

Software Engineering

Group Project Phase 2- Project Update

Roopa Chowdary Cherukuri-RC23M Beligini Keerthana-KB23U Likhita Ganipineni-LG23H

Introduction:

In the rapidly evolving landscape of natural language processing and artificial intelligence, understanding the quality and usability of code generated by language models has become a crucial area of research. This investigation delves into three distinct yet interconnected research questions, each shedding light on different facets of the code generation process by ChatGPT.

The paper examines the quality of code from ChatGPT by utilizing code analysis tools like linters to identify common issues, contributing to discussions on AI-driven code generation capabilities. It also explores if developer interactions with ChatGPT can predict the likelihood of code integration into a production environment, offering insights into the practical use of AI-generated code. The paper investigates how the quality of ChatGPT-generated answers changes during a conversation, revealing potential improvements or discrepancies in subsequent iterations and highlighting the dynamic nature of AI-generated responses.

Together, the paper contributes to a comprehensive examination of the capabilities, challenges, and implications of ChatGPT in the context of code generation. The findings hold significant implications for both the research community and the broader software development industry, shaping our understanding of the strengths and limitations of AI-driven code generation tools.

Research Questions Addressed:

1. What types of quality issues (for example, as identified by linters) are common in the code generated by ChatGPT?
2. Can we forecast whether a developer will incorporate the code in a production environment by analyzing their interactions with ChatGPT during a conversation?
3. How does the quality of answers regenerated by ChatGPT differ from the ones generated initially?

Research Question1: (Roopa Chowdary)

Dataset used: 20231012_235320_discussion_sharings.json

Methodology implemented till now:

Step 1: Define Quality Metrics: Establish criteria, such as indentation, variable naming conventions, and syntax rules, to objectively assess code quality. This framework sets predefined rules for a comprehensive evaluation.

Step 2: Selection of Linter

Choose a linter, an automated code analysis tool tailored to specific programming languages, to identify issues, enforce coding standards, and enhance code consistency.

Step 3: Automated Analysis

Apply the selected linter systematically to each code file in a dataset using a script or program. This automated process efficiently identifies issues and deviations from defined quality metrics, generating reports for streamlined code review and assessment on a larger scale.

Pending Methodology:

Step 4: Ranking and Prioritization: Quantify and record the number of issues identified by the linter, initiating the ranking and prioritization of code quality issues based on their prevalence.

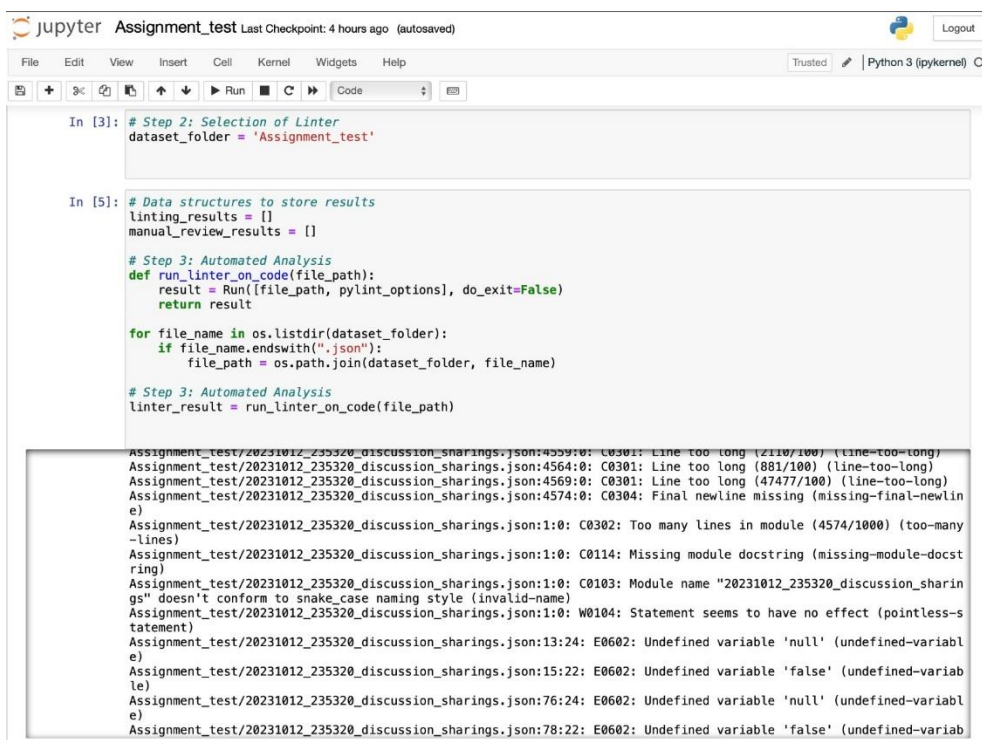
Step 5: Manual Review: Introduce a manual review process triggered by a predefined threshold, enhancing the assessment of code quality by incorporating human judgment.

Step 6: Data Labeling: Categorize code snippets based on the severity of identified issues, assigning labels such as 'high' for enhanced structuring and prioritization.

Step 7: Training a Machine Learning Model

Implement a classification model such as naïve bayes to predict severity labels. Evaluate the model's performance using metrics like accuracy and classification reports for a comprehensive understanding of code quality assessment.

Results:



```
jupyter Assignment_test Last Checkpoint: 4 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O

In [3]: # Step 2: Selection of Linter
dataset_folder = 'Assignment_test'

In [5]: # Data structures to store results
linting_results = []
manual_review_results = []

# Step 3: Automated Analysis
def run_linter_on_code(file_path):
    result = Run([file_path, pylint_options], do_exit=False)
    return result

for file_name in os.listdir(dataset_folder):
    if file_name.endswith('.json'):
        file_path = os.path.join(dataset_folder, file_name)

# Step 3: Automated Analysis
linter_result = run_linter_on_code(file_path)

Assignment_test/20231012_235320_discussion_sharings.json:4559:0: C0301: Line too long (2110/100) (line-too-long)
Assignment_test/20231012_235320_discussion_sharings.json:4564:0: C0301: Line too long (881/100) (line-too-long)
Assignment_test/20231012_235320_discussion_sharings.json:4569:0: C0301: Line too long (47477/100) (line-too-long)
Assignment_test/20231012_235320_discussion_sharings.json:4574:0: C0304: Final newline missing (missing-final-newline)
Assignment_test/20231012_235320_discussion_sharings.json:1:0: C0302: Too many lines in module (4574/1000) (too-many-lines)
Assignment_test/20231012_235320_discussion_sharings.json:1:0: C0114: Missing module docstring (missing-module-docstring)
Assignment_test/20231012_235320_discussion_sharings.json:1:0: C0103: Module name "20231012_235320_discussion_sharings" doesn't conform to snake_case naming style (invalid-name)
Assignment_test/20231012_235320_discussion_sharings.json:1:0: W0104: Statement seems to have no effect (pointless-statement)
Assignment_test/20231012_235320_discussion_sharings.json:13:24: E0602: Undefined variable 'null' (undefined-variable)
Assignment_test/20231012_235320_discussion_sharings.json:15:22: E0602: Undefined variable 'false' (undefined-variable)
Assignment_test/20231012_235320_discussion_sharings.json:76:24: E0602: Undefined variable 'null' (undefined-variable)
Assignment_test/20231012_235320_discussion_sharings.json:78:22: E0602: Undefined variable 'false' (undefined-variable)
```

Research Question2. (Beligini Keerthana)

Dataset used:

Methodology implemented till now:

Data Loading and Preprocessing:

- Loaded the JSON file and Extracted prompts and answers from the JSON data.
- Created a new CSV file with prompts and answers.

Feature Engineering:

Performed sentiment analysis on the "Answer" column using VADER sentiment intensity analyzer.

Calculated the length of each conversation ("Answer") as a feature.

Combine TF-IDF vectors of prompts and answers with sentiment, conversation length, and user experience as features.

Sentiment Analysis:

Split the data into training and testing sets. Trained a DummyClassifier.

Evaluated the sentiment classifier's accuracy and generated a classification report.

Original Classification Task:

Extracted features and labels for the original classification task

Used the same features for the original classification task.

Initialised and just started training the random forest classifier for the original task.

Pending Methodology:

Training the RandomForest Model and calculating its performance accuracy, p. recision, recall, F1-score, and ROC AUC.

Results:

Sentiment Analysis - Random Classifier

Accuracy: 0.3684210526315789

Classification Report:

	precision	recall	f1-score	support
0	0.14	0.38	0.20	8
1	0.69	0.37	0.48	30
accuracy			0.37	38
macro avg	0.41	0.37	0.34	38
weighted avg	0.57	0.37	0.42	38

Research Question3:

Methodology:

Using the NLTK library and difflib module, this Python code computes evaluation metrics such as BLEU score and similarity ratio between original and regenerated answers. The code is broken down as follows:

1. Importing Required Libraries: The code imports pandas for data management, nltk's 'sentence_bleu' for calculating BLEU score, and difflib's 'SequenceMatcher' for computing similarity ratio.

2. Loading Dataset: It assumes there is a CSV dataset with columns named 'original_answer' and 'regenerated_answer'. This dataset is read into a pandas DataFrame.

3. Defining Functions:

- 'calculate_bleu(reference, candidate)': This function accepts a reference and a candidate sentence and computes their BLEU score using the NLTK's 'sentence_bleu' function.

- 'calculate_similarity(reference, candidate)': This function computes the similarity ratio between the reference and candidate sentences using the SequenceMatcher from difflib.

4. Evaluation: It creates two empty lists to record the calculated scores ('bleu_scores' and 'similarity_ratios').

- It uses 'zip' to run over each pair of original and regenerated responses, calculating BLEU scores and similarity ratios for each pair.

- The calculated scores are added to the appropriate lists.

5. Adding Evaluation Results to DataFrame: The estimated BLEU scores and similarity ratios are added to the existing DataFrame 'df' as new columns ('bleu_score' and 'similarity_ratio').

6. Results Printing: Finally, it displays the average BLEU score and similarity ratio across all answers in the DataFrame.

This code essentially computes the similarity between original and regenerated answers in a dataset using the BLEU score (a commonly used metric for evaluating machine-translated sentences) and a similarity ratio based on character-level similarity using SequenceMatcher.

Pending Methodology:

For now, we have a plan on how to approach this question, so to explain that we have given an example in a simple manner. The data cleaning and preprocessing is not done on the given data as of this point so extracting the data and applying this to obtain the desired results would be our priority in the coming days and after that we must do the same process by using the dataset.

Example:

```
1 # Input two numbers from the user
2 num1 = float(input("Enter first number: "))
3 num2 = float(input("Enter second number: "))
4
5 # Find the minimum of the two numbers using the min() function
6 minimum = min(num1, num2)
7
8 # Print the minimum value
9 print(f"The minimum of {num1} and {num2} is: {minimum}")
10
```

Enter first number: 34
Enter second number: 3
The minimum of 34.0 and 3.0 is: 3.0
> |

```
1 def find_minimum(a, b):
2     return min(a, b)
3
4 # Input two numbers from the user
5 num1 = float(input("Enter first number: "))
6 num2 = float(input("Enter second number: "))
7
8 # Find the minimum of the two numbers using the function
9 minimum = find_minimum(num1, num2)
10
11 # Display the minimum number
12 print(f"The minimum of {num1} and {num2} is {minimum}")
```

```
Enter first number: 34
Enter second number: 3
The minimum of 34.0 and 3.0 is 3.0
>
```

Both approaches achieve the same result: they take two numbers from the user and find the lowest value among them. The main distinctions are in code style, variable naming, and minor variances in user interaction or output formatting.