Deliverable 2
ML for SE

- Jacopo Fabi 0293870

Introduzione

- Per la predizione della difettosità della classi durante il processo di sviluppo software, si utilizza il machine learning
  - Si vuole prevenire l'avvento di difetti, concentrandosi su quelle classi che più probabilmente saranno buggy
  - Si effettua la predizioni tramite *classificatori*, che sfruttano dati presenti e passati per predirre dati futuri

- L'obiettivo del progetto è quello di effettuare un'analisi finalizzata a misurare l'efficacia di diverse tecniche di *Feature Selection*, *Balancing* e *Cost Sensitive*, in relazione all'accuratezza di diversi classificatori per la predizione della buggyness delle classi nei seguenti progetti *Apache*:
  1. Bookkeeper
  2. Avro

Progettazione

- È stato realizzato un programma Java per valutare l'accuratezza dei classificatori utilizzati
- L'applicativo è strutturato in due fasi: costruzione e analisi del dataset

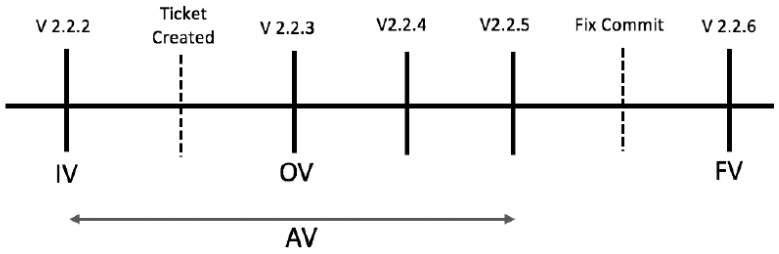
- Costruzione dataset
  - Si recuperano i dati necessari da Jira e GitHub
  - Si effettua una sanificazione dei dati per avere un dataset il più corretto possibile
  - Si genera il dataset da fornire in input all'algoritmo di machine learning

- Analisi dataset
  - Split in training e testing set per l'esecuzione di *Walk Forward*
  - Si applicano le tecniche di *feature selection*, *balancing* e *cost sensitive*
  - Si valuta l'accuratezza dei diversi classificatori utilizzati

Progettazione – Jira

- Il programma realizzato, per ogni release del progetto, recupera tutte le classi a loro appartenenti, ne calcola le metriche e ne determina la buggyness.

- Si utilizza *Jira* per identificare lo storico dei difetti delle varie classi, recuperando tutti i ticket relativi a *Bug Fix*, sfruttando le API offerte dalla piattaforma stessa.
- Ogni ticket restituisce le seguenti informazioni:
  - *Injected Version*
  - *Fixed Versions*



Recuperiamo l'**OV** dalla data di creazione del ticket.

Potrebbe verificarsi, inoltre, che **FV** e **IV** non sono presenti, casi che analizzeremo successivamente.

Progettazione – Git

- Si utilizza *GitHub* per ottenere i dati della repository relativa al progetto in esame, sfruttando la libreria JGit
  - Tramite *clone()* oppure *checkout()* si effettua una copia locale della repository
  - Si ottiene la lista di tutte le release presenti su GitHub
  - Si ottiene la lista di tutti i commit per ogni release

```
tagList = git.tagList().call();
RevWalk walk = new RevWalk(this.git.getRepository());

for (Ref tag : tagList) {
    String tagName = tag.getName();
    String releaseName = tagName.substring((releaseFilter + Parameters.TAG_FORMAT).length());
    ...
    RevCommit c = walk.parseCommit(tag.getObjectId());
    Date releaseDate = DateHandler.getDateFromEpoch(c.getCommitTime() * 1000L);
    String tagName = tag.getName();
    String releaseName = tagName.substring((releaseFilter + Parameters.TAG_FORMAT).length());
    GitRelease release = new GitRelease(this.git, c, releaseName, releaseDate);
    ...
```

```
LogCommand logCommand = this.git.log();
Iterable<RevCommit> logCommits = logCommand.call();
...
for (RevCommit c : logCommits) {
    Date date = DateHandler.getDateFromEpoch(c.getCommitTime() * 1000L);
    ObjectId parentID = null;
    ...
    GitCommit commit = new GitCommit(c.getId(), date, c.getFullMessage());
    this.commitList.add(commit);
}
```

Progettazione – Merging Git w/ Jira (1)

- Il mapping dei dati ricavati da Jira e da GitHub è necessario per avere informazioni consistenti

- Si mantengono solamente le release presenti sia su Jira che su GitHub
  - Se la release non è presente su Jira, non siamo in grado di conoscere la lista di AV
  - Se la release non è presente su GitHub, non siamo in grado di ottenere la lista delle classi

- Si mantengono solamente i ticket che hanno un commit associato
  - Ci interessano quei ticket che hanno risolto un bug tramite un commit su GitHub

- Si mantengono solamente i commit che hanno un ticket associato
  - Se il commit non fa riferimento ad alcun ticket non abbiamo modo di determinare la buggyness delle classi presenti in quella release

Progettazione – Merging Git w/ Jira (2)

- Dopo aver effettuato il mapping si ha che ogni commit ottenuto, relativo alla risoluzione di un ticket, contiene nel suo commento l'identificativo del ticket stesso
- Questo permette di ottenere la *lista delle classi* coinvolte nella risoluzione di un Bug ed etichettarle come *difettose* nelle versioni [IV,FV)
  - Da quando il bug è stato introdotto (IV) alla versione precedente al fix (FV)

- Per recuperare i file *.java* presenti in una release, si utilizza la libreria JGit
  - Recuperiamo tutti i file presenti al momento del commit

```
while (treeWalk.next()) {
String classPath = treeWalk.getPathString();
    if (classPath.contains(Parameters.FILTER_FILE_TYPE)) {
        String className = PathHandler.getNameFromPath(classPath);
        ProjectClass projectClass = new ProjectClass(classPath,className,this);

        objectId = treeWalk.getObjectId(0);

    // Calcolo e setto la size della classe
        Metrics metrics = new Metrics();
        metrics.calculateSize(objectId, reader);
        projectClass.setMetrics(metrics);
        classList.add(projectClass);
    }
}
```

Progettazione – Sanificazione dei ticket (1)

- Prima di procedere con l'analisi delle classi buggy, è necessario sanificare l'elenco di ticket ottenuti da Jira, in modo da costruire un dataset che non presenti informazioni parziali o errate.

- Sanificazione della lista di *fixed versions* associata ad un ticket Jira
  - Lista vuota: si identifica come FV la prima release successiva alla data di risoluzione del ticket
  - Lista con più di una FV: si identifica come FV la più vecchia tra le versioni nella lista

- Scartiamo i ticket che non hanno una FV dopo la sanificazione
  - Non conoscendo la release in cui è stato fixato il bug, non siamo in grado di ottenere la lista di AV

Progettazione – Sanificazione dei ticket (2)

- Scartiamo i ticket che presentano informazioni errate su **IV**, **OV** e **FV**:
  1. **IV > OV = FV**, il bug è stato introdotto dopo averlo fixato
  2. **IV = OV = FV**, il bug è stato introdotto e risolto nella stessa release, quindi non ci sono AV
  3. **OV = FV** e **IV** assente, si ricade nel caso precedente anche applicando *Proportion*

- Dopo la sanificazione, rimangono solamente i ticket che presentano:
  1. **IV < OV = FV**, conosciamo tutte le informazioni per ricavare la lista di AV
  2. **OV < FV**, prediciamo l'IV applicando *Proportion*

Progettazione – Proportion

- Come già detto, alcuni ticket presenti su Jira sono caratterizzati dall'assenza di informazioni relative alle **IV**, motivo per cui non è possibile ricavare la lista di **AV**.
- Utilizziamo la tecnica **Proportion**, che ci permette di *stimare* la versione in cui un Bug è stato introdotto, potendo colmare così la mancanza dell'**IV**
  - Si segue l'idea per cui esiste una certa costante di proporzionalità tra il numero di versioni nell'intervallo [IV,FV] ed il numero di versioni nell'intervallo [OV,FV]
  - Costante di proporzionalità $\mathrm{P} = \frac{FV - IV}{FV - OV}$, costante per tutti i bug di un progetto

- Per quei ticket che mancano dell'**IV**, si effettua la predizione sfruttando la costante **P**, applicando la formula: $IV = FV - P * (FV - OV)$
- Nota l'**IV**, possono essere assegnate le **AV** ad ogni ticket, che corrispondono all'intervallo [IV,FV)

Progettazione – Metriche (1)

- Dopo aver analizzato la difettosità delle classi, vengono calcolate le seguenti metriche per le varie classi *java* individuate ad ogni release*:*
- ***Size****:* numero di linee di codice
- ***Age****:* differenza in settimane tra la data della release e quella di creazione della classe
- ***LOC Touched****:* numero di linee di codice modificate della classe
- ***LOC Added****:* numero di linee di codice aggiunte alla classe
- ***Max LOC Added****:* numero massimo di linee di codice aggiunte alla classe tra tutte le revisioni
- ***Avg LOC Added****:* media tra tutti i LOC added delle revisioni che hanno toccato la classe nella release
- ***Churn****:* differenza *[lineAdded – lineDeleted]* in una classe
- ***Max Churn****:* valore massimo di *Churn* di una classe tra tutte le revisioni
- ***NumberRevisions****:* numero di revisioni in cui è stata modificata la classe nella release
- ***NumberBugFixes****:* numero di bug fixati sulla classe nella release
- ***Nauth****:* numero di autori che hanno apportato modifiche alla classe

Progettazione – Metriche (2)

- La libreria JGit è uno strumento molto potente che ci permette di analizzare le modifiche introdotte su una precisa classe ad ogni commit effettuato, sfruttando l'oggetto *DiffEntry*, che corrisponde all'analogo del comando *git diff.*
  - Ogni oggetto *DiffEntry* rappresenta la differenza tra un commit ed il suo parent
  - Recuperata la lista di *DiffEntry*, si vanno a calcolare tutte le metriche relative a cambiamenti sulle classi, iterando su questi oggetti

- Il calcolo della buggyness viene effettuato quando ci si trova davanti a oggetti *DiffEntry* relativi a commit di tipo *bug fix*
  - La modifica di una classe in questo preciso commit indica che è stato risolto un bug, e quindi la classe era buggy in tutte le AV riportate sul ticket

```
/*
 * Imposta la buggyness di una classe in tutte le Affected Versions. Viene settata la buggyness partendo
 * dall'ultima AV (versione precedente al Fix) fino alla prima AV (injected version)
 */
public void setBuggyness(GitCommit fixCommit,String pathClass) {
    JiraTicket fixTicket = fixCommit.getTicket();
    List<JiraRelease> affectedVersions = fixTicket.getAffectedVersions();

    for (JiraRelease av:affectedVersions) {
        GitRelease gitAv = getReleaseByName(av.getName());
        if (gitAv != null) {
            ProjectClass projClass = gitAv.getProjectClass(pathClass);
            if (projClass!=null) {
                projClass.setBuggy(true);
            }
        }
    }
}
```

Progettazione – Weka (1)

- Per analizzare il dataset generato, si utilizzano le API offerte dal software ***Weka***.
- I classificatori utilizzati sono:
  1. *RandomForest*
  2. *NaiveBayes*
  3. *Ibk*

- Le tecniche valutate per lo studio dell'adeguatezza sono:
  1. Feature selection: No selection – Best First
  2. Balancing: No sampling – Oversampling – Undersampling – SMOTE
  3. Cost sensitive: No Cost Sensitive – Sensitive Threshold – Sensitive Learning

- *Walk Forward* è la tecnica implementata ed utilizzata per splittare il dataset in *TrainingSet* e *TestingSet*.

Progettazione – Weka (2)

- L'analisi del dataset viene realizzata nel seguente modo:
  1. Si converte il file *.csv* contenente il dataset in un file *.arff* così da fornirlo in input a Weka
  2. Per ogni classificatore, per ogni tecnica di feature selection, per ogni tecnica di balancing e per ogni tecnica di cost sensitivity
     - si esegue una *run* di walk forward
     - si calcolano le metriche di accuratezza (**Precision**, **Recall**, **AUC**, **Kappa**)
     - si calcola il risultato di ogni singola combinazione tra classificatore e tecniche utilizzate come la media dei risultati ottenuti dalle varie run

```
public List<WekaResult> runWalkForward() {
    int releasesNumber = getReleasesNumber(getDataset());
    List<WekaResult> resultList = new ArrayList<>();

    //per ogni classificatore, per ogni metodo di feature selection, per ogni metodo di balancing, per ogni iterazione di walk forward
    //viene salvato il risultato
    for(String classifierName : this.classifiers) {
        for(String featureSelectionName : this.featureSelectionMethods) {
            for(String resamplingMethodName : this.resamplingMethods) {
                for (String costSensitive : this.costSensitiveApproach) {
                    String configuration = String.format("Classifier: %s%nFeatureSelection: %s%nResampling: %s%nCostSensitive: %s%n--------", classifier
                    Logger.getLogger(WekaProject.class.getName()).info(configuration);
                //con walk-forward partiamo dalla seconda release come test set perche non abbiamo un training set per la prima
                //terminiamo con l'ultima release come test set che avra tutte le precedenti come training set
                WekaResult mean = new WekaResult(classifierName, featureSelectionName, resamplingMethodName, costSensitive);
                for(int i = 2; i < releasesNumber; i++) {
                    WekaResult result = new WekaResult(classifierName, featureSelectionName, resamplingMethodName, costSensitive);
                    Instances[] trainTest = splitTrainingTestSet(getDataset(), i);
                    runWalkForwardIteration(trainTest, result, i);
                    resultList.add(result);
                    mean.setTotalValues(result);
                }
                mean.calculateMean(releasesNumber-2);
                resultList.add(mean);
                }
            }
        }
    }
    return resultList;
}
```

Risultati – Bookkeeper (1)

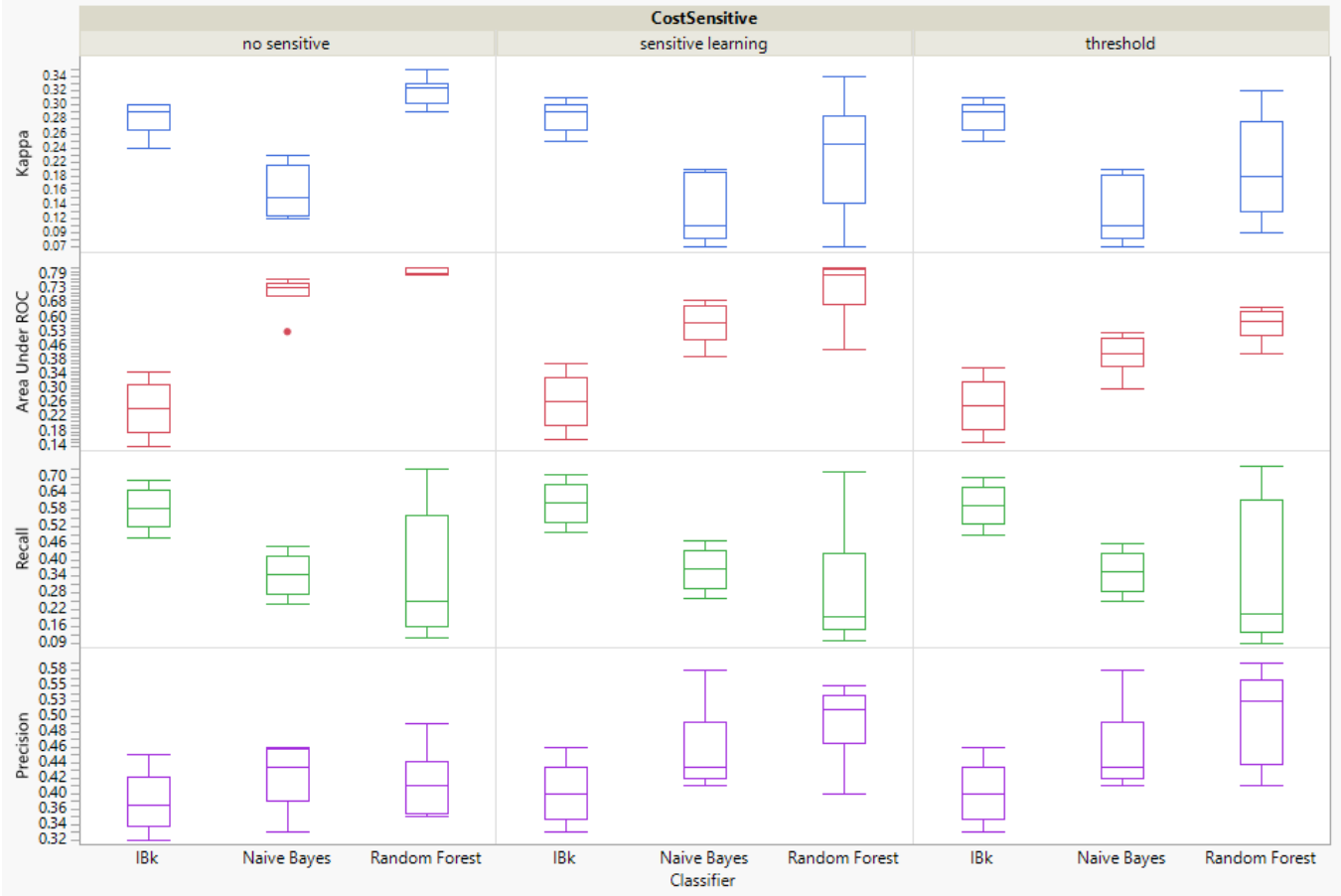
- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Feature Selection*
- ***IBk*** presenta una maggiore accuratezza usando Best First
  1. Best First: **0.41** – **0.65** – **0.30** – **0.29**
  2. No feature: 0.34 – 0.52 – 0.20 – 0.26
- ***NaiveBayes*** risulta più accurato senza feature selection, classifica meno positivi ma migliora l'accuratezza rispetto a un classificatore dummy
  1. Best First: 0.46 – 0.45 – 0.53 – 0.11
  2. No feature: 0.46 – 0.25 – 0.65 – **0.22**
- ***RandomForest*** presenta una maggiore accuratezza usando Best First, diminuisce la Recall ma tutte le altre metriche migliorano.

Risultati – Bookkeeper (2)

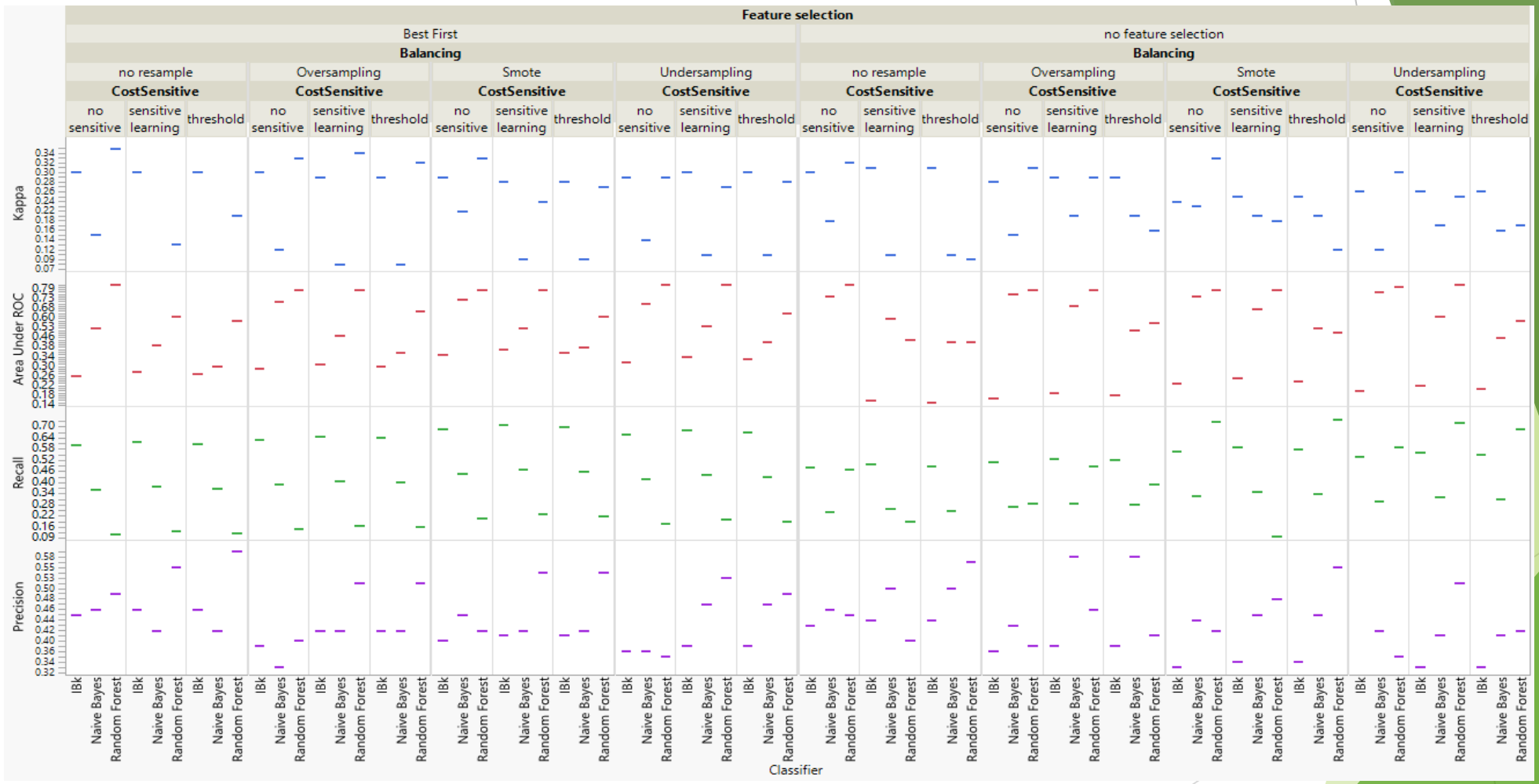
- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Balancing*
- ***IBk*** presenta una maggiore accuratezza senza l'uso di tecniche di balancing
  - Le tecniche migliorano la Recall ma Precision e Kappa diminuiscono notevolmente
- ***NaiveBayes*** presenta una maggiore accuratezza con SMOTE
  - La Precision è simile tra le varie tecniche, ma SMOTE migliora notevolmente le altre metriche
- ***RandomForest*** presenta la maggiore accuratezza usando SMOTE e Undersampling
  1. SMOTE: 0.52 – 0.20 – 0.74 – 0.27
  2. Undersampling: 0.46 – 0.41 – 0.75 – 0.30
- Possiamo dire che Undersampling massimizza la Recall per tutti i classificatori

Risultati – Bookkeeper (3)

- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Cost Sensitivity*
- ***IBk*** presenta un'accuratezza simile per qualsiasi tecnica di cost sensitivity utilizzata
  - Le tecniche migliorano la Recall ma Precision e Kappa diminuiscono
- ***NaiveBayes*** presenta un'accuratezza simile per qualsiasi tecnica di cost sensitivity utilizzata
  - Senza l'uso di una tecnica di cost sensitivity si massimizzano i valori di AUC e Kappa
- ***RandomForest*** presenta la maggiore accuratezza usando la tecnica *Sensitive Learning*
  1. Sensitive Learning: 0.50 – 0.22 – 0.75 – 0.24
  2. Sensitive Threshold: 0.55 – 0.22 – 0.6 – 0.16
- *Sensitive Threshold* migliora leggermente la Precision per RandomForest, ma Sensitive Learning massimizza le altre metriche



Risultati – Bookkeeper (4)



Risultati - Conclusioni

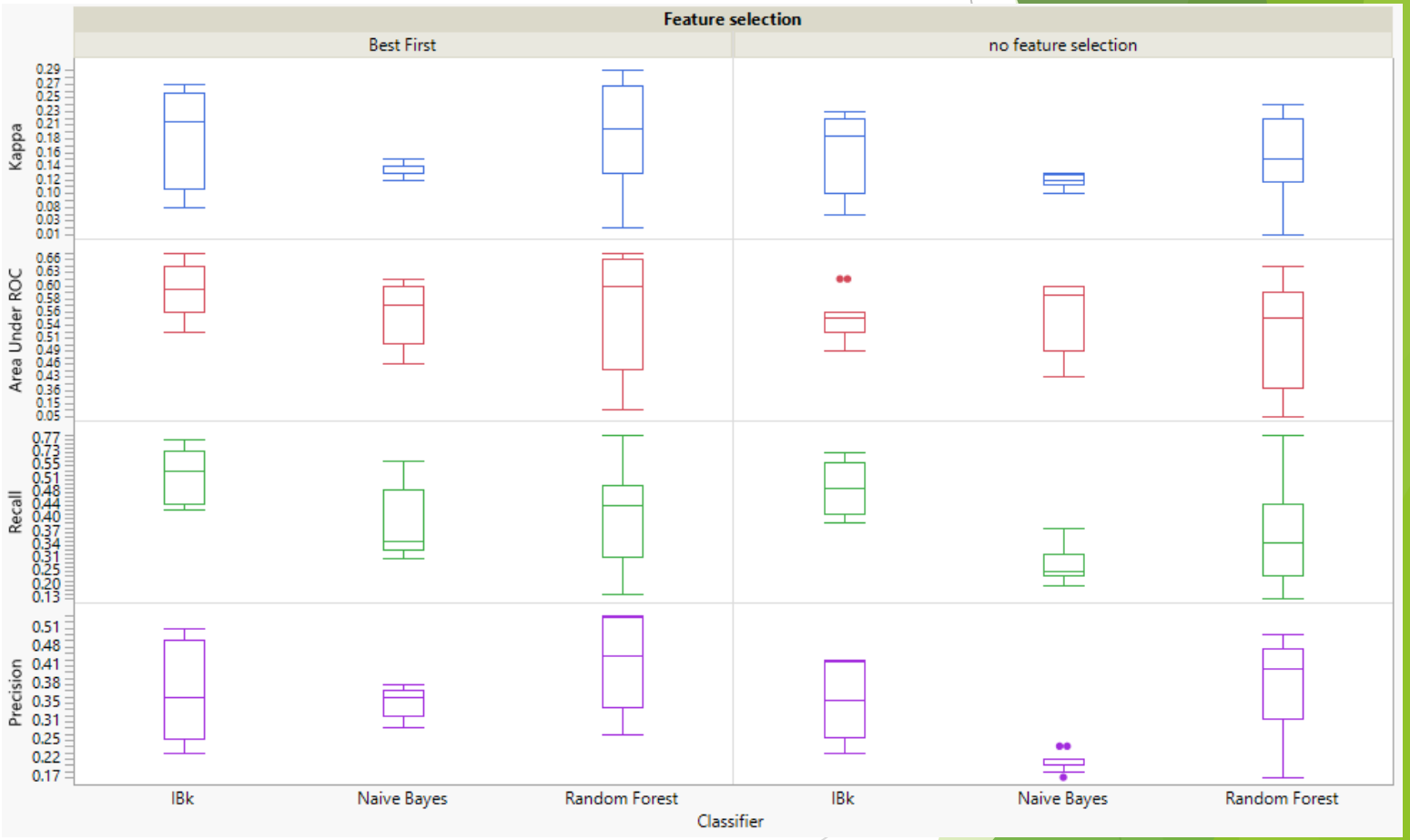
- Se si vuole massimizzare la Precision, **RandomForest** offre le migliori prestazioni utilizzando ***Best First/No resample/Sensitive Threshold****:*
  - *Precision 0.62*
  - *Recall molto bassa 0.11*

- Se si vuole massimizzare la Recall, **IBk** garantisce le migliori prestazioni utilizzando ***Best First/SMOTE,*** indipendentemente dalla tecnica di *sensitivity* scelta:
  - *Recall in [0.66; 0.70]*
  - *Anche le altre metriche presentano dei valori accettabili*

- L'accuratezza migliore viene raggiunta utilizzando **IBk** con ***Best First/No resample/Sensitive Learning***:
  - Precision 0.46 – Recall 0.62 – AUC 0.3 – Kappa 0.32
  - Si ha un buon compromesso tra Precision e Recall rispetto alle altre combinazioni di tecniche e classificatori

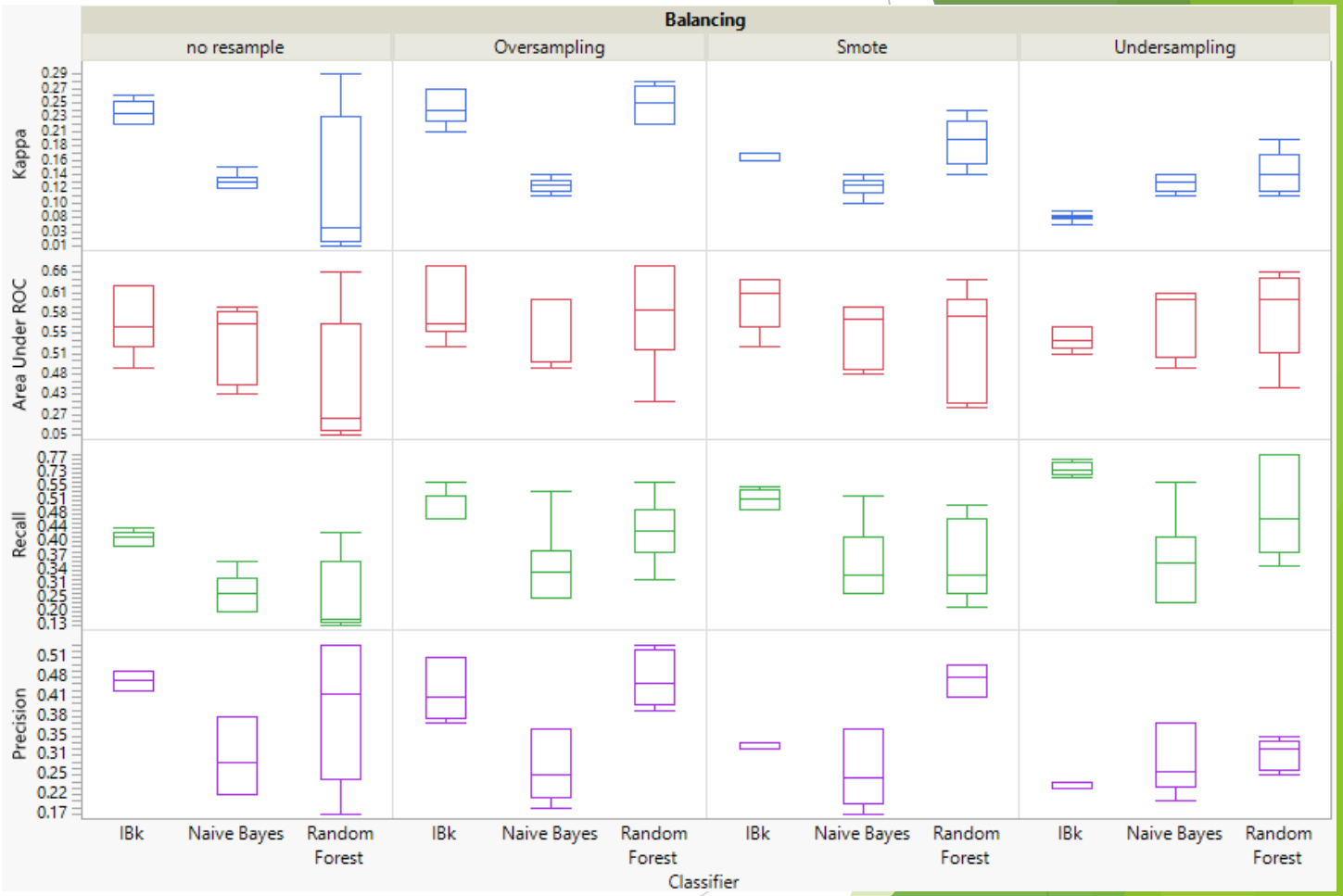
Risultati – Avro (1)

- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Feature Selection*
- Tutti quanti i classificatori presentano una maggiore accuratezza utilizzando *Best First*
- Tutte quante le metriche vengono migliorate, di conseguenza possiamo dire che il dataset è caratterizzato da alcune features poco correlate con la variabile di interesse, che non vengono considerate da Best First andando a migliorare l'accuratezza.



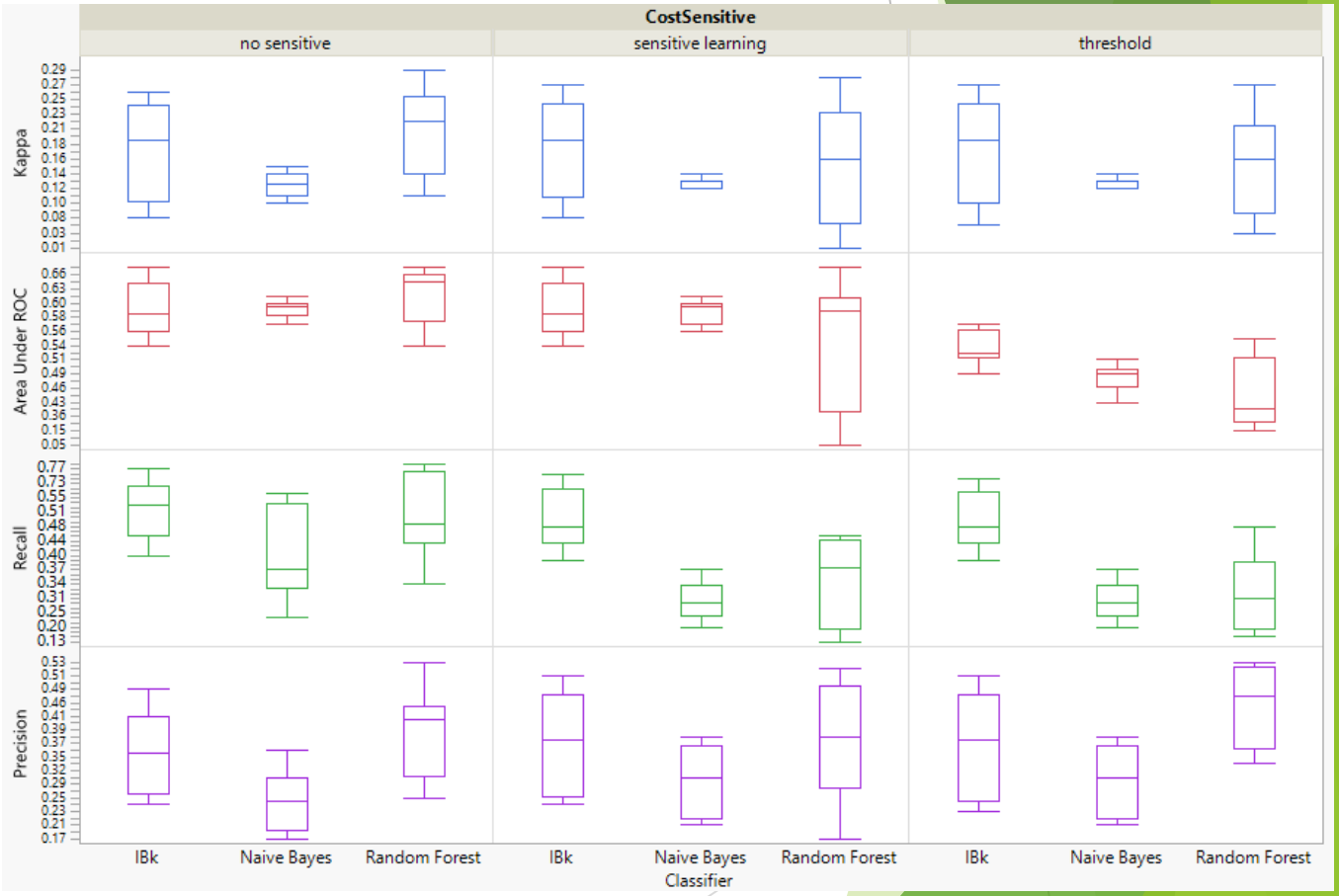
Risultati – Avro (2)

- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Balancing*
- ***IBk*** presenta una maggiore accuratezza con Oversampling e senza l'uso di tecniche di balancing
  - Oversampling migliora la Recall a discapito della Precision
- ***NaiveBayes*** presenta una maggiore accuratezza con Undersampling
  - La Precision è simile a prescindere dalla tecnica usata, ma Undersampling migliora notevolmente le altre metriche
- ***RandomForest*** presenta la maggiore accuratezza usando SMOTE e Oversampling
  1. SMOTE: 0.51 – 0.35 – 0.58 – 0.18
  2. Undersampling: 0.48 – 0.44 – 0.58 – 0.25

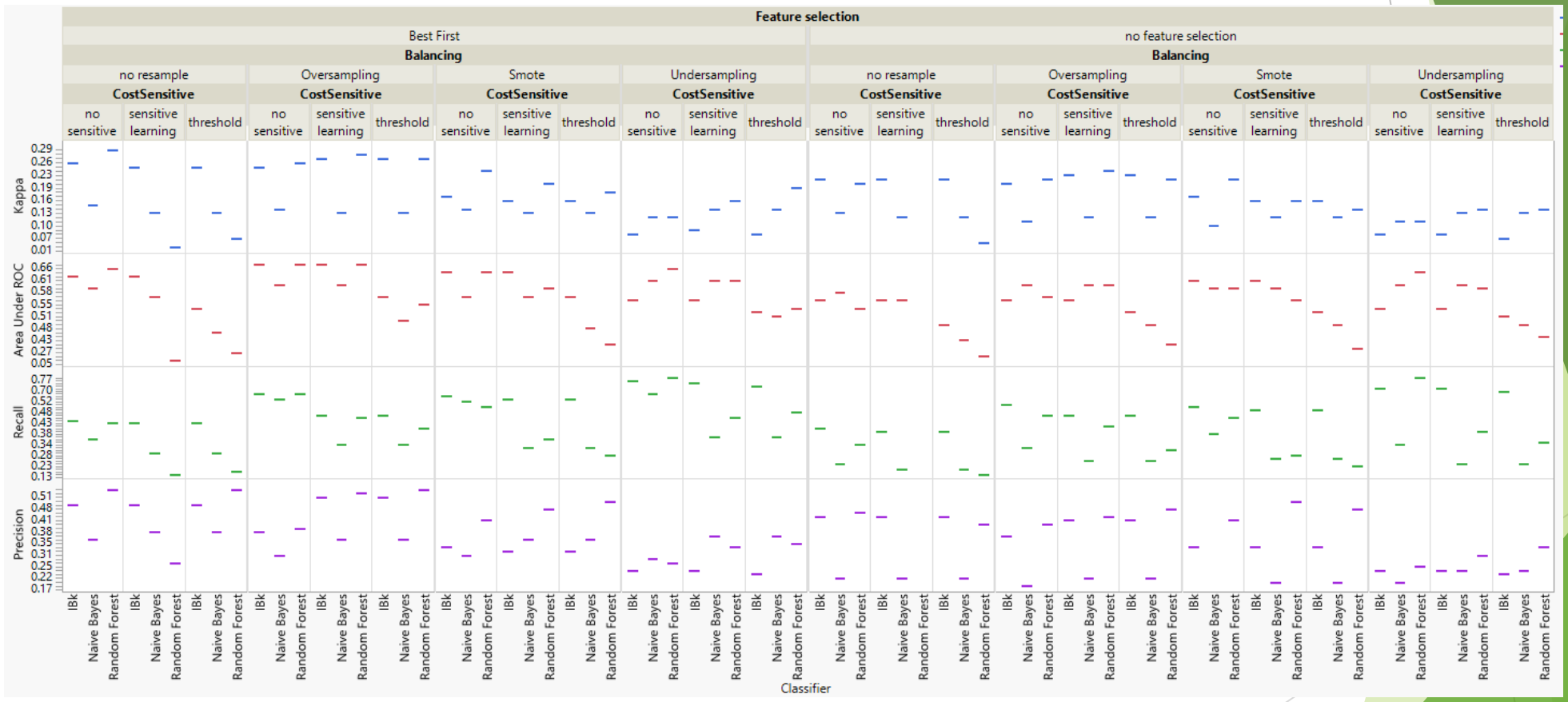


Risultati – Avro (3)

- Analizziamo i valori medi ottenuti dalle iterazioni di Walk Forward usando diverse tecniche di *Cost Sensitivity*
- ***IBk*** presenta un'accuratezza simile per qualsiasi tecnica di cost sensitivity utilizzata
  - Le tecniche migliorano la Precision ma la Recall diminuisce
- ***NaiveBayes*** presenta un'accuratezza simile per qualsiasi tecnica di cost sensitivity utilizzata
  - Senza l'uso di una tecnica di cost sensitivity si massimizza la Recall
- ***RandomForest*** presenta la maggiore accuratezza senza l'uso di una tecnica di cost sensitivity
  1. No Sensitive: 0.43 – 0.51 – 0.64 – 0.23
  2. Sensitive Threshold: 0.50 – 0.31 – 0.43 – 0.18
- *Sensitive Threshold* migliora leggermente la Precision per RandomForest, ma senza l'uso di una tecnica di cost sensitivity si massimizzano le altre metriche



Risultati – Avro (4)



Risultati - Conclusioni

- Se si vuole massimizzare la Precision, **RandomForest** offre le migliori prestazioni utilizzando ***Best First/No resample/No Sensitive****:*
  - *Precision 0.54*
  - *Anche le altre metriche presentano dei valori accettabili*

- Se si vuole massimizzare la Recall, **RandomForest** garantisce le migliori prestazioni utilizzando ***Undersampling/No Sensitive****,* indipendentemente dalla tecnica di *feature selection* scelta:
  - *Recall 0.75*
  - *Precision bassa 0.25*

- L'accuratezza migliore viene raggiunta utilizzando **IBk** con ***Best First/Oversampling/Sensitive Learning***:
  - Precision 0.49 – Recall 0.50 – AUC 0.66 – Kappa 0.28
  - Si ha un buon compromesso tra Precision e Recall rispetto alle altre combinazioni di tecniche e classificatori

- In *Avro* abbiamo un maggior numero di versioni che ci permettono di avere un dataset più ampio, ma la qualità del dataset risulta essere molto bassa perché si registra un massimo di **0.29** per i valori di Kappa
  - La predizione è poco accurata, questo perché i classificatori si discostano di poco dall'accuratezza di un classificatore dummy

Links

- Repository GitHub: https://github.com/jacopofabi/ISW2-deliverable2
- Travis CI: https://travis-ci.com/github/jacopofabi/ISW2-deliverable2
- SonarCloud: https://sonarcloud.io/dashboard?id=jacopofabi_ISW2-deliverable2