Analyzing Software using Deep Learning

Bug Detector

Project Report

Gokul Kannan – 3429345

University of Stuttgart

1. Project Goal:

The main goal of this project is to classify between **correct** and **incorrect** if conditions in the provided JavaScript source code using deep learning techniques. A binary classifier model has to be created in python to complete this task. Lines which are identified as bugs, shall be reported for future operations such as automatic correction or to raise warnings in an IDE.

2. Available Resources:

1. 10000 json files to train the model each consists of,
   - JavaScript source code (Dictionary key: "raw_source_code")
   - **Abstract Syntax Tree (AST)** representation (Dictionary key: "ast")
   - Sequence of Tokens form the source code (Dictionary key: "tokens")
   - **Important Note :** Provided dataset does not has any incorrect if conditions.
2. Fasttext encoding model (.bin) file.

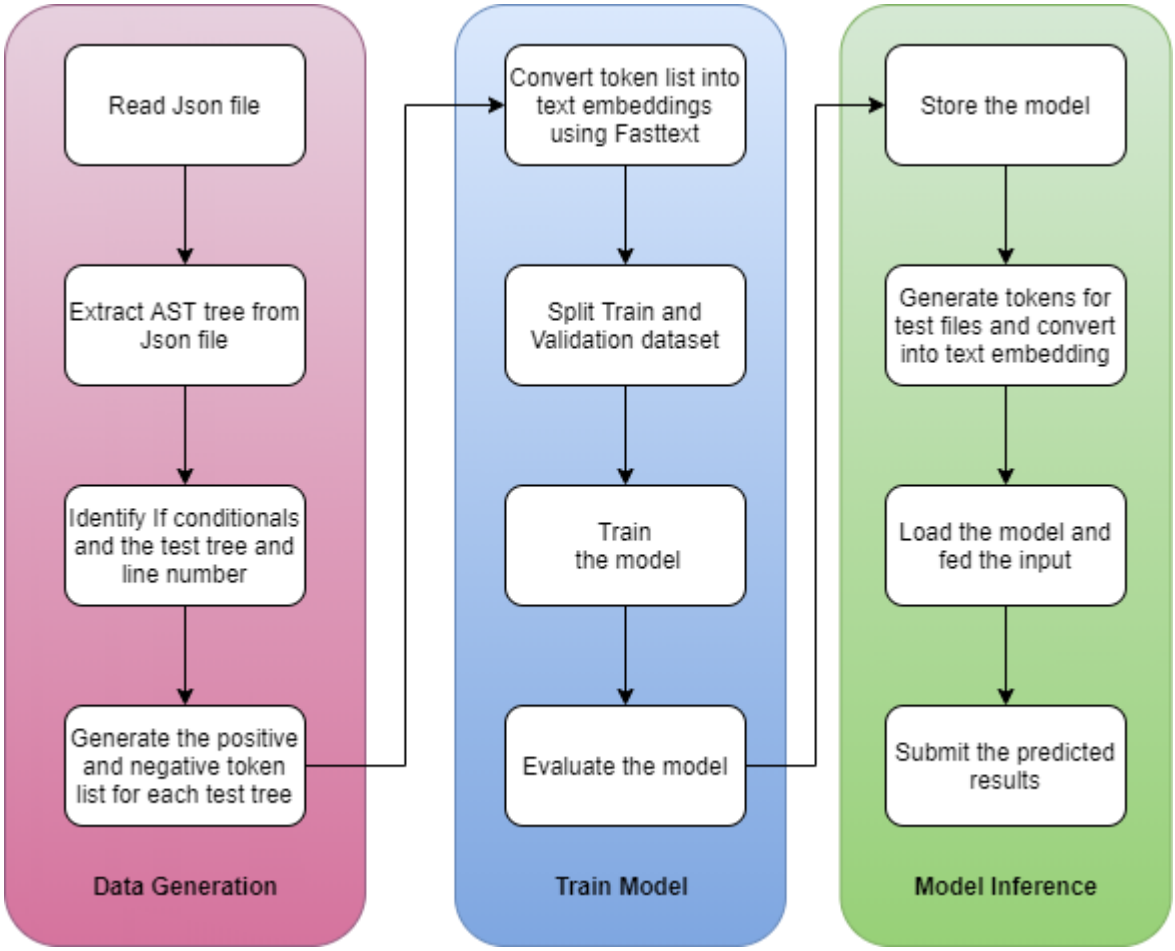
3. Software Requirements:

- The binary classifier has to developed with the following software specifications only
  - Python 3.8
  - Pytorch 1.5.0
  - and some other additional 3rd packages if needed
  - Fasttext text encoding (other sophisticated encoding shall also be used)

4. Project Requirement:

- Identify **5 types of bugs** (mentioned in section 7) in the provided JavaScript source code and **report the line number** in a dictionary format written in an output json file.
- The file name is the key and the list of line number of bugs is the values in the dictionary.
- The bug types are only limited to **If condition** and the following **else if** conditions.
- If some "if conditions" extend to more than a single line, the starting line number shall be reported as bugs
- The provided dataset does not have any incorrect code, so in order to train a balanced network generate negative test case (with Bug) from the correct code.

5. Software Architecture:



6. Approach:

- In this approach, I had used the provided AST tree format to extract the JavaScript source code information by traversing the AST
- The AST tree of all if condition was only considered for token generation
- The **test** tree of **IfStatement** was parsed into string tokens based on the type of expression
- Incorrect samples were generated from the Correct samples based on the expression type and the bug type mentioned in the Project details.
- Token list exceeding a predefined maximum length will be removed from the list
- "**_END**" string was added at the end of the token to mark the end the condition.
- This list of string tokens was encoded using **Fasttext** text embedding.
- Split the obtained token into train and validation sets. (80: 20 split)
- Finally, the complete train dataset tensor is fed to the deep learning model for training
- The model was evaluated with unseen validation dataset.

- The **hyper parameters** such as batch size, epoch, learning rate, optimizer and number of neurons were tuned based on the accuracy and loss values on unseen data (validation data) obtained with different value pairs.
- It was found that with batch size = 64, epochs = 10, and 100 hidden neurons the algorithm performed better.
- After this evaluation, the model is trained with complete set of data (both train and validation dataset)
- The model is stored and will be loaded back during inference.

7. Bug Types:

Type 1	**Incomplete conditional expression**
Description	Existing conditions that misses a sub expression
Correct condition	if (tokenList && tokenList.length === 1) { … }
Incorrect condition	if (tokenList.length === 1) { … }
Correct token	[“Identifier”, &&, “length”, “===”, 1, “_END”]
Incorrect token	[“length”, “===”, 1, “_END”]

Type 2	**Incorrectly ordered Boolean expression**
Description	Logical expression that are not commutative
Correct condition	if (tokenList && tokenList.length) {…}
Incorrect condition	if (tokenList.length && tokenList) {…}
Correct token	[“Identifier”, &&, “length”, “_END”]
Incorrect token	[“length”, &&, “Identifier”, “_END”]

Type 3	**Wrong Identifier**
Description	using incorrect identifier in a conditional expression
Correct condition	if (isArrayLike (obj1, obj2)) { obj2 = obj1}

Incorrect condition	if (isArrayLike (**wrong_id**, obj2)) { obj2 = obj1}
Correct token	[“isArrayLike”, “**obj1**”,”obj2”, “_END”]
Incorrect token	[“isArrayLike”, “**wrong_id**”,”obj2” “_END”]

Type 4	**Negated Condition**
Description	Any expression or sub expression with a negation operator
Correct condition	1. if (**!**data) {} 2. if (obj && **!** data) {}
Incorrect condition	1. if (data)) {} 2. if (obj && data)
Correct token	1. [“**!**”, ”data” “_END”] 2. [“obj”, “&&”, “**!**”, ”data” “_END”]
Incorrect token	1. [”data” “_END”] 2. [“obj”, “&&”, ”data” “_END”]

Type 5	**Wrong Operator**
Description	Binary operator that is used inside a conditional expression
Correct condition	if (index **<** tokenList.length) {…}
Incorrect condition	If (index **>** tokenList.length) {…}
Correct token	[“index”, **<** , “length”, “_END”]
Incorrect token	[“index”, **>**, “Identifier”, “_END”]

Some examples of change in operator

Correct operator	**Wrong Operator**		**Correct operator**	**Wrong Operator**		**Correct operator**	**Wrong Operator**		**Correct operator**	**Wrong Operator**
>	<		!=	==		&	&&		+	-
<	>		==	!=		|	||		-	+
>=	<=		===	!==		<<	>>		*	/
<=	>=		!==	===		<<	>>		/	*

8. Model Architecture:

Layer	Input	Output	Activation
LSTM	100	100 hidden neurons	Default
Drop out	100	20 %	None
Linear	100	64	Tanh
Linear	64	1	Sigmoid

BugFinderModel(

(lstm): LSTM(100, 100, batch_first=True)

(dropout): Dropout(p=0.2, inplace=False)

(linear_1): Linear(in_features=100, out_features=64, bias=True)

(tanh_1): Tanh()

(linear_2): Linear(in_features=64, out_features=1, bias=True)

(output): Sigmoid()

)

Model Parameters:

- Batch Size : 64
- Epochs : 10
- Optimizer: Adam (learning rate = 0.001)

9. References:

1. Pradel, Michael & Sen, Koushik. (2018). DeepBugs: a learning approach to name-based bug detection. Proceedings of the ACM on Programming Languages. 2. 1-25. 10.1145/3276517.
2. http://software-lab.org/teaching/summer2020/asdl/
3. https://esprima.org/demo/parse.html
4. https://github.com/austinbyers/esprima-ast-visitor
5. https://esprima.readthedocs.io/en/latest/
6. https://github.com/michaelpradel/DeepBugs