Towards Automating Code Reviews

Muntazir Fadhel

[1]Department of Computing and Software
McMaster University

April 14, 2020

Problem Statement

Code reviews are an important practice for software quality assurance, but they are expensive.

Code Review Participation		
During the previous week how often did you..	How often do you act as a code reviewer?	How often do you author code reviews?
At least once per day	39%	17%
A couple times during the week	36%	48%
Once during the week	12%	21%
None	13%	14%

Figure: Results of a code review survey conducted at Microsoft [Mac+18]

At the same time however, a survey of over 600 companies found that 45% of code reviewers still feel as if they do not have adequate time to complete code reviews [COD]!

Existing Solutions

In the development of defect finding and code quality tools, research has been dominated by two main approaches:

1. **Logico Deductive**: Leverages the well-defined properties of programming languages to influence the design of development tools.
   - Static analysis tools, which utilize powerful abstractions, definitions, algorithms, and proof techniques.
2. **Data Driven**: Leverages statistical distributional properties estimated over representative software corpora to influence the design of development tools.
   - Probabilistic models of source code, designed to estimate a distribution over all possible source files
   - Machine learning models trained to classify source code embeddings.

Question: Can these types of tools be used as comprehensive code reviewing solutions?

Review: GitHub Pull Request Flow

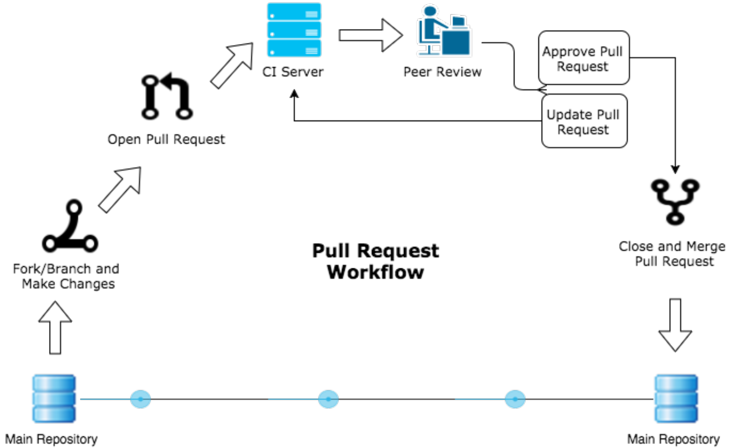


Figure: Illustrates a typical Pull Request workflow on GitHub

Review: Language Models

Goal

To build a model that can estimate the distribution of a language as accurate as possible.

The probability of a word sequence $W = w_0, w_1, w_2, ...w_N$ can be decomposed into cascading probabilities using the chain rule:

$$p(W) = p(w_0, w_1, w_2, w_3...w_N) = \prod_{i=1}^{N} p(w_i | w_{i-1}...w_1, w_0)$$

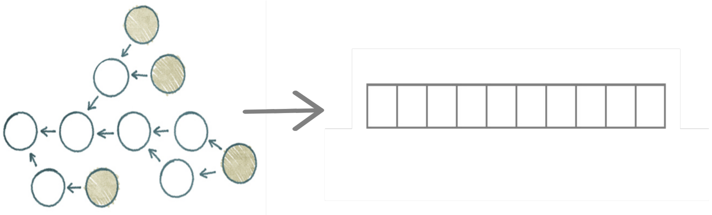
N-gram Language Models: Make the approximation that the probability of a word depends only on the identity of n preceding words:

$$n = 2 \rightarrow p(W) = \prod_{i=1}^{I} p(w_i | w_0...w_{i-1}) \approx \prod_{i=1}^{I} p(w_i | w_{i-1})$$

Machine Learning models typically operate over numerical data which makes it difficult to develop such models over source code directly.

Definition

Lower dimensional data structure that **preserves** properties and **relationships of interest** from the original graph.



Analyzing GitHub Code Review Comments

Developed an SVM classifier with an accuracy of 92.5% in classifying code review comments in one of 13 categories.

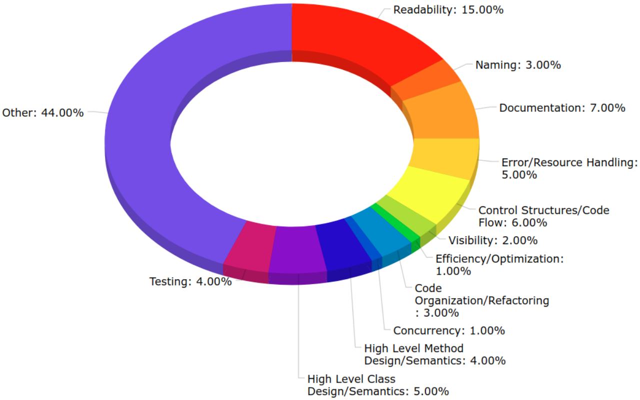
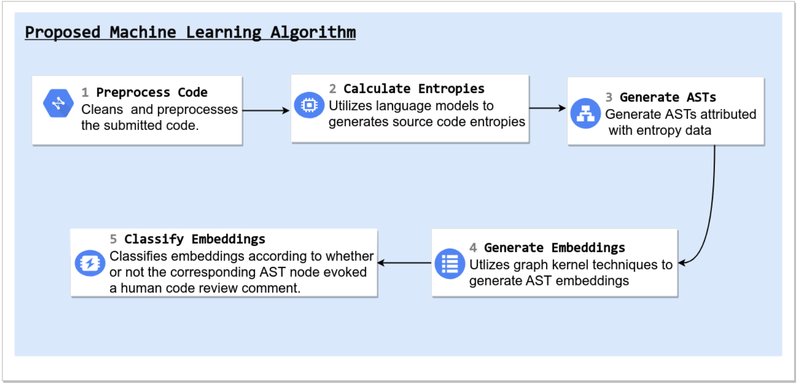


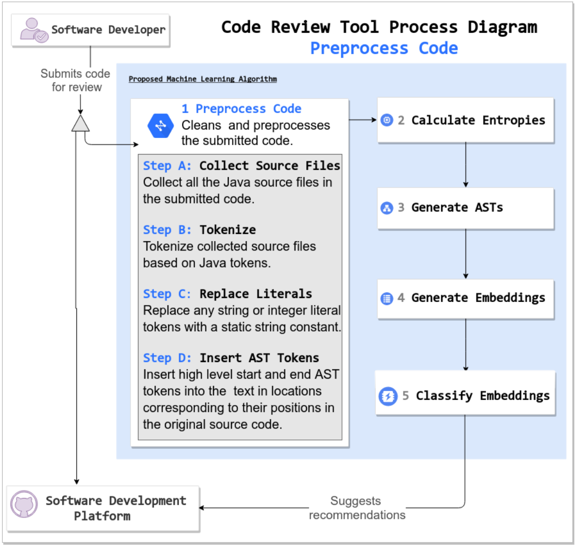
Figure: Results of classifying 32,000 comments mined from GitHub

Proposed Algorithm Overview

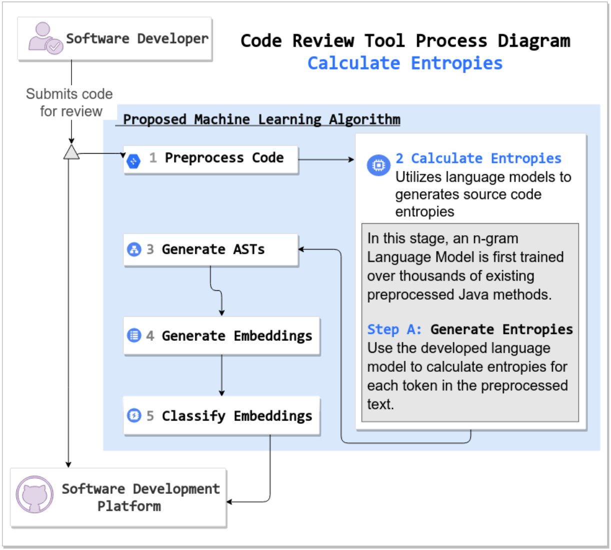
Goal: To train a machine learning model to learn human-like code inspection behaviour in flagging suspicious code.



Proposed Algorithm: Stage 1 - Preprocess Code



Proposed Algorithm: Stage 2 - Calculate Entropies

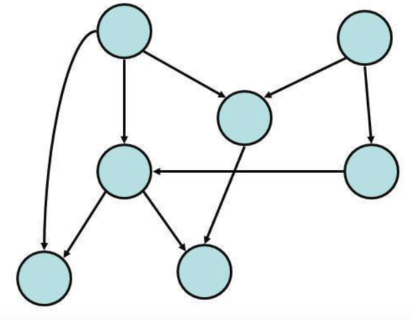


Proposed Algorithm: Stage 3 - Generate ASTs

In this stage, Abstract Syntax Trees (AST) are generated from the preprocessed code

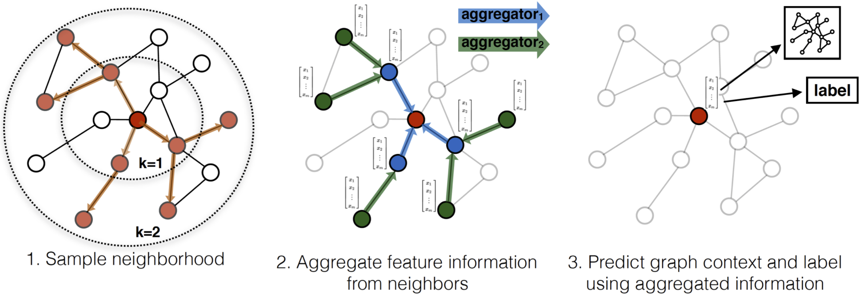
Additionally, nodes in each AST are **attributed** with the following values:

1. Entropy
2. Corresponding Line Number
3. Entropy Standard Deviation
4. Added
5. Commented



Proposed Algorithm: Stage 4 - Generate Embeddings

In this stage, generated AST's are converted into embeddings using the GraphSage algorithm [HYL17].



Proposed Algorithm: Stage 5 - Classify Embeddings

In this stage, the embeddings that were generated from AST nodes are classified using a Neural Network (NN) as to whether the code representation of the node evoked a human code review comment or not.

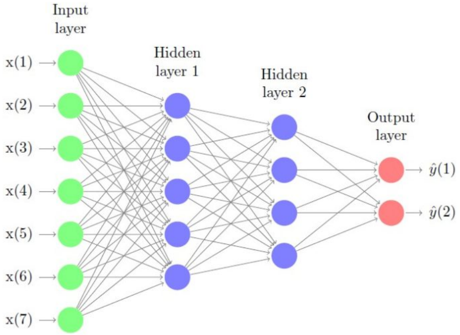


Figure: An example of a Feed-Forward Neural Network (FFNN) with two hidden layers, and two output states.

Experimental Setup

Experimental data set consists of source files mined from Pull Requests in 1572 repositories on GitHub.

Attribute	Count
Source Files	59,130
Total AST Nodes	36,873,189
Non-commented AST Nodes	36,833,803
Commented AST Nodes	39,386
Commits	48,438
Lines of Code	28,853,416

- Entropy Calculation: **5-gram** language model trained over top 200 most forked Java GitHub repositories
- GraphSage Configuration: Embedding Size=**64**, Hops=**5**, Sampling Size=**10**
- Embedding Classifier Configuration: Optimizer=**Adam**, Hidden Layers=**1**, Epochs=**10**, Batch Size=**100**

Experimental Results: Classification Metrics

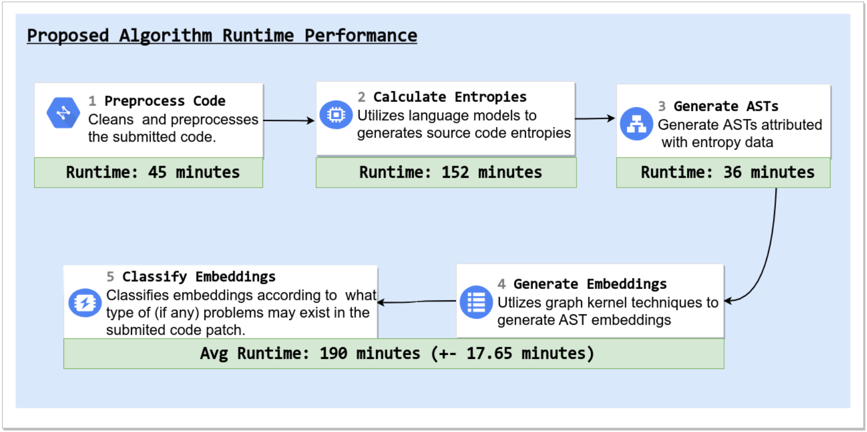
Final embeddings are partitioned into train-test, 80%-20% splits, over which the the developed NN classifier is trained and evaluated respectively.

Class	Precision	Recall	F_1 Score	Support
Non-commented	94.31	84.52	89.14	7,191,250
Commented	23.45	67.80	34.85	7,264
Macro	58.88	76.16	**62.00**	

Table: Classification metrics summarizing the proposed algorithm's performance on the test data set averaged over 10 runs.

Note: Simple accuracy measures are **not** sensitive to class imbalance, which is why the F_1 Score has been chosen to represent classification effectiveness.

Experimental Results: Runtime Performance



Reviewing Behaviour Replicated By Our Algorithm

Figure: Code reviewing behaviour that was replicated by the proposed algorithm over a code snippet [2].

However, there were also over 5 other instances found in which throwing general exceptions weren't considered problematic by the same reviewer.

- The experimental tool **matched** these behaviours as well!

[2] Sourced from https://github.com/ballerina-platform/ballerina-lang/pull/9481

Discussion

- Code reviewer inspection behaviour varies greatly between reviewers, likely a result of differences in:
  1. Experience
  2. Technical Ability
  3. Available Time
  4. Software Development Philosophy
  5. Mood
  6. Reviewee
- Code reviewers themselves are also inconsistent
  - Human error, changes in software development knowledge
- Algorithm effectiveness would likely improve with more data, which is difficult to collect using code review comments alone.

Bibliography

CODACY. *Codacy*. URL: https://www.codacy.com (cit. on p. 2).

William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216 (cit. on p. 12).

L. MacLeod et al. "Code Reviewing in the Trenches: Challenges and Best Practices". In: *IEEE Software* 35.4 (July 2018), pp. 34–42. ISSN: 0740-7459. DOI: 10.1109/MS.2017.265100500 (cit. on p. 2).