Using Natural Language Processing to Determine Version Control Commit Authors

Cameron Little
CSCI 404 Winter 2015
Computer Science Department
Western Washington University
Bellingham, WA 98225
littlec8@students.wwu.edu

Abstract

This paper describes the approaches and results of using natural language processing to determining authorship of commits in version control systems based on the message provided alongside the commit. Several commonly used algorithms were used, both through commonly used toolkits and custom implementations. The training and test data used were pulled from differing sized open source projects on GitHub and an private repository with a broad history of contributors.

1 Introduction

Today, virtually all software development projects of significant scope are maintained using some form of version control. These systems allow tracking changes throughout the life of a project and attributing those changes to the people responsible. Often, due to issues with workflow or a developer's environment, this attribution is incorrect. This causes issues when code is being reviewed, or when old code is being revisited.

For example, when a new developer is tasked with updating some portion of a codebase, they often run into places where the decision making process is poorly documented and reach out to the original author to request more information. In cases where attribution information is incorrect, this process becomes more difficult.

This project explores how natural language processing can be used to help identify the author of any commit.

1.1 Git

I chose git as the version control system to analyze. This is primarily due to the fact that I am most familiar with git out of any version control system. My implementation needs to interact with a repository, and prior knowledge of command syntax and output formats reduce the time needed to set up a framework for working with commits. Second, git is widely used. Any derived products from this project will have a wide potential user base. Finally, many open source projects use git, which gives me a large amount of sample data. Repositories from the website GitHub are used for most of the training data.

2 Related Work

The general goal of this project is to determine authorship of text based on short, domain specific messages. I was unable to find any projects or publications directly related to version control and natural language processing, but a number of papers are available that analyze both SMS messaging and Twitter posts. Many of these are concerned with classifying features such as the topic from the message contents, rather than the individual producing the text. For example, one paper was concerned with how SMS and Twitter are used differently, as well as determining subject (Munro, 2012).

The most relevant prior work was conducted by Green (2013) on identifying authors of Twitter tweets based on features derived from tweet contents. Green preformed several different experiments with the goal of comparing two different feature sets. The researchers ran into many of the same issues I did and had comparable results.

In the broader scope of natural language processing, author identification has been the subject of much research, thought the documents being classified have been of a much broader form and over a wider variety of topics.

3 Data

I chose three primary sets of data, all public repositories on GitHub, and ran my programs on sev-

eral auxiliary repositories throughout my testing and coding.

The largest data set I chose was Twitter's Bootstrap CSS Framework, available at `https://github.com/twbs/bootstrap`. Bootstrap has, at the time of writing, 11,075 commits from 628 contributors. The topics of commits range over cleanup, documentation, linting (code standards), javascript, css, html, and accessibility.

The second data set I tested is the Atom Package Manager (APM), a package manager for the open source text editor, Atom. APM has, at time of writing, 1,373 commits from 33 contributors. APM can be found online at `https://github.com/atom/apm`.

I also tested on a private GitHub repository I have access to containing a large scale web application's frontend code, called CCLWeb. CCLWeb has 752 commits from 5 contributors, so is faster to test on.

3.1 Data Challenges

Many of the challenges in this project arise from the particularities of the data being evaluated. Most version control repositories have several common features that make evaluation problematic.

First, a small set of individuals generally have "ownership" over a specific codebase and are the authors of the majority of the commits. I've used this as a baseline metric: my goal is to beat not only random guessing, but guessing the most prolific committer. This can skew many metrics by making the dominant factor in the probability of any one author their percent of contribution.

As previously mentioned, messages are generally short, an average of two sentences. This significantly reduces the number of features available to extract from a message.

Another challenge is the lack of diversity in commit messages. Because they concern work around a specific repository, the topic and keywords tend to be similar across authors within a single repository.

4 System Description

I conducted four different training and testing algorithms on all three data sets. All three are implemented in Python 2.7 with heavy use of object oriented design and memoization. Computationally heavy components are saved to disk using Python's pickle module to save time in testing.

Each classifier uses the committer's author as a label.

4.1 Algorithms

Naive Bayes The first algorithm I implemented is simple Naive Bayes multiple classification. The initial code is based upon a prior assignment, but modified to better accommodate the testing environment. This algorithm uses a custom tokenizer primarily based on whitespace and punctuation removal. A dictionary of the top 2500 words are used as features. A feature set is computed for each author in the training data.

To determine authorship, the probability of the commit message as a tokenized sequence is computed for each feature set, and the one with the highest probability is chosen.

Naive Bayes reduces text into a set of feature probabilities for each class. When testing, the sample text's feature probabilities are computed and compared to the known class probabilities. The closest class is chosen.

Trigrams The second algorithm tested was an ngram based model. I chose trigrams as a relatively efficient balance between speed and complexity. My approach computes the frequencies of trigrams appearing among tokens within the training data per author. A model is generated for each author in the training data. The probability of a test message is computed for each model, weighted by the percent of commits made by that author. The most probable is chosen. Initially I didn't weigh commits by percent, but the model always chose the most common author from training as the committer for every test case.

My trigrams model uses a very simple whitespace tokenizer.

Ngrams compute the likelihood of any series of n tokens appearing. For a given test sequence, the probabilities of each ngram appearing is computed per class, and the most likely class is chosen.

NLTK The next algorithm I used was NLTK's naive bayes classifier. NLTK is a Python Natural Language ToolKit that offers a broad range of proven implementations of common NLP algorithms.

My NLTK approach first generates a frequency distribution of tokens within a commit message using `nltk.FreqDist`. Tokens

are generated using Python's `split()` function as suggested in the NTLK documentation. I then train and test using the `nltk.classify.NaiveBayesClassifier`.

Top Committer My final algorithm is the most simple and is my baseline. The email associated with the highest amount of commits is always guessed when classifying test data.

This algorithm makes it's best guess based on the fact that those who commit most are most likely to be any given commit's author.

4.2 Experiment

My python code is comprised of several modules, which contain several classes each. A small help message can be printed by running `python main.py -h`. Each test can be specified explicitly, or all tests can be run. The final, non-named argument is used as a path to a git repository from which commits are used to train and test.

I chose a ratio of 3:1 for training data vs testing data. Commits are randomly ordered; the first three quarters are used to train and the rest to test. The order is retained in a pickled file for results comparison. Commits are reshuffled on deletion of the file.

5 Results

Overall, none of the algorithms achieved great accuracy, but did provide interesting results. As can be seen in Table 1 on page 4, my custom implementation of Naive Bayes was the only algorithm to outperform the baseline Top Commenter algorithm. Both trigrams and NLTK's Naive Bayes preformed very poorly, with NLTK's Naive Bayes the worst.

6 Discussion

I expected far better results from both Trigrams and NLTK.

I expected Trigrams to capture the grammatical and sentence structure better than simple token frequency. Because of the similarities in content, I suspected that sentence structure would play a larger role in identifying authorship. I ran a few quick tests with higher n ngrams but saw a reduction in accuracy.

Due to NLTK's acceptance in the NLP community, I expected it's implementation to outperform mine, but it did far worse. I suspected this could have to do with the simple tokenization I was doing, so I tested using the same tokenizer used in my Naive Bayes implementation with no significant improvements.

The real challenge with this specific application of natural language processing is the limited variety of data each repository has. Features that are generally considered clutter, such as stop words and punctuation, may be more significant to a unique person than general word choice.

6.1 Future Work

I suspect my implementation could be improved significantly through tailoring each phase of my process toward the situation. Tokenization could be improved by including sentence features (such as sentence ending punctuation, breaks, and style). The algorithms I used to test were quite general. More specific and advanced algorithms exist that would be better suited for the form of text I'm analyzing.

References

Rachel M. Green and John W. Sheppard. 2013. *Comparing Frequency- and Style-Based Features for Twitter Author Identification*. The John Hopkins University, Baltimore, MD.

Robert Munro and Christopher D. Manning. 2012. *Short message communications: users, topics, and in-language processing*. Stanford University, Stanford, CA.

	Bootstrap	**APM**	**CCLWeb**
Naive Bayes	45.55%	64.52%	78.19%
Trigrams	13.17%	15.54%	28.19%
NLTK	7.05%	17.01%	20.21%
Top Committer	25.94%	62.76%	61.70%

Table 1: Results