CLOWiz: An NLP-Based Question Categorization Application

CLOWiz: An NLP-Based Question Categorization Application

Natural Language Processing

Prepared for Mr. Shahzeb Khan
Lecturer
Computer Science Department
FAST-NUCES PWR

Prepared by
Farouq Haider (20P-0091)
Adda Hussain Qazi (20P-0488)
Aabideen (20P-0006)
Shahzaib Niaz (20P-0558)
Hassaan Waheed (20P-0474)
Arslan Mumtaz (20P-0143)
Nabeel Yaseen (20P-0486)
Munzir Kalim Ahmed (20P-0477)
Javeria Naeem (20P-0456)
Muhammad Shaheer (20P-0480)
Muhammad Umar (20P-0062)
Azhar Jadoon (20P-0631)

March 29, 2024

Contents

Chapter 1

Technology Used

1.1 Frontend Development:

1.1.1 React:

A JavaScript library for building client-side interfaces. React was used to develop the frontend components, including the main "Sort" component, and "Result" component.

1.1.2 HTML/CSS:

HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) were used for structuring and styling the frontend user interface, along with the "style.css" file for additional styling.

1.2 Backend Development:

The backend development follows the Model-View-Controller (MVC) pattern, which typically involves technologies such as:

1.2.1 Server-side Framework: Flask

For our project, we exclusively utilized Flask as the server-side framework, responsible for handling server-side logic and interactions.

1.3 Language Processing

This section delves into the code's language processing functionalities and the libraries utilized within:

Libraries Used in Code:

The code snippet relies on the following libraries for its language processing tasks:

- `flask` - for web application development
- `flask_cors` - for handling Cross-Origin Resource Sharing (CORS) in Flask

- `re` - for regular expression operations
- `pandas` - for data manipulation and analysis
- `nltk` - Natural Language Toolkit for text processing
- `sklearn` - for machine learning and natural language processing utilities

These libraries collectively support functionalities like text preprocessing, vectorization, and similarity calculations.

Code Functionality:

The code implements features for text preprocessing, vectorization, and cosine similarity calculations. It establishes a Flask route (`/predict`) to handle user input and generate predictions based on TF-IDF vectorization and cosine similarity.

1.4 Testing:

1.4.1 Cypress:

A JavaScript end-to-end testing framework for testing web applications.

1.4.2 React Testing library:

A testing library for React applications, used for functional and layout testing.

1.5 Documentation:

1.5.1 Latex:

A typesetting system used for creating structured documents such as reports or research papers.

Table 1.1: Technologies used Table

Technology Name	**Used for**
React.JS	Frontend
Flask	Backend
Cypress and RTL	Testing
Google Colaboratory (Python)	Model
Figma	UI Modeling
Latex and Google Slides	Reporting

Chapter 2

General Overview

2.1 Problem Statement:

In educational institutions, instructors often need to categorize questions based on Course Learning Outcomes (CLOs) to ensure alignment between course objectives and assessments. However, manually categorizing questions can be time-consuming and prone to errors. Automating this process can streamline assessment creation and improve instructional quality.

Overview Approach to Solving the Problem Statement

1. Problem Understanding

- Understand the requirements and objectives of the problem statement.
- Identify the main tasks involved in the project, such as processing Course Learning Outcomes (CLOs), categorizing questions, and integrating frontend and backend components.

2. Data Preparation

- Gather and preprocess the necessary data, including Course Learning Outcomes (CLOs) descriptions and questions.
- Clean the data by removing noise, such as special characters and irrelevant punctuation.
- Tokenize the text data into individual units, such as words or phrases.
- Normalize the text by converting all tokens to a uniform case (e.g., lowercase).
- Lemmatize the words to reduce them to their base or dictionary form.

3. Feature Extraction

- Utilize a pre-trained Sentence Transformer model to generate embeddings for both the CLO descriptions and the questions.
- Convert the text data into numerical form (embeddings) to capture semantic similarities between the CLOs and questions.

4. Model Training

- Train a machine learning model, such as a Support Vector Machine (SVM) classifier, to categorize questions based on their similarity to the CLO descriptions.
- Use the embeddings generated in the feature extraction step as input features for the model.
- Assign each CLO description an index to serve as labels for the model.
- Evaluate the effectiveness of the trained model using metrics such as accuracy.
- Perform hyperparameter tuning to optimize the model's performance.

5. Backend Development

- Implement a backend system following the Model-View-Controller (MVC) pattern.
- Develop APIs or endpoints for processing CLOs and questions.
- Integrate the trained model into the backend system to classify questions based on CLO similarities.
- Implement database management to store and retrieve data as needed.

6. Frontend Development

- Develop frontend components using React, including the main "App" component, "Sort" component for inputting CLOs and questions, and "Result" component for displaying relevant CLOs.
- Design user interfaces for inputting CLOs and questions and viewing the results.
- Utilize HTML/CSS for structuring and styling the frontend components.

7. Testing

- Configure testing frameworks such as Cypress and React Testing Library (RTL) for functional and layout testing.
- Create and execute tests locally or through automated pipelines to ensure the proper functionality and appearance of the application.

2.2 Implementation Details:

Table 2.1: Implementation Details

Frontend	
Components	We have developed an app in React consisting of three components: the main "App" component, where we define the routes for the pages; the "Sort" component, where we design screens for inputting CLOS and questions; and the "Result" component, where we design screens for displaying CLOS-based questions. Additionally, we have a "style.css" file for designing the screens. Once these components and styles are completed, we pass them to the backend developer for integration with the model and database.
Backend	
Integration	Our implementation approach is streamlined: Flask facilitates communication with our Python model, with a "/predict" endpoint managing interactions from the React client. Through API calls triggered by the client, CLOs and questions are sent for sorting. Leveraging NLP techniques from the Model Team, our predict method efficiently sorts questions based on provided CLOs. Recognizing the Model code's requirement for a significant volume of additional questions during runtime, we optimized for efficiency by circumventing the training process. Instead, we intelligently extracted and integrated a modified version of the TF-IDF methodology from their code, eliminating the need for additional question input.
Model/AI	
Training Steps	Model training involves several critical steps, beginning with data preparation. This initial phase entails cleaning, tokenizing, normalizing, and lemmatizing the text data to ensure it's in an optimal format for processing. Following this, we employ a pretrained Sentence Transformer model for feature extraction, generating embeddings for both the Course Learning Outcomes (CLO) descriptions and the questions. These embeddings are crucial for capturing semantic similarities between text passages. The core of the process is training a Support Vector Machine (SVM) classifier, which uses the embeddings of the CLO descriptions as input features and their corresponding indices as labels, allowing the model to classify questions based on their similarity to the CLO descriptions. To evaluate the effectiveness of the trained SVM model, we use metrics such as accuracy, ensuring the model's efficiency in predicting the most relevant CLOs for given questions. Additionally, we perform hyperparameter tuning with methods like GridSearchCV to further enhance the model's performance. Once the model has been satisfactorily trained and evaluated, it is saved using joblib for future deployment, ensuring readiness for application in predicting relevant CLOs based on questions.

Testing	
Testing Steps	The application was tested using Cypress for end-to-end testing and React Testing Library (RTL) for unit testing. Cypress was used to validate the application's functionality as a whole, simulating user interactions and ensuring the expected behavior across different pages and components. RTL, on the other hand, focused on testing individual components in isolation to verify their functionality and behavior. This combined testing approach provided comprehensive coverage, enhancing the overall quality and reliability of the application.
Report Writing	
Introduction	The documentation team has created and maintained clear and comprehensive documentation covering all phases of development. The deliverables include every report draft submitted as well as all presentation slides that have been created to date. They ensure that all aspects of the project are well-documented to facilitate understanding, collaboration, and maintenance throughout the project life cycle for stakeholders and team members.

2.3 Basic Workflow

1. Open the frontend CLOWiz Panel with the Sort.jsx form.
2. Enter the CLOs, their descriptions, and the questions in the correct format.
3. Click on Sort. This will send an API call to our Flask backend.
4. The backend separates our CLOs' descriptions from their names and uses the code supplied by the Model Team to categorize the provided questions to their relevant CLOs.
5. The result will then be returned and stored in the browser, and the Result.jsx page will be shown with the categorized CLOs.
6. You can now choose to Sort CLOs again.

Chapter 3

Any Other Relevant Information

3.1 Model Training Approach

3.1.1 Data Preparation

- Objective: Prepare and preprocess text data to ensure it's in a suitable format for machine learning models.
- Steps:
  - Cleaning: Remove noise such as special characters, irrelevant punctuation, and numbers that don't contribute to understanding the CLOs or questions.
  - Tokenization: Break down the text into individual units (tokens) such as words or phrases.
  - Normalization: Convert all tokens to a uniform case (e.g., lowercase) to ensure consistency.
  - Lemmatization: Reduce words to their base or dictionary form (lemma), helping the model to recognize variations of the same word as a single entity.

3.1.2 Feature Extraction

- Objective: Convert text data into numerical form (embeddings) to capture semantic similarities between the CLO descriptions and the questions.
- Tool: Use a pre-trained Sentence Transformer model to generate embeddings for both the CLO descriptions and the questions. This step converts the text into vectors that represent their semantic meaning.

3.1.3 Model Training

- Objective: Train a machine learning model that can classify questions based on their similarity to the CLO descriptions.
- Model: Support Vector Machine (SVM) classifier.
- Process:
  - Use the embeddings generated in the Feature Extraction step as input features.

- Assign each CLO description an index to serve as labels.
- Train the SVM classifier to associate questions with the relevant CLO index based on the embeddings' similarities.

3.1.4 Evaluation

- Objective: Assess the effectiveness of the trained SVM model in predicting relevant CLOs for given questions.
- Metrics: Use accuracy and potentially other metrics like precision, recall, and F1 score to evaluate model performance.

3.1.5 Hyperparameter Tuning

- Objective: Optimize the model's performance by fine-tuning its hyperparameters.
- Technique: Employ GridSearchCV or similar methods to systematically explore a range of hyperparameters, finding the best combination for the SVM model.

3.1.6 Model Deployment

- Objective: Make the trained model available for future use in predicting the most relevant CLOs for new questions.
- Tool: Save the trained SVM model using joblib or a similar serialization library for easy loading and inference in the deployment environment.

3.2 Input and Output

Input

- Questions
- CLO with description

Output

- Sorted Questions in respective CLO's

Chapter 4

Team Segregation

4.1 Team Members and Roles

Table 4.1: Team Segregation: Clearly defines the roles and responsibilities of each team member.

Team members	**Roles**
Farouq Haider (20P-0091)	Project Manager/Backend Engineer
Aabideen (20P-0006)	AI Engineer
Shahzaib Niaz (20P-0558)	AI Engineer
Hassaan Waheed (20P-0474)	AI Engineer
Arslan Mumtaz (20P-0143)	UI/UX Designer
Nabeel Yaseen (20P-0486)	Frontend Engineer
Munzir Kalim Ahmed (20P-0477)	Frontend Engineer
Adda Hussain Qazi (20P-0488)	Backend Engineer
Javeria Naeem (20P-0456)	Application Tester
Muhammad Shaheer (20P-0480)	Application Tester
Muhammad Umar (20P-0062)	Report Architect
Azhar Jadoon (20P-0631)	Report Architect