Spring 2023 Semester Project Report

Ali Essonni

June 2023

1 Introduction

Graph embedding techniques have emerged as a popular approach for representing complex data structures. These techniques have found success in various domains, including recommendation systems and social network analysis. One common task in many applications is the comparison and ranking of items. While traditional methods often focus on assigning numerical values to items for absolute rankings, human decision-making processes tend to involve comparing pairs of items rather than providing absolute rankings.

This preference for comparisons over rankings is evident in our everyday life [1]. For example, choosing between two pairs of glasses is typically easier than comparing and ranking different brands of glasses. Recognizing the importance of this cognitive process, we aim to explore the construction of graph embeddings for items using their triplet comparison data and the Bradley-Terry model.

The Bradley-Terry model is a well-established method for modeling paired comparisons and has been extensively applied in fields such as sports analytics [2], psychology, and marketing research. By leveraging the inherent pairwise comparisons present in triplet data, we aim to develop a novel graph embedding approach that captures the underlying relationships between items. We think of capturing these relationships in the notion of shortest path distances between nodes in the representation, meaning that items that are perceived as similar by humans should be either adjacent or have the smallest possible distance, whereas items that are very dissimilar should be far from each other.

The primary objective of this research is to investigate the feasibility and effectiveness of integrating the Bradley-Terry model into the graph embedding framework. Specifically, we seek to answer the question of whether the Bradley-Terry model can be integrated into a graph embedding technique for triplet comparison data.

To achieve these goals, we will conduct a comprehensive comparative analysis using real-world datasets. We will compare the performance of different graph construction methods based on various metrics such as accuracy, number of ties, and mean distances. Four types of graph generation methods will be studied, including unweighted graphs and variations aiming to generate weighted graphs using different strategies.

The outcome of this project is expected to produce at least one promising graph representation generation method that can be further refined and improved in future work. This research is particularly significant as it has the potential to enhance the performance of related tasks, such as "Search based on Comparisons," where users are expected to identify the item closest to what they have in mind by comparing it to a set of items, rather than directly querying for the answer.

2 Related Work

The relevant related work to this project was mainly published by Lucas Maystre, especially his thesis [2], which was on choice models, estimating parameters of such choice models using various methods, and also their applications in the context of sports predictions.
Another important work published by Lucas Maystre and Pr. Matthias Grossglauser [3] in which they present the Iterative Luce Spectral Ranking Algorithm, an algorithm which we will use extensively in our project to create rankings from the triplet comparisons.

On the subject of creating embeddings from triplet comparisons, there has been mostly work on generating euclidian based representations, we can cite the work of Laurens van der Maaten and Kilian Weinberger [4] which introduced the t-distributed Stochastic Triplet Embedding technique or t-STE, this technique aims to accentuate the impact of a triplet on the learning of distances between items, essentially, it collapses similar points and repels dissimilar points in the embedding whenever this does not result in additional constraints violations. With this, the t-STE method achieve some of the most interesting results today in terms of accuracy of the representation.

Our method will essentially aim to achieve similar accuracy results but using a graph embedding instead of a Euclidian on, the hypothesis is that graphs can be as good or even better than Euclidian embeddings in capturing and showing similarities between items.

3 Description of Dataset

To evaluate the proposed graph embedding approaches for triplet comparison data using the Bradley-Terry model, we utilize three distinct datasets from different domains: actor photos, musicians, and foods. Each dataset contains triplet comparisons, where (a, b, c) indicates a preference or similarity relationship between items a, b, and c. Specifically, in the actor photos dataset, (a, b, c) denotes that actor a is facially more similar to actor c than actor b. Similarly, in the musicians dataset, (a, b, c) represents that musician c has a musical style more similar to musician a than musician b. Lastly, in the foods dataset, (a, b, c) signifies that food c is more similar to food a than food b.

3.1 Actor Photos Dataset

The actor photos dataset consists of triplet comparisons sourced from the who-it.at website. This website helps collect triplet comparisons in a playful way. In fact, users are asked to think of an actor in their head, they are shown at each round, a set of 4 actors' images and are asked to click on the actor that most resembles the one they have in mind, then the algorithm put in place should converge to that actor.
This way, every round, assuming that the search finishes succesfully, we generate 4 triplet comparisons, the target doesn't change from round to round, whereas the $'better'$ item is the one chosen by the user, and the $'worse'$ item is each of the 3 other items that were not chosen.
This dataset aims to capture the subjective perception of facial similarity between actors. Each triplet comparison (a, b, c) helps to establish the relative facial resemblance among the actors involved.

3.2 Musicians Dataset

The musicians dataset comprises triplet comparisons reflecting the similarity of musical styles. This dataset comes from the paper [5]. Each triplet comparison (a, b, c) represents the judgment that musician c possesses a musical style more akin to musician a compared to musician b. This dataset encompasses a diverse range of musical genres and aims to capture the intricate relationships between musicians based on their artistic expressions.

3.3 Foods Dataset

The foods dataset includes triplet comparisons that capture the likeness of various food items. This dataset comes from the paper [6], and represents the subjective judgments of individuals regarding the culinary similarities between foods. Each triplet comparison (a, b, c) expresses the preference that food c is more similar to food a than food b. The dataset encompasses a wide variety of cuisines, ingredients, and flavors, reflecting the complexity of food preferences.

By utilizing these diverse datasets from different domains, we aim to assess the generalizability and effectiveness of the proposed graph embedding approach across various applications.

3.4 Useful Statistics

	Actors	Music	Food
# Unique items	552	400	100
# Comparisons	63271	9107	190375
Avg # comparisons per target	166.94	23.59	1903.75
Min # comparisons per target	3	1	1627
Max # comparisons per target	1326	141	2265

Table 1: Statistics of the different datasets

The datasets used in our study show notable disparities in terms of the number of choice items and comparisons they contain.

The foods dataset, being the oldest, contains a significantly larger amount of comparison data compared to the other two datasets. Conversely, the other two datasets provide comparison information on a greater number of choice items than the food dataset. This observation suggests that the foods dataset may yield a more comprehensive representation due to its higher ratio of comparisons to items. However, it is important to consider the possibility that the foods dataset may also include some level of noise, which could potentially impact our ranking method and result in a less accurate representation.

Another noteworthy statistical aspect is the minimum frequency of each target appearing in the dataset. In the actors and music datasets, there are targets with only 1 or 3 comparisons. This indicates the presence of targets for which comparison information needs to be inferred. Consequently, we will account for this disparity in the number of comparisons in one of our graph generation methods.

4 Algorithms

The goal during this project was to come up with and implement different graph representation generation methods based on triplet comparison data. For this, we have created several algorithms which can be found on our Github repository .
The idea was to develop a class that mimics the functionality of existing machine learning (ML) libraries. This class would enable training on a given dataset, while also providing the capability to generate diverse graph representations. Furthermore, the class should be equipped with testing and evaluation mechanisms that allow for the assessment of these generated representations using various relevant metrics.
The proposed class would operate in a manner similar to other established ML libraries, following a standardized workflow. First, it would accept a training dataset, enabling the learning of underlying patterns and relationships within the data. Subsequently, the class would utilize this learned knowledge to generate graph representations that encapsulate the essential characteristics of the original data, as specified by users.
To ensure the effectiveness and quality of the generated graph representations, comprehensive testing and evaluation mechanisms would be integrated into the class. Various relevant metrics would be employed to assess the performance and fidelity of the generated graphs.
The two most important files are: *pairwise.py* and *ilsr_graph.py*. The first one contains utilitarian functions for creating rankings based on triplets comparison data and the Bradley-Terry model, and the second contains the class IlsrGraph, a class which can generate different graph representations after learning parameters for each choice item with respect to each specific target.
In the following sections, we will explain in more details some of the most important algorithms that make up these files and how they are used. Note that this is not an extensive description of the two

files cited above, nor is it a documentation, it is a simple collection of pseudo-codes that explains on a high level what is the procedure followed by some of the most important functions.

4.1 Theory

Before jumping to the algorithms, it is useful to review some of the concepts that we have based our methods on. These concepts range from choice models to graph specific characteristics that are useful to know in order to fully understand the underlying reasons for choosing the methods we chose.

4.1.1 Bradley-Terry Model

The Bradley-Terry model is a choice model which was aimed at ranking chess players based on their match outcomes. It treats each outcome as an independent Bernoulli trial, where every player i is characterized by a $'strength'$ parameter $\gamma_i \in \mathbf{R}_{>0}$ and the probability of player i winning over player j is [2]:

$$\mathbf{P}[i \succ j] = \frac{\gamma_i}{\gamma_i + \gamma_j} \tag{1}$$

In our case, since the results of triplet comparisons are relative to a target, for each triplet (a, b, c) we are interested in : $\mathbf{P}[a \succ b|c]$ which means that every choice item will have a different strength parameter based on the target we are conditioning on.
From this model, we can devise an algorithm for inferring the parameter value of each choice item with respect to a specific target. In this context comes the 'Iterative Luce Spectral Ranking' algorithm, which computes the maximum likelihood estimate of the strength parameters given a list of pairwise comparisons. This algorithm considers the parameters of the model as the stationary distribution of some Markov Chain and computes the transition rates of that specific Markov chain given some observations (number of times i wins over j and vice-versa), then it updates the stationary distribution until convergence.
To use this algorithm, we will have to group the triplet comparisons by target, and feed each batch of triplets to the Iterative Luce Spectral Ranking algorithm or ILSR for short in order to get back parameter with respect to a specific target, and do this for all the targets of the dataset.

4.1.2 Graph Distance

In the following sections, we will extensively use the notion of distance to determine whether our representation is accurate or not, but we first need to define what we mean by distance.
In the context of graphs, we use the notion of shortest path distance or geodesic distance.
In the case of unweighted graphs, it is simply the number of edges in a shortest path connecting two nodes, whereas in weighted graphs, it is the sum of weights of edges forming a shortest path.
A poistively weighted undirected graph alongside the shortest path distance can be seen as a metric space, in fact it satisfies all of the following axioms:

- $d(x, x) = 0$
- $x \neq y \implies d(x, y) > 0$
- $d(x, y) = d(y, x)$
- $d(x, z) \leq d(x, y) + d(y, z)$

Hence, we can consider all of the representations that will be discussed in the upcoming sections, as metric spaces and this will help us accurately and correctly evaluate these representations.

4.1.3 Hyperparameters

After using the *ilsr* algorithm provided by the choix library, we get back parameters for each choice item with respect to each target. We rank the items by descending value of their parameters so that we get a list of most similar items for each target.
At this point, we may choose to connect the target to its k most similar items using unweighted or

weighted edges. While in the case of unweighted edges, the task is relatively straightforward, for weighted edges, there are some details that need to be taken into consideration:

- First on being the noise that might be induced by the triplet comparisons which would reflect on the parameters' values by having differences up to a very large number of digits after the decimal point. We deal with this issue by introducing a hyperparameter called the $'precision'$ which we use to round the values of our parameters up to a certain point, this would ensure that items with the same parameter value up to the 10^{th} decimal place get connected to the target with an edge of the same weight.
- Another issue is: by how much should we increment the edges to a target when connecting to less similar items, for this we introduce another hyperparameter called the $'increment'$. The value of this hyperparameter and the one above are specificly chosen for each dataset in order to maximize the accuracy of the representation.

4.2 Generating parameters and rankings

The first task that we need to perform in order to create our graph representation is to leverage the triplet comparison data to generate a ranking for items with respect to every target. To do this, we make use of choix's *pairwise_ilsr* method that takes triplet comparisons with respect to a single target and generates a parameter for each item.

Algorithm 1 Generating parameters for choice items wrt each target

```
Require: triplets, $\alpha$ (regularization)
  params ← {}
  for target in targets do
    comparisons ← triplets[target]
    items ← list of items compared to the target
    target_params ← choix.ilsr_pairwise(comparisons, $\alpha$)
    // sort the parameters in descending order and change the order of items accordingly
    tuples ← [ ]
    for i in range(lenght of items) do
      tuples[i] ← (items[i], target_params[i])
    end for
    params[target] ← tuples
  end for
  return params =0
```

The function above generates a ranking for items with respect to each target. The ranking will help us connect each target to its k most similar items, whereas the parameters will helps us see by how much the similarity decreases when we go down in rank, this will be helpful in constructing weighted edges. This function can be found in the pairwise.py file of our repository.

4.3 Unweighted Graph Representation

In this section, we will discuss the algorithm by which we construct our unweighted graph representation, the first kind of representation that we tested during this project. Having inferred a ranking with respect to each target using the algorithm in 4.2, we now have to connect each target to its k 'closest neighbours' meaning; the items that have the highest parameter value.

In practice, this function is broken down into 2 separate functions in the *IlsrGraph* class. The first one: *_construct_unweighted_edges*() keeps the first k tuples that appear in the parameters list computed in 4.2 for each target in a dictionary.
Then, *_add_unweighted_edges*() takes the result of the previous function and creates an undirected edge from the target to each of its k neighbours with weight 1, then adds the edges to our graph which is a networkx graph.
This is done in order to ensure modularity and ease of maintainability.

Algorithm 2 Unweighted edge construction

```
Require: params, targets, $k$
Ensure: edges
  edges ← [ ]
  for target in targets do
    for tuple in the first k tuples of params[target] do
      edges.append((target, tuple[0], 1)) // (from, to, weight)
    end for
  end for
  return edges =0
```

4.4 Weighted Graph Representation

4.4.1 Introduction

In the previous section, we discussed our unweighted graph generation method. The second kind of graph representation we tested during this project was the weighted graph representation, where each target is connected to its most similar items with weighted edges.
But a very natural question arises "How do we compute the weights of edges?".
We explored different ways of computing edge weights that we will present in the following sections.
Before starting, one other detail about our implementation of weighted edge construction :
For every weighted graph representation we have a function that generates a dictionary of the following type:

$$\{target_i : \{weight_1 : [neighbour_1, neighbour_2, ...], ...\} \} \tag{2}$$

And another function which takes that dictionary and actually constructs the edges and adds them to our graph representation, we will first present the second function as it is reused by all the edge generation methods.

Algorithm 3 Add weighted edges to graph

```
Require: edges, targets
  edges_to_add ← [ ]
  for target in targets do
    for tuple in edges[target].items() // tuple = (weight, [neighbours...]) do
      for neighbour in tuple[1] do
        edges_to_add.append((target, neighbour, tuple[0]))
      end for
    end for
  end for
  graph.add_weighted_edges_from(edges_to_add) //networkx function =0
```

4.4.2 Weights based on Increment

The function below takes as input *params* which is a dictionary of the form:
$\{target : [(neighbour1, param1), ...]\}$, a list of targets, k a hyperparameter designating the number of neighbours to each target, increment which is a hyperparameter specifying by how much we increment the weight from target to neighbour when we drop a rank, and finally precision specifies by how much we round the parameters (to avoid noise).
It gives as output, a dictionary of the type $\{target : \{weight_i : [neighbour1, neighbour2, ...]\}...\}$ meaning that for each target, we get a dictionary where the key is the weight and the value is a list of neighbours that have to be connected to the target using an edge of that specific weight.
The weight is computed in the following way, first we get a list of unique parameters rounded to a given precision, then the weight of neighbours that have a specific parameter with respect to a given target is:

$$1 + increment \cdot (index\ of\ that\ parameter\ in\ the\ set\ of\ parameters) \tag{3}$$

Algorithm 4 Weighted edge construction based on increment

```
Require: params, targets, $k$, increment, precision
  edges ← {}
  for target in targets do
    target_neighbours ← [ ]
    target_params ← [ ]
    for i in range(k) do
      target_neighbours[i] ← params[target][i][0]
      target_params[i] ← round(params[target][i][1], precision)
    end for
    target_params ← set(target_params) // removing duplicates
    weights ← [ ]
    for i in range(length of target_params) do
      weights[i] ← 1 + i · increment
    end for
    edges_per_target ← {}
    for i in range(length of target_neighbours) do
      for j in range(target_params) do
        if round(params[target][i][1]) == target_params[j] then
          edges_per_targetweights[j].append(params[target][i][0])
          Break
        end if
      end for
    end for
    edges{target} ← edges_per_target
  end for
  return edges =0
```

4.4.3 Weights based on parameter value

Another way of computing the weights of our edges is to take into consideration the value of the parameters, and increase the weights based on the jump in parameter value. In this case the weight of an edge is given by the following recursion:

$$weight_i = 1 + increment \cdot (param_1 - param_i) \tag{4}$$

Algorithm 5 Weighted Edge construction based on parameter value

```
Require: params, targets, k, increment, precision
  edges ← {}
  for target in targets do
    take the list of tuples from params related to target
    sort that list in descending order of the parameter value
    round the parameters up to the precision parameter
    for each unique parameter value in the sorted list, append to edges[weight] the given neighbour
  end for
  return edges =0
```

The idea behind this is to capture more of the information given by the parameters from the Bradley-Terry model, into the weights of our graph's edges. If the parameter value jump from one item to the other is big, the so should be the weights of the corresponding edges. This would yield in an overall more accurate representation and more in line with our expectation.

4.5 Distinction between targets

A final strategy that we tested during this project is to take into consideration the difference between targets in the number of comparisons they appear in, and the number of items they have been compared to. In fact, we hypothesise that the following heuristic could help us create a more accurate representation.

$$\frac{\#comparisons}{\#neighbours} \tag{5}$$

The actors with a high value of the above heuristic are considered as having enough information to infer an accurate ranking for the items they have been compared to, whereas the targets which have a low value of the heuristic don't have enough information to infer an accurate ranking.
The two types of targets should not be influencing our representation on the same scale. We should first base our representation on the rankings with respect to confident targets (the ones with high value of the heuristic) and then add on the information given by the less confident actors changing as little as possible the existing structure of the graph.
For this, we introduce the concept of baseline neighbour

Definition 1. *Each target that is not an isolate in the graph can have a baseline neighbour, which is the first adjacent node with lowest edge weight connecting it to the target, that also appears in the ranking of the target.*

We will connect each target to its neighbours in a sequential way following the ordering of their heuristic value (highest first). The weight of each edge will be computed in the following way:

$$weight_i := weight_{baseline} + increment \cdot (param_i - param_{baseline}) \tag{6}$$

Having this goal in mind, we create a function that does the following:

Algorithm 6 Weighted graph based on heuristic

```
Require: targets, params, k, increment, precision
  compute the heuristic value for each target
  rank the targets based on their heuristic value from highest to lowest
  connect the first target to its k neighbours in the graph
  while at least one edge is added do
    for each target do
      if target is not an isolate in the graph then
        Find baseline_weight
        Find baseline_neighbour
        Compute weights of edges using formula (6)
        Connect target to its first k neighbours using computed edges
      end if
    end for
  end while=0
```

4.6 IlsrGraph

We package all of the above algorithms into a single class *IlsrGraph* which can create a graph representation based on triplet comparison data. A typical usecase of the *IlsrGraph* class would involve calling the following functions:

4.6.1 Init

```python
def __init__(self, nodes):
    """
    :param nodes: list of unique nodes
    Initialize an undirected graph with given nodes
    """
```

Code Listing 1: __init__

The init function takes in a list of all unique items being compared in the triplet comparison data, and generates a networkx undirected graph with these items as nodes in the graph.

4.6.2 Fit

```
def fit(self, train, alpha=None):
    """
    :param train: dataframe of columns ['better', 'worse', 'target']
    Computes parameters for each node wrt to each target using the ilsr
        pairwise algorithm
    """
```

Code Listing 2: fit

The fit function takes the training data (triplet comparisons) as input as well as a regularization parameter α, and feeds them to choix's ilsr_pairwise function to get back a list of tuples (item, parameter) with respect to each target.

4.6.3 construct_graph

```
def construct_graph(self, k, weighted=False, increment=0.01, precision=10,
    from_params=False):
    """
    :param k: int, number of edges to add to each node
    :param weighted: bool, whether to add weights to edges
    :param increment: float, parameter by which to increment weights
    :param precision: int, rounding precision for parameters
    :param from_params: bool, whether weight should be function of parameters'
        value jump
    :return: graph representation from ilsr parameters
    """
```

Code Listing 3: construct_graph

The construct_graph function is a function that constructs a graph representation based on the parameters it receive. It can construct an unweighted graph, weighted or weighted based on parameter value.

4.6.4 list_of_graphs

```
def list_of_graphs(self, end, step=1, weighted=False, increment=0.01,
    precision=10, from_params=False, stats=False, test=None):
    """
    :param end: int, iteration limit
    :param step: int, iteration step
    :param weighted: bool, whether to add weights to edges
    :param increment: float, parameter by which to increment weights
    :param precision: int, rounding precision for parameters
    :param from_params: bool, whether weight should be function of parameters'
        value jump
    :return: list of graph representations from ilsr parameters
    """
```

Code Listing 4: list_of_graphs

The list_of_graphs function as its name suggests creates a list of graph representations where the parameter that is changing is k: the number of neighbours of each target, or for the params weighted graph, the number of iterations. It is especially important because it helps reduce the computation time when creating a lot of graph for testing purposes. It can also test every newly created representation to reduce even more running time, in that case the output is the list of graphs as well as the statistics.

5 Results

In this section, we will present the results of our different representations based on the datasets that have been gathered.
Our representations were tested based on different relevant metrics such as:

- The accuracy defined as follows: For each triplet $t_i = (a, b, c)$ we define the following function: $A_i = \mathbf{1}_{\{d(c,a)<d(c,b)\}} + \frac{1}{2} \cdot \mathbf{1}_{\{d(c,b)<d(c,a)\}}$. Then the accuracy is $\frac{1}{n} \cdot \sum_i A_i$
- The mean distance to 'better' and 'worse' items for all the targets. This will help us see how the 'better' and 'worse' distance to target evolve when adding more and more edges. Logically both should decrease, but we should see a difference between the two such that it is on average more costly to go to the worse item that to go to the better one.
- The number of connected components. We would ideally have one giant connected component encompassing all the compared items, but it is possible that some small components still be present. In general, we would like to see how this number varies as the number of edges in the representation grows.
- The fraction of ties. This number is particularly important when combined with the accuracy, as it gives us a more comprehensive understanding of the performance of the representation. Since a tie comes down to making a guess, we want as little ties as possible in our representation.
- The average degree of our representation. We would like to see how this number evolves with the number of iterations of our algorithms.

Note that we don't have a ground truth representation on which we might be able to test our specific techniques, we are essentially trying to learn the distances between items given triplet comparisons between these items, this is why we are more interested in the ability to predict correctly a test triplet, rather than quantify by how much items are similar or dissimilar.
We say that our representation correctly predicts the triplet (a, b, c) if the distance from a to c is smaller than that from b to c in the representation.

5.1 Expectations

Let us first discuss what would be considered as a good representation.
We have to put in place what would be considered as an ideal representation, in order to improve on our iterations.
The first condition would be to accurately predict 80% or more of the triplet comparisons and anything above 90% accuracy would be considered very good.
Our representation also needs to have a low number of ties. We define a tie as the following:

Definition 2. *A representation is said to tie on the triplet* (a, b, c) *if* $d(a, c) = d(b, c)$

As the more ties we have, the more our representation becomes random, in the sense that if we query for the 3 most similar items to a specific target, and we get that the first 5 most similar items have the same distance to the target, then we will be forced to make a random choice. This behaviour is not wanted, especially when we are trying to traverse the graph in the context of a comparison based search algorithm [7].
We also ideally want our representation to be connected (every pair of vertices (u, v) should have at least one path connecting them), but this depends on the quality of our dataset and on the value of the ratio :

$$\frac{\#comparisons}{\#items\ being\ compared\ to} \tag{7}$$

For each target. The value of the heuristic might be very low for some targets, thus we might not be interested in those targets at all as they have a smaller chance of being queried for.
Finally, we want our representation to be sparse, meaning that we have to achieve very high values of the accuracy for a very low number of edges added.

5.2 Unweighted Graph representation Results

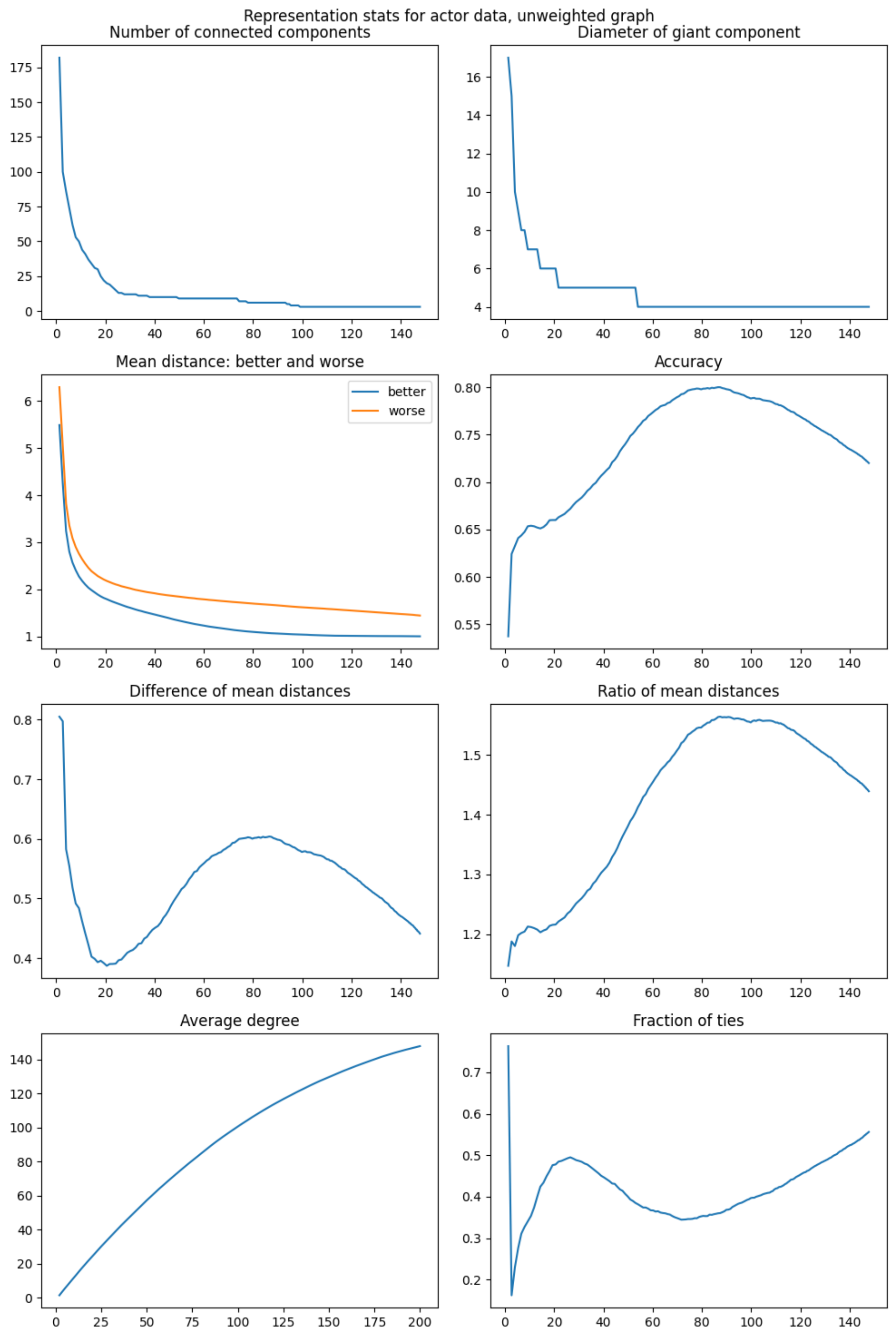

Figure 1: Statistics for Unweighted representations on actors dataset

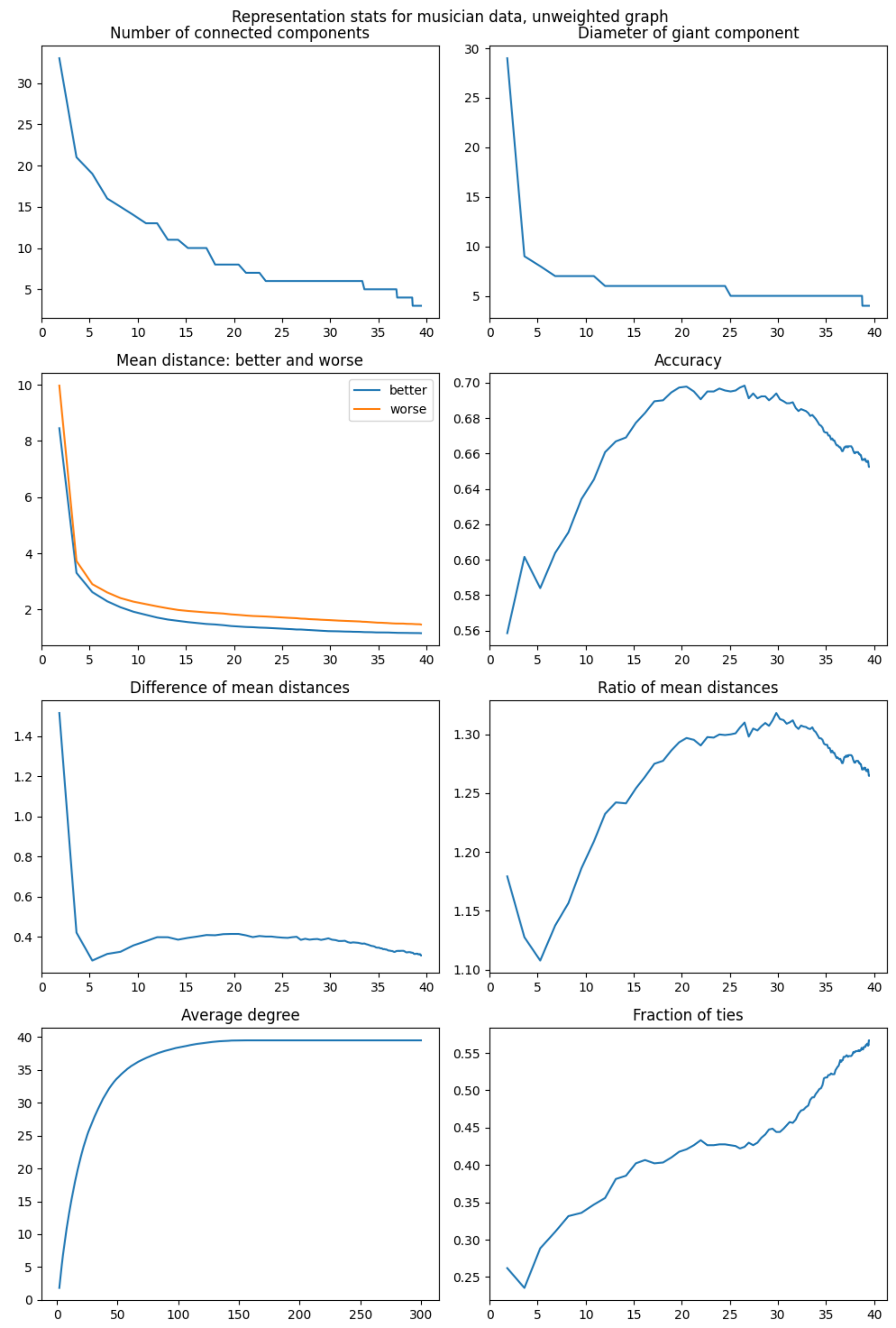

Figure 2: Statistics for Unweighted representations on musicians dataset

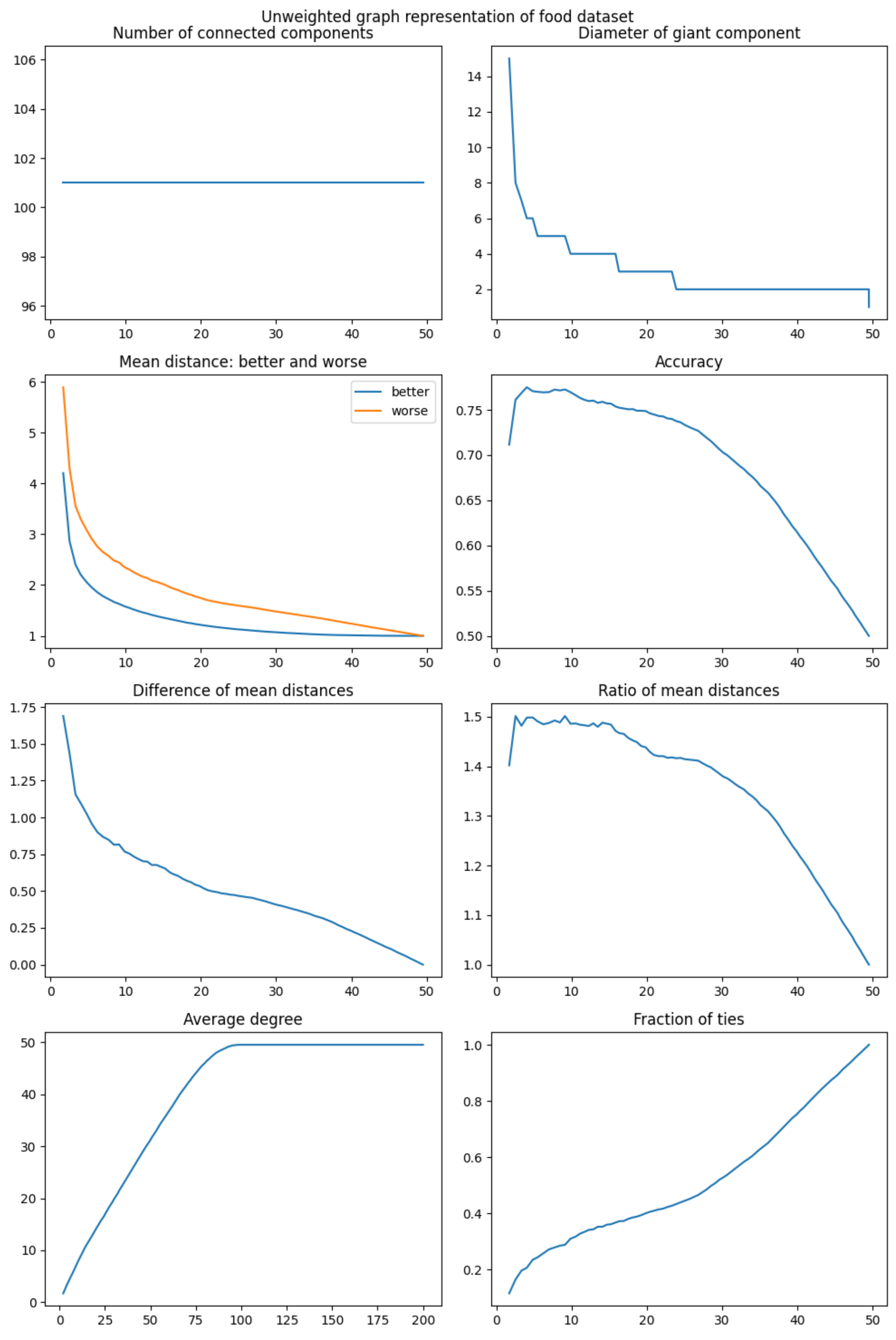

Figure 3: Statistics for Unweighted graph represetation on food dataset

The unweighted graph representation yielded accuracies of 0.8, 0.7, and 0.76 for the actors, musicians, and foods datasets, respectively, which indicates a promising initial outcome. However, it is evident that the proportion of ties is considerably high, surpassing 0.55 in some cases. This observation implies that our representation retains a significantly greater level of randomness than desired. This phenomenon can be attributed to the continual addition of edges with uniform weight (i.e., all edges possess a weight of 1) since this is an unweighted graph, which means that for every triplet (a, b, c) in the test dataset, if the ranking with respect to c is the following: And c is connected to

Rank	**Item**
1	a
2	d
3	b
4	e

Table 2: Ranking with respect to target c

its 3 most similar items, then a and b would have the same distance to c rendering the answer of our representation to this triplet (a, b, c) a simple random guess.
This can also be seen in the difference of mean distances to better and worse, which both converge to 1 as the number of added edges grows.

5.3 Weighted Graph representation Results

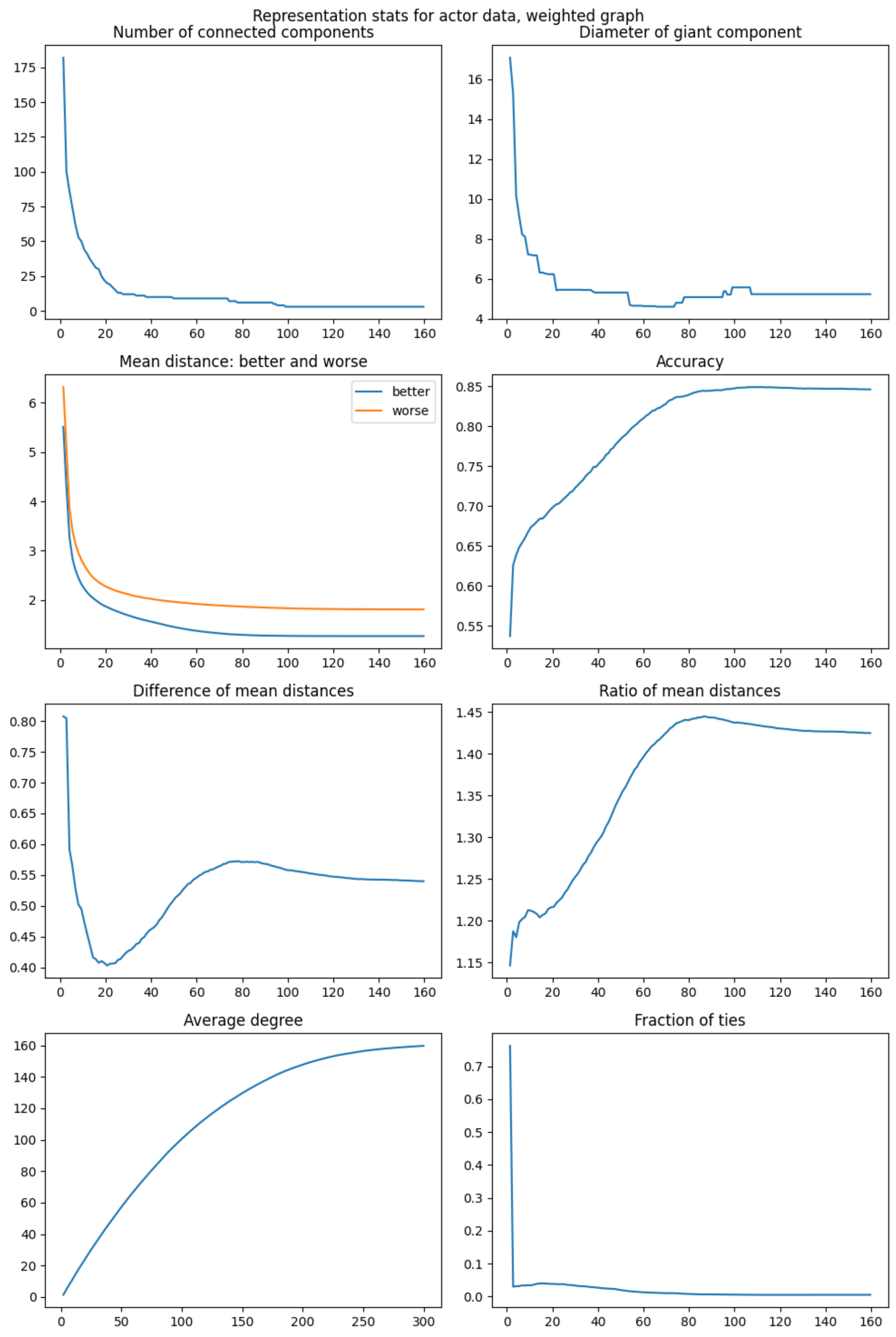

Figure 4: Statistics for Weighted Representations on actors datasets

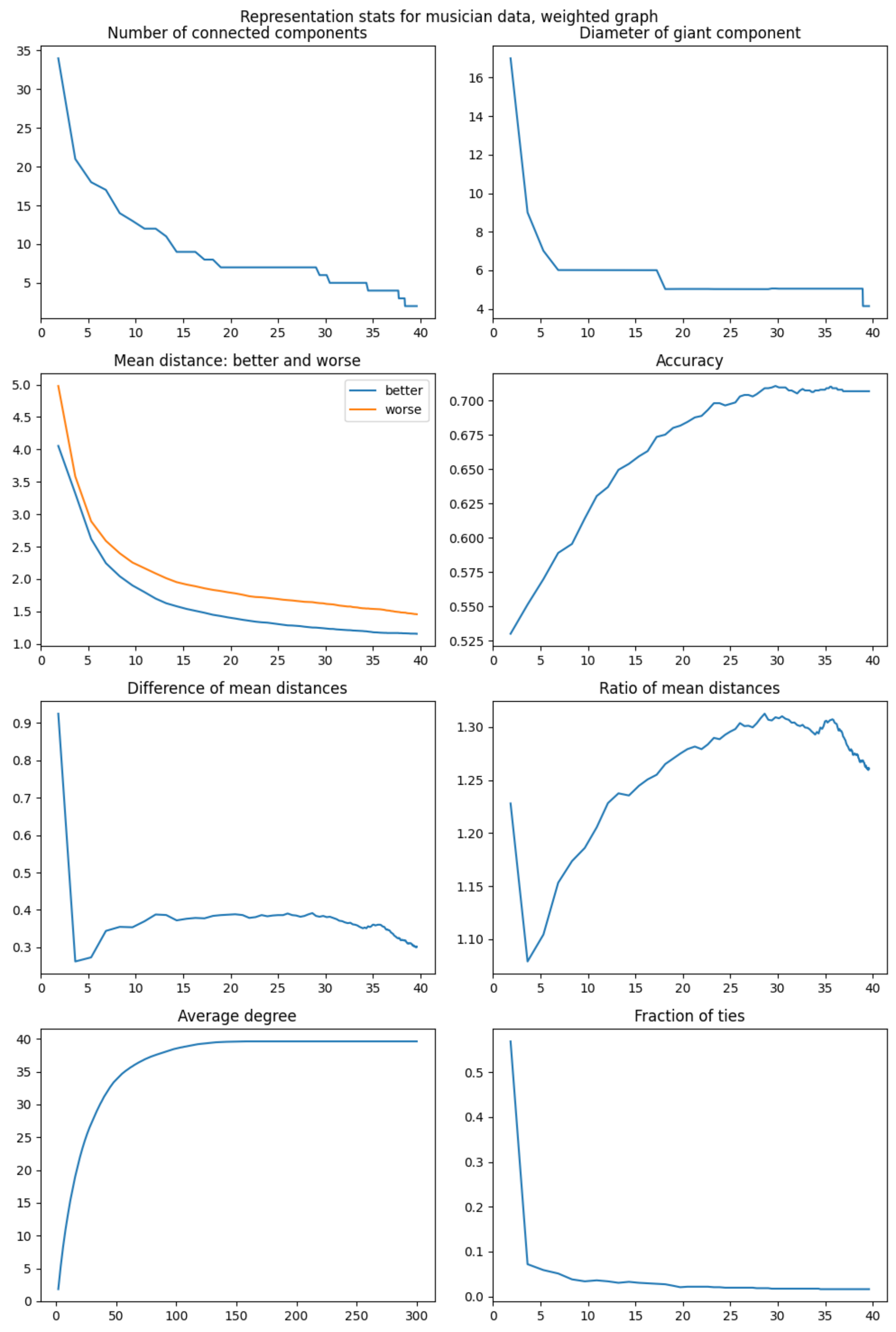

Figure 5: Statistics for Weighted Representations on musicians datasets

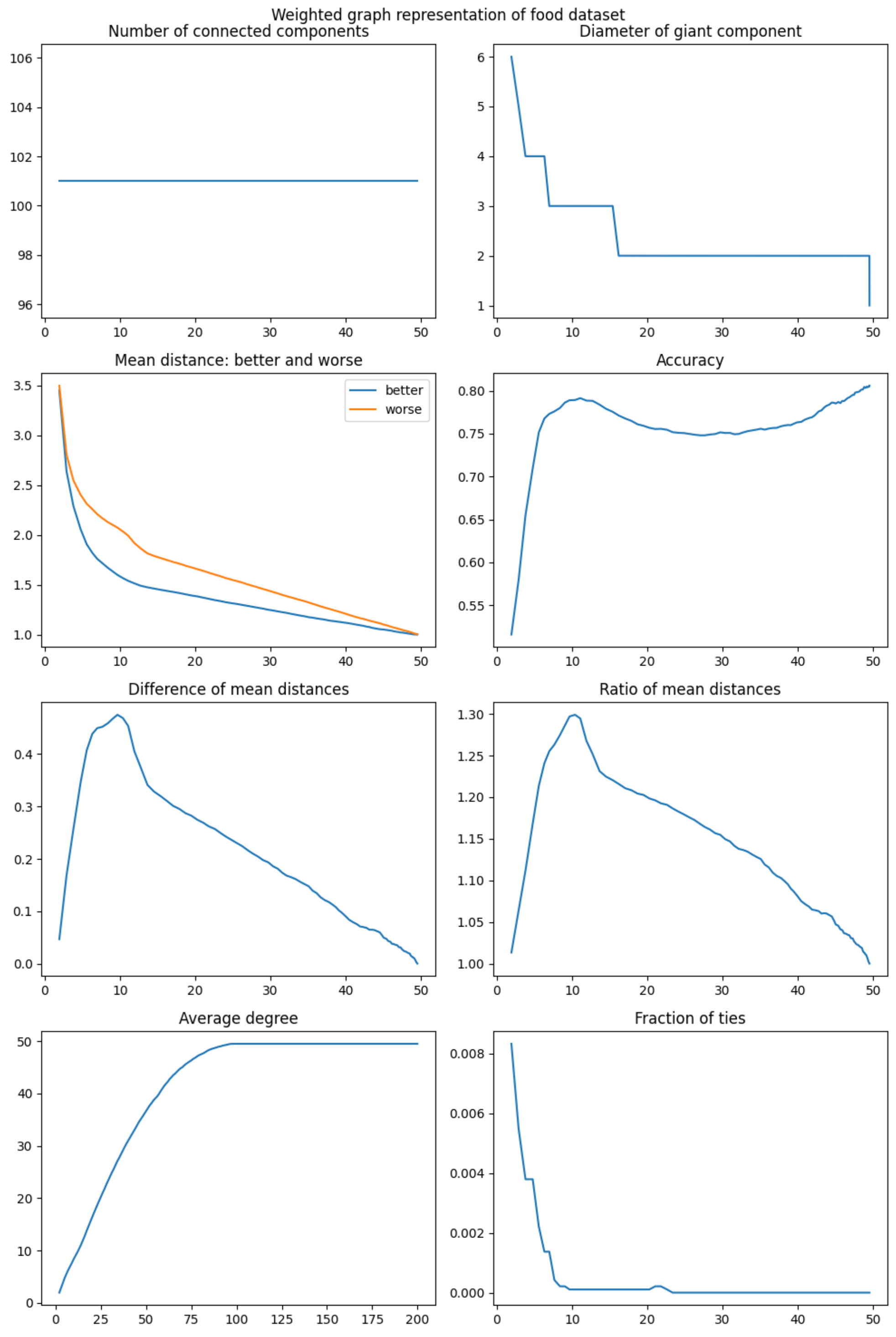

Figure 6: Statistics for Unweighted graph represetation on food dataset

The weighted graph representation was aimed at removing as much ties as possible by adding an increment to the edges of items of increasing rank. This would ensure that even if a target is connected to its better and worse items, it is still more costly to go the worse node than to the better one. From the figures, we can see that the accuracy of our representation goes up: 0.85, 0.70, 0.80 for the actors, musicians and foods datasets. But the most important point is that the fraction of ties is converging to a considerably smaller value fr the 3 datasets. This means that indeed, adding edges of different weights decreases the number of ties in our representation as expected.
But now that we have delt with the high number of ties, we need to increase the accuracy of our

representation for the 3 datasets, and to do this, we perform cross-validation in order to choose the best possible hyperparameter value for each dataset.
Then we move on to testing a slightly different graph generation approach.

5.4 Weighted Graph based on params values

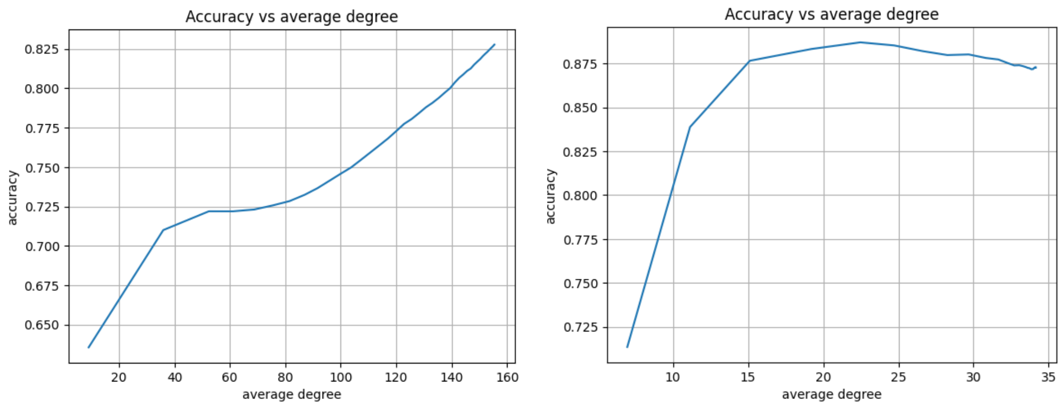

Figure 7: Accuracy of Actors' and Musicians' Graph Representation based on params values

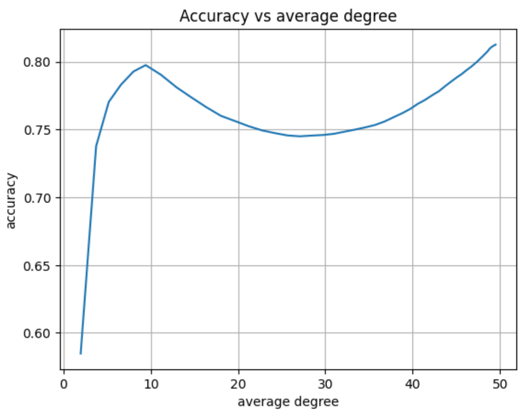

Figure 8: Accuracy of Foods Graph Representation based on params values

We can see that we achieve an accuracy of 0.83, 0.88 and 0.81 for the actors, musicians and foods datasets respectively.
We can thus confidently say that this last graph generation approach is overall the most accurate one, and has the desired properties of having as little ties as possible and having the mean distances to better and worse converge to different values.
However, a drawback of this approach lies in the fact that the highest accuracy is attained when the graph's average degree is at its peak, indicating that greater accuracy is achieved by adding more edges. This behavior is not appreciated as it means that in order to maximize the accuracy of our representation, each target should be connected to all its neighbours.
We would rather prefer, a representation which converges to very high values of accuracy for lower average degrees of the representation.
This would be particularly useful for comparison based search algorithms which have to traverse the graph. It is better in this case to have a sparse graph rather than a dense one, as it means less computations and smaller convergence times.

5.5 Representation Visualizations

Having examined the effectiveness of our representations, we can now proceed to visually analyze them to determine if they align with our understanding. Specifically, we expect items that we perceive as similar to be positioned as neighboring nodes in the graph or have a relatively short distance (shortest path), while dissimilar items should remain unconnected, or have a very long shortest path.
To visualize the representations, we use the spring layout, which treats edges as springs holding nodes together, and nodes as repelling objects, the lower the edge weight, the closer the nodes are. This is the closest we can get to a euclidian representation style.
We can clearly see, in the case of the foods and actors datasets, two distinct clusters of items, in the actors dataset, we can clearly see that the two clusters separate actors of different sex, whereas in the foods dataset, the two clusters are for sweet and salted foods. On the other hand, for the musicians dataset, the representation is a bit more chaotic, we can't clearly see clusters of choice items, and this is in line with our expectations as music genres are very vast, and we would need a considerably higher amount of comparisons than the number of triplets present in the current dataset, in order to see some structure in the graph representation.

5.5.1 Actors' Graph Representation

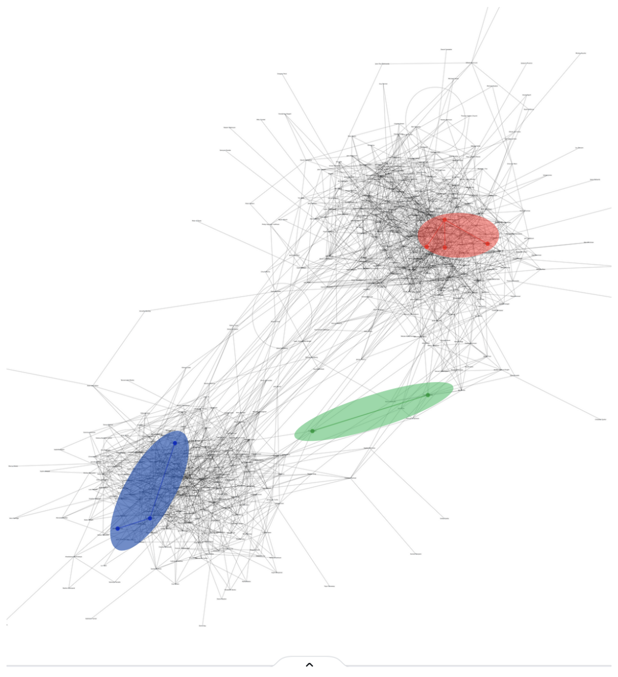

Figure 9: Actors' graph representation, 10 neighbours

The actors' graph representation exhibits two clusters that are clearly visible in the upper cluster, we can see that actors *Matt Damon*, *Leonardo DiCaprio*, *Chris Hemsworth* and *Ryan Gosling* are adjacent in the graph and very close to each other. The same with actors *Salma Hayek*, *Penélope Cruz* and *Rosa Salazar* which are in the lower cluster.
These two results are very in line with our expectations since the actors in each of the sets are very similar in terms of facial traits. However, there are some edges that make less sense, for instance the one in green between *Amy Smart* and *Eddie Murphy*, these two actors have very different facial traits, yet they are adjacent in the graph.
Although it hurts the accuracy of the representation, these edges help gather the nodes into a single connected component, which is a desired feature of the representation.

5.5.2 Musicians Graph Representation

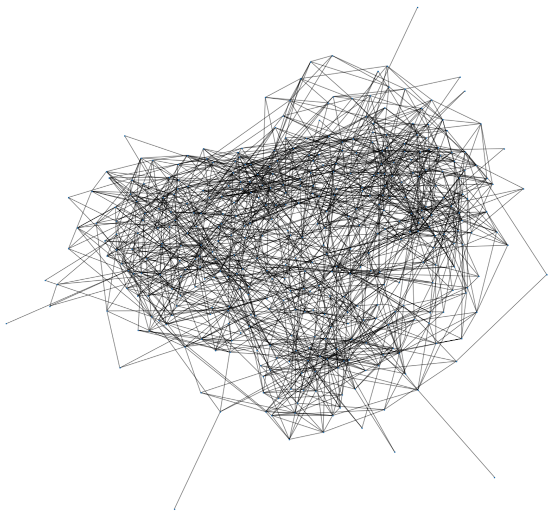

Figure 10: Musicians Graph Representation, 5 neighbours

For the musicians dataset, the story is very different. The graph representation by connecting each target to its 5 most similar choice items, doesn't show much underlying structure. A possible explanation was given in the introduction of part 5.5. But we can say in summary that relationships between music styles are more complex than those of human faces, there are no two obvious clusters, but there may be multiple different clusters that are connected to each others in some sense, and we might need more data than what we have in order to clearly visualize the different clusters.

5.5.3 Foods Graph Representation

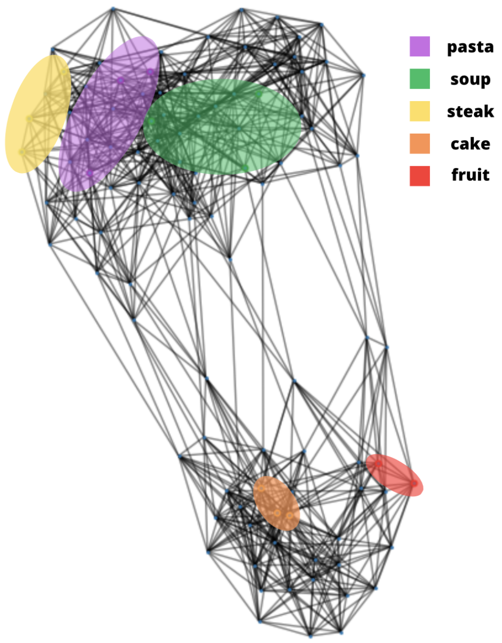

Figure 11: Foods Graph Representation, 10 neighbours

Finally, for the foods dataset, we can see that our representation is very accurate and in line with our expectations, salted and sweet foods are separated, and we can even see small clusters of food types in these two clusters, for instance, in the salted foods, we see that steaks are adjacent to each other, as well as soups and pasta, whereas for sweet foods, we can say the same for cakes and fruits.

6 Discussion and Conclusion

In summary, the primary goal of this project was to address the crucial issue of developing an effective graph representation using the Bradley-Terry model and leveraging triplet comparison data. The aim was to train the representation in such a way that when presented with the question "which item from the set $\{a, b\}$ is more similar to c?", it would consistently provide the accurate answer. In this context, the notion of the "correct answer" refers to the consensus among humans regarding which item should be considered more similar to c.

By emphasizing the consensus among humans, the project aimed to align the graph representation with our collective understanding of similarity. Human consensus, derived from the collective knowledge and intuition of individuals, serves as a benchmark to evaluate the accuracy and effectiveness of the representation. This approach acknowledges that notions of similarity can be subjective, but it strives to capture the commonly agreed-upon patterns of similarity.

As stated in the results section, some criteria that our representation needs to satisfy are the following: an accuracy of over 80% for the datasets we are testing on, the representation should be a connected graph, or at least have a giant component encompassing most of the items in the datasets, and finally, the representation should be sparse, meaning that it should achieve a high accuracy for a low number of edges added.

After having tested multiple graph construction algorithms, we conclude that the algorithm presented in section 4.4.3 yields the best possible representation when considering the criteria above. The graph representation employed in this study establishes connections between each target item c in triplets of the form (a, b, c) and its k most similar items. These connections are determined based on the rankings derived from the Iterative Luce Spectral Ranking algorithm, and edges are assigned increasing weights as we go down in rank.

This representation consistently achieves 82% or more accuracy for the different datasets, it has a very low ratio of ties, meaning that there are very few edges with the same weight for each specific target, and it yields a connected graph for relatively small values of k.
All of these characteristics are needed for a representation that could eventually be the backbone of a comparison-based search algorithm.

We also conclude that this representation yields meaningful visualizations of clusters of similarities between items that are being compared, especially when there are two obvious clusters in the dataset, namely, sweet and salted foods for the food dataset and male, female for the actors' face images dataset. Whereas for items that have more complicated relationships, notably, music genres, we see that the representation doesn't give us a satisfying visualization of clusters or genres even though it achieves a high accuracy.

Having talked about the qualities of our representation, it is also worth pointing out some of the limitations of our graph construction method, the first one being computation time, which is relatively high for a construction algorithm that would be used for constructing multiple graphs in a row. The bottleneck being the ILSR algorithm which takes a considerable amount of time to converge to the parameters of the Bradley-Terry model.
The second limitation is the fact that our graph isn't as sparse as we might want it to be. One way by which we can mitigate this issue is by constructing an originally dense graph then sparsifying it without changing much of the underlying structure, this has been part of the work done in the context of my colleague Augustin's semester project.

7 Acknowledgements

Implementing our algorithms, we made use of some python libraries that made it convenient for us to concentrate on the high-level problems we were trying to solve, these libraries are the following:

- Pandas [8]: a very useful data analysis and manipulation tool, especially used to load our datasets and perform various actions on them.
- Numpy [9]: a fundamental package for scientific computing using python, numpy provides a list of very useful functions that also make dealing with data using python much easier.
- Choix [2]: a python library that provides inference algorithms for models for models based on Luce's choice axiom, this library was used in order to infer the parameters of the Bradley-Terry model using the Iterative Luce Spectral Ranking algorithm.
- NetworkX: a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks

I would like to thank EPFL for this amazing opportunity of working alongside researchers and trying for ourselves a research project to give us more perspective for future career choices.
I would also like to thank Surya for his guidance and continuous feedback throughout the semester that helped me overcome a lot of challenges, and helped me get better at various subjects.
I would also like to thank Pr. Matthias Grossglauser for his insightful suggestions during the midterm presentation and also towards the end of the semester, it helped me think of different ways we can improve our results.
Finally I would like to thank my colleagues Christelle and Augustin with whom I had a lot of discussions and who had great ideas that were implemented both in this project as well as their semester projects which were related to triplet comparisons and graph embeddings.

References

[1] Ammar Ammar and Devavrat Shah. "Ranking: Compare, don't score". In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. Sept. 2011, pp. 776–783. DOI: 10.1109/Allerton.2011.6120246.

[2] Lucas Maystre. "Efficient Learning from Comparisons". eng. PhD thesis. Lausanne: EPFL, 2018. DOI: 10.5075/epfl-thesis-8637.

[3] Lucas Maystre and Matthias Grossglauser. "Fast and Accurate Inference of Plackett– Luce Models". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/2a38a4a9316c49e5a833517c45d31070-Paper.pdf.

[4] Laurens van der Maaten and Kilian Weinberger. "Stochastic triplet embedding". In: *2012 IEEE International Workshop on Machine Learning for Signal Processing*. 2012, pp. 1–6. DOI: 10.1109/MLSP.2012.6349720.

[5] Daniel Ellis et al. "The Quest for Ground Truth in Musical Artist Similarity." In: Jan. 2002.

[6] Michael J. Wilber, Iljung S. Kwak, and Serge J. Belongie. *Cost-Effective HITs for Relative Similarity Comparisons*. arXiv:1404.3291 [cs]. Apr. 2014. DOI: 10.48550/arXiv.1404.3291. URL: http://arxiv.org/abs/1404.3291 (visited on June 12, 2023).

[7] Daniyar Chumbalov, Lucas Maystre, and Matthias Grossglauser. "Scalable and Efficient Comparison-based Search without Features". en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 1995–2005. URL: https://proceedings.mlr.press/v119/chumbalov20a.html (visited on June 7, 2023).

[8] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[9] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.