Deliverable

ISW2 - Davide Falessi

Daniele Mariano

0294300

Indroduzione

- Il corso ha posto principalmente l'attenzione verso l'utilizzo delle tecniche di **machine learning** per analizzare il nostro target ed eseguire **testing** sulle parti calde del sistema di nostro interesse.
  - Le parti calde rappresentano quelle classi dentro le quali abbiamo più probabilità di trovare quello che stiamo cercando; in questo modo si ottimizzano le analisi di questo tipo.
- Le operazioni di retrieving di **bug**, o di potenziali **classi buggy**, sono fondamentali per evitare che l'utente si trovi di fronte a **failure** del codice. Ovvero comportamenti osservati dall'utente che non erano previsti dal sistema, questi sono effetto di (almeno) un bug (difetto).
- L'obiettivo quindi è quello di cercare di non avere bug nel codice ed inoltre di indirizzare lo sviluppo per prevenirne futuri cercando di monitorare il **techincal debt**.

Strumenti

- Per ottenere le informazioni necessarie alla nostra analisi si è utilizzato lo stack GithHub, Jira, SonarCLoud, Java e WEKA.
  - **Git** è stato utilizzato sia per il versioning del nostro progetto che in combinazione con **Jira** per il reporting delle informazioni di monitoraggio dei bug nello sviluppo del progetto target.
  - **Java** è il linguaggio scelto per lo sviluppo del nostro progetto utilizzando l'IDE Eclipse ed IntelliJ (per poter sfruttare i tool di entrambi).
  - **SonarCloud** è un tool mediante la cui analisi vengono rilevati bug, code smell o più in generale criticità nelle repository linkate fornendo il feedback necessario per migliorare la scrittura del codice.
  - **Weka** è una raccolta di algoritmi di apprendimento automatico per attività di data mining; ovvero per la preparazione, la classificazione, la regressione, il clustering, l'estrazione di regole di associazione e la visualizzazione dei dati.

Specifica

- Per la deliverable è stato quindi sviluppato un programma che permettesse, specificando il nome del progetto da analizzare, di:
  - Considerare e localizzare la **bugginess** delle classi che lo compongono.
    - Questa analisi sarà messa in risalto attraverso la produzione di un documento, un file CSV ottenuto in output, che mostrerà, tra gli altri valori raccolti, la bugginess (calcolata utilizzando il metodo proportion) della classe attraverso un booleano.
      - True se il valore è maggiore di zero, false altrimenti.
      - Le prime due colonne di questo file saranno la version (parametro che ordinerà la nostra lista) ed il nome del file, mentre le successive rappresentano le metriche scelte tra l'elenco proposto dal professore.
  - comparare l'**accuratezza** (Precision, Recall, AUC e Kappa) di tre **classificatori** (Random Forest, Naive Bayes e Ibk) utilizzando la tecnica di validazione **Walk Forward**.
    - Anche per questa analisi si produrrà un file CSV che raccoglierà sulle colonne le informazioni riguardanti il Dataset, la TrainingRelease, Classifier, Precision, Recall, AUC e Kappa.

Roadmap

- La guideline qui proposta ha l'intento di esporre i passi che hanno portato all'ottenimento dei dati seguendo appunto i seguenti step:
  - **Implementazione**:
    - Retrieving dei dicket da Jira,
    - Collecting delle informazioni della repository,
    - Binding dei dati,
    - Calcolo della bugginess,
    - Calcolo delle metriche,
    - Scrittura dei file di output.
  - **Technical debt:**
    - SonarCloud.
  - **Rsultati:**
    - Analisi dei CSV.

Implementazione

Step 1

- Il primo step è il **retrieving dei ticket da Jira**.
- Per fare ciò nel main viene costruita l'url (parametrica) che se interrogata restituisce un Json contenente le informazioni che noi andiamo a mappare nelle rispettive liste.

```
String url = "https://issues.apache.org/jira/rest/api/2/search?jql=project=%22"
           + projName + "%22AND%22issueType%22=%22Bug%22AND(%22status%22=%22closed%22OR"
           + "%22status%22=%22resolved%22)AND%22resolution%22=%22fixed%22&fields=key,resolutiondate"
           + ",affectedVersion,versions,created&startAt="
       + i.toString() + "&maxResults=" + j.toString();
List<Date> createdarray = main.GetJsonFromUrl.dateArray(url, i , j , "created");
List<Date> resolutionarray = main.GetJsonFromUrl.dateArray(url, i , j , "resolutiondate");
List <String> keyArray = main.GetJsonFromUrl.keyArray(url, i, j);
List <String> version = main.GetReleaseInfo.versionArray(url, i ,1000, "name");
List<Ticket> ticket = GetJsonFromUrl.setTicket(createdarray,resolutionarray,version,keyArray);
```

- A questo punto è bene chiarire gli elementi principali che troviamo nel progetto, questi sono:
  - **Ticket**, **Release**, **Commit** e **Classi**.
- Per quanto riguarda il primo elemento, utilizzando il codice implementato per la prima deliverable la **RetrieveTicket**, sono stati raccolti i Ticket nelle due liste differenziate una per la data di creazione e l'altra per la data di risoluzione del ticket.
  - Ad un ticket corrisponde una commit.
- Mentre la relazione gerarchica che intercorre tra i rimanenti tre attori principali nel progetto è invece rappresentata dal fatto che ad una determinata Release sono associate un numero di Commit ad ognuna delle quali troviamo associato un altro numero di Classi che sono state modificate.

1.1

Step 2

- Il secondo step è il **collecting delle informazioni dalla repository**.
  - Memori della struttura gerarchica precedentemente menzionata, andando avanti nel codice del main vediamo come raccogliamo le informazioni necessarie ad avere un quadro circa le commit e le release del nostro progetto.
- Per raccogliere le informazioni su questi ulteriori due elementi si è cercato di riutilizzare il codice del professor Falessi ed il codice scritto per la 1° deliverable. Con la **getReleaseInfo** si sono ottenute le release da Jira per il progetto in questione. Mentre con la **getGitInfo** otteniamo commit dalla repository.

```
List <Commit> commit = GetGitInfo.commitList();
int size = GetReleaseInfo.getReleaseList().size();
List<Release> releases = GetReleaseInfo.getReleaseList();
```

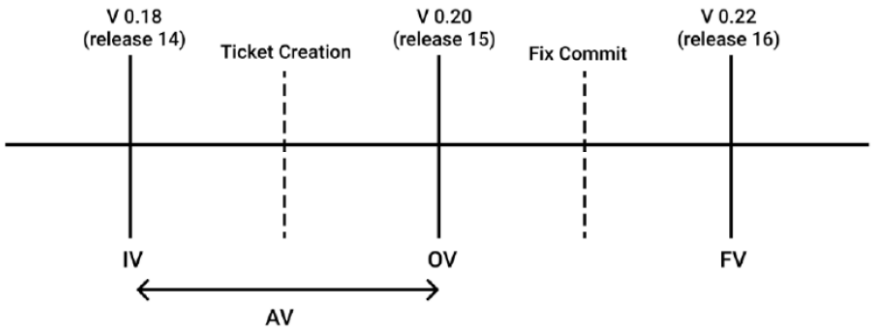
Step 3

- Sempre procedendo con l'analisi del main consideriamo il terzo step. Questo è diviso in tre parti:
  - l'**associazione della commit ad una determinata release** (attraverso la dateComparator);
  - il **collecting delle affected version** (utilizzando la returnAffectedVersion);
  - l'**associazione della classe ad una determinata release** (attraverso la setClassToRelease).

```
dateComparator(releases,commit);
for(Ticket t : ticket) {
    GetJsonFromUrl.returnAffectedVersion(t, releases);
}
List<Ticket> ticketConCommit = GetGitInfo.setClassVersion(ticket,commit,releases);
Proportion.checkIV(ticketConCommit);
GetReleaseInfo.setClassToRelease(releases, commit);
```

Concetti di **affected version** (AV) e **proportion**

- Un progetto è un insieme di classi che sono quindi un insieme di release che è a sua volta un insieme di versioni (ovvero ogni tot di versioni abbiamo una release).
  - Ad ogni commit quindi corrisponde una specifica versione.
- Per determinare quando una classe è difettosa (buggy) abbiamo analizzato due metodi: SZZ (basato sul concetto di snoring classes) e Proportion.
  - Quest'ultimo, di nostro interesse in quanto è il metodo utilizzato all'interno del nostro progetto, si occupa di andare a vedere il ciclo di vita del difetto.
  - L'intuizione sta nel considerare proporzionale lo spazio tra l'IV (injected version) e OV (opening version) e quello tra l'OV e la FV (fixed version).



$$P = (FV - IV) / (FV - OV)$$

$$\text{Predicted } IV = FV - (FV - OV) * P$$

- Per il calcolo di P abbiamo 3 varianti:
  - Approccio cold start:
    - Calcoliamo P come la mediana, se non ho bug precedenti utilizzo dati presi da altri progetti.
  - Approccio incrementale:
    - Calcoliamo P come media tra i difetti corretti nelle versioni precedenti. Sfruttiamo l'assunzione che il ciclo di vita è costante tra i difetti usando tutta la storia passata, parte della quale risulta però obsoleta.
  - Approccio moving window (utilizzato nel progetto):
    - Calcoliamo P come la media tra l'ultimo 1% dei difetti. Utilizzo sempre lo stesso numero di bug che però traslano in avanti con l'andare avanti nel tempo.

3.2

Step 4

- Il quarto step rappresenta il **calcolo della bugginess**.
  - Eseguito sulla metà delle release.
- Questo viene eseguito nella computeBuggyness dopo che sono state associate le commit alla release successivamente all'associazione delle classi alla release.

```
List<Release> halfReleases = releases.subList(0, size/2);
GetReleaseInfo.assignCommitListToRelease(halfReleases, commit);
computeBuggyness(halfReleases);
```

Step 5

- Il quinto step è rappresentato dal **calcolo delle metriche** utilizzando la getMetrics per poi eseguire la setMetric ciclicamente sulle release per assegnargli le metriche calcolate.
  - Le metriche sono state calcolate utilizzando le commit ottenute da Jgit (implementazione Java del sistema di controllo della versione Git).

```
JSONArray jsonArray = GetDiffFromGit.getMetrics(halfReleases);
for (Release r: halfReleases) {
    GetDiffFromGit.setMetric(r, jsonArray);
}
```

Step 6

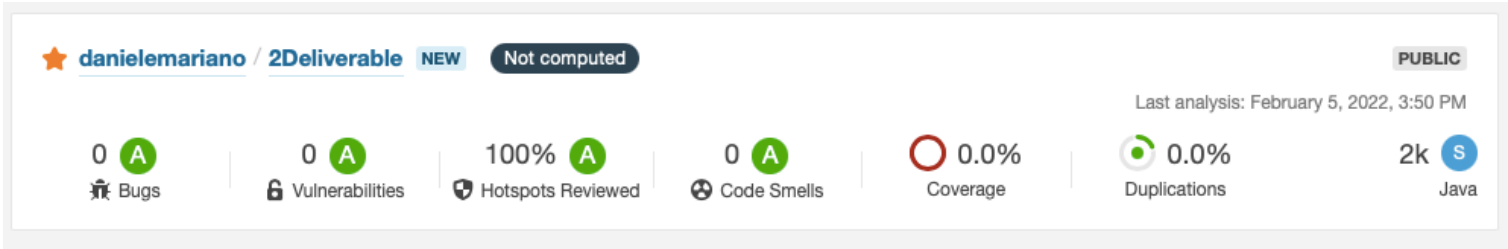
- Il sesto step è rappresentato della **scrittura dei due file csv**.
  - Il primo utilizzando la csvFinal.
  - Mentre il secondo la csvByWeka per differenziare i due file in quanto questo è quello con i valori prodotti dall'analisi condotta mediante i tool di Weka.
    - Ogni CSV prodotto con il Walk Forward viene tradotto in un file ARF in questo modo può essere passato a WEKA e poi cicliamo per ognuno dei tre classificatori da utilizzare.

```
csvFinal(halfReleases, 0);

List<WekaData> wekaList = new ArrayList<>();
List<String> trainingSet = TestWekaEasy.makeTrainingSet(halfReleases);
List<String> testingSet = TestWekaEasy.makeTestingSet(halfReleases);

for(int z = 1; z< testingSet.size()+1; z ++) {
    wekaList.addAll(TestWekaEasy.wekaAction(testingSet.get(z-1), trainingSet.get(z-1), z,
            getDefectiveInTraining(halfReleases, z)));
}
CsvMaker.csvByWeka(wekaList, halfReleases);
```

Technical debt

sonarCloud

- La repository contente il codice è stata linkata al tool sonarCloud che ha eseguito un'analisi circa la sicurezza, i bug, le vulnerabilità ed i code smell.
- Sono state infine eseguite delle revisioni affinché il debito tecnico fosse ricondotto a zero e potessimo massimizzare tutti gli altri forniti dal tool, come da specifica.



Risultati

Dati

- Si propone un esempio dei valori ottenuti in output dei due file csv finali prodotti:

Release	className	LOC	Age	CHG	MAX_CHG	AVG_CHG	Bug Fixed	NAuth	Number of Commit	LOC Added	AVG_LOC Added	MAX_LOC Added	Buggy
1	bookkeeper-benchmark/src/main/java/org/apache/bookkeeper/benchmark/MySqlClient.java	0	570	207	171	63	0	0	0	0	3.070.752	6906	false
1	bookkeeper-benchmark/src/main/java/org/apache/bookkeeper/benchmark/TestClient.java	302	570	207	207	52	0	3	27	604	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/Bookie.java	0	570	207	207	138	0	0	0	0	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/BookieException.java	0	570	207	207	155	0	0	0	0	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/BufferedChannel.java	0	570	207	207	165	0	0	0	0	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/EntryLogger.java	0	570	207	207	172	0	0	0	0	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/FileInfo.java	0	570	207	207	177	0	0	0	0	3.070.752	6906	false
1	bookkeeper/src/main/java/org/apache/bookkeeper/bookie/LedgerCache.java	0	570	207	207	181	0	0	0	0	3.070.752	6906	false

Dataset	#trainingRelease	%training	%DefectiveInTraining	%DefectiveInTesting	Classifier	Balancing	FeatureSele	Cost Sensitive	Sensitivity	TP	FP	TN	FN	Recall	Precision	AUC	Kappa
1	1	14	13	19	NaiveBayes	NoSampling	true	no Cost Sensitive	0	1.0	0.18282548	0.1828254	0.0	1.0	0.5479452054	0.9099030470	0.6185621052079783
1	1	14	13	19	NaiveBayes	OverSampling	true	no Cost Sensitive	0	1.0	0.18282548	0.1828254	0.0	1.0	0.5479452054	0.9081024930	0.6185621052079783
1	1	14	13	19	NaiveBayes	UnderSampling	true	no Cost Sensitive	0	1.0	0.18282548	0.1828254	0.0	1.0	0.5479452054	0.9138677285	0.6185621052079783
1	1	14	13	19	NaiveBayes	Smote	true	no Cost Sensitive	0	1.0	0.18282548	0.1828254	0.0	1.0	0.5479452054	0.9121537396	0.6185621052079783
1	1	14	13	19	NaiveBayes	NoSampling	true	Sensitive Treshold	90	0.95	0.16897506	0.1689750	0.05	0.95	0.5547445255	0.8905124653	0.6114642774848529
1	1	14	13	19	NaiveBayes	NoSampling	true	Sensitive Treshold	83	0.9875	0.17728531	0.1772853	0.0125	0.9875	0.5524475524	0.9051073407	0.6201449717079893
1	1	14	13	19	NaiveBayes	NoSampling	true	Sensitive learning	90	1.0	0.17728531	0.1772853	0.0	1.0	0.5555555555	0.9099030470	0.6273764258555133

Grazie per l'attenzione