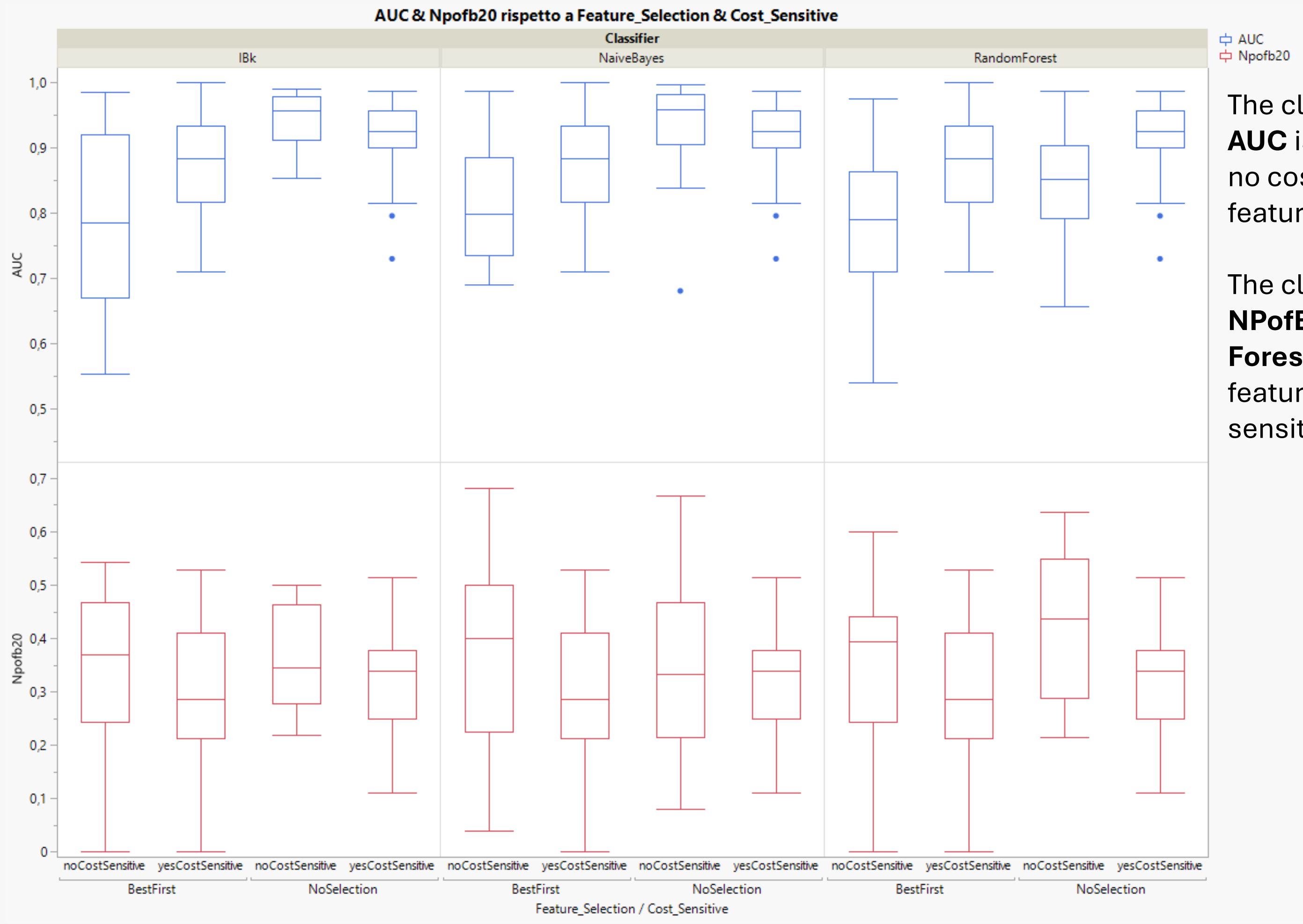
BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

AUC & NPOFB20



The classifier with highest **AUC** is **Naïve Bayes** with no cost sensitivity and no feature selection.

The classifier with highest **NPofB20** is **Random Forest**, without both feature selection and cost sensitivity.

BOOKKEEPER RESULTS

- No Fake Releases
- Yes Fake Releases

AVRO RESULTS

- No Sampling
- SMOTE

CONCLUSIONS

Proj.	Feature Selection	Cost Sensitivity
Bookkeeper	It has always allowed for achieving higher recalls, almost always higher precision, but it shows fluctuating results about AUC and NPofB20.	It has never improved precision but has always increased the recall, while causing slight degradations in both AUC and NPofB20.
Avro	It always yields the highest results for precision and recall, even though each one for different cost sensitivity configurations, it consistently decreases AUC and shows fluctuating results for NPofB20.	It has never improved precision but has always achieved higher recall; moreover, it tends to slightly decrease both NPofB20 and AUC.

CONCLUSIONS

Fake Releases in Bookkeeper	Sampling in Avro
While precision, recall and NPofB20 alweys decreased using fake releases, AUC had some improvements thanks to this strategy, but it made interquantile range between the 25° percentile and 75° percentile increase a lot, impling an increase in the dispersion of data too, so results are less reliable, and the presence of outlier streghtened this thesis, so in general this is not considered a reliable stratgy.	Tends to decrease precision spread, but it doesn't give a general improvement to its performances, while the recall is almost alweys decreased, thus also AUC gets fluctuating changes, but it has a good effect on NPofB20, which tends to increase even though spread gets a bit higher.

CONCLUSIONS

- **Recall** is the most important performance metric in this study.
  - ➢ We want to make sure that **bugs are found**.

- **NPofB20** can help deciding if the **effort** required to examine code is out of budget, optimizing the decision-making for the prioritization of activities.
  - ➢ Crucial when resource allocation is as important as prediction accuracy

- **AUC** helps understanding the model **ranking quality**[1], the higher it is, the higher are the probabilities assigned to positive instances.
  - ➢ It is effective for **comparing** different models since it evalutes performances over all thresholds.

- **Feature selection** process is good because allows getting higher precision and recall, but care if the ranking quality characteristic becomes significant.
- **Cost-sensitive** classifiers is good for achieving higher recall, but caution is needed when using them since they negatively impact other metrics.
- **SMOTE** balancing can be taken into account in contexts where the effort management is key.
- **Fake releases** strategy is definitely not reliable.

1. How well it separates positive and negative classes.

THREATS TO VALIDITY

- **Construct Validity**: the accuracy in translating theoretical concepts to concrete measurements
  - the mapping between tickets and commit can mislead the labeling of classes since it may involve biases

- **Internal Validity**: the degree to which one can be confident that the cause-effect relationship established in a study cannot be explained by other factors
  - Chosen metrics may not represent properly the buggyness of a class, other factors like team practices and expertise might play a role

- **External Validity**: the extent to which one can generalize the findinigs of a study to other situations, people, settings and measures
  - Analyzed repositories may not generalize the code base for other software projects, or it may overrepresent some types of projects

- **Conclusion Validity**: it refers to the correctness of supported conclusions
  - Performance metrics considered could not be the adequate in other contexts, or the datasets could be too small to be statistically significant

- **Reliability**: it refers to the consistency and repeatability of the measures
  - Results depend on certain tools, like Jira, GitHub, the complexity metrics extractor, and eventual errors of theirs can affect their reproducibility

CONCLUSIONS

THANKS FOR YOUR ATTENTION!

Project links:

Pagina gIthub

Pagina sonarcloud