VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Winter Semester 2024-25

J- Component Final Report

COURSE CODE: CSI4001

COURSE NAME: Natural Language Processing and Computational Linguistics

PROJECT TITLE:

"Autonomous Tagging of Stack Overflow Questions"

GUIDED BY:

Prof. Sharmila Banu K

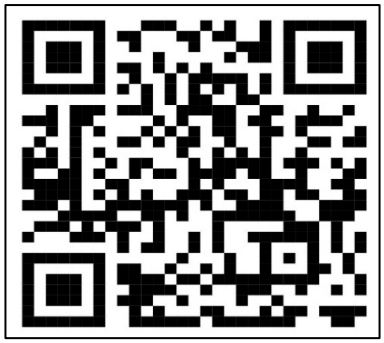
TEAM MEMBER DETAILS:

1. Poorvi Rai - 21MID0243
Took lead on core development, built the machine learning pipeline, and prepared the report.

2. Keshav Sharma - 21MID0182
Contributed to coding and assisted in debugging and preprocessing tasks.

3. Moudipa Jana - 20MID0056
Designed the web interface and created the final presentation used for review.

J – Component GitHub Repository



1. Introduction

Stack Overflow is a highly frequented platform where developers ask technical questions and provide answers. These posts are tagged with relevant topics to help organize content and improve searchability. However, many new users fail to label their questions correctly, resulting in misclassification and reduced visibility. This project focuses on building a system to automatically assign appropriate tags to newly posted questions using machine learning. The goal is to improve content discoverability and ensure that queries are categorized effectively.

2. Problem Statement

The major issue addressed in this project is the incorrect tagging of questions by users on Stack Overflow. Manually assigned tags are often either irrelevant or too generic, leading to poor organization and search performance. By automating the tagging process based on the content of the title and body of the question, the system aims to provide more accurate and consistent categorization.

3. Modules Overview

This project is divided into several key modules, each responsible for a specific function in the workflow from input to output.

3.1. Preprocessing Module

This module prepares the input text for analysis. The clean_text() function standardizes contractions and removes unnecessary spaces. Then, remove_punctuation() and lemitizeWords() tokenize the input and reduce words to their base form using lemmatization. Common stop words are removed using stopWordsRemove(). All these steps are integrated in preprocess_text() to ensure consistency.

3.2. Feature Extraction Module

After preprocessing, the text needs to be converted into a format suitable for machine learning models. This is achieved using a pre-trained TF-IDF vectorizer

(tfidfVectorFinal.pkl). The feature vectors generated from the title and body are combined using hstack() from scipy.sparse.

3.3. Model Prediction Module

This module loads a trained multi-label classifier (clfFinal.pkl) and uses it to predict tags based on the combined feature vectors. The binary predictions are then translated into readable tags using MultiLabelBinarizer.

3.4. Web Interface Module

The Flask-based web interface enables user interaction. The '/' route accepts input and returns predictions in JSON, while the '/output' route displays the results in a formatted HTML page. Flask's session object is used to temporarily store the prediction results for continuity between pages.

3.5. Result Handling

For convenience and session-level history, the predicted tags, title, and body are stored in both the session object and a list named Total[]. This helps retain a running history of processed questions during a session.

4. Packages Used

4.1. NLTK

The Natural Language Toolkit (NLTK) was primarily used for text preprocessing. It provided the WordNetLemmatizer to reduce words to their base forms and the list of English stop words, which were filtered out to improve the relevance of input data during model training and prediction.

4.2. re and string

The re (regular expressions) and string libraries were essential for normalizing raw text. They helped in expanding contractions (e.g., "don't" to "do not"), removing unwanted characters, and stripping punctuation, making the text cleaner for vectorization.

4.3. scikit-learn

Scikit-learn played a central role in both feature extraction and model training. The TF-IDF Vectorizer converted the cleaned text into numerical form. It also provided tools to train the multi-label classifier and convert predicted label vectors back into readable tag names using the MultiLabelBinarizer.

4.4. pickle

The pickle module was used for model persistence. It allowed us to save and later load the pre-trained classifier and TF-IDF vectorizer efficiently, ensuring that the prediction process is fast and does not require retraining.

4.5. scipy.sparse

From SciPy, the sparse module was specifically used for the hstack() function. This allowed us to merge the separate TF-IDF vectors generated from the question title and body into a single feature vector, which could then be passed to the classifier.

4.6. Flask

Flask served as the lightweight backend web framework. It handled routing between different pages (such as input and output screens), processed user inputs, and managed session-level storage for prediction results, making the tool user-friendly and interactive.

5. Methodology and Architecture

The project follows a structured pipeline from user input to final tag prediction. Each step is modular and supports reusability and scalability.

5.1 User Input and Interface

Users interact with the system through a simple Flask-based web interface. A form allows them to enter the question title and body. Once submitted, this input is passed to the backend for processing.

5.2 Text Preprocessing

Before feeding the input into the model, it is cleaned and standardized to remove noise:

- Punctuation is stripped from the text.
- Words are lemmatized using WordNetLemmatizer to reduce them to their base form.
- Common stop words are filtered out to retain only meaningful terms.

This step is implemented in a dedicated preprocessing script to keep the workflow modular.

5.3 Feature Vector Generation

After preprocessing, the text is transformed into numerical vectors using a pre-trained TF-IDF vectorizer. Separate vectors are created for the title and the body. These are then combined into a single feature vector using hstack() from scipy.sparse, forming the final input to the classifier.

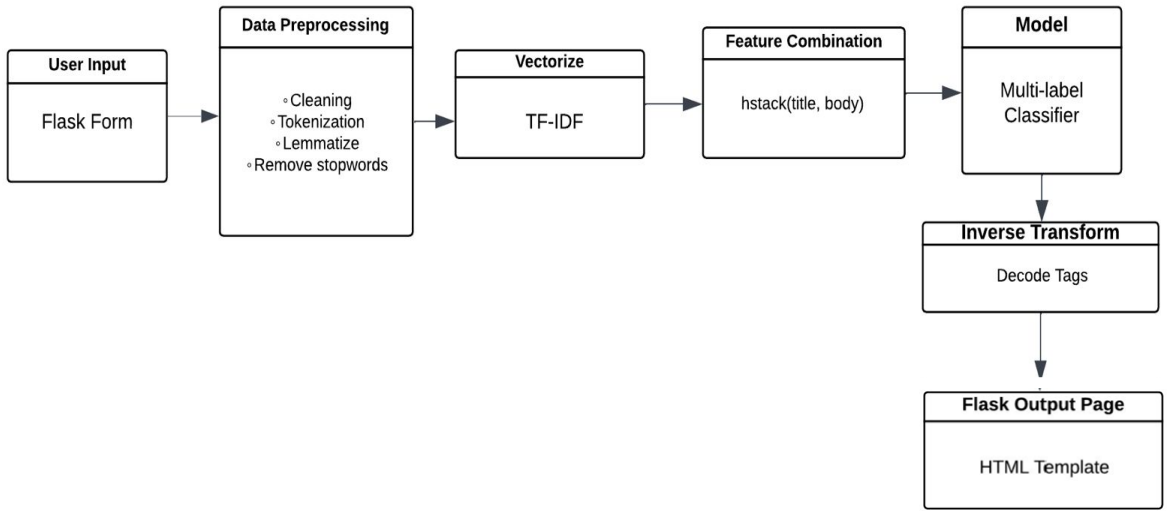
5.4 Model Loading and Prediction

The combined feature vector is passed into a trained multi-label classification model loaded via pickle. The classifier predicts a binary array indicating the presence or absence of each tag. This binary output is decoded back into human-readable tags using MultiLabelBinarizer.

5.5 Displaying the Results

Predicted tags are stored using Flask's session object. These tags, along with the original question title and body, are displayed to the user on a results page (output.html). Additionally, predictions are saved in an in-memory list for session-level history tracking.

5.6 Architecture Diagram:



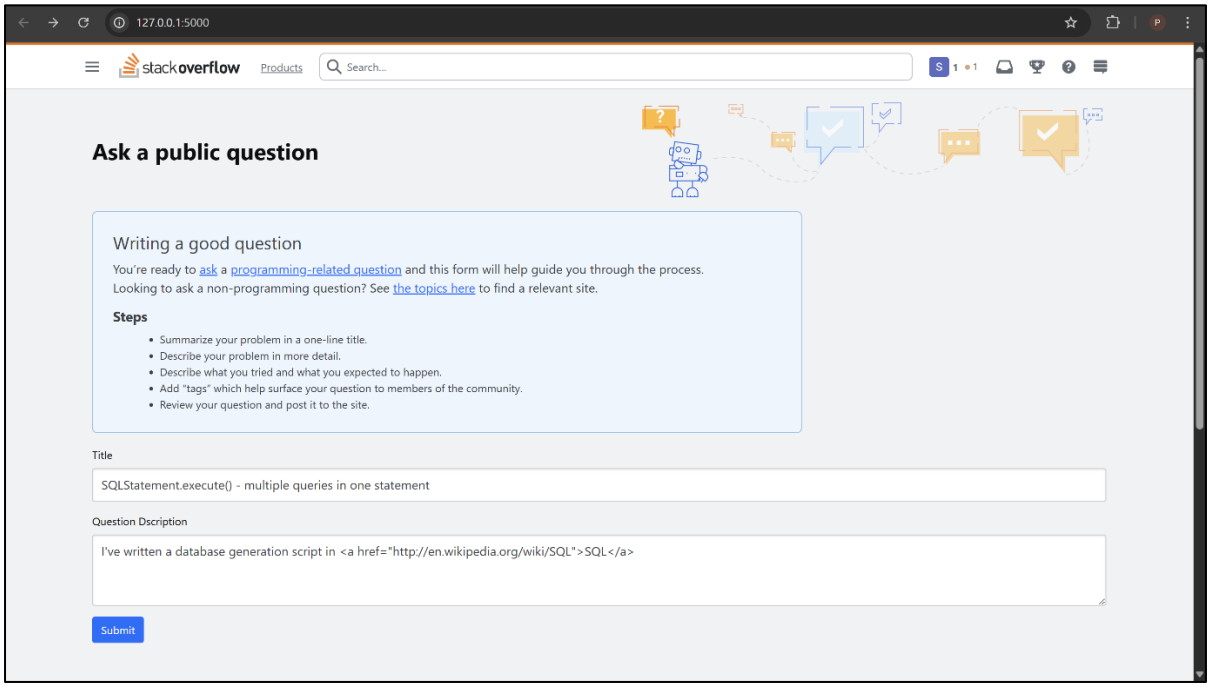
6. Results

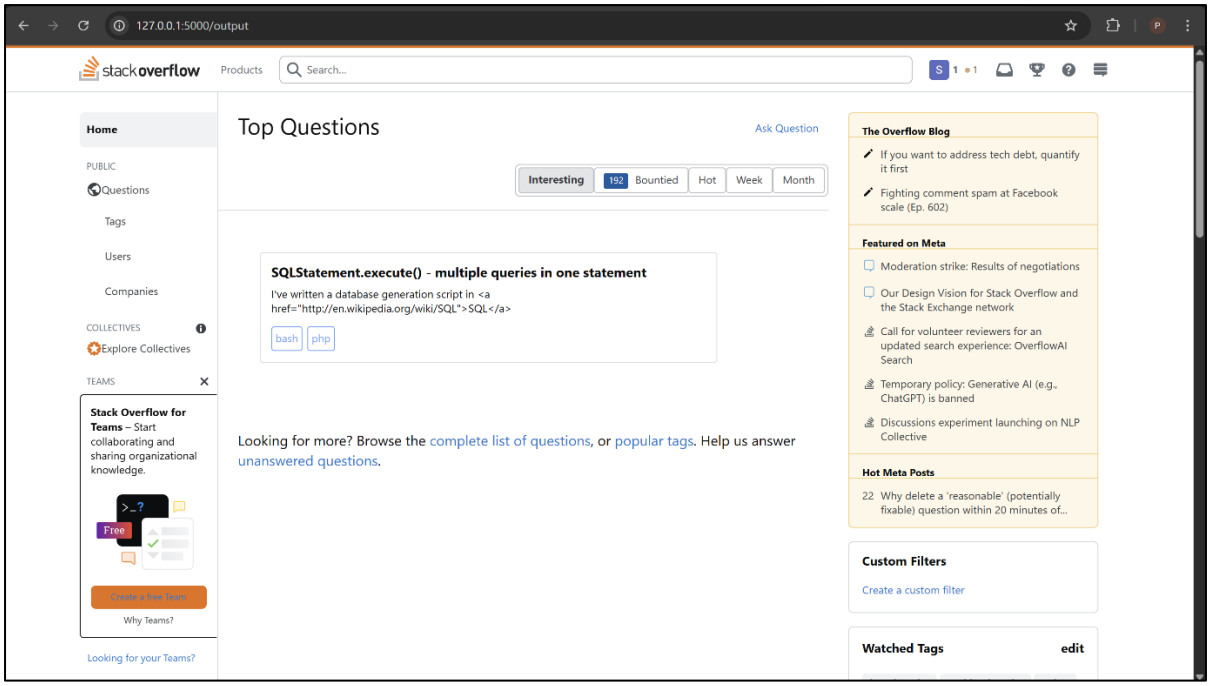
Example Input:

Title: "SQLStatement.execute() – multiple queries in one statement"
Body: "I've written a database generation script in <a href> style SQL."

Predicted Tags:

['bash', 'php']





The model was able to extract meaningful tags from the question body, despite the presence of HTML tags and less-structured input. The predictions were consistent with actual Stack Overflow tags.

7. Constraints and Limitations

The model can only predict tags from a predefined set of tags that were used during training. This means that any new or rare tags not part of the training set cannot be predicted.

Vague or poorly phrased questions may lead to inaccurate predictions. Ambiguities in the question's phrasing can confuse the model, resulting in incorrect or missing tags.

The current implementation does not interface directly with the live Stack Overflow platform. This limits the model's ability to fetch real-time data or integrate with the Stack Overflow database for up-to-date tag recommendations.

The accuracy of the model is dependent on the quality of preprocessing and feature extraction steps. Any shortcomings in these steps can reduce the overall accuracy of the tag predictions.

8. Challenges and Fixes

The dataset was imbalanced, which affected the model's ability to predict some tags effectively. To address this, we applied stratified sampling and class balancing techniques during model training.

There was a noticeable prediction lag, especially with large inputs. This was resolved by pre-loading the vectorizer and classifier with pickle, reducing the processing time significantly.

Tag ambiguity, where similar tags like python and python-3.x were confused, was another challenge. We resolved this by limiting the tag space to the most frequent and distinct tags, which improved the model's consistency and accuracy.

References

1. https://scikit-learn.org/stable/
2. https://www.nltk.org/
3. https://flask.palletsprojects.com/
4. https://towardsdatascience.com/multilabel-text-classification-done-right-using-scikit-learn-and-stacked-generalization-f5df2defc3b5/