Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Triennale in Informatica

Tecniche di deep learning per il riconoscimento di errori nei programmi

Relatore:
Chiar.mo Prof.
Maurizio Gabbrielli

Presentata da:
Matteo Vannucchi

Sessione I
Anno Accademico 2021-2022

Abstract

Il ruolo dell'informatica, in un mondo in progressiva digitalizzazione di ogni singolo aspetto della vita dell'individuo, è ormai diventato chiave del suo funzionamento. Con l'aumentare della complessità del codice e delle dimensioni dei progetti, il rilevamento di errori diventa sempre di più un'attività difficile e lunga. Meccanismi di analisi del codice sorgente tradizionali sono esistiti fin dalla nascita dell'informatica stessa, e il loro ruolo all'interno della catena produttiva di un team di programmatori non è mai stato cosi fondamentale come lo è tuttora. Questi meccanismi di analisi però non sono esenti da problematiche: il tempo di esecuzione su progetti grandi e la percentuale di falsi positivi possono infatti diventare un grosso problema. Per questi motivi meccanismi fondati su *Machine Learning*, e più in particolare *Deep Learning*, sono stati sviluppati negli ultimi anni. Questo lavoro di tesi si pone quindi l'obbiettivo di esplorare e sviluppare un modello per il riconoscimento di errori in un qualsiasi file sorgente scritto in linguaggio sia C sia C++.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Capitolo 1

Introduzione teorica

1.1 Code2Vec

1.1.1 AstContext

Capitolo 2

Dataset

In questo capitolo tratteremo la generazione del dataset posto alla base del modello che andremo a creare poi nel **??**, vederemo prima il dataset originale utilizzato e poi come è stato aumentato tramite l'utilizzo di ulteriori analizzatori statici per migliorarne la precisione delle rilevazioni, andando a ridurre il numero di falsi positivi. Verrà poi presentato come le rilevazioni degli analizzatori statici sono utilizzate per la associazione fra un *code snippet* e il relativo errore, poi come da quest'ultimo venga ricavato il codice in formato di *ast context vector*.

2.1 Dataset originale

Come detto in precedenza questo dataset non è stato generato partendo da zero ma facendo riferimento al dataset creato da [1]. Il dataset consiste di circa 3000 progetti di GitHub, scritti in linguaggi C e C++, che rispettano due requisiti: hanno una licenza ridistribuibile e hanno almeno 10 stelle. Il secondo requisito ci serve per garantire che i progetti all'interno del dataset soddisfino dei requisiti di qualità, infatti come precedenti studi hanno mostrato (come ad esempio [2]) si può utilizzare il numero di stelle su GitHub come un *proxy* per la qualità del codice stesso.

Il dataset contiene per ogni progetto una serie di analisi effettuate: l'analisi di Doxygen che estrae le coppie codice-commento e l'analisi di Infer che produce un report di analisi statica degli errori. Visto l'utilizzo che ne sarebbe stato fatto di questo dataset l'analisi di Doxygen è stata scartata. In Figura 2.1 si può vedere la struttura tipica di uno dei circa 3000 progetti presenti.

Come si può notare ogni progetto contiene anche un Makefile, elemento fondamentale perché gli analizzatori statici che andremo ad aggiungere spesso richiedono l'esistenza di

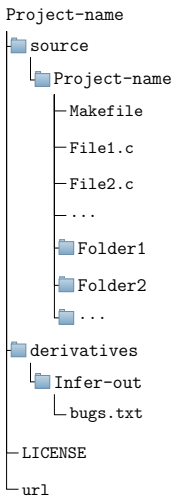

Figura 2.1: La struttura della directory di un progetto del dataset iniziale

un Makefile funzionante.

2.2 Analizzatori di codice statici

Un'analizzatore di codice è un programma che prende in input uno o più file e genera un report degli errori, cioè una lista di coppie del tipo <Errore, Posizione>. Di questi analizzatori ne esistono due macro categorie: statici e dinamici. Gli analizzatori statici sono programmi che effettuano controlli solo sul codice a livello testuale e che quindi non eseguono in nessuna maniera il codice, gli analizzatori dinamici sono invece analizzatori più complessi che effettuano controlli a *run-time* andando quindi ad'eseguire il codice stesso.

Gli analizzatori non sono però perfetti, infatti nell'insieme degli errori trovati si possono spesso trovare dei falsi positivi, cioè frammenti di codice segnati come erronei ma che in realtà non presentato nessun tipo di problema. Scopo appunto del dataset aumentato è quello di ridurre il numero di falsi positivi.

2.2.1 Analisi a livello di progetto

La maggior parte degli analizzatori statici inoltre è in grado di lavorare a livello di progetti, andando quindi a risolvere correttamente gli *include* (nel caso di C e C++), e quindi generando un output più significativo. Alcuni di questi per far ciò hanno bisogno

di quello che viene chiamato *compilation database* e, per soddisfare questo requisito, esistono strumenti appositi che utilizzano il Makefile per generarlo, nel caso di questo lavoro è stato utilizzato un programma chiamato Bear

2.2.2 Analizzatori ulteriori utilizzati

Come analizzatori statici ulteriori da aggiungere in più a Infer, di cui ogni elemento del dataset ha già l'analisi sua associata, sono stati scelti i seguenti tre:

- L'analizzatore Cppcheck che, a detta degli autori, ha come scopo principale il ridurre il numero di falsi positivi.
- Il compilatore GCC che nonostante sia un compilatore vero e proprio ha anche funzionalità per l'analisi statica dei programmi attraverso la flag *-fanalyzer*.
- Similarmente a GCC come terzo viene scelto il compilatore Clang che attraverso un suo tool chiamato Clang-Check è in grado di effettuare analisi statiche.

Non sono invece stati usati analizzatori dinamici, questo perché il loro utilizzo in modo automatizzato è un operazione complicata se non quasi impossibile. Infatti quasi tutti i programmi prendono o dei parametri all'esecuzione o degli input durante l'esecuzione, ma fornire questi dati in modo consistente e sensato per il programma e in modo automatizzato rende il tutto veramente difficile.

L'utilizzo di essi però potrebbe portare a risultati molto interessanti poiché parte dei falsi positivi degli analizzatori statici deriva dal non poter decidere se frammenti di programmi sono o non sono eseguiti e quindi gli analizzano tutti, ma quello che può succedere è che se in un frammento di programma che non viene sicuramente mai eseguito c'è un errore, l'analizzatore statico lo riferisce quello dinamico, correttamente, no.

Bibliografia

[1] Ben Gelman, Banjo Obayomi, Jessica Moore, and David Slater. Source code analysis dataset. *Data in brief*, 27:104712, 2019.

[2] Michail Papamichail, Themistoklis Diamantopoulos, and Andreas Symeonidis. User-perceived source code quality estimation based on static analysis metrics. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 100–107. IEEE, 2016.