S E 492 Final Report

Sdmay23-20: Automated Annotation of Source Code for ML Applications

Robby Rice, Tanner Dunn, Amon McAllister, Max Sutcliffe, Gavin Canfield

Intro

This document outlines topics related to our senior design project. Our project was Automated Annotation of Source Code for Machine Learning Applications. The primary objective was to build a dataset of labeled code syntax with corresponding metadata:

- Token: String representation of the code
- Label: Clasifier of the token
- Level: Relative abstract placement of the token separated into 3 levels (low, medium, high)

Secondarily, we tried out a few machine learning techniques to further our understanding of the dataset and the outlined purpose of helping understand the black box of code-to-code machine learning.

Requirements

Functional Requirements	- End product must include annotation of source code. - Automated annotation tool must correctly chunk and label sections of code. - Dataset must be in a feasible format to be used for ML purposes
Resource Requirements	- If we end up training a neural network, sufficient GPU power is needed to train large models and evaluate performance.
Aesthetic Requirements	N/A
User Experience Requirements	- Labels must be either clearly marked and understood or explained.
Economic Requirements	N/A
Environmental Requirements	N/A
UI Requirements	- Documentation on how to run various scripts must be provided.

Standards

IEEE 2841-2022: This standard is called the “IEEE Approved Draft Framework and Process for Deep Learning Evaluation.” We must eventually analyze the processes and performance of our algorithm if we end up creating a new one. This standard will help in creating the framework and infrastructure we need to evaluate it.

Security Concerns

We don't have security concerns related to our dataset or ML pipeline. The only security concerns are related to the open-source dataset creation tool. We must ensure that our tool isn't used for malicious purposes or contains vulnerabilities that will impact our codebase.

Design Evolution

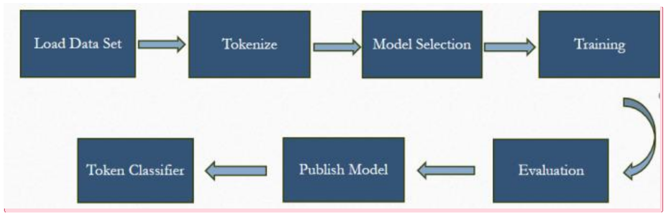
Our first design iteration included our dataset creation tool. This tool statically took in hard-coded files and produced a dataset. This tool evolved by making file input more dynamic and making our code more containerized by separating functionality into many classes.

To prepare for the completed dataset, we started by implementing an end-to-end ML Pipeline on a similar problem, NLP chunking. NLP chunking deals with sentences and then aims to label parts of speech, such as nouns, proper nouns, adjectives, etc. This is like our Dataset, as we have code chunks and their corresponding labels. Once the dataset was completed, we tuned the ML pipeline to take in our dataset, train the model, and create a custom token classifier.

Implementation and Design

Our dataset creation tool was created with Java and used the JavaParser library to create and Abstract Syntax Tree of code sample. We then visit each of the AST nodes to create a JSON object of each sample. Each of these JSON sample objects are added to a file to create our dataset. We add different properties to the object such as depth level, id then the token and the corresponding level. The structure of the dataset is an array of json objects, with each file being an object, then within that object we have each token/ label pair that we found in the AST tree created.

To implement the machine learning pipeline, we used the Hugging Face library. Below highlights our design process to implement our Machine Learning model. Starting with Loading in our dataset made by our dataset creation tool. Next, we tokenized this dataset, selected a model from the hugging face library and trained the model on our data. From here we evaluated the accuracy of the model and published the model. For the last step we implemented a custom token classifier that allows a user to input a token and our trained model returns a label and it's estimated accuracy score.



Commented [DC1]: Do we want this diagram in here?

Testing

We created unit tests for the code dataset creation tool. Other parts of our code weren't tested but were subject to code reviews.

For the Machine Learning Pipeline, we split our data into training and testing samples. We then would train on multiple models in the hugging face library and implemented the model had the highest accuracy.

Related Work

Applications of this work can relate to code-to-code tools. One of the most popular tools in this space right now is GitHub Copilot. This tool is integrated into your IDE and uses AI to autocomplete and create code from a prompt.

Future Work

Future work for this project will be handed over to other PhD candidates for further development and to collect more insights from our ML pipeline. Due to the complexity and size of the model, training the model on the entire dataset was not accomplished. The next steps would be to get access to the HPC and train the model on the full datasets and evaluate the results. With our dataset creation tool, we also created different datasets based on two different processes. First, just to get an initial dataset, we created levels of labels ambiguously. We separated the labels out in a high, medium and low-level manner, based on what we felt fit where. The second dataset we created had a bit more systematic approach. The quote-on-quote 'level' has now become the depth of the AST tree. This means that the user can choose what depth of the tree they are getting tokens at. This approach can yield more predictable results as we will see the same labels at the same depths. This will give the PhD candidates the opportunity to compare the different datasets and criteria's effect on the accuracy of the model.

Due to the complexity and size of the model we are training our dataset on, training it on a personal machine on the full dataset is not feasible and running the model on the HPC will be required. With late access to the HPC we were not able to get to this and this will be one of the next steps so the model can be evaluated on the entire dataset.

Appendix I: Operation Manual

To use our dataset creation tool, do the following:

- Copy source code from our GitLab
- Make sure Java, Maven, and your choice of Java IDE is installed
- Once project is loaded, make sure there are .java files in your resources folder to run off of
- Build, compile, and run the Maven project
- Output files will be in a dataset.json file

To run the machine learning model, do the following:

- Copy Source code from GitLab
- Open the HuggingFaceWithDataset.ipynb
- When prompted for a hugging face token enter the following token: ‘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’
- The end of the notebook will have a working token classification interface for you to use!

Commented [DC2]: @Amon if you could give simple instructions on running the model that would be great. Nothing Complex and doesn't have to be perfect.

If you want just the token classifier

- Get the source code from GitLab
- Open **UIforModel.ipynb**
- Run the entire ipynb
- At the bottom there is token classification interface for you to use!

Appendix II: Alternative Designs

Dataset Creation Tool

The dataset relied on use producing labels for code tokens in java.

Alternative design dataset creation tool: We initially tried to produce all our own labels for java code. We had an excel sheet that was mapping common java code i.e. ‘if statements’, ‘class declarations’ to our own custom labels. We soon realized that producing custom labels for every token in java including libraries would take far too long and be very cumbersome.

Dataset

Alternative design Dataset: We were going to use an existing data set that was created for CodeSyntax. This data set had their custom labels and had the tokens broken up and their position tracked. We were going to add custom leveling for the labels. We also wanted to add their labels to the tokens. We were thinking we could break the file up into distinct levels, some tokens would be considered high, some low, based on our discretion.

Commented [SF3]: @Rice, Robert E I forget exactly why we didn't end up going this route.

Commented [RE4R3]: I think we wanted a more custom solution, or that we were going to use this dataset if we couldn't figure out problems in the dataset creation code but I think we got them worked out

Alternative ML Approach

Commented [DC5]: @McAllister, Amon K I remember you talking a bit about this if you could fill out this section?

An alternative ML approach we could have used was multi-class sentiment analysis. While this may not have worked for all cases, it would have been significantly worse for the smaller chunks of code and better for the high-level labels that tend to be longer. For one ML approach to cover the needs of all the data of different lengths and complexities was overly optimistic. I hypothesize that having different approaches for the differing code lengths would have been beneficial and increased accuracy. Doing sentiment analysis on the longer text would have solved the issue with the tokenizer breaking up the longer text when that was not the desired outcome. Furthermore, the data we used for one ML could be used again for the other just trimmed to the appropriate size for what we are trying to predict.