Graph Theory

(Project)

Classification of Web Documents using KNN

Session 2021-2025

Submitted By

Syed Hashir Husnain 2021-CS-1
Muhammad Kabir Ahmad 2021-CS-4

Submitted To

Mr. Waqas Ali

Department of Computer Science

University of Engineering and Technology
Lahore, Pakistan

Contents

List of Figures

List of Tables

1 Abstract

This project aims to classify web documents using the K-Nearest Neighbors (KNN) method with a graph-based approach, leveraging a comprehensive machine-learning pipeline. The pipeline encompasses key stages including data collection, processing, and feature extraction. Initially, raw data is gathered from web sources. Subsequently, the data undergoes preprocessing and feature extraction to transform it into a graph representation. The graph structure captures the relational information between various elements within the documents. The dataset is then divided into training and testing subsets for model development and evaluation. Through the KNN algorithm, documents are classified based on their similarity to neighboring instances in the feature space. Performance metrics are employed to assess the efficacy of the trained model in accurately categorizing web documents. This holistic approach offers a systematic framework for efficient and effective classification of web content, with potential applications in information retrieval and content organization.

2 Introduction

In the vast digital landscape of the internet, the ability to effectively organize and categorize web content is paramount for efficient information retrieval and content organization. This challenge forms the basis of our project, which aims to classify web documents utilizing the K-Nearest Neighbors (KNN) method coupled with a graph-based approach. Through the utilization of a comprehensive machine learning pipeline, we endeavor to construct a robust framework that navigates through the complexities of web data and facilitates accurate classification.

At the heart of our methodology lies a meticulous pipeline comprising crucial stages essential for the success of our endeavor. The journey commences with the collection of raw data sourced from the expansive realm of the internet. Leveraging cutting-edge technologies such as Node.js, Cheerio, and Axios, we harness the power of web scraping to aggregate diverse and relevant datasets from various online sources. This initial step lays the foundation for subsequent processing and analysis.

With a trove of raw data at our disposal, the next imperative phase revolves around preprocessing and feature extraction. This pivotal stage involves a series of transformative steps aimed at refining and structuring the data to extract meaningful insights. To enhance the quality and relevance of the dataset, we employ a suite of preprocessing techniques. These include the removal of stopwords, numbers, and punctuations, along with the elimination of duplicate entries. Furthermore, we employ lemmatization to standardize words to their base form, thereby ensuring consistency and coherence within the dataset.

Having distilled the data into a refined form, we proceed to feature extraction, a pivotal step in our pipeline. Here, we employ tokenization techniques to break down the text into individual tokens, each representing a distinct unit of meaning. These tokens serve as the building blocks for constructing a graph representation of the dataset. By mapping the relationships between tokens in the form of a graph structure, we capture the intricate interplay and semantic connections inherent within the web documents.

The transformation of data into a graph-based representation lays the groundwork for our classification framework. Dividing the dataset into training and testing subsets, we embark on the journey of model development and evaluation. Leveraging the KNN algorithm, documents are classified based on their proximity to neighboring instances in the feature space. This proximity-based approach enables us to leverage the inherent similarities and relationships encoded within the graph structure for accurate classification.

To gauge the efficacy and performance of our classification model, we employ a battery of performance metrics. Accuracy, precision, recall, and the F1 score serve as the litmus test for evaluating the model's ability to accurately categorize web documents. Through meticulous evaluation and analysis, we gain valuable insights into the strengths and limitations of our approach, paving the way for iterative refinement and optimization.

In summary, our project endeavors to provide a holistic and systematic framework for the efficient classification of web content. By amalgamating cutting-edge technologies with sophisticated machine-learning techniques, we aim to unlock new frontiers in information retrieval and content organization. With the potential for diverse applications across various domains, our approach holds promise in addressing the ever-evolving challenges posed by the dynamic landscape of the internet.

3 Graph Based Approach for K-Nearest Neighbour

A graph-based approach utilizing K-Nearest Neighbors involves the traditional machine learning pipeline.

- Data Collection
- Pre-processing (Feature Engineering)
- Feature Selection

- Model Training
- Model Evaluation

3.1 Data Collection

3.1.1 Web scrapping

The input data is of document type which contains text. We scrapped different websites which contains articles related to different topics. We scrapped these websites and extracted our desired data. For scrapping the web, we utilize the tool i.e., Cheerio.

What is Cheerio? Cheerio is a lightweight and fast library for parsing and manipulating HTML documents in Node.js. It provides a simple and efficient interface for traversing the DOM (Document Object Model) structure of HTML pages, similar to jQuery in web browsers. With Cheerio, you can easily select and manipulate elements, extract data, and perform various operations on HTML content using familiar jQuery-like syntax within Node.js environment. It is commonly used in web scraping and data extraction tasks where HTML parsing is required.

Table 1: Details of technology used in scrapping of web

Language	**Node JS**
Request Handling	Axios
Document Handling	Cheerio

The scrapped documents are related to following topics:

- Health and Fitness
- Food
- Business and Finance

3.1.2 Challenges

- **BrightData**
  First we thought of using BrightData for web scrapping. But BrightData is not providing services in Pakistan.
- **Scrapper API**
  Secondly, we tried Scrapper API. Scrapper API is also not providing its services, like BrightData, in Pakistan.
- **Puppeteer**
  There was some error in installing Puppeteer package.

3.1.3 Input Data

We scrapped total 45 documents for the above mentioned topics. This means we have multi-class annotated data.

Table 2: Details of Scrapped Data

Data type	Text
Data Structure	Document
Total Documents	45
Dataset	Multi-Class Annotated Data

Table 3: Categorization of Scrapped Documents

Health and Fitness	15
Food	15
Business and Finance	15

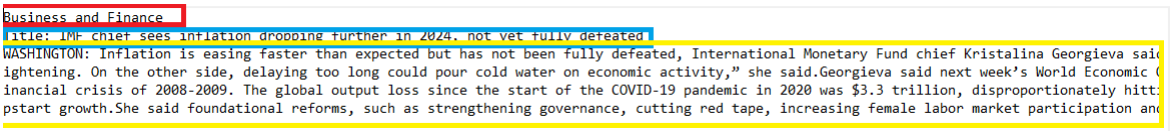

Figure 1: Structure of each Document, Red: Topic, Blue: Title, Yellow: Article

3.2 Pre-processing

Preprocessing is crucial in natural language processing (NLP) tasks to clean and prepare text data for further analysis. In the provided code, the following preprocessing steps are performed:

a. **Removing Stopwords**: Stopwords are common words like "and", "the", "is", etc., which do not carry significant meaning. Removing them helps reduce noise in the data.

b. **Removing Numbers**: Numerical digits are removed as they typically don't contribute much to the semantics of the text.

c. **Removing Punctuation**: Punctuation marks such as commas, periods, and exclamation marks are removed to simplify the text.

d. **Removing Duplicates**: Duplicate words within the text are removed to avoid redundancy.

e. **Correcting Misspelled Words**: Misspelled words are corrected using a spell checker to improve the accuracy of subsequent analysis.

f. **Lemmatization**: Lemmatization reduces words to their base or dictionary form (lemmas). For example, "running" becomes "run".

g. **Tokenization**: Tokenization divides the text into smaller units called tokens, typically words or punctuation marks. It's a crucial step before further processing.

h. **Adding Synonyms and Antonyms**: Synonyms and antonyms of each word are added to enrich the vocabulary and capture more semantic information.

i. **Stemming**: Stemming reduces words to their root form (stem). For example, "running" and "ran" both stem to "run". This step helps in normalizing variations of words.

3.2.1 Feature Extraction

Feature extraction involves converting raw text data into a numerical representation that can be used for machine learning algorithms. In the provided code, the main feature extraction technique used is:

a. **Graph Representation**: The text data is transformed into directed graphs, where nodes represent words and edges represent relationships between words. This graph-based representation captures semantic relationships between words and allows for efficient comparison between documents as shown in figure:

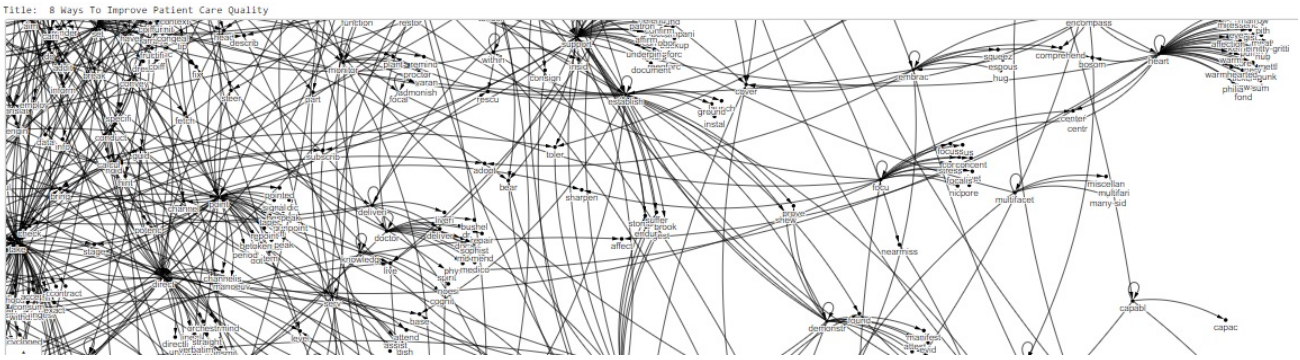

Figure 2: Graphical Representation of Feature Extraction using mentioned Techniques

By doing Techniques like:

. Removing Stopwords

. Removing Numbers

. Removing Punctuation

. Removing Duplicates

. Correcting Misspelled Words

. Lemmatization

3.2.2 Feature Selection

Feature selection involves choosing a subset of relevant features to improve the performance of machine learning models and reduce computational complexity. In the provided code, explicit feature selection techniques select relevant features based on the semantic relationships between words. By doing Techniques like:

. Tokenization

. Adding Synonyms and Antonyms

. Stemming

4 Splitting Data

After preprocessing and feature extraction, the dataset containing a total of 61 data graphs is split into two subsets: training data and test data.

1. **Total Data Graphs**: Initially, the dataset consists of 61 data graphs, each representing a document or webpage.
2. **Training Data**: From the total dataset, 44 data graphs are selected to form the training set. These data graphs will be used to train the classification model.
3. **Test Data**: The remaining 17 data graphs are separated to form the test set. These data graphs are not used during training but are reserved for evaluating the performance of the trained model.
4. **Purpose of Splitting**: The purpose of splitting the dataset into training and test sets is to assess how well the classification model generalizes to unseen data. By training the model on one subset and evaluating it on another, we can estimate its performance on new, unseen documents.
5. **Randomness or Stratification**: Random splitting ensures that the data in both subsets are representative of the overall dataset, while stratified splitting may ensure that class distribution is maintained in both subsets.

Overall, the splitting of data into training and test sets facilitates the evaluation of the classification model's performance and helps identify potential issues such as overfitting or underfitting.

5 Training Data Using KNN Algorithm

The training of the data involves using the K-Nearest Neighbors (KNN) algorithm, which is implemented as follows:

1. **KNN Function Definition**: A function named `KNN` is defined, which takes the training graphs, test graphs, and the value of 'k' (number of neighbors) as input parameters. This function is responsible for performing the classification using the KNN algorithm.
2. **Similarity Calculation**: Within the `KNN` function, the similarity between each test graph and all training graphs is calculated using a similarity measure. The Jaccard similarity measure is utilized, considering both node and edge similarity between graphs. Jaccard similarity is calculated based on the intersection and union of nodes and edges between two graphs.

3. **Neighbor Selection**: After computing the similarity between each test graph and all training graphs, the KNN algorithm selects the 'k' nearest neighbors for each test graph based on their similarity scores. These neighbors are chosen based on their highest similarity scores with the test graph.

4. **Majority Voting**: Once the 'k' nearest neighbors are identified for each test graph, the majority class label among these neighbors is determined. The class label that occurs most frequently among the neighbors is assigned as the predicted label for the test graph.

5. **Prediction**: The predicted labels for the test graphs are then returned as the output of the `KNN` function. These predicted labels represent the classification results obtained by the KNN algorithm based on the training data.

Overall, the training of the data using the KNN algorithm involves calculating the similarity between graphs, selecting the nearest neighbors, and assigning class labels based on majority voting. This process enables the KNN algorithm to make predictions on unseen test data based on the patterns learned from the training data.

6 Model Evaluation

In the provided code, model evaluation is performed to assess the performance of the K-Nearest Neighbors (KNN) algorithm on the test data. Here's how the evaluation process, accuracy, recall, precision, and F1 score are implemented:

1. **Evaluation Process**: The code iterates over the predicted labels and true labels of the test data to compare them and evaluate the model's performance.

2. **Accuracy Calculation**: The accuracy of the model is calculated by counting the number of correctly classified instances and dividing it by the total number of predictions. The formula used for accuracy calculation is:
$$Accuracy = \frac{NumberofCorrectPredictions}{TotalNumberofPredictions} \times 100\%$$
From this We get the accuracy of 90% in our project.

3. **Confusion Matrix**: A confusion matrix is computed to visualize the performance of the model across different classes. The confusion matrix provides a breakdown of the number of true positives, true negatives, false positives, and false negatives.

4. **Recall Calculation**: Recall (also known as sensitivity) is calculated for each class using the formula:
$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$
The recall score indicates the model's ability to correctly identify positive instances for each class.

5. **Precision Calculation**: Precision is computed for each class using the formula:
$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$
Precision measures the model's ability to avoid false alarms for each class.

6. **F1 Score Calculation**: The F1 score, which is the harmonic mean of precision and recall, is calculated for each class using the formula:
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$
The F1 score balances precision and recall, providing a single metric to evaluate the model's performance for each class.

Overall, model evaluation in the provided code assesses the KNN algorithm's effectiveness in classifying the test data and provides metrics to measure its performance across different classes.

7 OverView Methodology Daigram

The whole project methodology that we describe above is visually represented in diagram as follow:

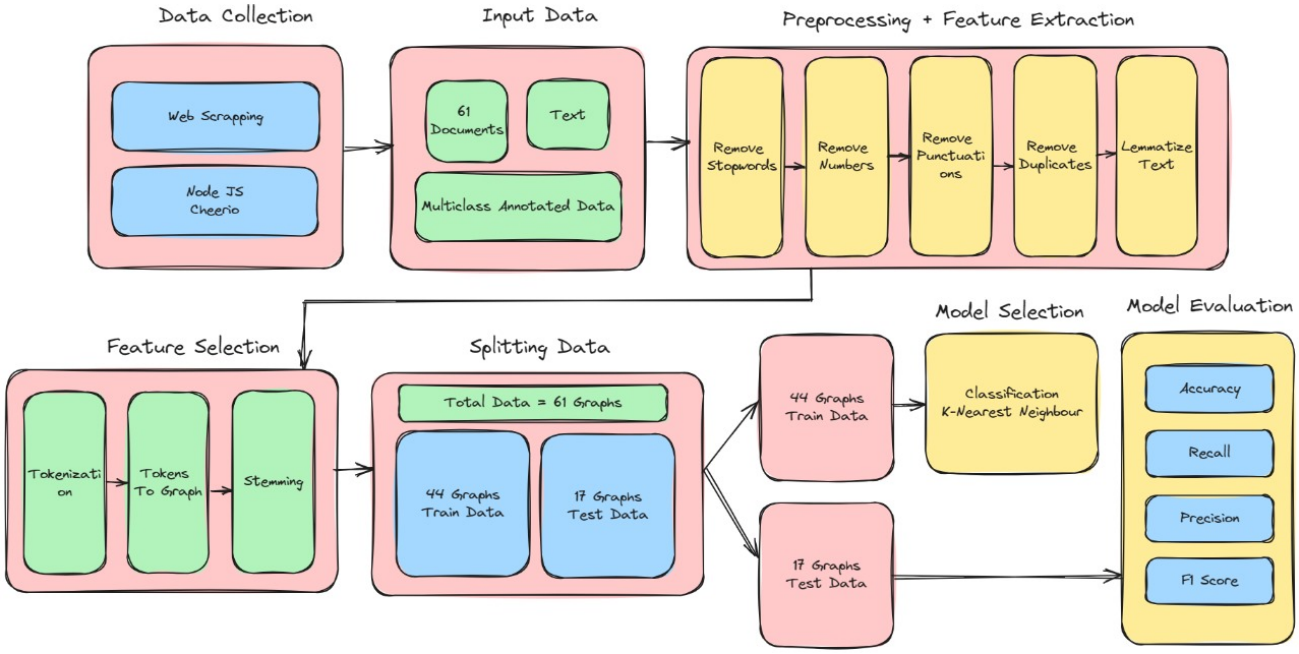

Figure 3: Project Overview Daigram

8 Project Code

To visit Document pdf, click here.
To excess collab File click here.
To excess collab File click here.