Classification of Documents Using Graph-Features & KNN

CS-380 Graph Theory

Session: 2021-2025

Project Supervisor

Waqas Ali

Submitted by:

Muhammad Tahir	2021-CS-34
Abdur Rehman	2021-CS-44

Department of Computer Science

University of Engineering and Technology, Lahore

Pakistan

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

With a daily stream of papers into cyberspace, the exponential growth of digital content on the internet has reached astounding dimensions in the dynamic world of modern information technology. The need to effectively handle and organize this enormous volume of data has made the task of document classification crucial. Conventional classification techniques, which mostly depend on vector models, have faced considerable difficulties in handling the vast amount and intricacy of textual data. These methods frequently perform poorly and have limited accuracy, mostly because they are unable to catch the subtleties of text structure. The need to investigate alternate approaches that can better handle the varied and dynamic nature of digital content has therefore grown. To address these issues, the application of graph-based methods has become a viable way to improve the efficacy and efficiency of document classification.

1.2 Objectives

In light of this, the main goal of this project is to create a novel document categorization system that utilizes the KNN algorithm in combination with graph-based characteristics. The particular goals include:

- Assembling an extensive dataset with a variety of document kinds that fall into pre-established categories.
- Capturing the innate dependencies and relationships in every document by converting it into a structured graph representation.
- Finding recurrent patterns and features in a collection of training documents by identifying and extracting common subgraphs.
- Graph similarity measures are used to implement the KNN method, enabling precise and effective document classification.
- Using the trained model to categorize test texts according to how close they are to training examples in the feature space defined by shared subgraphs.

2 Literature Review

2.1 Classification of Web Documents Using a Graph Model

The k-Nearest Neighbor (k-NN) classification algorithm is extended in this study by the authors to handle data represented as graphs instead of typical numerical feature vectors. The suggested method is used to classify online texts, which are represented as graphs. It does this by using a graph-theoretical distance measure to compute distance within the k-NN algorithm. Three different datasets are used for experimental assessments using the leave-one-out method, which compares the performance of the suggested method with the conventional vector model approach that makes use of cosine similarity. Results show that the suggested method performs much better in terms of accuracy than the vector model and also shows considerable advantages in terms of execution time. The authors hope to expand on this research by applying other classification techniques, including the distance weighted k-Nearest Neighbors method, to the graph domain. Experiments will also be carried out to find the ideal number of nodes for every network, taking into account the trade-off between execution time and performance. To efficiently optimize both execution time and performance, the authors suggest that the graph size be varied according to the specific requirements of each document.

2.2 A graph distance metric based on the maximal common subgraph

This research paper introduces a new graph distance measure, providing a fresh perspective on comparing or matching graphs. While edit distance is a common method, it requires setting costs for edit operations, which can be complex. The proposed measure, based on the maximal common subgraph (MCS) of two graphs, offers a solution. MCS represents the largest subgraph present in both graphs. The paper demonstrates that this new measure fulfills the properties of a metric, ensuring its mathematical reliability for measuring distance. What sets this measure apart is its independence from predefined edit costs, potentially broadening its applicability across different scenarios.

Formula:

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{\max(|G_1|, |G_2|)}$$

where $mcs(G_1, G_2)$ represents the maximal common subgraph between G_1 and G_2.

3 References

- A. Schenker, M. Last, H. Bunke and A. Kandel, "Classification of Web documents using a graph model," Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings., Edinburgh, UK, 2003, pp. 240-244 vol.1, doi: 10.1109/IC - DAR.2003.1227666.
- Horst Bunke, Kim Shearer, A graph distance metric based on the maximal common subgraph, Pattern Recognition Letters, Volume 19, Issues 3–4, 1998, Pages 255-259, SSN 0167-8655, https://doi.org/10.1016/S0167-8655(97)00179-

4 Methodology

4.1 DataCollectionandPreparation

4.1.1 Scraping

We searched for separate blog websites for each assigned topic or category to find the most relevant ones. Since each website had its own HTML structure and implementation, we had to create a unique script for each. Our group was assigned specific topics, and here are the corresponding websites we used:

- Travel
- Health and Fitness
- Science and Education

After analyzing the layout of each website, we opted for selenium:

- **Selenium**: With the help of this powerful web automation library, web elements may be interacted with smoothly and with exact control over web browsers. Selenium greatly expedites the scraping process by automating operations like surfing online pages, interacting with forms, and extracting data.

These technologies were crucial in negotiating the form and structure of each website, guaranteeing effective data extraction while allowing for dynamic content and interactions.

Category	Number of Documents	Avg Words
Travel	24	1460
Health and Fitness	15	1230
Science and Education	15	960

Table 1: Scraped articles

4.1.2 Preprocessing

We're performing preprocessing on text data extracted from a CSV file. Let's break down the preprocessing steps:

- **Tokenization Function**: This function divides a piece of text into individual words or tokens. For example, given the sentence "The cat jumps over the fence," the tokenize function would produce the list of tokens ["the", "cat", "jumps", "over", "the", "fence"].
- **Stop-word Removal Function**: This function removes common words known as stop words from a list of tokens. Stop words are words like "the," "and," "is," etc., which often occur frequently but carry little semantic meaning. For instance, if we have the list of tokens ["the", "cat", "jumps", "over", "the", "fence"], after stop-word removal, it becomes ["cat", "jumps", "over", "fence"].
- **Stemming Function**: The stemming function reduces words to their base or root form. For example, it converts words like "running" and "ran" to the base form "run." So, if we have the list of tokens ["running", "cats", "and", "dogs"], after stemming, it becomes ["run", "cat", "and", "dog"].

Before Preprocessing
Our bodies are very smart when it comes to surviving under harsh conditions. A fantastic â€" and somewhat infamous example â€" is adaptive thermogenesis, more commonly known as â€œstarvation mode.â€ But what is â€œstarvation modeâ€ and what role does it play in a weight loss journey? Hereâ€™s what you need to know. Is â€ Starvation Modeâ€™ a Real Phenomenon or Just a Myth? The concept of starvation mode is simple: If you donâ€™t eat enough, your body will believe youâ€™re starving. In turn, your body may maintain some of your weight and fat as an attempt to preserve energy, making it harder for you to lose weight. â€œStarvation modeâ€ only happens when a substantial caloric deficit is maintained for an extended period. While the exact timeline will vary depending on several factors, one study shows that it can happen after about a week of an extreme daily caloric restriction. Symptoms of â€œstarvation modeâ€ Thereâ€™s just no getting around it. Your body needs calories to work correctly. If you donâ€™t eat enough, your body will slip into â€œadaptive thermogenesis,â€ the scientific term for â€œstarvation mode.â€ This will slow your whole metabolism down, causing several side effects that will become more and more noticeable the longer you maintain the calorie deficit. Hereâ€™s a quick rundown of the most common â€œstarvation modeâ€ symptoms: Lethargy Constipation Depression Inability to concentrate
After Preprocessing
bodi smart come surviv harsh condit fantast – somewhat infam exampl – adapt thermogenesi commonli known “ starvat mode ” “ starvat mode ” role play weight loss journey ? need know ? starvat mode ? real phenomenon myth concept starvat mode simpl ? eat enough bodi believ ? starv turn bodi may maintain weight fat attempt preserv energi make harder lose weight “ starvat mode ” happen substanti calor deficit maintain extend period exact timelin vari depend sever factor one studi show happen week extrem daili calor restrict symptom “ starvat mode ” ? get around bodi need calori work correctli ? eat enough bodi slip “ adapt thermogenesi ” scientif term “ starvat mode ” slow whole metabol cau sever side effect becom notic longer maintain calori deficit ? quick rundown common “ starvat mode ” symptom lethargi constip depress inabl concentr feel unusu cold reduc appetit exact symptom may vari person person usual accompani gener decrea weight loss rate start experienc side effect contact doctor nutritionist soon possibl “ starvat mode ” impact weight loss “ starvat mode ” metabol rate may decrea significantli especi lose much weight quickli metabol rate refer number calori burn daili make crucial calor intak outtak formula exact number vari depend fast lose bodi weight good rule thumb adapt thermogenesi kick one week sever calor restrict restrict total daili energi intak basal metabol rate bmr bmr amount energi calori bodi need basics—breath circul blood regul bodi temperatur repair cells—when rest think calori ? burn stay bed day ? affect thing like age sex weight height bodi composit

Table 2: Before and After Preprocessing

4.2 Graph Construction

After preprocessing, the subsequent step involved constructing a directed graph for each document. To accomplish this task, we employed the networkx library due to its user-friendly interface and compatibility with other libraries for subsequent processing and visualization tasks. In the graph creation process, we preserved the order of words from the preprocessed document. Each unique word served as a node in the graph. For every two consecutive words in the document, an edge was established, with the tail pointing to the preceding word and the head pointing to the succeeding word. Initially, each edge was assigned a weight of 1, which incremented by 1 for every occurrence of such relationships. Below is the code snippet illustrating this process:

```
def create_graph(text):
    G = nx.DiGraph()
    previous_word = None
    for word in list(text):
        if word not in G:
            G.add_node(word)
        if previous_word:
            if G.has_edge(previous_word, word):
                G[previous_word][word]['weight'] += 1
            else:
                G.add_edge(previous_word, word, weight=1)
        previous_word = word
    return G
```

Figure 1: Graph Construction

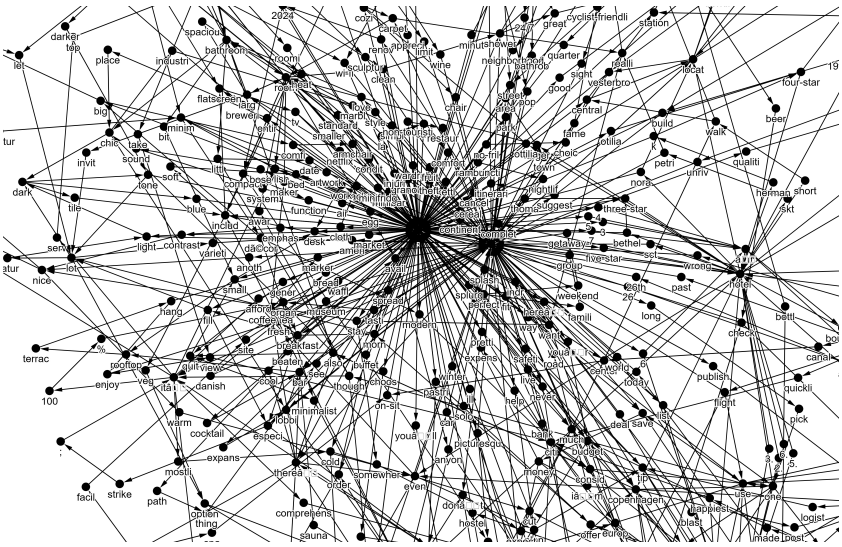

Figure 2: Graph Representation

4.3 Feature Extraction via Common Subgraphs

The next step is to identify the feature(s) that are necessary for correctly categorizing documents. This feature is known as the maximal common subgraph (MCS), which is the greatest common subgraph that is shared by two graphs. First, we identify all common nodes between two graphs in order to find an MCS between them. Next, for every node in the shared node set, we check its connectivity with all other nodes to determine whether an edge is present in both graphs. We add such shared edges to the MCS only once they are verified. The code sample that demonstrates this procedure is shown below:

```python
def find_mcs(graph_list):

    mcs_graph = nx.Graph()
    common_nodes = set.intersection(*[set(g.nodes) for g in graph_list])
    mcs_graph.add_nodes_from(common_nodes)
    for node1 in common_nodes:
        for node2 in common_nodes:
            if all(g.has_edge(node1, node2) for g in graph_list):
                mcs_graph.add_edge(node1, node2)
    return mcs_graph


def mcs_distance(graph1, graph2):

    mcs = find_mcs([graph1, graph2])
    return 1 - len(mcs.edges) / max(len(graph1.edges), len(graph2.edges))
```

Figure 3: Maximal Common Subgraph

4.4 Classification with KNN

In essence, K-nearest neighbors (KNN) classification operates by determining which graph is most similar to the input graph. This involves assessing the similarity between graphs by examining their shared structure. In cases of ambiguity, where multiple graphs exhibit identical similarity, the class label with the highest occurrence frequency is selected as the predicted class. The distance measure utilized to quantify the dissimilarity between two graphs mirrors the approach outlined in the first paper we reviewed:

$$\text{distMCS}(G_1, G_2) = 1 - \left| \frac{\text{mcs}(G_1, G_2)}{\max(|G_1|, |G_2|)} \right|$$

Here, $\text{mcs}(G_1, G_2)$ represents the maximal common subgraph (MCS) shared between G_1 and G_2, while $|G_1|$ and $|G_2|$ denote the sizes of the respective graphs.

```python
def k_nearest_neighbour(train_data, test_instance, k):

    distances = []
    for train_instance, category in train_data:
        distance = mcs_distance(test_instance, train_instance)
        distances.append((category, distance))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    class_counts = defaultdict(int)
    for neighbor in neighbors:
        class_counts[neighbor[0]] += 1
    predicted_class = max(class_counts, key=class_counts.get)
    return predicted_class
```

Figure 4: K-nearest neighbors (KNN)

4.5 Evaluation

For evaluation, we considered the following metrics:

- **Precision**: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It is calculated as $\frac{\text{true positives}}{\text{true positives+false positives}}$.
- **Recall**: Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positives in the dataset. It is calculated as $\frac{\text{true positives}}{\text{true positives+false negatives}}$.
- **F1-Score**: The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision+recall}}$.
- **Confusion Matrix**: A confusion matrix is a table summarizing the performance of a classification model, presenting the counts of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions made by the model.

Initially, we applied the k-nearest neighbors (KNN) algorithm to classify each test instance by comparing it with the training instances. The dataset was split into 20% test data and 80% training data. Subsequently, we utilized the `sklearn.metrics` module to collect the predicted classes and true labels for each test instance, enabling us to evaluate the performance of the classification algorithm.

4.5.1 On 80 20 Split

Following is the result with the graphs on a 80 20 split:

	Precision	**Recall**	**F1-Score**
HealthAndFitness	0.75	1.00	0.86
ScienceAndEducation	1.00	0.67	0.80
Travel	1.00	1.00	1.00
Accuracy	0.92	0.89	0.89
Macro Avg	0.89	0.89	0.89
Weighted Avg	0.92	0.89	0.89

Table 3: Evaluation Metrics

5 Project Management

The entire project was managed on GitHub, accessible here. To enhance collaboration and project tracking, we utilized GitHub's issue tracking system. Each issue served as a milestone, aiding in organizing tasks and monitoring progress throughout the project lifecycle.

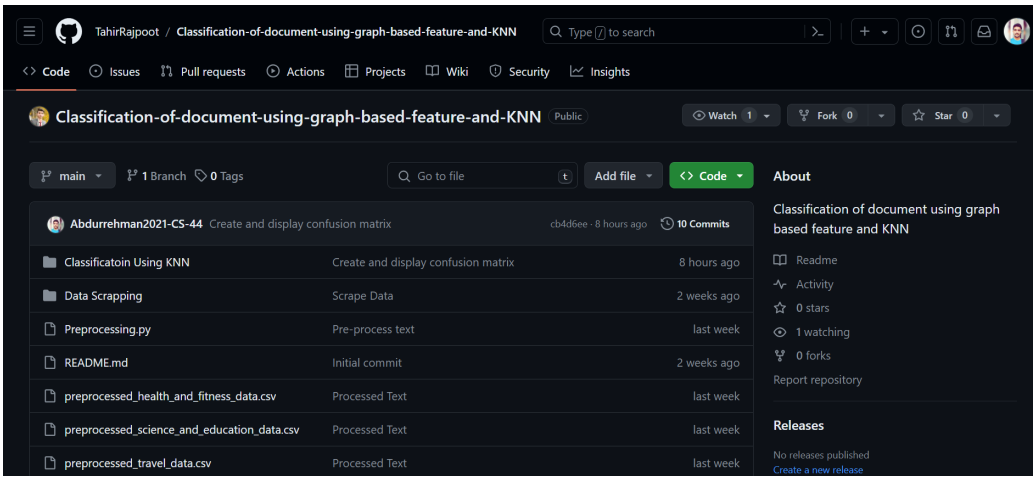

Figure 5: Project Management

6 Challenges Faced

7 Conclusion

In this project, we ventured beyond traditional vector-based approaches and delved into the realm of graph-based methods for document classification. Leveraging the networkx library, we meticulously constructed directed graphs from preprocessed documents, ensuring the preservation of word order and the capture of sequential connections. Our focus on feature extraction revolved around maximal common subgraphs (MCS), proving effective in identifying crucial patterns for classification. Along our journey, we encountered methodological challenges, including complexities in web scraping and algorithm optimizations, which we addressed through meticulous planning and continuous refinement. Despite the inherent difficulty in implementing MCS, our persistent efforts culminated in the development of a robust classification model capable of navigating the intricacies of document classification with remarkable efficacy.