Automated Text Classification

Shubharthak Sangharasha

The University of Adelaide

4533_COMP_SCI_7417_7717 Applied Natural Language Processing

Lecturer: Dr. Orvila Sarker

Table of Contents

1. Abstract **2**

2. Introduction **2**

2.1. Project Goal 2

2.2. Data Source and Collection Stratergy 3

3. Automated Text Classification **3**

3.1. Cleansing Techniques 3

3.2. Processed Columns 4

4. Graphical Representation of the Data **5**

4.1. Word Cloud 5

4.2. Tag Frequency 6

4.3 Temporal Trends 6

4.4 Engagement Metrics 7

5. Data Categorisation **7**

5.1. Categorisation Statergy 8

5.2. Category Definitions and Examples 8

5.3 Categorisation Results 9

6. Implementation Details **10**

6.1 System Architecture 10

6.2 Key Libraries 12

6.3 Execution 12

7. Conclusion **13**

1. Abstract

Natural Language Processing (NLP) comes with its own challenges for software developers tasks. Online forums like Stack Overflow (SO) can provide valuable resources, yet using the enormous number of posts can be difficult and time-consuming. This project developed an organized NLP Knowledge Base by scraping, cleaning, visualizing, and classifying NLP-related posts from Stack Overflow.

This was targeted on posts tagged with using the Stack Exchange API. We cleansed, normalized, and tokenized all retrieved data. Visualizations like word clouds, tag frequency plots, and temporal analysis helped derive key insights. This involved categorization of posts based on relevant keywords, tasks, types of questions, and libraries used, making it easier to find relevant info.

We built a web application for the above to be traversable by simply displaying all of these categories, visualizations, and knowledge base using Flask as a backbone of the web application. The end-product is a system that helps developers expedite needs to comprehend common NLP problems and solutions discussed over the Stack Overflow community.

2. Introduction

2.1 Project Goal

As software developers increasingly encounter projects requiring Natural Language Processing (NLP) techniques – such as text summarization, sentiment analysis, and text similarity identification – a readily accessible knowledge base of common problems and effective solutions becomes crucial. Manually searching through extensive online resources like Stack Overflow for specific NLP challenges can be highly inefficient, especially under project deadlines.

The main objective of this project was to satisfy this need by designing and creating an automated pipeline to build a domain-specific NLP Knowledge Base. This system is designed to maintain an exhaustive record of problems presented on Stack Overflow by developers and individual contributions to their solutions (the accepted answers).

The main idea behind this knowledge base is to organize these posts in a way that will allow developers especially the beginners of the NLP domain, to:

- Understand the basic and advanced NLP concepts with real-word examples
- Appreciate well-known implementation challenges and pitfalls
- Rapidly discover potential solutions and code snippets posted by more experienced peers
- Find out the trends inside the NLP Development Community & its popular tools.

The Web App is the main component of this project, a functional web application written on top of the Flask framework, providing an interactive front-end to the knowledge base generated, allowing you to navigate it, explore queries and visualize the gathered and processed information.

This is a purpose-built resource that, in conjunction with the web interface, should save developers a lot of time and effort compared to browsing through the raw data in existing forums.

2.2 Data Source and Collection Strategy

The main source of information for this knowledge base comes from Stack Overflow (SO), which is a popular Q&A platform for programmers and developers. To gather relevant posts in an organized way, we opted to use the Stack Exchange API.

Target Tag: The key filter for our data collection was the [nlp] tag on Stack Overflow. Since NLP issues often intersect with other technologies, we also included all related tags linked to each [nlp] post (like python, nltk, spacy, tensorflow, pytorch, bert).

Data Fields Collected: The script named data_collector.py was crafted to pull essential metadata and content for each relevant question:

- **question_id:** A unique identifier for the post
- **title:** The title of the question
- **description (body):** The complete HTML body of the question
- **tags:** A list of all related tags
- **creation_date:** The timestamp indicating when the question was posted
- **view_count:** How many times the question has been viewed
- **score:** The overall upvote/downvote score of the question
- **answer_count:** The total number of answers submitted
- **is_answered**: A boolean that shows if the question has at least one answer
- **accepted_answer:** The content of the answer marked as accepted by the asker (if available)
- **other_answers:** The content of other non-accepted answers (limited to the top 5 by votes)

Data Volume: For this assignment, we needed a dataset with at least 20,000 posts. The data collection script was designed to manage API pagination (pagesize, page) and to handle rate limit backoff, allowing us to gather data on this scale.

Output Format: The raw data we collected, which includes questions and their corresponding answers, was systematically saved into a CSV file located at ../data/nlp_stackoverflow_dataset.csv This format makes it easy to load and manipulate the data using analysis libraries like Pandas.

3. Automated Text Classification

3.1. Cleaning Techniques:

The preprocessing pipeline utilized a series of standard NLP cleaning techniques in a step-by-step manner:

1. HTML Tag Removal: We used the BeautifulSoup library to sift through the HTML content found in the description, accepted_answer, and other_answers fields. All HTML tags were stripped

away to reveal the plain text. We had the option to keep or discard code blocks (<code <prep>), but for this project, we decided to remove them to concentrate on the natural language discussions. HTML entities (like & and <) were also decoded into their respective characters.

2. URL Removal: We employed regular expressions (from the re module) to spot and remove hyperlinks (both http/https and www. formats) from the text, as they usually don't add value for content analysis.

3. Lowercasing: We converted all text to lowercase. This helps maintain consistency and ensures that variations of the same word with different capitalizations (like "NLTK" and "nltk") are treated as the same during analysis and categorization.

4. Punctuation Removal: We eliminated standard punctuation marks (such as commas, periods, question marks, and brackets) using Python's string.punctuation and str.translate method. This makes the text cleaner and allows us to focus on the essential words.

5. Tokenization: The cleaned text was broken down into individual words or tokens using the word_tokenize function from the Natural Language Toolkit (nltk) library. This step is crucial for many NLP tasks, including stop word removal and frequency analysis.

6 Stop Word Removal: We took out common English words that don't really add much meaning—like "a," "the," "is," "in," and "on"—from the token lists for sections such as descriptions and answers. We did this using the standard English stop word list from nltk.corpus.stopwords. However, we kept stop words in the title column during the first round of processing. This was to help with keyword-based categorization rules that might depend on phrases like "how to" or "what is.".

3.2. Processed Columns:

The preprocessing steps were applied to the primary text fields of the dataset loaded from ../data/nlp_stackoverflow_dataset.csv New columns were added to the DataFrame to store the cleaned and processed versions:

- processed_title: Preprocessed version of the question title
- processed_description: Preprocessed version of the question body
- processed_accepted_answer: Preprocessed version of the accepted answer body
- processed_other_answers: List containing preprocessed versions of other answer bodies
- processed_tags: List of tags, converted to lowercase

The resulting DataFrame, containing both original and processed data, was saved to a new CSV file:

../data/nlp_stackoverflow_dataset_preprocessed.csv. As indicated in the execution output log, this preprocessing stage took approximately 31.00 seconds to complete for the dataset used.

4. Graphical Representation of the Data (Data Visualization)

Visualizing the characteristics and trends within the Stack Overflow NLP dataset provides valuable insights into the community's focus areas, popular tools, and activity patterns. The data_visualizer.py script was used to generate several informative plots based on the preprocessed data. All generated visualization images were saved within the ../data/visualizations/ directory and are subsequently used by the Flask web application.

4.1. Word Clouds

Word clouds provide a quick and easy way to visualize the most common terms found in a piece of text. We created two word clouds:

- **Title Word Cloud:** This visualization showcases the key terms that stand out in the titles of the NLP-related questions after some basic cleaning up, like making everything lowercase and removing punctuation. It might still keep some stop words that are important for understanding the structure of the questions. This makes it easier to spot recurring themes or types of questions at a glance.

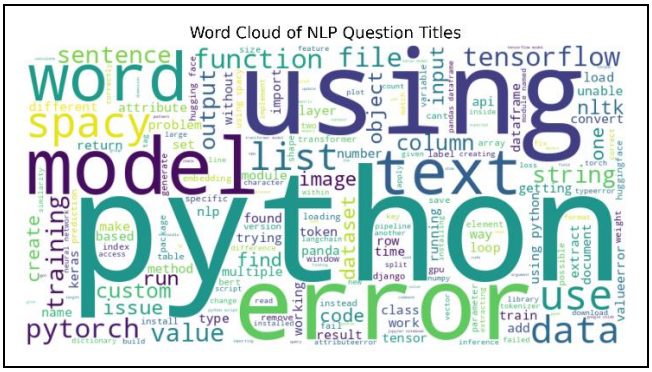

Figure 1: Word Cloud representing frequent terms in Stack Overflow post titles related to NLP.

- **Description Word Cloud:** This word cloud zooms in on the main body or description of the questions. Here, we applied more thorough cleaning, including removing stop words, to really highlight the essential technical terms, libraries, and concepts that are discussed in detail throughout the posts.

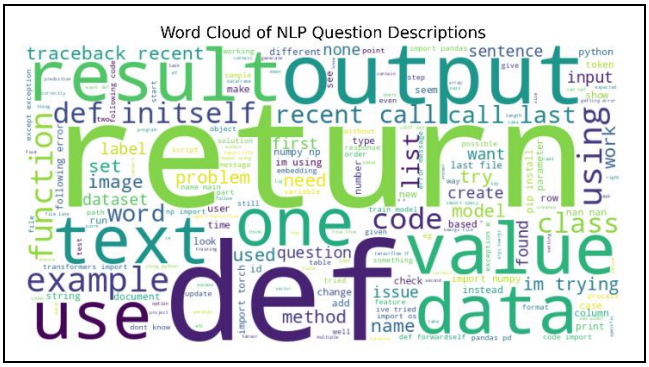

Figure 2: Word Cloud representing frequent terms in Stack Overflow post descriptions related to NLP.

4.2. Tag Frequency

Getting a grip on the technologies and concepts that often come up alongside NLP is really important. We created a horizontal bar chart to showcase how often the top 20 tags appear together with the [nlp] tag. This visual really highlights the ecosystem that supports NLP development on Stack Overflow.

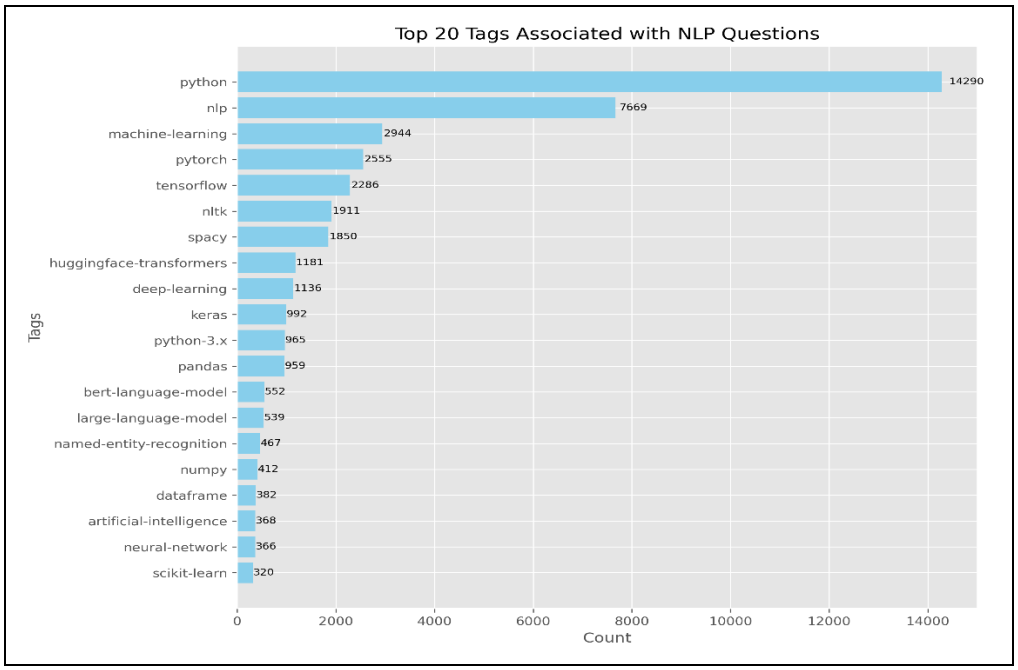

Figure 3: Top 20 most frequent tags co-occurring with the [nlp] tag on Stack Overflow.

4.3. Temporal Trends

To get a clearer picture of how interest and activity in NLP have changed on Stack Overflow, we created a line plot. This plot illustrates the number of questions tagged with [nlp] that were posted over time, usually grouped by month and year. It's a great way to see when the community was really engaged, when there might have been some dips, or when things stayed pretty steady in terms of discussions around NLP topics.

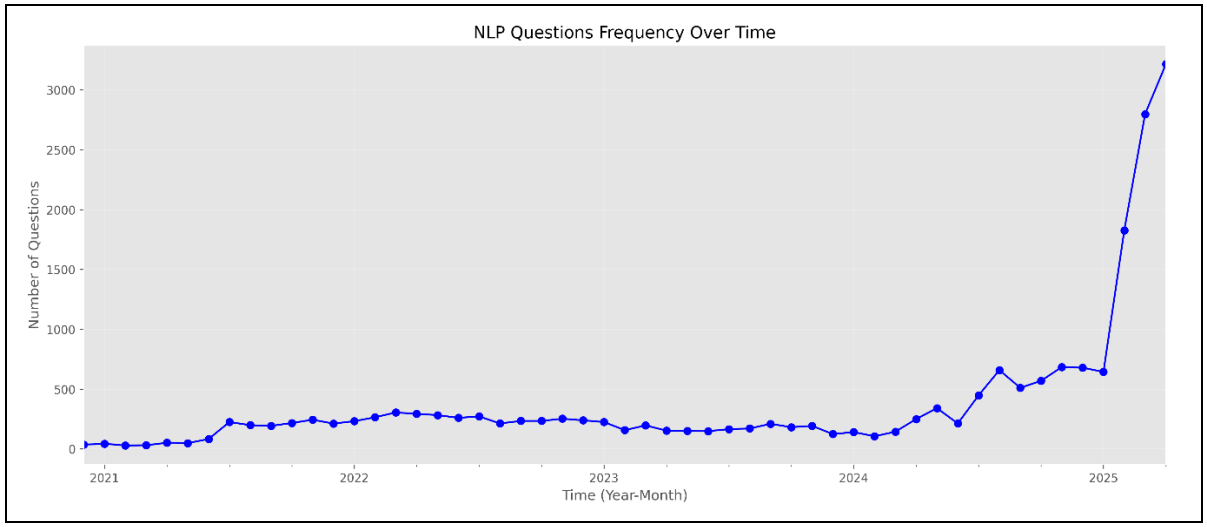

Figure 4: Frequency of NLP-tagged questions posted on Stack Overflow over time.

4.4. Engagement Metrics

The relationship between question visibility (views) and community response (answers) can be insightful. A scatter plot (img) was generated to visualize this relationship. Due tothe potentially wide range and skewed distribution of views and answers, a logarithmic scale was applied to both axes Log(Views + 1) vs. Log(Answers + 1) for better visual clarity and pattern identification. A regression line was also included to indicate the general trend.

The entire visualization generation process, as reported in the execution log, was completed efficiently in approximately 15.51 seconds. These visualizations collectively provide a multi-faceted overview of the dataset's characteristics.

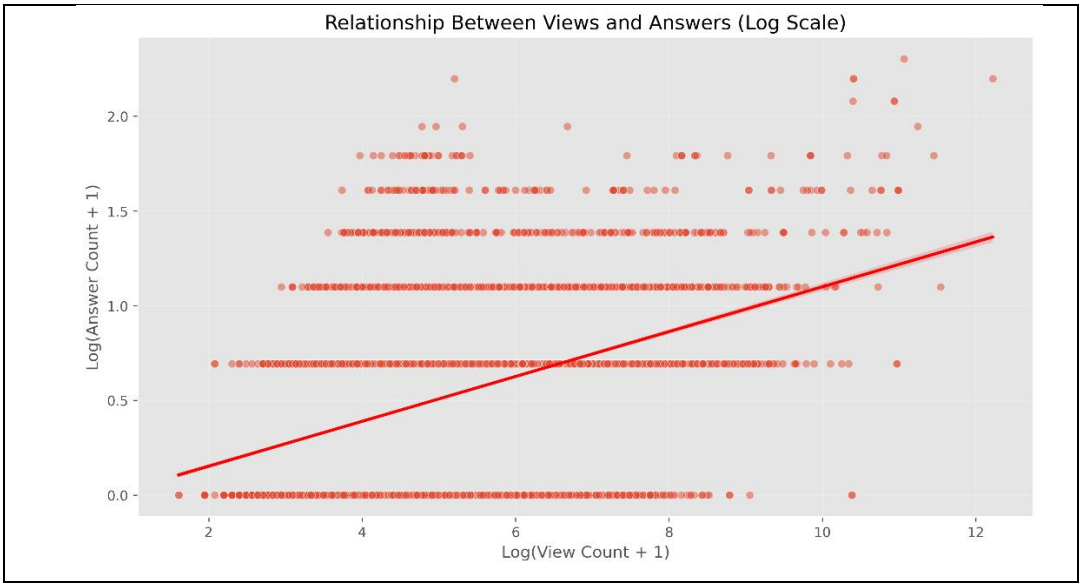

Figure 5: Scatter plot showing the relationship between view count and answer count for NLP posts (Log Scale).

5. Data Categorization

While the raw dataset and visualizations provide valuable insights, it's crucial to sort through the multitude of posts and organize them into meaningful categories. This step is key to building a truly functional knowledge base. The aim of categorization is to cluster similar questions, making it easier for developers to find topics that matter to them—like all the posts related to "Named Entity Recognition" or "spaCy implementation issues." The categorizer.py script played a vital role in automating this categorization process. The categorized data files that emerged from this effort are what the web application relies on for its core content.

5.1. Categorization Strategy

Given the vastness of the dataset, it just wasn't practical to categorize everything by hand. So, the project turned to an automated, rule-based categorization method that mainly focused on analyzing the textual content in the processed_title column, while also pulling in details from the tags

column to help identify libraries. Here's how it works:

- **Keyword Matching:** We set up groups of keywords and phrases tied to specific categories (like using ['classification', 'classifier'] for "Text Classification").
- **Pattern Recognition:** We applied some straightforward regular expressions to spot common question formats (for instance, using r'\bhow\b' for "How" type questions).
- **Library Name Identification:** We searched for mentions of well-known NLP libraries and frameworks (think "nltk", "spacy", "huggingface", "bert").

It's worth noting that a single post could fit into multiple categories across different schemes. For example, a post titled "How to implement NER using spaCy?" might be categorized under "Implementation Issues", "Named Entity Recognition", and "spaCy".

5.2. Category Definitions Examples

We implemented four unique categorization schemes to offer various perspectives on the data:

1. **Keyword-based Categorization:** This approach grouped posts according to general keywords that reflect the type of problem or discussion.
    - **Implementation Issues:** Posts that ask "how-to" or discuss code examples (Keywords: "how to", "implement", "code").
    - **Understanding Concepts:** Posts looking for explanations or definitions (Keywords: "what is", "explain", "concept", "why").
    - **Error Troubleshooting:** Posts that describe errors, bugs, or seek solutions (Keywords: "error", "bug", "fix", "issue", "failed").
    - **Library Usage:** Posts that specifically mention interactions with a library (Keywords: "spacy", "nltk", etc. - this overlaps with the Library-based scheme but also captures general usage questions).
    - **Performance Issues:** Posts that deal with speed, memory, or efficiency (Keywords: "slow", "performance", "memory", "efficient").
2. **Task-based Categorization:** This scheme focused on grouping posts related to specific, well-defined NLP tasks.
    - **Text Classification:** (Keywords: "classification", "categorization").
    - **Named Entity Recognition**: (Keywords: "ner", "named entity").
    - **Tokenization:** (Keywords: "tokenization", "tokenize").
    - **Word Embedding:** (Keywords: "word2vec", "glove", "embedding").
    - **Sentiment Analysis:** (Keywords: "sentiment", "emotion", "polarity").

- **Part-of-Speech Tagging:** (Keywords: "pos", "part of speech", "tagging").

3. **Question Type-based Categorization:** This straightforward scheme categorized posts based on the main interrogative word found in the title, giving insight into the intent behind the question. Examples: How, What, Why, When, Where.
4. **Library-based Categorization:** This scheme specifically grouped posts that mentioned certain NLP libraries, frameworks, or models, using both the title text and associated tags. Examples: NLTK, spaCy, Hugging Face (including "transformer"), TensorFlow, PyTorch, BERT, Word2Vec, Gensim.

5.3. Categorization Results

../data/nlp_stackoverflow_dataset_preprocessed.csv file and assigned posts to categories within each of the four schemes. The minimum requirement of 10 posts per category was easily met for all generated categories.

Summary of Categorization Outputs:

Categorization Scheme	Number of Categories Generated	Example Categories & Post Counts
Keyword-based	18	Library Usage (4700), Implementation Issues (4492), Error Troubleshooting (3751), Understanding Concepts (1163), Named Entity Recognition (1096), Word Embeddings (699), Data Collection (666)
Task-based	16	Named Entity Recognition (1096), Word Embeddings (699), Part-of-Speech Tagging (600), Tokenization (513), Text Classification (480), Stemming (197), Sentiment Analysis (119)
Question Type based	5	How (4542), When (1002), Why (844), What (573), Where (53)
Library-based	16	TensorFlow (2858), PyTorch (2649), NLTK (1916), spaCy (1910), Hugging Face (1553), BERT (709), Word2Vec (257), Gensim (233), GPT (139),

		WordNet (120)

We ended up categorizing a whopping 15,482 unique posts across all schemes, which is way more than the minimum requirement of just 100 posts per project. We made sure to save the categorized data in an organized way. For each scheme, like task_based, we created a subdirectory in ../data/categories/. Within each of these subdirectories, we generated a separate CSV file for every category—like ../data/categories/task_based/Text_Classification.csv—containing only the posts that fit into that specific category. This neat structure makes it super easy to access posts related to a certain topic or library through the web application. Plus, the whole categorization process was pretty quick, wrapping up in about 7.21 seconds.

6. Implementation Details

The project was developed as a cohesive system comprising a data processing pipeline and a web application interface, both implemented using Python 3. The code was structured into distinct modules and directories to promote organization, reusability, and maintainability.

6.1 System Architecture

The project consists of two main parts:

1. **Data Processing Pipeline (src/ directory):** Orchestrated by main.py, this pipeline handles the backend data tasks.
   - **data_collector.py:** Interacts with the Stack Exchange API for data fetching
   - **preprocessor.py:** Cleans and preprocesses text data
   - **data_visualizer.py:** Generates plots and visualizations
   - **categorizer.py:** Categorizes posts based on defined schemes
   - **main.py:** The main script to run the pipeline stages via command-line arguments
2. **Web Application (web-app/ directory):** A Flask application providing the user interface.
   - **app.py:** The main Flask application file containing route definitions, logic to load data/categories/visualizations, and search functionality
   - **templates/:** Contains HTML templates (using Jinja2 syntax) for rendering web pages (e.g., index.html, categories.html, category.html, visualizations.html, search.html, about.html, base.html)
   - **static/:** Stores static assets
     - **css/style.css:** Custom CSS for styling the web interface
     - **js/script.js:** JavaScript for client-side enhancements (e.g., dark mode toggle, dynamic content loading/sorting)
     - **img/:** Directory where visualization images (.png) are copied to be served by the web

app

- **copy_visualizations.py:** A utility script to copy generated plots into the static/img directory
- **requirements.txt:** Lists Python dependencies for the web application (Flask, pandas, gunicorn)

This modular design separates data processing logic from the presentation layer, making the system easier to manage and extend.

6.2 Key Libraries

The project utilized several standard Python libraries:

1. **Data Pipeline (src/):**
   - requests: For API calls
   - pandas: For data manipulation and CSV handling
   - nltk: For NLP preprocessing (tokenization, stop words)
   - beautifulsoup4: For HTML parsing
   - re: For regular expressions
   - matplotlib & seaborn: For generating visualizations
   - wordcloud: For creating word clouds
   - argparse: For command-line argument handling
   - os, time: For system interactions
2. **Web Application (web-app/):**
   - Flask: Micro web framework for building the web interface
   - pandas: Used within Flask routes to load and process CSV data for display
   - gunicorn: WSGI server, suitable for production deployment
   - os, json, datetime, logging, urllib.parse: Standard libraries used within app.py
3. **Frontend (served by Flask):**
   - HTML5, CSS3, JavaScript
   - Bootstrap 5: CSS framework for layout and components
   - Font Awesome: For icons
   - Google Fonts (Poppins, Roboto Mono): For typography
   - highlight.js: For code syntax highlighting in posts
   - jQuery: JavaScript library for enhanced interactivity

6.3 Execution

The system has two primary execution modes:

1. **Data Pipeline:** Executed via the command line from the src/ directory.

bash python main.py [--api-key YOUR_API_KEY] [--skip-collection] [...]

Arguments control which stages run (data collection, preprocessing, visualization, categorization).

2. **Web Application:** Executed from the web-app/ directory.

- Development:

  bash:

  ```
  # First, copy visualizations if needed
  python copy_visualizations.py
  # Then, run the Flask development server
  python app.py
  ```

- Production (Using Gunicorn):

  bash:

  ```
  gunicorn -w 4 -b 0.0.0.0:5000 app:app
  ```

The Python code across all modules is well-commented. The web application uses Flask routes to serve HTML pages, load data dynamically from the generated CSV files, and handle search queries.

GitHub Repository: The complete source code for both the pipeline and the web application, dataset files (or instructions to generate them), and generated outputs are available at:

https://github.com/shubharthaksangharsha/nlp-assingment-2

7. Conclusion

This project has successfully created an automated pipeline along with a web application designed to build an NLP Knowledge Base using data from Stack Overflow. The system efficiently gathers, preprocesses, visualizes, and organizes posts tagged with [nlp], filling a crucial gap for developers who are facing NLP challenges.

The pipeline showcased its effectiveness by producing a preprocessed dataset, insightful visualizations, and various sets of categorized posts based on keywords, tasks, question types, and libraries. The Flask web application serves as an essential user interface, allowing users to interactively explore these categories and visualizations, as well as perform full-text searches across the collected posts. This integrated system makes it much easier to find relevant discussions, implementation examples, and solutions within the extensive Stack Overflow database compared to searching manually.

Looking ahead, there are several potential enhancements to consider:

- Fine-tuning categorization rules, possibly by incorporating machine learning models
- Enhancing the web application's UI/UX, such as adding pagination and more interactive filtering options

- Setting up a system for regular data updates
- Possibly deploying the web application on a robust server setup like Gunicorn

Overall, the project meets the assignment requirements by providing both a well-structured knowledge base and an easy-to-use web interface.

1. References

[1] Bird S, Klein E, Loper E. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc."; 2009 Jun 12.

[2] Hunter JD. Matplotlib: A 2D graphics environment. Computing in science & engineering. 2007 May 1;9(03):90-5.

[3] McKinney W. Data structures for statistical computing in Python. SciPy. 2010 Jun 28;445(1):51-6.

[4] Waskom ML. Seaborn: statistical data visualization. Journal of Open Source Software. 2021 Apr 6;6(60):3021.

[5] Abodayeh A, Hejazi R, Najjar W, Shihadeh L, Latif R. Web scraping for data analytics: A BeautifulSoup implementation. In2023 Sixth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU) 2023 Mar 14 (pp. 65-69). IEEE.

[6] Fu X, Yu S, Benson AR. Modelling and analysis of tagging networks in Stack Exchange communities. Journal of Complex Networks. 2020 Oct 1;8(5):cnz045.

[7] Bhatia A, Wang S, Asaduzzaman M, Hassan AE. A study of bug management using the Stack Exchange question and answering platform. IEEE Transactions on Software Engineering. 2020 May 11;48(2):502-18

[8] Swillus M, Zaidman A. Sentiment overflow in the testing stack: Analyzing software testing posts on Stack Overflow. Journal of Systems and Software. 2023 Nov 1;205:111804.