Universiteit Utrecht

Faculty of Science

Master Thesis

Artificial Intelligence

Identifying Topic-Specific Opinion Leaders Through Graph Embedding

---

Author:
Luuk Buijsman (6665500)

Primary supervisor:
Dr. S. Wang
Secondary supervisor:
Dr. M.W. Chekol

March 8, 2022

1 Abstract

In social networks, some users have an exceptional ability to affect the opinions and behaviours of others. Such people are known as opinion leaders. Accurately identifying opinion leaders can assist greatly in the study of information flow throughout social networks, in addition to providing valuable insights for marketing purposes. Furthermore, contemporary graph embedding tools can significantly improve the process of identifying opinion leaders. Despite this, little research has focused on combining graph embedding and opinion leader detection. Consequently, this thesis focuses on research that integrates these two areas together. I do this by first creating graph embeddings that capture the information contained in the data. Then, I feed these embeddings to an opinion leader detection algorithm that is designed to use the information captured by the embeddings to create a ranking of users, with the highest-ranking users being the designated opinion leaders. I compare these results with several benchmarks of opinion leader detection methods that do not use graph embeddings. The quality of opinion leaders is similar for each method, though not all models designate the same users as opinion leaders. To conclude, I discuss this research in the broader context of the field of graph embedding and opinion leader detection, and provide suggestions for further research. All code used in this thesis can be found on GitHub[1].

[1] https://github.com/LuukBuijsman/Master-Thesis

Contents

2 Introduction

2.1 Background

One of the main distinctive features of the contemporary internet is the way it allows people to interact with each other like never before. It allows them to convey their opinions regarding a particular topic and engage in discussions not just with acquaintances such as friends, colleagues and family members, but also with complete strangers. Consequently, social networking websites

such as Twitter, Facebook, and Reddit, which serve as essential communication platforms aimed at facilitating these interactions, are among the most popular websites on the contemporary internet [1]. On these websites, some users have a remarkable ability to influence the opinions and viewpoints of others. Through their activity on social media, these users can affect many other people, thereby shaping the general public opinion on certain topics. Consequently, such users are known as opinion leaders [2]. Perhaps unsurprisingly, the task of finding these opinion leaders by distinguishing them from "regular" (non-opinion leader) users in a social network is known as Opinion Leader Detection (OLD) [3].

In the area of online communication, microblogging is becoming increasingly more popular [4]. This form of communication is characterized by short messages which typically contain a real-time update of the user's life, or cover a certain thematic. Twitter is arguably the most well-known microblogging website, and, as mentioned earlier, one of the most popular websites in general. It allows users (also known as *twitterers*) to publish short messages called *Tweets* with a maximum length of 280 characters. Twitter also functions as a social network by allowing users to *follow* other users (without requiring permission). The Twitter homepage (also known as the *timeline*) then shows a stream of Tweets from twitterers a user has chosen to follow (their *friends*), so they can keep up-to-date with their latest developments and easily interact with Tweets sent by their friends. The main forms of interaction between users comes from liking, replying to, or Retweeting a Tweet. When someone Retweets a Tweet, they publish an exact copy of the Tweet to their followers, even if the followers do not follow the original author of the Tweet.

As mentioned earlier, OLD involves finding opinion leaders in a social network. A common way to represent data in a large variety of real-life situations, including social network data, is through the use of graphs. They consist of a series of nodes as well as edges connecting the different nodes together [5]. Graphs will be discussed in more detail in section 3.1. Every OLD method discussed in this paper also uses a graph for its analysis. Given the importance of graphs, it is no surprise that there exists a large number of models dedicated to analysing them in order to discover information hidden inside. Such analytical methods include node classification [6], link prediction [7], node clustering [8] and visualizing large-scale high-dimensional data [9].

For instance, a graph can be constructed from social media data based on user interactions. This graph can then be used to detect certain communities or anomalous individuals, or it can be used to predict whether two users will interact with each other in the future.

Despite the widespread use of graphs and graph analytical methods, there are some major downsides in the form of high computational and spatial costs. Consequently, a significant portion of contemporary research focuses on improving upon these aspects of graph analysis. One application that is gaining popularity in recent years is graph embedding, which involves converting a graph into a low-dimensional space in which the information contained in the original graph is maintained [5]. Such an embedding typically comes in the form of a series of low-dimensional vectors representing the graph which enable both high compu-

tational and spatial efficiency. While most embedding methods only consider the structural information contained in a graph in order to construct the graph embedding, there are several methods which enable the use of extra information (such as semantic properties) in creating the embedding [10]. Furthermore, existing embeddings can be refined and augmented using semantic information [11]. Since these two extensions of classical embedding methods both add extra information to the embedding process, the quality of the resulting graph embedding could be higher than traditional graph embeddings that only consider structural information.

2.2 Motivation

This research will focus on the intersection between graph embedding and opinion leader detection applied to social media data. Finding opinion leaders might prove to be lucrative for marketing purposes, as they can be used to promote products or services and raise awareness to certain topics [12]. Furthermore, since opinion leaders have a large influence over the spread of information throughout a network, finding and studying them can aid in understanding the flow of information throughout a social network [2]. Generally, opinion leaders are selected based on all data available from the social network [3].

However, it stands to reason that a user can be an opinion leader regarding a certain topic, but not other topics. For example, if a user is an opinion leader on physics it does not necessarily mean that this same user is also an opinion leader on politics. Because of this, the TwitterRank [4] method is proposed which aims to find opinion leaders for each topic available in the data. In this research, I will also endeavour to distinguish between opinion leaders for different topics as opposed to 'general' opinion leaders which are agnostic to the topics present in the data.

The types of graphs that are typically used for OLD only contain structural information [13]. However, social media data contains much more information that could be relevant for finding opinion leaders, such as semantic information that can be extracted from the data. Incorporating this extra information into the experiments might result in higher-quality analysis. Some of the graph embedding methods briefly mentioned above serve as a way to include semantic information in the embedding in addition to structural information. As such, this paper will use a graph embedding for the process of finding opinion leaders.

Furthermore, graph analytical methods such as opinion leader detection suffer from high computational costs because of the high-dimensional data in graphs [3, 14], which severely diminishes the scalability of these methods. Graph embeddings offer a solution in this regard, by converting a graph into a low-dimensional vector. By applying opinion leader detection methods to these embeddings, their computational efficiency increases greatly, which allows these analytical methods to be applied even to large-scale graphs.

As such, because graph embeddings allow better scalability and offer the potential to include additional information for the opinion leader detection methods, these algorithms are a valuable tool to better identify opinion leaders in

social networks.

2.3 Research focus

This paper will focus on defining a new method for detecting opinion leaders by using graph embedding methods. The graph embedding method that will be used allows for the inclusion of semantic information in addition to structural information from the graph, which could lead to embeddings of higher quality. As such, the first research question is as follows:

- What is the influence of including both semantic and structural information in a graph embedding on opinion leader detection, compared to using only structural information?

As discussed above, one of the goals of this research is to find topic-specific opinion leaders through topic-specific embeddings. The TwitterRank method [4] also creates topic-specific rankings for their opinion leaders. However, this method does not use embeddings, whereas my method will. Therefore, the second research question will concern the effect of embeddings on the OLD process:

- What is the influence of using graph embeddings on opinion leader detection?

Since the embeddings can include semantic information, it is important to decide which features to include in the creation of the embedding. Specifically, the stance of a user regarding a certain topic (be it positive or negative) might have a major effect on finding opinion leaders [15]. Therefore, the third research question is dedicated to determining the effect of this feature specifically:

- What is the influence of a user's stance (positive or negative) on detecting opinion leaders?

Finally, the available data spans multiple months. Consequently, the data can be split by creation date. This allows the investigation of how opinion leaders develop over time in a social network. For this reason, the final research question is:

- How do opinion leaders develop over time in a social network?

To answer these questions, I will use a large collection of Dutch Twitter messages regarding COVID-19 [16]. I will use these data as a case study to compare several methods with the one I propose in this thesis.

2.4 Outline

The next section will review related work on both graph embedding and OLD methods when applied specifically to social media data. Following this review, I

will outline how I have the most appropriate methods for both graph embedding and OLD to be used in my experiments.

Section 4 will cover the Twitter data used in the experiment, the data collection process, as well as an explanation of the baseline dataset. In section 5, I will give an in-depth explanation of the algorithms I used in my experiments in addition to explaining the construction of the graph for the experiments, as well as outlining the features from the data that will be used for the construction of the graph.

In section 6, I explain how I am going to conduct the experiments using the methods as defined in section 5, how I will evaluate them and explain the hyperparameter settings that I explored during my research.

Then, in section 7 I showcase all results from the experiments and analyze them as well, while I will also answer the research questions.

Afterwards, in section 8 I will have a general discussion about the results and place them in the broader context of the fields of both graph embedding and opinion leader detection.

Finally, in section 9 I conclude with an outlook on how to proceed given the results of this thesis as well as some interesting avenues to explore in future research.

3 Literature review

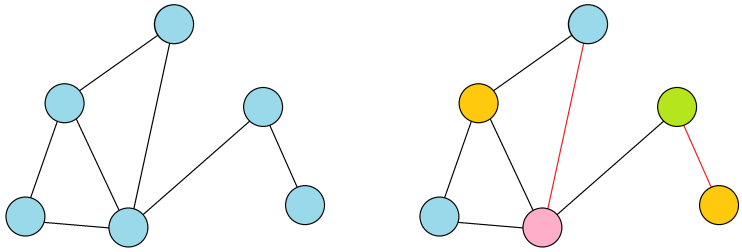
3.1 Graph Embedding

The two types of graphs that I will discuss in this section are *homogeneous graphs* and *heterogeneous graphs*. A homogeneous graph is a graph in which all nodes belong to a single type and all edges also belong to only one type. Conversely, a heterogeneous graph is a graph in which there are multiple types of nodes and/or multiple types of edges [17]. A graph with multiple types of nodes but only one type of edges is also known as a node-heterogeneous graph, and unsurprisingly, a graph with multiple types of edges but only one type of nodes can also be called an edge-heterogeneous graph [18]. Figure 1 shows examples of both a homogeneous and a heterogeneous graph.

Not all graph embedding methods have been shown to work with both homogeneous and heterogeneous graphs. As such, when deciding upon a suitable graph embedding method, it is of vital importance to consider the desired type of graph, as this can limit the number of available options. Most embedding methods will function properly with homogeneous graphs, and when a method has also been shown to work with heterogeneous graphs I will specifically indicate this.

Graph embedding methods can be divided up into several classes [17]:

- Matrix Factorization
- Edge Reconstruction
- Graph Kernel



(a) Homogeneous graph (b) Heterogeneous graph

Figure 1: (a) shows a homogeneous graph where the nodes and edges all have the same type; (b) shows a heterogeneous graph with four types of nodes and two types of edges.

- Generative Models
- Deep Learning

3.1.1 Matrix Factorization

Embedding methods using matrix factorization involve depicting a graph as a matrix, then factorizing that matrix to create the embeddings [5]. As such, this typically involves converting a high-dimensional representation (the matrix) into a low-dimensional representation (the factorization), while at the same time preserving the information contained within the graph (structural properties). Methods that use matrix factorization construct the matrix representation of the graph either using Laplacian eigenmaps [19, 20, 21] or through node proximity [22, 23, 24].

Embeddings created using this class of methods generally have good performance on local graph reconstruction problems [5] and link prediction problems [19]. However, they neglect global structural properties of the graph in favour of local structural properties, which might affect performance when information regarding those properties is desired. Furthermore, the high computational complexity makes it difficult to scale these methods up to large graphs. Finally, using the node proximity to create the matrices typically result in overfitting, which negatively impacts the reliability of these methods [14].

3.1.2 Edge Reconstruction

This class of embedding models attempts to ensure that edges that are created based on the node embeddings are as close to the edges in the original graph as possible. That objective can be accomplished by either maximizing the edge reconstruction probability, or by minimizing the edge reconstruction loss [17].

Maximizing the edge reconstruction probability is done by maximizing the probability of re-creating all observed edges in the original graph using the node embeddings. In order to achieve this, every node pair that was connected by an

edge in the original graph should be connected in the embedding as well [25]. Furthermore, every node pair that was not connected in the original graph also should not be connected in the embedding. This is modelled through pairwise node proximity: nodes that are close to each other in the embedding space are likely to be connected by an edge in the original graph [7].

Conversely, minimizing the edge reconstruction loss builds on the insight that an edge between two nodes indicates the relative importance of these two nodes to each other: nodes that are connected to a target node are more relevant to that target node than nodes that are not connected to it [17]. Therefore, minimizing the edge reconstruction loss can be modelled as creating similar embeddings for connected nodes, while creating dissimilar embeddings for unconnected nodes [26].

Since these models focus so heavily on pairwise node proximity, the embeddings prioritize local structure over global structure. This means that edge reconstruction-based models are unsuited for modelling entire graphs, but rather are designed to embed small communities within graphs instead.

3.1.3 Graph Kernel

Contrary to the edge reconstruction methods discussed in the previous section, the graph kernel methods focus primarily on properly representing the global graph structure, as opposed to the local structure [27]. It accomplishes this by breaking up a graph into subgraphs (also known as *graphlets*), and embedding these subgraphs based on how often they occur in the complete graph [28].

The primary drawback of kernel methods is that they fail to concisely capture local structure of a graph, but focus only on the global structural information. As such, information regarding the local structures is lost.

3.1.4 Generative Models

Generative models create graph embeddings by modelling all nodes in a graph as a vector of its latent variables [29]. The intuition behind this way of creating embeddings is that the original graph was generated by some underlying model, which the embedding attempts to approximate [17]. Node embeddings that are placed close to each other in the embedding space correspond to nodes close to each other in the original graph as well, which enables the embedding to accurately model the graph.

An advantage of generative models is that it is possible to use more than merely structural information to create embeddings, which allows the embedding space to also incorporate semantic information. However, these types of models are notoriously difficult to train, and as such require a lot of data to be trained properly [30].

3.1.5 Deep Learning

In the area of graph embedding, a large portion of the most commonly-used methods applied specifically to social media data fall under the umbrella of

deep learning methods. In this class of models, a clear distinction can be made between models that use random walks, and those that do not. A random walk is defined as a stochastic process starting at a certain node where a random neighbouring node is chosen every step [31]. The probability of choosing a neighbouring node is often dependent on several factors, such as the weight of the edge between the starting node and candidate target node. As such, it is typically used as a measure of the similarity between two nodes. In this way, a graph is represented in the embedding as a series of paths generated by random walks. Nodes that are similar to each other in the original graph will also share similar embeddings due to the random walk procedure, whereas nodes that are dissimilar will have different embedding values [17]. Therefore, creating embeddings using random walks preserves the structural properties of the graph.

Models using random walks Embedding models that use random walks to create their embeddings typically are easily scalable because multiple random walks can be executed at the same time, allowing for easy parallelization [17, 31]. This means that they can be applied to large networks (such as certain communities on social networking sites such as Twitter and Facebook) with relative ease. Running these models will not take as long as some of the other types of embedding models, while also offering high spatial efficiency, which could make them a very attractive options in certain circumstances. Furthermore, embeddings created through random walks are also very robust: if a small part of the original graph changes, there is no need to re-train the entire model. The existing model can be updated by replacing the random walks in the affected region of the graph with new random walks. This way, only the new random walks will have to be run on the new graph while allowing the remainder of the random walks to remain unchanged. Therefore, adjusting the model to small changes in the original graph takes little time, which adds to the flexibility and robustness of these methods.

However, the random walk typically conveys more information regarding a node's local neighbourhood within a path, which in turn means it might not be able to capture the global structure of the original graph compared to the local structure. This might affect the quality of the embedding and its down-stream applications, especially if those applications rely heavily on the global graph structure, and not so much on the local structure. Some methods attempt to alleviate this problem through several means, as I will discuss below.

Node-based embeddings The first major model using such random walks for creating its graph embedding is the DeepWalk model [31]. It takes as its input a graph (either homogeneous or heterogeneous) and outputs a matrix of latent vertex representations as the network embedding in the form of vectors. It maximizes the probability of observing the last k nodes and the next k nodes in the random walk using an optimization method inspired by the Skip-gram architecture [32], thereby preserving higher-order proximity between nodes: it

retains the structure of the original graph. The model generates multiple random walks each of length $2k + 1$ and performs this maximization at each step. One great advantage of DeepWalk compared to most other graph embedding methods is that it has been shown to work for both homogeneous and heterogeneous models.

Similar to DeepWalk, node2vec [33] preserves higher-order proximity between nodes. However, contrary to DeepWalk, node2vec instead uses a biased-random walk strategy with the aim of balancing between a breadth-first search (BFS) and a depth-first search (DFS). Choosing the right balance between these two strategies enables node2vec to preserve community structure as well as structural equivalence between nodes. Furthermore, this balance between BFS and DFS also allows node2vec to more effectively capture the global structure in a graph, which, as mentioned above, is typically one of the weaknesses of graph embedding models.

While both DeepWalk and node2vec both embed the data based solely on structure, social networks often contain information about the users (nodes) that extends beyond mere structural information. This is what the Attributed Social Network Embedding (ASNE) model [10] attempts to capture. It learns representations by preserving both structural proximity and attribute proximity to take advantage of attribute homophily: the observation that people similar to each other tend to become friends [34]. Following this principle, it places users close to each other in vector space based on both structural and attribute information. As such, it outperforms node2vec on social media data.

Even though ASNE has not been tested on heterogeneous graphs, the attribute information can be used to create differences between nodes. As such, even though technically all nodes are of the same type (namely simply a user in the social network), by assigning different attributes to nodes they can functionally still become different from each other, even in a homogeneous graph.

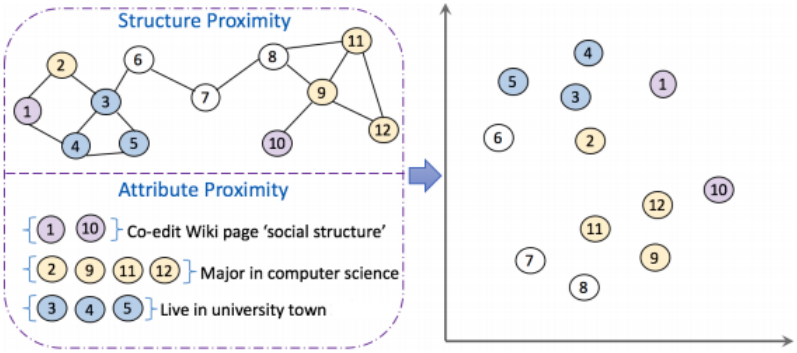


Figure 2: An illustration of social network embedding. The numbered nodes correspond to users. Users with the same colour share the denoted attribute. Image taken from [10].

As illustrated in figure 2, the ASNE model attempts to place nodes together based on both their structural information as well as their attribute information. For example, nodes 1 and 10 are far away from each other in the original graph, but the embedding places them closer together because they share the same value for a particular attribute. However, even when placing these two nodes closer together, they are both also clearly placed close to their structural neighbours.

Since the DeepWalk, node2vec, and ASNE models all function similarly and have all been successfully tested on social media data [10], I will be using these three models in my thesis to create the embeddings. As such, I will discuss these approaches in more detail in section 5.

Edge-based embeddings The aforementioned embedding methods are all node-based. Even though node2vec has a way of indirectly representing edges, it is still important to represent edges directly rather than indirectly, because an edge representation vector generated from the embedding vectors of its endpoints cannot preserve the complete properties of this edge, as argued in [35]. Hence, the authors propose edge2vec: an edge-based embedding method aimed at directly embedding the structural information held by edges in a social network [35]. The main applications are related to link prediction and finding similarly structured communities within the overall network. In order to preserve both global and local structural information, edge2vec combines a deep autoencoder with a Skip-gram model. The authors found the performance of edge2vec to be higher than other node-based embeddings like node2vec on edge-based applications (such as link prediction).

Another situation in which edge-based embeddings could be more useful than node-based embeddings is when trying to compare pairs of nodes with each other. This way, the edge embedding allows easy comparison of node pairs. For example, when predicting the probability that two users in a social network will interact with each other (link prediction), the edge embedding outperforms other state-of-the-art node embedding models [36].

Models not using random walks Embedding models that do not use random walks to create their embeddings typically require no feature engineering and are better at capturing both local and global structural information from the original graph compared to models using a random walk. However, the lack of a random walk typically results in significantly higher spatial and computational costs, meaning that such models might be less viable options for certain projects [17].

Most embedding methods struggle to learn a proper embedding when given an extremely sparse, heterogeneous network. Nonetheless, the Deep User-Image Feature (DUIF) model [37] is capable of transforming sparse, heterogeneous data into homogeneous, low-dimensional representations. This is done through maximizing a modularity measure which represents how well a network is partitioned. The modularity is maximized indirectly by training a deep convolutional neural

network to learn intermediate features of the data. This results in a computationally efficient deep learning framework capable of creating embeddings even for very sparse, heterogeneous networks.

3.1.6 Evaluation methods

When creating graph embeddings an important question to consider is how good the graph embeddings actually are. Obviously, "good" is a rather vague term: it depends entirely on what the graph embedding will be used for. Sometimes, a graph embedding might fail to capture certain nuances in the data. However, if this information is not important to the down-stream application that the embedding will be used for, then it will not detract from the quality of the model. Conversely, if the embedding method fails to capture a key aspect of the information contained in the original graph, the embedding might not perform well at all on its intended task.

Node Classification Keeping this important consideration in mind, the first commonly-used task to evaluate the quality of graph embeddings is node classification [38]. This task involves correctly labelling the nodes in a graph from a finite set of categorical values. Applications for which node classification is a useful evaluation method include the assigning of users to groups such as demographic values or political alignments [39]. Node classification works by training a classifier on the node representations created by the graph embedding models, and evaluating the classifier using traditional classification evaluation metrics such as Macro-F1 (arithmetic mean F1-score of all per-class F1 scores) and Micro-F1 (Global proportion of correctly classified observations) [40]. The most commonly-used classifier used for node classification is the logistic regression model [10, 31, 41].

Link Prediction The second popular evaluation method for graph embeddings is link prediction, which involves finding missing links in networks, as well as predicting which links might form in the network in the future [42]. Applications of link prediction include predicting the complete network in case of missing information [43, 44] or recommender systems which help people find new friends in social networks [45], recommend experts in academic networks [46], or provide alternative and/or interesting products in online shopping [47]. In essence, link prediction measures an embedding's ability to capture the structural properties contained in a graph.

Link prediction works by predicting the probability that an edge exists between two given nodes. This creates a probability score for all tested node pairs. We can then consider node pairs with $P > 0.5$ to have an edge, and node pairs with $P <= 0.5$ to not have an edge between them [48, 42]. Now we have a classification problem, to which we can compare the test labels to determine the performance of the classifier.

Some metrics that can be used to evaluate these scores are traditional metrics such as accuracy, precision, recall, and F1-score. Another metric that can be

used is the area under the receiver operating characteristic (AUROC) which measures the degree to which a classification model is capable of distinguishing between different classes [48]. In this case, the AUROC measures how well the classifier can distinguish between positive test examples (node pairs that share an edge) and negative test examples (node pairs that do not share an edge). The AUROC is often used to evaluate link prediction tasks [10, 49, 50, 51, 52, 53].

Because it is used often to evaluate contemporary graph embedding methods, including the three methods I use in my thesis (DeepWalk, node2vec, and ASNE), I will use the link prediction task and the AUROC measure to evaluate the quality of my graph embeddings.

3.2 Opinion Leader Detection

The methods related to Opinion Leader Detection (OLD) discussed here will be split up over two main categories: topological measures, and data mining and machine learning methods [3].

3.2.1 Topological measures

Firstly, topological measures are characterized by their focus on identifying nodes as opinion leaders based on their structural location within the network. These methods rank nodes based on their position in the network, and the highest-ranking nodes are consequently selected as being opinion leaders. The PageRank [54] algorithm is a widely-used ranking algorithm for determining the importance of web pages. The main idea behind PageRank is that the importance of a node is related to the importance of its neighbouring nodes and it is based on the eigenvector centrality measure. In its original implementation, PageRank was created to determine the importance of webpages. The intuition behind PageRank is that if a webpage is referred to often by other webpages then it is likely an important webpage. Furthermore, if the websites pointing to a page were themselves also important, that would further increase the importance of the page they point to. This essentially creates a recursive definition of importance leading to a cascade of rankings throughout the network.

However, as argued in [55], the PageRank algorithm cannot be used directly to identify opinion leaders since it fails to take into account the novelty of the information provided by each node. The authors claim that if a node adds no new information to what is already known in the network, it is not a good opinion leader. To incorporate this factor into opinion leader detection, the authors propose the InfluenceRank model [55], which ranks nodes based on their information novelty as well as their importance in the network. They compare it against several other models, including PageRank and Information-Novely-based Ranking (which includes only the information novelty measure also used by InfluenceRank), and InfluenceRank scores higher than all other tested methods.

An additional extension of PageRank by the name of TwitterRank [4] adds topical similarity to the OLD process, specifically for analysing Twitter net-

work data. The authors confirm the presence of homophily [34] on Twitter [4]. Because of this, they argue that the influence of an opinion leader varies depending on the topic they discuss. If a large portion of their followers are not interested in a particular topic, their influence drops significantly. Conversely, if the majority of their followers show interest in a topic, the opinion leader's influence increases dramatically. Therefore, the authors propose a topic-sensitive TwitterRank model. The model uses a directed graph to capture the network, and employs a random walk to move around the network. This random walk is different for each topic, such that a set of topic-specific TwitterRank vectors is created, which measures a user's influence for each individual topic. These vectors can also be aggregated to form a measure of a user's overall influence. This aggregated measure of influence allows for easy comparison to other models. Because of TwitterRank's widespread influence, in addition to its specific application on Twitter data, I use this method in my thesis to find opinion leaders.

Yet another extension of PageRank comes in the form of TrustRank [15], which uses a signed network to consider both positive and negative links to a node in order to identify opinion leaders. Concretely, TrustRank is built on the idea that negative links signal distrust in the leader, and as such, lower their rank as an opinion leader. As such, a user's opinion leader score is based upon not just the size of the influence of their comments, but also on whether the influence of each comment is influential or detrimental. When comparing this method to similar models using negative links, TrustRank performs significantly better.

The final extension of PageRank that I will discuss here is the LeaderRank method [56]. Its main difference with PageRank is the introduction of a new node (the *ground node*) that is connected to every other node in the network by a bi-directional edge. As such, the graphs created by LeaderRank are strongly connected, since every node is separated from any other node by at most one node. This strong connectivity in turn means that LeaderRank is guaranteed to converge: networks that are not as well connected run the risk of a random surfer getting "stuck" in nodes that are not connected to any other node. As such, they often require an extra parameter representing a probability to jump to any random node in the network, just to get out of this problem. This parameter then needs to be tuned as it can have a major effect on how the model is trained, which removes some flexibility from the models [54, 56]. However, since LeaderRank's network is so strongly connected there is no need for such a parameter, which in turn adds to the flexibility of the algorithm. The added flexibility of LeaderRank compared to PageRank, alongside its strong performance in finding opinion leaders [56] make it an excellent choice for this task, which is why I will be using this method in my experiments as well.

Finally, the SNERank [13] model uses the Attributed Social Network Embedding (ASNE) graph embedding model which I discussed earlier in section 3.1 to calculate the difference between nodes in vector space. The embedding creates a vector representation that is based upon both the network structure and the text content; hence, these values are taken into consideration in vec-

tor space. This results in an opinion leader ranking based on the calculated weight/distance in the low-dimensional representation. This method overall outperforms TwitterRank on the tested data. Since I also use TwitterRank and the ASNE model in my thesis, I also use SNERank. This allows me to test the robustness of this model and see if it will outperform TwitterRank on other datasets as well.

3.2.2 Data mining and machine learning methods

The next major category of opinion leader detection methods is data mining and machine learning methods.

The first framework discussed here uses the k-means clustering algorithm to separate the nodes and identify candidate opinion leaders [57] on a forum for predicting stock prices. These candidate opinion leaders are then further refined based on the correlation between their sentiments and the actual stock price changes to determine their effectiveness. The candidates with the highest effectiveness are selected as the true opinion leaders. This model shows that sentiment analysis can help with identifying opinion leaders regarding a particular topic or sentiment in a social network.

The next framework relies on defining a trust measure between nodes to determine which nodes are opinion leaders [58]. The authors argue that there exists a positive correlation between the similarity among users and the strength of their trust, which is why this can be used for OLD. The model thus selects opinion leaders based on the similarity (determined by the overall number of positive comments received from other users) which is another measure of the centrality of a node in a network. Another machine learning-based model is the TCOL-Miner model [59]. It selects opinion leaders in several steps. Firstly, a graph is constructed from the data. Next, the model uses the H-clustering algorithm [60] to discover community structure in the network. Afterwards, the communities detected in the previous step are used to select opinion leaders through a second clustering step, using the k-means clustering algorithm. Finally, the candidate opinion leaders are ranked, and the highest-ranking candidates are selected as actual opinion leaders.

In the first step, a homogeneous, directed, weighted graph is constructed from the data (the data is from a forum for car enthusiasts), with users as the nodes. A directed edge exists from user a to user b if user a responded to an article written by user b. The weight of an edge is determined by a similarity measure, which in this case is defined by how many common neighbours the two nodes have, as well as the time of the day that the users are typically active.

Secondly, the H-clustering (or hierarchical clustering) algorithm [60] is used to detect communities. Each individual node is selected as a community. Then, a pair of nodes becomes grouped together in a community if the weight between these two nodes is higher than with any other neighbour. So for example, for two nodes a and b, if the edge between a and b is larger than any edge connecting to node a and it is larger than any edge connecting to node b, they become a community together. When two nodes are combined in such a way, a check is

performed to evaluate the quality of this new community. This is known as the modularity gain [60] and is calculated using the sum of the total similarity between nodes in a community. If this value is below a certain threshold, the iteration stops. Next, these communities are used to determine the candidate opinion leaders. A score is calculated for each node based on several factors, namely the total number of articles published, the probability that a user's articles will get replies from other users, the degree of expertise (total number of articles) a user has written in a specific domain, and the probability that a user will reply to articles written by other users. Then, the k-means algorithm is used on each community to group nodes with similar scores together. After sorting the clusters by score, the opinion leaders are selected from each high-scoring cluster until the number of opinion leaders has reached the specified number.

3.2.3 Evaluation methods

Evaluation methods for Opinion Leader Detection are not as clear-cut as they are for Graph Embedding. Many papers use different methods to evaluate their OLD models, and unfortunately, there seems to be no consensus on which evaluation method is best [61]. Despite this, there exist some methods that are still used regularly, which I will cover here.

Expert rating The first method involves using an expert to provide a ranking of opinion leaders in a network [62, 63]. This method simply involves letting an expert provide a subjective rating of comments made by users in the network as either having a strong or weak influence. These ratings can then be aggregated for each user to provide a rating of the users themselves. These experts are typically asked to use criteria such as social competence/popularity, knowledge and expertise and leadership qualities to inform their ratings [61].

Core Radio Core Radio [64] is a metric based on the intuition that opinion leaders generally interact with a large number of users at a high frequency. As such, this method scores a user based on the proportion of interactions in the network that involve that user. The more a user is involved in interactions, the more likely this user is to be an opinion leader [65].

Coverage, Diversity, and Distortion Another way of determining how an opinion leader affects the overall network is through measuring the *coverage* [55] of a user: this metric covers how many users are influenced by the comments of another user (where influence is measured through InfluenceRank as discussed above), with the intuition being that opinion leaders influence a larger number of people than non-opinion leaders.

Furthermore, as discussed by [55], opinion leaders that are able to cover their area of expertise from multiple perspectives are superior than those that cannot. This quality is captured by the *diversity* metric which measures the dissimilarity of the messages created by a user.

Finally, the authors argue that even though opinion leaders should cover their area of expertise from multiple perspectives, they should also be a driving force behind the way that this topic is discussed throughout the entire network. As such, the *distortion* metric measures the dissimilarity between the sub-topics that a user talks about and the sub-topics that are discussed in the entire network. For this metric, lower values of dissimilarity are one indicator for opinion leadership, since it indicates that the user discusses the same subjects as the overall network.

Combining the results from these three metrics together gives a nuanced perspective of opinion leaders, namely a user who influences a large number of other users, covers a topic from numerous perspectives and discusses the same subjects within a topic as the rest of the network.

SIR The SIR model [66] was originally designed as a way of modelling the way diseases spread throughout a population. However, this method has also been used for modelling the spread of information throughout a network [67], which is why it can also be applied to the task of assessing opinion leaders. The main idea behind this application is that opinion leaders cause information to spread throughout a network much faster and better than non-opinion leaders. As such, regular users might only cause the information to slowly spread throughout part of the network, whereas opinion leaders might cause the information to spread rapidly throughout the entire network.

The SIR model works by creating a graph where each node has one of three statuses in keeping with the origin as a model of spreading disease: susceptible, infected, or recovered [13]. Initially, a certain number of nodes are infected, whereas the remainder of the nodes are susceptible. Then, every iteration neighbouring nodes to infected nodes have a change of becoming infected as well, and infected nodes have a change of recovering from their infection.

To apply this to the task of assessing an opinion leader method, we can set the opinion leaders (as found by that method) as the initially infected nodes. Furthermore, we can set the probability of an infection spreading to be based on the influence of the opinion leader nodes. Then, we can let the model run and observe how fast the information spreads throughout the network by counting the number of influenced people (# infected + # recovered). The intuition dictates that good opinion leaders spread the information throughout the network quickly and cover the entire network, rather than only a part of it [67].

As such, this task can be used to effectively compare different opinion leader methods: if the SIR spreads much more quickly and further for one model than for another, this indicates that the first model has identified better opinion leaders than the second [3].

Since this is a method that allows for comparing different models to each other without needing expert ratings, I will be using this in my thesis to evaluate the quality of the chosen opinion leader detection methods.

Kendall's τ The Kendall τ correlation method is used to determine how similar two ranking lists are to each other [68]. The metric can be used to compare two ranking lists created by opinion leader detection methods with each other to see if the opinion leaders generated by both methods are similar to each other [4].

If the Kendall's τ correlation is positive (up to 1), this indicates that there is a high degree of agreement between the two ranking lists. On the other hand, if the correlation is negative (down to -1), this signals that there is a high degree of disagreement between the two ranking lists. For example, users that are ranked highly by one list are ranked badly by the other list, and vice versa.

I will use this evaluation method in my thesis as well. This method, alongside SIR, can give me a more complete picture of the similarities and differences between opinion leader detection models.

4 Data

4.1 Data introduction

The data that I used in this experiment is a large collection of messages on Twitter (also known as *Tweets*) in Dutch [16], all written between February 2020 and November 2020. The Tweets all concern the COVID-19 pandemic: the data was filtered based on several COVID-19-related keywords such as "corona", "covid", "huisarts" (general practitioner) and "mondkapje" (face mask).

Each Tweet is represented by a Tweet object as defined by the Twitter API[2]. As such, it contains attributes like the time of creation, the raw text contained in the Tweet, its accompanying URL and the user who posted it.

Furthermore, preliminary data analysis performed in [16] assigned a topic and a sentiment to each Tweet. These provide extra information that can be added to the input of the embedding model. The sentiment can be either positive or negative, and the topics include "face masks", "social distancing", and "vaccinations".

4.2 Data collection

The data was collected through twiqs.nl. This website is a service provided by Surf, and the Netherlands eScience Center. It collects Dutch Tweets and makes them publicly available to the research community, in addition to providing analyses done on these data. The earliest Tweets in the collection were written in February 2020 as this was the month in which the first COVID-19 patient was diagnosed in the Netherlands. The latest publication date included in the collection of Tweets is November 2020. Using the Twitter API it was possible to discern the language in which a Tweet is written by courtesy of the *lang* feature. Tests show that more than 80% of the Tweets written in Dutch during

[2]https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/Tweet

that period were contained in the collection [16]. An outline of the number of collected Tweets per month is outlined in Table 1.

Month	**Number of Tweets**	**Per day**	**Per hour**
February 2020	14,852,678	512,126	21,338
March 2020	21,180,942	683,256	28,507
April 2020	18,715,900	623,863	25,994
May 2020	18,044,679	582,086	24,253
June 2020	20,807,966	693,598	28,899
July 2020	19,154,442	617,885	25,745
August 2020	20,314,042	655,291	27,303
September 2020	20,340,753	678,025	28,251
October 2020	**21,987,100**	**709,261**	**29,513**
November 2020	19,370,135	654,671	26,902

Table 1: Total number of Dutch Tweets collected. Data in bold was used in this thesis for further filtering and processing.

The collection of Tweets was then filtered, to extract only the Tweets concerning COVID-19 [16]. This was done by filtering on a list of keywords which were selected because of their relation to COVID-19. An overview of the keywords can be found in Table 2.

Category	**Keyword**	**English Translation**
Disease	corona	
	covid	
Health care	huisarts	doctor
	mondkapje	face mask
Government	rivm	national health organization
Social	flattenthecurve	
	blijfthuis	stay home
	houvol	hang in there

Table 2: Keywords used for filtering Tweets related to COVID-19

The filtering was agnostic to the case of the letters and also considered words which had one of the keywords as a substring, like the word *coronavirus* which contains the keyword *corona*.

As mentioned above, a topic was assigned to each Tweet, and due to the time limit associated with the thesis as well as the limited availability of computing resources, I only used the data from the topic "social distancing" from the month of October 2020. This subsection of the data contains a total of 56,173 Tweets written by 24,588 unique users.

To construct the social network, information about both the nodes and edges is needed. Typically, social networks like these define their graphs using users as nodes and interactions (following/friendship relations) as edges [10]. However,

the data described in Table 1 only contained user ids, but no information about the following relationships.

Because of this, I manually collected the following relationships between the users through the Twitter API. By collecting the followers for each user and filtering out the followers that did not appear in the data described in Table 1, only the followers for each user in the dataset were left. From this, I extracted the friends for each user by making use of a simple observation: if user i follows user j, then that means that user j is a friend of user i.

Nonetheless, this method might not be entirely accurate: I extracted the followers as they were at that point in time (around September 2021) whereas the tweets were collected much earlier than that (see table 1). It is impossible to guarantee that the following relationships between the users were already established when the data was collected, as Twitter does not collect that kind of historical data. However, when considering the alternatives, and finding that sufficiently large networks on Twitter tend to be relatively stable in the following structure [69], I decided that this would still be the best course of action.

4.3 Data description

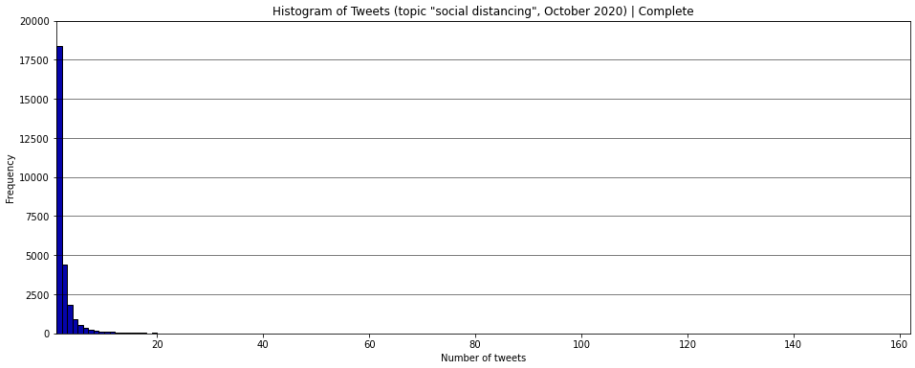
As I discussed above, the data that I used for my experiments contains a total of 56,173 Tweets written by 24,588 unique users. Figure 3 shows two histograms of these Tweets. The maximum number of Tweets written by a single user is 162, but as can be seen in Figure 3a, the vast majority of users only have a few Tweets. In fact, 66.61% of all users only have published one Tweet, and 98.25% of all users have 10 Tweets or less.

The data set contains several important attributes for each Tweet, which served various purposes throughout the experiments. Table 3 contains a list of the relevant attributes, alongside a description and a short explanation of what I used them for. Essentially, the Tweets themselves and the stance probabilities were used to construct the semantic information for each user, while the remainder of the attributes were used for structural information to create the network graph.

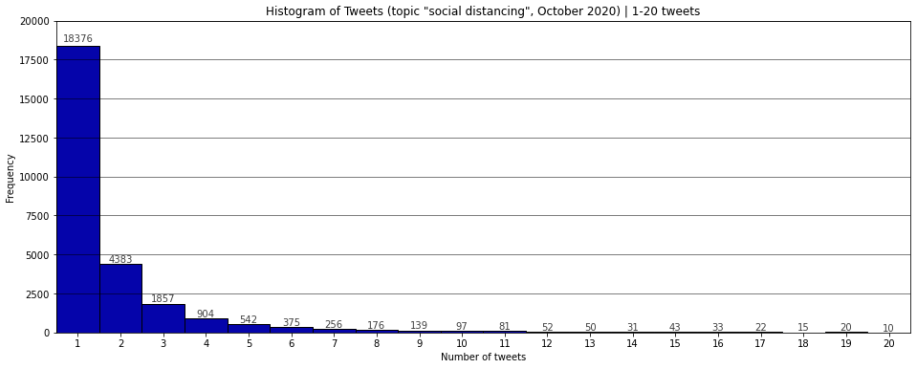
To be able to track changes in the network over time, I split the data by week, creating 5 weeks worth of data. However, some computational issues arose with weeks 2 and 3 of the data due to their size, which is why I split those two weeks up into two parts each. Some descriptive statistics regarding these slices of the dataset can be found in Table 4.

4.4 Baseline dataset

To verify the results of all models, I used another dataset as baseline. Running the experiments on this data as well provides additional corroboration for the results found on the *social distancing* dataset.



(a) Complete histogram



(b) Partial histogram showing distribution of users with 1-20 Tweets

Figure 3: Histogram showing the distribution of Tweets created per user. (a) shows the complete histogram (the maximum number of Tweets written by a single user is 162), whereas (b) shows a more detailed overview of the distribution of users with 1-20 Tweets.

4.4.1 Facebook dataset

The baseline dataset that I used is the Facebook dataset[3] that is also used to test node2vec [33]. This Facebook data originates from the Stanford Network Analysis Project (SNAP) [70, 71].

It contains anonymized friend lists from Facebook in the form of user profiles which represent the nodes, and friendship relations between pairs of users which represent the edges. This information is sufficient to construct a graph which I can use to conduct the experiments and form a baseline: the graph contains 4.039 nodes and 88.234 edges.

It is, however, important to note that the Facebook dataset does not contain any attribute information like the Tweet texts or stance. Thus, it is less rich

[3]https://snap.stanford.edu/data/ego-Facebook.html

Attribute name	Description	Usage
id_str	Unique Tweet identifier	Used to extract more information about the Tweet from the Twitter API
created_at	Tweet creation date and time	Used for filtering the Tweets by date
user_mentions	Represents other Twitter users mentioned in the Tweet	Used for determining how often users interact with each other
full_text	Complete Tweet text, including mentions, hashtags and URLs	Used to determine a user's total text attribute
user	User information such as user id, screen name, and followers count of the Tweet's creator	Used to construct the network and find user interactions
prob_soc_dist	A list containing three probabilities for the Tweet's stance: supports, rejects, irrelevant	Used to determine a user's overall stance regarding the topic of social distancing
followers_list	A list of all users that follow the Tweet's creator	Used to construct the network graph
following_list	A list of all users that the Tweet's creator follows	Used to construct the network graph

Table 3: Explanation of Tweet attributes contained in the data set

Week	**1**	**2 (1)**	**2 (2)**	**3 (1)**	**3 (2)**	**4**	**5**
# Tweets	9.604	5.473	5.449	7.448	7.318	9.091	7.311
# nodes	5.980	3.999	4.192	5.880	5.607	5.707	4.606
# edges	288.571	180.386	183.225	262.117	249.168	281.078	215.321

Table 4: Information regarding the sizes of each week of data. Weeks 2 and 3 were split up into two parts because of their sizes, in order to solve computational issues.

with information compared to the *social distancing* dataset. Despite this, using the Facebook dataset will allow me to compare the results of my models with the results of reported in the original node2vec paper [33].

5 Methodology

5.1 Network definition

5.1.1 Twitter graph

The main network that forms the input for the embedding model is a homogeneous, directed, unweighted graph $G = (\mathcal{U}, \mathcal{E}, \mathcal{A})$ where $\mathcal{U}$ is the node set containing all Twitter users (or *Twitterers*), $\mathcal{E}$ are the links between the Twitterers, and $\mathcal{A}$ are the attributes of the Twitterers. A *Twitterer* is defined here as a collection of all Tweets made by a single user, in addition to all Tweets in

which that user is mentioned. E is the edge set. A directed edge exists from Twitterer i to Twitterer j if i (the *follower*) follows j (the *friend*). Isolated nodes (nodes that are not connected to any other edges in the graph) have been removed from the graph. As suggested in [13], the frequency with which Twitterer i retweets Twitterer j's Tweets could serve as a measure of the quality of their relationship since the more they retweet each other, the more interests they share. Therefore, the weight of an edge can be defined as:

$$(1 + f_{ij})/(N_i + f_i) \tag{1}$$

In (1), N_i is the number of friends user i has, f_{ij} is the number of times user i retweeted the Tweets of j, and f_j is j's total number of retweets.

However, due to the limited number of Tweets available for each user (as shown in Figure 3), the differences between the weights calculated in equation 1 for all users are negligible: all weights still end up extremely close to 1. As such, including the weight will not have any major effect on the quality of the embedding nor the quality of the selected opinion leaders. Therefore, I chose to exclude the weight for the experiments using this dataset, which results in the graph being a homogeneous, directed unweighted graph.

With this graph, there are generally two ways in which opinion leaders can be determined. The first involves finding the topic-specific opinion leaders using the topic-specific graphs. The second involves finding the overall opinion leaders using the main graph containing all topics. The latter method might prove to be useful in comparing the effectiveness of this model against other models that do not have the ability to find topic-specific opinion leaders. Another way in which this might be done is to aggregate the topic-specific opinion leaders together to find the overall opinion leaders. However, since the dataset contains only a single topic, for the purposes of my analysis the topic-specific opinion leaders and the overall opinion leaders are considered to be identical. Thus, I will not make any additional distinction between these two types of opinion leaders in the remainder of this thesis. It will be relevant when discussing the TwitterRank[4] OLD algorithm later, which is why I clarify this distinction here.

5.1.2 Facebook graph

The network that I use for the baseline experiments differs slightly from the network described above. It is a homogeneous, undirected, unweighted graph $G = (\mathcal{U}, \mathcal{E})$ where $\mathcal{U}$ is the node set containing all Facebook users, and $\mathcal{E}$ are the undirected edges (or links) between the nodes. Such an edge exists between two users if there exists a friendship relation between those users on Facebook. As opposed to the *social distancing* Twitter network, the edges in the Facebook network are undirected since, contrary to Twitter, users cannot create a connection with other users without needing consent from the other party. Instead, both users need to give their consent before a connection (in the form of a friendship relation) is established. Hence, the graph created from this data is also undirected. Furthermore, the Facebook data contains no information

that I could use to estimate an edge weight, which is why the graph is also unweighted, like the Twitter graph.

5.2 Graph Embedding method

The graph embedding method that I used is the Attributed Social Network Embedding (ASNE) model [10]. The ASNE model is the only embedding model incorporating more than merely the structural information. Therefore, since both the network structure as well as attribute information (text content, text topic, overall sentiment, etc.) play a pivotal role in this social network, the ASNE model is a prime candidate for creating the required embedding. The extra attribute information should result in higher-quality embeddings than only using structural information. ASNE uses a homogeneous, unweighted, directed graph. The network is defined as $G = (\mathcal{U}, \mathcal{E}, \mathcal{A})$ where $\mathcal{U}$ are the users, $\mathcal{E}$ are the links between users, and $\mathcal{A}$ are the attributes of the users. Each edge e_{ij} can be associated with a weight s_{ij} indicating the connection strength between u_i and u_j.

The goal of this embedding method is to preserve both structural proximity as well as attribute proximity. Structural proximity is denoted by links between users. If there is a link between two nodes then there is a direct proximity between them. Conversely, if a node is within the *context* of another node, there is indirect proximity between them. A context is generated for a node by performing a random walk over the network [31, 10]. Just like the random walk used by the node2vec [33] method, the ASNE model employs the use of a biased random walk strategy balancing between breadth-first search and depth-first search.

Based on this definition of structural proximity, the key lies in estimating the pairwise proximity of nodes. The conditional probability of node u_j on u_i is a softmax function which measures the likelihood that node u_j is connected to node u_i:

$$p(u_j|u_i) = \frac{\exp(f(u_i, u_j))}{\sum_{j'=1}^{M} \exp(f(u_i, u_{j'}))} \tag{2}$$

To account for a node's structural proximity with regards to all its neighbours, a conditional probability of a node set is defined by assuming conditional independence:

$$p(\mathcal{N}_i|u_i) = \prod_{j \in \mathcal{N}_i} p(u_j|u_i) \tag{3}$$

By maximizing this conditional probability, the model preserves global structural proximity. Concretely, the global structure is modelled by a likelihood function using this conditional probability of a node set:

$$l = \prod_{i=1}^{M} p(\mathcal{N}_i|u_i) = \prod_{i=1}^{M} \prod_{j \in \mathcal{N}_i} p(u_j|u_i) \tag{4}$$

This likelihood function l can be used to create the actual embedding model. The authors use a neural network with multiple hidden layers to better capture the non-linearities of real-world networks, as shown in Figure 4. Specifically, the pairwise proximity of nodes i and j is calculated by taking the embedding vector of i and passing it through all hidden layers with their weights, biases, and activation functions. Finally, that result is multiplied by the weight vector of j to get the pairwise proximity.

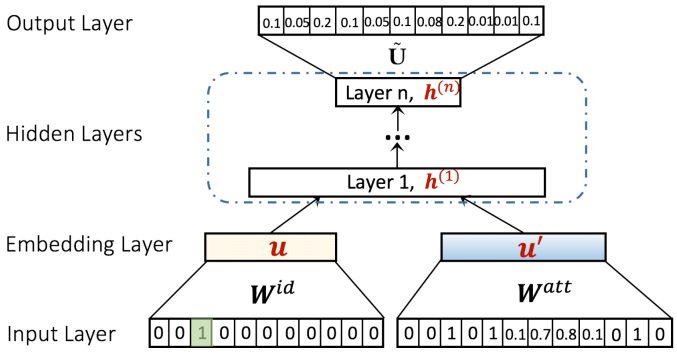


Figure 4: Framework of the Attributed Social Network Embedding (ASNE) model. Image taken from [10].

Attribute proximity is denoted by the proximity of nodes as evidenced by their attributes. The attribute intersection of two nodes indicates their attribute proximity. Firstly, all attributes are converted to a generic feature representation: discrete attributes are converted to a set of binary features via one-hot encoding. Conversely, continuous attributes are transformed into a real-valued vector. The vector representations of all attributes are then concatenated to form the generic feature vector. This vector is then used as input for the same neural network as was used for the structural proximity. The result of this is the attribute proximity.

So, to conclude, an overview of the ASNE model. The input layer (as can be seen in Figure 4) consists of two parts: the structural fragment (which is a one-hot encoded vector where only the node being considered has the value 1 and all other nodes have the value 0) and the attribute fragment (which contains the generic feature vector as discussed earlier). Next, the embedding layer takes the input and converts it into two dense vectors as described above: the one-hot vector gets converted into a dense vector which captures structural information, and the feature vector gets converted into a dense vector which captures attribute information. A vector is dense if it consists primarily of non-zero values, as opposed to a sparse vector which in turn consists mainly of zero values [72].

These two dense vectors are then fed into the hidden layers, consisting of a multi-layer perceptron. The hidden layers multiply the vectors by the layer's weight, add a bias, and pass it through an activation function, before passing

it to the next layer. The activation function used in this paper is the soft-sign function. The soft-sign function is similar to the tanh function, but where tanh converges exponentially, soft-sign converges polynomially [73]. Each successive hidden layer contains only half as many neurons as the last. The authors argue that this is done in order to enable the model to better learn the abstract features of the data.

Finally, the output vector of the last hidden layer is transformed into a probability vector, which contains the predictive link probability of a node to all nodes in the network. This transformation is done by multiplying the output of the hidden layer by the weight of the corresponding node. This is then fed into the softmax function to obtain the probability vector. This vector contains the predictive link probability between any pair of nodes in the network.

5.3 Opinion Leader Detection method

The OLD method I have used to complement the embedding created by the ASNE model is the SNERank [13] method. This OLD method was specifically designed to supplement the ASNE model. Furthermore, it has been shown to outperform the other main candidate TwitterRank [4]. The SNERank method uses the node embeddings created by the graph embedding method to calculate a score for each node in the graph. These node scores can then simply be sorted to find the most influential users (highest scoring nodes) and the least influential users (lowest scoring nodes). Specifically, the algorithm calculates the extent to which a particular node can influence other nodes in the network, by calculating the similarity between the embeddings of that node and any other node. I calculate the similarity of two nodes i and j as follows:

$$W_{i,j} = \begin{cases} \exp(u_j \cdot u_i), & \text{if } j \text{ is } i\text{'s friend} \\ 0 \text{ , otherwise} \end{cases} \tag{5}$$

where u_i is the embedding of user i and u_j is the embedding is user j [13]. The embeddings here are the concatenated embeddings, rather than the individual embeddings as this has been shown to increase performance for downstream applications [10]. The final score for all nodes can then be calculated with:

$$AR_i^{t+1} = (1-d)\frac{sum_{i\in|M|}(W_{i,j})}{sum(W)} + d\sum_{i\in|M|}\frac{W_{i,j}}{\sum_{j'\in Neighbour(i)} W_{i,j'}} AR_i^t \tag{6}$$

where M represents the nodes of the network, AR_i^t is the ranking value of user i in iteration t and d is the dumping factor in the interval [0, 1]. The original paper used $d = 0.85$ [13], and I opted to stick with this same value as well.

Since the ASNE embedding algorithm creates embeddings based on both structural and attribute information, and SNERank evaluates nodes based on their embeddings, the score calculated by SNERank (and by extension their final rank) is also based upon both structural and attribute information. The nodes

that are closest together in the embedding vector space have similar structural and attribute information and are thus more likely to influence each other [13]. Thus, the highest-scoring nodes as selected by SNERank also have the highest probability of influencing the highest number of other users in the network on the corresponding topic of the data. For this reason these nodes can be considered opinion leaders.

5.4 Baseline algorithms

In order to properly gauge the effectiveness of the described method, I used several other algorithms to ensure ample opportunity to compare the results. Specifically, I used the DeepWalk [31] and node2vec [33] Graph Embedding models, as well as the TwitterRank [4] and LeaderRank [56] Opinion Leader Detection models.

DeepWalk The DeepWalk [31] model takes as its input a graph and produces node embeddings in the form of vector representations of the nodes. The method is primarily based off of language modeling, and the authors adapt those methods to create graph embeddings after observing that the distribution of nodes in random walks is extremely similar to the distribution of words in natural language.

This observation indicates that methods capable of creating embeddings for natural language might also be capable of generating embeddings for graphs as well. Specifically, the authors employ the use of the Skip-Gram language model in the DeepWalk architecture. Skip-Gram is a model that aims to find the most-related words for a given word. Concretely, given a certain word, Skip-Gram attempts to predict the context words that best fits that target word. It is primarily based on the Distributional hypothesis, which posits that when words appear in similar contexts, their meanings generally are found to be very similar as well [74].

This can then be adapted from natural language to graphs by treating nodes visited by random walks as words in a sentence: nodes that appear in similar contexts (surrounding nodes) are likely to be similar to each other as well. Thus, these nodes should also be placed close together in the embedding vector space created by the embedding algorithm. Following this line of reasoning, the optimization function for DeepWalk is as follows:

$$\min \phi \ -\log \mathrm{P}((v_{i-w}, ..., v_{i+w}) | \ \phi(v_i)) \tag{7}$$

where ϕ is the mapping function between the nodes and their vector representations, v_i is node i, w is the size of the context of the random walk, and $\phi(v_i)$ is the vector representation of node v_i.

Essentially, (7) creates vector embeddings from the nodes in the graph by using their context to get information about the node itself. As such, the embeddings capture the neighbourhood structure of the nodes, ensuring that the

structural properties of the original graph are maintained. When incorporating the Skip-Gram model into this, it removes the ordering constraint, thereby changing (7) to:

$$\mathrm{P}((v_{i-w},...,v_{i+w})|\,\phi(v_i)) = \prod_{\substack{j=i-w \\ j\neq i}}^{i+w} \mathrm{P}(v_j|\phi(v_i)) \tag{8}$$

where $\mathrm{P}(v_j|\phi(v_i))$ is approximated by a hierarchical softmax function. The reason that this probability needs to be approximated rather than calculated directly is simple: to calculate this probability precisely would require a massive amount of time and computational resources, thereby decimating the scalability of the algorithm altogether. As such, the authors opt to use a hierarchical softmax to approximate the probability distribution rather than calculating it directly, which in turn introduces additional error into the model, but speeds up the computation enormously.

The hierarchical softmax works as follows: first, it creates a binary tree where every leaf is a node of the graph. Thus, to find $\mathrm{P}(v_j|\phi(v_i))$, we can maximize the probability that a certain path (in this case: the path from the root to node v_i) occurs in the tree. This particular path in the tree can be represented by a series of points in the tree: we can travel through all these points to arrive at node v_i. Therefore, the objective function here aims to maximize the probability that we find this path specifically:

$$\mathrm{P}(v_j|\phi(v_i)) = \prod_{t=1}^{\log|V|} \mathrm{P}(b_t|\phi(v_i)) \tag{9}$$

In (9), $|V|$ is the number of nodes in the original graph, and $\mathrm{P}(b_t|\phi(v_i))$ is a binary classifier which decides whether to go left or right at every branch in the binary tree.

The end result of this entire process involving the hierarchical softmax is a tremendous improvement in computation speed: since a balanced binary tree like the one used here has a depth of $log(|V|)$ [75], at worst we need to evalauate $log(|V|)$ nodes to determine the final probability of a node. Therefore, instead of $\mathcal{O}(|V|^2)$, the complexity of the Skip-Gram model becomes $\mathcal{O}(|V|log(|V|))$.

When the probabilities have been calculated according to (9), the vector representations of node v_i are updated according to (7).

After training the model, the embedding vectors of all nodes reflect the structural context of the original graph, so nodes that are close to each other in the graph also have similar embedding vectors, which can then be used for downstream applications such as opinion leader detection.

I used the implementation of DeepWalk provided by the authors of the original paper [31], which can be found on GitHub[4]. I was able to use DeepWalk for any dataset without needing to change any fundamental aspects of the code.

[4]https://github.com/phanein/deepwalk

node2vec The node2vec model takes as its input a homogeneous graph, weighted or unweighted, directed or undirected, and returns a mapping of the nodes of the graph to a low-dimensional vector space as its output, very similar to DeepWalk. The objective function that node2vec aims to optimize is:

$$\max_{f} \sum_{u \in V} \log Pr(N_S(u)|f(u)) \tag{10}$$

This objective function serves to maximize the log-probability of observing a neighbourhood $N_S(u)$ in the network for a node u conditioned on its feature representation f.

To accomplish the goal of maximizing this objective function, the authors propose the use of a biased random walk strategy to sample the neighbourhood nodes. The random walk combines both the breadth-first search (BFS) and depth-first search (DFS) search strategies. The reason for this is that this combination allows the model to learn both the local and the global network structure. the BFS is restricted to only the nearby nodes to get a view of the neighbourhood of each node. Furthermore, when using BFS, the nodes in a neighbourhood are typically sampled numerous times, leading to a reduction in variance of the distribution of the source node's neighbourhood. Conversely, DFS is capable of exploring nodes further away from the source node: this enables it to learn the more global network structure, thus gaining a more high-level view of the communities present within the network.

As mentioned earlier, the random walk is biased based on the edge weights and two parameters p and q which guide the walking process. The parameter p denotes the *return parameter*, controlling the likelihood of instantly revisiting a node during the random walk. If it is high then the random walk is encouraged to do more exploration, whereas if it is low the random walk is more likely to stay close to the starting node. The parameter q designates the *in-out parameter*, which allows the random walk to distinguish between nodes that are "inward" or nodes that are "outward". With large values of $q(> 1)$, the model will obtain a local view of the network, thereby simulating a BFS, whereas with small values of $q(< 1)$, the random walk is more likely to explore, thereby mimicking DFS.

By applying the stochastic gradient descent algorithm on the random walks, we gain the optimized result of the objective function (10), thereby learning the optimized vector representation. The random walks used by node2vec are computationally efficient concerning both space and time when compared to using traditional BFS or DFS approaches. This, combined with its flexible balance of learning local and global representations of the network make node2vec it an exquisite option to use when creating embeddings.

I used the implementation of node2vec provided by the authors of the original paper [33], which can be found on the SNAP website[5]. I was able to use node2vec for any dataset without needing to make any fundamental changes to the code.

[5] http://snap.stanford.edu/node2vec/

TwitterRank The TwitterRank method uses a directed cyclical unweighted graph where each node corresponds to a Twitter user, and an edge exists between two users if one follows the other, with the direction going from the follower to user they follow (also known as the *friend*). A topic-specific TwitterRank is then assigned to each user for each topic that is present in the data. Topics are identified by a Latent Dirichlet Allocation model.

LDA is an unsupervised machine learning model with the goal of determining topic information from text data. It uses a bag-of-words representation, which entails that the documents (in this case a document is a collection of all Tweets by a single user) are managed as vectors containing the counts of each word. Using this, each document is represented by a probability distribution over several topics, whereas a topic is represented by a probability distribution over several words. The LDA model then learns these two distributions from the data. These distributions confer information regarding which topics people usually write about, in addition to a representation of which topics are contained in each document. The end result from applying the LDA model is contained in three matrices: a matrix containing the number of times a word in each user's Tweets has been applied to each topic, a matrix containing the number of times each word has been assigned to each topic, and a vector containing all unique words.

In order to assign a topic-specific TwitterRank to each user for each topic, the model uses a topic-specific random walk on the graph to create a topic-specific relationship network among twitterers, which is modelled as a transition probability from user i to user j:

$$P_t(i,j) = \frac{|\mathcal{T}_||}{\sum_{a:\, s_i\ follows\ s_a} |\mathcal{T}_\dashv|} * sim_t(i,j) \tag{11}$$

where $sim_t(i,j)$ is defined as:

$$sim_t(i,j) = 1 - |DT'_{it} - DT'_{jt}| \tag{12}$$

The transition probability in (11) from user i to user j for a topic t consists of two parts: firstly, the fraction of Tweets that a user i sees that have been posted by user j. Essentially, this encapsulates the idea that the more Tweets user i sees from user j, the larger the influence of user j on user i. The second part of the transition function consists of the similarity of the interest users i and j for topic t. The more similar the interests of the two users are, the higher the transition probability from i to j. This transition probability models the idea that friends on Twitter who interact with each other frequently and both have a similar level of interest in a particular topic are more likely to influence each other on that specific topic.

There exists the possibility that some users would share a following relationship in a loop, without sharing this relationship with users not in the loop. Therefore, the model also uses a teleportation vector to capture the probability of jumping to a pseudo-random other user (node) that is not necessarily con-

nected by an edge. This teleportation vector is added to prevent the random walk from getting "stuck" in the same group of nodes every iteration.

Finally, the TwitterRank of a user gets calculated by iteratively updating the transition probability matrix and the teleportation vector:

$$\overrightarrow{TR_t} = \gamma P_t * \overrightarrow{TR_t} + (1-\gamma)E_t \tag{13}$$

where $\overrightarrow{TR_t}$ is a vector of TwitterRank scores for all users in the network for topic t, γ is the weight parameter, P_t is the matrix of transition probabilities as defined in (11) for topic t, and E_t is the teleportation vector for topic t.

The weight parameter γ controls the probability of teleportation. Lower values of γ correspond to a higher probability of teleporting to a random user in the network, thus promoting a higher exploration rate of the random walk over the graph. When (13) is applied to all topics in the data, the result is a vector for each user containing their influence score for each topic. This vector can also be aggregated by summing over all topics multiplied by each topic's weight. A topic's weight can be determined in several ways, but the simplest way is to simply calculate the weight for each topic by how often it is present in the data. The aggregated value of all topic-specific TwitterRank scores corresponds to a Twitter user's general influence across all topics in the data.

LeaderRank LeaderRank uses a graph where each node corresponds to a user, and a directed edge corresponds to a relationship between two users, with the direction of the edge corresponding to the following relationship. In case the following relationship is mutual, the edge is bidirectional as well.

LeaderRank operates by assigning each node the value 1. Then, in each iteration, a node's value is evenly divided among its neighbours, so if a node with value 1 has 4 neighbours, each neighbour receives 0.25. This iterative process continues until a steady state is reached (until the values in all nodes no longer changes).

Before starting the iteration process, LeaderRank adds a *ground node* to the graph which is connected to every other node in graph by a bidirectional edge. This step makes the graph very strongly connected, which ensures that the iteration process converges without needing to define hyperparameters to control the walking process.

This process corresponds to a random walk on the graph:

$$p_{ij} = \frac{a_{ij}}{k_i^{out}} \tag{14}$$

where p_{ij} denotes the probability that the random walk at i moves to node j in the next iteration, a_{ij} has the value 1 if there exists a directed edge from i to j and 0 otherwise, and k_i^{out} is the number of outgoing links of i (the number of users that i follows.

A node's value can then be determined by:

$$s_i(t+1) = \sum_{j=1}^{N+1} \frac{a_{ji}}{{k_j}^{out}} s_j(t) \tag{15}$$

where $s_i(t)$ is the value of node i at time t and N is the number of nodes in the graph. As explained above, the initial scores $s_i(0) = 1$ for all nodes i (*except* for the ground node) and $s_g(0) = 0$ for the ground node.

After convergence, the values of each node correspond to the scores assigned to them by the LeaderRank model. The higher a node's score, the more that node is considered to be an opinion leader. As such, the node with the highest value is considered to be the best opinion leader by LeaderRank.

5.5 Evaluation methods

Link prediction To evaluate the graph embedding methods, I use the link prediction task. The reason for this is that link prediction is better suited for my data than node classification, as the latter requires the nodes to have labels, which is not the case in my data.

An added benefit is that ASNE, DeepWalk, and node2vec have all been evaluated using link prediction in the past, which gives me a great opportunity to compare the results and hopefully draw more meaningful conclusions.

The graph embeddings are fed to the link prediction method alongside the test set (how the test set is created is explained in section 6.2.1).

Then, for each node pair in the test set, the model calculates the cosine similarity between the embeddings of the two nodes:

$$sim(A, B) = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{16}$$

where A is the embedding of node 1, B is the embedding of node 2, and n is the number of dimensions of the embedding.

This results in a cosine similarity value between -1 and 1, where -1 is perfect dissimilarity between the two embeddings and 1 is perfect similarity. These values are then used together with the test set labels to calculate the AUROC.

SIR To evaluate the opinion leader detection methods, I use the SIR model. This method attempts to model the spreading of information throughout a network, starting from several key nodes. Each node in the network can have three states, denoted by "Susceptible", "Infected", and "Recovered". Initially, only the key nodes will be Infected, whereas the other nodes are all Susceptible. Every iteration, Susceptible nodes that are connected to Infected nodes have a chance to become Infected themselves. In my implementation, similar to [13], this chance is defined as a probability λ_1 if the text similarity between the target node and one of the initial nodes is greater than a threshold ϵ. Otherwise, the probability is λ_2, with $\lambda_1 >= \lambda_2$. Every iteration, Infected nodes also have a chance to become Recovered, with probability $\frac{1}{k_{in}}$, where $\frac{1}{k_{in}}$ is the average

in-degree of all users in the network. To run the SIR model on an opinion leader detection model, I take the top 20 opinion leaders as defined by the OLD model, and set those as the initial infected nodes. Then, I let the SIR model run until convergence. The total number of Infected + Recovered users are the users that have been influenced the initial nodes. As such, the higher that number is, the better the initial nodes are at spreading influence throughout the network. The speed at which the SIR model reaches convergence is also important, as it indicates that the opinion leaders are better/worse at spreading their information throughout a network at a high speed.

Kendall's τ Kendall's τ is a correlation measure for ordinal lists, which is why it can be used to evaluate the rankings created by the opinion leader detection methods [4]. It provides an indication of the strength of the monotonic relationship between two ordinal lists, between -1 and 1. Intuitively, a τ of 1 corresponds to high values in list a being associated with high values in list b, whereas a value of -1 corresponds to high values in list a being associated with low values in list b [68].

The correlation measure τ between two lists a and b is defined as:

$$\tau = \frac{P - Q}{\sqrt{(P + Q + T) * (P + Q + U)}} \tag{17}$$

where P is the number of concordant pairs, Q is the number of discordant pairs, T is the number of ties only in a, U is the number of ties only in b. Following these definitions, if a tie occurs for the same pair in both a and b, it will not be added to either T or U.

A pair of observations X_i, Y_i and X_j, Y_j is concordant if $X_i > X_j$ and $Y_i > Y_j$ or $X_i < X_j$ and $Y_i < Y_j$. Conversely, such a pair is discordant if $X_i > X_j$ and $Y_i < Y_j$ or $X_i < X_j$ and $Y_i > Y_j$.

6 Experimental setup

6.1 Research questions

I performed several experiments using both the social distancing Twitter data (section 4.3) and the Facebook data (section 4.4) to provide an answer to the research questions. In this section, I will reiterate the four research questions from section 2.3, alongside an explanation of how I plan to answer these.

What is the influence of including both semantic and structural information in a graph embedding on opinion leader detection, compared to using only structural information? To test the effect of both semantic and structural information on opinion leader detection, I compared the ASNE model [10] combined with SNERank [13] against the node2vec model [33] and the DeepWalk model [31], both combined with SNERank as well. Since ASNE uses both semantic and structural information and the node2vec and DeepWalk

models only considers the network structure when creating the embedding, this comparison should yield a clear result as to the effectiveness of incorporating semantic information in a graph embedding. Furthermore, to eliminate differences in the creation of the structural embeddings by all three models, I also tested the ASNE model without including attribute information, so that the embeddings can only contain structural information. This will provide a baseline to compare the node2vec and DeepWalk models with as well.

What is the influence of using graph embeddings on opinion leader detection? To determine the influence of graph embeddings, ASNE, DeepWalk, and node2vec in combination with SNERank will be compared to the TwitterRank [4] and LeaderRank [56] methods. The TwitterRank and LeaderRank models uses a graph without creating an embedding for it first. Therefore, comparing both of these different classes of models will provide a clear indication of the effects of using a graph embedding.

What is the influence of a user's stance (positive or negative) on detecting opinion leaders? One of the features that could have a major effect on the process of finding opinion leaders is the stance of a user. Using the ASNE model I can include this feature in the embedding, which means it will in turn also be included in the OLD process. As such, to find the effect of a user's stance on detecting opinion leaders, I have created embeddings both with and without this feature.

How do opinion leaders develop over time in a social network? The social distancing Twitter data spans one month, as described in section 4. I have split these data up into segments spanning one week each. Then, I trained several opinion leader detection models on each week of the data and found a set of opinion leaders for each model. This allows me to see how the opinion leaders develop and change over the course of time.

6.2 Procedure

The experiments consisted of three phases: preprocessing the data, training the models, and evaluating the models. Figure 5 shows a flowchart outlining these steps. In this section I will cover each of these steps in more detail.

6.2.1 Data preprocessing

Structural and attribute information To ensure that the data was in the proper input format for the majority of the main algorithms (ASNE, DeepWalk, node2vec and LeaderRank), I had to preprocess the datasets in several steps: I had to create an edge list containing all the edges in the graph, and I had to create an attribute list containing the values of all attributes for all nodes.

For the social distancing Twitter dataset, I split the data by week number to acquire five datasets, each corresponding to approximately one week of data.

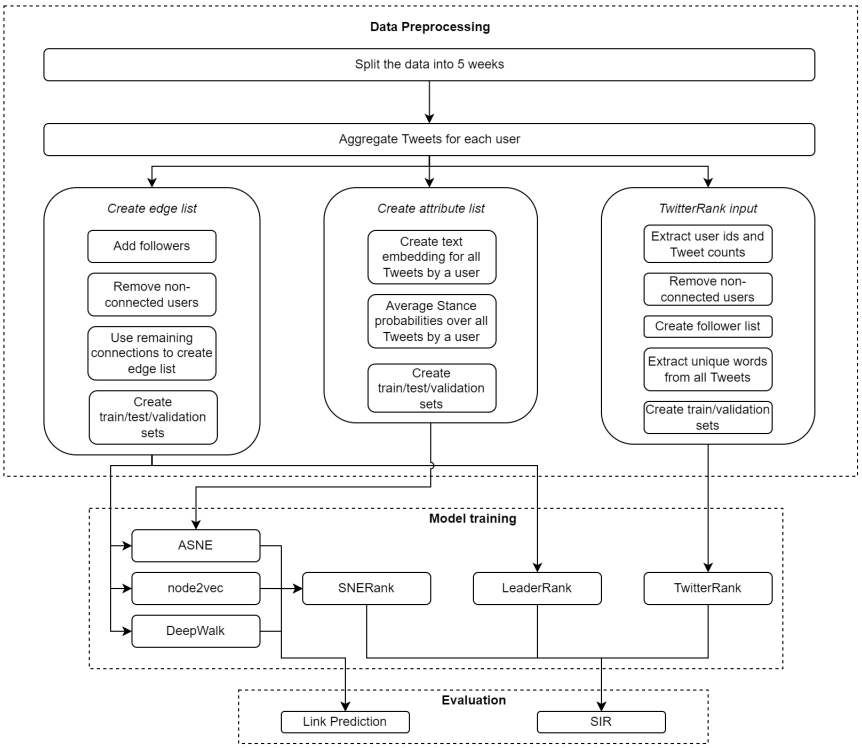


Figure 5: Visualization of the experimental procedure. The data preprocessing creates the edge list, attribute list, and TwitterRank input for each of the 5 weeks in the data; the model training trains the GE and OLD models; the evaluation assesses the quality of each model.

During the training phase, the networks of weeks 2 and 3 turned out to be too large and caused memory errors which made me unable to train the models on these networks. Therefore, I split the data from weeks 2 and 3 up into two parts each and trained those separately.

To create the input of each time interval, I first aggregated all tweets for each user in order to view the data per user rather than per tweet. Next, I added the lists of followers and friends for each user (since these variables were collected separately, as explained in section 4.2). However, these lists contained *all* users ids that were connected on Twitter to the users in the dataset. Therefore, I removed any user ids that did not have a single Tweet in the dataset, so that the remaining connections could be used as edges in the graph. To prevent overfitting, I randomly sampled 10% of the links for hyperparameter tuning, and another 10% as the test set. The models were trained on the remaining 80% of the links, just as done in [10].

Then, to construct the attribute list, for each user I converted the text of

all their tweets combined into a vector through TF-IDF, as explained in section 5.2. This vector forms the Text attribute. However, since an overwhelming majority of the users only published a single Tweet in the dataset, the TF-IDF vector provided mainly useless information: since TF-IDF creates vector representations based on the inverse document frequency, it is unable to provide information that would be helpful in discerning between different nodes for graph embedding with only one single document as reference material.

Therefore, I added a pre-trained language model named FastText [76] to create vector representations of the Tweet text as well. With these two ways to obtain the text attribute, I could compare both to observe which one would be more suited given the current dataset. Thus, I created two variants of the attribute information: one containing the text attribute as created by TF-IDF, and one containing the text attribute as created by FastText.

The Stance attribute consists of three probabilities (supports, rejects, irrelevant: see table 3). Every Tweet has one of each of these three probabilities, that together sum up to 1. Therefore, in order to calculate these probabilities for a user I simply averaged the probabilities from all of a user's Tweets, which resulted in three average probabilities for that user. All three of these average probabilities were added to the attribute list as part of the Stance attribute of a user.

The Text and Stance attributes together make up the attribute information for the nodes, which the ASNE model will use for creating its embedding. The edge list constitutes the structural information, which is what the ASNE, DeepWalk, node2vec, and LeaderRank models will use for creating their embedding.

TwitterRank input The input required for TwitterRank differed slightly from the input required for the other algorithms. Specifically, TwitterRank requires:

- A list of user ids
- A list of Tweet counts for each user
- A list of followers for each user
- A list with Tweet content (words that made up the Tweets) for each user

To create these attributes, I aggregated all tweets for each user, just like with the other algorithms. Afterwards, I could directly extract the user ids and Tweet counts for all users.

I filtered the followers in the same way as before: by removing user ids who did not have a single Tweet in the dataset. Then, the follower input for each user was simply a binary list of size n where n is the total number of users. Each follower would be represented by a 1 in the list, and all other users by a 0.

Finally, to get the Tweet content for each user I simply extracted the unique words from all Tweets made by a single user and concatenated them together in a list to be given to TwitterRank as input.

6.2.2 Training the models

Graph Embedding To properly train the models, I first experimented with different hyperparameter settings to determine the best values for each model and each dataset. The values I selected to run these tests with are adapted from several papers also testing these models [4, 10, 33, 31] and can be found in Table 5 with the values generating the best results displayed in bold.

Model	**Parameter**	**Social distancing dataset**			**Facebook dataset**		
ASNE	batch size	64	**128**	256	64	**128**	256
	learning rate	0.1	0.01	**0.001**	0.1	0.01	**0.001**
node2vec	p	0.5	**1**	2	0.5	**1**	2
	q	0.5	**1**	2	0.5	**1**	2
	Walk length	20	40	**80**	20	**40**	80
	Walks per node	**10**	20	40	**10**	20	40
DeepWalk	Walk length	20	**40**	80	20	**40**	80
	Walks per node	**10**	20	40	**10**	20	40
TwitterRank	γ	0.1	**0.2**	0.3	0.1	0.2	**0.3**

Table 5: Tested hyperparameter settings for both datasets. Best-performing values are displayed in bold.

After finding the best values for the hyperparameters of each model, I trained all embedding models on the datasets using these values for the hyperparameters. To try and ensure that the models were trained until convergence, I trained them for 100 epochs each. Observing the results does confirm that the models converged after this time, so I deemed it not necessary to increase the number of epochs for any of the models.

I also trained multiple versions of the ASNE model with different attribute parameters on week 1 of the data:

- A version including both the Text and Stance features
- A version with only the Text feature
- A version with only the Stance feature
- A version with no attribute information at all

Training these models and comparing the results on the link prediction task will provide insight in the specific effects of these attributes alone.

For the baseline Facebook dataset I did not have any attribute information. As such, the input for all embedding models is simply the edge list containing the structural information of the graph.

Model robustness The robustness of a graph embedding model depends in part on how it performs with more sparse networks. Therefore, I also trained the three graph embedding models (ASNE, node2vec, and DeepWalk) on the data

from week 1 with varying training link ratios. The change in performance as less and less links are available for training indicates the robustness of the graph embedding model: the faster the decrease in performance with less training links, the less robust a model is [10].

Ablation Since there is some randomness involved in creating the test set (selecting random nodes to check in the link prediction task), I also ran an ablation study to determine the effect of randomness on the results of the study.

I generated 10 different test sets using the exact same procedure (so the only factor affecting changes between the test sets is the randomness involved) and tested the data from week 1 on all these test sets.

If the randomness does not have a major effect on the quality of the results, the link prediction results should be (nearly) identical. If they are not, this indicates that the random variation might drastically affect the results.

Opinion Leader Detection I then trained three Opinion Leader Detection models: TwitterRank and LeaderRank directly on the data, and SNERank on the embeddings created by DeepWalk, node2vec, and the various versions of ASNE. For TwitterRank I used the best value for γ as found by the hyperparameter testing (Table 5), whereas LeaderRank and SNERank did not require any hyperparameter tuning.

Similar to the graph embeddings, for the Facebook dataset I simply used the edge list as input for LeaderRank, and the embeddings from ASNE, node2vec and DeepWalk for SNERank. Unfortunately, since this dataset does not contain any Tweet information, TwitterRank could not be used on this dataset.

6.2.3 Evaluation

In order to evaluate the quality of the graph embeddings, I used the link prediction task as explained in section 5.5. The AUROC score of the models provides an indication of the quality of the embedding.

To evaluate the results of the Opinion Leader Detection methods, I used Kendall's τ [68], as well as a variant of the SIR model [67]. With the SIR model I tested several different hyperparameters as well, to determine the influence of text similarity on the spread of information, similar to what is discussed in [13]. Similar to the other experiments, to test the hyperparameter values I used week 1 of the social distancing data. The specific values of the hyperparameters that I tested can be found in Table 6 and are based off of [13].

Parameter	Values	
λ_1	0.1	0.3
λ_2	0.1	0.3
ε	0.2	0.4
# initial infected	20	50

Table 6: Hyperparameters used in the SIR model.

7 Results and discussion

7.1 SIR hyperparameter tuning

Figure 6 contains two of the results from the SIR hyperparameter testing. The complete results can be found in Appendix A.

The hyperparameters λ_2, ϵ, and N_0 (as explained in section 5.5) offer little influence on the spread of the information throughout the network for any of the five models. The λ_1 parameter, however, does affect the rate at which the information spreads through the network: SIR models with $\lambda_1 = 0.1$ converge more slowly compared to SIR models with $\lambda_1 = 0.3$. Despite this slower convergence, all models do achieve a similar spread of information after converging, which can be observed near the final iterations of the graphs and in the zoomed-in parts of the plots.

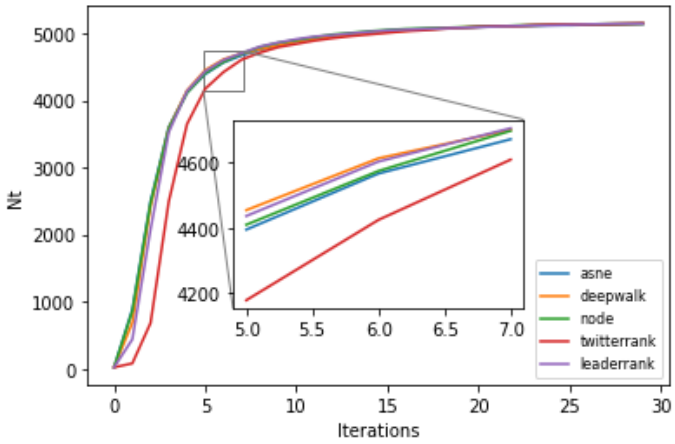
Since $\lambda_1 = 0.3$ offers faster convergence of the SIR models, I used this value for the evaluation of the opinion leader detection models. Because the other parameters do not seem to impact the results with the chosen value ranges, I used $\lambda_2 = 0.1$, $\epsilon = 0.2$, and $N_0 = 20$, as these correspond to values used in [13] (alongside $\lambda_1 = 0.3$), thus allowing for easier comparison between their results and my own. The results for these hyperparameter values can be observed in Figure 6.

7.2 Model robustness

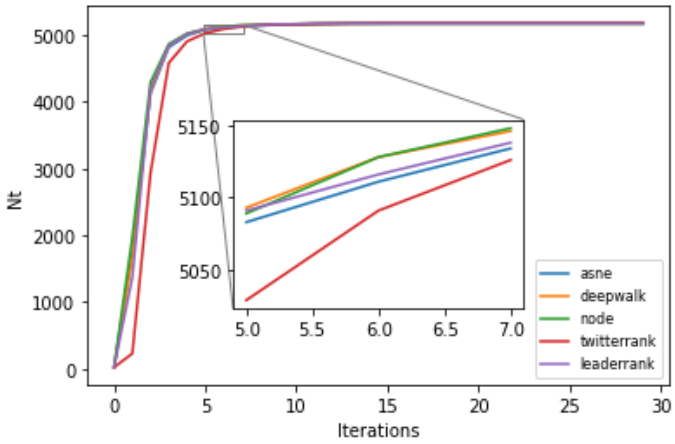
Figure 7 shows the performance of ASNE, DeepWalk and node2vec with regards to different levels of training links. One important observation is that ASNE decreases much more quickly than node2vec and DeepWalk do. Furthermore, node2vec hardly decreases at all and DeepWalk's performance only worsens slightly even with less than half of the links used for training.

This indicates that the effect of attribute information (which is only present in ASNE, but not in node2vec or DeepWalk) does not compensate for link sparsity, as the performance decrease in ASNE is stronger than in the other two embedding models. Node2vec's higher performance on all levels of network sparsity compared to DeepWalk could be explained by the balanced random walk that node2vec uses, which allows it to more easily maintain higher-order proximity information that DeepWalk does not have access to.

Comparing these results to a similar test done in [10], the performance of ASNE on the link prediction task is similar with high ratio of links used for



(a) λ_1: 0.1, λ_2: 0.1, ϵ: 0.2, N_0: 20



(b) λ_1: 0.3, λ_2: 0.1, ϵ: 0.2, N_0: 20

Figure 6: Two results from testing the different combinations of hyperparameter values from Table 6 on week 1 of the data. N_0 is the number of initial infected nodes before iteration starts. In the legend, "asne" corresponds to the ASNE embedding model + SNERank, "deepwalk" is DeepWalk + SNERank, "node" is node2vec + SNERank.

training, but the performance drop as the network sparsity increases is greater on this data than reported for most datasets in [10], with the exception of the CITESEER dataset, which features a similar decrease in performance.

The relative decrease in performance from both node2vec and DeepWalk on my dataset is similar to what is reported in [10], but both models perform vastly superior on all levels of network sparsity in that paper than on my data. I explore this unexpected result in more detail in section 7.4.

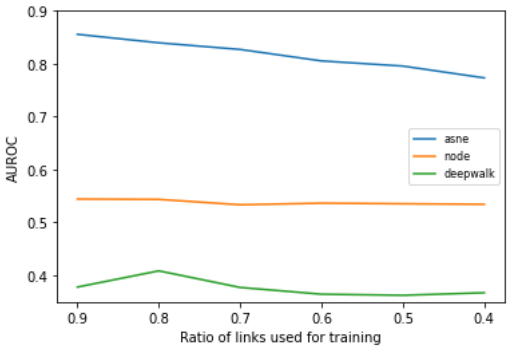


Figure 7: Performance of link prediction on week 1 of the data with varying levels of network sparsity.

7.3 Ablation

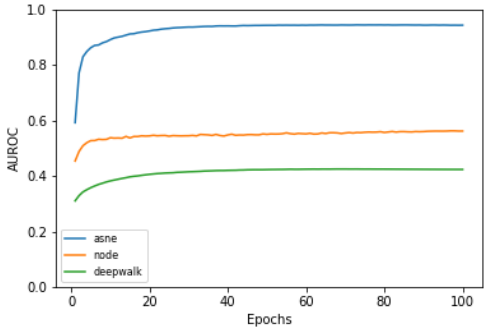
Figure 8 shows two of the results of the ablation study to investigate the effect of randomness on the outcome of the link prediction task. Each image represents the results of one of the 10 randomly generated test sets as described in section 6.2.2. The complete results showing all 10 graphs can be found in Appendix 17. Figures 8 reveals that the results of the tests are essentially identical, indicating that the effect of randomness is negligible on the outcome of the link prediction task.

Since the results on the link prediction task are nearly identical, I postulate that the way in which the data sets are split using randomness does not play a noticeable role in finding opinion leaders either, since the embeddings will be of the same quality regardless of how the data for training and testing has been split.

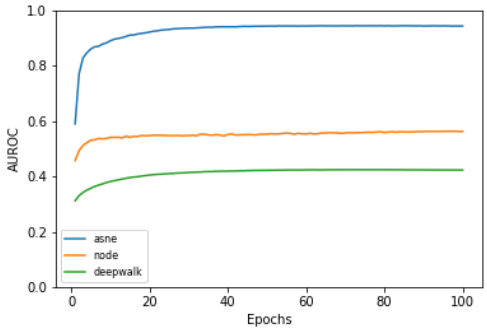
7.4 Graph Embedding

7.4.1 Social distancing dataset

As the results of the link prediction task show, the models perform similarly over all weeks of the data. This indicates that both the structural and attribute information contained in the data are consistent over time, which enables the embedding models to learn these embeddings that also perform similarly over time. The ASNE model is the slowest to converge, but it converges to a much higher AUROC value (0.94) compared to both DeepWalk and node2vec. DeepWalk converges to AUROC values of 0.42, whereas node2vec converges to approximately 0.56.



(a)



(b)

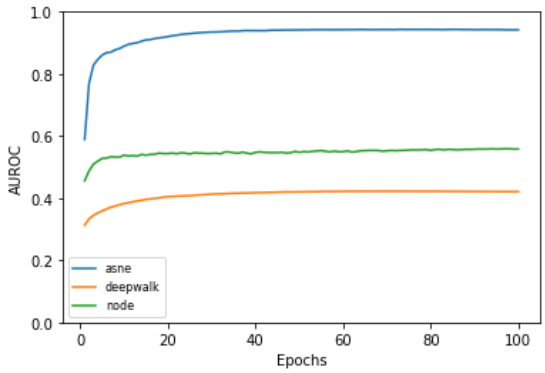
Figure 8: Partial results from the link prediction task on week 1 of the social distancing data for the ASNE, DeepWalk and node2vec graph embedding models. Each image corresponds to a different randomly generated test set.

7.4.2 ASNE variants

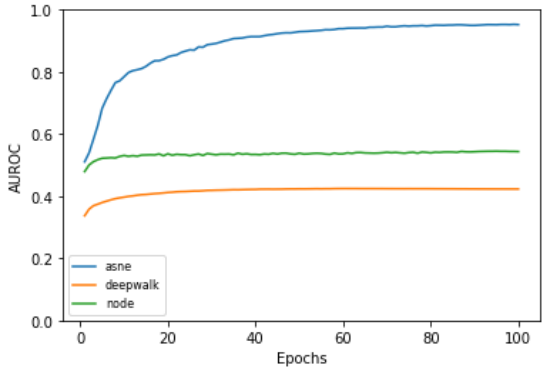
Figure 10 contains the results of the DeepWalk and node2vec models for reference, in addition to several variants of the ASNE model:

- ASNE with FastText and Stance attributes (asne_fasttext)
- ASNE with TF-IDF and Stance attributes (asne_tfidf)
- ASNE without any attribute information (asne_without_attr)
- ASNE with the Stance attribute but no text attribute (asne_only_stance)

As can be seen in Figure 10, both the "ASNE with FastText and Stance" variant and "ASNE with only Stance" variants converge to the same value, but "ASNE with only Stance" converges much faster, indicating that the text attribute is actually slowing down the training process for ASNE on this dataset.



(a) Week 1



(b) Week 4

Figure 9: Partial results of the Link Prediction task on the ASNE, DeepWalk, and node2vec graph embedding models on the social distancing dataset. Results from all weeks can be found in Appendix C.

Furthermore, "ASNE with TF-IDF" has a very irregular training process, with the performance fluctuation slightly between different iterations. Even though there is still an overall positive trend, it converges to a lower value than the other ASNE variants. Finally, the "ASNE without attributes" variant shows similar training progress to the "ASNE with FastText and Stance" but performs slightly worse at all iterations, indicating that the overall effect of attribute information is positive.

Despite the performance of ASNE being substantially higher than that of DeepWalk and node2vec, this dramatic increase in performance cannot be credited solely to the presence of attribute information in ASNE (which is not used by DeepWalk and node2vec): Figure 10 shows that when excluding the attribute information from ASNE, the model's performance only drops slightly (from 0.94 to 0.92).

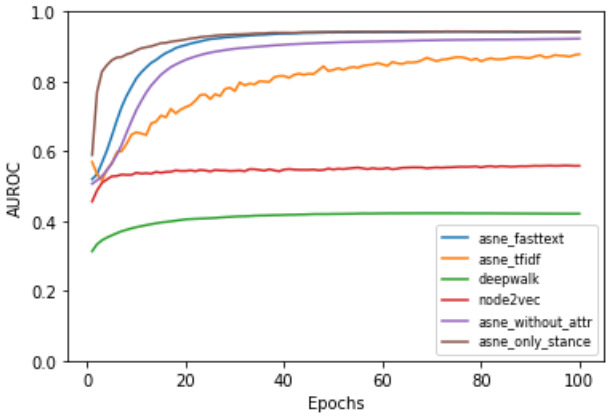


Figure 10: Performance of link prediction on week 1 of the data, including several variants of ASNE.

Comparatively, when trained on the exact same data, DeepWalk only has an AUROC value of 0.42, and node2vec has a value of 0.55. Figure 9 indicates that the results of the link prediction task are stable across all weeks in the data, which shows that the training is systematically underwhelming for DeepWalk and node2vec.

Furthermore, as Figure 8 shows, the randomness factor has also been accounted for regarding its influence on the embedding results: there are no substantial differences between any of the experiments conducted on the 10 different randomly generated test sets. As such, the only remaining factor influencing results are the embedding models themselves.

Therefore, the way in which ASNE creates the embeddings makes up the majority of the difference in embedding quality, compared to any other factor such as attribute information or randomness.

7.4.3 Facebook dataset

The performance of both DeepWalk and node2vec increases only marginally over the course of the training process on the Facebook dataset, as shown in Figure 11. Conversely, ASNE follows a similar curve to the social distancing datasets, and reaches near-perfect AUROC when it converges.

Given that the Facebook dataset has also been used to test both DeepWalk and node2vec in [33], this result is quite alarming, as in that paper both models achieved an AUROC value of up to 0.9680, compared to DeepWalk's 0.51 and node2vec's 0.54 in my experiment.

The dataset and source code are, to the best of my knowledge, the same in both experiments. I also used the same hyperparameter settings as reported in [33].

When looking into the embedding vectors created by DeepWalk and node2vec,

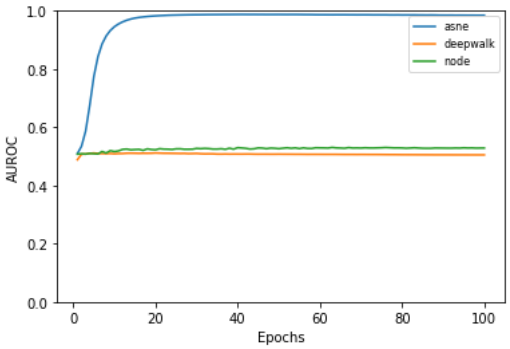


Figure 11: Performance of link prediction on the Facebook dataset.

I noticed that the vast majority of the nodes were vastly different from each other. To calculate their similarity, I used the cosine similarity measure. Almost all node embedding pairs had a cosine similarity of near 0. Compared to the average pairwise cosine similarity of ASNE (0.43), the average pairwise cosine similarity of DeepWalk (0.02) and node2vec (0.04) are substantially lower.

This might be the reason why the link prediction task has trouble differentiating between node pairs that should have an edge between them, and node pairs that should not. In that case, the AUROC hovering around 0.5 would make sense given that the link prediction task is primarily based on the cosine similarity.

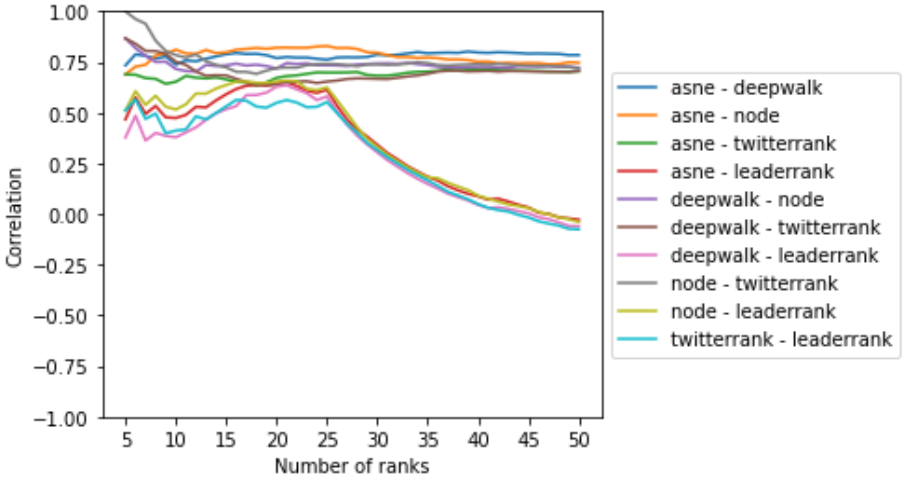
However, due to time constraints associated with the Master thesis I was unable to allocate additional time to investigating this finding further. As such, exploring these results more could prove a fruitful area of research.

7.5 Opinion Leader Detection

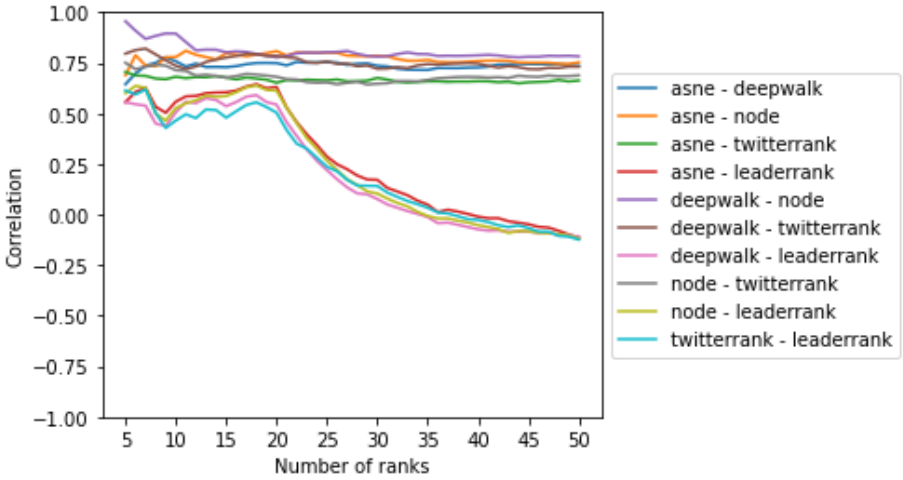
7.5.1 Kendall's τ

The pairwise correlation displayed in Figure 12 between the algorithms follows a similar trend for each week of the data. Generally, the correlations between ASNE, DeepWalk, node2vec, and TwitterRank are quite high (around 0.75) whereas the correlations between LeaderRank and the other methods starts off high as well, but tapers off towards 0 as k increases. This shows that the methods typically agree well on the top opinion leaders, but LeaderRank's selection process differs from the others for lower ranks, so as the number of ranks increases, the ranking created by LeaderRank starts to diverge from the rankings created by the other methods.

The graphs in Figure 12 display the pairwise correlation between the opinion leader detection models for different numbers of ranks. The higher the Kendall's τ correlation between two models, the more similar the orderings of the data are. As can be seen in Figure 12, the agreement between ASNE, DeepWalk, node2vec and TwitterRank is consistently high (larger than 0.50). This indicates that the



(a) Week 1



(b) Week 4

Figure 12: Pairwise correlations between ranking lists. *Number of ranks* corresponds to the top k ranks in the ranking lists. Results from all weeks can be found in Appendix D.

rankings created by these four models are monotonously related: high values in one of these rankings is associated with high values on the other rankings.

Contrary to the previously mentioned four models, the LeaderRank model is an outlier as its correlations with any of the other models is not stable at all. It does start off similarly with high correlations for the first several ranks. However, as the number of ranks increases, the correlation quickly tapers off

towards 0, and sometimes even below 0.

Note that the Kendall's τ correlation does not provide any evidence as to which users are *actually* opinion leaders: it simply measures how similar the rankings created by the opinion leader detection models are. This important distinction can be illustrated by a simple example: if every model simply selects the first 20 nodes in the dataset as opinion leaders, then they would all share a Kendall's τ correlation of 1 (indicating perfect agreement). However, it is highly unlikely that the first 20 nodes in the dataset will actually be the real opinion leaders. More importantly, for the purposes of the Kendall's τ correlation, which users are actually opinion leaders is irrelevant: whether the first 20 or last 20 nodes are the real opinion leaders, the Kendall's τ correlation will still be 1, simply because the models are in agreement.

As such, the results in Figure 12 indicate that the models are in high agreement over the opinion leaders, with the exception of LeaderRank, which shows moderate to high agreement with the other models only for the top 20 opinion leaders.

7.5.2 Shared opinion leaders

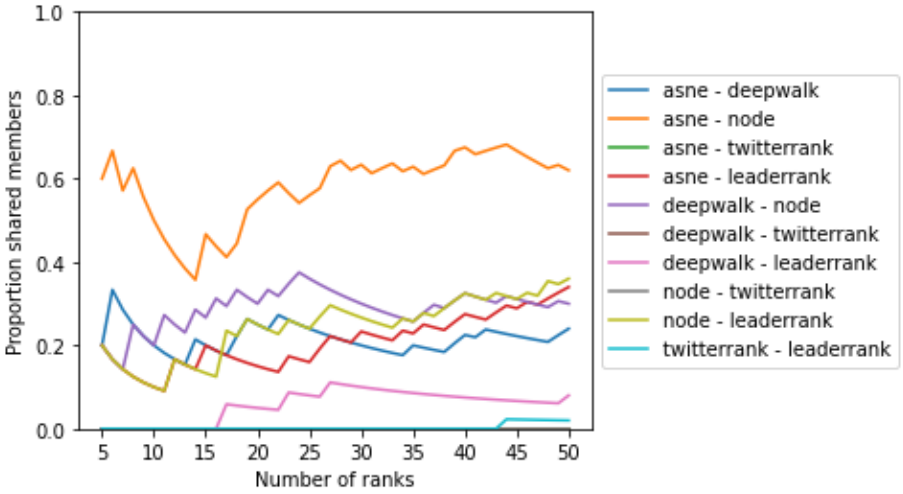
The graphs in Figure 13 indicate that there is substantial overlap between the opinion leaders selected by the embedding-based ranking algorithms (ASNE + SNERank, node2vec + SNERank, and DeepWalk + SNERank). Furthermore, LeaderRank and the aforementioned embedding-based also share a small to moderate proportion of opinion leaders. Conversely, TwitterRank shares *no* opinion leaders with the other four ranking algorithms in most weeks, indicating that its method of selecting opinion leaders is substantially different from those methods.

This difference might be explained by the design of the algorithms: TwitterRank is designed heavily around the Tweet content of the users in addition to structural information, as explained in section 5.4, whereas node2vec, DeepWalk, and LeaderRank primarily involve identifying opinion leaders based on structural information. ASNE does also take Tweet content into consideration, but it also involves other factors such as the structural information and the user's stance.

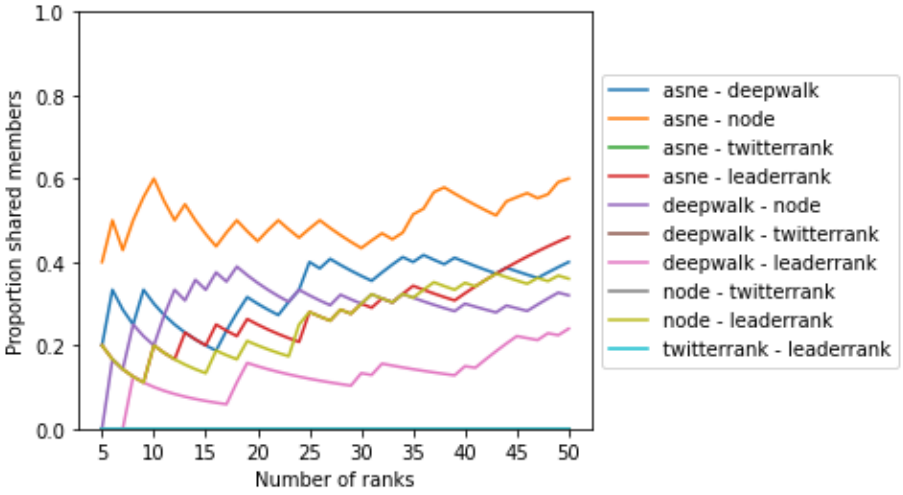
Since the social distancing dataset contains few Tweets for most users (as outlined in section 4.3), TwitterRank might not be able to properly distinguist between different users due to the data sparsity.

7.5.3 SIR

The SIR results shown in Figure 14 show very similar spreading rate as well as similar total spread (measured by total number of users infected) for most models across the several weeks spanning the social distancing data. The exception to this is the TwitterRank model, which generally converges at a slightly slower rate than the other models.



(a) Week 1

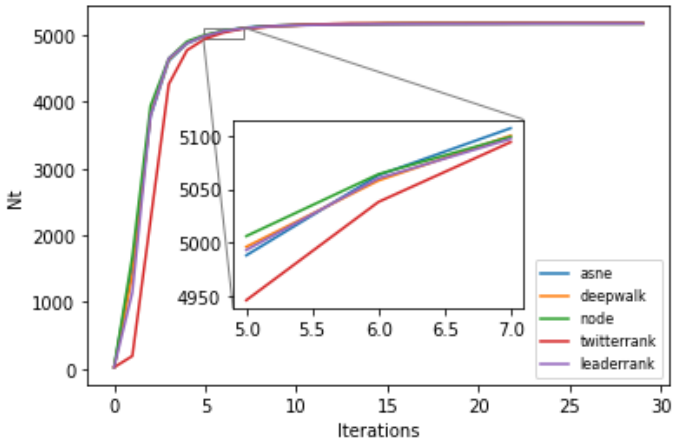


(b) Week 4

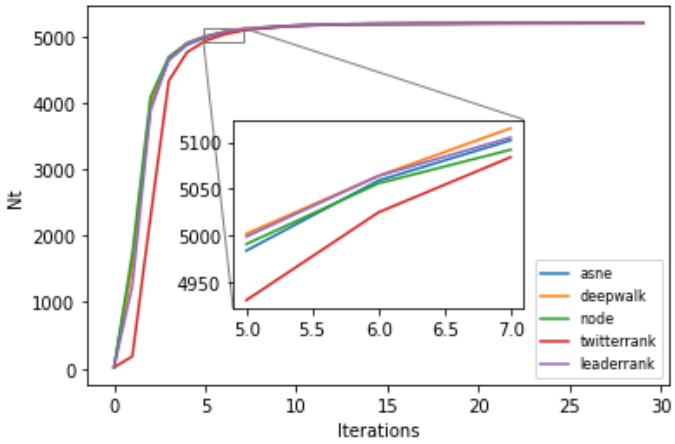
Figure 13: Proportion of shared opinion leaders between ranking lists. OLs are shared between ranking lists when they occur in both lists, regardless of position. *Number of ranks* corresponds to the top k ranks in the ranking lists. Results from all weeks can be found in Appendix E.

I have not included the variants of ASNE as shown in Figure 10 because their performance on the SIR task was identical to the ASNE model with FastText and the user stance.

Since the SIR model measures how well the chosen opinion leaders cause



(a) Week 1



(b) Week 4

Figure 14: Partial results of the SIR model for the ASNE + SNERank, DeepWalk + SNERank, node2vec + SNERank, TwitterRank, and LeaderRank methods. Results from all weeks can be found in Appendix F.

a cascade of spreading information throughout the network, we can conclude that the opinion leaders found by the five models perform very similarly in this regard. The curves are remarkably similar, disregarding the minute change in the curve of TwitterRank. Therefore, even though the opinion leaders chosen by the models are not always the same (as can be seen in Table 7, whichever users are classified as opinion leaders still spread the information throughout the network efficiently.

In [13], the authors perform a similar analysis with SIR, where the values of λ_1, λ_2 and ϵ affect how far the information spreads throughout the network: with

$\lambda_1 = 0.3$, $\lambda_2 = 0.1$, and $\epsilon = 0.2$, the text attribute influences the information spread, which leads to their SNERank model outperforming TwitterRank.

On the social distancing dataset, with the same values for λ_1, λ_2, and ϵ, I do not observe this same result. This might be affected by the content of the text attribute of this dataset: in [13], the text attribute of each user is made up of 500 Tweets. However, for the social distancing dataset, most users only have a single Tweet, and no user has more than 162 Tweets, as shown in Figure 3. This means that there is a lot less text information making up the attribute for each user. As such, there might simply not be enough data to replicate the difference that was observed in [13].

Nonetheless, the graphs in Figure 14 do indicate that the selected opinion leaders spread the information around effectively and efficiently, regardless of the method that was used to select the opinion leaders.

7.6 Qualitative results

7.6.1 Top 5 opinion leaders

Week		1	2 (1)	2 (2)	3 (1)	3 (2)	4	5
ASNE	1	Matthijs85	JohnZuyderduyn	Matthijs85	EelcoHoecke	GrootKo	Matthijs85	ducom99
	2	JohnZuyderduyn	OBraeckenssieck	ZilteBotte	Hannesz1956	Hannesz1956	ZilteBotte	lewinskylou2
	3	Hannesz1956	Hannesz1956	2deKamerFVD	OBraeckenssieck	Bos_M	OBraeckenssieck	OBraeckenssieck
	4	GrootKo	Ingeborgvraagt	Hannesz1956	Gaia_Universe	adrianxleconte	poeetweet	Hannesz1956
	5	OBraeckenssieck	FlapFriesland	OBraeckenssieck	moeva18	FTI_momentum	FlapFriesland	widtvoet
DeepWalk	1	Breinbrouwsels	EricStallinga	2deKamerFVD	EelcoHoecke	colourbird00	2deKamerFVD	LidwienNews
	2	MastersLars	FlapFriesland	GoThorium	OBraeckenssieck	celalaltuntas1	johnjljacobs	esther241101
	3	jokebronkhorst	FLP05480448	burgercomiteeu	vlimmertje	frelke75	FlavioPasquino	Breinbrouwsels
	4	Hannesz1956	burgercomiteeu	Matthijs85	flakes010	Bengegenislam	OBraeckenssieck	MastersLars
	5	twopcharts_nl	HrothN	PeterK43579783	Wegaandiep	ter_borgh	MarianneCramer	rinushoogstad
node2vec	1	Matthijs85	AYingyang2012	Jetemetet	Ri_qua0910	AmberBrouwer2	Ron_ZCNH	TheRightNL
	2	twopcharts_nl	Ingeborgvraagt	CorrieBult	Hoofdzuster1	pim_braakhuis	poeetweet	AsbaiBadr
	3	GrootKo	EricStallinga	Matthijs85	laCarretje	AdriaanBeenen	Matthijs85	ducom99
	4	elliedeb2	ZiggoWebcare	2deKamerFVD	EelcoHoecke	werthernieland	SpyMacho	Edvdijk
	5	OBraeckenssieck	PeterdeLeur	Wulffraat	OBraeckenssieck	eradus2911	ann_castrel	TonvanDam
TwitterRank	1	pApowerX	haitskevdlinde	coco_riga	Martin68676610	Rob020111	jorispinter3	jop_n
	2	arnovj	Schnitger2017	Rikkkiee	EddieBree	pAUL91872260	SMeerbeeck	AnSophEls
	3	NLCryptoOracle	eetschrijver	AbrahamOmer1	dan_kroon	oltho57	_Ramona27_	cvrauwdeunt
	4	Specifiek1	LMirjamv	ottomanius	bastion2211	KOKCON	NathalieBosman2	IENouwen
	5	DamnJen_	MissG47013113	RudivanZandwijk	CarloGiugie	Geleiding	Fred_1277	BrouwersKarin
LeaderRank	1	MinPres	hugodejonge	MinPres	rivm	volkskrant	rivm	wierdduk
	2	Matthijs85	nrc	mauricedehond	RTLnieuws	RTLnieuws	FTM_nl	nrc
	3	thierrybaudet	telegraaf	FTM_nl	SaskiaBelleman	Nieuwsuur	Matthijs85	shossontwits
	4	hugodejonge	umarebru	Matthijs85	wierdduk	telegraaf	RTLnieuws	umarebru
	5	telegraaf	chrisklomp	RTLnieuws	hugodejonge	hugodejonge	vanranstmarc	op1npo

Table 7: Top 5 opinion leaders for the Opinion Leader Detection methods over all weeks.

Table 7 shows the top 5 opinion leaders selected by the 5 opinion leaders for all the weeks making up the social distancing dataset. Upon inspection of the selected users, these choices make intuitive sense: many of the users are either

prominent Dutch politicians or news outlets, whose Tweets typically have a high engagement rate and reach a large audience. Some examples include:

- "Matthijs85" is the Twitter account of Matthijs Pontier, a Dutch politician in the Pirate party who Tweets regularly during the pandemic and whose Tweets get a lot of interaction.
- "thierrybaudet" is the Twitter account of Thierry Baudet, the chairperson of the Dutch right-wing political party Forum for Democracy, whose Tweets regularly generate controversy, which results in them reaching a massive audience.
- "telegraaf" is the Twitter account of the Telegraaf, the largest Dutch daily newspaper. They Tweet very often about the news, and especially during the pandemic their Tweets receive a high engagement rate when reporting on new development surrounding the Coronavirus.
- "MinPres" is the Twitter account of Mark Rutte, the current Prime Minister of the Netherlands. Just like the other politicians he often Tweets about new developments concerning the pandemic. Since these Tweets are often published shortly after new announcements have been officially made public, it stands to reason that these Tweets could be many people's first time hearing the news. Because of the topic's inherent controversial subject matter, these Tweets always receive a huge amount of responses and spark large debates in the replies. As such, it is no surprise that the "MinPres" Twitter account is selected as an opinion leader.

The remainder of the users are mostly Twitterers with a relatively small number of followers that either publish a lot of Tweets, or have published one or two Tweets that have become very popular. For example:

- "esther241101" is a user with less than 100 followers who has published 30 Tweets during the period of the social distancing dataset, which is more than 99.9% of users in the network. While none of their Tweets have become popular, the relatively high number of Tweets still makes them stand out.
- "frelke75" is a user with less than 700 followers who has published 5 Tweets during the period of the social distancing dataset, one of which gathered over 2500 likes, indicating that it reached a large audience.

Though these are not definitive indicators as to why these users were selected as opinion leaders, they do make sense from an intuitive standpoint, especially given the extremely limited amount of Tweet information per user.

Furthermore, some similarities can be observed between the embedding-based algorithms: the user "OBraeckenssieck" appears in the top 5 for all three methods throughout the weeks, but it does not appear in the top 5 for either

TwitterRank or LeaderRank in any week. This user has a large number of followers (11.326) and has published 12 Tweets during October 2020, which would put him in the top-1% most active Twitter users in the social distancing dataset.

However, this user has a large in-degree in the graph (537), compared to its out-degree (87). It is possible that the embedding algorithms value this property higher than TwitterRank or LeaderRank, which is why they rank this user so highly.

7.6.2 Network visualization

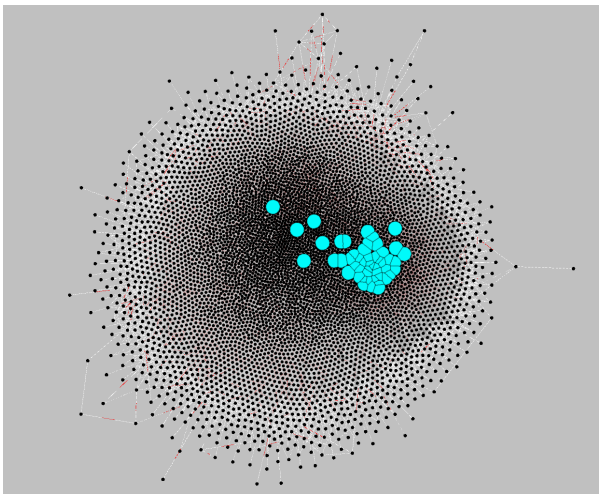
The visual representation of the network in Figure 15 shows the opinion leaders closely grouped together. This area of the network contains nodes with relatively high degree, with the nodes towards the right of the figure specifically having a high in-degree compared to out-degree. This is confirmed by the observation that all the opinion leaders depicted here have a high in-degree.

Intuitively, this is a logical observation: opinion leaders influence other users, and the easiest way to do that is to directly affect a large number of users through Twitter's following system. By reaching a large number of people with each Tweet, the chances of influencing a larger number of people with those Tweets also increase. Conversely, by having a small number of followers, it is more difficult to influence a large number of people because each Tweet does not necessarily reach the same large audience. As such, it is logical for the opinion leaders as depicted here to have a large in-degree, corresponding to a large number of followers.

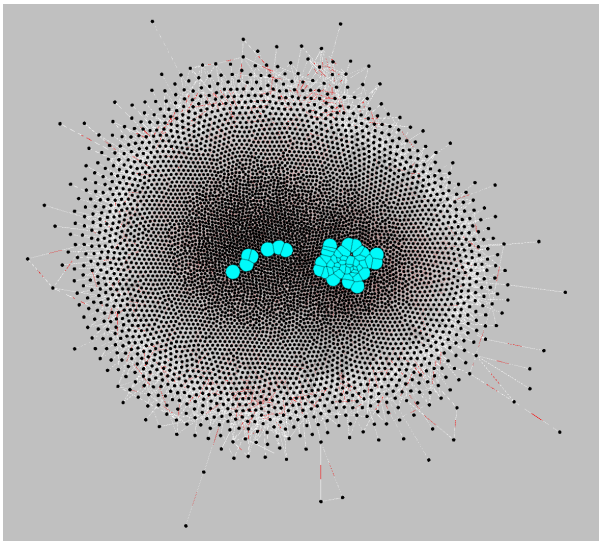
Even though the network has changed slightly between weeks 1 and 4, the opinion leaders are relatively stable in terms of their overall position within the network. This indicates that the large in-degree is a trait that most opinion leaders possess which remains stable over time. Even though not the exact same nodes have been selected as opinion leaders in both weeks, they do all share a similar position within their respective networks.

7.7 Research questions

What is the influence of including both semantic and structural information in a graph embedding on opinion leader detection, compared to using only structural information? The ASNE model can incorporate both semantic and structural information when creating its embeddings, whereas embedding models such as DeepWalk and node2vec can only incorporate structural information into their embeddings. Figure 7, 8, 9, 10, and 11 all show the differences between ASNE, DeepWalk and node2vec, with ASNE clearly outperforming the other two options. However, Figure 10 shows the comparison between different variants of the ASNE model, both with and without attribute information. It is this result that best showcases the influence of using both semantic and structural information, compared to using only structural information. When semantic information is included in the ASNE model, the performance on the link prediction task increases by 2%, as the AUROC



(a) Week 1



(b) Week 4

Figure 15: Network visualization of week 1 and 4 of the social distancing dataset. Black dots correspond to nodes, the white/red lines are the edges connecting the nodes together. The large cyan dots represent the top-50 opinion leaders as found by the ranking algorithms. Nodes are placed according to the force-directed layout configuration, which attempts to minimize cross-over between edges. Images were created using Graphia [77].

score increases from 0.92 to 0.94. By looking at Figure 9 I deduce that these performances are similar over time, showing that DeepWalk and node2vec sys-

tematically score lower than any of the variants of ASNE. When looking at Figure 14, there is no discernable difference between the performance of the ASNE model, and that of DeepWalk and node2vec, which both only used structural information.

As such, I argue that the semantic information improves the quality of the graph embeddings, but not the quality of opinion leader detection models.

What is the influence of using graph embeddings on opinion leader detection? To determine the influence of graph embeddings on opinion leader detection, I tested different classes of models on the social distancing data. As Figures 12 shows, there is a large difference between ASNE, DeepWalk, node2vec and LeaderRank which might indicate that this is caused by graph embeddings. However, that same figure also shows that there is fairly minor difference between ASNE, DeepWalk, node2vec and TwitterRank, even though the latter also does not use a graph embedding method.

Therefore, I argue it is unlikely that this difference is caused by the presence or absence of a graph embedding, rather that it is more likely caused by another factor. For example, the influence of the text attribute might affect the chosen opinion leaders more, as both SNERank and TwitterRank use this information to decide on their opinion leaders.

Furthermore, Figure 14 shows that there is essentially no difference between ASNE, DeepWalk, node2vec and LeaderRank since both the spread of information as well as the rate at which the information spreads are nearly identical. Even though there is a slight difference in the rate at which information spreads for TwitterRank compared to these other four models, it is not nearly large enough to imply any substantial differences between the models' performance exists.

One of the main motivations for using graph embeddings is the computational efficiency that they offer. However, in my experiments, running the graph embedding models followed by an opinion leader detection method took up significantly more time compared to running just the opinion leader detection methods that did not require embeddings (TwitterRank and LeaderRank).

As such, according to the findings in this thesis, graph embedding has a negligible influence on opinion leader detection.

What is the influence of a user's stance (positive or negative) on detecting opinion leaders? As is shown in Figure 10, the ASNE model containing only the stance probabilities as attribute information converges faster than any other version of ASNE, and, after convergence, achieves the same AUROC score as the ASNE model including both the stance and the FastText text attribute on the link prediction task.

Additionally, when comparing the ASNE model with only the stance to the ASNE model without any attributes, the former clearly has a superior performance on the link prediction task, since it converges faster and attains a higher AUROC score.

Given these two observations, I argue that the Stance attribute has a positive on the quality of graph embeddings.

However, when taking into account the conclusion to the previous question, and observing no major differences between models with the Stance and models without the Stance attribute on either the Kendall's τ correlation metric or the results of the SIR model, I argue that the overall effect of the user's stance on opinion leader detection is negligible, just like the effect of graph embedding in general.

It is still possible that this factor could provide a positive influence to opinion leader detection if it was modelled differently (perhaps without graph embedding), but the results from my experiments do not allow me to make such a conclusion. That being said, this is certainly an area worth investigating, given the increase in performance on link prediction.

How do opinion leaders develop over time in a social network? Table 7 shows the development of the top 5 opinion leaders over time, as selected by all five opinion leader detection models. It is clear that the selected opinion leaders are not stable over time. Even though several users appear in multiple lists across time and across different models, there is also a lot of variation in the users that appear in the top 5 opinion leaders.

A critical observation to make here is that this is in part due to the sparsity of the data. Many users did not publish a Tweet in every single week of the social distancing dataset. On the contrary, many users only appear in one of the weeks due to only having published a single Tweet in the entire dataset. This indicates that it is entirely possible a user would have been an opinion leader across all weeks, but they just happened to not have a single Tweet for some weeks, which means they are automatically ineligible to become an opinion leader for those weeks.

This observation makes the results in Table 7 seem more volatile than they likely are in reality. Despite these confounding factors, numerous users still appear in multiple weeks. As such, I argue that the opinion leaders would likely have been rather stable if the dataset had contained more Tweets from all users over time. This is supported by the network visualization in 15, which shows that the opinion leaders hold a relatively stable position within the network over time. Although this is far from the only factor influencing the selection of opinion leaders, it does lend additional credit to the postulation that the volatility of the attribute information might be the cause of the changes in opinion leaders over time.

8 General discussion

8.1 Data

As mentioned in section 4.3, the vast majority of users only have 1 Tweet in the dataset, which makes it difficult to draw conclusions regarding the effectiveness

of some of the methods. For example, the SNERank and TwitterRank models rely heavily on the text attribute, and the SIR model also bases its probability of spreading infection on the similarity between text attributes. Furthermore, I wonder if the data truly is representative of the topic "social distancing". Since Twitter is heavily censoring Covid-19 misinformation [78], some parts of the discussion might be lost. To combat this, some people have started using different terms to describe the same phenomenon, with the express purpose of avoiding Twitter's censorship. For example, instead of talking about "social distancing", people might be talking about the "1.5 meter circle", or the "1.5 meter civilization".

These terms are not picked up by our filter, and as such we missed out on potentially useful data regarding particular perspectives concerning social distancing. Another way of avoiding Twitter's misinformation filters is to intentionally misspell key words in the Tweet (for example "socail dlst" instead of "social distancing").

Repeating the experiments from this thesis with a larger, richer dataset might yield results that are more consistent both within themselves and compared to previous research [13, 31, 33].

8.2 Evaluating Opinion Leader Detection models

When experimenting with opinion leader detection methods, having a way to compare the results of the methods is of vital importance. Comparing the methods, not only to each other, but to a baseline or ground truth can add much needed context to the experiment. Because of this, I was stunned during my literature research that there seemed to be no industry standard way of evaluating OLD models.

As covered in section 3.2.3, many authors devise their own measure for evaluating the OLD model they discuss in their paper [3, 13, 55, 61, 62, 63, 64, 65]. This makes sense because this way, they can adjust the metric to properly fit with their model, which makes it easier for them to perform the evaluation.

However, the cost of this is that there is a plethora of different evaluation metrics, none of which can be considered to be the standard way of evaluating a model. This hampers the ability to properly compare different models with each other, as the results might change depending on which evaluation method is used. As such, one method will always hold an advantage over the other(s).

Unfortunately, this problem is further exacerbated by the fact that the definition of what exactly constitutes an opinion leader is notoriously vague [61]. This makes it even more difficult to compare different methods against each other. If one has a different definition of an opinion leader, then both models will likely find the best opinion leaders, *according to their own definition*. However, comparing the two results with each other might prove to be hard due to the different definitions.

Defining an industry standard definition of an opinion leader, as well as an accompanying evaluation method, ideally combined with a versatile ground

truth dataset would go a long way in aiding cross-over examinations of OLD methods to truly discern which method is best.

9 Conclusion

Here I first summarize the main findings and contributions of my thesis:

- Semantic information improves the performance of the ASNE model on the link prediction task, but this increase in performance does not carry over to the opinion leader detection task.
- Graph embeddings provide no clear advantage on opinion leader detection: the selected opinion leaders are not noticeably better for models using graph embeddings.
- The user's stance improves graph embeddings, but just like the other semantic information, this performance increase does not carry over to the opinion leader detection task.
- Opinion leaders fluctuated slightly over time, but that might be because of the lack of data for each user: the network visualization indicates that all opinion leaders share structural properties in the original network.

9.1 Future research

Over the years, Twitter has introduced new features on its platform such as quote Tweets. The dataset I used did not incorporate quote Tweets yet, nor was I able to find any research that did. However, these quote Tweets might contain information that could be used to better identify opinion leaders. Therefore, including quote Tweets in addition to retweets when gathering data might yield valuable new insights into the field of social network analysis.

In this thesis, I tried out two methods of text embedding: TF-IDF and FastText [76]. I observed substantial influences of both these attributes on the link prediction task. Therefore, testing out more methods of text embedding such as Predictive text embedding [79] or Spherical text embedding [80] could provide new information regarding text embedding and attribute information in social network analysis.

The attributes I included in my experiments were rather limited by both the computational resources at my disposal and by the data itself. Thus, future research could include more attribute information, to try and discover new sources of information for graph embeddings and opinion leader detection as well. Examples of such features include edge weights or demographical information that might be able to take advantage of attribute homophily.

As my experiments certainly showed, performance can vary greatly depending on the dataset. Thus, testing out existing methods on new datasets can lead to new information regarding the interactions between certain model features.

The stance attribute that I used in my experiments has some noteworthy similarities to signed network embeddings [81]. An interesting approach might be to model topic-specific edge weights in a network using this stance attribute. For instance, if both users share the same attitude towards a certain topic they share an edge with a positive edge weight, whereas if they have conflicting attitudes towards that topic the edge weight is negative instead.

References

[1] Dorothy Neufeld. The 50 most visited websites in the world. 2021.

[2] Everett M Rogers, Arvind Singhal, and Margaret M Quinlan. *Diffusion of innovations*. Routledge, 2014.

[3] Seyed Mojtaba Hosseini Bamakan, Ildar Nurgaliev, and Qiang Qu. Opinion leader detection: A methodological review. *Expert Systems with Applications*, 115:200–222, 2019.

[4] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 261–270, 2010.

[5] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[6] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[7] Xiaokai Wei, Linchuan Xu, Bokai Cao, and Philip S Yu. Cross view link prediction by learning noise-resilient representation consensus. In *Proceedings of the 26th international conference on World Wide Web*, pages 1611–1619, 2017.

[8] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[9] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web*, pages 287–297, 2016.

[10] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

[11] Melisachew Wudage Chekol and Giuseppe Pirrò. Refining node embeddings via semantic proximity. In *International Semantic Web Conference*, pages 74–91. Springer, 2020.

[12] Kenny K Chan and Shekhar Misra. Characteristics of the opinion leader: A new dimension. *Journal of advertising*, 19(3):53–60, 1990.

[13] Jiaxing Luo, Yajun Du, Ruomiao Li, and Fei Cheng. Identification of opinion leaders by using social network embedding. In *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, pages 1412–1416. IEEE, 2019.

[14] Mengjia Xu. Understanding graph embedding methods and their applications. *SIAM Review*, 63(4):825–853, 2021.

[15] Yi Chen, Xiaolong Wang, Buzhou Tang, Ruifeng Xu, Bo Yuan, Xin Xiang, and Junzhao Bu. Identifying opinion leaders from online comments. In *Chinese national conference on social media processing*, pages 231–239. Springer, 2014.

[16] Shihan Wang, Marijn Schraagen, Erik Tjong Kim Sang, and Mehdi Dastani. Dutch general public reaction on governmental covid-19 measures and announcements in twitter data. *arXiv preprint arXiv:2006.07283*, 2020.

[17] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[18] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 793–803, 2019.

[19] Asan Agibetov. Graph embeddings via matrix factorization for link prediction: smoothing or truncating negatives? *arXiv preprint arXiv:2011.09907*, 2020.

[20] Marcus Chen, Ivor W Tsang, Mingkui Tan, and Tat Jen Cham. A unified feature selection framework for graph embedding on high dimensional data. *IEEE Transactions on Knowledge and Data Engineering*, 27(6):1465–1477, 2014.

[21] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):40–51, 2006.

[22] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114, 2016.

[23] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[24] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[26] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[27] David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California ..., 1999.

[28] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[29] Basma Alharbi and Xiangliang Zhang. Learning from your network of friends: A trajectory representation learning model based on online social ties. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 781–786. IEEE, 2016.

[30] Tuan MV Le and Hady W Lauw. Probabilistic latent document network embedding. In *2014 IEEE International Conference on Data Mining*, pages 270–279. IEEE, 2014.

[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[33] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[34] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[35] Changping Wang, Chaokun Wang, Zheng Wang, Xiaojun Ye, and Philip S Yu. Edge2vec: Edge-based social network embedding. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 14(4):1–24, 2020.

[36] Na Zhao, Hanwang Zhang, Meng Wang, Richang Hong, and Tat-Seng Chua. Learning content–social influential features for influence analysis. *International Journal of Multimedia Information Retrieval*, 5(3):137–149, 2016.

[37] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4274–4282, 2015.

[38] Lise Getoor and Christopher P Diehl. Link mining: a survey. *Acm Sigkdd Explorations Newsletter*, 7(2):3–12, 2005.

[39] Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 487–494, 2007.

[40] Jiliang Tang, Charu Aggarwal, and Huan Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM international conference on data mining*, pages 54–62. SIAM, 2016.

[41] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[42] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, 2015.

[43] Myunghwan Kim and Jure Leskovec. The network completion problem: Inferring missing nodes and edges in networks. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 47–58. SIAM, 2011.

[44] David J Marchette and Carey E Priebe. Predicting unobserved links in incompletely observed networks. *Computational Statistics & Data Analysis*, 52(3):1373–1386, 2008.

[45] Luca Maria Aiello, Alain Barrat, Rossano Schifanella, Ciro Cattuto, Benjamin Markines, and Filippo Menczer. Friendship prediction and homophily in social media. *ACM Transactions on the Web (TWEB)*, 6(2):1–33, 2012.

[46] Milen Pavlov and Ryutaro Ichise. Finding experts by link prediction in co-authorship networks. *FEWS*, 290:42–55, 2007.

[47] Cuneyt Gurcan Akcora, Barbara Carminati, and Elena Ferrari. Network and profile based measures for user similarities on social networks. In *2011 IEEE International Conference on Information Reuse & Integration*, pages 292–298. IEEE, 2011.

[48] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015.

[49] Debra S Goldberg and Frederick P Roth. Assessing experimentally derived interactions in a small world. *Proceedings of the National Academy of Sciences*, 100(8):4372–4376, 2003.

[50] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42, 2009.

[51] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644, 2011.

[52] Yang Yang, Nitesh Chawla, Yizhou Sun, and Jiawei Hani. Predicting links in multi-relational and heterogeneous networks. In *2012 IEEE 12th international conference on data mining*, pages 755–764. IEEE, 2012.

[53] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, 2008.

[54] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[55] Xiaodan Song, Yun Chi, Koji Hino, and Belle Tseng. Identifying opinion leaders in the blogosphere. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 971–974, 2007.

[56] Linyuan Lü, Yi-Cheng Zhang, Chi Ho Yeung, and Tao Zhou. Leaders in social networks, the delicious case. *PloS one*, 6(6):e21202, 2011.

[57] Jiangjiao Duan, Jianping Zeng, and Banghui Luo. Identification of opinion leaders based on user clustering and sentiment analysis. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 377–383. IEEE, 2014.

[58] Hongpeng Cao, Jun Wang, and Zhe Wang. Opinion leaders discovery in social networking site based on the theory of propagation probability. In *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 700–704. IEEE, 2018.

[59] Yi-Cheng Chen, Ju-Ying Cheng, and Hui-Huang Hsu. A cluster-based opinion leader discovery in social network. In *2016 conference on technologies and applications of artificial intelligence (TAAI)*, pages 78–83. IEEE, 2016.

[60] Yi-Cheng Chen, Wen-Yuan Zhu, Wen-Chih Peng, Wang-Chien Lee, and Suh-Yin Lee. Cim: community-based influence maximization in social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(2):1–31, 2014.

[61] Katrin Jungnickel. New methods of measuring opinion leadership: a systematic, interdisciplinary literature analysis. *International Journal of Communication*, 12:23, 2018.

[62] Baocheng Huang, Guang Yu, and Hamid Reza Karimi. The finding and dynamic detection of opinion leaders in social network. *Mathematical problems in engineering*, 2014, 2014.

[63] Kaisong Song, Daling Wang, Shi Feng, and Ge Yu. Detecting opinion leader dynamically in chinese news comments. In *International conference on web-age information management*, pages 197–209. Springer, 2011.

[64] Xiao Yu, Xu Wei, and Xia Lin. Algorithms of bbs opinion leader mining based on sentiment analysis. In *International Conference on Web Information Systems and Mining*, pages 360–369. Springer, 2010.

[65] Chun Wang, Ya Jun Du, and Ming Wei Tang. Opinion leader mining algorithm in microblog platform based on topic similarity. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pages 160–165. IEEE, 2016.

[66] William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the royal society of london. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.

[67] Robert M May and Alun L Lloyd. Infection dynamics on scale-free networks. *Physical Review E*, 64(6):066112, 2001.

[68] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[69] Reza Motamedi, Soheil Jamshidi, Reza Rejaie, and Walter Willinger. Examining the evolution of the twitter elite network. *Social Network Analysis and Mining*, 10(1):1–18, 2020.

[70] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.

[71] Julian J McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *NIPS*, volume 2012, pages 548–56. Citeseer, 2012.

[72] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Oxford University Press, 2017.

[73] Tomasz Szandała. Review and comparison of commonly used activation functions for deep neural networks. In *Bio-inspired Neurocomputing*, pages 203–224. Springer, 2021.

[74] Magnus Sahlgren. The distributional hypothesis. *Italian Journal of Disability Studies*, 20:33–53, 2008.

[75] Robert Endre Tarjan. Updating a balanced search tree in o (1) rotations. *Information Processing Letters*, 16(5):253–257, 1983.

[76] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[77] Tom C Freeman, Sebastian Horsewell, Anirudh Patir, Josh Harling-Lee, Tim Regan, Barbara B Shih, James Prendergast, David A Hume, and Tim Angus. Graphia: A platform for the graph-based visualisation and analysis of complex data. *bioRxiv*, 2020.

[78] Hans Rosenberg, Shahbaz Syed, and Salim Rezaie. The twitter pandemic: The critical role of twitter in the dissemination of medical information and misinformation during the covid-19 pandemic. *Canadian journal of emergency medicine*, 22(4):418–421, 2020.

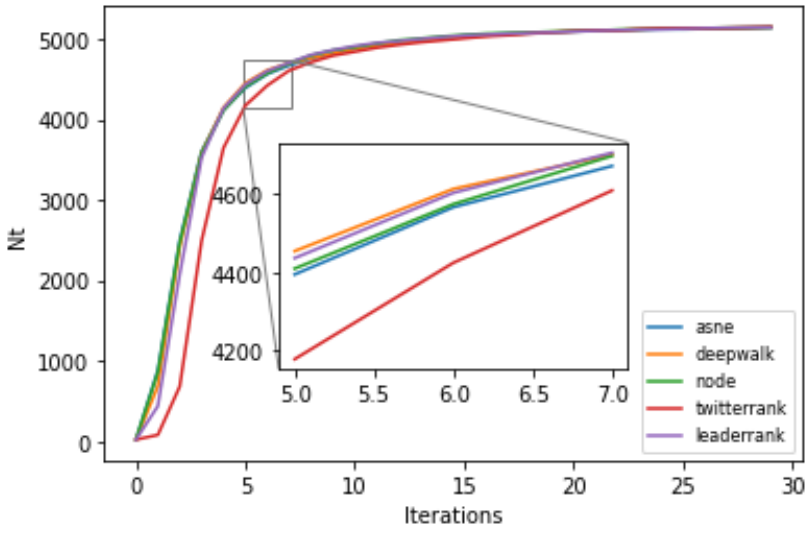
[79] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1165–1174, 2015.

[80] Yu Meng, Jiaxin Huang, Guangyuan Wang, Chao Zhang, Honglei Zhuang, Lance Kaplan, and Jiawei Han. Spherical text embedding. *Advances in Neural Information Processing Systems*, 32:8208–8217, 2019.

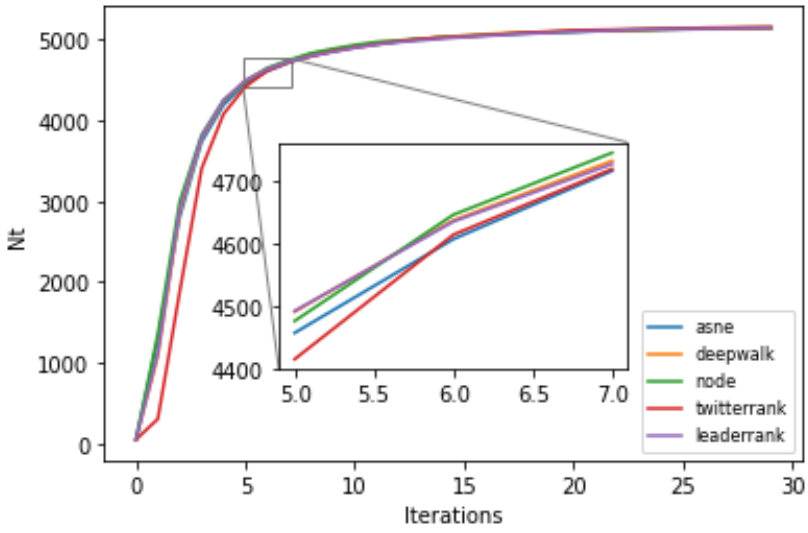
[81] Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. Signed network embedding in social media. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 327–335. SIAM, 2017.

Appendices

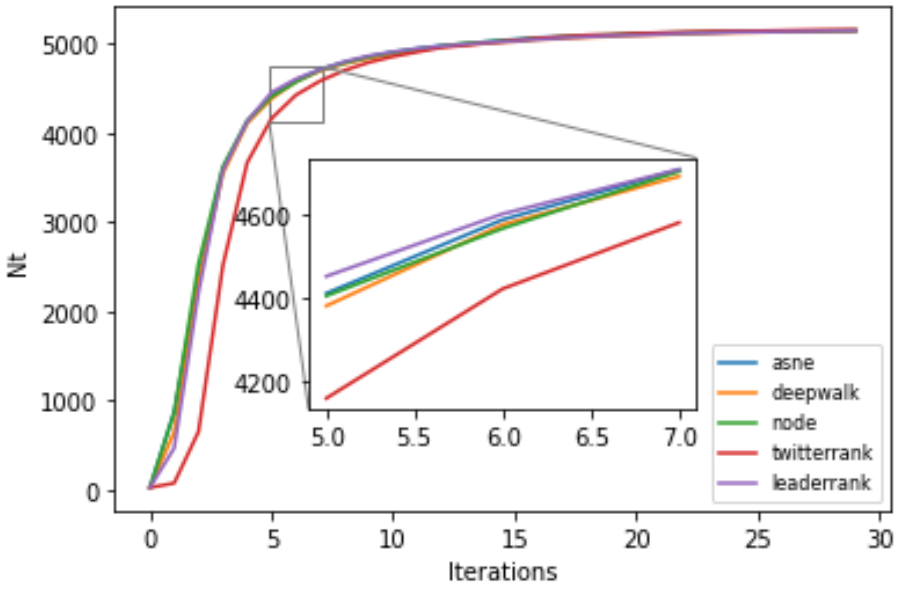
A SIR Hyperparameter results



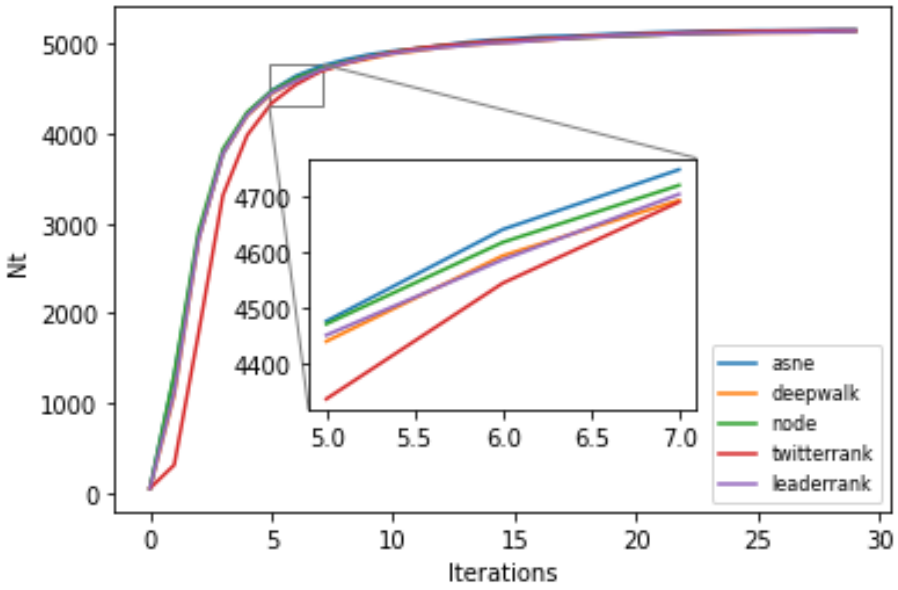
(a) λ_1: 0.1, λ_2: 0.1, ϵ: 0.2, N_0: 20



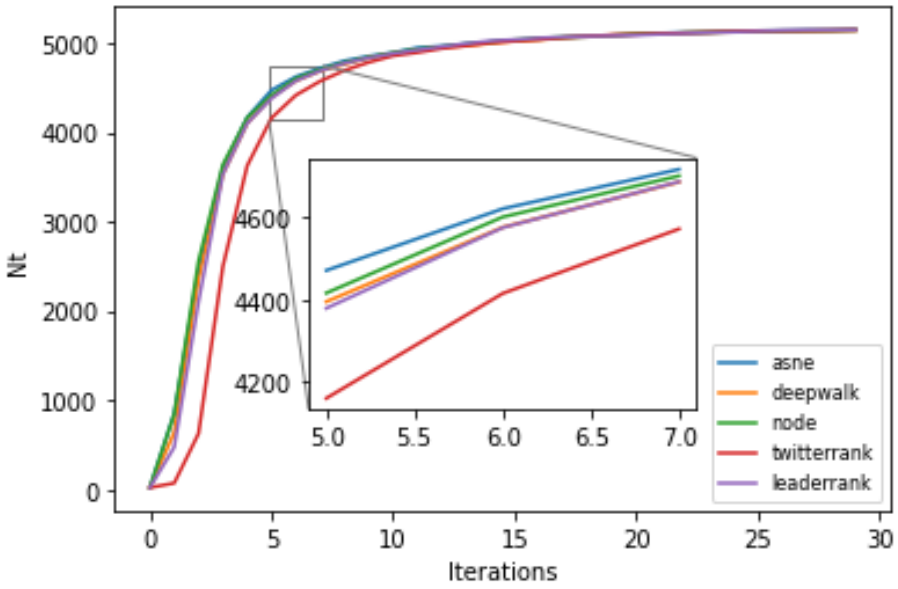
(b) λ_1: 0.1, λ_2: 0.1, ϵ: 0.2, N_0: 50



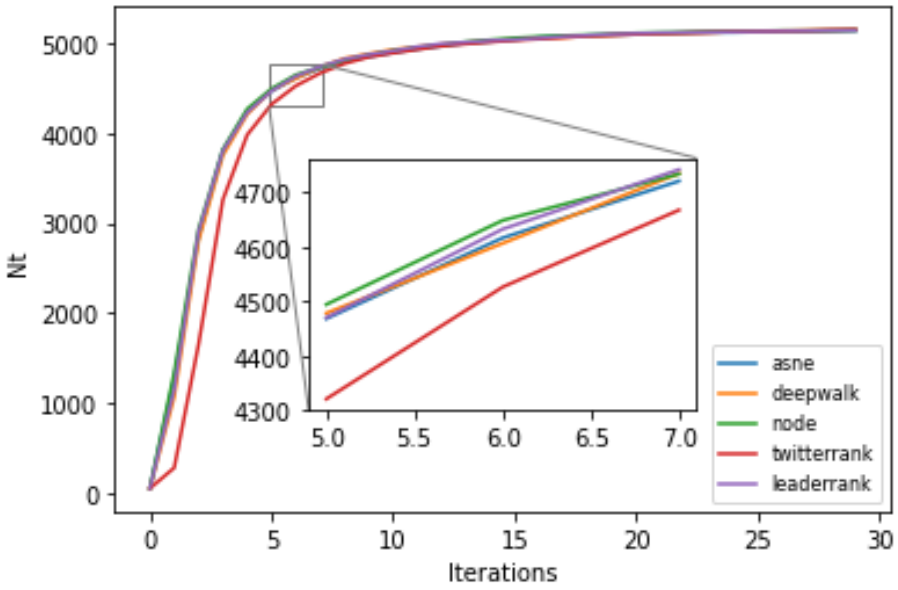
(c) λ_1: 0.1, λ_2: 0.1, ϵ: 0.4, N_0: 20



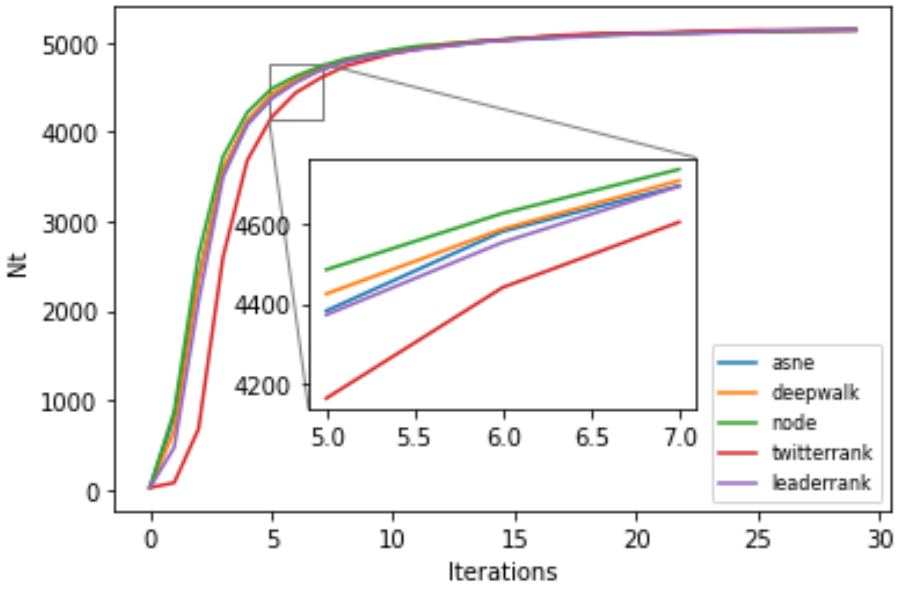
(d) λ_1: 0.1, λ_2: 0.1, ϵ: 0.4, N_0: 50



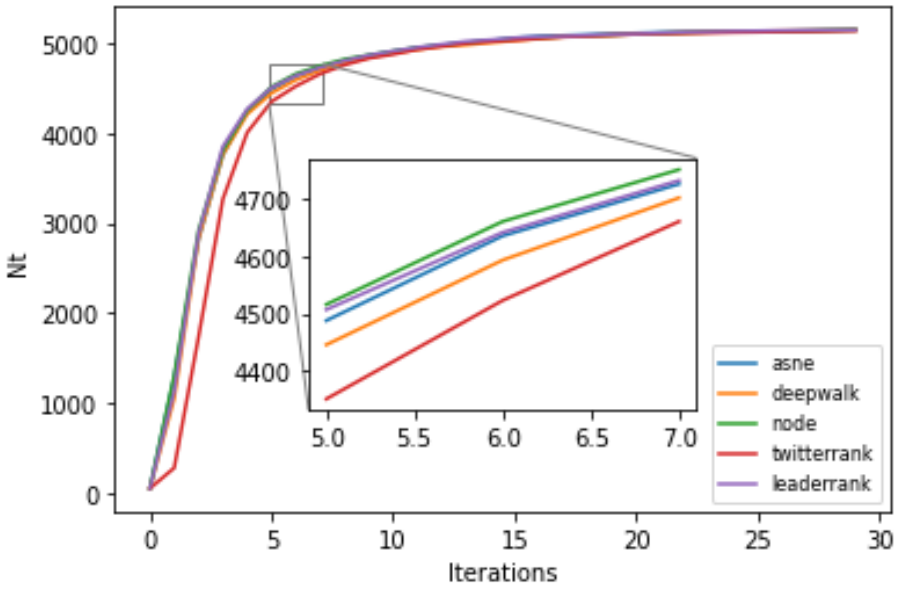
(e) λ_1: 0.1, λ_2: 0.3, ϵ: 0.2, N_0: 20



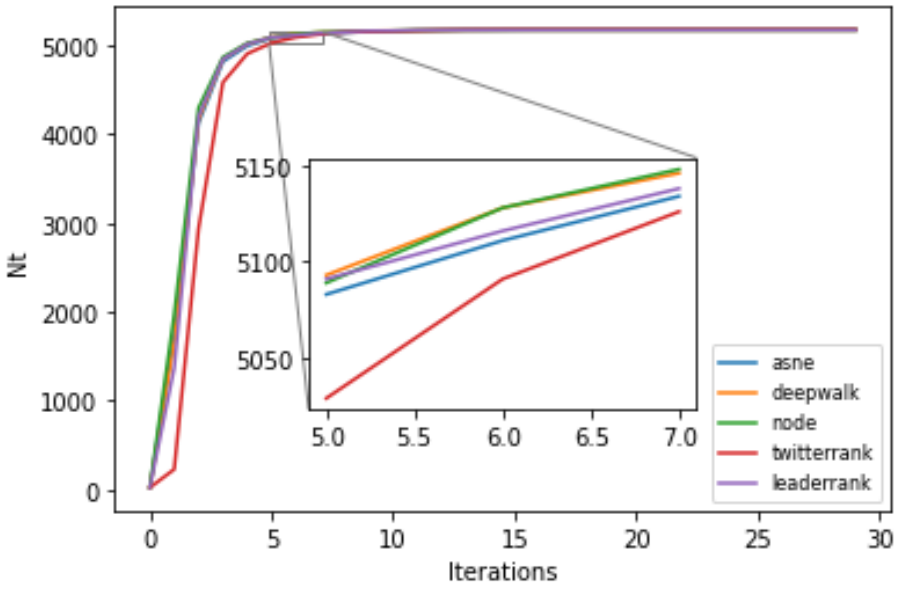
(f) λ_1: 0.1, λ_2: 0.3, ϵ: 0.2, N_0: 50



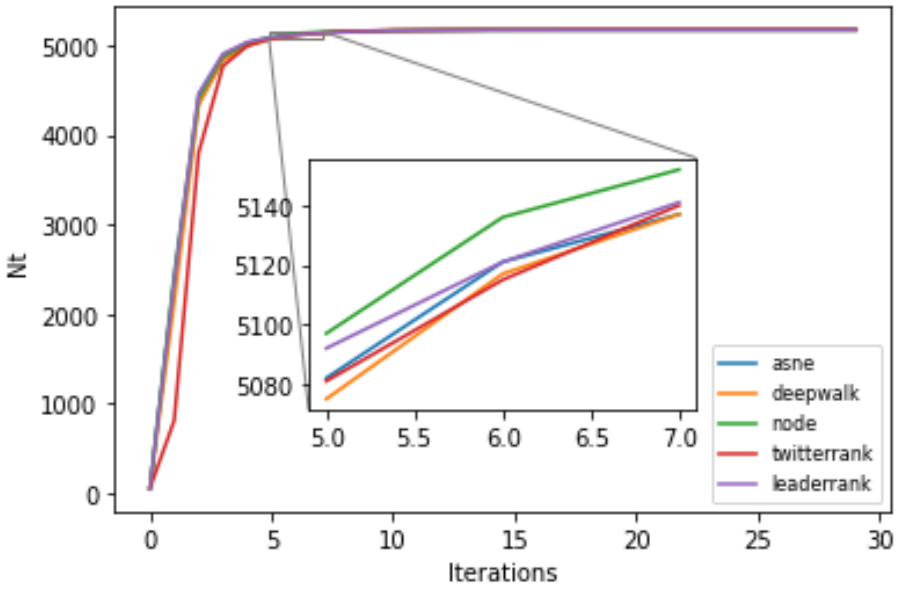
(g) λ_1: 0.1, λ_2: 0.3, ϵ: 0.4, N_0: 20



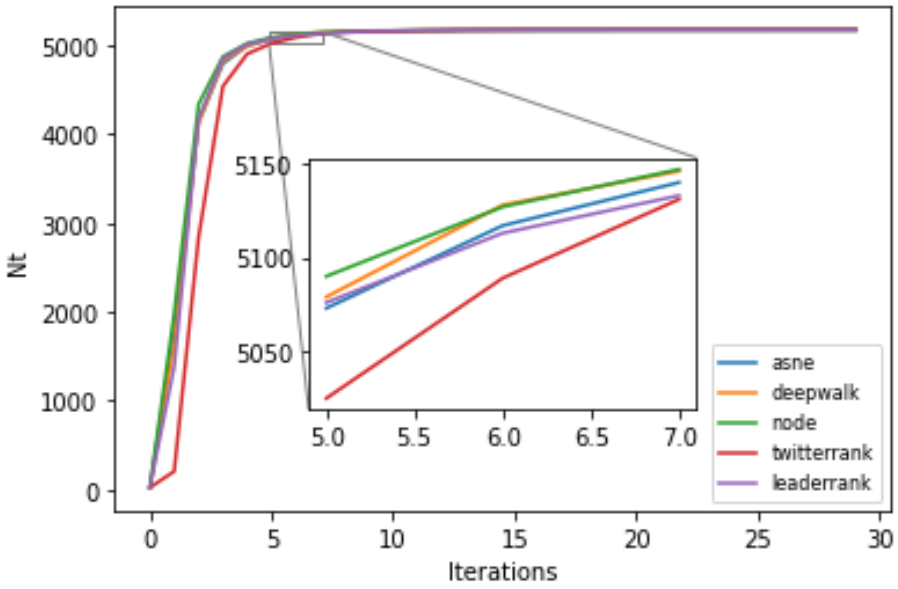
(h) λ_1: 0.1, λ_2: 0.3, ϵ: 0.4, N_0: 50



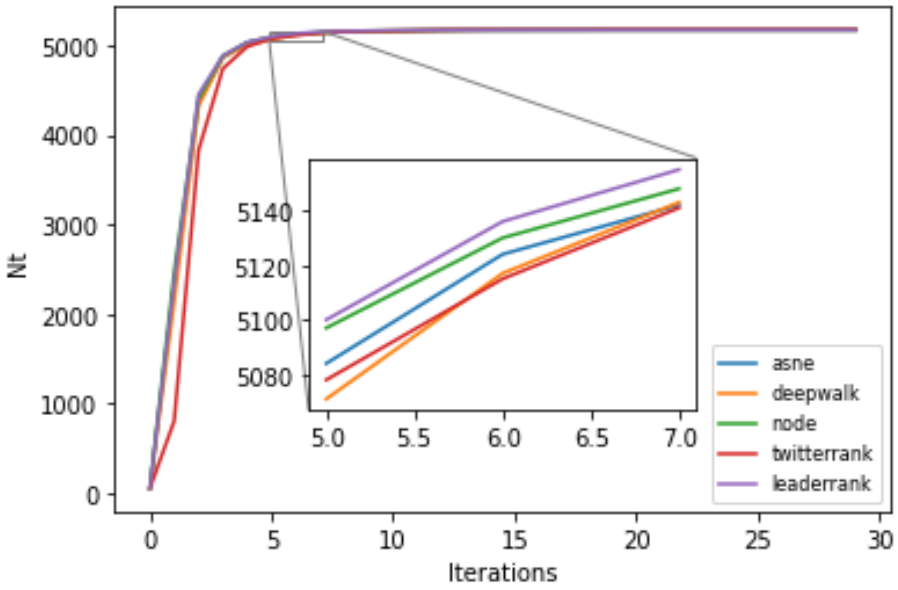
(i) λ_1: 0.3, λ_2: 0.1, ϵ: 0.2, N_0: 20



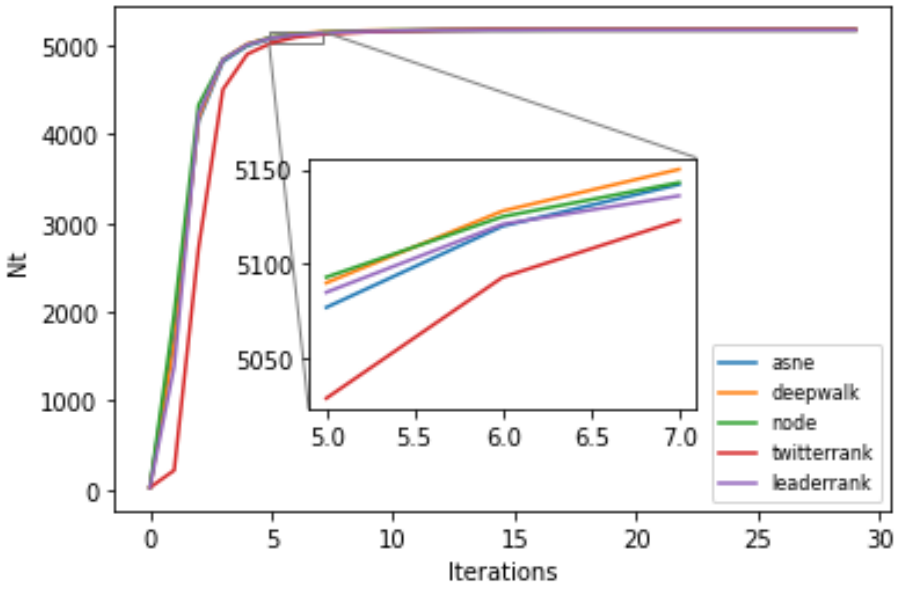
(j) λ_1: 0.3, λ_2: 0.1, ϵ: 0.2, N_0: 50



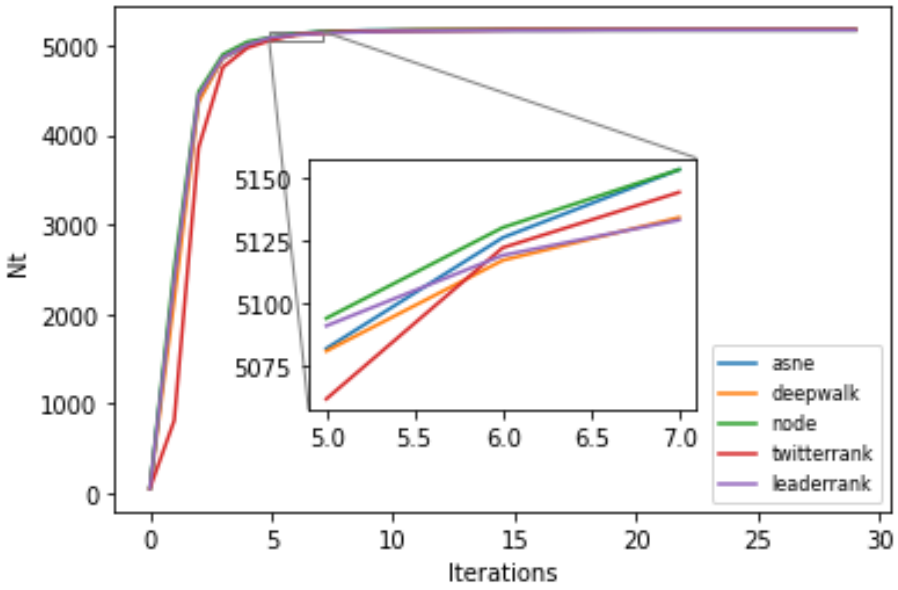
(k) λ_1: 0.3, λ_2: 0.1, ϵ: 0.4, N_0: 20



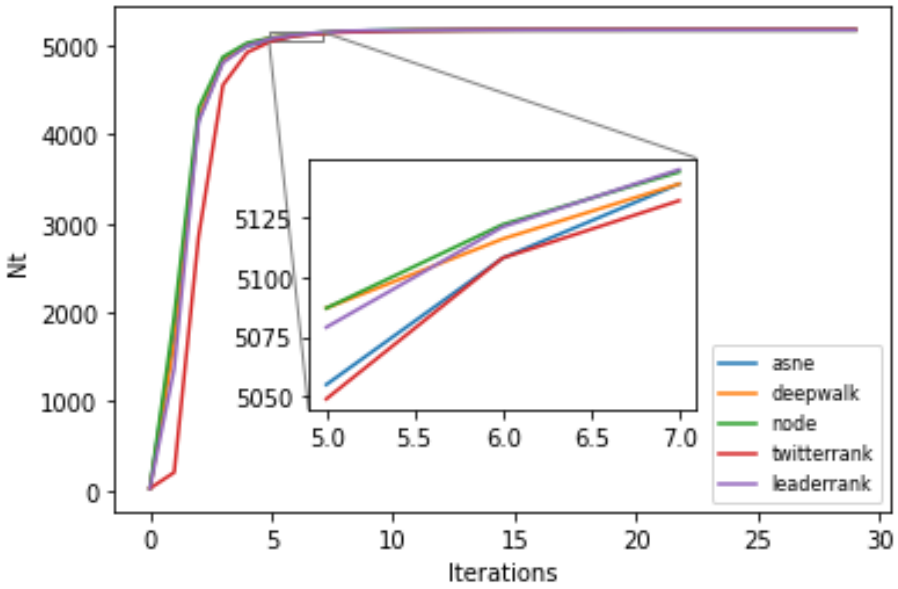
(l) λ_1: 0.3, λ_2: 0.1, ϵ: 0.4, N_0: 50



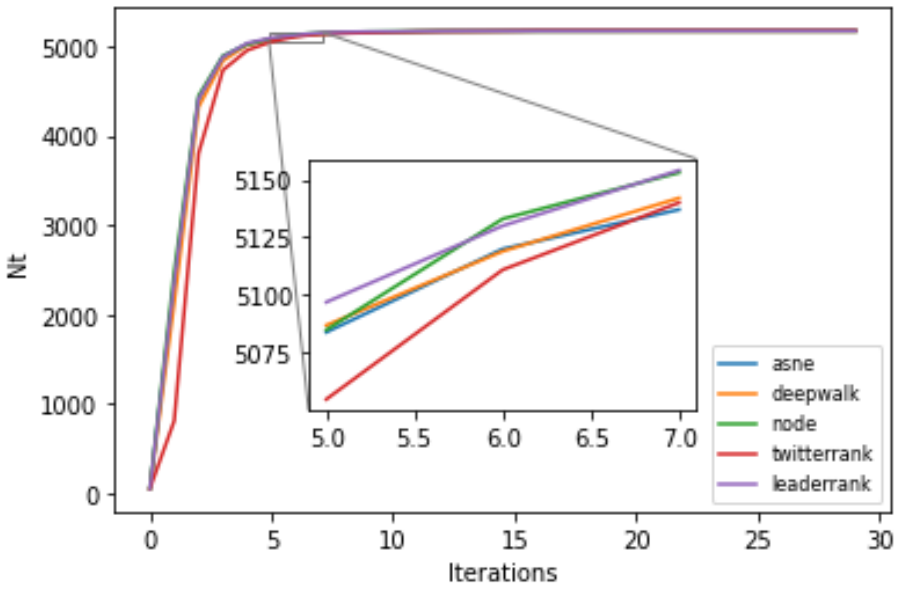
(m) λ_1: 0.3, λ_2: 0.3, ϵ: 0.2, N_0: 20



(n) λ_1: 0.3, λ_2: 0.3, ϵ: 0.2, N_0: 50



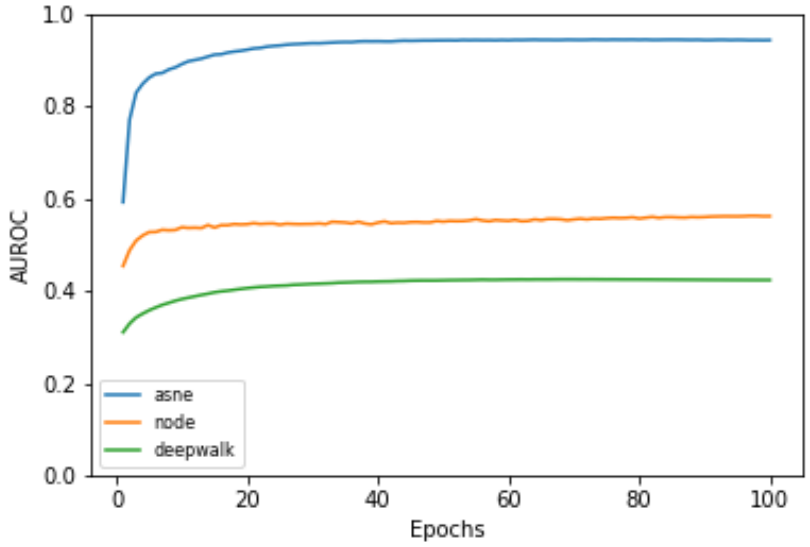
(o) λ_1: 0.3, λ_2: 0.3, ϵ: 0.4, N_0: 20



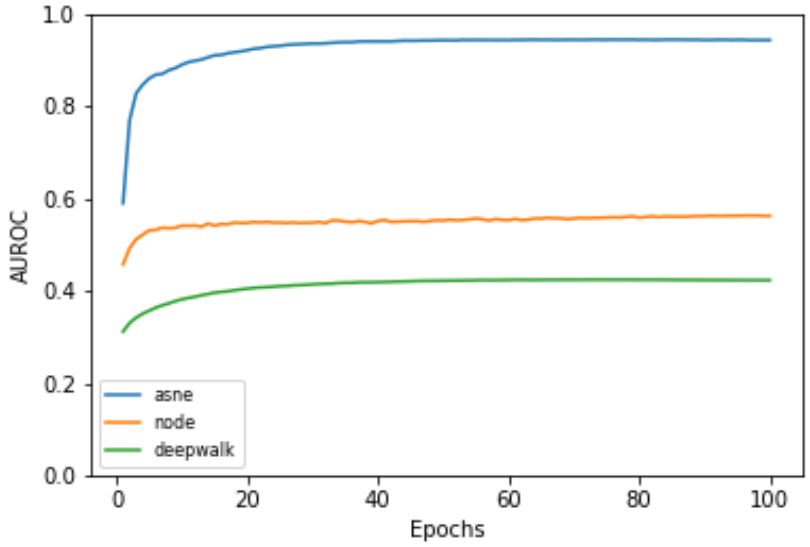
(p) λ_1: 0.3, λ_2: 0.3, ϵ: 0.4, N_0: 50

Figure 16: Complete results of testing the different combinations of hyperparameter values from Table 6 on week 1 of the data. N_0 is the number of initial infected nodes before iteration starts. In the legend, "asne" corresponds to the ASNE embedding model + SNERank, "deepwalk" is DeepWalk + SNERank, "node" is node2vec + SNERank.

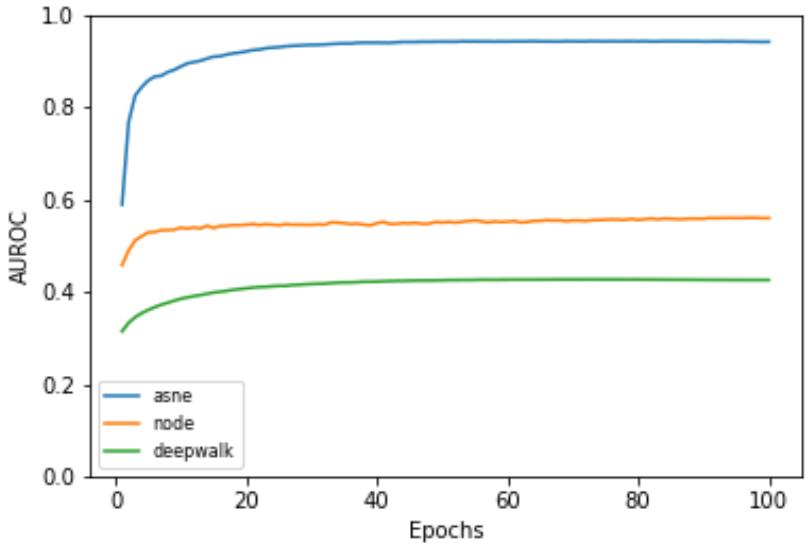
B Ablation results



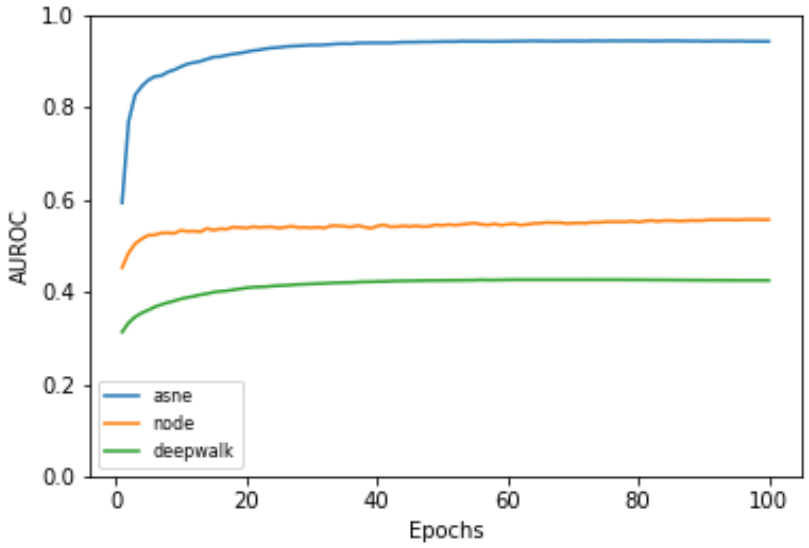
(a)



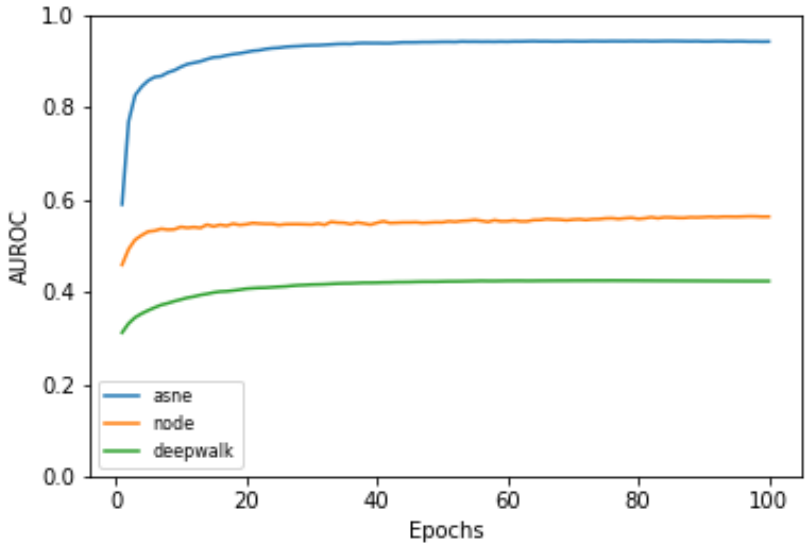
(b)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
0
20
40
60
80
100
Epochs
asne
node
deepwalk

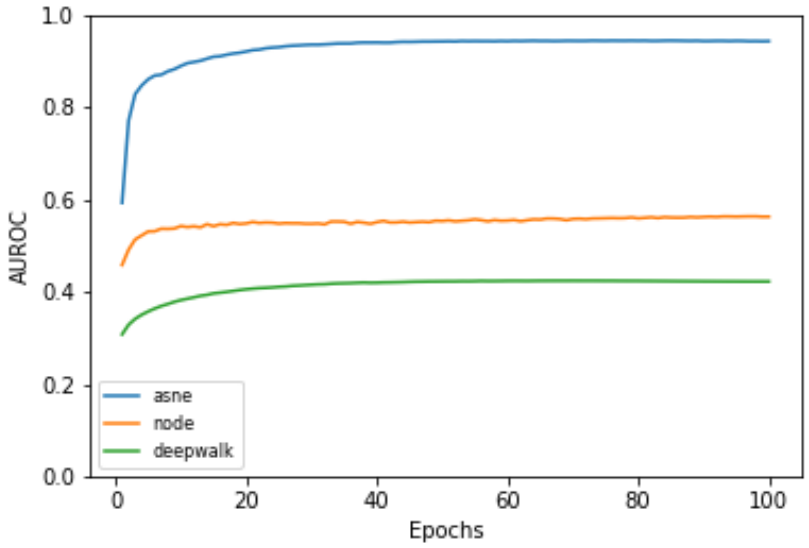
(c)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
0
20
40
60
80
100
Epochs
asne
node
deepwalk

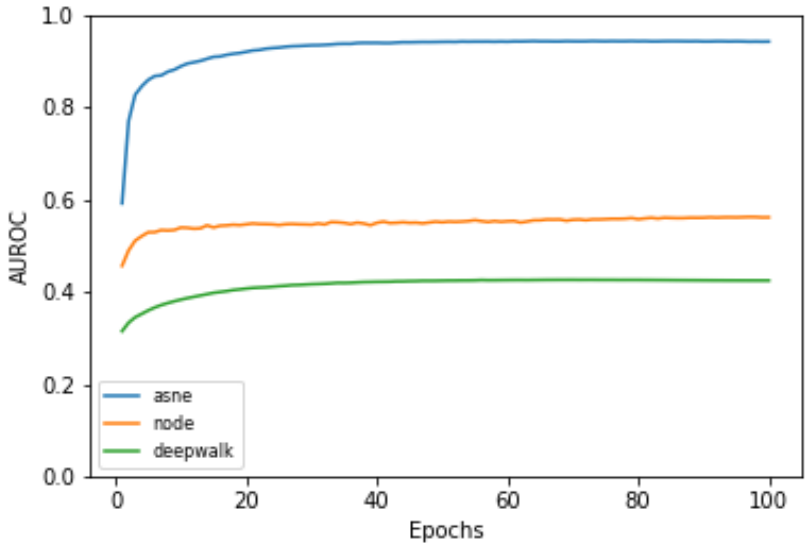
(d)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
asne
node
deepwalk
0
20
40
60
80
100
Epochs

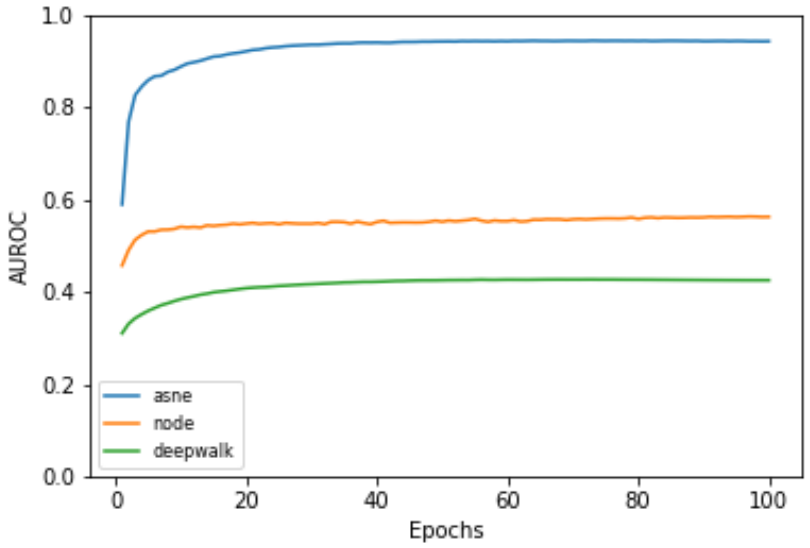
(e)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
asne
node
deepwalk
0
20
40
60
80
100
Epochs

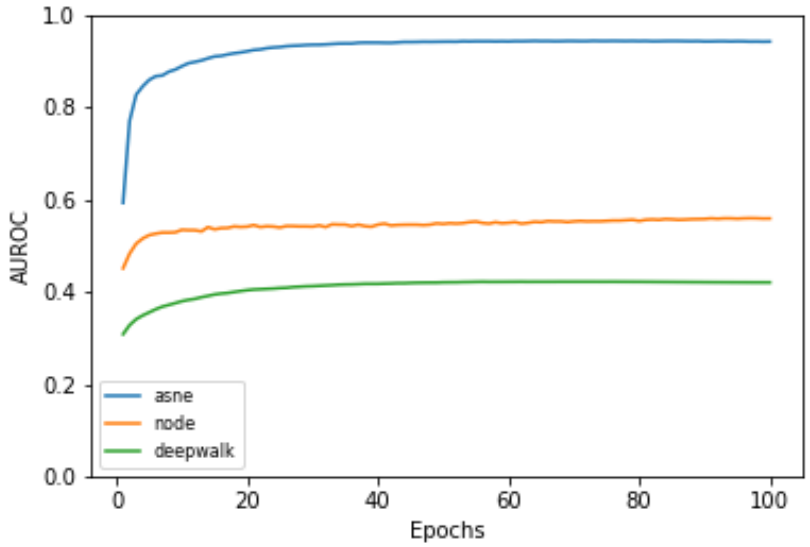
(f)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
asne
node
deepwalk
0
20
40
60
80
100
Epochs

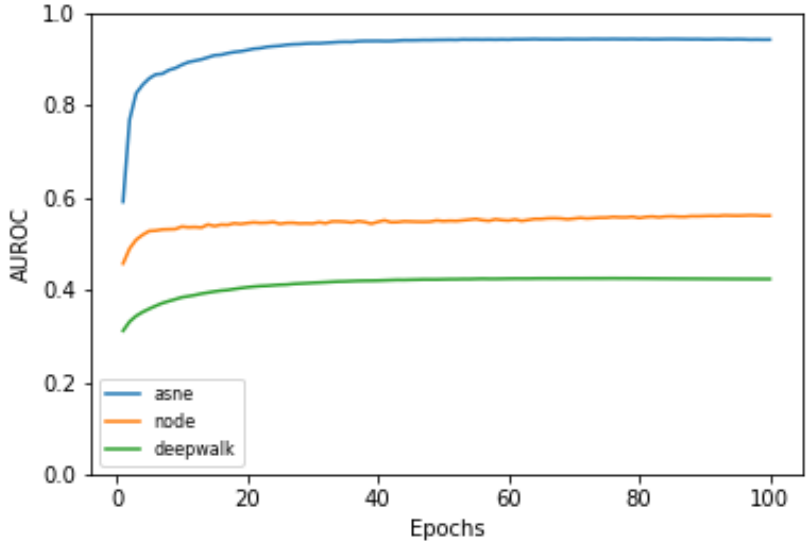
(g)


1.0
0.8
0.6
0.4
0.2
0.0
AUROC
asne
node
deepwalk
0
20
40
60
80
100
Epochs

(h)



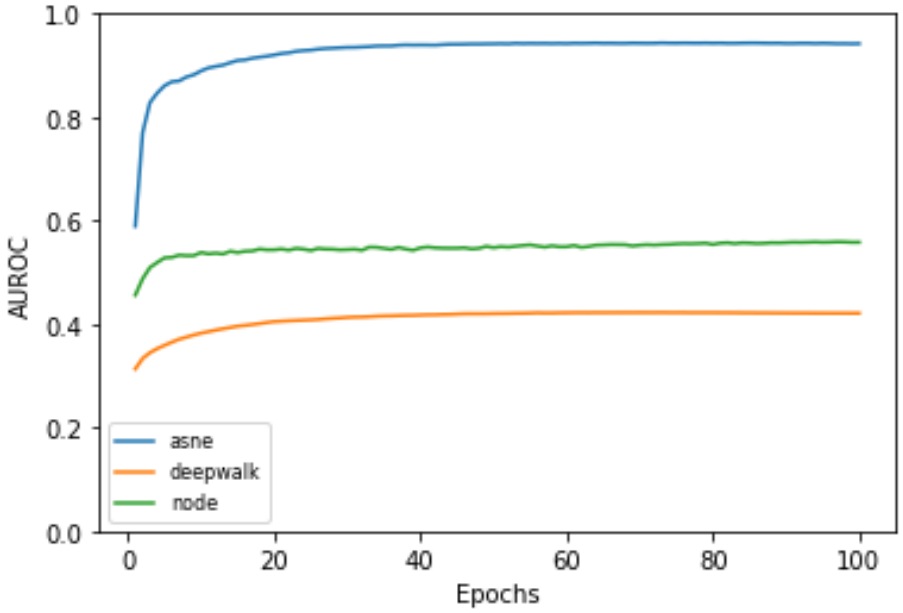
(i)



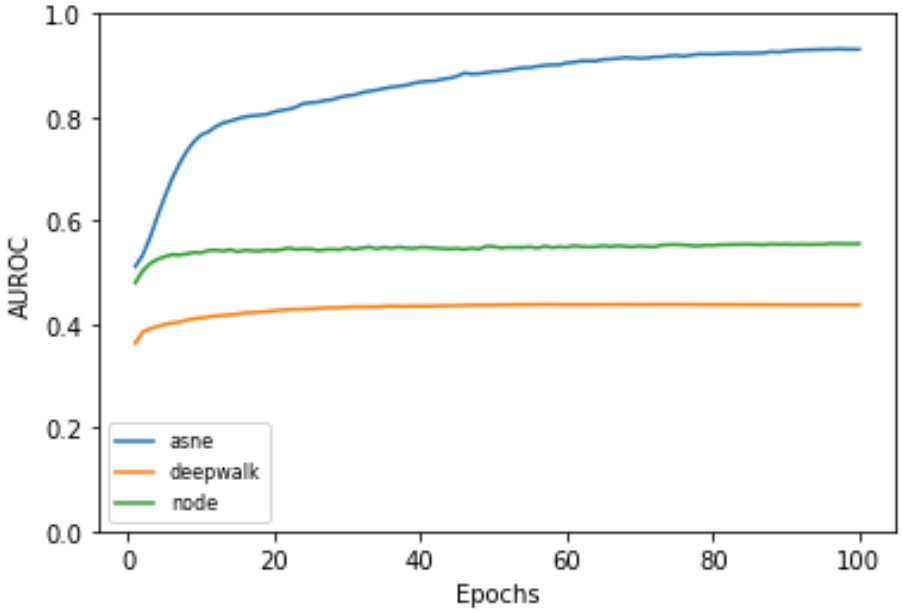
(j)

Figure 17: Link prediction results on week 1 of the data for the ASNE, DeepWalk and node2vec graph embedding models. Each image corresponds to a different randomly generated test set.

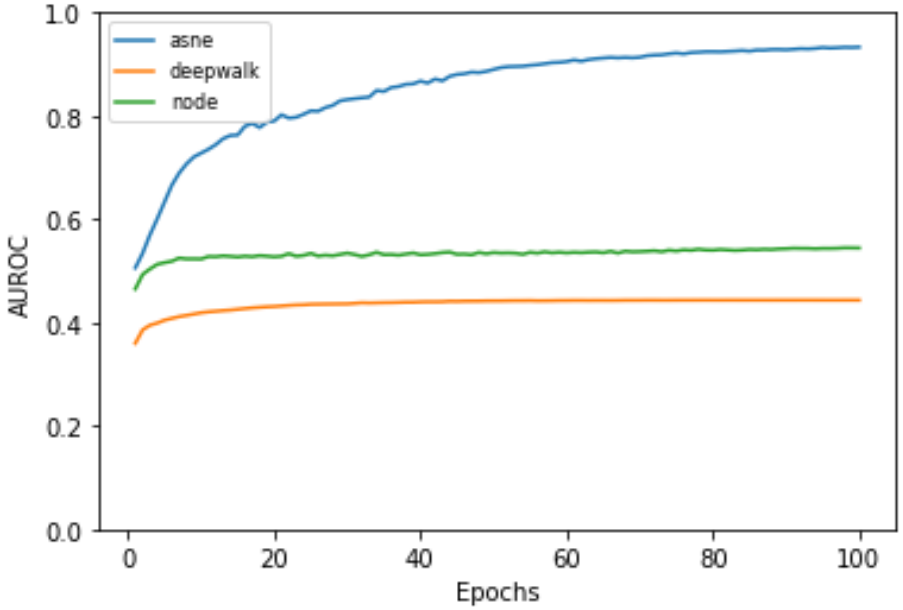
C Link prediction results



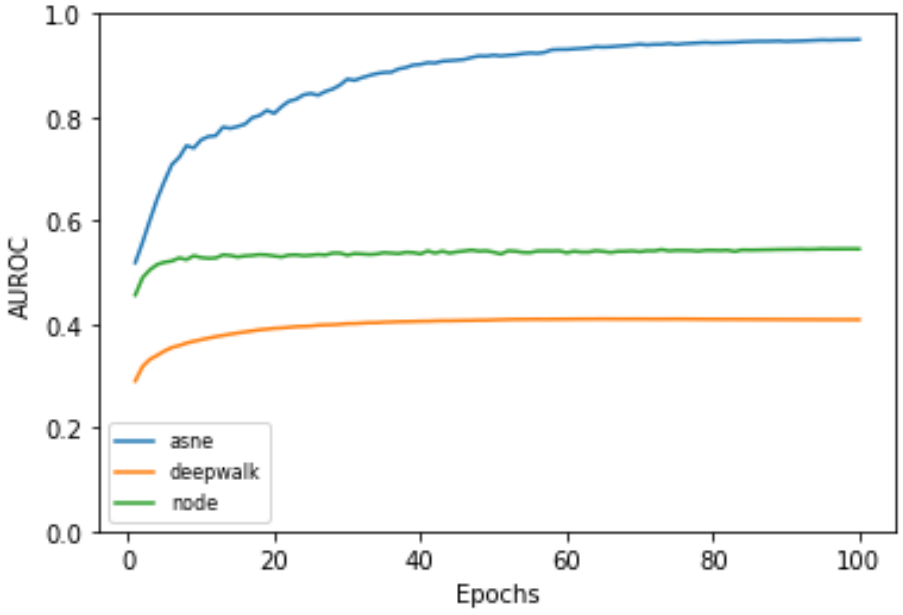
(a) Week 1



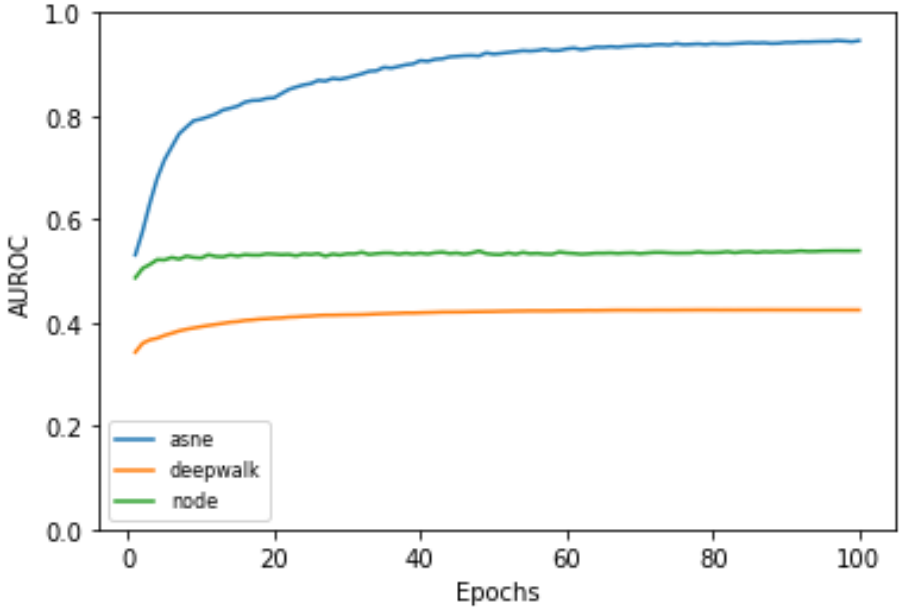
(b) Week 2, part 1



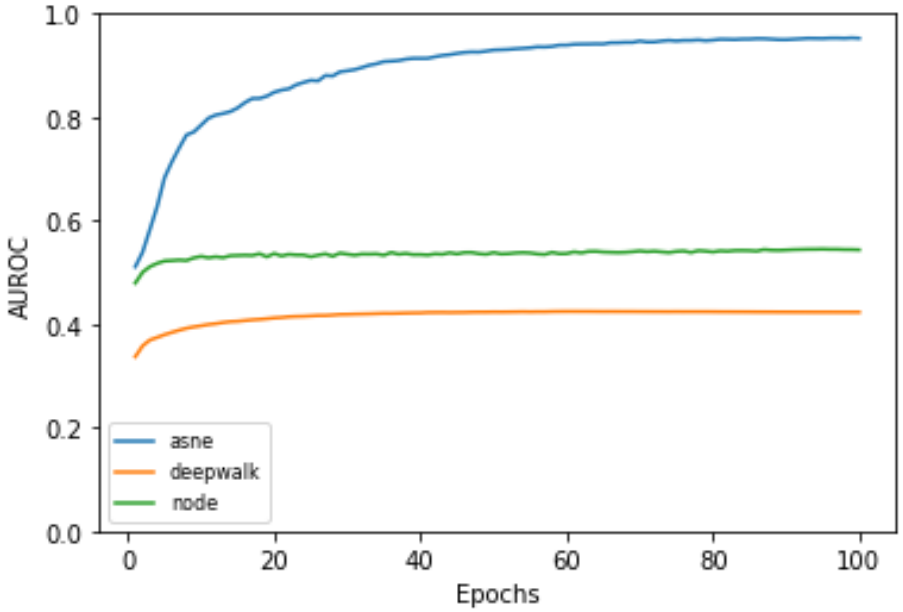
(c) Week 2, part 2



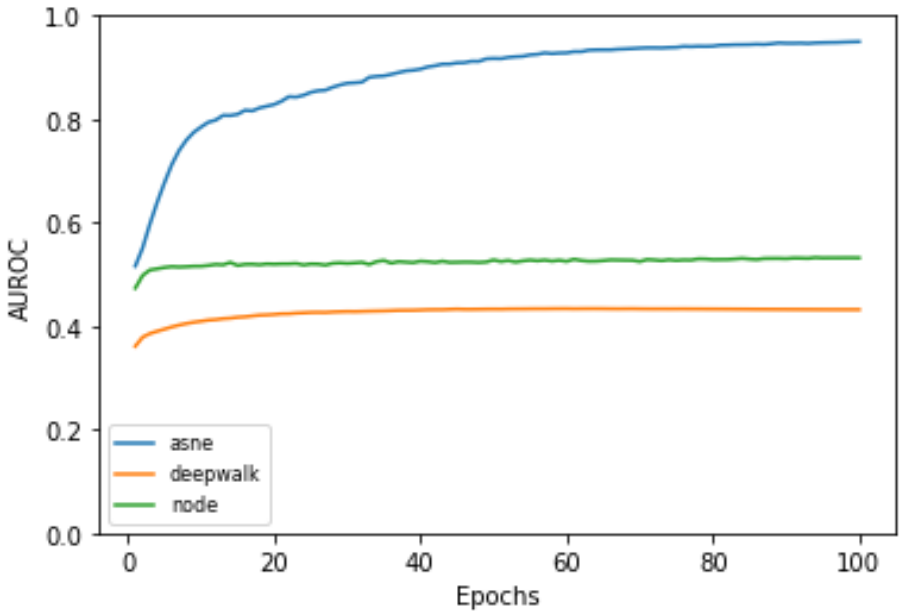
(d) Week 3, part 1



(e) Week 3, part 2



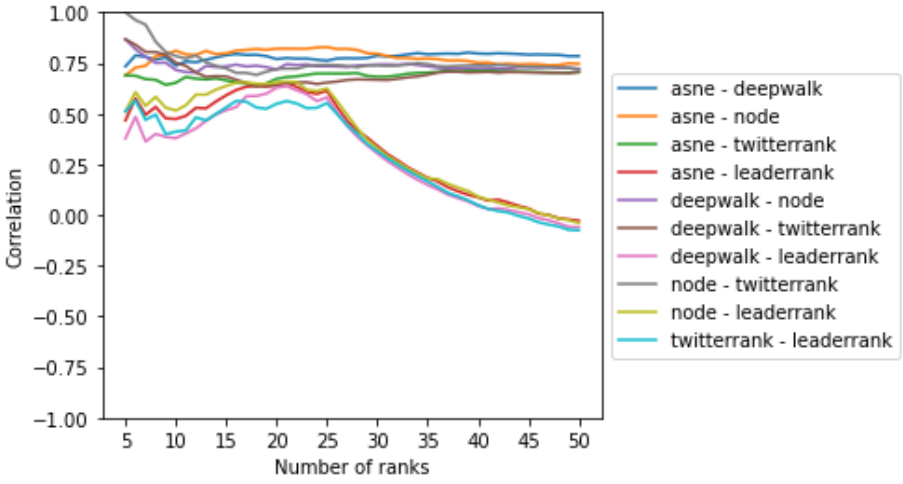
(f) Week 4



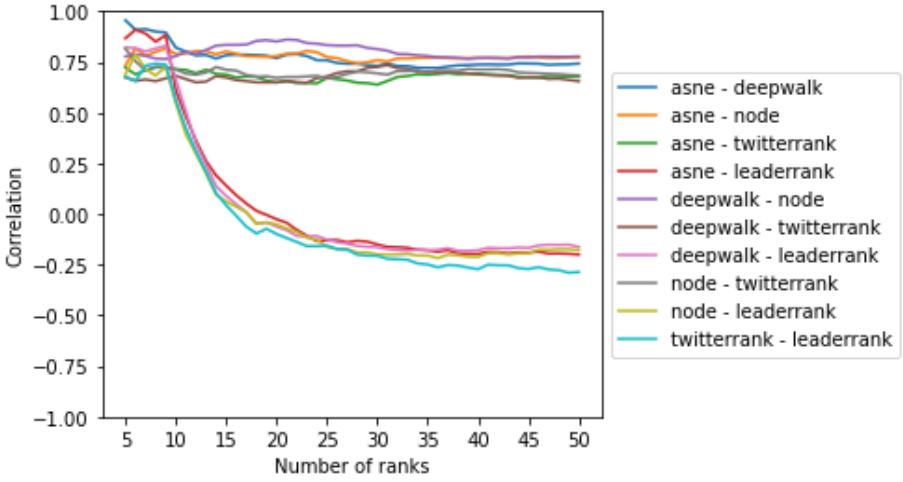
(g) Week 5

Figure 18: Complete results of the Link Prediction task on each week of the data for the ASNE, DeepWalk, and node2vec graph embedding models.

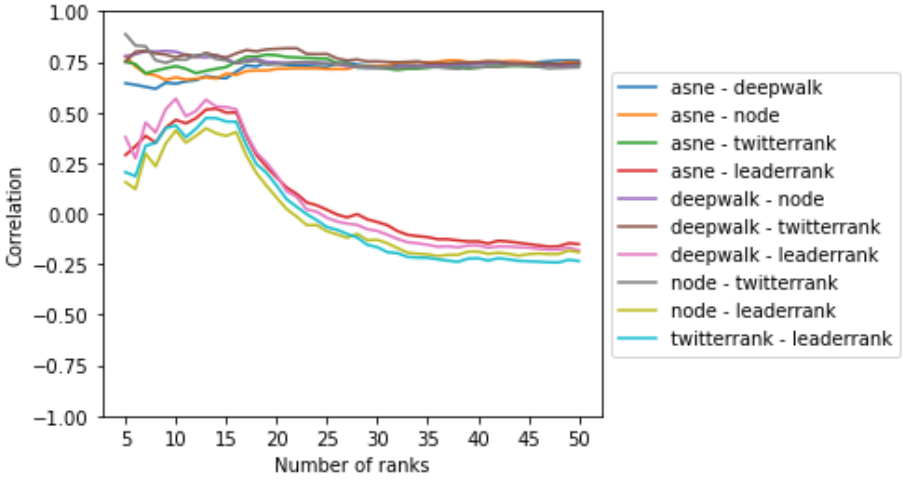
D Kendall's τ results



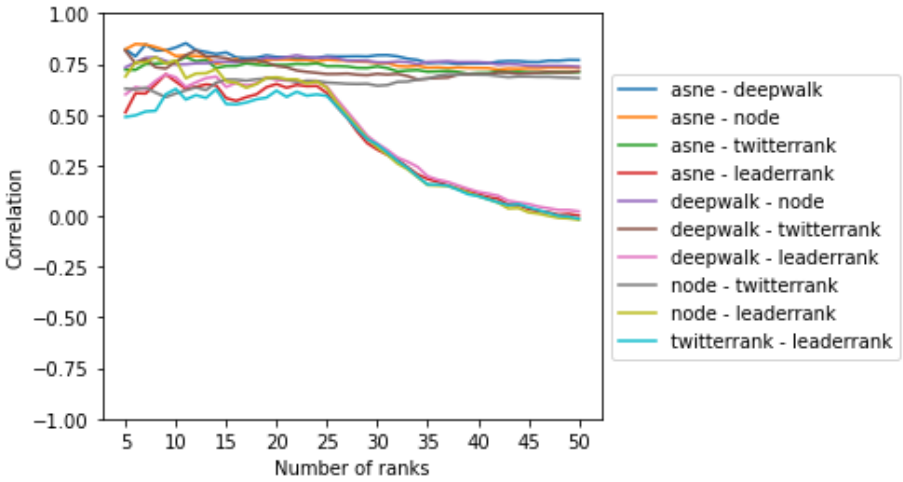
(a) Week 1



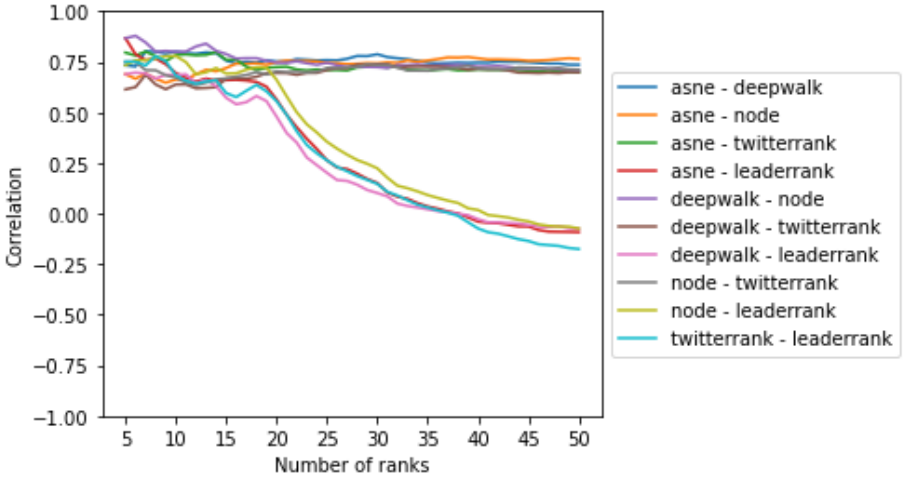
(b) Week 2, part 1



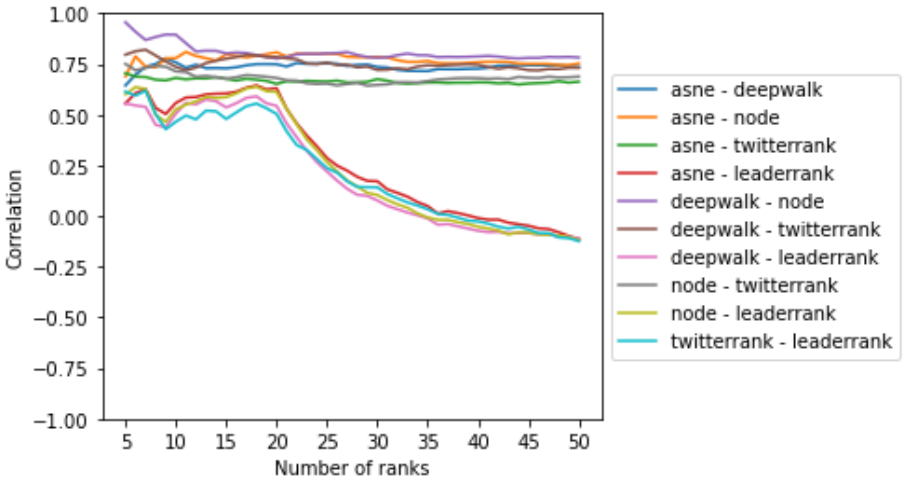
(c) Week 2, part 2



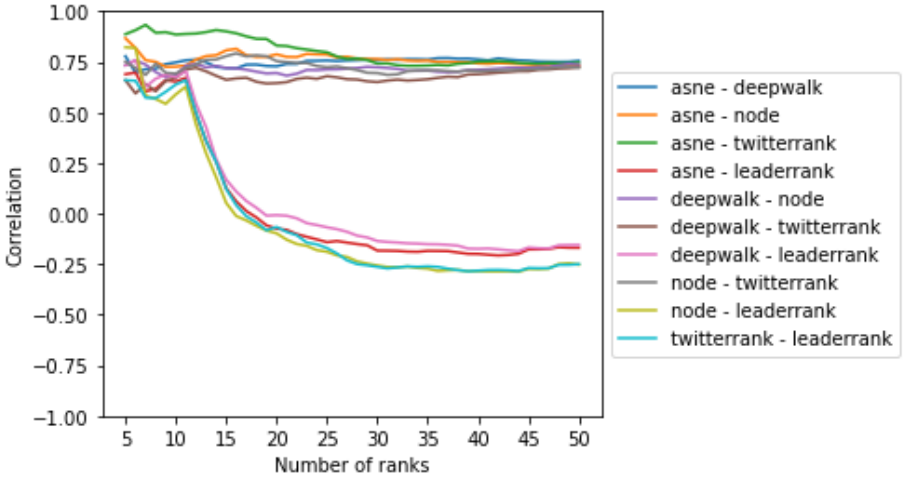
(d) Week 3, part 1



(e) Week 3, part 2



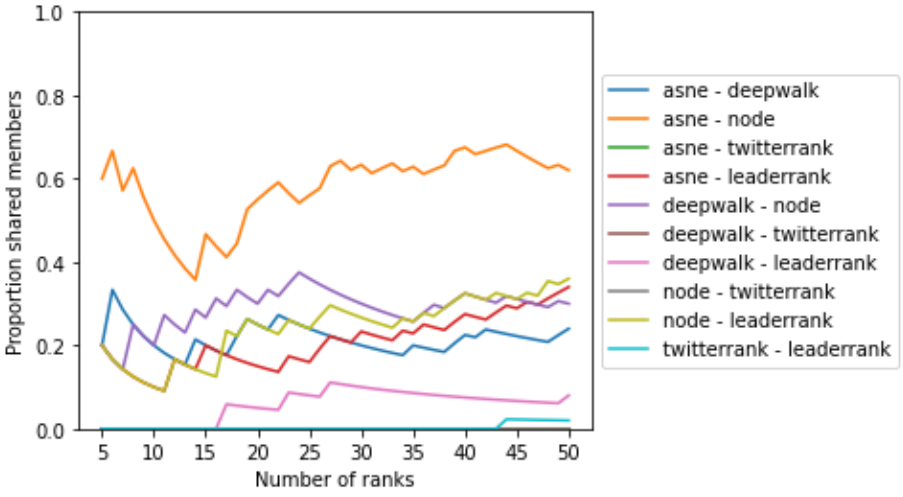
(f) Week 4



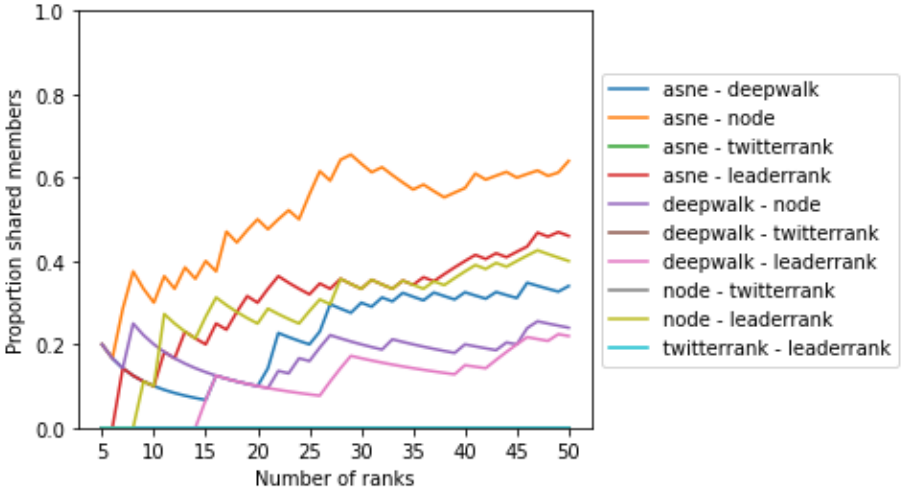
(g) Week 5

Figure 19: Pairwise correlations between ranking lists. *Number of ranks* corresponds to the top k ranks in the ranking lists.

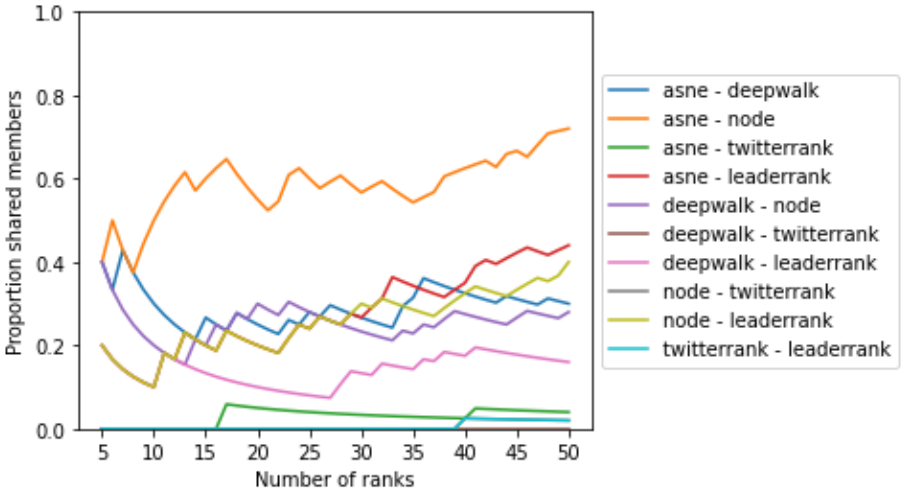
E Shared opinion leaders results



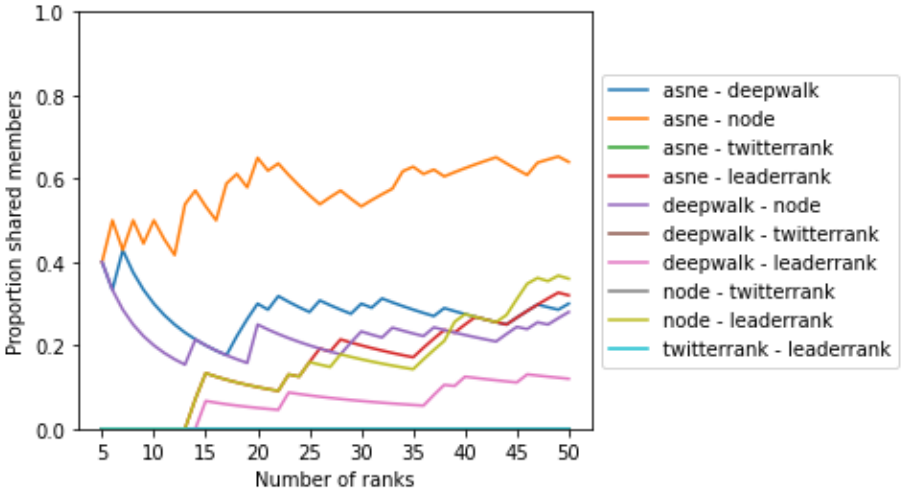
(a) Week 1



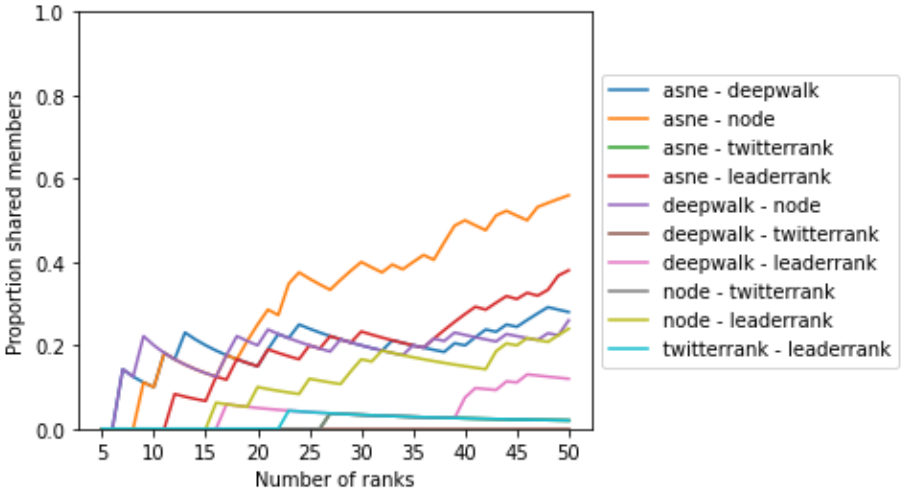
(b) Week 2, part 1



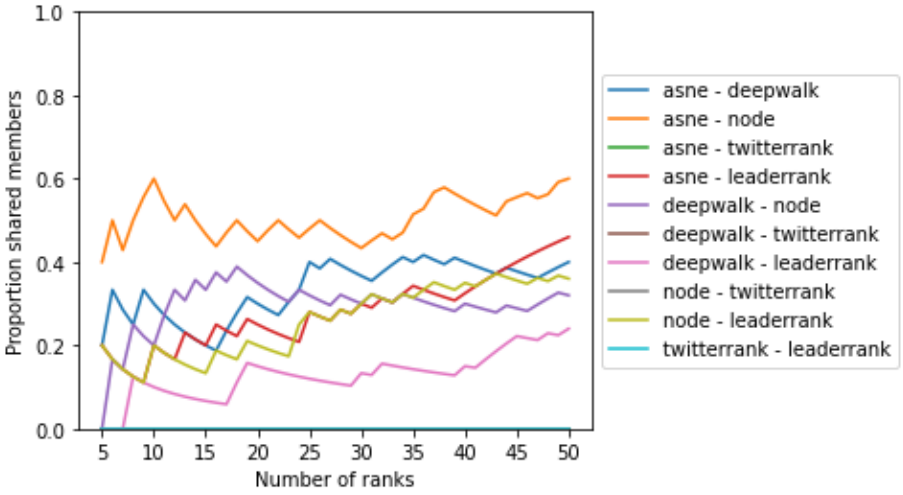
(c) Week 2, part 2



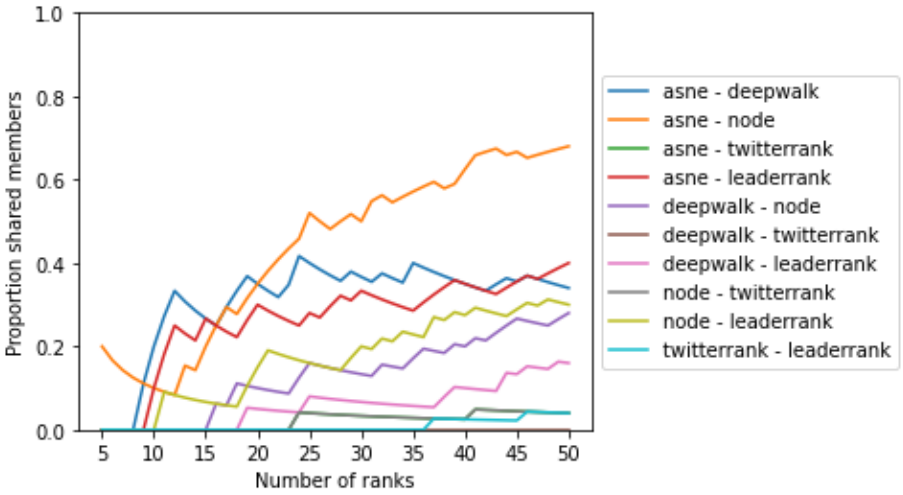
(d) Week 3, part 1



(e) Week 3, part 2



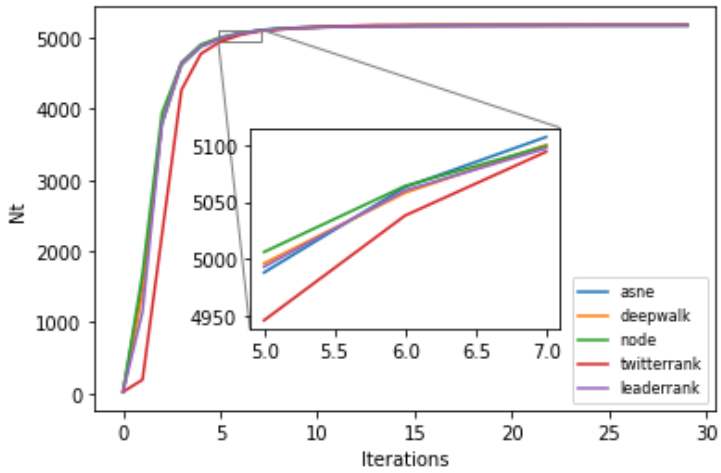
(f) Week 4



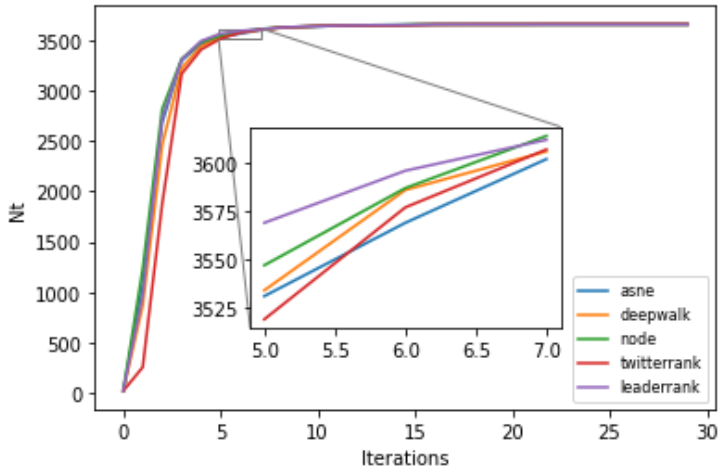
(g) Week 5

Figure 20: Proportion of shared opinion leaders between ranking lists. OLs are shared between ranking lists when they occur in both lists, regardless of position. *Number of ranks* corresponds to the top k ranks in the ranking lists.

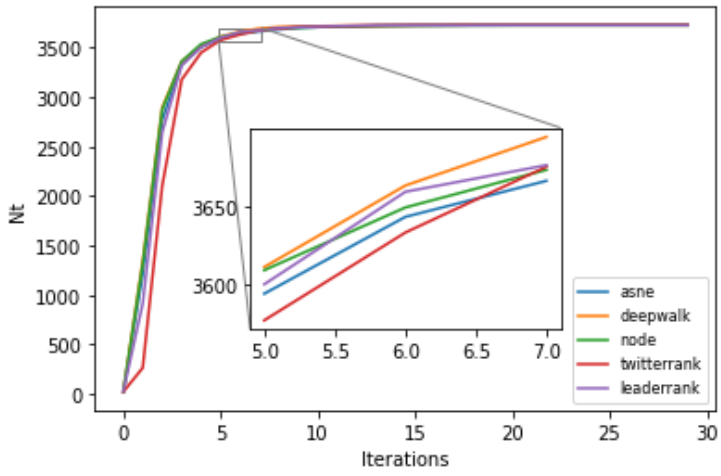
F SIR results



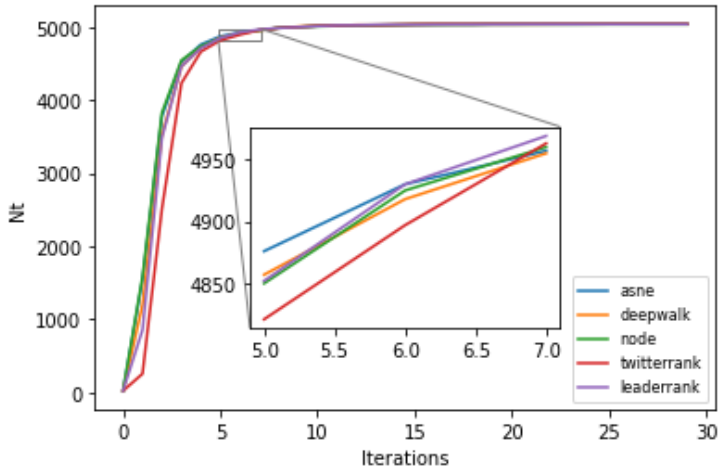
(a) Week 1



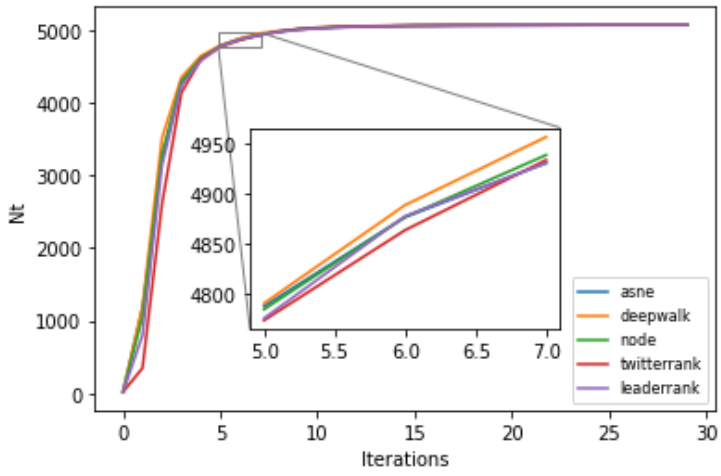
(b) Week 2, part 1



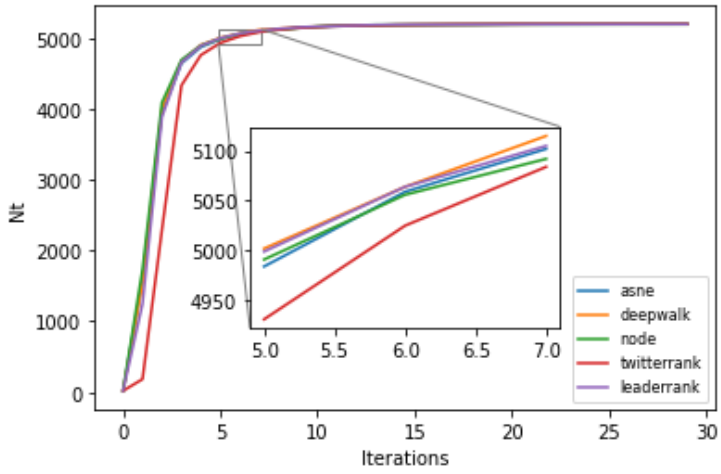
(c) Week 2, part 2



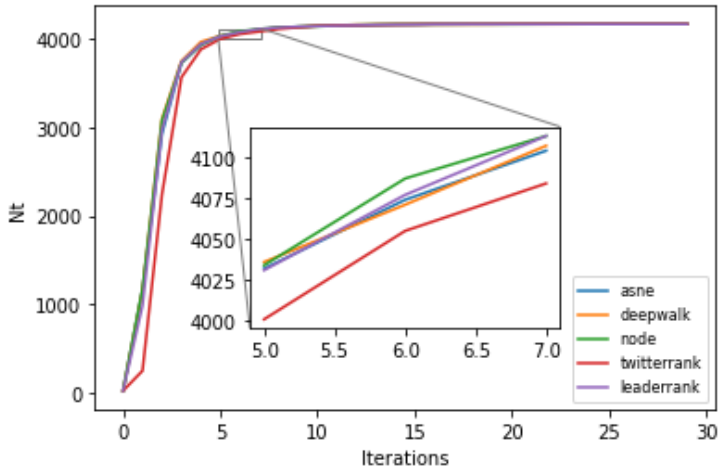
(d) Week 3, part 1



(e) Week 3, part 2



(f) Week 4



(g) Week 5

Figure 21: Results of the SIR model on each week of the data for the ASNE + SNERank, DeepWalk + SNERank, node2vec + SNERank, TwitterRank, and LeaderRank models.