REPORT MACHINE LEARNING FOR SOFTWARE ENGINEERING

Enrico Ciarla - 0309950

Indice

- Introduzione
  - Contesto / Problema / Obiettivo
- Progettazione
  - Creazione del Dataset / Labeling delle Classi / Accuratezza del Dataset
- Tecniche di Validazione
- Walk Forward
- Caso Studio
- Feature Selection
- Cost Sensitivity
- Risultati e discussione
  - BookKeeper / ZooKeeper

Introduzione – Contesto (1/2)

- Il costo dei bug nel software è stimato essere 59.5 miliardi/anno solo negli USA
- Il software viene scritto in diversi linguaggi di programmazione, da molteplici persone e su un lungo periodo di tempo per:
  - Fixare i bug
  - Aggiungere nuove feature
  - Migliorare la qualità del codice

Introduzione – Contesto (2/2)

- Come prevenire i bug?
  - Usare tool di analisi statistica (e.g. sonarqube)
  - Monitorare il ***technical debt***
- Come trovare i bug?
  - Test di unità per testare la corretta funzionalità
  - CI/CD per fare automaticamente build/test
  - Code review per trovare bug e controllare la qualità del codice

Introduzione - Problema

- Il QA (Quality Assurance) è costoso…
- Dato un periodo limitato di tempo, come possiamo prioritizzare le nostre risorse di QA sugli elementi più rischiosi del nostro software?

Introduzione - Obiettivo

- Soluzione:
  - Tecniche di Machine Learning (ML) per la prevenzione dei difetti
- Step:
  1. Scegliere il progetto di cui si vuole fare la predizione dei difetti
  2. Ottenere i dati grezzi da Jira e Git
  3. Creare un dataset con diverse metriche relative ad ogni release ed etichettare una classe come buggy oppure no
  4. Pulire il dataset togliendo i dati non affidabili
  5. Usare le tecniche di ML sul dataset per ottenere un predittore
- Caso studio sui progetti Apache BookKeeper e ZooKeeper

Progettazione – Creazione del Dataset (1/3)

- Sul repository Git per ogni classe in ogni release vengono calcolate le seguenti metriche:

Metrica	Descrizione
Size	LOC
LOC_touched	Somma su ogni revisione di LOC aggiunte+modificate+eliminate
NR	Numero di revisioni
NAuth	Numeri di autori
LOC_added	Somma su ogni revisione di LOC aggiunte
MAX_LOC_added	Il massimo di LOC aggiunte sulle revisioni
AVG_LOC_added	La media di LOC aggiunte
Churn	Somma sulle revisioni di LOC aggiunte-eliminate
MAX_Churn	Il massimo Churn sulle revisioni

Progettazione – Creazione del Dataset (2/3)

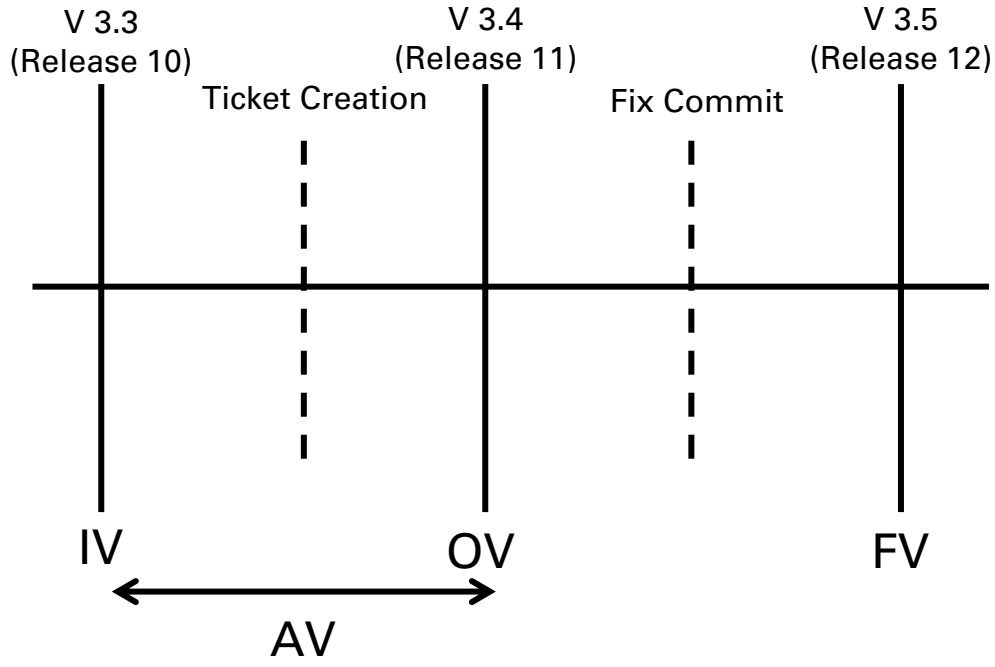
- Su Jira considerare i ticket ritornati dalla query:
  - *Type == "Bug" AND (status == "Closed" OR status == "Resolved") AND Resolution == "Fixed"*
- Jira mette a disposizione una entry dove segnare le versioni affette dal bug relativo al ticket (e.g. *Affected Versions*: 3.2, 3.3, 3.7)
- Andare quindi sul repository Git e ricercare il commit relativo al ticket in questione:
  - Le classi modificate in tale commit vengono segnate come difettose nelle release segnate nel campo *Affected Versions*

Progettazione – Creazione del Dataset (3/3)

- Problema
  - Non sempre il campo *Affected Versions* è disponibile
- Soluzione
  - Utilizzare un metodo per fare il labeling delle *Affected Versions* quando il campo non è disponibile

Progettazione – Labeling delle Classi (1/4)

- Da Jira abbiamo le seguenti informazioni
  - Data di creazione del ticket
  - Data di chiusura del ticket (fix)
- Possiamo quindi andare su Git ed ottenere
  - Opening Version (versione successiva alla creazione del ticket)
  - Fixed Version (versione successiva al commit del fix)
- Come troviamo l'Injected Version?



Esempio di ciclo di vita di un difetto: Injected Version (IV), Opening Version (OV), Fixed Version (FV), Affected Versions (AV)

Progettazione – Labeling delle Classi (2/4)

- Non possiamo sapere quante versioni occorrono tra l'IV e l'OV
- **Proportion** method
  - Intuizione: se è ampio il tempo tra OV e FV, lo sarà anche quello tra IV e FV
  - P = (FV-IV) / (FV-OV)
  - Predicted IV = FV – (FV-OV)*P
- Per calcolare P nel caso studio è stato usato l'approccio *incremental*
  - P viene calcolato come media dei bug fixati nelle versioni precedenti

Progettazione – Labeling delle Classi (3/4)

- Alla fine dei passaggi descritti dovremmo avere un dataset (file .csv) con le seguenti colonne:
  - Release
  - Class
  - Size
  - LOC_touched
  - NR
  - Nauth
  - LOC_added
  - MAX_LOC_added
  - AVG_LOC_added
  - Churn
  - MAX_Churn
  - Buggy (yes/no)

Progettazione – Labeling delle Classi (4/4)

release	class_name	size	LOC_touched	NR	NAuth	LOC_added	MAX_LOC_added	AVG_LOC_added	churn	MAX_churn	buggy
5	java/jmx/com/yahoo/zookeeper/jmx/server/ConnectionMXBean.j	13	278	6	2	264	66	44	264	66	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/DataTreeBean.java	35	384	6	2	356	89	59	356	89	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/DataTreeMXBean.jav	8	190	6	2	176	44	29	176	44	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/ZooKeeperServerBea	59	402	6	2	380	95	63	380	95	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/ZooKeeperServerMX	15	302	6	2	288	72	48	288	72	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/FollowerBean	6	118	6	2	104	26	17	104	26	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/FollowerMXB	3	110	6	2	96	24	16	96	24	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LeaderBean.	6	122	6	2	108	27	18	108	27	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LeaderElecti	15	180	6	2	164	41	27	164	41	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LeaderElecti	5	142	6	2	128	32	21	128	32	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LeaderMXBe	3	110	6	2	96	24	16	96	24	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LocalPeerBe	19	202	6	2	184	46	30	184	46	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/LocalPeerMX	5	144	6	2	128	32	21	128	32	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/QuorumBean	18	186	6	2	168	42	28	168	42	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/QuorumMXB	5	138	6	2	124	31	20	124	31	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/RemotePeer	18	190	6	2	172	43	28	172	43	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/RemotePeer	5	138	6	2	124	31	20	124	31	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/ServerBean.j	12	160	6	2	144	36	24	144	36	no
5	java/jmx/com/yahoo/zookeeper/jmx/server/quorum/ServerMXBe	5	138	6	2	124	31	20	124	31	no
5	java/jmx/com/yahoo/zookeeper/server/ManagedZooKeeperServ	78	584	6	2	552	138	92	552	138	no
5	java/jmx/com/yahoo/zookeeper/server/ObservableDataTree.java	75	472	6	2	444	111	74	444	111	no
5	java/jmx/com/yahoo/zookeeper/server/ObservableNIOServerCn	49	352	6	2	332	83	55	332	83	no
5	java/jmx/com/yahoo/zookeeper/server/ObservableZooKeeperSe	21	214	6	2	200	50	33	200	50	no
5	java/jmx/com/yahoo/zookeeper/server/ZooKeeperObserverNotif	33	316	6	2	296	74	49	296	74	no
5	java/jmx/com/yahoo/zookeeper/server/quorum/ManagedQuorum	168	1032	6	2	972	243	162	972	243	no
5	java/jmx/com/yahoo/zookeeper/server/quorum/ObservableFollov	46	324	6	2	304	76	50	304	76	no
5	java/jmx/com/yahoo/zookeeper/server/quorum/ObservableFollov	22	232	6	2	212	53	35	212	53	no
5	java/jmx/com/yahoo/zookeeper/server/quorum/ObservableLead	48	340	6	2	320	80	53	320	80	no
5	java/jmx/com/yahoo/zookeeper/server/quorum/ObservableLead	22	232	6	2	212	53	35	212	53	no

Progettazione – Accuratezza del dataset

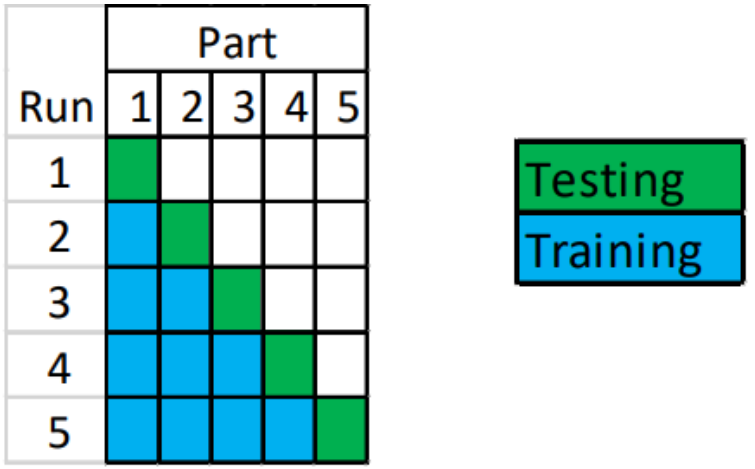
- Il classificatore è l'algoritmo che ci aiuterà a classificare la nostra classe come buggy oppure no
- Questo diventa più accurato più accurati sono i dati che gli diamo in pasto
- Se in input gli diamo dati sbagliati, le sue previsioni saranno sbagliate
- **Snoring**: una classe è snoring quando è affetta solo da difetti che non sono ancora stati fixati (FN)
- Per questo togliamo dal dataset la metà delle release più giovani, in quanto sono più probabilmente affette da snoring

Tecniche di Validazione

- Obiettivo: addestrare un classificatore con il nostro dataset in modo da poter predire le classi difettose
- Ma quale modello è più accurato in un determinato contesto?
- Possiamo usare una tecnica di validazione e vedere quale classificatore si comporta meglio in determinati casi
- Nel nostro caso abbiamo bisogno di una tecnica di validazione per le *time-series*, questo perché l'ordine dei dati conta
- Infatti non sarebbe corretto usare difetti in release future per predire difetti in release passate
- Per esempio, per predire la release 10 vogliamo usare dati che cronologicamente vengano prima della release 10

Walk Forward

- Il dataset viene diviso in *parti*.
- Una *parte* è la più piccola unità ottenibile che non può essere ulteriormente ordinata
- Ora le *parti* vengono ordinate cronologicamente, e ad ogni esecuzione tutti i dati che vengono prima della *parte* da predire vengono usati come *training set* e la *parte* da predire come *testing set*
- L'accuratezza del modello è calcolata come media sulle esecuzioni



Caso Studio

- Per il caso studio sono state comparate le metriche di performance di tre classificatori:
  - RandomForest
  - NaiveBayes
  - Ibk
- Utilizzando come tecnica di validazione Walk Forward
- Per migliorare le performance dei classificatori sono poi usate tecniche di *feature selection* e *cost sensitivity*

Feature Selection

- Sono tecniche che cercano di ridurre le colonne da dare in input al classificatore
- Lo scopo è quello di cercare di ridurre il costo del learning e fornire una migliore qualità di dati
- Esistono due approcci principali: *filtro* e *wrapper*
- Per il caso studio è stato scelto l'approccio con il filtro
- Gli approcci a filtro sono indipendenti dal classificatore che viene usato
- La logica che viene usata è quella di selezionare feature che abbiano un'alta correlazione con la variabile che si predice e una bassa correlazione con le altre variabili
- Quindi si vuole massimizzare la correlazione con la variabile da predire e minimizzare la correlazione interna tra le varie feature

Cost Sensitivity

- L'idea da cui parte questa tecnica è che non tutti gli errori sono uguali
- Nei normali classificatori viene predetto il caso positivo se la probabilità è maggiore del 50%, mentre viene predetto il negativo se la probabilità è minore del 50%
- Nel nostro caso farsi scappare una classe *buggy* e classificarla come *non buggy* (FN) e molto più costoso di classificare una classe *non buggy* come *buggy* (FP)
- Quindi vogliamo cambiare la percentuale del 50%; questo è quello che fa l'approccio di *thresholding*
- Assegniamo un costo ad ogni tipo di errore

TP	FN
FP	TN

0	CFN
CFP	0

- La threshold è CFP/(CFN+CFP)
- Nel nostro caso consideriamo CFN = 10 * CFP

Risultati - BookKeeper

Risultati – Discussione - BookKeeper

- Ibk
  - La tecnica di Cost Sensitivity non ha portato particolari migliorie alle metriche
  - La tecnica Feature Selection ha portato un evidente miglioramento ai punti della distribuzione sopra la mediana, infatti in tutte le metriche la parte superiore della scatola è più alta
  - Ricordiamo che Feature Selection riduce anche il costo del learning, quindi usarla in questo caso è sicuramente una scelta ottimale
  - Applicare Feature Selection + Cost Sensitivity da gli stessi effetti della sola Feature Selection
- Naive Bayes
  - Cost Sensitivity migliora sensibilmente la *precision*, mentre peggiora le altre metriche
  - Feature Selection migliora *precision* e *recall*, mentre peggiora di poco *AUC* e *kappa*
  - Cost Sensitivity + Feature Selection migliorano molto la *precision*, ma peggiora le altre metriche
  - Anche in questo caso Feature Selection sembra un buon compromesso tenendo conto il costo di learning
- Random Forest
  - Cost Sensitivity porta dei peggioramenti evidenti su *recall* e *kappa* e lascia sostanzialmente uguali precision e AUC
  - Feature Selection non evidenzia nessuna migliora sulle metriche, tuttavia teniamo sempre in conto il costo del learning
  - Cost Sensitivity + Feature Selection porta peggioramenti su *recall* e *kappa* mentre amplia la distribuzione della precision, anche nella parte bassa della distribuzione, quindi non è un miglioramento

Risultati - ZooKeeper

Risultati - Discussione - ZooKeeper

- Ibk
  - Cost Sensitivity non ha alcun effetto sulle metriche
  - Feature Selection migliora di poco la *precision* e il costo del learning
  - Cost Sensitivity + Feature Selection migliora ulteriormente la *precision*; in questo caso sembra essere la tecnica migliore
- Naive Bayes
  - Cost Sensitivity peggiora le metriche
  - Feature Selection migliora *precision* e *recall*, infatti le mediane sono più alte
  - Cost Sensitivity + Feature Selection da più o meno gli stessi risultati della sola Feature Selection
- Random Forest
  - Cost Sensitivity in questo caso aumenta di tanto la *precision*, infatti una mediana di circa 0,90 significa che la metà delle volte ha performato più di 0,90; tuttavia peggiora le altre metriche
  - Feature Selection migliora di poco la *precision*
  - Cost Sensitivity + Feature Selection da gli stessi effetti della sola Cost Sensitivity, alzando ulteriormente la mediana sulla *precision*
  - Feature Selection sembra essere la tecnica che si comporta meglio in questo caso

Link sonarcloud

- Codice per la creazione del dataset:
  - https://sonarcloud.io/project/overview?id=MetalIord97_DatasetCreator
- Codice per fare la valutazione tramite Walk Forward:
  - https://sonarcloud.io/project/overview?id=MetalIord97_DatasetAnalyzer