Clustering in monitored hands-on exercises to find common paths and outliers

Zeqi Zhou, zeqi@live.unc.edu

University of North Carolina at Chapel Hill, Department of Computer Science

Abstract

Interest in computer science has been growing, leading to increased enrollment in computer science courses. Hands-on coding exercises are a key component of these courses, allowing instructors to evaluate student understanding and track progress. Code clustering is an effective method for grading, providing collective feedback, and identifying common errors or unique solutions. Despite its benefits, existing research on code clustering has limitations, including subjective evaluations and a lack of focus on outlier detection. Our study assesses the accuracy and performance of three clustering methods and ChatGPT-4 using a newly created labeled dataset. In addition, we propose a new outlier detection algorithm, Weight-based Agglomerative Clustering (WAC), designed to identify unique or erroneous code solutions. We compared our algorithm's accuracy and performance with existing outlier detection methods. Our results demonstrate a clearer understanding of the effectiveness of the evaluated clustering methods. And the evaluation of our outlier detection algorithm indicating our method outperforms existing.

Introduction

Computer science courses rely heavily on hands-on exercises to facilitate learning and evaluate student understanding. These exercises allow students to apply programming concepts in real-world scenarios, offering a practical approach to mastering coding skills. Instructors and teaching assistants play a crucial role in guiding students through these exercises, providing feedback, and assessing their progress.

Teaching assistant systems and clustering methods are invaluable tools in this context, helping educators manage large classes and provide effective feedback. Clustering methods group similar code solutions, allowing instructors to offer collective feedback to students with common coding patterns. This approach streamlines the teaching process and reduces redundancy in feedback, making it easier for instructors to identify and address common errors or misunderstandings.

However, existing teaching assistant systems and clustering methods have limitations. Many rely on user studies to evaluate their effectiveness, with subjective assessments and surveys as the primary means of validation. These evaluations often lack quantitative metrics to measure the accuracy and reduces redundancy in feedback, making it easier for instructors to identify and address common errors or misunderstandings.

Objectives:

1. Created a labeled dataset and evaluated the accuracy and performance of three current clustering methods using different embeddings and Chat GPT.

2. Provided a new outlier detection algorithm and compared accuracy and performance with existing algorithms for detecting outliers

Data & Algorithm

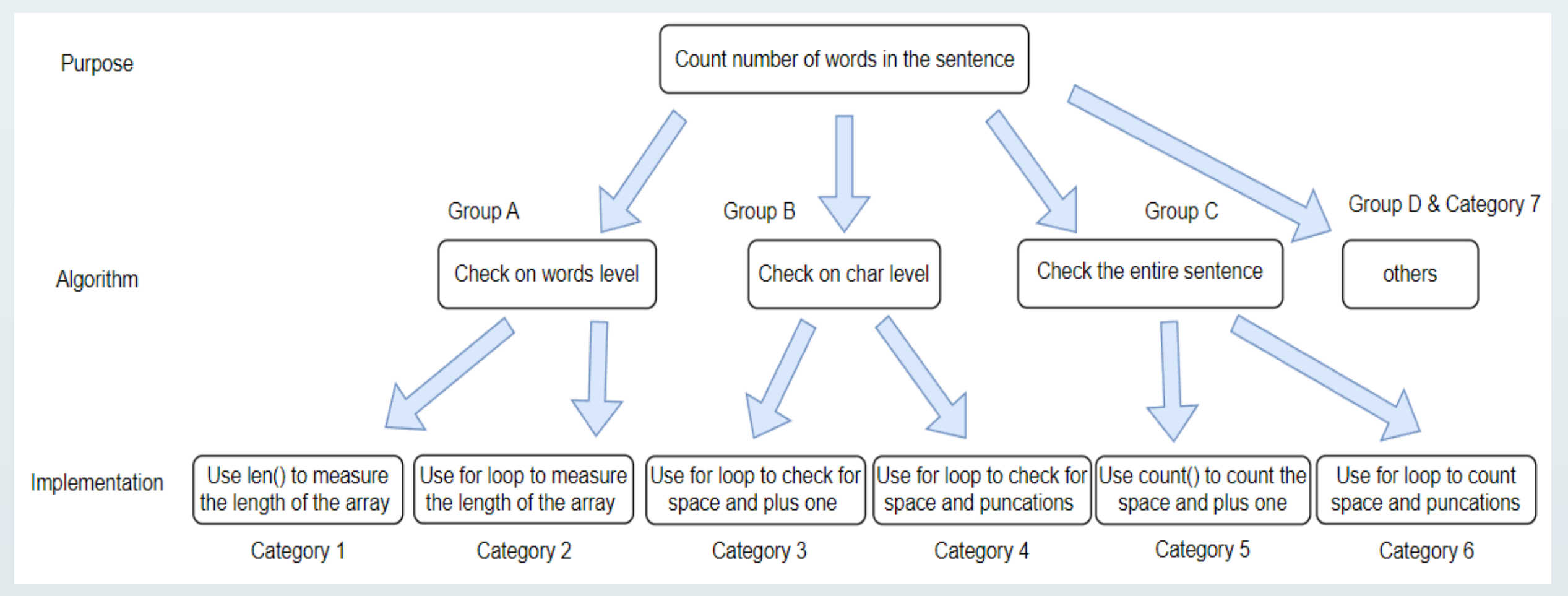
Data source and Labeling

The data used in this study comes from **COMP 116** taught in Fall 2020 by Professor John Majikes, a fundamental-level Python course. The dataset consists of solutions **to two questions from Assignment 2**, submitted by **198 students**. These questions involve basic string manipulation tasks, making them suitable for evaluating code clustering methods and outlier detection algorithms.

Past research in code comparison has categorized code similarity into three levels: purpose, algorithm, and implementation:

- Purpose
  What question does this program solve
  (All the answers are the same at the purpose level)
- Algorithm/Function
  Which algorithms or functions do students use
- Implementation
  How do them implemented their algorithm

According to this idea, we developed a hierarchical labeling scheme that includes these three levels to understand the diverse strategies students use in their hands-on coding exercises.

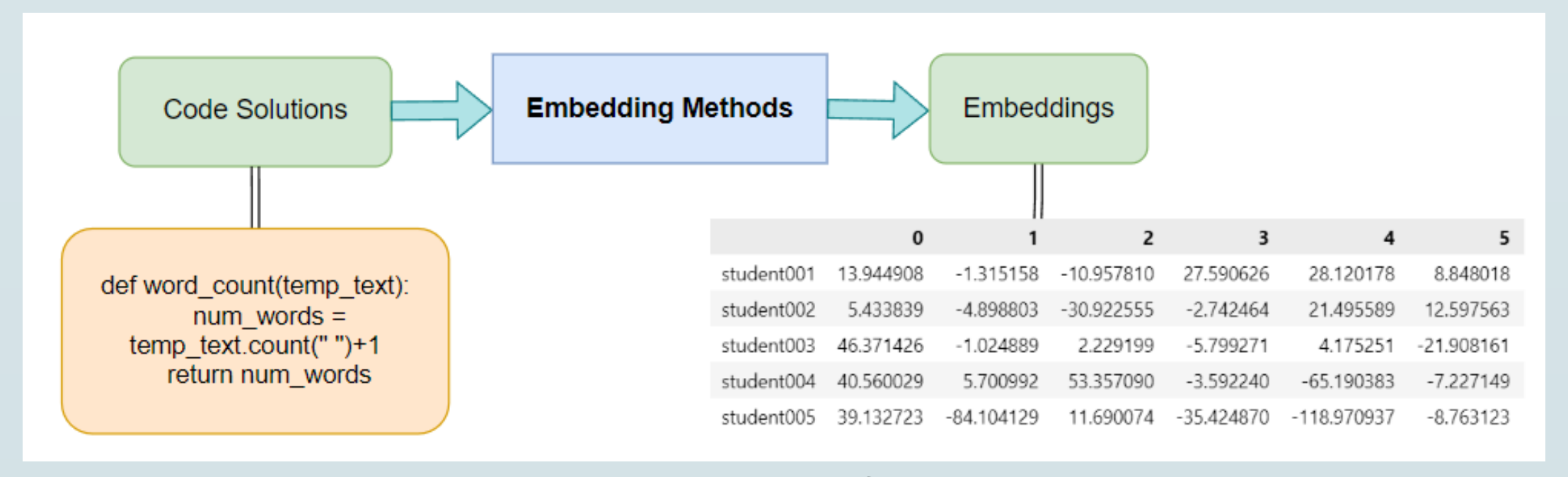

Purpose
Count number of words in the sentence
Algorithm
Group A
Check on words level
Group B
Check on char level
Group C
Check the entire sentence
Group D & Category 7
others
Implementation
Use len() to measure the length of the array
Category 1
Use for loop to measure the length of the array
Category 2
Use for loop to check for space and plus one
Category 3
Use for loop to check for space and puncations
Category 4
Use count() to count the space and plus one
Category 5
Use for loop to count space and puncations
Category 6

Labeling Scheme of Question 2

Weight-based Agglomerative Clustering (WAC) method

An outlier in a coding solution refers to a submission that significantly deviates from the typical patterns found in a set of code solutions. Outliers often contain unnecessary or redundant code, which could indicate innovative approaches or misunderstandings. Thus, we introduced Weight-based Agglomerative Clustering (WAC) method that specifically designed for detecting outlier solutions.

Process Overview

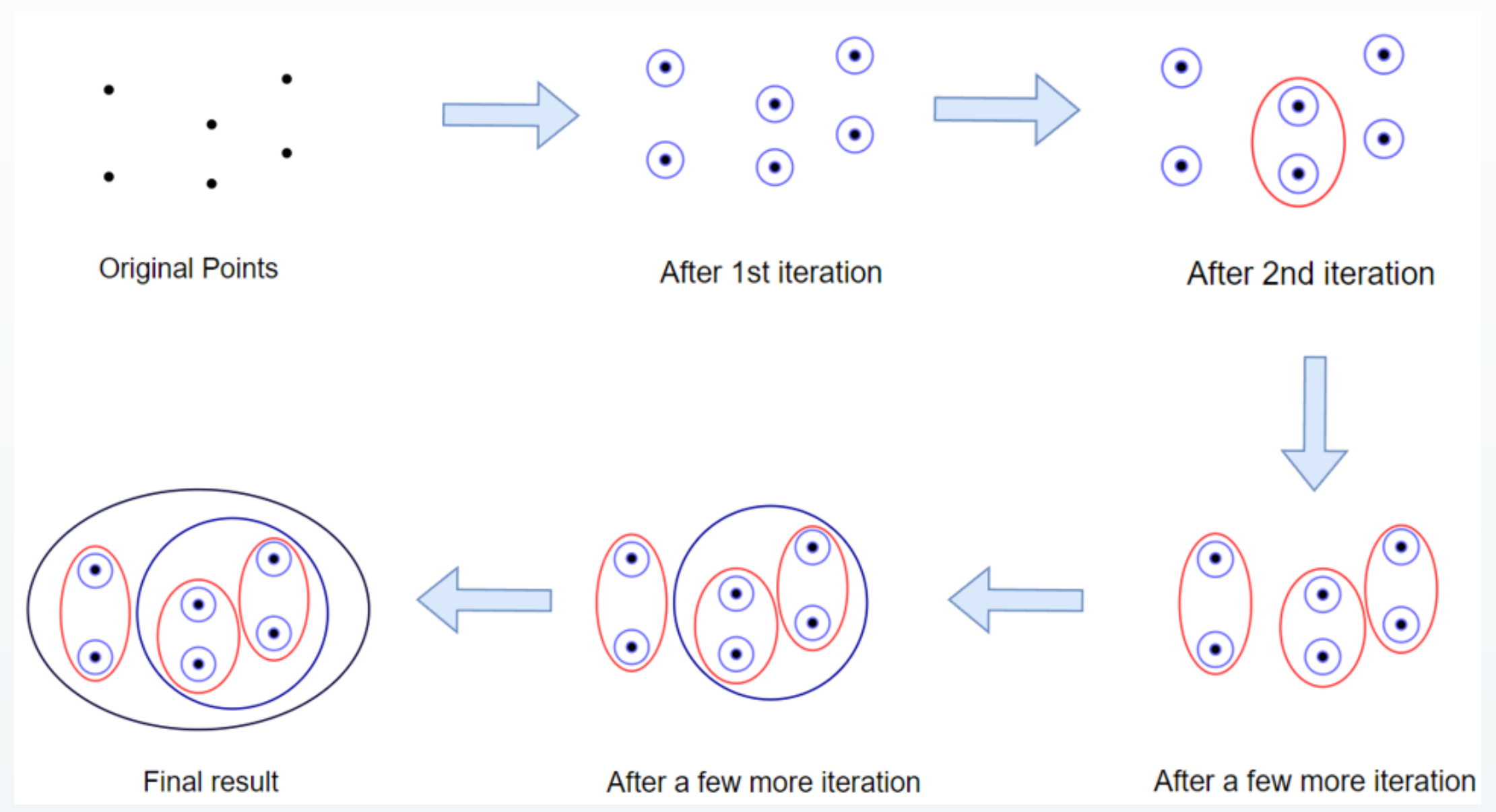


```
def word_count(temp_text):
    num_words = temp_text.count(" ")+1
    return num_words
```

	0	1	2	3	4	5
student001	13.944908	-1.315158	-10.957810	27.590626	28.120178	8.848018
student002	5.433839	-4.898803	-30.922555	-2.742464	21.495589	12.597563
student003	46.371426	-1.024889	2.229199	-5.799271	4.175251	-21.908161
student004	40.560029	5.700992	53.357090	-3.592240	-65.190383	-7.227149
student005	39.132723	-84.104129	11.690074	-35.424870	-118.970937	-8.763123

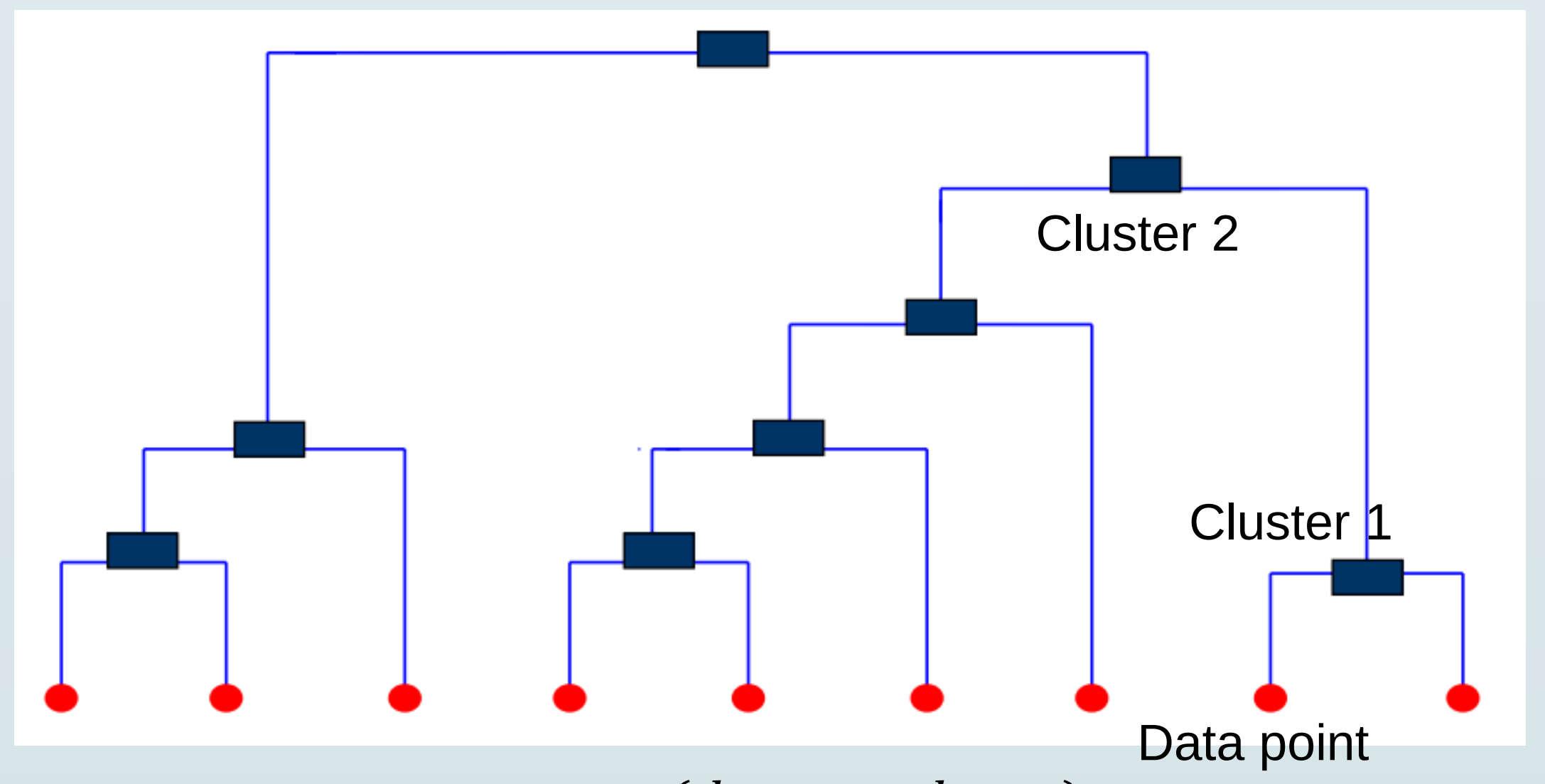
Embedding Transformations

- **Embedding Transformation:** The process begins by transforming code solutions into embeddings using Keywords Count Embedding, which, according to our testing result, is the best embedding for our outlier detection algorithm. This transformation converts each code solution into a data point, providing a numerical representation for further analysis.



- **Agglomerative Clustering:** Each data point as an initial cluster. The distance matrix is then constructed to quantify the differences between code solutions. The algorithm then merges the two closest clusters into a single cluster, updating the distance matrix to reflect the new cluster structure. This process is repeated iteratively, continuously merging clusters until all data points form a single cluster. The sequence of merges is recorded in a dendrogram, a tree-like diagram that visually represents the clustering process and the distances at which clusters were merged.

- **Cluster Weight Calculation:** After constructing the dendrogram, each cluster's weight is calculated. The weight of a cluster is defined as the vertical distance from the cluster to its parent node, divided by the square of the cluster's size. This weighting method helps identify clusters that are significantly distant from others but contain a smaller number of data points, indicating potential outliers.



$$W(C) = \frac{(d_{C_{parent}} - d_{C_{child}})}{{n_{C_{child}}}^2}$$

$$d_C = \textit{the threshold distance of cluster } C$$

$$n_C = \textit{the number of samples in cluster } C$$

- **Outlier Identification:** To identify outliers, the algorithm selects clusters with the highest weights. The expected number of outliers can be adjusted based on the instructor's needs. For example, if the instructor expects five outliers, the algorithm will select the five clusters with the highest weights. This flexibility allows instructors to tailor the outlier detection process to their specific requirements.

Result

The evaluation process involved assessing the accuracy and performance of various clustering methods and comparing the results. This study evaluated three clustering methods—**Edit Distance, Walk Similarity, and Keywords Count Embedding**—along with ChatGPT-4, using a labeled dataset we created.

Training and Testing: The dataset was divided into training and testing subsets, with 66% used for training the clustering algorithms and the remaining 33% for testing.

ChatGPT – 4 was provided with the description of each category but without any labeled sample and then was asked to classify students' solutions.

Algorithm	Question 2				Question 3			
	Algorithm Level		Implementation Level		Algorithm Level		Implementation Level	
	F1	Visual	F1	Visual	F1	Visual	F1	Visual
Edit Distance	0.87	Normal	0.79	Best	0.71	Normal	0.23	Better
Walk Similarity	0.85	Better	0.49	Normal	0.78	Better	0.13	Normal
Keywords Count Embedding	**0.97**	**Best**	**0.71**	**Better**	**0.89**	**Best**	**0.36**	**Best**
ChatGPT-4	0.80	N/A	0.38	N/A	0.48	N/A	N/A	N/A

F1 Score and Visual Evaluation of Different Clustering Methods

	Number of Students (N)					
Algorithm	N = 20	N = 30	N=40	N=50	N=60	N=70
Edit Distance	61.31	88.52	131.56	127.78	163.49	178.78
Walk Similarity	67.71	83.61	93.10	97.52	126.24	135.88
Keywords Count Embedding	**0.23**	**0.24**	**0.34**	**0.41**	**0.44**	**0.53**

Runtime of Different Methods with Different Class Size

Findings:

- important tradeoffs: While the keywords count embedding has the best overall F1, it requires initial efforts to set parameters
- Performance of Walk Similarity is significantly affected by the complexity of the label.
- Keywords Count Embeddings offer the best performance, making it ideal for real-time monitoring
- The runtime of the Edit Distance method demonstrates a significant increase as the number of students (N) grows.

Algorithm	Precision	Accuracy	Outliers Detected	True Positives	Performance (ms)
WAC	**0.93**	**0.93**	**15**	**14**	**769**
DBSCAN	0.79	0.90	14	11	438
OPTICS	0.47	0.83	53	25	344
Isolation Forest	0.73	0.94	33	24	338

Evaluation of Outlier Detection Algorithms

Findings:

- DBSCAN and OPTICS are methods used by previous research on code clustering
- Isolation Forest is a classical outlier detection model
- Our model has the highest precision among the four models